Universidad Zaragoza
1474

# Tesis Doctoral

## Hardware-Aware Algorithms for Efficient Molecular Dynamics and Genome Data Analysis


Autor

Lorién López Villellas

Directores

Jesús Alastruey Benedé
Pablo Enrique Ibáñez Marín



Programa de Doctorado en Ingeniería de Sistemas e Informática
**Escuela de Doctorado**

2026



**Repositorio de la Universidad de Zaragoza - Zaguan**
**https://zaguan.unizar.es**

# Abstract

Modern scientific research increasingly relies on High-Performance Computing (HPC) to execute large-scale simulations and analyze large volumes of data. As the complexity of scientific research grows and processor frequency scaling plateaus, further performance gains rely on more sophisticated architectural paradigms. Unlocking the full potential of complex systems requires software to adopt a hardware-aware design that aligns algorithms and data structures with the specific characteristics of the underlying hardware. This thesis focuses on improving efficiency through this hardware-aware software design and optimization methodology in two critical domains of computational science: molecular dynamics (MD) simulations and genome data analysis.

The first contribution of this thesis is ILVES, a family of parallel algorithms for enforcing constraints in molecular dynamics simulations. While state-of-the-art MD methods, such as SHAKE, LINCS, and P-LINCS, suffer from slow convergence, ILVES achieves high-accuracy results at low computational cost. Moreover, ILVES efficiently constrains additional degrees of freedom beyond the capabilities of existing methods. This enables the use of larger integration time steps, resulting in up to a 1.65× increase in simulation throughput within GROMACS using the same computational resources.

Second, we introduce GenArchBench, a benchmark suite comprising 13 computationally intensive kernels extracted from widely used genomic tools. This contribution focuses on optimizing and evaluating these kernels—traditionally tailored for x86—on Arm and RISC-V architectures that have gained substantial adoption and relevance over the past decade. We present architecture-aware optimization strategies of GenArchBench kernels to effectively exploit the capabilities of these architectures, with particular emphasis on their vector instruction sets: Arm's Scalable Vector Extension (SVE) and the RISC-V Vector (RVV) extensions. Furthermore, we perform an in-depth performance evaluation across two Arm, two RISC-V, and two reference x86 systems to assess the maturity and readiness of the Arm and RISC-V architectures for large-scale genomic data analysis pipelines.

Lastly, this thesis presents Singletrack, a novel backtrace algorithm for gap-affine and dual gap-affine sequence alignment designed to improve the performance of one of the most time- and memory-intensive steps in genome data analysis pipelines. Singletrack significantly reduces the memory footprint of the alignment process, requiring between 2× and 5× less memory than classical dynamic programming approaches. By integrating Singletrack into state-of-the-art sequence alignment libraries, such as KSW2 and WFA2-lib, we demonstrate that this approach also reduces pressure on the memory hierarchy, yielding improvements in overall execution performance.

# Resumen

La ciencia moderna depende cada vez más de la computación de alto rendimiento para la ejecución de simulaciones a gran escala y el análisis de grandes volúmenes de datos. A medida que crece la complejidad de los retos científicos y el aumento de la frecuencia de los procesadores se estanca, las mejoras de rendimiento dependen de arquitecturas *hardware* cada vez más sofisticadas. Explotar todo el potencial de estos complejos sistemas requiere que las aplicaciones se diseñen con consciencia del *hardware* que las va a ejecutar, alineando los algoritmos y las estructuras de datos con las características específicas del *hardware* subyacente. Esta tesis se centra en la mejora de la eficiencia, mediante esta estrategia de optimización y diseño consciente del *hardware*, de dos áreas críticas de la ciencia computacional moderna: las simulaciones de dinámica molecular y el análisis de datos genómicos.

La primera contribución de esta tesis es ILVES, una familia de algoritmos paralelos para imponer restricciones en simulaciones de dinámica molecular. Mientras que los métodos del estado del arte, como SHAKE, LINCS y P-LINCS, presentan una convergencia lenta, ILVES alcanza una alta precisión a un bajo coste computacional. Además, ILVES es capaz de imponer restricciones a grados de libertad adicionales que los métodos existentes no pueden manejar de manera eficiente. Esto permite el uso de pasos de tiempo de integración más altos, lo que se traduce en un aumento de hasta 1,65× en el rendimiento total de las simulaciones ejecutadas en Gromacs con los mismos recursos computacionales.

En la segunda contribución presentamos GenArchBench, una batería de *benchmarks* que comprende 13 núcleos de ejecución extraídos de aplicaciones ampliamente utilizadas en el campo de la genómica. Estos núcleos de ejecución están tradicionalmente preparados para ejecutarse en la arquitectura x86. Este trabajo se centra en la optimización y evaluación de GenArchBench en las arquitecturas Arm y RISC-V, que han ganado una tracción y relevancia significativas durante la última década. Hemos adaptado y optimizado los núcleos de GenArchBench para explotar las características de estas dos arquitecturas, con especial énfasis en sus extensiones vectoriales: SVE en Arm y RVV en RISC-V. Además, evaluamos el rendimiento de GenArchBench en dos sistemas Arm, dos sistemas RISC-V y dos sistemas x86 de referencia. Este análisis nos permite estudiar hasta qué punto las arquitecturas Arm y RISC-V son competitivas en la actualidad frente a x86 para la ejecución de aplicaciones genómicas.

En la tercera y última contribución, presentamos Singletrack, un nuevo algoritmo de *backtrace* para el alineamiento de secuencias con las funciones de coste *gap-affine* y *dual gap-affine*. El alineamiento de secuencias representa uno de los pasos más costosos, tanto en tiempo como en memoria, en las aplicaciones de análisis de datos genómicos. Singletrack reduce significativamente el consumo de memoria del proceso de alineamiento, requiriendo entre 2× y 5× menos memoria que los algoritmos clásicos basados en pro-

gramación dinámica. Además, integramos Singletrack en las librerías KSW2 y WFA2-lib y demostramos que la reducción de memoria obtenida al utilizar Singletrack reduce la presión sobre la jerarquía de memoria, lo que se traduce en mejoras en el rendimiento.

# Acknowledgments

These past four years have been immensely rewarding, and I am deeply grateful to everyone who has supported me along the way. I have been fortunate to be surrounded by people who are not only brilliant and skilled but also remarkably kind. Their selfless willingness to help others has made this journey much brighter. The following lines are intended to express my gratitude to all of them.

To my supervisors, Chus, Pablo, and Santiago (who deserves a place on the front page as an official supervisor), for "tricking" me into this path, for their guidance, for the stimulating technical discussions, for sharing their love of knowledge and science, for trusting me, and for being three of the humblest people I know, even though they have no reason to be.

To Miquel, who welcomed me into his research group back in 2020 when I moved to Barcelona, and who has been a constant source of support for every professional milestone since then.

To the very diverse ILVES team, Carl Christian, Pablo G, Juanjo, and Pablo E, for the countless discussions on physics, mathematics, chemistry, and computer science through gargantuan emails that have quadratically converged into a great piece of work.

To everyone at the Grupo de Arquitectura de Computadores de Zaragoza (gaZ) for making me feel at home from day one and for always offering help without hesitation. May there be many more *tapas* days, *sidrería* dinners, and *migas* in Alpartir.

To the many people I have met and collaborated with at the Barcelona Supercomputing Center (BSC) for creating such a vibrant and inspiring environment. The list of names would be far too long to include here. I owe a special debt of gratitude to my *papis*, Noelia and Víctor, for always having a bed ready for my visits.

To the people at AMD Dublin for their hospitality. I am especially grateful to Linus, Lucian, and Tobi for making me feel part of the team from the very first week; to the other interns, Francesco, Ioannis, and Riccardo, for their friendship, laughter, and support throughout the internship; to Tim for driving all the way to lonely Citywest just to have lunch with us; and of course to the canteen team for the unforgettable *miercotarta*.

To all my closest friends, Adrián R, Adrián S, Bazo, Chipi, Irene (and Maya), Isaac, Laura, Manuel, Miguel, and Matilde, for all the conversations, game nights, vacations, laughs, and countless moments that have made these years unforgettable.

To my parents, Nacho and Pilar, and my sister Izarbe, for all their unconditional love and support, and to my grandparents, who are no longer here.

To Esther, the kindest person I know, for always listening to me, supporting me, and loving me.

# Acronyms

**General**

- HPC: High-Performance Computing
- FP: Floating Point
- DL: Deep Learning
- GUI: Graphical User Interface
- RSS: Resident Set Size

**Hardware**

- ISA: Instruction Set Architecture
- CPU: Central Processing Unit
- DSP: Digital Signal Processor
- FPGA: Field-Programmable Gate Array
- GPU: Graphics Processing Unit
- NPU: Neural Processing Unit
- SIMD: Single-Instruction Multiple-Data
- SVE: Arm Scalable Vector Extension
- RVV: RISC-V Vector Extension
- $\mu$OP: Micro-Operation
- MOP: Macro-Operation
- L1i: Level 1 Instruction cache
- L1d: Level 1 Data cache
- L2: Level 2 cache
- LLC: Last Level Cache
- DRAM: Dynamic Random Access Memory

- SRAM: Static Random Access Memory
- DIMM: Dual In-line Memory Module
- NUMA: Non-Uniform Memory Access
- MT: Mega-Transfers
- CMG: Core Memory Group
- CCD: Core Complex Die
- CCX: Core Complex
- SMT: Simultaneous Multithreading

**Algorithms**

- DP: Dynamic Programming
- AMD: Approximate Minimum Degree ordering
- OPM: Overlapping Partitioning Method
- MSA: Multiple Sequence Alignment
- SK: Suzuki-Kasahara
- ABEA: Adaptive Banded Signal to Event Alignment
- BPM: Bit-Parallel Myers
- CHAIN: Seed Chaining
- FAST-CHAIN: SIMD Seed Chaining
- DBG: De Bruijn Graph
- FMI: FM-Index
- KCNT: K-mer Counting
- NN-BASE: Neural Network-Based Base Calling
- NN-VARIANT: Neural Network-Based Variant Calling
- PILEUP: Pileup Counting
- SW: Smith-Waterman
- BSW: Banded Smith-Waterman
- POA: Partial-Order Alignment

- WFA: Wavefront Alignment
- BiWFA: Bidirectional Wavefront Alignment

**Molecular Dynamics**

- MD: Molecular Dynamics
- PBC: Periodic Boundary Conditions
- PBD: Protein Data Bank
- PME: Particle Mesh Ewald
- NVT: Canonical ensemble - constant number of particles (N), volume (V), and temperature (T)
- NPT: Isothermal-isobaric ensemble - constant number of particles (N), pressure (P), and temperature (T)
- NVE: Microcanonical ensemble - constant number of particles (N), volume (V), and energy (E)

**Genomics**

- ONT: Oxford Nanopore Technologies
- PacBio: Pacific Biosciences
- CIGAR: Compact Idiosyncratic Gapped Alignment Report

# Contents


**1. Introduction** **21**
- 1.1. Molecular Dynamics . . . 22
- 1.2. Genome Data Analysis . . . 22
- 1.3. Contributions . . . 23
  - 1.3.1. ILVES: A Family of Algorithms for Constrained Molecular Dynamics 23
  - 1.3.2. GenArchBench: A Genomics Benchmark Suite for Emerging Architectures . . . 24
  - 1.3.3. Singletrack: Reducing Memory Consumption and Improving Performance of Gap-Affine Sequence Alignment . . . 25
- 1.4. Thesis Structure . . . 26

**2. ILVES: A Family of Algorithms for Constrained Molecular Dynamics** **27**
- 2.1. Introduction . . . 27
- 2.2. Background . . . 29
  - 2.2.1. Mathematical Foundations of Constrained Molecular Dynamics . . 29
  - 2.2.2. Classification of Constraints in Molecular Dynamics . . . 31
  - 2.2.3. The SHAKE Algorithm . . . 32
  - 2.2.4. The LINCS and P-LINCS Algorithms . . . 33
  - 2.2.5. Other Constraint Solvers . . . 36
- 2.3. The Need to Solve the Constraint Equations Accurately . . . 37
- 2.4. The ILVES Algorithms . . . 38
  - 2.4.1. Newton's Method Applied to Constraint Equations . . . 39
  - 2.4.2. ILVES-PC . . . 40
  - 2.4.3. ILVES-M and ILVES-F . . . 42
- 2.5. Experimental Setup . . . 47
- 2.6. Simulations . . . 47
- 2.7. Evaluation . . . 51
  - 2.7.1. Reliability Study . . . 51
  - 2.7.2. Physics Analysis . . . 54
  - 2.7.3. Performance Analysis . . . 56
- 2.8. Conclusion . . . 69
- 2.9. Future Work . . . 70

**3. GenArchBench: A Genomics Benchmark Suite for Emerging Architectures** **73**
- 3.1. Introduction . . . 73
- 3.2. Background . . . 75
  - 3.2.1. Sequencing Technologies . . . 76

3.2.2. Genome Data Analysis Pipelines and Tools . . . 76
3.3. GenArch Benchmark Suite . . . 78
3.4. Target Hardware Systems . . . 83
3.4.1. Arm Systems . . . 83
3.4.2. RISC-V Systems . . . 85
3.4.3. x86 Systems . . . 86
3.5. Porting GenArchBench Kernels to Arm . . . 87
3.5.1. Exploiting Vectorization . . . 88
3.5.2. Optimized Libraries . . . 90
3.5.3. Algorithmic and Code Optimizations . . . 91
3.6. Porting GenArchBench Kernels to RISC-V . . . 92
3.6.1. Software and Libraries Support . . . 93
3.6.2. Development Toolchain Support . . . 93
3.6.3. ISA Support . . . 94
3.7. Evaluation in Arm . . . 96
3.7.1. Single-Threaded Performance . . . 96
3.7.2. Multi-Threaded Performance . . . 98
3.7.3. Instruction Mix Comparison . . . 100
3.7.4. Microarchitecture Bottleneck Analysis . . . 101
3.7.5. Energy Consumption . . . 104
3.7.6. Evaluation of a Real Genomics Tool . . . 105
3.8. Evaluation in RISC-V . . . 107
3.8.1. Single-Threaded Scalar Execution . . . 107
3.8.2. Multi-Threaded Scalar Execution . . . 108
3.9. Related Work . . . 109
3.10. Conclusion . . . 110
3.11. Future Work . . . 112

**4. Singletrack: Reducing Memory Consumption and Improving Performance of Gap-Affine Sequence Alignment 115**
4.1. Introduction . . . 115
4.2. Background . . . 117
4.2.1. Pairwise Sequence Alignment . . . 117
4.2.2. Gap-Affine and Dual Gap-Affine Alignment . . . 118
4.2.3. Classical Backtrace Algorithm . . . 119
4.3. The Singletrack Algorithm . . . 121
4.4. Singletrack in Suzuki-Kasahara Formulation . . . 125
4.5. Singletrack in Wavefront Alignment Algorithm . . . 127
4.6. Experimental Setup . . . 131
4.7. Evaluation . . . 132
4.7.1. Single-Threaded Results . . . 132
4.7.2. Multi-Threaded Results . . . 135
4.8. Conclusion . . . 137
4.9. Future Work . . . 138

**5. Conclusion and Future Work** **139**
5.1. Conclusion . . . . . 139
5.2. Future Work . . . . . 141
5.3. Conclusiones . . . . . 142
5.4. Trabajo Futuro . . . . . 144

**A. Further Discussion on the Need to Solve the Constraint Equations Accurately** **147**

**Bibliography** **153**

# List of Figures


2.1. Diagram of constraints in molecular dynamics . . . . . 31
2.2. Steps to solve a banded linear system in parallel through Gauss-Jordan elimination and the Schur complement method . . . . . 42
2.3. Representations of the simulated molecules . . . . . 49
2.4. Workflow of the calculation of observable quantities . . . . . 53
2.5. Drifts per degree of freedom as a function of the tolerance in satisfying the constraints . . . . . 55
2.6. Temperature evolution over time of the barnase system in the NVE ensemble using different constraint tolerances . . . . . 56
2.7. Single-threaded speedup over SHAKE of MP-LINCS, ILVES-PC, ILVES-M and ILVES-F . . . . . 58
2.8. Average number of iterations per step required by each solver to converge. 59
2.9. Multi-threaded speedup over SHAKE of MP-LINCS, ILVES-M, ILVES-F, and ILVES-PC . . . . . 61
2.10. Multi-threaded speedup over serial execution of MP-LINCS, ILVES-PC, ILVES-M, and ILVES-F . . . . . 62
2.11. Percentage of the total execution time of GROMACS spent in the constraint solver (ex water) . . . . . 64
2.12. Simulation throughput using different constraint solvers . . . . . 65
2.13. Execution time of the block for solving constraints as a function of the average relative error in satisfying them for different constraint solvers . . 66
2.14. Simulation throughput when constraining H-bonds and H-angles . . . . . 67

3.1. Workflow diagram of common genome analysis pipelines . . . . . 75
3.2. Speedup of SIMD kernels over their scalar version on the Arm and x86 systems using the large inputs . . . . . 88
3.3. Execution times of GenArchBench kernels on the Arm and x86 systems . 97
3.4. Speedup over serial execution of GenArchBench kernels on the Arm and x86 systems . . . . . 99
3.5. Instruction mix of GenArchBench kernels on Arm and x86 . . . . . 101
3.6. Microarchitecture bottlenecks of GenArchBench kernels on the Arm and x86 systems . . . . . 103
3.7. Energy-to-solution results of GenArchBench kernels on the Arm and x86 machines . . . . . 105
3.8. Execution times of BWA-MEM2 in the Arm and x86 machines . . . . . 106
3.9. Speedup over serial execution of BWA-MEM2 on the Arm and x86 machines 106

3.10. Single-threaded and scalar execution time of GenArchBench kernels in the RISC-V machines and SKX . . . 107
3.11. Speedup over serial execution of GenArchBench kernels on the Unmatched machine . . . 108

4.1. Classical gap-affine backtrace example . . . 121
4.2. Singletrack gap-affine backtrace example . . . 124
4.3. Computed cells of the DP matrix and wavefronts generated by WFA during the alignment of an example . . . 130
4.4. Multi-threaded speedups of KSW2, KSW2+ST, WFA, BiWFA, and WFA+ST relative to single-threaded BiWFA . . . 136
4.5. Multi-threaded memory usage in GiB of KSW2, KSW2+ST, WFA, BiWFA, and WFA+ST . . . 137

A.1. Total energy of a simple pendulum as a function of time for three different values of timestep . . . 148
A.2. The evolution of the total energy of a pendulum over time for different values of the tolerance . . . 149
A.3. Sign of the relative error of the total energy for the pendulum for two different constraint solvers as a function of the time . . . 151

# List of Tables

2.1. Main features of the computing nodes used to execute the simulations . . 47
2.2. Molecular systems simulated in this work . . . . . 48
2.3. Relevant input parameters employed in the stages of the simulations . . . 50
2.4. Experimentally measured and calculated (MD) thermodynamic quantities 52

3.1. Summary of kernels included in GenArchBench . . . . . 79
3.2. Summary of the evaluated systems' characteristics . . . . . 84
3.3. Instruction mix mapping from Fujitsu Advanced Performance Profiler to our DynamoRIO tool categories . . . . . 100
3.4. Micro-architecture bottlenecks mappings from each system's profiler categories to our aggregated categories . . . . . 102

4.1. Main features of the computing nodes used to execute the experiments . . 132
4.2. Single-threaded runtime and memory usage of KSW2, KSW2+ST, WFA, BiWFA, and WFA+ST . . . . . 133

A.1. Parameters defining a simple pendulum . . . . . 147
A.2. The parameters for the first set of simulations of the pendulum . . . . . 148
A.3. The parameters for the second set of simulations of a pendulum . . . . . 149
A.4. The parameters of a third set of simulations of the pendulum . . . . . 150

# List of Algorithms

2.1. Greedy algorithm for partitioning a graph into $k$ pieces . . . . . 45

4.1. Classical gap-affine backtrace . . . . . 120
4.2. Singletrack gap-affine backtrace . . . . . 122
4.3. Singletrack dual gap-affine backtrace . . . . . 123
4.4. Singletrack gap-affine backtrace in WFA . . . . . 128

# Chapter 1.

# Introduction

The twenty-first century has witnessed a profound transformation in the way science is conducted. Many disciplines that were once driven primarily by observation and experimentation have become increasingly dependent on computational methods to formulate hypotheses, analyze data, and validate results [1]. From climate modeling [2] and materials science [3] to molecular biology [4] and genomics [5], modern research frequently relies on large-scale simulations and the processing of vast amounts of data. These inherently resource-intensive activities demand high performance and scalability [6], a domain known as High-Performance Computing (HPC). As the complexity of scientific problems and experimental throughput continue to grow, a corresponding increase in computational efficiency is required to keep pace.

For several decades, the scientific and engineering communities benefited from the steady improvements in computer processors (CPUs) as predicted by Dennard scaling [7] and Moore's law [8]. Dennard scaling allowed clock frequencies to rise alongside shrinking transistors without increasing power density; meanwhile, the growing transistor budgets afforded by Moore's law enabled architects to reinvest in microarchitectural enhancements (e.g., larger caches, wider superscalar pipelines, and deeper out-of-order execution engines) that improved single-thread performance [9]. Together, these two trends delivered a period of "free performance" in which applications became faster with each processor generation and little or no developer intervention was required [10]–[14]. However, Dennard scaling ended in the mid-2000s, and the single-thread gains once sustained by Moore's Law have significantly slowed in recent years [10]–[14]. Consequently, performance gains are no longer achieved simply by waiting for the next hardware generation.

Modern computing systems achieve performance gains through a combination of parallelism and heterogeneous computing built upon complex microarchitectures [11], [15], [16]. Parallelism is exploited at multiple levels: from distributed-memory parallelism across nodes in a cluster, to thread parallelism within multicore processors, down to instruction-level parallelism in superscalar and out-of-order pipelines, and finally to data-level parallelism via single-instruction multiple-data (SIMD) units. Complementarily, heterogeneous architectures leverage domain-specific accelerators such as graphics processing units (GPUs), digital signal processors (DSPs), and neural processing units (NPUs), enabling specialized tasks to be executed more efficiently.

However, maximizing performance on such complex systems requires more than mere deployment; it demands effective utilization of resources, including leveraging parallelism at every level, efficient use of available hardware accelerators, careful management of deep

memory hierarchies, and minimizing pipeline stalls. In practice, this involves aligning algorithms and data structures with the characteristics of the underlying hardware, that is, adopting a hardware-aware software design approach.

This thesis focuses on improving the efficiency of two cornerstones of modern computational science: molecular dynamics (MD) simulations and genome data analysis. Both are central in life sciences and healthcare and are increasingly coupled in research pipelines [17]–[19]. Despite their distinct algorithmic profiles, both domains require significant computational resources, and therefore demand efficiently leveraging the hardware, including efficient use of the memory hierarchy, vectorization, and parallelization. The central premise of this work is that a hardware-aware strategy that aligns algorithmic design and optimization with architectural constraints unlocks gains in performance, accuracy, and scalability.

## 1.1. Molecular Dynamics

Molecular dynamics (MD) simulations constitute a powerful research tool with broad scientific and technological applications [20]–[22]. They are employed to address a wide range of problems in chemistry and biology, including the development of novel materials [22], the design of new drugs [23]–[27], and the understanding of interaction- or mutation-driven biological processes [28]–[30]. Representative examples include applications in cancer research [31] and in the study of infectious diseases such as SARS-CoV-2 [24], [25].

Fundamentally, MD computes the time evolution of molecular systems by numerically integrating Newton's equations of motion. This enables the estimation of a wide variety of physical quantities whose experimental measurement is often difficult or even infeasible [32], [33].

The impact of molecular simulation continues to grow steadily [34]. A particularly significant boost has been provided by the solution of the protein-folding problem through AlphaFold2 [35], which has dramatically increased the number of proteins with available three-dimensional structures. This unprecedented availability of structural data is expected to further stimulate molecular simulations.

MD is characterized by intensive floating-point arithmetic, sparse linear algebra, and a large number of integration steps, during which small numerical inaccuracies can accumulate and compromise the physics of the simulated system. Consequently, ensuring reliability and numerical stability is as critical as achieving high computational throughput.

## 1.2. Genome Data Analysis

With the advent of sequencing technologies, genome data analysis tools have become increasingly important in biomedical and healthcare research. The wide accessibility to sequencing technologies has enabled the development of personalized medicine [36], [37] for the early detection of diseases, such as hereditary cancers (e.g., Lynch syndrome and breast cancer [38], [39]) and other genetic conditions [40]. Furthermore, genome sequencing has

been shown to be crucial in drug development [41], and COVID-19 outbreak control [42], [43].

As sequencing technologies evolve, the cost of sequencing a human genome drops exponentially, and the current sequencing data production is outpacing Moore's Law [8]. This trend has shifted the primary limitation from data acquisition to data processing, leading to an increasing computational bottleneck in genome analysis pipelines [44]. These pipelines typically consist of heterogeneous chained algorithms that demand high-performance computing resources to handle the massive volumes of biological data [45].

## 1.3. Contributions

### 1.3.1. ILVES: A Family of Algorithms for Constrained Molecular Dynamics

In molecular dynamics, a common strategy to increase simulation throughput is to constrain the fastest degrees of freedom, such as bond lengths, thereby allowing larger integration time steps without compromising accuracy. The de facto state-of-the-art algorithms for this purpose—SHAKE, LINCS, and P-LINCS—are integrated into most molecular dynamics packages and widely adopted across the field. Despite their impact, these methods exhibit limitations: all converge slowly when high numerical accuracy is required, and the LINCS and P-LINCS algorithms cannot handle general angular constraints, limiting further increases in time step.

In this contribution, we introduce ILVES, a family of parallel algorithms that converge so rapidly that it is practical to solve bond-length and associated angular-constraint equations as accurately as the hardware allows.

We integrate ILVES into the widely-used GROMACS molecular dynamics package and demonstrate that it is superior to the state-of-the-art when constraining bond lengths, especially under high accuracy requirements. Due to its better convergence properties, we also show that if the time step is increased up to 3.5 fs from the default 2 fs by enforcing angular constraints, ILVES enables a 1.65× increase in simulated time using the same computational resources and wall-clock time, an outcome unattainable with current methods. This advance can significantly reduce the computational cost of most all-atom molecular dynamics simulations while improving their accuracy and extending access to larger systems and longer time scales.

#### Publications

The primary results of this contribution were published in the following works:

- L. López-Villellas, C. C. Kjelgaard Mikkelsen, et al., "Accurate and efficient constrained molecular dynamics of polymers using Newton's method and special purpose code", *Computer Physics Communications*, vol. 288, p. 108 742, Jul. 2023, ISSN: 0010-4655. DOI: 10.1016/j.cpc.2023.108742
- L. López-Villellas, C. C. K. Mikkelsen, et al., "ILVES: Accurate and efficient bond length and angle constraints in molecular dynamics", *Journal of Chemical Theory*

*and Computation*, vol. 21, no. 18, pp. 8711–8719, Sep. 2025, ISSN: 1549-9626. DOI: 10.1021/acs.jctc.5c01376

Additionally, this contribution builds upon a research base established in the following publications:

- C. C. Kjelgaard Mikkelsen, L. López-Villellas, et al., "How accurate does Newton have to be?", in *Parallel Processing and Applied Mathematics*, Springer International Publishing, 2023, pp. 3–15, ISBN: 9783031304422. DOI: 10.1007/978-3-031-30442-2_1
- C. C. Kjelgaard Mikkelsen, L. López-Villellas, et al., "Newton's method revisited: How accurate do we have to be?", *Concurrency and Computation: Practice and Experience*, vol. 36, no. 10, Jul. 2023, ISSN: 1532-0634. DOI: 10.1002/cpe.7853
- C. C. Kjelgaard Mikkelsen and L. López-Villellas, "The need for accuracy and smoothness in numerical simulations", in *Parallel Processing and Applied Mathematics*, Springer Nature Switzerland, 2025, pp. 3–16, ISBN: 9783031856976. DOI: 10.1007/978-3-031-85697-6_1
- C. C. Kjelgaard Mikkelsen and L. López-Villellas, "How accurate is Richardson's error estimate?", *Concurrency and Computation: Practice and Experience*, vol. 37, no. 27-28, Nov. 2025, ISSN: 1532-0634. DOI: 10.1002/cpe.70305

### 1.3.2. GenArchBench: A Genomics Benchmark Suite for Emerging Architectures

In recent years, the CPU ecosystem has undergone significant diversification. The long-standing dominance of the x86 architecture is evolving into a heterogeneous landscape where Arm and RISC-V processors are rapidly gaining relevance. Arm CPUs now power prominent systems in the TOP500 list, including the Fugaku supercomputer, which held the first position from 2020 to 2022. In parallel, the open-source RISC-V architecture is attracting massive global investment as a strategic alternative for next-generation HPC and specialized computing. However, this diversification creates distinct challenges for portability and optimization. Code that performs well on one architecture may suffer performance degradation on another due to variations in instruction sets and microarchitecture [52]–[55]. In this context, algorithms must be re-evaluated and often optimized to exploit the specific architectural features of different platforms.

In this contribution, we introduce the GenArch benchmark suite, which compiles 13 computationally demanding kernels from widely used genomic tools. We port these kernels to Arm, leveraging the recently introduced Scalable Vector Extension (SVE), Arm-specific HPC libraries, and algorithmic and code optimizations tailored to the A64FX and Graviton3 processors. We also port GenArchBench to RISC-V, exploiting the RVV vector extension. We discuss the challenges encountered during the RISC-V porting process, including the maturity of the development toolchain, availability of libraries, and the support level of the Instruction Set Architecture (ISA).

We perform a comprehensive performance characterization of GenArchBench across two Arm CPUs (A64FX and Graviton3) and two RISC-V CPUs (HiFive Unmatched and Allwinner D1), comparing them against reference Intel (Xeon Skylake Platinum 8160) and AMD (EPYC 7742 Rome) x86 processors. Our results show that the A64FX requires aggressive optimizations to overcome its limited out-of-order resources and high memory latency. In contrast, the Graviton3 delivers more competitive overall performance for genomics workloads. Meanwhile, the RISC-V ecosystem still faces substantial software and hardware challenges before it can efficiently handle production-scale genomics tasks.

#### Publications

This contribution has been published in the following works:

- L. López-Villellas, R. Langarita-Benítez, et al., "GenArchBench: A genomics benchmark suite for Arm HPC processors", *Future Generation Computer Systems*, vol. 157, pp. 313–329, Aug. 2024, ISSN: 0167-739X. DOI: 10.1016/j.future.2024.03.050
- L. López-Villellas, E. Pineda-Sánchez, et al., "RISC-V for genome data analysis: Opportunities and challenges", in *2023 38th Conference on Design of Circuits and Integrated Systems (DCIS)*, IEEE, Nov. 2023, pp. 1–6. DOI: 10.1109/dcis58620.2023.10335997

### 1.3.3. Singletrack: Reducing Memory Consumption and Improving Performance of Gap-Affine Sequence Alignment

Sequence alignment is one of the most time-consuming tasks in genome data analysis. Classical dynamic programming (DP) algorithms are particularly memory-intensive, especially when computing gap-affine and dual gap-affine alignments. Existing strategies to reduce memory consumption often sacrifice either speed or alignment accuracy.

In this contribution, we present Singletrack, an efficient algorithm for backtrace gap-affine and dual gap-affine alignments that requires only storing a single DP matrix while preserving exact alignment results. Compared to classical DP algorithms, Singletrack removes the need to store additional matrices (i.e., 2 for gap-affine and 4 for dual gap-affine), significantly reducing memory consumption and, in turn, reducing pressure on the memory hierarchy and improving overall performance. Most importantly, Singletrack is a general backtrace method compatible with state-of-the-art DP-based algorithms and heuristics, such as the Suzuki-Kasahara (SK) and the Wavefront Alignment (WFA) algorithms.

We experimentally demonstrate that Singletrack reduces memory consumption for both SK and WFA algorithms, lowering SK usage by 2× and 4× and WFA usage by 3× and 5× for gap-affine and dual gap-affine alignments, respectively. Moreover, replacing KSW2's memory-reduction technique with Singletrack accelerates its SK implementation by up to 1.4× at the cost of a lower memory consumption reduction, while Singletrack increases the performance of the WFA implementation in WFA2-lib by 1.2–2.1×.

### Publications

This contribution has been published in the following work:

- L. López-Villellas, C. Iñiguez, et al., "Singletrack: An algorithm for improving memory consumption and performance of gap-affine sequence alignment", *Bioinformatics*, vol. 42, no. 5, Apr. 2026, ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btag183

## 1.4. Thesis Structure

The remainder of this thesis is organized as follows:

- **Chapter 2** presents the first contribution, the ILVES family of algorithms, and evaluates their reliability, accuracy, performance, and scalability against state-of-the-art constraint methods.
- **Chapter 3** details the second contribution, introducing the GenArch benchmark suite and characterizing the performance and behavior of its kernels across Arm, RISC-V, and x86 processors.
- **Chapter 4** describes the third contribution, the Singletrack backtrace algorithm, and analyzes its impact on memory consumption and alignment time compared to widely used sequence alignment tools.
- **Chapter 5** concludes the thesis by summarizing the primary contributions and discussing potential directions for future research.

# Chapter 2.

# ILVES: A Family of Algorithms for Constrained Molecular Dynamics

*This chapter presents the ILVES family of parallel algorithms, designed to address the accuracy and efficiency limitations of state-of-the-art constraint solvers. We provide a detailed analysis of why high numerical accuracy is desirable when solving constraints, demonstrating that inaccurate constraint solving acts as a source of spurious external forces that cause unphysical energy drifts. By leveraging Newton's method and custom sparse linear solvers, ILVES achieves rapid convergence, making it practical to solve equations with the highest possible precision. Furthermore, ILVES supports constraining angles in parallel, enabling a 1.65× increase in simulation throughput by using larger integration time steps.*

## 2.1. Introduction

Molecular dynamics (MD) simulations [32], [33] have had a profound impact across many fields of science and technology [20]–[22], [31]. They are particularly important in chemistry and medicine, with applications ranging from the design of drugs and catalysts [23]–[27] to understanding interaction- or mutation-driven biological processes [28]–[30]. A major advantage of MD is that it provides atomic-level information (including positions, velocities, and forces) enabling the study of phenomena that are often inaccessible or impractical to observe experimentally [57], [58]. Recent advances, particularly in artificial intelligence, are further expanding the reach of MD. For instance, protein structure predictions via AlphaFold [35] are already facilitating large-scale analyses of protein behavior and interactions, with implications across a wide range of applications [59]–[65].

It is generally accepted that, to ensure reliability, the discrete integration of the equations of motion in molecular dynamics must include at least five steps per vibrational period of every degree of freedom, such as bond lengths, bond angles, or dihedral angles. This requirement imposes an upper limit on the integration time step, i.e., the interval between consecutive simulation steps. Since each step involves a fixed number of arithmetic operations, the maximum time step directly limits the total physical time that can be simulated for a given computational effort. Consequently, MD simulations often constrain the fastest internal degrees of freedom to fixed values. When done carefully so as not to distort the underlying physical model or its resulting dynamics and thermodynamics, removing the shortest vibrational periods allows larger time steps, enabling longer simulated times without increasing computational cost.

The imposition of constraints starts with the fastest degrees of freedom and proceeds gradually toward slower ones [66], allowing the time step to be increased at each stage. In production MD simulations, it is common practice to constrain bond lengths involving hydrogen atoms (H-bonds) or all bond lengths (all-bonds), which enables time steps of approximately 2 fs.

The most widely used algorithms for enforcing bond constraints are SHAKE [67], LINCS [68], and P-LINCS [69], the parallel variant of LINCS. Despite their historical importance and widespread adoption, these algorithms exhibit limitations that are addressed in this chapter.

Both SHAKE and (P-)LINCS exhibit slow convergence, which makes them inefficient when high constraint accuracy is required. As shown in [46], [70], [71] and further demonstrated in this chapter, inaccurate solutions of the constraint equations lead to a drift in the total mechanical energy, ultimately compromising the reliability of molecular dynamics simulations.

These limitations are further amplified when additional degrees of freedom are constrained. To date, there have been no successful parallel implementations of SHAKE beyond H-bond constraints, making its use for all-bonds or bond-angle constraints computationally expensive. Increasing the number of constrained degrees of freedom also degrades SHAKE's convergence rate, further degrading its performance [72]. Although (P-)LINCS is generally faster than SHAKE and supports parallel execution, it is fundamentally restricted to bond-length constraints and cannot be extended to additional degrees of freedom such as bond angles [73].

In this chapter, we first discuss how inaccuracies in solving constraint equations can undermine the reliability of molecular dynamics simulations. Even small errors can accumulate over time, leading to systematic drifts in energy and distortions in the simulated dynamics. This analysis is presented in [46], [47], [50], [51].

We then introduce the ILVES family of algorithms, designed to overcome both the accuracy limitations of SHAKE and (P-)LINCS and their inability to efficiently handle additional constrained degrees of freedom. This family comprises three algorithms: ILVES-PC, ILVES-M, and ILVES-F.

ILVES-PC was introduced in [46] as a proof of concept for applying Newton's method to molecular constraint solving. It solves the same system of differential-algebraic equations as SHAKE but replaces the nonlinear Gauss–Seidel iteration with Newton's method and a custom direct solver, improving convergence from linear to quadratic. ILVES-PC supports all-bonds constraints for peptides and proteins and exploits shared-memory parallelism. When integrated into Gromacs, it enables highly accurate constraint enforcement at a lower computational cost than SHAKE and (P-)LINCS, while retaining comparable performance under low accuracy requirements. A distinctive aspect of ILVES-PC is its use of special-purpose code: the linear solver consists of dedicated, branchless routines tailored to the fixed structure of amino acid residues, reducing control overhead by exploiting the regular topology of peptide chains.

ILVES-M and ILVES-F were subsequently introduced in [47] to generalize the idea behind ILVES-PC to arbitrary molecular systems. Unlike ILVES-PC, both algorithms are fully general and comparable in scope to SHAKE and (P-)LINCS.

ILVES-M adopts the same Newton-based formulation as ILVES-PC but replaces the specialized solver with a custom parallel general-purpose linear solver. In addition, it supports distributed-memory parallelism and exploits SIMD vectorization. Although ILVES-M is fully general, in contrast to ILVES-PC, it achieves nearly the same performance, with only about a 10% performance reduction.

ILVES-F modifies the equations solved by ILVES-M (and ILVES-PC) without altering the physical results of the simulations, thereby replacing Newton's method with a quasi-Newton approach. Although quasi-Newton methods exhibit linear rather than quadratic convergence, in [48], [49] we showed that, under the finite precision of floating-point arithmetic, fast linear convergence is sufficient to reach the desired solution accuracy using a number of iterations comparable to that required by quadratic convergence. Leveraging this observation, ILVES-F enables the use of a more efficient custom linear solver and delivers nearly a twofold performance improvement over ILVES-M.

Beyond improving performance for bond-length constraints, ILVES-M and ILVES-F leverage the existing GROMACS infrastructure to impose bond-angle constraints in parallel with low computational overhead, which is not feasible with SHAKE or (P-)LINCS. Our results show that constraining hydrogen-bond angles and increasing the time step to 3.5 fs enable ILVES-M and ILVES-F to achieve a 1.65× increase in simulated time with the same computational resources and wall-clock time as simulations with the standard 2 fs time step.

## 2.2. Background

### 2.2.1. Mathematical Foundations of Constrained Molecular Dynamics

In this section, we present the mathematical fundamentals of constrained MD.

We begin by defining our notation. Vectors are written using bold lowercase letters, e.g., $\boldsymbol{x}$, while matrices are written using bold uppercase letters, e.g., $\boldsymbol{A}$. If $\boldsymbol{A} = [a_{ij}]$ is any matrix, then the transpose matrix $\boldsymbol{A}^T$ is the matrix $\boldsymbol{B} = [b_{ij}]$ given by $b_{ij} = a_{ji}$. By default, all vectors are column vectors, and when a row vector is needed, a column vector is explicitly transposed. The Euclidean inner product $s = \boldsymbol{x} \cdot \boldsymbol{y}$ between two vectors $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$ and $\boldsymbol{y} = (y_1, y_2, \ldots, y_n)^T \in \mathbb{R}^n$ is given by $s = \boldsymbol{x}^T\boldsymbol{y} = \sum_{j=1}^{n} x_i y_i$. The corresponding Euclidean norm $\|\boldsymbol{x}\|_2$ is the non-negative real number given by $\|\boldsymbol{x}\|_2 = \sqrt{\boldsymbol{x}^T\boldsymbol{x}} = \sqrt{\sum_{j=1}^{n} x_j^2}$. If the function $\boldsymbol{f} = (f_1, f_2, \ldots, f_l)^T : \mathbb{R}^n \to \mathbb{R}^l$ is differentiable, then the Jacobian $\boldsymbol{F}(\boldsymbol{x})$ of $\boldsymbol{f}$ at the point $\boldsymbol{x} \in \mathbb{R}^n$ is the matrix $\boldsymbol{A} = [a_{ij}] \in \mathbb{R}^{l \times n}$ of first order partial derivatives, i.e.,

$$a_{ij} := \frac{\partial f_i}{\partial x_j}(\boldsymbol{x})\,. \tag{2.1}$$

In a system of $s$ atoms, let $m_i > 0$ denote the mass of the $i$th atom and let $\boldsymbol{m}_i \in \mathbb{R}^3$ and the diagonal mass matrix $\boldsymbol{M} \in \mathbb{R}^{3s \times 3s}$ be given by

$$\boldsymbol{m}_i := (m_i, m_i, m_i)^T, \quad \boldsymbol{M} := \operatorname{diag}(\boldsymbol{m}_1^T, \boldsymbol{m}_2^T, \ldots, \boldsymbol{m}_n^T)\,. \tag{2.2}$$

In addition, let $\boldsymbol{q}_i, \boldsymbol{v}_i \in \mathbb{R}^3$ denote the position and velocity of the $i$th atom, and let $\boldsymbol{f}_i \in \mathbb{R}^3$ represent the force acting on it (due to any atom or field). The vectors $\boldsymbol{q}, \boldsymbol{v}, \boldsymbol{f} \in \mathbb{R}^{3s}$, which gather all positions, velocities, and forces are defined as follows

$$\boldsymbol{q} := (\boldsymbol{q}_1^T, \boldsymbol{q}_2^T, \ldots, \boldsymbol{q}_s^T)^T\,; \quad \boldsymbol{v} := (\boldsymbol{v}_1^T, \boldsymbol{v}_2^T, \ldots, \boldsymbol{v}_s^T)^T\,; \quad \boldsymbol{f} := (\boldsymbol{f}_1^T, \boldsymbol{f}_2^T, \ldots, \boldsymbol{f}_s^T)^T\,. \tag{2.3}$$

Throughout this chapter, $s$ is the number of atoms (hence, the total number of Cartesian coordinates for positions is $3s$) and $n$ is the number of imposed constraints.

The phenomena of interest for MD span a wide range of time scales. The subset that can be simulated is limited by the time step ($\Delta t$), which is defined as the gap between consecutive simulated times; for force fields-based MD, $\Delta t$ is typically in the range of femtoseconds (fs). Therefore, setting it to values that are as high as possible (without distorting the dynamics of the simulated system) is a major task. To accomplish this, a widely used method involves imposing constraints on the molecular internal degrees of freedom. Numerous algorithms to perform the corresponding calculations have been proposed, with SHAKE [67], LINCS [68], and P-LINCS [69] being the most frequent ones to be applied to molecules other than water.

Molecular dynamics is based on Newton's second law, which can be written as follows for a system consisting of $s$ atoms:

$$\dot{\boldsymbol{q}}(t) = \boldsymbol{v}(t)\,, \tag{2.4a}$$
$$\boldsymbol{M}\dot{\boldsymbol{v}}(t) = \boldsymbol{f}(t)\,, \tag{2.4b}$$

where $\boldsymbol{q}$, $\boldsymbol{v}$, $\boldsymbol{f} \in \mathbb{R}^{3s}$ are the vectors of the atomic positions, atomic velocities and total forces acting on the atoms; $\boldsymbol{M}$ is the matrix of (constant) atomic masses and the dot on top indicates time ($t$) derivative.

If we impose a set of $n$ bond constraints on the system (2.4), then the equations of motion change to a system of differential-algebraic equations:

$$\dot{\boldsymbol{q}}(t) = \boldsymbol{v}(t)\,, \tag{2.5a}$$
$$\boldsymbol{M}\dot{\boldsymbol{v}}(t) = \boldsymbol{f}_{\text{ext}}(t) - \boldsymbol{G}(\boldsymbol{q}(t))^T\boldsymbol{\lambda}(t)\,, \tag{2.5b}$$
$$\boldsymbol{g}(\boldsymbol{q}(t)) = \boldsymbol{0}\,. \tag{2.5c}$$

Here $\boldsymbol{f}_{\text{ext}}$ is the total external force acting on the atoms (which can be calculated using a force field); $\boldsymbol{g} : \mathbb{R}^{3s} \to \mathbb{R}^n$ is the constraint function

$$\boldsymbol{q} \to (g_1(\boldsymbol{q}), g_2(\boldsymbol{q}), \ldots, g_n(\boldsymbol{q})) \tag{2.6}$$

and $\boldsymbol{G}(\boldsymbol{q}) \in \mathbb{R}^{n\times 3s}$ is the Jacobian of $\boldsymbol{g}$ with respect to the vector $\boldsymbol{q}$ of atomic coordinates. If the $i$th constraint is a bond length constraint between atoms $a_i$ and $b_i$, then we may choose

$$g_i(\boldsymbol{q}) = \frac{1}{2}\left(d_i^2 - \|\boldsymbol{q}_{a_i} - \boldsymbol{q}_{b_i}\|^2\right). \tag{2.7}$$

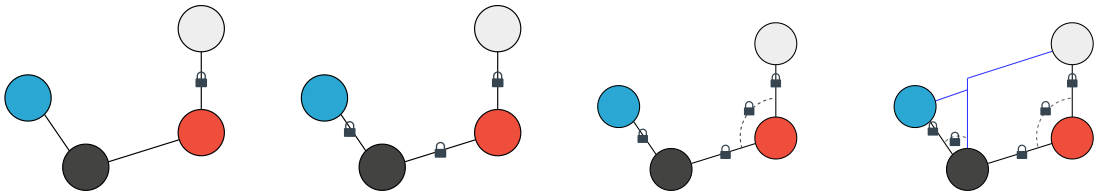
where $d_i$ (commonly a constant) is the desired distance between the atoms $a_i$ and $b_i$. This particular choice ensures that the constraint function is infinitely differentiable at every

point $\boldsymbol{q}$. The same construction can be extended to constraints acting on other degrees of freedom, such as bond angles (i.e., the angle formed by atoms $a_i$ and $b_i$, and a third atom) and dihedral angles (i.e., the angle defined by four atoms).

The vast majority of computer simulations rely on discrete time steps. Among the standard integrators used to update positions (whether strictly following Newton's second law or a modification of it) are the Verlet [74] algorithms (e.g., Leapfrog and Velocity Verlet), as well as the Stochastic Dynamics and Langevin Dynamics algorithms [75].

### 2.2.2. Classification of Constraints in Molecular Dynamics

In this section, we introduce the molecular degrees of freedom that can be constrained in order to increase the integration time step in molecular dynamics simulations. Figure 2.1 presents a diagram of the different types of constraints considered throughout this chapter.



(a) H-bonds (b) all-bonds (c) H-angles (d) dihedral-angles

Figure 2.1.: Diagram of constraints in molecular dynamics.

At the time of writing this thesis, the most commonly constrained degrees of freedom in molecular dynamics simulations are covalent bond lengths involving hydrogen atoms (H-bonds, see Figure 2.1a) or all covalent bond lengths in the system (all-bonds, see Figure 2.1b). Constraining H-bonds allows the integration time step to be increased to 2 fs [66], and to approximately 2.5 fs when all bond lengths are constrained, compared to the typical time step of about 1 fs used when no constraints are applied. As a consequence, the number of integration steps required to reach a given physical time is effectively halved, leading to a substantial reduction in computational cost, even after accounting for the additional overhead associated with constraint solving. For this reason, the vast majority of production molecular dynamics simulations apply bond-length constraints.

The choice between H-bond and all-bond constraints is generally determined, or at least recommended, by the force field. CHARMM36 [76] and CHARMM36m [77] support only H-bond constraints, whereas other force fields such as AMBER [78] and OPLS/AA [79] allow both H-bond and all-bond constraints. In practice, H-bond constraints are the standard in production simulations. By contrast, all-bond constraints are typically used in specific simulations or in combination with additional approximations, such as mass repartitioning or virtual sites [80], [81], to enable time steps larger than 2 fs.

Mass repartitioning increases the masses of hydrogen atoms above 1 atomic mass unit while compensating by reducing the masses of the corresponding heavy atoms, thereby shifting high-frequency vibrational modes to lower frequencies. Virtual sites, in turn, determine the positions of hydrogen atoms as fixed functions of the positions of nearby

heavy atoms, effectively removing hydrogen degrees of freedom from the integration while still allowing them to participate in nonbonded interactions. Although these approaches can produce suitable results for certain observables and thermodynamic quantities [82], the additional assumptions they introduce into the physical model can significantly affect some kinetic properties, such as diffusion in lipid membranes or typical timescales of protein–ligand binding [83]–[85].

An increase of the time step up to 4 fs [66], [72], [86], [87] can be achieved by constraining all-bonds and bond angles involving hydrogen atoms (H-angles), as illustrated in Figure 2.1c. Unlike mass repartitioning or virtual sites, H-angle constraints do not rely on approximations and are chemically and physically justifiable, as quantum harmonic oscillators more closely resemble constraints than classical harmonic oscillators.

H-angle constraints correspond to fixing the angle formed by three atoms connected through two covalent bonds, typically involving a central heavy atom bonded to two hydrogens or to one hydrogen and one heavy atom. While the most accurate way to enforce angular constraints is to constrain internal coordinates directly [86], this approach is rarely used in practice due to its computational complexity and overhead. Instead, H-angle constraints are commonly implemented through a procedure known as *triangularization*, in which an angular constraint between atoms A, B, and C, connected by bonds A–B and B–C, is replaced by an additional, virtual bond between atoms A and C. This is the approach adopted by GROMACS when `constraints=h-angles` is used. Despite the substantial increase in the time step, production molecular dynamics simulations employing H-angle constraints remain relatively rare and are mostly limited to specialized studies [72], [81], [86], [88]–[95]. As will be shown later in this chapter, this limited adoption is likely due to the difficulty that state-of-the-art constraint solvers face in handling H-angle constraints efficiently while preserving overall simulation performance.

An even more aggressive strategy for increasing the integration time step is to constrain angular degrees of freedom beyond H-angles or to impose additional constraints on dihedral angles, which describe the relative torsional orientation of four consecutively bonded atoms (see Figure 2.1d).

Dihedral degrees of freedom are associated with some of the slowest internal motions in molecular systems and can, in principle, enable time steps well beyond 4 fs. However, dihedral constraints introduce strong nonlinear couplings between atoms and are considerably more complex to enforce than bond or angle constraints. As a result, their practical use has been limited, and only a small number of algorithmic approaches have been proposed to address this problem [96].

### 2.2.3. The SHAKE Algorithm

The SHAKE algorithm solves the system of differential algebraic equations 2.5 using a pair of staggered uniform grids with fixed time step $h$ (being $h := \Delta t$). The central

equations can be stated as follows

$$\boldsymbol{v}_{k+1/2} = \boldsymbol{v}_{k-1/2} + h\boldsymbol{M}^{-1}\big(\boldsymbol{f}_{ext}(t_k) - \boldsymbol{G}(\boldsymbol{q}_k)^T\boldsymbol{\lambda}_k\big)\,, \tag{2.8a}$$

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + h\boldsymbol{v}_{k+1/2}\,, \tag{2.8b}$$

$$\boldsymbol{g}(\boldsymbol{q}_{k+1}) = \boldsymbol{0}\,, \tag{2.8c}$$

where $k \in \mathbb{N}$ is the time index, i.e.,

$$t_k = kh, \quad t_{k+\frac{1}{2}} = \left(t + \frac{1}{2}\right) h$$

and

$$\boldsymbol{q}_k \approx \boldsymbol{q}(t_k), \quad \boldsymbol{v}_{k+\frac{1}{2}} \approx \boldsymbol{v}\left(t_{k+\frac{1}{2}}\right). \tag{2.9}$$

Equation (2.8c) is usually a nonlinear equation for the unknown vector of Lagrange multipliers $\boldsymbol{\lambda}_k$, namely $\boldsymbol{g}(\boldsymbol{\phi}_k(\boldsymbol{\lambda})) = \boldsymbol{0}$ where $\boldsymbol{\phi}_k$ is a function obtained by combining (2.8a), (2.8b):

$$\boldsymbol{\phi}_k(\boldsymbol{\lambda}) := \boldsymbol{q}_k + h\left(\boldsymbol{v}_{k-\frac{1}{2}} + h\boldsymbol{M}^{-1}\big(\boldsymbol{f}_{ext}(t_k) - \boldsymbol{G}(\boldsymbol{q}_k)^T\boldsymbol{\lambda}\big)\right). \tag{2.10}$$

Despite its historical importance in constrained molecular dynamics, SHAKE exhibits several well-known limitations. The original SHAKE algorithm solves the nonlinear system using the Gauss–Seidel method, in which constraints are enforced through successive local corrections rather than through a global solve as in direct methods such as Gaussian elimination. Each iteration adjusts one constraint at a time while keeping the others held fixed, so the effect of a correction propagates only gradually through the system as multiple sweeps are performed. This method converges slowly and only locally, under relatively restrictive conditions [97], [98]. Therefore, achieving high accuracy is computationally expensive, and as the number of constrained degrees of freedom increases, the convergence rate deteriorates significantly, making SHAKE impractical for constraints beyond simple bond lengths.

SHAKE is also generally regarded as poorly suited for parallelization [99]. Although parallel variants have been proposed [99], [100], they exhibit limitations and have not seen widespread adoption (as discussed later in this section). Notably, the implementation of SHAKE in Gromacs, which at the time of writing is the most widely used molecular dynamics package, remains sequential. Other popular packages, such as NAMD [101], [102], apply SHAKE in parallel only in embarrassingly parallel settings, that is, when the constraint problem decomposes into multiple independent linear systems. Typical examples include simulations involving multiple independent molecules or cases where constraints are applied exclusively to H-bonds.

### 2.2.4. The LINCS and P-LINCS Algorithms

The (P-)LINCS algorithms, where P-LINCS denotes the parallel version of LINCS, share the same conceptual framework. In what follows, all statements apply to both LINCS and P-LINCS.

Like SHAKE, the (P-)LINCS algorithms are based on Verlet integration and employ a pair of staggered grids with a uniform time step $h$. (P-)LINCS enforces constraints on bond-length functions $\boldsymbol{g} : \mathbb{R}^{3N} \to \mathbb{R}^n$ where the individual components $g_i$ of $\boldsymbol{g}$ must have the form

$$g_i(\boldsymbol{q}) = \|\boldsymbol{q}_{a_i} - \boldsymbol{q}_{b_i}\|_2 - d_i, \quad i = 1, 2, \ldots, n. \tag{2.11}$$

Here $d_i$ is the length of the bond between atoms $a_i$ and $b_i$ whose coordinates are $\boldsymbol{q}_{a_i}$ and $\boldsymbol{q}_{b_i}$ respectively. The Jacobian $\boldsymbol{G}$ of this specific choice of constraints satisfy the elementary identity

$$\forall\, \boldsymbol{q}, \boldsymbol{s} \in \mathbb{R}^{3s} \;:\; (G(\boldsymbol{q})\boldsymbol{s})_i = \left( \frac{\boldsymbol{q}_{a_i} - \boldsymbol{q}_{b_i}}{\|\boldsymbol{q}_{a_i} - \boldsymbol{q}_{b_i}\|} \right)^T (\boldsymbol{s}_{a_i} - \boldsymbol{s}_{b_i}), \quad i = 1, 2, \ldots, n, \tag{2.12}$$

which is used repeatedly in the derivation of the algorithm [68].

We now summarize the LINCS procedure. Given the current positions $\boldsymbol{q}_k$ and velocities $\boldsymbol{v}_{k-\frac{1}{2}}$ the next state vectors $\boldsymbol{q}_{k+1}$ and $\boldsymbol{v}_{k+\frac{1}{2}}$ are computed as follows. An initial approximation $\boldsymbol{q}^*_{k+1}$ is computed using

$$\boldsymbol{q}^*_{k+1} = \boldsymbol{q}_k + h\boldsymbol{v}_{k-\frac{1}{2}} + h^2\boldsymbol{M}^{-1}\boldsymbol{f}_k. \tag{2.13}$$

where $\boldsymbol{f}_k = \boldsymbol{f}_{\text{ext}}(\boldsymbol{q}_k)$ is the total external force acting on the atoms at the current positions $\boldsymbol{q}_k$. This definition corresponds to a single step of the Verlet method for the differential equation

$$\dot{\boldsymbol{q}}(t) = \boldsymbol{v}(t), \tag{2.14}$$

$$\boldsymbol{M}\dot{\boldsymbol{v}}(t) = \boldsymbol{f}_{\text{ext}}(t). \tag{2.15}$$

In general, one cannot expect that $\boldsymbol{q}^*_{k+1}$ solves the constraint equation $\boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{0}$, which is why LINCS performs at least one additional step. Specifically, the algorithm computes

$$\boldsymbol{q}^0_{k+1} = \boldsymbol{q}^*_{k+1} - \boldsymbol{T}_k \left( \boldsymbol{G}_k \boldsymbol{q}^*_{k+1} - \boldsymbol{d} \right) \tag{2.16}$$

where the central matrix $\boldsymbol{T}_k$ and the vector $\boldsymbol{d}$ are given by

$$\boldsymbol{T}_k = \boldsymbol{M}^{-1}\boldsymbol{G}_k^T(\boldsymbol{G}_k\boldsymbol{M}^{-1}\boldsymbol{G}_k^T)^{-1}, \tag{2.17}$$

$$\boldsymbol{d} = (d_1, d_2, \ldots, d_n)^T, \tag{2.18}$$

and then (optionally) continues with the iteration

$$\boldsymbol{q}^{l+1}_{k+1} = \boldsymbol{q}^l_{k+1} - \boldsymbol{T}_k \left( \boldsymbol{G}_k \boldsymbol{q}^l_{k+1} - \boldsymbol{p}(\boldsymbol{q}^l_{k+1}) \right), \quad l = 0, 1, 2, \ldots, \tag{2.19}$$

until the norm of the residual $\boldsymbol{g}(\boldsymbol{q}^l_{k+1})$ is sufficiently small. Here the central function $\boldsymbol{p} : \mathbb{R}^{3s} \to \mathbb{R}^n$ is given by

$$p_i(\boldsymbol{q}) = \sqrt{2d_i^2 - \|\boldsymbol{q}_{a_i} - \boldsymbol{q}_{b_i}\|_2^2} \tag{2.20}$$

and the next state vectors are computed as follows

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_{k+1}^{l_{\max}} \tag{2.21}$$

$$\boldsymbol{v}_{k+\frac{1}{2}} = \frac{\boldsymbol{q}_{k+1} - \boldsymbol{q}_k}{h}. \tag{2.22}$$

where $l_{\max}$ is the final value of the index $l$.

The LINCS algorithm hinges on our ability to apply the operator $\boldsymbol{T}_n$. Specifically, it is necessary to solve linear systems of the form

$$\left(\boldsymbol{G}_k \boldsymbol{M}^{-1} \boldsymbol{G}_k^T\right) \boldsymbol{x} = \boldsymbol{y} \tag{2.23}$$

where the right-hand side vector $\boldsymbol{y}$ and the unknown vector $\boldsymbol{x}$ are both elements of $\mathbb{R}^s$. Since the inception of the algorithm, the solution $\boldsymbol{x}$ has been approximated using a truncated Neumann series. Specifically, the coefficient matrix $\boldsymbol{G}_n \boldsymbol{M}^{-1} \boldsymbol{G}_n^T$ is subjected to a diagonal scaling, i.e.,

$$\boldsymbol{G}_k \boldsymbol{M}^{-1} \boldsymbol{G}_k^T = \boldsymbol{S}^{-1} \left(\boldsymbol{S} \boldsymbol{G}_k \boldsymbol{M}^{-1} \boldsymbol{G}_k^T \boldsymbol{S}\right) \boldsymbol{S}^{-1} \tag{2.24}$$

where $\boldsymbol{S} \in \mathbb{R}^{s \times s}$ is a diagonal matrix such that the diagonal entries of the matrix $\boldsymbol{S} \boldsymbol{G}_n \boldsymbol{M} \boldsymbol{G}_k^T \boldsymbol{S}$ are all equal to one. This matrix is then viewed as a perturbation of the identity matrix $I_s$, i.e.,

$$(\boldsymbol{S} \boldsymbol{G}_k \boldsymbol{M}^{-1} \boldsymbol{G}_k^T \boldsymbol{S}) = \boldsymbol{I}_s - \boldsymbol{A}_k \tag{2.25}$$

and the action of the inverse of $\boldsymbol{S} \boldsymbol{G}_k \boldsymbol{M}^{-1} \boldsymbol{G}_k^T \boldsymbol{S}$ is approximated using a truncated Neumann series, i.e.,

$$\begin{aligned} \boldsymbol{x} = (\boldsymbol{G}_k \boldsymbol{M}^{-1} \boldsymbol{G}_k^T)^{-1} \boldsymbol{y} &= \boldsymbol{S} (\boldsymbol{S} \boldsymbol{G}_k \boldsymbol{M}^{-1} \boldsymbol{G}_k^T \boldsymbol{S})^{-1} \boldsymbol{S} \boldsymbol{y} \\ &= \boldsymbol{S} (\boldsymbol{I}_s - \boldsymbol{A}_n)^{-1} \boldsymbol{S} \boldsymbol{y} \approx \boldsymbol{S} \left( \sum_{j=0}^{I-1} \boldsymbol{A}_k^j \right) \boldsymbol{S} \boldsymbol{y} \end{aligned} \tag{2.26}$$

for a modest value of the positive integer $I$ which is typically chosen as 4 or 8. The parameter $I$ is known as `lincs-order` in GROMACS. It is important to appreciate the fact that while there are known molecules for which this series does not converge —see the original paper [68]—, there is ample experience to suggest that the approximation works well in practice.

P-LINCS is a straightforward parallelization of LINCS, leveraging the fact that the matrix-vector multiplications in equation (2.26) can be efficiently distributed across multiple computational work units.

The convergence of the Neumann series in (P-)LINCS is at best linear, and there exist physically relevant cases for which it does not converge at all [68], [69], [103]. As a result, achieving high accuracy when solving the equations can be computationally expensive. Additionally, the method can break down when coupled angular constraints are present, a limitation explicitly noted in the GROMACS manual [80]. In extreme cases, such as those reported in [103], these convergence issues are severe enough that (P-)LINCS must be replaced by SHAKE.

### 2.2.5. Other Constraint Solvers

Beyond SHAKE and (P-)LINCS, a wide range of constraint algorithms and methods has been proposed over the years [88], [89], [96], [99], [100], [104]–[118]. Among these, RATTLE [104] and SETTLE [105] stand out due to their widespread adoption and relevance.

RATTLE enforces constraints on both positions and velocities, in contrast to SHAKE and (P-)LINCS, which constrain only positions. Although the algorithm remains relevant, most production molecular dynamics simulations reported in the literature apply constraints exclusively to positions.

SETTLE is specifically designed to impose constraints on water molecules by treating them as rigid bodies. For this restricted but ubiquitous case, SETTLE is significantly more efficient than applying general-purpose solvers such as SHAKE or (P-)LINCS. Since water is commonly used as the solvent in molecular dynamics simulations, SETTLE is typically selected as the solvent-constraint solver.

Many other proposed constraint algorithms suffer from limitations that render them either slower than the original SHAKE in practice or applicable only to systems with very specific structural properties.

M-SHAKE [112], for example, formulates the constraint equations as dense linear systems and solves them using Gaussian elimination with partial pivoting. This approach is inherently limited to small molecules, as the computational cost of computing an LU factorization of a dense matrix of dimension $n$ scales as $\mathcal{O}(n^3)$.

MILC-SHAKE [114] computes an LU factorization of a tridiagonal matrix rather than a fully dense matrix, allowing the LU factorization to be computed with linear time complexity. The related MILCH-SHAKE algorithm [115] follows a similar strategy but is tailored specifically to $n$-alkanes. Despite their favorable asymptotic complexity, both methods are restricted to linear chains and n-alkanes, respectively, and therefore lack general applicability.

The approaches proposed in [99], [100] approximate the constraint matrices as symmetric positive semi-definite and solve the resulting systems using the conjugate gradient method. These methods offer two principal advantages: the relative ease of parallelizing the CG algorithm and the potential to accelerate convergence through preconditioning. However, a significant drawback is the difficulty of constructing preconditioners whose effectiveness can be mathematically guaranteed. Despite their conceptual appeal, these methods have seen limited adoption in practical molecular dynamics simulations.

Further algorithms have been developed to impose constraints on degrees of freedom other than bond lengths, such as bond angles [88], [89], [111] and dihedral angles [96], [118]. More sophisticated angular constraint expressions have been proposed [88], [89] to move beyond simple distance constraints; these methods rely on solving equations one after another like SHAKE. This approach converges slowly and is challenging to parallelize. The execution time of $\theta$-SHAKE [88] has the same order of magnitude as SHAKE's, which is significant for constraints on bond angles. Moreover, its efficiency relies on the low connectivity of the bonds of the tackled molecules.

Overall, SHAKE and (P-)LINCS remain the most widely used general-purpose constraint

solvers, whereas RATTLE and SETTLE are primarily employed in more specific contexts, namely for enforcing constraints on both positions and velocities or for constraining water molecules, respectively. Other proposed algorithms have not seen widespread adoption, for a variety of practical and methodological reasons.

## 2.3. The Need to Solve the Constraint Equations Accurately

Presently, MD simulations usually accept a large relative error when solving the constraint equations. In GROMACS, the default SHAKE tolerance (`shake-tol`, defined as the maximum relative error allowed when solving constraints) is $10^{-4}$. There exists no such demanded tolerance for P-LINCS, which has been said to cause unphysical dynamics and temperatures of thousands of Kelvin due to fast rotation of $NH_3$ groups [119]. Nevertheless, it is generally assumed that the average accuracy of P-LINCS with the default GROMACS parameters is typically similar to SHAKE's. Other MD packages, like Amber [120], are a bit more demanding (Tol = $10^{-5}$) [121]. Notwithstanding, tighter satisfaction of the constraints is sometimes imposed, frequently in simulations performed in the NVE (microcanonical) ensemble [103], [121]–[131], though also in simulations with a thermostat [103], [132]–[137] (NVT, NPT). For example, in references [103], [121]–[137] the authors set a tolerance Tol for the constraints between $10^{-7}$ and $10^{-10}$ (except [124], which uses Tol = $10^{-12}$).

As demonstrated later in this chapter and in [46], insufficient accuracy in solving the constraints is equivalent to applying undesired, spurious, and random external forces, which generate a non-negligible drift in the energy of the simulated system, thereby undermining the trustworthiness of simulations in the microcanonical (NVE) ensemble. This has led several studies to state that constraint equations must be solved down to the limit of machine precision [95], [138].

Simulations with a thermostat (NVT, NPT ensembles) also present such undesired energetic drifts, which contribute to making the conserved quantity (also called conserved energy) of the thermostat (e.g., Nosé-Hoover, V-rescale) become non-conserved. Due to this, there is no guarantee that the equations of the thermostat are satisfactorily solved, hence there is no guarantee that the simulation corresponds to the sought ensemble, which makes its reliability drop [139], [140]. Moreover, drift introduced by the inaccuracy of the constraint solution distorts the time required to reach the target temperature ($T$). This can be observed in the V-rescale thermostat [140], which calculates a rescaling factor for the velocities that, on average, is expected to make the temperature of the system approximately equal to the desired temperature $T$ after a simulated time $\tau_T$ (being $\tau_T$ an input parameter of the simulation). However, due to the inaccurate constraint solving, an additional amount of energy is injected into, or extracted from, the system, which makes the average time for reaching $T$ deviate from $\tau_T$ in an unknown manner. In addition, imposing constraints inaccurately systematically misestimates bond lengths and makes them randomly change their values in an irregular manner. Moreover, artifactual regimes arise as periods where the averages of the lengths of the bonds differ from the values set by constraints, which alternate with periods where the bond lengths remain nearly

unchanged [46].

Other research works have also found non-negligible distorting effects due to inaccurate constraint solving: in [70] it is stressed that GROMACS' default parameters lead to non-converged results and make temperatures of the simulated system unreliable, which is fixed if constraints are accurately solved. Other research indicates that inaccuracy in constraints can lead to wrong densities [141] or to collective motion artifacts, like spurious phase transitions from liquid to an icy state [122].

In [50], [51], we study how introducing inaccuracies in molecular dynamics simulations, not only during the constraint phase, undermines the validity of Richardson extrapolation as a tool for estimating discretization errors.

The array of inconveniences due to inaccurate constraint solving can be largely mitigated if the constraint forces are calculated with the largest possible accuracy (for the chosen numerical precision) instead of the default values in GROMACS of $10^{-4}$—for SHAKE—or undetermined—for LINCS—. However, the state-of-the-art algorithms SHAKE and (P-)LINCS exhibit slow convergence, so achieving high constraint accuracy with these methods incurs a substantial computational cost.

Appendix A provides a more detailed discussion supporting the case for solving the constraint equations as accurately as the hardware will allow.

## 2.4. The ILVES Algorithms

In general, applying direct solvers to linear systems requires $\mathcal{O}(n^3)$ floating-point operations, where $n$ is the number of equations. In the context of constraint solvers, $n$ corresponds to the number of constraints [142]. However, the linear systems arising from constraints in molecular dynamics exhibit a highly structured sparsity pattern that directly reflects the linear and sparse topology of molecular graphs. As a result, for general molecules, direct solvers can be applied in $\mathcal{O}(n)$ time by exploiting this structure [142]–[144]. Building on this observation, the ILVES algorithms presented in this thesis exploit molecular topology to significantly accelerate the solution of the constraint equations compared to state-of-the-art algorithms.

ILVES is a family of algorithms for imposing constraints in molecular dynamics that compute discrete approximations to the solution of the same system of differential-algebraic equations as SHAKE. The key difference lies in the numerical method employed. While SHAKE relies on the nonlinear Gauss–Seidel method, which converges locally but only linearly, the ILVES algorithms combine Newton's method with direct solvers, yielding substantially faster convergence rates.

In this chapter, we present three algorithms belonging to the ILVES family: ILVES-PC ("peptide chains"), ILVES-M ("main"), and ILVES-F ("fast"). These algorithms represent successive steps toward generality and performance, while preserving the core idea of exploiting molecular sparsity through direct solvers.

We originally proposed ILVES-PC in [46] as a proof of concept demonstrating that Newton's method combined with direct solvers can be efficiently applied to molecular constraint equations. ILVES-PC is restricted to peptides and proteins with all-bond

constraints and is therefore not intended for general production simulations. It supports shared-memory parallelism and employs a custom thread-parallel Gauss–Jordan elimination based on the Schur complement method [145] to solve the subsystems corresponding to individual amino acids in a protein.

ILVES-M and ILVES-F, introduced in [47], generalize the approach of ILVES-PC to an arbitrary set of atoms with constrained bonds. As such, they support the same use cases as state-of-the-art constraint solvers and are suitable for production molecular dynamics simulations. Both algorithms leverage shared-memory parallelism, distributed-memory parallelism, and SIMD vectorization.

ILVES-M solves the same system of differential-algebraic equations as ILVES-PC and SHAKE, but employs a custom general-purpose direct solver. Similarly to ILVES-PC, ILVES-M exploits shared-memory parallelism through a custom thread-parallel LU factorization based on the Schur complement method [145]. For distributed-memory parallelism, the solver is extended using the Overlapping Partitioning Method (OPM) [146]. When executed across multiple domains, ILVES-M therefore behaves as a quasi-Newton method. Nevertheless, its convergence remains extremely fast, typically requiring very few, and often zero, additional iterations compared to single-domain execution. Although ILVES-M is fully general, unlike ILVES-PC, its performance is only approximately 10% lower, largely due to further optimizations, most notably exploiting SIMD vectorization.

ILVES-F is a variant of ILVES-M that further reduces computational cost by employing a fixed symmetric approximation of the coordinate matrix [111]. This symmetry allows the replacement of the LU factorization with an LDLT factorization, improving computational efficiency. Moreover, since the LDLT factorization needs to be computed only once per time step, the overall computational cost is nearly halved. Due to the symmetric approximation, ILVES-F behaves as a quasi-Newton method even in shared-memory executions. Nonetheless, its convergence is exceptionally fast [48], [49], delivering better performance than ILVES-M in most scenarios.

In this section, we provide a detailed description of ILVES-PC, ILVES-M, and ILVES-F, as well as their parallel implementations, which have been integrated into Gromacs. This software framework is used throughout this work to analyze the performance and accuracy of the proposed algorithms. Although the implementations are fine-tuned for Gromacs, they can be migrated to other molecular dynamics packages, such as NAMD [101], [102], Amber [120], Charmm [147], Lammps [148], or Desmond [149].

### 2.4.1. Newton's Method Applied to Constraint Equations

We begin by recalling Newton's method for solving systems of nonlinear equations, which serves as the basis for the constraint solvers developed in this work. We then adapt this general framework to the specific nonlinear constraint equations found in molecular dynamics.

Let $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$ be a differentiable function and consider the problem of solving the following system of equations:

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}, \tag{2.27}$$

with respect to $\boldsymbol{x} \in \mathbb{R}^n$. If the Jacobian $\boldsymbol{F}(\boldsymbol{x})$ of $\boldsymbol{f}$ at any point $\boldsymbol{x}$ is nonsingular, then Newton's method is given by

$$\boldsymbol{F}(\boldsymbol{x}_l)\boldsymbol{z}_l = \boldsymbol{f}(\boldsymbol{x}_l)\,, \tag{2.28a}$$

$$\boldsymbol{x}_{l+1} = \boldsymbol{x}_l - \boldsymbol{z}_l\,. \tag{2.28b}$$

where the initial approximation $\boldsymbol{x}_0$ must be chosen by the user. In general, Newton's method will converge locally and quadratically to a zero $\boldsymbol{z}$ of $\boldsymbol{f}$ if the Jacobian $\boldsymbol{F}(\boldsymbol{z})$ at $\boldsymbol{z}$ is nonsingular, i.e., there exists a constant $C$ such that

$$\frac{\|\boldsymbol{z} - \boldsymbol{x}_{l+1}\|_2}{\|\boldsymbol{z} - \boldsymbol{x}_l\|_2^2} \to C, \quad l \to \infty, \quad l \in \mathbb{N}. \tag{2.29}$$

for all choices of $\boldsymbol{x}_0$, provided that $\|\boldsymbol{z} - \boldsymbol{x}_0\|_2$ is sufficiently small.

We now formulate Newton's method for the nonlinear constraint equation (2.8c). To this end, we first use the chain rule of differentiation to compute the Jacobian of the function $\boldsymbol{\lambda} \to \boldsymbol{g}(\boldsymbol{\phi}_k(\boldsymbol{\lambda}))$ and find that

$$\frac{\partial}{\partial \boldsymbol{\lambda}}(\boldsymbol{g} \circ \boldsymbol{\phi}_k)(\boldsymbol{\lambda}) = \boldsymbol{G}(\boldsymbol{\phi}_k(\boldsymbol{\lambda}))\frac{\partial \boldsymbol{\phi}_k}{\partial \boldsymbol{\lambda}}(\boldsymbol{\lambda}) = -h^2\boldsymbol{G}(\boldsymbol{\phi}_k(\boldsymbol{\lambda}))\boldsymbol{M}^{-1}\boldsymbol{G}(\boldsymbol{q}_k)^T \tag{2.30}$$

which is why we define the matrix function $\boldsymbol{A} : \mathbb{R}^{3s} \times \mathbb{R}^{3s} \to \mathbb{R}^{n\times n}$ given by

$$\boldsymbol{A}(\boldsymbol{x}, \boldsymbol{y}) := -h^2\boldsymbol{G}(\boldsymbol{x})\boldsymbol{M}^{-1}\boldsymbol{G}(\boldsymbol{y})^T\,. \tag{2.31}$$

In general, each matrix $\boldsymbol{A}(\boldsymbol{x}, \boldsymbol{y}) = [a_{ij}]$ is both sparse and structurally symmetric because $a_{ij} = a_{ji} = 0$ if the $i$th and $j$th bond do not share an atom [46]. However, we cannot expect that $\boldsymbol{A}$ is symmetric in the sense that $\boldsymbol{A} = \boldsymbol{A}^T$, unless, of course, $\boldsymbol{x} = \boldsymbol{y}$.

Newton's method for solving equation (2.8c) with respect to the Lagrange multipliers $\boldsymbol{\lambda}_k$ can be written as

$$\boldsymbol{A}\big(\boldsymbol{\phi}_k(\boldsymbol{\lambda}_{k,l}), \boldsymbol{q}_k\big)\boldsymbol{z}_{k,l} = \boldsymbol{g}\big(\boldsymbol{\phi}_k(\boldsymbol{\lambda}_{k,l})\big), \tag{2.32a}$$

$$\boldsymbol{\lambda}_{k,l+1} = \boldsymbol{\lambda}_{k,l} - \boldsymbol{z}_{k,l}, \tag{2.32b}$$

It is common to initialize the search for $\boldsymbol{\lambda}_k$ using $\boldsymbol{\lambda}_{k,0} = \boldsymbol{0}$, and we have used this value to good effect. We emphasize that equation (2.32a) is a *linear* equation with respect to the unknown vector $\boldsymbol{z}_{k,l}$ and the solution of several such systems is usually required to advance the simulation from time $t_k$ to time $t_{k+1}$.

### 2.4.2. ILVES-PC

ILVES-PC is specifically designed to solve the constraint equations arising in peptides or protein simulations with all bonds constrained. Developing code for given types of molecules is the extension to software of a concept that has already proven to be very successful with hardware. The Anton supercomputers were conceived to perform MD simulations of proteins and other biological molecules [150]. Their specific design greatly enhances the performance of these systems compared with general-purpose computers.

ILVES-PC also utilises specific features of widely simulated systems in order to increase the performance compared with general-purpose algorithms.

Peptide chains (peptides and proteins) are polymers of variable length formed by repeating blocks of atoms. Each of the building blocks of a peptide chain (residues or amino acids) has a given connectivity pattern, which is found essentially unchanged in biological molecules (occasionally further atoms can be attached to the protein, e.g. in glycoproteins, or the protein structure can get modified, e.g. at the chromophore of the Green Fluorescent Protein; in addition, hydrogen atoms in carbon rings of side chains can lie in alternative positions). However, 20 given—proteinogenic—amino acid connectivities largely dominate the structure of peptides and proteins.

We can make an abstraction of a peptide chain as a graph $G = (V, E)$. The vertices $V = \{1, 2, 3 \ldots, n\}$ represent the atoms and the edges $E \subseteq V \times V$ represent the bonds. Specifically, we have $(a, b) \in E$ if and only if atoms $a$ and $b$ are bonded. The graph is undirected because $(a, b) \in E$ if and only if $(b, a) \in E$. From $G$ we can build a new graph $L(G)$ where each vertex represents a bond and two bonds are connected if and only if they share an atom. The graph $L(G)$ is known as the line-graph of $G$.

The coefficient matrix of the linear system solved by ILVES-PC, i.e., the matrix $\boldsymbol{A}$ of Equation (2.32a), has the same structure as the adjacency matrix of the line graph $L(G)$ of the graph $G$ that represents the peptide chain. Since the peptide chain is composed of 20 different building blocks, the main features of the matrix $\boldsymbol{A}$ can also be described using 20 different submatrices. Each submatrix represents the constraint interactions specific to one of the 20 proteinogenic amino acids.

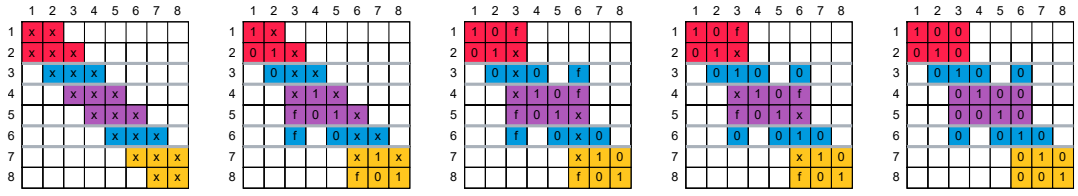
We have written subroutines for solving the linear subsystems corresponding to each of these submatrices. To solve the entire linear system, we iterate over the subsystems of the linear system, calling the required subroutine. We ensure that submatrices corresponding to identical amino acids have the same structure by always numbering the bonds of each amino acid in the same order. This allows us to generate loop-free subroutines that store the relevant data contiguously in memory and do direct memory access instead of relying on the auxiliary data structures and the indirect memory access patterns that are so typical of direct solvers for sparse linear systems. By choosing the bond numbering of each amino acid, we can reduce the fill-in during the factorization of the matrix. Fill-ins are entries that are exactly zero in the original matrix, but become non-zero during the factorization process.

In order for ILVES-PC to work with peptide chains with arbitrary bond numbering, the topology of each protein is analyzed during the initialization phase of the simulation to identify individual amino acid residues. The bonds are then renumbered to match the ordering required by the ILVES-PC implementation.

ILVES-PC exploits shared-memory parallelism by assigning a subset of amino acids of a peptide chain to different hardware threads. We ensure that each thread is assigned a similar number of matrix elements. To concurrently apply Gauss-Jordan elimination to the submatrices corresponding to different amino acids, we rely on the Schur complement method [145].

Consider the example presented in Figure 2.2. We want to perform the Gauss-Jordan elimination at the banded coordinate matrix of Figure 2.2a (which exemplifies $\boldsymbol{A}$) using

three threads or workers (magenta, purple, and yellow). First (Figure 2.2b), each of the threads works with its own private submatrix, trying to fill the lower left-hand (subdiagonal) corner with zeros and the diagonal with ones. The threads also update the shared (blue) rows using mutual exclusion (mutex) locks. They repeat this step (Figure 2.2c) in the upper right-hand corner of their submatrix. These two steps may produce fill-ins. Then, the master thread processes the submatrix comprising the shared (blue) rows (Figure 2.2d), while the other two threads wait until this step is complete. Finally (Figure 2.2e), all threads work in parallel to clean the fill-ins produced in steps Figure 2.2b and Figure 2.2c. In the case of ILVES-PC, the thread-private data corresponds to constraints within a given amino acid, while the shared rows correspond to the constraint that connects two amino acids (i.e., a peptide bond).



(a) (b) (c) (d) (e)

Figure 2.2.: Steps to solve a banded linear system in parallel through Gauss-Jordan elimination and the Schur complement method. An analogous approach applies to solving the system with LU/LDLT decompositions and backward/forward substitution. The entries of the matrix are represented with $x$ and the fill-ins with an $f$. The magenta, purple, and yellow entries are private to threads, while the blue entries are shared between threads.

Since ILVES-PC has exact prior knowledge of the structure of the linear systems it solves, it can avoid coarse-grained synchronization mechanisms such as global thread barriers. Instead, this structural awareness enables the use of fine-grained synchronization mechanisms. We use lightweight mutex locks to protect the shared rows of the linear system from data races, so that each mutex involves only a pair of threads. Likewise, during the update phase at the end of each Newton step, the positions and velocities of each atom shared by multiple threads are protected by individual mutex locks.

The ILVES-PC code, integrated into GROMACS 2021.0, is available at https://github.com/LorienLV/_PAPER_ILVES-PC.

### 2.4.3. ILVES-M and ILVES-F

ILVES-M solves the same system of equations (2.8) as ILVES-PC and SHAKE, applying Newton's method to compute the solution of the constraint equations. Unlike ILVES-PC, however, ILVES-M employs a custom general-purpose linear solver that works with any input set of atoms and constrained bonds, making it applicable to any molecular dynamics simulation.

ILVES-F also solves the system of equations (2.8) as SHAKE, but it uses a quasi-Newton

method to solve the constraint equations. In this case, the central matrix $\boldsymbol{A}\big(\boldsymbol{\phi}_k(\boldsymbol{\lambda}_{k,l}), \boldsymbol{q}_k^0\big)$ is a function of $l$, but it can be approximated by a constant matrix, i.e.,

$$\boldsymbol{A}\big(\boldsymbol{\phi}_k(\boldsymbol{\lambda}_{k,l}), \boldsymbol{q}_k^0\big) \approx \boldsymbol{A}\big(\boldsymbol{q}_k^0, \boldsymbol{q}_k^0\big) \tag{2.33}$$

simply because $\boldsymbol{\phi}_k(\boldsymbol{\lambda}) = \boldsymbol{q}_k + \mathcal{O}(h)$. It is therefore natural to replace Newton's method with the quasi-Newton method

$$\boldsymbol{A}\big(\boldsymbol{q}_k^0, \boldsymbol{q}_k^0\big)\boldsymbol{z}_{k,l} = \boldsymbol{g}\big(\boldsymbol{\phi}_k(\boldsymbol{\lambda}_{k,l})\big)\,, \tag{2.34}$$

$$\boldsymbol{\lambda}_{k,l+1} = \boldsymbol{\lambda}_{k,l} - \boldsymbol{z}_{k,l} \tag{2.35}$$

This approach is standard within the field of numerical analysis and different names such as the "simplified Newton's method" [151] or the "chord method" [152] are used. It is the foundation of the Constant Constraint Matrix approach used in [111] in molecular dynamics.

We investigated the use of a quasi-Newton method in place of Newton's method in [48], [49]. In those works, we show that although quasi-Newton methods remove the quadratic convergence of Newton's method, achieving fast linear convergence is sufficient to reach the desired solution accuracy without significantly increasing the number of iterations, given the finite precision of modern computers (e.g., 64-bit double precision or 32-bit single precision). As shown in the performance results presented later, this approach enables ILVES-F to outperform ILVES-M in most practical scenarios.

ILVES-M and ILVES-F leverage three levels of parallelism: distributed memory parallelism, shared memory parallelism, and vectorization.

In distributed-memory simulations with Gromacs, the domain decomposition assigns a disjoint set of atoms (and their constraints) to each domain. Additionally, each domain stores a copy of the atoms that lie outside the domain but are connected to local atoms within a distance of up to $I$ bonds, where $I$ is a user-defined parameter (`lincs-order` in the case of P-LINCS). Given this data layout, in order to exploit distributed memory parallelism in ILVES-M and ILVES-F, we apply a natural extension of the Overlapping Partitioning Method (OPM) [146] from banded to general sparse systems. In our OPM implementation, each domain computes an independent Newton step using both its local and neighbor atoms without any inter-domain communication during the solve. This Newton step is computed using a general multi-threaded LU/LDLT factorization based on the Schur complement [145], described later. After each Newton iteration, each domain broadcasts the updated coordinates of its local atoms to all domains that include those atoms as neighbors, ensuring that every domain holds consistent, up-to-date positions before the next iteration begins. By applying OPM (only in simulations with multiple domains), ILVES-M behaves as a quasi-Newton method similar to ILVES-F. However, the number of iterations to convergence is rarely affected.

It is important to note that in distributed-memory simulations, Gromacs frequently rebuilds the list of atoms assigned to each domain, a data structure known as the neighbor list. This list determines which atoms are considered local to each domain and which neighboring atoms must be included for calculations such as constraints. The frequency of these updates is controlled by the `nstlist` parameter, whose default value is 10 steps.

Each time the neighbor list is recomputed, the constraint solver must be reinitialized to reflect possible displacements of atoms between domains. Since the neighbor list is frequently recomputed, the performance of the initialization has a significant impact on the overall algorithm performance. Hence, we designed and implemented an initialization of ILVES-M and ILVES-F, which minimizes the time spent in this stage.

We reuse and generalize the representation of atoms and bonds introduced in the description of ILVES-PC. In each domain, the atoms and bonds handled by the constraint solver can be represented as an undirected graph $G = (V, E)$, where vertices $V$ represent the atoms connected with at least one constrained bond and the edges $E \subseteq V \times V$ represent the constrained bonds. Since the simulation can contain more than one molecule, and it is common to constrain only a subset of the bonds, for example, bonds that connect a hydrogen bond (H-bonds), $G$ is usually a disconnected graph. From $G$, we can compute its line-graph $L(G)$ in which each vertex represents a constrained bond, and two constrained bonds are connected if they share one vertex in $G$. The adjacency matrix of the line-graph has the same sparsity pattern as the coordinate matrices of the linear systems solved by ILVES-M and ILVES-F.

The graph $L(G)$ can be partitioned such that all the edges of $L(G)$ that do not cross partitions comprise each thread's private partition, while edges that connect vertices of $L(G)$ which lie in different private partitions comprise the shared partition. The data in the private partitions can be processed simultaneously, as they are independent of one another. In contrast, the data in the shared partition must be processed separately, in mutual exclusion from the private partitions, to ensure correctness and avoid conflicts.

While both P-LINCS and ILVES-M/F employ a partitioning of $L(G)$ for parallel execution, their methodologies differ significantly. P-LINCS partitions $L(G)$ directly dividing its vertices into roughly equal sets, followed by computation to identify edges crossing partitions. On the other hand, ILVES-M/F tries to minimize the edges crossing partitions using a greedy algorithm to partition $G$ rather than $L(G)$. This strategy is fully general, unlike ILVES-PC, which relies on detecting individual amino acids in proteins to define partitions.

Graphs in MD exhibit very low connectivity and, additionally, GROMACS assigns close indices to connected atoms. Our partitioning algorithm sequentially assigns vertices of $G$ to partition $p$ while the partition size remains below a target value plus an offset. The final vertex in partition $p$ is chosen based on the number of edges cut and the similarity of the partition size to the target size. This greedy algorithm runs in $\mathcal{O}(|E|)$ time, where $|E|$ is the number of edges in $G$, and produces partitions comparable to those generated by general but slower graph-partitioning libraries like METIS [153], often cutting a similar or smaller number of edges relative to the number of partitions. Our partitioning approach produces significantly fewer cuts than P-LINCS', which results in notably less serial work and less synchronization between threads in cases where zero-cut partitions are possible, such as simulations of several small molecules.

The pseudocode of our greedy approach for partitioning a graph into $k$ partitions is presented in Algorithm 2.1. For the results discussed in this study, the parameter `maxDeviation` was set to 0.05, and `cutPenalty` was assigned a value of 10.

After computing the partitioning of $L(G)$, we use a modified parallel implementation

**Algorithm 2.1:** Greedy algorithm for partitioning a graph into $k$ pieces

**Input:** $k$: number of partitions, neighs: sorted adjacency list where neighs[$v$] contains the list of neighboring vertices for vertex $v$
**Output:** Array ids with the partition id of each vertex

```
1  Function KWAY_PART(k, neighs) begin
2      Definition: binary_search(v, l) return the position of v in the list l.
3      maxDeviation, cutPenalty ← problem-specific constants
4      ids ← vector of size |neighs| initialized to −1
5      v ← 0
6      for p ← 0 to k − 1 do
7          targetPsize ← (|neighs| − v)/(k − p)
8          maxPsize ← targetPsize × (1 + maxDeviation)
9          bestVertex ← −1
10         bestScore ← ∞
11         psize ← 0
12         ncuts ← 0
13         while v < |neighs| and psize ≤ maxPsize do
14             if ids[v] = p then
15                 ncuts ← ncuts − 1
16             else
17                 ids[v] ← p
18             psize ← psize + 1
19             i ← binary_search(v, neighs[v])
20             for neigh in neighs[v] from index i + 1 onward do
21                 if ids[neigh] < p then
22                     ids[neigh] ← p
23                     ncuts ← ncuts + 1
24             score ← |psize − targetPsize| + ncuts × cutPenalty
25             if score < bestScore then
26                 bestVertex ← v
27                 bestScore ← score
28             v ← v + 1
29         v ← bestVertex + 1
30     for v ← v to |neighs| − 1 do
31         ids[v] ← k − 1
32     return ids
```

of the Approximate Minimum Degree ordering algorithm (AMD) [154] to minimize the fill-ins (zero entries that become non-zero) in the part of the linear system corresponding to each partition. This step completes the initialization of ILVES-M/F, which is then ready to apply the constraints in each step until reinitialization is required after GROMACS recomputes its neighbor list.

ILVES-M and ILVES-F enforce constraints at each step through three primary phases: i) constructing the relevant linear systems and calculating the maximum relative error, ii) solving the linear systems, and iii) updating atomic positions. These phases comprise a Newton iteration and are iteratively executed until the desired tolerance for the maximum relative error is reached.

During the first phase, each thread constructs the corresponding portion of the linear system for its private partition, while the master thread also handles the shared partition. In this phase, the private and shared partitions can be computed concurrently. The computation of the maximum relative error overlaps with the construction of the linear system. In distributed-memory simulations, domains exchange their maximum relative errors after this step is complete. Note that for ILVES-F, the coordinate matrix of the linear system of equations to solve ($\boldsymbol{A}$) remains constant for all iterations and is thus computed just once per simulated time. This step is vectorized using the GROMACS SIMD intrinsics interface.

Next, the Lagrange multipliers $\boldsymbol{\lambda}$ are calculated by solving the linear system in parallel through LU/LDLT factorization for ILVES-M/F. The factorization is followed by forward and backward substitutions. ILVES-F requires a single LDLT factorization per step because $\boldsymbol{A}$ remains unchanged for all the iterations in a given simulated time. This process follows the same steps previously detailed for ILVES-PC in Figure 2.2.

Finally, the atomic positions are updated. Updates to the positions of atoms in private partitions are processed in mutual exclusion to updates of atoms in the shared partition. In distributed-memory simulations, the updated atomic positions are exchanged between domains, ensuring that each domain has the latest positions of its neighboring atoms.

ILVES-M and ILVES-F require coarser-grained synchronization mechanisms than ILVES-PC and also rely on storing and accessing auxiliary data structures to handle the linear systems. In contrast, ILVES-PC knows the structure of the linear systems in advance and does not require any auxiliary structures. Despite this, ILVES-M and ILVES-F have significantly less code, which improves maintainability, enables partial vectorization, and facilitates distributed-memory parallelism, features that would require substantially more effort to implement in ILVES-PC. Furthermore, ILVES-M and ILVES-F are fully general and can handle any set of constraints, making them directly applicable to current and future production simulations.

The ILVES-M and ILVES-F code, integrated into GROMACS 2021.0, is available at https://github.com/LorienLV/_PAPER_ILVES.

## 2.5. Experimental Setup

We developed two modified versions of GROMACS 2021.0. One integrates ILVES-PC and is available at https://github.com/LorienLV/_PAPER_ILVES-PC. The other integrates ILVES-M and ILVES-F and is available at https://github.com/LorienLV/_PAPER_ILVES.

To ensure a fair comparison, we generated a modified version of P-LINCS that guarantees constraints are satisfied within a given tolerance, as SHAKE or the ILVES algorithms; we refer to this variant as MP-LINCS. This was accomplished by repeating P-LINCS' correction phase, controlled by the `lincs-iter` parameter in the original implementation, until the desired maximum relative error in solving the constraints is met. Although this modification introduces additional synchronization points in parallel executions, potentially affecting performance, it is important to note that it is straightforward to modify ILVES to execute a fixed number of iterations without checking the tolerance, thereby eliminating the same synchronization points introduced in MP-LINCS.

While GROMACS' implementation of SHAKE itself is not parallelized, it can still be used in parallel simulations without domain decomposition, i.e., without distributed-memory parallelism. In such cases, SHAKE runs on a single thread, while the rest of the simulation proceeds in parallel. This approach was used to obtain the results reported for SHAKE in parallel simulations.

Table 2.1.: Main features of the computing nodes used to execute the simulations.

| | |
|---|---|
| Processor | 2 × Intel Xeon Platinum 8480+ |
| Cores | 2 × 56 |
| Frequency | 2 GHz |
| L1i cache (I + D) | 32 KB + 48 KB (per core) |
| L2 cache | 2 MB (per core) |
| LLC | 105 MB (shared) |
| Main Memory | 256 GB DDR5 (16 × 16 GB 4800 MHz DIMMs) |
| Interconnection | ConnectX-7 NDR200 InfiniBand (100 Gb/s per node) |

The simulations presented in this chapter were performed on a multi-node cluster. Each node is equipped with two Intel Xeon Platinum 8480+ chips, each with 56 cores. The main features of the nodes are summarized in Table 2.1. GROMACS 2021.0 was built on this system using GCC version 10.1.0. Unless stated otherwise (primarily in the performance results) GROMACS was compiled and executed in double precision (FP64), which increases numerical accuracy at the expense of execution speed.

## 2.6. Simulations

We have conducted simulations on a diverse set of molecular systems, including three solvated proteins, a solvated protein-DNA complex, an organic solvent, and a membrane protein embedded in a lipid bilayer surrounded by water. Table 2.2 summarizes the key characteristics of the systems, while Figure 2.3 provides a visual representation. Biological

macromolecules, such as DNA and proteins, have long been of interest for medical and biotechnological purposes. Benzene, as a representation of small molecules, has important chemical and industrial applications [155]. Proteins embedded in a lipid bilayer include channels, transporters, and signal receptors, among others, which play a vital role in cell biology and medical research [156]. This diversity of molecules allowed us to evaluate the constraint solvers across a wide range of molecular structures.

Table 2.2.: Molecular systems simulated in this work. In the Force Field column, C22+CMAP is Charmm22+CMAP, C36m is Charmm36m, and C36 is Charmm36. Although ff19SB defines residue-specific CMAP corrections, the protein topology of the DNA-protein system contains a generic CMAP matrix (taken from the `frcmod.ff19SB_XXX` internal file of AmberTools24) for all residues due to conversion with `amb2gro_top_gro.py`, ensuring compatibility with pre-2025 GROMACS versions.

| **System** | **# Molec. entities (PDB ID)** | **# Mol.** | **# Resid./ Atoms** | **# Constr. all-bonds/ H-bonds** | **Force Field/ Water Model** |
|---|---|---|---|---|---|
| Solvated barnase [157] | barnase (1A2P) | 1 | 108/1,705 | 1,721/836 | C22+CMAP [158] |
| | Water | 62,848 | 188,544 | — | Tip3p [159] |
| | $NA^+$, $CI^-$ | 4/11 | 15 | — | C22+CMAP [158] |
| Solvated nuclease [160] | nuclease (2SNS) | 1 | 149/2,398 | 2,459/1,224 | C22+CMAP [158] |
| | Water | 115,602 | 346,806 | — | Tip3p [159] |
| | $CI^-$ | 10 | 10 | — | C22+CMAP [158] |
| Solvated COVID protease [161] | protease (5R7Y) | 1 | 304/4,645 | 4,697/2,297 | C22+CMAP [158] |
| | Water | 61,958 | 185,874 | — | Tip3p [159] |
| | $NA^+$, $CI^-$ | 8/4 | 12 | — | C22+CMAP [158] |
| Pure benzene [162] | benzene | 2000 | 24,000 | 24,000/12,000 | Gromos54a7 [163] |
| Solvated DNA-proteins [164] | histones (3LEL) | 16 | 1,540/25,257 | 25,933/13,078 | ff19SB [165] |
| | DNA | 4 | 588/18,712 | 20,178/6,670 | OL21 [166] |
| | Water | 83,872 | 251,616 | — | OPC4 [167] |
| | $NA^+$ | 428 | 428 | — | Ions [168], [169] |
| Aquaporin 1 embedded in lipid bilayer [170] | Aqp1 (1J4N) | 4 | 996/15,072 | 15,480/7,755 | C36m [77] |
| | MMPC | 1,129 | 133,222 | 131,837/81,193 | C36 [76] |
| | Water | 80,598 | 241,794 | — | Tip3p [159] |
| | $NA^+$, $CI^-$ | 223/235 | 458 | — | C36 [76] |

Below, we outline the procedures on which the simulations are based. All the systems, except the human aquaporin-1 embedded in a lipid bilayer, were prepared through a standard protocol involving solvation, minimization, heating, and equilibration. The most important parameters used in the preparation and also in the production phase of the simulations are listed in Table 2.3. The membrane protein system was prepared using the membrane-builder wizard from the `CHARMM-GUI` online tool [171]–[174]. We used different force fields for the simulations (Table 2.2), and explicit Tip3p water molecules [159] were used to solvate the macromolecular systems, except for the protein-DNA complex, which

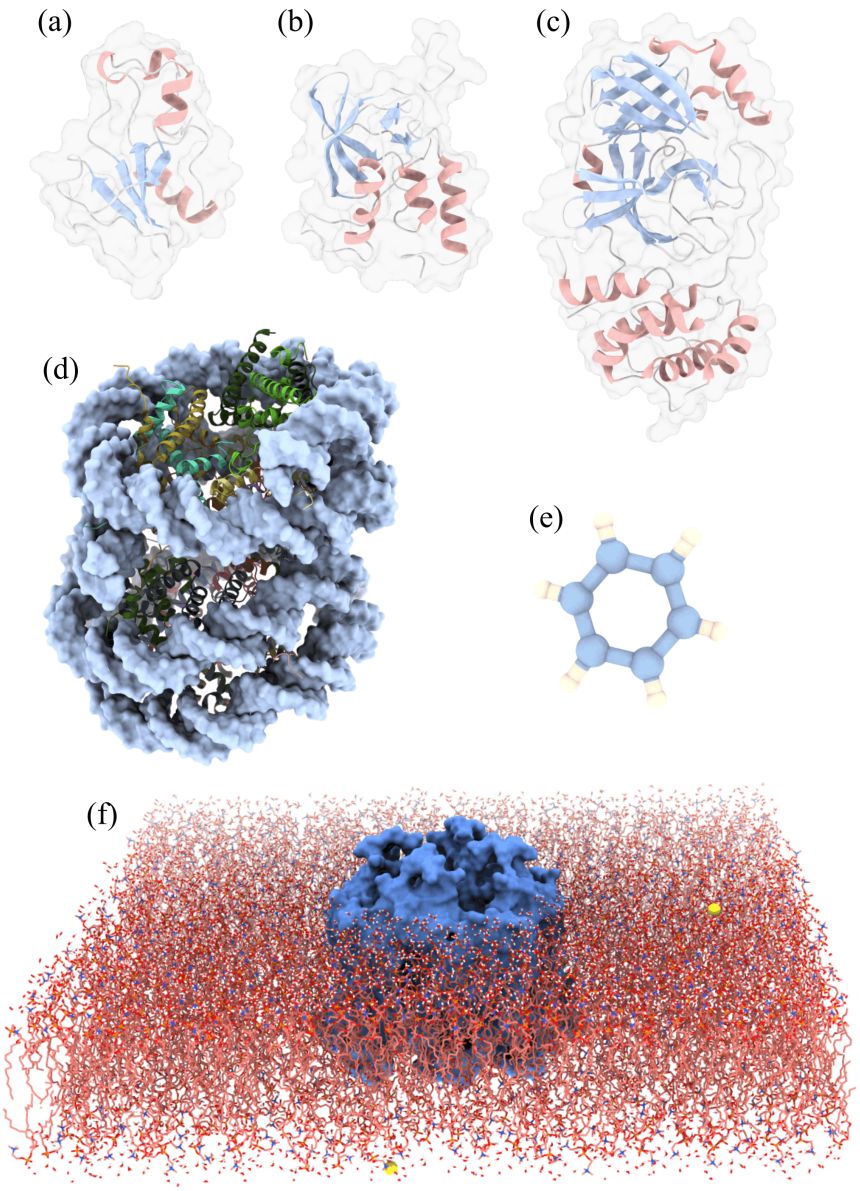



Figure 2.3.: Representations of the simulated molecules, covering solvated proteins: (a) barnase (PDB 1A2P), (b) nuclease (PDB 2SNS) and (c) Covid-19 main protease (PDB 5R7Y); (d) a solvated protein-DNA complex (PDB 3LEL); (e) an organic solvent (benzene); and (f) a tetrameric protein embedded in a lipid bilayer.

Table 2.3.: Relevant input parameters employed in the stages of the simulations.

| **Step** | **General Settings** | **Thermodynamic Ensemble (baths)** | **Physical Conditions** | **Time step / Simulated Time** |
|---|---|---|---|---|
| Min. | Integrator: steepest descent<br>Neighboring searching: grid<br>rcoul (PME): 1.0 $(0.9)^b$ nm<br>rvdw (cut-off): 1.0 $(0.9)^b$ nm<br>PBC: xyz<br>Constr: none | — | Emtol: 1 kJ/mol<br>Temp: 0 K<br>Press: 0 atm | max 20K steps |
| Heating | Integrator: md<br>Neighboring searching: grid<br>rcoul (PME): 1.0 $(0.9)^b$ nm<br>rvdw (cut-off): 1.0 $(0.9)^b$ nm<br>vdw-mod: potential-shift-verlet<br>PBC: xyz<br>Constr: all-bonds | Termost: Berendsen<br>$\tau_T = 0.1$ ps | Temp: Ini-T +<br>ramp (n $\times$ 50 K)<br>Press: 0 atm | 1 fs / n $\times$ 50 ps |
| Equil. 1 | Same as Heating | NVT<br>Termost: v-rescale<br>$\tau_T = 0.1$ ps | Temp: Final-T<br>Press: 0 atm | 1 fs / 100 ps |
| Equil. 2 | Same as Heating | NPT<br>Termost: v-rescale<br>$\tau_T = 0.1$ ps<br>Barost: Berendsen<br>$\tau_P = 2.0$ ps | Temp: Final-T<br>Press: 1 atm | 2 fs / 100 ps |
| Equil. 3 | Same as Heating | NPT<br>Termost: v-rescale<br>$\tau_T = 0.1$ ps<br>Barost: Parrinello-Rahman<br>$\tau_P = 2.0$ ps | Same as<br>Equil. 2 | 2 fs / 200 ps |
| Prod. | Same as Heating +<br>Constr: h-bonds,<br>all-bonds or h-angles | Same as Equil. 3 | Same as<br>Equil. 2 | 2 fs or 3.5 fs /<br>0.1 ns or 2 ns /<br>or 3 ns or 5.25 ns |

was solvated with the OPC4 water model [167]. The ionizable residues of proteins were protonated at pH 7 (GROMACS default), except for the case of barnase, which was set to pH 4.1 because the experimental thermodynamic observables used as a control were obtained under this condition. Chloride and/or sodium ions were added to neutralize the systems or to achieve the desired ionic strength. A cubic box was used for the membrane system, while a truncated dodecahedral box was used for the rest of the systems. The Verlet cutoff scheme [74], [175] was applied for van der Waals interactions, and the Particle Mesh Ewald (PME) method [176] was used to account for electrostatics. Both methods were used with a radius cutoff of 1.0 nm (0.9 nm for the membrane protein), while all the preparation steps imposed Periodic Boundary Conditions (PBC).

The minimization step was performed using the steepest descent algorithm [177], followed by an NVT gradual heating protocol, increasing temperature by 50 K every 50 ps with a 1 fs time step, until the target temperature was reached. The equilibration process began with a 100 ps NVT simulation with a 2 fs time step using the V-rescale thermostat [140] ($\tau_T = 0.1$ ps). During this step, position constraints were applied to all the heavy atoms (protein, DNA, lipids, and water). This was followed by a 100 ps simulation in the NPT ensemble using a 2 fs time step and the heavy atoms now unrestrained. V-rescale thermostat ($\tau_T = 0.1$ ps) and the Berendsen barostat [178] (p = 1 atm, $\tau_P = 2.0$ ps) were settled. Finally, the system underwent 200-ps NPT simulation with a 2 fs time step, employing the V-rescale thermostat ($\tau_T = 0.1$ ps) and the Parrinello-Rahman barostat [179] (p = 1 atm, $\tau_P = 2.0$ ps). Temperature corrections due to the thermostat were applied every 10 steps (GROMACS' `nsttcouple` default value). Input files from our simulations can be retrieved from https://github.com/LorienLV/_PAPER_ILVES.

The final equilibrated configuration served as the starting point for the production phase of the simulations used in our reliability and performance studies. Unless stated otherwise, the production simulations were carried out in the NPT ensemble. The physics results also include simulations performed in the NVE ensemble.

## 2.7. Evaluation

### 2.7.1. Reliability Study

We replicate the methodology presented in [180], [181], which has been shown to accurately reproduce protein folding thermodynamics using both SHAKE and P-LINCS, to assess the reliability of the ILVES algorithms. This evaluation is based on the calculation of three thermodynamic observables: namely, the change in enthalpy upon unfolding $\Delta H_{unf}$ (or upon folding, with opposite sign); the change in heat capacity upon the same event, $\Delta Cp_{unf}$; and the corresponding change in Gibbs free energy (i.e., the protein conformational stability), $\Delta G_{unf}$. The analysis was conducted by simulating two well-studied proteins: barnase and nuclease (see Table 2.2), which have been similarly modeled in [180], [181]. Each simulation consists of 1 million steps with a time step of 2 fs, corresponding to a total simulated time of 2 ns. The main parameters used in the preparation and production runs are summarized in Table 2.3.

Table 2.4 displays the results of our reliability study. ILVES-PC is not included in this

Table 2.4.: Experimentally measured and calculated (MD) thermodynamic quantities of unfolding ($\Delta H_{unf}$ and $\Delta Cp_{unf}$) for barnase ($T_m = 323.0 \pm 0.7$ at pH ~4.1 [181]) and nuclease ($T_m = 326.9 \pm 0.4$ at pH ~7.0 [181]). The listed Gibbs free energies ($\Delta G_{unf}$) are calculated through the Gibbs-Helmholtz equation using either the measured or MD thermodynamics. P-LINCS is configured with GROMACS default parameters, while MP-LINCS uses `lincs-order=4`. All of the simulations are performed with `constraints=h-bonds`.

| **Protein (Method)** | **Solver** | **Tol.** | $\mathbf{\Delta H_{unf}}$ **at** $\mathbf{T_m}$ **(kcal/mol)** | $\mathbf{\Delta Cp_{unf}}$ **(kcal/mol · K)** | $\mathbf{\Delta G_{unf}}$ **at 298 K (kcal/mol)** |
|---|---|---|---|---|---|
| Barnase (Exp.) | — | — | $118.7 \pm 4.9$ | $1.4 \pm 0.1$ | $7.8 \pm 1.6$ |
| Barnase (MD) | SHAKE | $10^{-4}$ | $113.9 \pm 3.1$ | $1.1 \pm 0.1$ | $7.8 \pm 1.1$ |
| | SHAKE | $10^{-10}$ | $115.2 \pm 3.1$ | $1.3 \pm 0.1$ | $7.6 \pm 1.4$ |
| | MP-LINCS | $10^{-4}$ | $116.9 \pm 3.1$ | $1.2 \pm 0.0$ | $7.9 \pm 1.2$ |
| | MP-LINCS | $10^{-10}$ | $118.6 \pm 3.1$ | $1.3 \pm 0.1$ | $7.9 \pm 1.3$ |
| | P-LINCS | — | $116.3 \pm 3.1$ | $1.2 \pm 0.0$ | $7.9 \pm 1.2$ |
| | ILVES-M | $10^{-4}$ | $113.6 \pm 3.1$ | $1.0 \pm 0.1$ | $7.8 \pm 1.2$ |
| | ILVES-M | $10^{-10}$ | $117.4 \pm 3.1$ | $1.1 \pm 0.2$ | $8.0 \pm 1.3$ |
| | ILVES-F | $10^{-4}$ | $116.4 \pm 3.1$ | $1.2 \pm 0.0$ | $7.8 \pm 1.2$ |
| | ILVES-F | $10^{-10}$ | $115.6 \pm 3.1$ | $1.0 \pm 0.1$ | $7.9 \pm 1.2$ |
| Nuclease (Exp.) | — | — | $82.1 \pm 4.7$ | $2.3 \pm 0.1$ | $4.2 \pm 1.6$ |
| Nuclease (MD) | SHAKE | $10^{-4}$ | $78.1 \pm 4.8$ | $2.0 \pm 0.1$ | $4.2 \pm 1.4$ |
| | SHAKE | $10^{-10}$ | $73.5 \pm 4.8$ | $1.9 \pm 0.1$ | $4.1 \pm 1.3$ |
| | MP-LINCS | $10^{-4}$ | $74.6 \pm 4.8$ | $1.8 \pm 0.1$ | $4.2 \pm 1.3$ |
| | MP-LINCS | $10^{-10}$ | $71.6 \pm 4.8$ | $1.7 \pm 0.2$ | $4.1 \pm 1.4$ |
| | P-LINCS | — | $70.8 \pm 4.8$ | $1.6 \pm 0.0$ | $4.2 \pm 1.1$ |
| | ILVES-M | $10^{-4}$ | $75.8 \pm 4.8$ | $2.0 \pm 0.1$ | $4.0 \pm 1.4$ |
| | ILVES-M | $10^{-10}$ | $74.6 \pm 4.8$ | $2.0 \pm 0.1$ | $4.0 \pm 1.4$ |
| | ILVES-F | $10^{-4}$ | $74.4 \pm 4.8$ | $2.0 \pm 0.0$ | $4.0 \pm 1.3$ |
| | ILVES-F | $10^{-10}$ | $73.1 \pm 4.8$ | $1.8 \pm 0.1$ | $4.1 \pm 1.3$ |

analysis; however, since it solves the same equations as ILVES-M using the same numerical method, it would yield identical results. The methodology of the study involves separately simulating—in the NPT ensemble—the folded and unfolded states of a solvated protein (see Figure 2.4) under identical conditions (force field, type and number of water molecules, ions, pH, temperature, pressure, time step, simulation length, etc.). To achieve precise averaging of thermodynamic quantities, the method requires representative sampling of the conformational space of both states. We used 40 simulation replicas for the folded state and 100 replicas for the unfolded state. The 100 protein structures representative of the unfolded state are a filtered sample of a larger set of completely unfolded conformations generated by the ProtSA server [182]. The average enthalpies obtained for the two states are subtracted to yield the enthalpy change upon unfolding, $\Delta H_{unf}$. From this calculation, the error is obtained as the sum of both errors: that corresponding to the folded ensemble and that of the unfolded ensemble. The calculation of $\Delta H_{unf}$ is performed at three temperatures (295 K, 315 K, and 335 K), allowing for the determination of the enthalpy change with temperature, which is typically linear. This linear relationship, whose slope

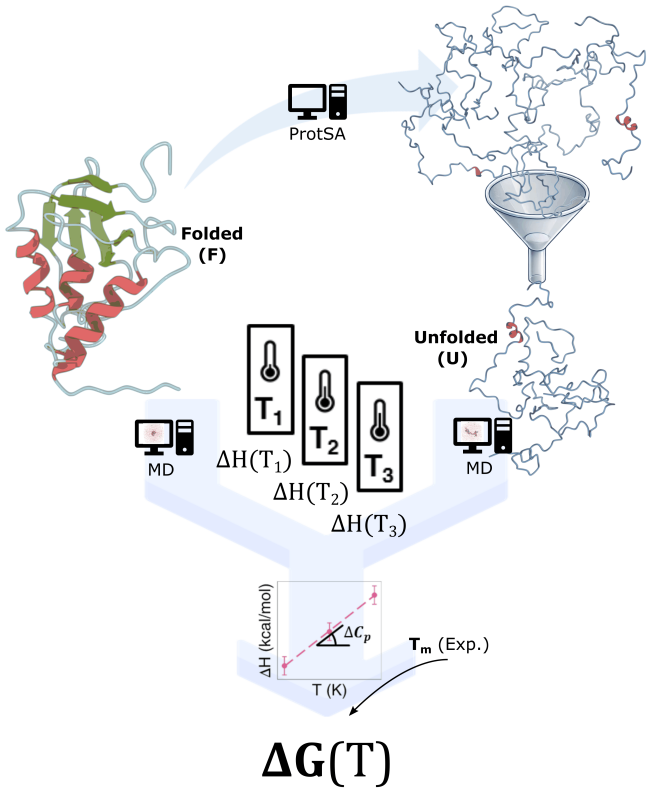



Figure 2.4.: Workflow of the calculation of observable quantities.

corresponds to the change in heat capacity, $\Delta \mathrm{Cp}_{unf}$, allows interpolating the enthalpy change at the mid-denaturation temperature ($\mathrm{T}_m$, taken from the literature), which is the enthalpy change that is commonly reported in scientific articles. The reported error in $\Delta \mathrm{Cp}_{unf}$ calculation is that obtained from the linear fitting. Once these two thermodynamic quantities are calculated ($\Delta \mathrm{H}_{unf}$ at the $\mathrm{T}_m$ and $\Delta \mathrm{Cp}_{unf}$), the Gibbs free energy change can be obtained through the Gibbs-Helmholtz equation [183]:

$$\Delta G(T) = \Delta H_m \left(1 - \frac{T}{T_m}\right) - \Delta C_p \left[T_m - T + T \ln\left(\frac{T}{T_m}\right)\right]. \tag{2.36}$$

The reported error in $\Delta \mathrm{G}(T)$ calculations (Table 2.4) is obtained by error propagation. In this assessment, we used the combination of force field and water model that has been shown to best reproduce experimental protein thermodynamics [180], [181]: Charmm22+CMAP [158] and Tip3p [159]. Note that the table does not include ILVES-PC; however, since ILVES-PC solves the same equations as ILVES-M using the same method, the results are effectively identical.

The results displayed in Table 2.4 indicate that for ILVES and state-of-the-art solvers, the evaluated thermodynamics $\Delta \mathrm{H}_{unf}(T_m)$ and $\Delta \mathrm{G}_{unf}(298\mathrm{K})$ fall within the error ranges defined by experimental measurements, for both proteins. In the case of $\Delta \mathrm{Cp}_{unf}$, the

MD-calculated values fall outside the experimental ranges in some of the setups assessed, but the differences are similar for all constraint solvers. This outcome supports the thesis that ILVES algorithms preserve system dynamics and do not introduce distortions larger than those due to traditionally used constraint solvers.

### 2.7.2. Physics Analysis

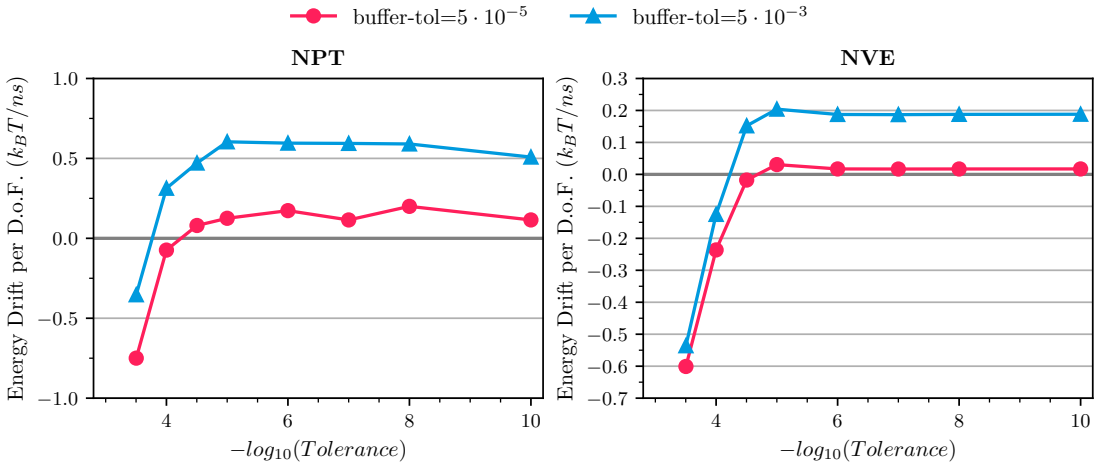
One of our goals is to understand the connection between energy drift in molecular dynamics (MD) simulations and the tolerance of the constraint solver. For this purpose, we simulated the barnase system in both the NPT and NVE ensembles using SHAKE as the constraint algorithm. We chose SHAKE over P-LINCS or ILVES because its relatively slow and steady convergence ensures that the largest relative error in bond lengths is essentially equal to the specified tolerance of the constraint algorithm.

All simulations start from the configurations previously described (see Table 2.3), with pressure set to 1 atm and temperature set to 298 K. In the NVE ensemble, this temperature is used only as an initial condition, and no thermostat is applied during production. Each run consists of 500K steps with a time step of 2 fs, corresponding to a total simulated time of 1 ns. For every configuration, three replicas are launched, and the reported results correspond to the average over replicas.

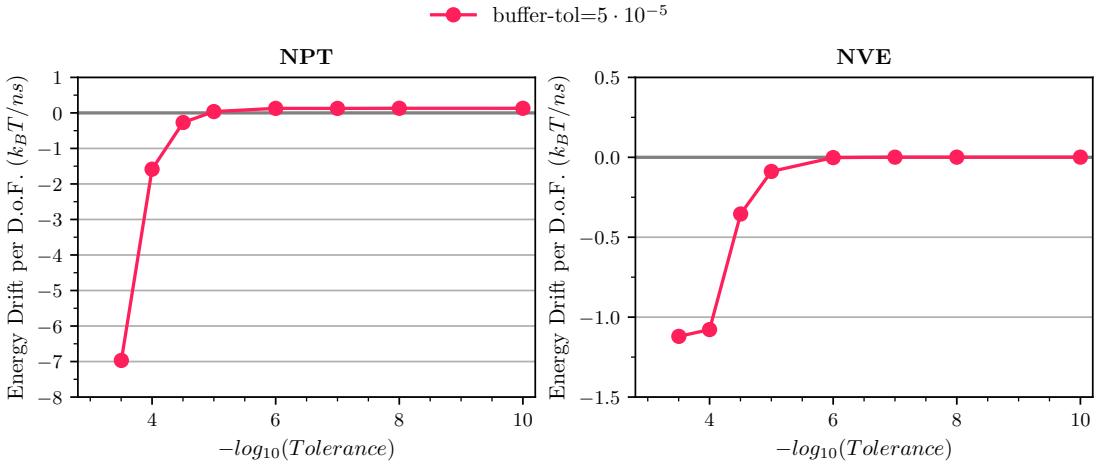
We also take into account the Verlet buffer size used in the pair-list neighbor search, which is one of the two main sources of energy drift in MD simulations, the other being the constraint algorithm itself. In GROMACS, the Verlet buffer size is controlled through the parameter `verlet-buffer-tolerance`, whose default value is $5 \cdot 10^{-3}$ kJ/(mol · ps). However, tightening this value to $5 \cdot 10^{-5}$ kJ/(mol · ps) allows the drift associated with the accuracy of the constraint solver to be observed more clearly. We study eight values of the constraint tolerance (Tol), from less to more accurate: $3.14 \cdot 10^{-4}$, $10^{-4}$, $3.14 \cdot 10^{-5}$, $10^{-5}$, $10^{-6}$, $10^{-7}$, $10^{-8}$, and $10^{-10}$.

Figure 2.5 presents the energy drift, computed as the slope of the energy–time relationship [71], divided by the number of degrees of freedom of the system, $N_{df}$. We define the latter as $N_{df} = 3s - n - 6$, where $s$ is the number of atoms in the protein, $n$ is the number of imposed constraints, and the $-6$ term accounts for rotations and translations. To convert from J/mol to units of $k_BT$, we divide by 2477.7, corresponding to a temperature of 298 K. This conversion is also applied to the NVE simulations, where no temperature is imposed during production but 298 K is used as the initial condition. When all-bonds are constrained, the Verlet buffer tolerance is set to $5 \cdot 10^{-5}$ kJ/(mol · ps), whereas for H-bond constraints we additionally report results using the GROMACS default value of $5 \cdot 10^{-3}$ kJ/(mol · ps).

For both H-bond and all-bond constraints, the energy drift decreases rapidly as the tolerance is reduced (from left to right on the x-axis). This behavior is consistent with the literature, where Tol $= 10^{-7}$ is commonly used to reduce energy drift to acceptable levels [130]–[132]. Our results suggest that the constraint tolerance Tol should be at least as small as $\tau = 10^{-6}$ to avoid significant energy drift. It can also be observed that, when only H-bonds are constrained, the default Verlet buffer tolerance of $5 \cdot 10^{-3}$ kJ/(mol · ps) introduces additional drift even at very strict constraint tolerances. For instance, at

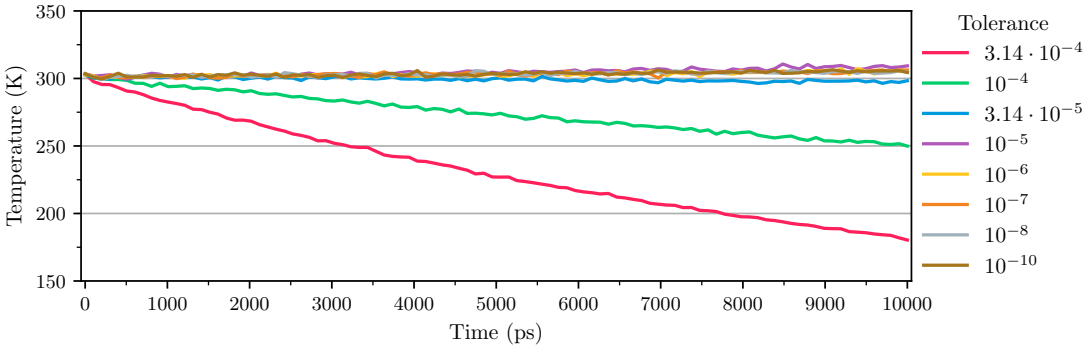



(a) H-bonds

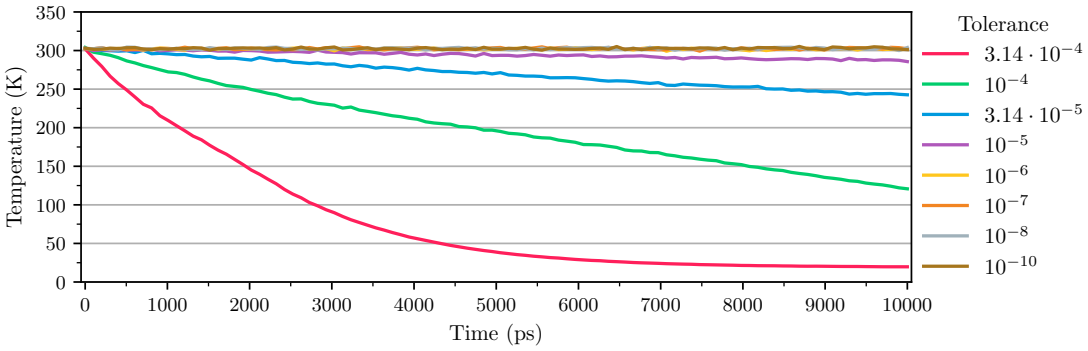



(b) all-bonds

Figure 2.5.: Drifts of the conserved energy of the thermostat (NPT ensemble, left) and of the energy (NVE ensemble, right) per degree of freedom as a function of the tolerance in satisfying the constraints in the barnase system. The top panels correspond to simulations in which H-bonds are constrained, whereas the bottom panels correspond to simulations with all-bonds constrained. The parameter `buffer-tol` corresponds to the GROMACS `verlet-buffer-tolerance` parameter.




(a) H-bonds




(b) all-bonds

Figure 2.6.: Temperature evolution over time of the barnase system in the NVE ensemble using different constraint tolerances. The top panels correspond to simulations in which only H-bonds are constrained, whereas the bottom panels correspond to simulations in which all bonds are constrained.

Tol $= 10^{-10}$ the drift is approximately 0.2 $k_B T$/ns with the default buffer, whereas it is almost zero when the tighter value of $5 \cdot 10^{-5}$ kJ/(mol $\cdot$ ps) is used.

Figure 2.6 shows the temperature evolution over time for the barnase simulations in the NVE ensemble for H-bond and all-bond constraints. In these simulations, we use the demanding Verlet buffer tolerance of $5 \cdot 10^{-5}$ kJ/(mol $\cdot$ ps). In both cases, higher constraint tolerances lead to a significant drift from the initial temperature. Conversely, when $\tau \leq 10^{-6}$, the temperature is approximately preserved over the analyzed time range. Similar observations have been reported by other authors, who show that low accuracy in constraint solving leads to inaccurate temperatures and that the default LINCS parameters may result in non-converged simulations [70].

### 2.7.3. Performance Analysis

In this subsection, we compare the performance of ILVES-PC, ILVES-M, and ILVES-F against state-of-the-art constraint solvers. The systems used in this comparison are: barnase (BARN), the COVID-19 main protease (COVID), a protein–DNA complex

(DNAP), benzene (BENZ), and a lipid bilayer system (LIPID). Note that ILVES-PC is limited to the BARN and COVID systems when all-bonds are constrained, as these are the only two systems consisting solely of proteins.

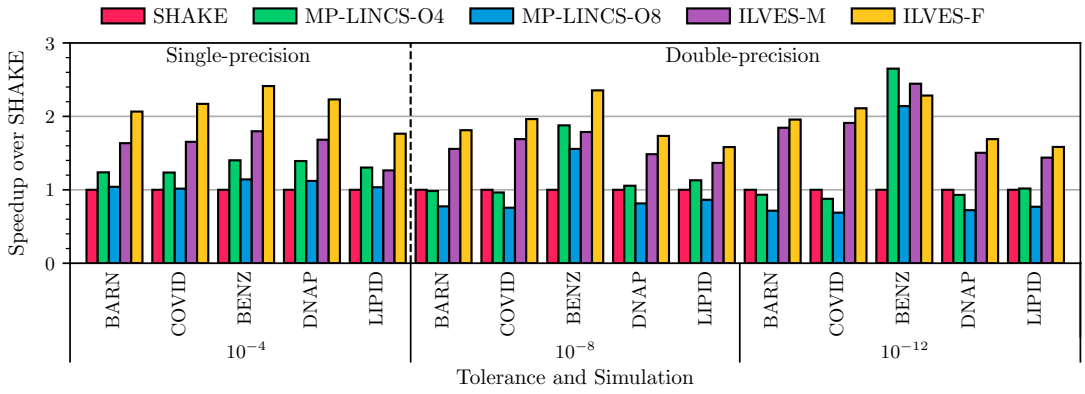
For the BARN, COVID, BENZ, and DNAP systems, simulations were run on a single Intel Xeon Platinum 8480+ chip using up to 56 threads and 1 task. The larger LIPID system, which contains 390k atoms including 149k explicit water molecules, was executed across up to 8 tasks with 56 threads per task, corresponding to a maximum of 4 nodes and 8 chips.

The time spent initializing the solvers can be high in distributed-memory simulations. For this reason, the execution times in our analysis include both initialization and processing for all solvers—except ILVES-PC, which was released as a proof-of-concept without distributed memory support or optimized initialization.

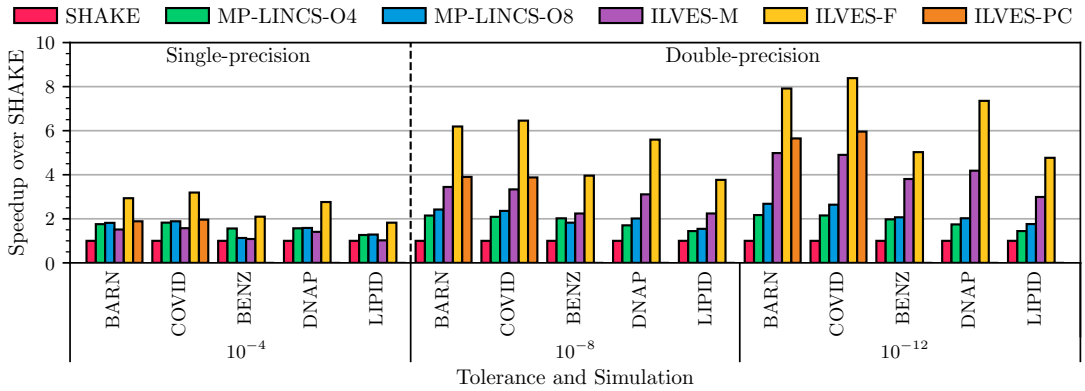
The speed of constraint solving is closely related to the minimum required accuracy. We thus considered three values of the referred tolerance (maximum allowed relative error for every constraint): Tol $= 10^{-4}$ (which is the default in GROMACS), Tol $= 10^{-8}$ and Tol $= 10^{-12}$. Since the numerical precision used in GROMACS also affects performance, simulations for Tol $= 10^{-4}$ are performed using GROMACS compiled in single-precision mode (FP32), whereas simulations with stricter tolerances are conducted with GROMACS compiled in double-precision mode (FP64).

We measured the performance of the algorithms on systems with constraints applied to H-bonds, all-bonds, or all-bonds combined with specific hydrogen-atom angles (specifically, H–X–H and X–O–H angles, where X represents a generic atom). For H-bond and all-bond simulations, which are typical in current molecular dynamics practice, we performed 50,000 steps. For simulations including angular constraints, we ran 1.5 million steps.

**Single-Threaded.** We begin by evaluating the single-threaded performance of the constraint algorithms for H-bonds and all-bonds constraints. Figure 2.7 displays the single-threaded speedup over SHAKE of ILVES-PC, ILVES-M, ILVES-F, and MP-LINCS (`lincs-order=4` and `lincs-order=8`). This metric is defined as the ratio of the single-threaded execution time of SHAKE to the single-threaded execution time of the solver. While ILVES-PC is tailored to protein simulations, it delivers only modest speedups over ILVES-M, which incorporates optimizations absent in ILVES-PC, the most significant being the vectorization of linear-system construction. For H-bonds constraints, MP-LINCS performs similarly to SHAKE in most simulations, while ILVES-M and ILVES-F show consistent speedups over both SHAKE and MP-LINCS across all tolerances, achieving maximum speedups of 2.4× and 3× over SHAKE and MP-LINCS, respectively. When constraining all-bonds and using Tol $= 10^{-4}$, ILVES-PC and ILVES-M achieve speedups between 1× and 1.8× compared to SHAKE, similar to MP-LINCS. For the same tolerance, ILVES-F outperforms both, delivering speedups between 1.9× and 3.2× over SHAKE and between 1.5× and 1.8× over MP-LINCS. As the tolerance becomes more demanding, all ILVES versions show greater speedups, with a maximum of 8.4× over SHAKE and 4.2× over MP-LINCS. The only case in Figure 2.7 where MP-LINCS outperforms ILVES-F (the consistently fastest ILVES variant) is the BENZ simulation at the tightest tolerance.




(a) H-bonds



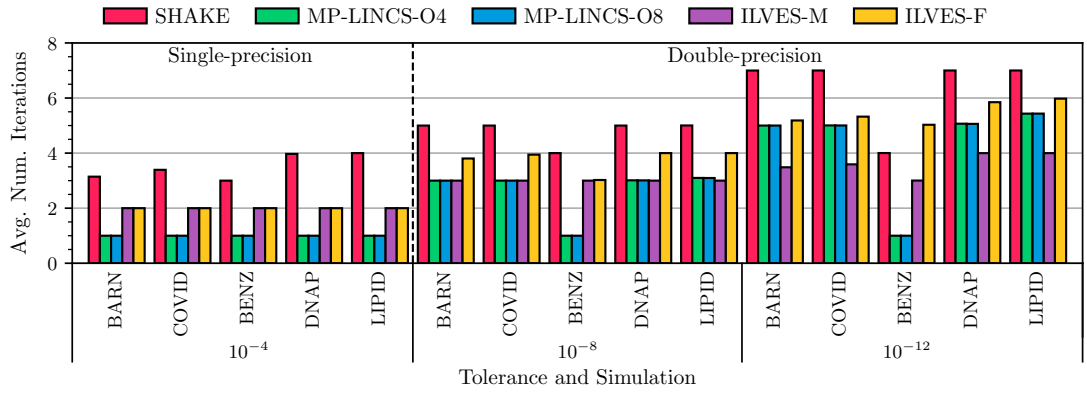

(b) all-bonds

Figure 2.7.: Single-threaded speedup over SHAKE of MP-LINCS (`lincs-order=4` and `lincs-order=8`), ILVES-PC, ILVES-M and ILVES-F.

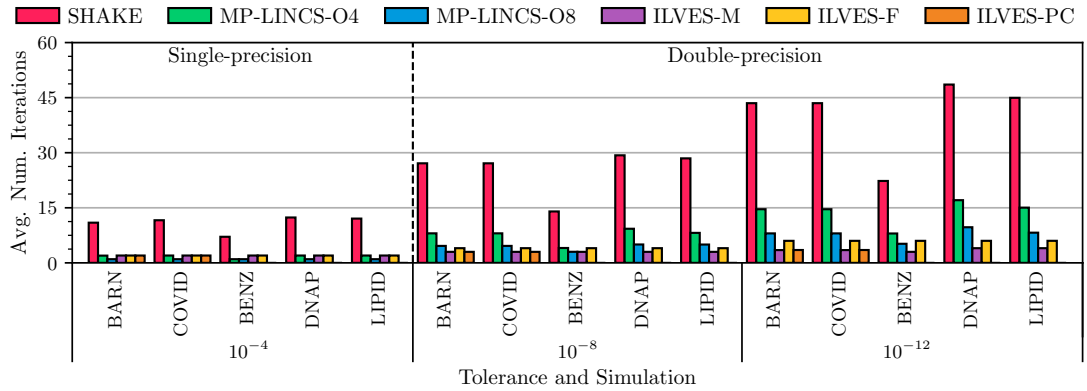
In it, each benzene molecule has only six constraints, and MP-LINCS achieves low errors in its projection step, sometimes avoiding any iteration of its correction step.

Note that the benzene molecule with constraints on just H-bonds is a very exceptional case where every constraint is fully decoupled from the rest. This makes the matrix that must be inverted (in the involved linear systems of equations) diagonal because there exists no atom that belongs to two different constraints, which makes the linear system suitable for a faster analytical solution. Relevant research works recommend constraining both bond lengths and bond angles in benzene [94].

**Iterations to Convergence.** We complement the single-threaded performance results by studying the number of iterations required by each solver to reach the target tolerance. Figure 2.8 shows the average number of iterations executed per simulation step by each solver. Overall, all solvers require substantially fewer iterations for H-bonds than for all-bonds. In the H-bond case, the molecule is composed of small clusters of atoms connected by constrained bonds, with each cluster disconnected from the others. This




(a) H-bonds




(b) all-bonds

Figure 2.8.: Average number of iterations per simulation step required for convergence by SHAKE, MP-LINCS (correction phase only), ILVES-PC, ILVES-M, and ILVES-F.

makes the constraints easier to satisfy, because the displacement of an atom affects only the atoms within the same cluster. In contrast, when all bonds are constrained, position updates affect all other atoms, increasing the coupling between constraints and making the nonlinear system more difficult to solve.

SHAKE requires the most iterations in all cases, reaching up to 49 for the most demanding configuration. ILVES-M and ILVES-PC require the same number of iterations to converge since they solve the same system of equations. In general, ILVES-F requires more iterations than ILVES-M and ILVES-PC, although each iteration is computationally cheaper. The `lincs-order` parameter affects MP-LINCS only when all bonds are constrained, with MP-LINCS-O8 requiring substantially fewer correction iterations than MP-LINCS-O4.

For moderate tolerances, MP-LINCS generally requires a similar or smaller number of iterations than the ILVES solvers. However, as the target tolerance becomes more demanding, the ILVES solvers require significantly fewer iterations to converge. Note

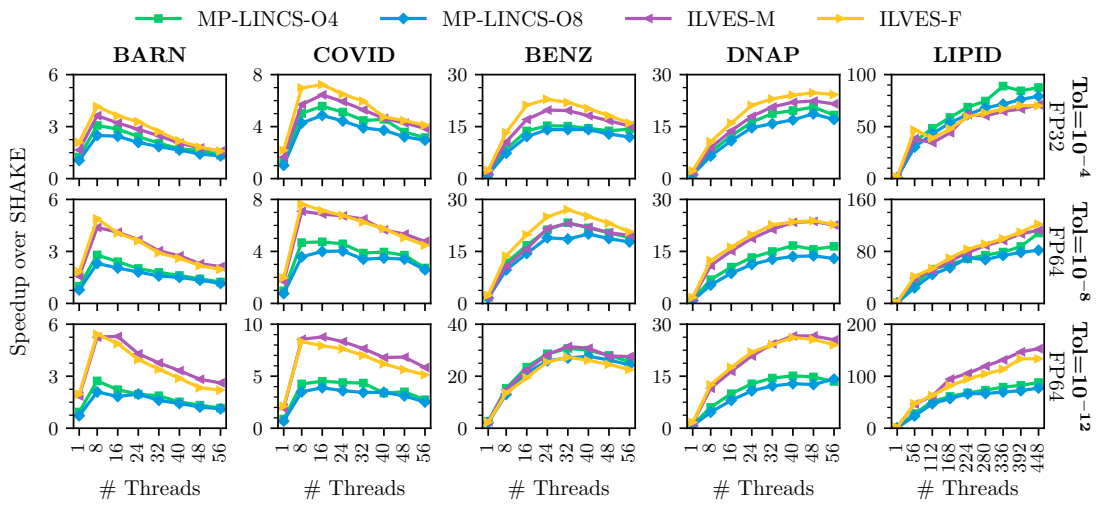
that the reported MP-LINCS iteration counts account only for the correction phase and do not include the initial projection phase.

**Multi-Threaded Speedup over SHAKE.** Figure 2.9 presents the speedup over SHAKE for different numbers of threads and tasks when constraining H-bonds and all-bonds. This metric is defined as the ratio of SHAKE's single-threaded execution time to the execution time of the given solver on $N$ threads. In all-bonds simulations (Figure 2.9-b), ILVES-PC (when available), ILVES-M, and ILVES-F achieve speedups over MP-LINCS across all simulations and tolerances, with a maximum of 158× over SHAKE and 14× over MP-LINCS. In H-bonds simulations (Figure 2.9-a), MP-LINCS delivers better parallel performance than in all-bonds simulations, thus narrowing the performance gap between the solvers. In these simulations, MP-LINCS only surpasses ILVES-M and ILVES-F in the LIPID simulation at Tol = $10^{-4}$. On the other hand, ILVES-F delivers better performance than MP-LINCS in the rest of the simulations for a maximum speedup of 134× over SHAKE and speedups over MP-LINCS up to 1.8×.

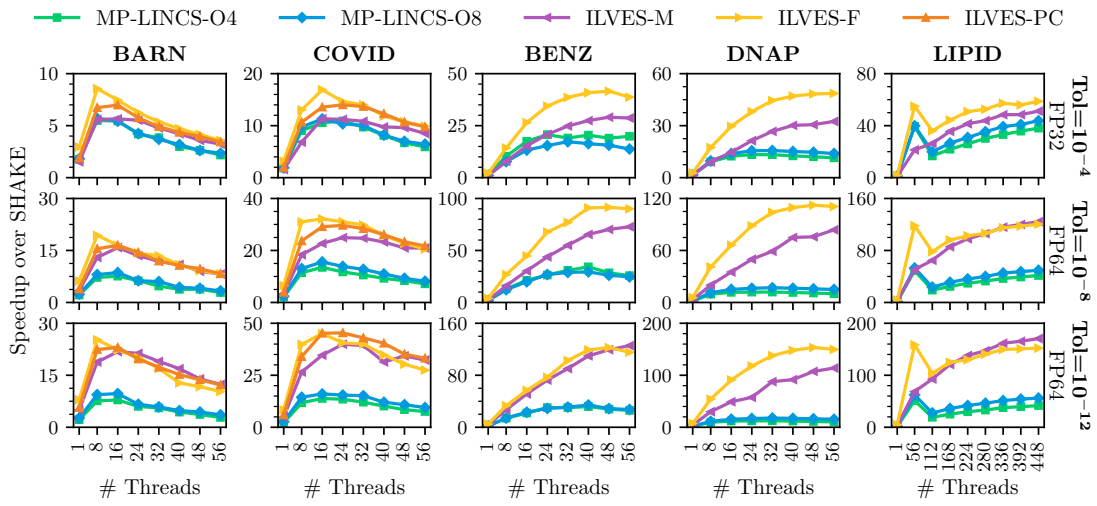
**Multi-Threaded Scalability.** Figure 2.10 shows the parallel scalability of the constraint algorithms. The scalability of a solver is defined as the ratio of the execution time of the solver using $N$ threads to the execution time of the solver using 1 thread. The parallel scalability of the ILVES algorithms is significantly better in the BENZ and DNAP simulations with `constraints=all-bonds`. In these simulations, the graph partitioning produced by the ILVES algorithms yields substantially less serial work than that produced by MP-LINCS. In the case of BENZ, the partitioning eliminates serial work entirely. The strong scalability of the BENZ system may seem counter-intuitive given that benzene molecules are small. However, the system consists of 2,000 benzene molecules, leading to a substantial parallel workload. While in other simulations the scalability of the ILVES algorithms and MP-LINCS is more similar, it is worth noting that ILVES delivers better performance than MP-LINCS in most cases. Consequently, for shorter execution times, the overhead of thread synchronization in ILVES becomes more pronounced, ultimately affecting its parallel scalability.

In most parallel simulations, peak performance is not achieved using the maximum number of threads available. As more threads are added, synchronization overhead increases, and if zero-cut partitions are not possible, the amount of serial code executed also rises. In distributed-memory simulations, this overhead is further worsened, as synchronization between threads is needed alongside communication between tasks. The performance toll of inter-task synchronization is particularly evident in the lipid bilayer simulation with `constraints=all-bonds`, where a performance drop occurs beyond 56 threads corresponding to the transition from a single task to multiple tasks. In contrast, this drop is absent with `constraints=h-bonds` due to the lack of bonds between atoms in different tasks, thereby eliminating the need for inter-task communication.

**Percentage of Time in Constraints.** Solving constraints is sometimes assumed to require a relatively small fraction of the total execution time of simulations. However, some

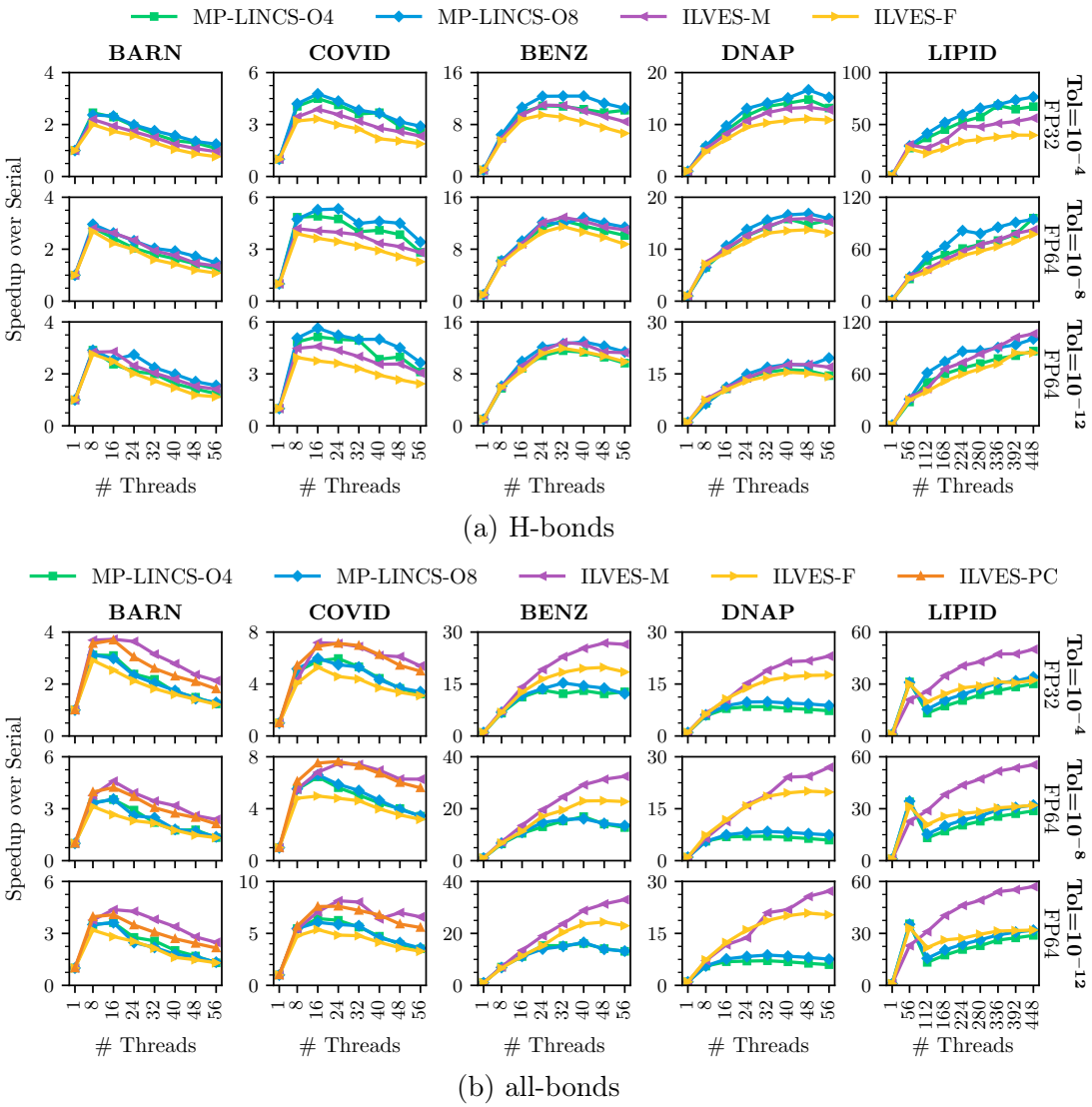



(a) H-bonds




(b) all-bonds

Figure 2.9.: Multi-threaded speedup over SHAKE of MP-LINCS (`lincs-order=4` and `lincs-order=8`), ILVES-M, ILVES-F, and ILVES-PC. The BARN, COVID, BENZ, and DNAP simulations are executed using a single task on a single chip of a computing node. The LIPID simulation is executed using up to 8 tasks (up to 4 nodes and 8 chips). (a) Constraints imposed on H-bonds; (b) Constraints imposed on all-bonds.




(a) H-bonds

(b) all-bonds

Figure 2.10.: Multi-threaded speedup over serial execution of MP-LINCS (`lincs-order=4` and `lincs-order=8`), ILVES-PC, ILVES-M, and ILVES-F. The BARN, COVID, BENZ, and DNAP simulations are executed using a single task on a single chip of a computing node. The LIPID simulation is executed using up to 8 tasks (up to 4 nodes and 8 chips). (a) Constraints imposed on H-bonds; (b) On all-bonds.

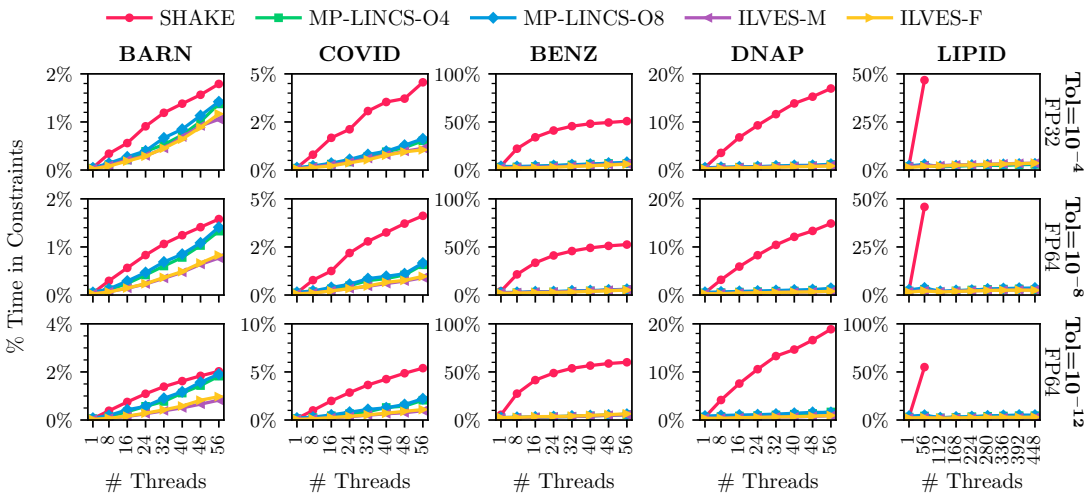
authors report that it can be as high as 50% or 60% [95], [184]. Figure 2.11 displays the percentage of the total simulation execution time spent in the constraint solver (excluding water) in the evaluated systems. With `constraints=h-bonds`, SHAKE accounts for up to 60%, MP-LINCS up to 6%, and ILVES up to 5%. With `constraints=all-bonds`, SHAKE accounts for up to 92% of the total execution time, MP-LINCS up to 42%, and ILVES up to 16%.

**Simulation Throughput.** Figure 2.12 reports the global simulation throughput, measured in nanoseconds simulated per day, for all the constraint solvers under study. These results complement Figure 2.11 to illustrate how the performance of the constraint solver impacts the overall simulation performance.

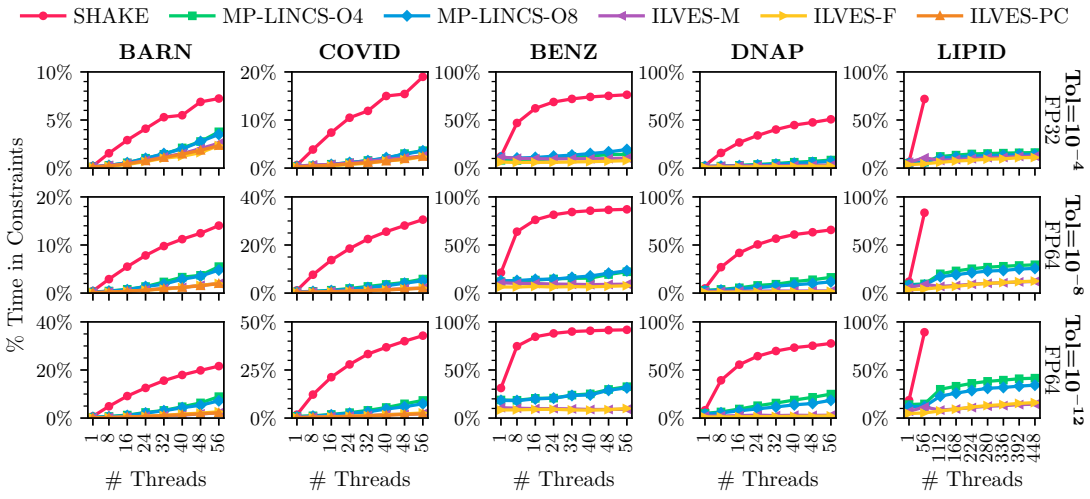
As previously shown, the time spent on H-bond constraints by ILVES and MP-LINCS is consistently low across all tested simulations and configurations, whereas SHAKE accounts for a significant fraction of the total execution time in several experiments. As a result, ILVES and MP-LINCS have a negligible impact on simulation throughput, while SHAKE incurs a substantial penalty.

For simulations with constraints on all bonds, both SHAKE and MP-LINCS can consume a significant share of the total execution time, and therefore both algorithms negatively impact the simulation throughput. This effect is clear in the BENZ and LIPID simulations shown in Figure 2.12-b, where ILVES delivers throughput improvements of up to 1.45× over MP-LINCS, with the largest gains achieved at the most demanding tolerances.

**Error vs. Time.** Although tolerance defines the maximum acceptable error, the rapid convergence of the ILVES algorithms often yields errors significantly below this threshold. This results in accuracy gains, providing a compelling reason to choose ILVES over MP-LINCS in simulations where their performance is similar. In Figure 2.13 we display the single-threaded execution time required for imposing the constraints in the COVID system as a function of the average relative error, which is defined as the average for $N_s$ steps and $n$ constraints that follows: $\left(\frac{1}{2\,n\cdot N_s}\right)\cdot\left(\sum_{k=1}^{N_s}\sum_{i=1}^{n}|d_i^2-(\boldsymbol{q}_{a_i}(t_k)-\boldsymbol{q}_{b_i}(t_k))^2|/d_i^2\right)$, where $\boldsymbol{q}_{a_i}(t_k)$, $\boldsymbol{q}_{b_i}(t_k)$ are the positions of both atoms joined by the $i$th constraint after applying the constraint forces corresponding to the $k$th step, and $d_i$ are the bond length constants. Every point of Figure 2.13 corresponds to a simulation performed with different parameters (values of the constraint tolerance for SHAKE; values of the constraint tolerance or number of iterations for ILVES-M and ILVES-F; values of the number of iterations —`lincs-iter`— and truncation of the Neumann series —`lincs-order`— for P-LINCS; values of the constraint tolerance and `lincs-order` for MP-LINCS). Since ILVES-PC and ILVES-M solve the same system of equations, their convergence rates are identical, and therefore ILVES-PC is not shown in the plot. If we compare algorithms that ensure that a minimum accuracy is satisfied, like MP-LINCS-O4 and ILVES-F, we observe that, for approximately the same execution time, the latter is far more accurate than the former. For example, for constraints on all-bonds, Figure 2.13 displays a point for MP-LINCS-O4 whose execution time is 32 s and whose average relative error is $5\cdot10^{-6}$;

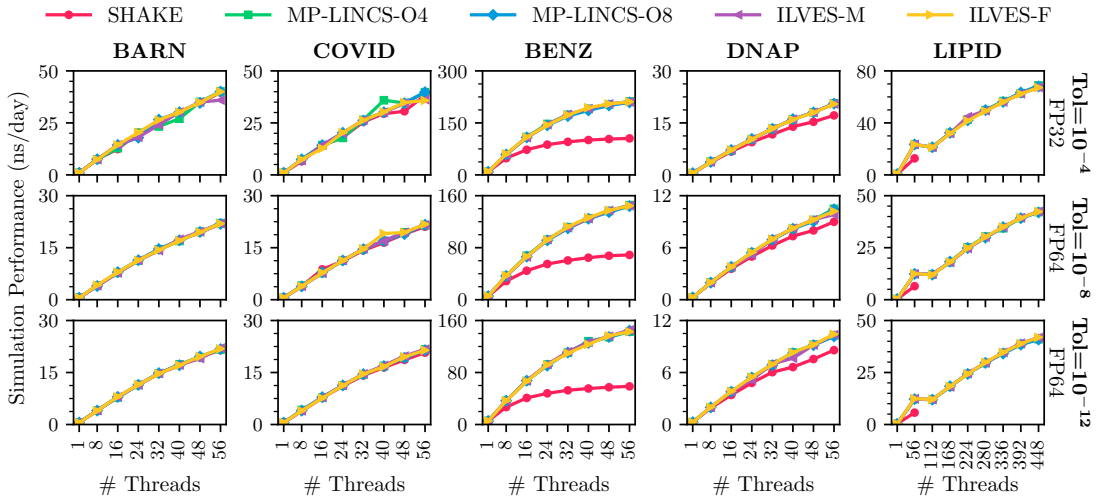



(a) H-bonds

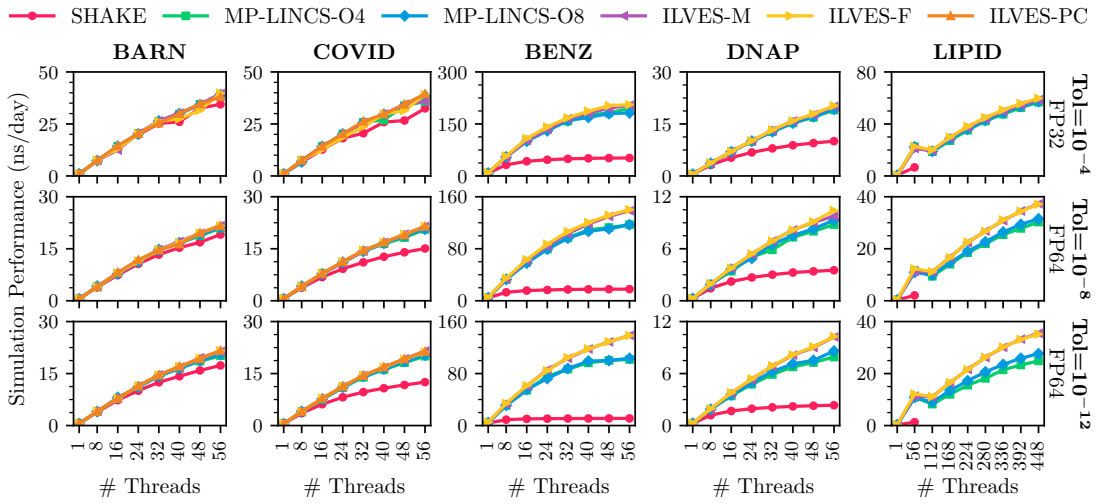



(b) all-bonds

Figure 2.11.: Percentage of the total execution time of GROMACS spent in the constraint solver (ex water). (a) Constraints imposed on H-bonds; (b) On all-bonds.

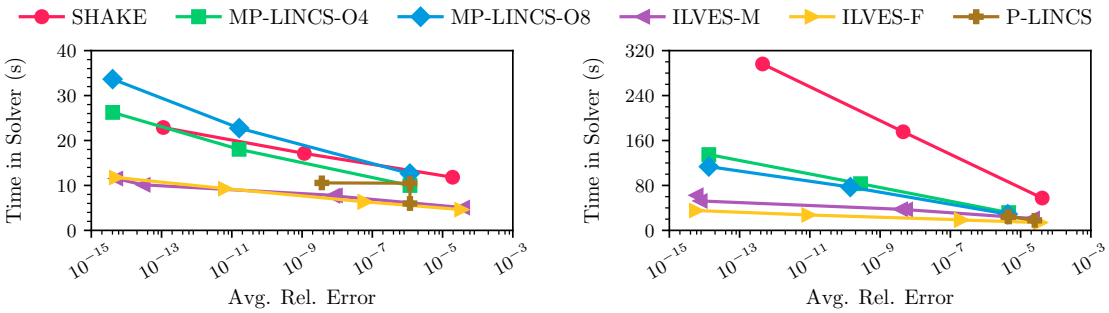



(a) H-bonds




(b) all-bonds

Figure 2.12.: Overall Gromacs simulation throughput, measured in nanoseconds simulated per day, obtained with different constraint solvers.




(a) H-bonds (b) all-bonds

Figure 2.13.: Execution time of the block for solving constraints as a function of the average relative error in satisfying them for different constraint solvers. (a) Constraints imposed on H-bonds; (b) Constraints imposed on all-bonds. Note that the y-axis is logarithmic and that the tolerance decreases from left to right.

it also displays a point for ILVES-F whose execution time is 27 s and whose average relative error is $9 \cdot 10^{-12}$. This feature also holds for constraints on hydrogen bonds: examples of points displayed in Figure 2.13 are (10 s, $10^{-6}$) for MP-LINCS-O4, (9 s, $7 \cdot 10^{-12}$) for ILVES-F and (10 s, $3 \cdot 10^{-14}$) for ILVES-M. This example indicates that, for similar execution times, the ILVES algorithms are between 500,000 and 30,000,000 times more accurate than P-LINCS algorithms. Figure 2.13 indicates that the fast convergence of the ILVES algorithms makes much more accurate solutions possible, requiring very low execution times, making it affordable to solve constraints near the limit of machine precision. We stress that increasing the accuracy of constraint solving is also desirable in simulations made with numerical single precision. In such cases, our tests indicate that the maximum enforceable tolerance is about $\mathrm{Tol} = 10^{-6}$ instead of the value $\mathrm{Tol} = 10^{-12}$ which corresponds to double precision.

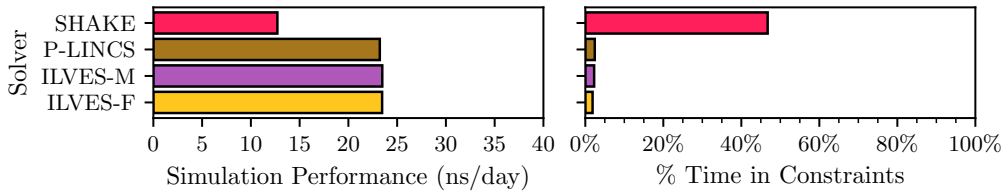
**Angular Constraints.** The discussion presented in this section so far corresponds to the case of imposing constraints on just bond lengths, which limits the maximum time step to approximately 2 fs. But, in addition to the possibility of achieving a higher degree of accuracy and computational savings that the ILVES family of algorithms provides for this very common set of constraints, its better convergence properties also allow us to cross a line unprecedented in the literature as far as we are aware. In what follows, we demonstrate that ILVES can be utilized to efficiently impose constraints on specific bond angles of hydrogen atoms, thereby enabling an increase in both the time step and the simulation throughput.

GROMACS supports constraining certain angles via the `constraints=h-angles` option. When enabled, this does not constrain all angles involving hydrogen atoms, but only a subset: specifically, angles between two hydrogens bonded to the same heavy atom (H–X–H) and angles between a hydrogen bonded to oxygen and the heavy atom connected

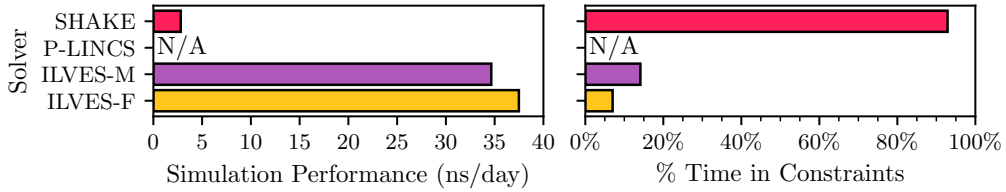
to that oxygen (X–O–H). This freezes some of the vibrational modes associated with hydrogen angular degrees of freedom, but not all.

Despite its availability, `constraints=h-angles` is generally considered experimental and has not been widely used to increase the time step in the literature. This is likely due to limitations of current constraint solvers: (P-)LINCS usually cannot handle coupled angle constraints [103], and SHAKE converges extremely slowly [93].

This behavior is illustrated in Figure 2.14, where the time step of the LIPID simulation is increased to 3.5 fs, identified as the upper limit for stable simulations with `constraints=h-angles`. The figure reports the simulated performance in nanoseconds per day and the fraction of total runtime spent on the constraint solver using SHAKE, P-LINCS, ILVES-M, and ILVES-F, under two configurations: the default settings (`constraints=h-bonds`, `ts=2 fs`, `Tol=`$10^{-4}$) and the new H-angle settings (`constraints=h-angles`, `ts=3.5 fs`, `Tol=`$10^{-4}$).




(a) H-bonds, 2 fs




(b) H-angles, 3.5 fs

Figure 2.14.: LIPID simulation performance (left) and percentage of the total execution time spent on the constraint solver (right) using a 2 fs time step with `constraints=h-bonds` (a) and a 3.5 fs time step with `constraints=h-angles` (b), across four constraint solvers: SHAKE, P-LINCS, ILVES-M, and ILVES-F. The performance is reported in nanoseconds simulated per day, using all 56 cores of an Intel Xeon Platinum 8480+ chip. P-LINCS is not compatible with `constraints=h-angles` and is therefore marked as N/A in the figure.

For this evaluation, the LIPID simulation was executed on all 56 cores of a single Intel Xeon Platinum 8480+ processor. The number of steps was increased from 50K in the previously reported results to 1.5 million to ensure stability when using H-angle constraints, corresponding to total simulated times of 3 ns for H-bonds and 5.25 ns for H-angles. The results indicate that increasing the time step from 2 fs to 3.5 fs causes SHAKE to dominate the simulation time, consuming 93% of the total runtime and severely limiting overall performance. P-LINCS cannot handle `constraints=h-angles` (marked

as N/A in the figure). In contrast, ILVES substantially improves performance, increasing the simulation rate from 23 ns/day to 38 ns/day (a 1.65× improvement) while accounting for only a small fraction of the runtime: 14% for ILVES-M and 7% for ILVES-F.

The results presented above demonstrate that ILVES enables a new strategy for increasing the integration time step without relying on potentially unphysical approximations, such as mass repartitioning. Nevertheless, further research is required. At present, time steps beyond 3.5 fs are not achievable in GROMACS, which does not support constraining a broader set of hydrogen-involving angles. Moreover, current force fields have not been parameterized to accommodate such angle constraints. Therefore, additional research is needed to determine how to correctly impose angle (and potentially dihedral) constraints, enabling ILVES to support time steps well above 4 fs, while assessing the impact of these constraints on the physical accuracy and stability of molecular dynamics simulations.

**Main Bottlenecks.** We identify three main bottlenecks in ILVES: the initialization cost, the distribution of work within each Newton iteration, and the communication overhead in parallel executions.

The initialization of ILVES is approximately 2 to 5 times more costly than that of P-LINCS, with a cost comparable to that of SHAKE. In simulations without domain decomposition, the solver is initialized only once, making this overhead negligible relative to the total execution time. In simulations with domain decomposition, however, the solver must be reinitialized every `nstlist` steps whenever the neighbor list is rebuilt, which can account for up to approximately 20% of the total constraint algorithm execution time. Despite this, all results reported in this section include initialization time, and ILVES remains as fast as or faster than MP-LINCS for the default values of `nstlist`.

Excluding communication overhead, the distribution of execution time within each Newton iteration differs between ILVES versions. In ILVES-PC and ILVES-M, the dominant cost is the solution of the linear system—via Gauss–Jordan elimination in ILVES-PC and via LU factorization with forward and backward substitution in ILVES-M—accounting for approximately 60% of each iteration. The construction of the linear system represents a significant secondary cost at approximately 25%. In ILVES-M, the vectorization of this construction step is substantially limited by vector gather operations, which involve loads from non-contiguous memory locations; using Intel's AVX-512 vector extension, these vector gathers significantly reduce the expected speedup from vectorization. The remaining time is spent updating atomic positions and velocities.

In ILVES-F, the LDLT factorization is performed only once per time step and accounts for less than 20% of the total solve time, depending on the target tolerance that affects the number of Newton iterations. The construction of the linear system is the dominant cost in ILVES-F, closely followed by the forward and backward substitution, with position and velocity updates taking the remainder. As in ILVES-M, the linear system's construction performance in ILVES-F is also limited by gather loads.

In parallel executions, communication is the dominant bottleneck of ILVES (as for P-LINCS) whether between threads in shared-memory execution or between nodes when domain decomposition is enabled (see Figure 2.10). Since the number of threads and tasks

is typically chosen to maximize performance in the force computation phase, which requires comparatively little communication, the amount of work available between successive communication points in the constraint phase is insufficient to achieve good parallel scalability in the general case. When constraining H-bonds, the number of communication points is lower than for all-bonds or H-angle constraints. In the latter two cases, the presence of sequential code to handle shared portions of the linear system further reduces parallel efficiency.

## 2.8. Conclusion

In this chapter, we have presented arguments supporting the claim that constraint equations in molecular dynamics must be solved accurately. In particular, we have shown that the default configurations of widely used MD packages can introduce errors that may accumulate over long simulations and affect their physical reliability.

To address the limitations of state-of-the-art constraint solvers, we have introduced the ILVES family of parallel algorithms, which combines Newton's method with sparse direct linear solvers. Owing to their significantly faster convergence, these methods enable both accurate and efficient constraint solving when compared with SHAKE and (P-)LINCS. The ILVES family comprises three parallel algorithms, ILVES-PC, ILVES-M, and ILVES-F, that share a common mathematical foundation.

In ILVES-PC, we exploit the repeating structural patterns in peptides and proteins to generate tailored, loop-free code for solving blocks of linear systems associated with these recurring molecular substructures. ILVES-PC leverages shared-memory parallelism and achieves performance comparable to, or often exceeding, that of SHAKE and (P-)LINCS under low-accuracy requirements, while delivering substantial speedups at higher accuracy. However, ILVES-PC is restricted to simulations of peptides and proteins in which all bonds are constrained.

ILVES-M extends the approach of ILVES-PC to create a fully generic constraint solver capable of handling any bond constraint. It leverages shared-memory parallelism, distributed-memory parallelism, and SIMD vectorization. Although ILVES-M performs roughly 10% slower than ILVES-PC due to its generality, it remains highly efficient for low-accuracy simulations and delivers significant speedups compared with state-of-the-art solvers at high accuracy.

ILVES-F redefines the equations of ILVES-M to enable the use of a faster custom linear solver. This modification transforms Newton's method underlying ILVES-M and ILVES-PC into a quasi-Newton method, which achieves linear rather than quadratic convergence. Despite this change, the linear convergence of ILVES-F is extremely fast, allowing solutions of a given accuracy to be reached in a comparable number of iterations to ILVES-PC and ILVES-M, while providing roughly a twofold performance improvement over ILVES-M.

Most importantly, while state-of-the-art solvers are limited to constraining bond lengths, ILVES can apply constraints to additional degrees of freedom in parallel, providing a foundation for increasing the simulation time step. We show that when the time step is

increased from 2 fs to 3.5 fs by constraining certain hydrogen angles using GROMACS' existing framework, P-LINCS fails to work, and SHAKE dominates the total execution time of GROMACS, significantly reducing performance. In contrast, our solvers enable a 1.65× increase in simulated time while using the same computational resources and wall-clock time.

## 2.9. Future Work

The tailored, loop-free code blocks used in ILVES-PC have proven highly effective at maximizing performance. However, a key limitation of ILVES-PC is that the code for each subblock in the linear system corresponding to the simulation constraints must be precomputed. This results in thousands of lines of code to handle the scenarios presented in this chapter, making ILVES-PC difficult to maintain. A future direction is to programmatically detect repeating subblocks at the beginning of a simulation and generate the corresponding code at run-time. This approach would simplify maintainability while preserving the same performance benefits.

We have shown that increasing the time step by applying angle constraints in combination with ILVES can yield substantial performance gains. Future research will explore how to reliably extend the time step further by constraining all hydrogen angles [73], dihedral angles [89], and other internal degrees of freedom. ILVES is expected to efficiently solve the equations associated with these additional constraints, thereby enabling larger time steps in simulations.

A thorough physical validation of the dynamics of simulations when constraining new degrees of freedom also remains an open and necessary task. The reliability study presented in this work assesses thermodynamic observables for simulations with bond constraints and a 2 fs time step; an equivalent study for simulations with additional degrees of freedom constrained and an increased time step has not been conducted here. Such work would complement existing validation studies performed with current constraint algorithms [72].

A related challenge is the reparameterization of widely used force fields to be consistent with angle constraints. Current force fields, including CHARMM [76], [77], AMBER [78], and OPLS/AA [79], were parameterized assuming flexible bond angles, and their direct use with angle constraints may not preserve the effective torsional landscape and simulation kinetics in all cases. While studies exist that successfully apply angle constraints without a dedicated reparameterization, reporting no significant effects on simulation kinetics [72], such a reparameterization would provide stronger guarantees of physical validity. Since ILVES enables the efficient application of angle constraints, we hope it will facilitate further study of these types of constraints and development of such reparameterizations.

The ILVES algorithms presented in this chapter have been designed and evaluated on CPUs. While many users still run their simulations on CPUs, molecular dynamics packages have increasingly focused on GPU implementations over the last few years, which often provide improved energy-to-solution performance compared with CPUs [185]. Current GPU implementations of constraint solvers are limited to constraining bond

lengths of hydrogen atoms, as constraining additional degrees of freedom requires extensive thread-to-thread communication.

Future work includes developing GPU versions of ILVES-M and ILVES-F capable of constraining hydrogen bond lengths. We expect these implementations to be a straightforward extension of the CPU version, in which each GPU thread handles either one atom or one cluster of bonded atoms—the same principle used by the GPU version of P-LINCS already implemented in Gromacs. The CPU-based results presented in this chapter suggest that these GPU versions of ILVES would achieve higher accuracy than current state-of-the-art solvers without increasing execution time.

A related open question is whether, by leveraging ILVES, it is possible to efficiently handle all-bonds or angle constraints in GPU simulations. Two potential approaches are an efficient GPU implementation of these constraint types, or offloading the constraint phase to the CPU while the rest of the simulation runs on the GPU. The latter has historically been avoided because of the cost of moving data between CPU and GPU, but recent architectures with unified memory, in which the CPU and GPU share a single address space [186], may make it more viable. Either approach would extend the gains demonstrated for all-bonds and angle constraints in this chapter to GPU-based simulations.

# Chapter 3.

# GenArchBench: A Genomics Benchmark Suite for Emerging Architectures

*This chapter introduces GenArchBench, a suite of 13 computationally intensive kernels extracted from widely used genomic tools. We port and optimize these kernels for the emerging Arm and RISC-V architectures, with a focus on exploiting Arm's SVE and RISC-V's RVV vector extensions. The chapter provides an in-depth performance characterization across Arm, RISC-V, and x86 reference systems, identifying current ecosystem limitations and highlighting optimization opportunities for hardware-aware genomic data analysis.*

## 3.1. Introduction

For many years, Arm processors have dominated the mobile device segment. Their energy efficiency and license-based business model have been the pillars underpinning this success. In recent years, Arm has burst onto the high-performance computing market with influential companies and consortia that have become licensees, such as Fujitsu, Amazon, Apple, NVIDIA, Samsung, AMD, Broadcom, HUAWEI, and Qualcomm. A prominent example is the Japanese supercomputer Fugaku, powered by Fujitsu's Arm-based A64FX processor, which held the top position in the TOP500 ranking from June 2020 to June 2022. Moreover, Amazon has been using Arm processors to power its cloud computing platform (AWS), starting in 2018 with the Graviton processor. They followed with the second generation of Graviton in 2019 and Graviton3 in 2022. NVIDIA Grace CPUs and Ampere servers will be leading further efforts to break through Arm in HPC.

Following a similar trajectory, although at a less mature stage, the RISC-V instruction set architecture (ISA) has gained significant momentum and popularity, largely due to its open nature, which allows for free use, modification, and distribution. This characteristic has made it an affordable and accessible alternative to traditional and proprietary ISAs, such as x86 and Arm. RISC-V's modular design allows for easy customization and scalability, making it adaptable to a wide range of applications and computing devices, from low-power devices to large-scale HPC supercomputers. The RISC-V project has garnered support from industry and academia, helping to drive its development forward. Developing RISC-V microprocessors has become a strategic priority for the European Union, as evidenced by many ongoing initiatives and research projects. To ensure competitiveness, it is crucial to consolidate a strong RISC-V community that can foster research and development of RISC-V technology for HPC-demanding applications,

including climate modeling [187], molecular simulations [188], machine learning [189], and computational genomics [45].

As a result, large-scale computing infrastructures, traditionally dominated by x86 and IBM Power processors, now face additional competitive alternatives in the form of Arm and RISC-V architectures. Nonetheless, the majority of scientific HPC software remains only partially adapted or insufficiently optimized for these emerging architectures.

Over the last decade, genome sequencing has become the cornerstone of genomics and modern precision medicine. Due to the rapid improvement of sequencing technologies, it is currently possible to sequence an individual's genome in less than 24h. This breakthrough has enabled effective personalized healthcare, allowing the diagnosis and treatment of diseases based on each person's unique genomic disposition [37]. Furthermore, genome sequencing has also been proven crucial in cancer studies [39], drug development [41], or COVID-19 outbreak control [42]. In the past 20 years, genome sequencing costs have dropped dramatically, and the amount of sequencing data produced yearly has increased exponentially. More notably, this increase in data production has outperformed the pace of Moore's law. As a result, a significant bottleneck in current genome sequencing analysis is placed on the computational side, executing computationally-intensive genomics tools and pipelines.

Historically, genome analysis pipelines have been designed and optimized primarily for x86 architectures. With the rapid emergence of Arm-based HPC servers and the growing investment in RISC-V processors, adapting and optimizing genomics software for these alternative architectures has become increasingly important. For that, we have selected 13 computationally-demanding CPU kernels from the most widely-used genomics tools, and we have included them in a benchmark suite called GenArchBench. All the kernels exploit multi-core parallelism and implement common stages from widely-used genome analysis pipelines such as base-calling, read mapping, variant calling, and de novo assembly. GenArchBench also provides input datasets and their corresponding ground-truth outputs, along with automated regression tests to verify correctness and facilitate reproducibility.

Beyond the benchmark suite itself, this work presents code adaptations and architecture-aware optimizations targeting both Arm and RISC-V processors. Several kernels have been vectorized using Arm Scalable Vector Extensions (SVE) and the RISC-V Vector Extension (RVV). The Arm implementations additionally leverage specialized HPC libraries and incorporate algorithmic and low-level code optimizations tailored to the microarchitectural characteristics of Arm processors. For RISC-V, we provide a detailed analysis of the current state of hardware platforms, the ISA, and the available toolchains, with the objective of assessing the maturity and suitability of RISC-V systems for genomics workloads. This analysis is intended both to assist developers interested in porting scientific applications to RISC-V and to inform processor designers about existing limitations and opportunities for improvement.

We further present an extensive performance characterization of GenArchBench across multiple architectures, including two Arm CPUs, two RISC-V CPUs, and two reference x86 systems. The evaluation compares the Fujitsu A64FX, Amazon Graviton3, Intel Xeon Skylake Platinum 8160, and AMD EPYC 7742 Rome, as well as the HiFive Unmatched

and Allwinner D1 RISC-V platforms, against the Skylake system.

For Arm processors, the characterization includes the kernels' instruction breakdown, single-threaded and multi-threaded performance evaluations, a microarchitecture bottleneck analysis, and an energy-to-solution study in the different processors. Ultimately, we evaluate the performance impact of these optimizations by integrating two of the accelerated kernels in a production-ready tool used in a myriad of genome analysis pipelines.

For RISC-V systems, the characterization focuses on single-threaded performance and parallel scalability on the multi-core HiFive Unmatched platform.

The GenArch Benchmark suite, with all kernels ported to Arm, is available at https://github.com/LorienLV/genarchbench, while the RISC-V port can be found at https://github.com/LorienLV/_PAPER-JP23_RISC-V_for_Genome_Data_Analysis_Opportunities_and_Challenges. Both repositories provide all necessary information to download the input datasets.

## 3.2. Background

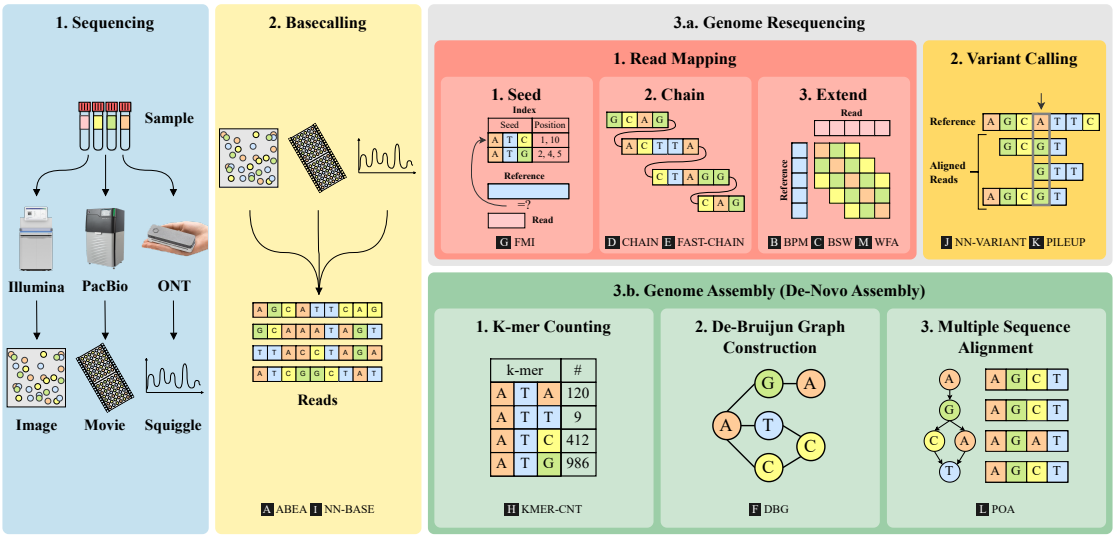



Figure 3.1.: Workflow diagram of common genome analysis pipelines. Going from (1) sequencing, through (2) basecalling, to (3.a) genome resequencing or (3.b) and genome assembly. The figure shows the different computational kernels used within each stage or tool.

Genome data analysis pipelines comprise multiple stages and computational tools, from sequencing biological samples to deriving meaningful data analysis results for scientists and healthcare professionals. This section introduces the main sequencing technologies, pipelines, and tools used in common genome analysis (Figure 3.1 shows a succinct graphic summary).

### 3.2.1. Sequencing Technologies

Before any computational analysis can be performed, biological DNA samples must be converted to digital data. This process is performed by the sequencing machines (Figure 3.1-1), and, despite the remarkable advances in the last decades, these machines are still unable to read a complete DNA molecule from end to end. Instead, sequencing machines allow reading relatively small chunks of DNA, called reads or fragments, from random locations within the donor's DNA genome. Afterwards, sequenced reads must be jigsawed together to reconstruct or reassemble the original donor's genome.

Sequencing machines are commonly categorized into three generations based on their technological advancements. The first sequencing technologies (Sanger et al. [190] and Maxam et al. [191]) were developed in 1977 and used to sequence the first draft of the human genome in 2000 [192]. Since then, sequencing technologies have evolved quickly, simplifying the sequencing process and increasing the data-production throughput. In the mid-2000s, second-generation technologies [193] were introduced and soon replaced first-generation technologies. Second-generation technology can generate fixed-length sequences of 100-300 bps at a throughput of tens of gigabytes per hour and with a low reading error rate (0.1% of the read length). At present, Illumina dominates the market of second-generation sequencing machines. Recently introduced third-generation technologies, known as long-read sequencing, can read variable-length sequences of considerable length (i.e., tens of kilo base-pairs) at the expense of lower production throughput (less than 10 Gb/hour) and higher reading error rate (0.1-10% of the read length). Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT) are the most notable manufacturers of third-generation sequencing technologies.

### 3.2.2. Genome Data Analysis Pipelines and Tools

Before any processing can be performed, the sequencing machines' raw signals must be transformed into sequences of nucleotides (A, C, G, T). This process is called basecalling (Figure 3.1-2). Typically, a specialized basecalling tool is used to perform this process tailored to each sequencing technology. For instance, Bonito [194] and Guppy [195] are two of the most widely-used tools for basecalling Oxford Nanopore's raw-signal output.

Once the sequences of nucleotides are decoded, sequenced reads must be processed and analyzed to derive meaningful biological insights. Although many different genome analyses can be performed using sequenced data, most analyses begin with either genome resequencing (3.1-3.a) or genome assembly (3.1-3.b). Both analyses seek to reconstruct the sample's genome by putting together all the sequenced reads.

#### Genome Resequencing

The most common approach to reconstructing the sample's genome is by resequencing and involves reconstructing the sample's genome using a previously known reference genome. For that, each sequenced read is located and matched to the most likely originating position in the reference genome, allowing for small differences (e.g., mismatches, insertions, and deletions). This process is called read mapping (Figure 3.1-3.a.1) and it is implemented

by many tools like BWA-MEM2 [196], [197], Minimap2 [198], Bowtie2 [199], [200], and GEM [201]. Read mapping is one of the most computationally expensive steps in all genome sequence analyses. Consequently, read mapping has been extensively studied and optimized.

Most sequence mappers are based on the seed-chain-extend technique. This technique implements three algorithmic steps to swiftly locate and align a sequence with a reference genome. During the first step, known as seeding (Figure 3.1-3.a.1.1), the mapper searches small subsequences of the reads (seeds) in the reference leveraging an index structure. The most widely-used indexes used for seeding are FM-Index [202] and hash-tables [203], [204]. Seeding reduces the potential number of locations in the reference where a sequence can match, decreasing the amount of work performed in subsequent steps. Afterwards, a chaining step (Figure 3.1-3.a.1.2) is performed to further reduce the list of possible matching locations in the reference. During the chaining step, all the mapped seeds are processed to find a collinear chain of seeds that can potentially match the input sequence. Finally, during the extension or alignment step (Figure 3.1-3.a.1.3), the input sequence is aligned against the candidate location in the reference genome, discovering the differences between the donor's sequence and the reference genome. Usually, a dynamic programming-based algorithm, such as Needleman–Wunsch [205] or Smith-Waterman-Gotoh [206], [207], is used to compute the alignment.

After sequence mapping, once the reads are located in the reference genome, a variant calling algorithm (Figure 3.1-3.a.2) determines the variants and mutations between the donor's genome and the reference genome. These variations provide crucial insights into the genetic makeup of the sequenced individual, potentially revealing genetic variations that may be associated with diseases and health conditions. Notable examples of widely-used variant callers are GATK Haplotype-Caller [208], Platypus [209], Clair [210], [211], DeepVariant [212] and Medaka [213].

### Genome Assembly

Despite the simplicity and effectiveness of genome resequencing, there is still a lack of high-quality reference genomes for many species. In those situations, genome de-novo assembly (Figure 3.1-3.b) is used to reconstruct the donor's genome from scratch, jigsawing the sequenced reads together.

Most popular de novo assembly methods rely on de Bruijn graphs. For a given set of sequences, its corresponding de Bruijn graph contains a node for each sequence's k-mer (i.e., sub-string of length $k$ nucleotides) and an edge that connects adjacent and overlapping k-mers. Before constructing the de Bruijn graph of a set of input sequences, the number of unique k-mers in the reads is counted (Figure 3.1-3.b.1) to prune the least frequent ones (likely artefacts of the sequencing process). Afterwards, the de Bruijn graph is constructed (Figure 3.1-3.b.2). Then, the consensus sequence is derived using multiple sequence alignment (MSA) algorithms (Figure 3.1-3) and the constructed de Bruijn graph. Notable examples of de Bruijn graph-based assemblers are Flye [214], Canu [215], and Racon [216].

### Metagenomics

Beyond genome resequencing, variant calling, and de novo assembly, many previously described analysis steps and tools can be found in other genome analysis pipelines. This is the case for many metagenomics analysis pipelines. Metagenomics pipelines seek to analyze genomic information from mixed microbial communities, providing insights into the diversity, interactions, and function of microorganisms present in an environmental sample. Metagenomics analyses are performed using tools such as Centrifuge [217], RawMap [218], UNCALLED [219], ReadFish [220], Kraken2 [221] and Clark [222]. These tools employ k-mer counting (Figure 3.1-3.b.1) and seeding techniques (Figure 3.1-3.a.1.1) for their analysis. Moreover, variant callers like GATK Haplotype Caller [208] and Platypus [209] are used to construct De Bruijn graphs (Figure 3.1-3.b.2) and correct artifacts produced during the mapping process (Figure 3.1-3.a.1). Furthermore, the chaining process (Figure 3.1-3.a.1.2) is also utilized for genome assembly when using alternative approaches based on de Bruijn graphs [223].

## 3.3. GenArch Benchmark Suite

The GenArch benchmark suite comprises 13 multi-threaded CPU kernels derived from the most widely used genomics tools and covers the most important genome sequencing steps. It includes ten kernels from the GenomicsBench [224] benchmark suite and three additional kernels: the Bit-Parallel Myers algorithm [225] (BPM), the Wavefront Alignment algorithm [226] (WFA), and FAST-CHAIN [227]. BPM and WFA complement the sequence alignment kernels of GenomicsBench to better capture contemporary trends. Additionally, FAST-CHAIN [227] is a recent vector-enabled reimplementation of the CHAIN kernel present in GenomicsBench, which allows us to further explore the capabilities of SVE.

Additionally, GenArchBench includes input datasets for each kernel (i.e., a small dataset and a large dataset per kernel) and their corresponding outputs to be used as ground truth. The small datasets have been sized to require single-threaded execution times no longer than a few minutes (for testing purposes); meanwhile, large datasets require several minutes (for performance evaluation purposes). For convenience, we provide automatic regression tests for all the kernels to verify the correctness of the outputs.

Although GPU-accelerated genomics tools are increasingly available and some kernels included in GenArchBench can exploit the capabilities of modern GPUs, many widely-used production pipelines—including BWA-MEM2 [197], Minimap2 [198], and GATK HaplotypeCaller [208]—rely primarily on CPU computation. GPU acceleration requires specialized hardware that is not universally available across machines and HPC clusters; for example, the Fugaku supercomputer, whose CPUs are evaluated in this work, does not include GPUs in its compute nodes. In this work, we focus on porting, accelerating, and evaluating the performance of genomics kernels on Arm and RISC-V processors.

This section presents GenArchBench kernels, briefly describing its functionality, which tools use them, and a description of their usage and inputs. Table 3.1 summarizes the name, use case, and application of origin of each kernel.

Table 3.1.: Summary of kernels included in GenArchBench.

| Kernel | Use-Case | Genomics Tool |
|---|---|---|
| ABEA | Basecalling | Nanopolish |
| BPM | Read mapping | GenArchBench |
| BSW | Read mapping | BWA-MEM2 |
| CHAIN | Read mapping | Minimap2 |
| FAST-CHAIN | Read mapping | Minimap2 |
| DBG | De-novo assembly | Platypus |
| FMI | Read mapping | BWA-MEM2 |
| KCNT | De-novo assembly | Flye |
| NN-BASE | Basecalling | Bonito |
| NN-VARIANT | Variant calling | Clair3 |
| PILEUP | Variant calling | Medaka |
| POA | De-novo assembly | SPOA |
| WFA | Read mapping | AnchorWave |

The GenArchBench benchmark suite is available at https://github.com/LorienLV/genarchbench.

### Adaptive Banded Signal to Event Alignment (ABEA)

ABEA is a dynamic programming algorithm that compares raw nanopore signals from ONT sequencing machines to a reference genome sequence. ABEA's implementation is based on the Suzuki-Kasahara (SK) [228] algorithm. This step is performed in some tools, such as Nanopolish [229], to correct errors produced in the basecalling process (Figure 3.1-2). For GenArchBench, we have used the CPU implementation of f5c [230], a version of ABEA based on Nanopolish's, optimized for both CPU-only and hybrid CPU/GPU executions. This implementation of ABEA exploits coarse-grain multi-threading by dividing the raw signals of the input between the available cores. Since the signals are not of regular size, f5c implements work-stealing to improve load balance. The small and large inputs comprise 1K and 10K raw FAST5 (ONT) reads from chromosome 22 of NA12878 and GRCh38 as the reference genome [231].

### Bit-Parallel Myers (BPM)

BPM [225] is a dynamic programming algorithm that finds all locations a query string of size $m$ matches a reference string of size $n$ with $k$ or fewer differences (Figure 3.1-3.a.1.3). It computes the approximate string matching of two strings in $\mathcal{O}(mn/w)$ time, where $w$ is the word size of the machine. BPM is used in read mapping tools, such as GEM-Mapper [201], Edlib [232], GraphAligner [233], or Hobbes [234]. For GenArchBench, we have used an in-house implementation of the algorithm that exploits multi-threading by assigning different pairs of strings to different threads. The small and large inputs comprise 100K and 10M sequence pairs from human sample SRR7733443 downloaded from the sequence read archive [235].

### Banded Smith-Waterman (BSW)

The Smith-Waterman algorithm [206] is a dynamic programming algorithm that computes the local sequence alignment of two sequences of length $m$ and $n$, respectively, in $\mathcal{O}(mn)$ time and space. A banded version of Smith-Waterman [236] is used to align sequences with a maximum of $w$ insertions/deletions, reducing the time and space complexity to $\mathcal{O}(wn)$ (Figure 3.1-3.a.1.3). BSW is used in variant discovery tools such as GATK [208], and in sequence alignment software like BWA-MEM [196], [197]. For GenArchBench, we have used BWA-MEM2's x86-vectorized implementation of BSW. In order to exploit multi-threading, the set of pairs of strings to align is dynamically divided between processors. The small and large inputs comprise 100K and 10M sequence pairs from human sample SRR7733443 [235].

### Seed Chaining (CHAIN)

Given the set of seeds from a DNA sequence (read) mapped to another sequence, such as the reference genome, the chaining step (Figure 3.1-3.a.1.2) aims to find a chain of co-linear seeds. This is a time-consuming step performed by alignment tools, such as Minimap2, and by de novo assemblers like Flye [214] or Canu [215]. We have used the implementation of CHAIN found in GenomicsBench that extends Minimap2's to exploit inter-task parallelism across reads. The small and large inputs comprise the seeds from 1K, and 10K reads of Pacbio's *Caenorhabditis elegans* worm sequence data [237].

### SIMD Seed Chaining (FAST-CHAIN)

The previously presented implementation of the CHAIN algorithm utilizes heuristics to stop executing when the result is sufficiently good. This speedups execution at the cost of accuracy, and it hinders the vectorization of the kernel. FAST-CHAIN [227] is an x86-vectorized version of CHAIN that removes the heuristics to exploit SIMD computation. As a result, FAST-CHAIN outputs accurate results and presents performance gains compared to CHAIN. FAST-CHAIN uses the same inputs as CHAIN.

### De Bruijn Graph Construction (DBG)

The De Bruijn graph (DBG) of an input set of reads is used to represent the overlaps between the substrings of length $k$ (k-mers) found in the input (Figure 3.1-3.b.2). Each node of the graph represents a k-mer and the edges connect adjacent k-mers in the input set. The construction of these graphs is a time-consuming step in de-novo assemblers like Flye [214], Canu [215], or Racon [216], and in variant callers such as GATK [208] and Platypus [209]. For GenArchBench, we have used the DBG construction of Platypus, which exploits parallelism by assigning different regions of the input to different threads. Both inputs employ chromosome 22 of BWA-MEM aligned records from the Platinum Genomes dataset [238]. The small input uses bases 16M-16.5M, while the large input uses the entire chromosome.

### FM-Index Search (FMI)

The FM-index is a compressed substring index based on the Burrows-Wheeler transform [239]. Given a sub-string $s$, the FM-index can be used to find the location of $s$ in the reference genome in $\mathcal{O}(|s|)$ time, where $|s|$ is the length of the sub-string (Figure 3.1-3.a.1.1). The FM-index data structure is used in sequence alignment tools such as BWA-MEM [196], [197] or Bowtie2 [200], and in metagenomic classification software like Centrifuge [217]. For GenArchBench, we have used the super-maximal exact match kernel of BWA-MEM2, which utilizes the FM-Index structure. This kernel exploits parallelism by dynamically assigning batches of reads among threads. The small and large inputs comprise 1M and 10M pairs of 151 bases from human sample SRR7733443 [235].

### K-mer Counting (KMER-CNT)

K-mer counting aims to count the number of occurrences of each k-mer in an input sequence (Figure 3.1-3.b.1). This task is performed in de novo assemblers such as Flye [214] or Canu [215] and in metagenomics classification software like Clark [222]. Additionally, note that the functionality of KMER-CNT is very similar to accessing large lookup tables, as done in state-of-the-art mappers like Minimap2 [198]. For GenArchBench, we have used the k-mer counting kernel of Flye. This implementation divides the input-reads between threads and relies on the thread-safe hash-map implementation of Libcuckoo library [240] to concurrently increase the number of individual k-mers shown by each thread. The small and large inputs comprise 1K and 50K *Escherichia coli* Oxford Nanopore reads sequenced by Loman Labs [241].

### Neural Network-Based Base Calling (NN-BASE)

ONT sequencing machines monitor changes in an electrical current as single strands of DNA or RNA pass through a protein nanopore. These changes in the electrical current are then converted to a sequence of nucleotide bases in the basecalling process (Figure 3.1-2). The analog signal inevitably contains ambiguities due to noise or measurement errors. Some basecallers, such as Guppy [195] and Bonito [194], rely on neural networks to solve these ambiguities, determining the most likely observed nucleotide in each part of the electrical current. For GenArchBench, we have used Bonito's deep-learning basecaller (NN-BASE), which depends on the PyTorch library [242]. Bonito splits the input signal into smaller chunks of regular size and feeds them to a PyTorch neural network that internally exploits multi-threading. The small and large inputs comprise 1 and 10 raw FAST5 reads from chromosome 20 of NA12878, obtained from the Nanopore WGS Consortium [231].

### Neural Network-Based Variant Calling (NN-VARIANT)

Variant calling is the process of detecting the differences (variants or mutations) between the aligned reads and the reference genome (Figure 3.1-3.a.2). This is a costly process performed by statistics-based variant callers, such as GATK HaplotypeCaller [208] or

Platypus [209], and deep-learning variant callers, such as Clair [210], [211], DeepVariant [212] or Medaka [213]. For GenArchBench, we have used the second generation of Clair variant caller (Clair3), based on the TensorFlow framework [243]. Clair3 exploits parallelism by dividing the input into regular-sized chunks, and each of these chunks is processed by one thread using TensorFlow. Our small and large inputs comprise 100K and 10M reference positions, respectively, of chromosome 20 of HG002 from NIST's Genome in a Bottle (GIAB) project [244]. We use Clair3's ONT pre-trained model `r941_prom_hac_g360+g422` [245].

### Pileup Counting (PILEUP)

Given the alignment data of a set of aligned reads to a region of a reference genome, usually a SAM or BAM file [246], pileup counting is the process of summarizing the base-pair information at each chromosomal position. This summary, called pileup, is the customary input for long-read neural network variant callers such as Clair [210], [211] or Medaka [213] (Figure 3.1-3.a.2). For GenArchBench, we have used the pileup counting implementation of Medaka, which exploits multi-threaded parallelism by distributing 100-kilobase regions of the reference genome between threads. The small input comprises bases 1-1499707 of the *Staphylococcus aureus* genome [195], and the large input comprises bases 1-1412827 of chromosome 20 of sample HG002 [244].

### Partial-Order Alignment (POA)

The construction of an overlap graph from a set of reads leads to an approximate representation of the original sample's genome. To determine the consensus genome of the sample, the alignment of all the reads against each other is performed in a process called multiple sequence alignment (MSA) (Figure 3.1-3.b.3). The Partial-Order Alignment (POA) algorithm [247] computes the MSA of all sequences by incrementally constructing a partially-ordered graph aligning new sequences to it using a dynamic programming algorithm such as Smith-Waterman [206] or Needleman-Wunsch [205]. The multiple sequence alignment (consensus sequence) is inferred from the graph by using the Heaviest Bundle algorithm [248]. POA is used in software packages such as Nanopolish [229] or Racon [216]. For GenarchBench, we have used the SIMD-optimized version of POA of the SPOA library [249]. SPOA exploits multi-threading by computing the partially-ordered graph of multiple sets of sequences in parallel. The small and large inputs comprise 1K and 6K sets of multiple sequences aligned to a reference genome, each containing between 5 and 115 sequences. This data comes from Minimap2's polishing step of the Flye-assembled *Staphylococcus aureus* genome [195].

### Wavefront Alignment (WFA)

The wavefront alignment algorithm (WFA) [226] is a pairwise alignment algorithm (Figure 3.1-3.a.1.3) that takes advantage of homologous regions between the sequences to accelerate the alignment process. As opposed to traditional dynamic programming algorithms that run in quadratic time, WFA time complexity is $\mathcal{O}(ns)$, proportional

to the read length $n$ and the alignment score $s$, using $\mathcal{O}(s^2)$ memory. The wavefront algorithm is used in tools such as wfmash [250], AnchorWave [251], or AncestralClust [252]. GenArchBench uses a custom multi-threaded implementation of the algorithm, in which each thread works in the alignment of a pair of strings. The small and large inputs comprise 100K and 1M sequence pairs from human sample SRR7733443 [235].

## 3.4. Target Hardware Systems

Our experimental setup spans three architectures and includes six platforms in total. For Arm, we evaluate an Arm A64FX processor (A64FX) and a c7g.16xlarge Amazon EC2 instance based on Graviton3. For RISC-V, we use the HiFive Unmatched from SiFive and the Allwinner D1 (D1-H). To provide baseline performance on a mature and widely adopted architecture in HPC environments, we also include two x86_64 systems: a dual-socket Intel Xeon Skylake Platinum 8160 machine (SKX) and a compute node equipped with a single AMD EPYC 7742 Rome processor (Rome).

Table 3.2 provides a summary of the evaluated systems, covering their principal characteristics, micro-architectural details, and documented or empirically measured memory-access latencies at each level of the memory hierarchy. The following subsections discuss each of these aspects in turn.

None of the RISC-V machines studied in this chapter are HPC-class machines; they are development or embedded boards. However, at the time this work were conducted, they represented the most powerful RISC-V hardware publicly available. Their evaluation nonetheless provides insight into the current state of the RISC-V ISA and toolchain, as well as the trends shaping the development of RISC-V processors.

### 3.4.1. Arm Systems

#### Fujitsu A64FX (A64FX)

The A64FX system [253], [254] is based on a Fujitsu A64FX processor, an Arm CPU designed for high-performance computing and notably deployed in the Fugaku supercomputer. The chip is organized into four core memory groups (CMGs), each constituting an independent NUMA domain with 12 compute cores and one assistant core dedicated to system tasks (running daemons, I/O, asynchronous MPI, etc.), resulting in 48 compute cores in total. The processor operates at a fixed frequency of 2.2 GHz.

Each core integrates private 64 KB L1 instruction and 64 KB L1 data caches. The architecture does not implement a private L2 cache; instead, each CMG provides an 8 MB shared last-level cache, resulting in four LLC slices across the processor.

Main memory is implemented using on-package HBM2. Every NUMA domain owns 8 GB of local HBM2 memory and can access the remaining domains through a ring bus, yielding 32 GB of total main memory. The peak theoretical memory bandwidth reaches up to $4 \times 256$ GB/s.

Memory latencies to remote NUMA domains were measured with the LMbench benchmark [255], while the remaining values are obtained from official documentation [254].

Table 3.2.: Summary of the evaluated systems' characteristics, organized into three groups: main system properties, micro-architectural details, and documented or measured memory-access latencies at each level of the memory hierarchy. *Neoverse V1 CPU defaults.

| | **A64FX** | **Graviton3** | **Unmatched** | **Allwinner** | **SKX** | **Rome** |
|---|---|---|---|---|---|---|
| **Year** | 2019 | 2022 | 2021 | 2021 | 2017 | 2019 |
| **# Cores** | $4 \times 12$ (+4) | 64 | $4 + 1$ | 1 | $2 \times 24$ | 64 |
| **Frequency** | 2.2 GHz | 2.6 GHz | 1.2 GHz | 1 GHz | 1-2.1 GHz | 1.5-2.25 GHz |
| **Power** | 120 W | N/A | N/A | N/A | $2 \times 150$ W | 225 W |
| **Mem. cap.** | $4 \times 8$ GB | $8 \times 16$ GB | 16 GB | 2 GB | $2 \times 6 \times 8$ GB | $16 \times 64$ GB |
| **Mem. tech.** | HBM2 | DDR5 4800 MHz | DDR4 933 MHz | DDR3 800 MHz | DDR4 2667 MHz | DDR4 3200 MHz |
| **Bandwidth** | $4 \times 256$ GB/s | 300 GB/s | N/A | N/A | $2 \times 120$ GB/s | 204.8 GB/s |
| **L1i** | 64 KB (4-way) | 64 KB | 32 KB | 32 KB | 32 KB (8-way) | 32 KB (8-way) |
| **L1d** | 64 KB (4-way) | 64 KB | 32 KB | 32 KB | 32 KB (8-way) | 32 KB (8-way) |
| **L2** | — | 1 MB | — | — | 1 MB (16-way) | 512 KB (8-way) |
| **LLC** | $4 \times 8$ MB (16-way) | 32 MB | 2 MB | — | $2 \times 33$ MB (11-way) | $16 \times 16$ MB (16-way) |
| **Extensions** | NEON/SVE 512b | NEON/SVE 256b | RV64GC+IMAC | RV64GCV (128b) | SSE/AVX2/AVX512 | SSE/AVX2 |
| **Pipeline** | out-of-order | out-of-order | in-order | in-order | out-of-order | out-of-order |
| **# General reg.** | 96 | N/A | N/A | N/A | 180 | 180 |
| **# SIMD/FP reg.** | 128 | N/A | N/A | N/A | 168 | 160 |
| **Issue width** | 7 ($\mu$OP) | 15 ($\mu$OP) | 2 instr. | 1 instr. | 8 ($\mu$OP) | 11 ($\mu$OP) |
| **Commit width** | 4 ($\mu$OP) | N/A | 2 instr. | 1 instr. | 4-8 ($\mu$OP) | 8 (MOP) |
| **ROB (entries)** | 128 | 256* | N/A | N/A | 224 | 224 |
| **LB (entries)** | 40 | 85* | N/A | N/A | 72 | 44 |
| **SB (entries)** | 24 | 90* | N/A | N/A | 56 | 48 |
| **RS (entries)** | 2×20 + 2×10 + 19 | N/A | N/A | N/A | 97 | 4×16 + 28 + 36 |
| **L1 Lat.** | 2.3–5 ns | 1.5 ns | N/A | N/A | 1.9 ns | 1.8 ns |
| **L2 Lat.** | — | 4.6 ns | — | — | 6.7 ns | 3.5 ns |
| **LLC Lat.** | 16.8 – 21.4 ns | 33.1 ns | N/A | — | 25.1 ns | 13.0 ns |
| **Local Mem. Lat.** | 118.2 – 126.4 ns | 153.5 ns | M/A | N/A | 86.2 ns | 121.5 ns |
| **Remote Mem. Lat.** | 187.7 – 242.3 ns | — | — | — | 144.0 ns | — |

Latencies depend on the instruction type and range from approximately 2.3 ns for scalar L1 accesses to over 200 ns for far-remote NUMA memory.

From a microarchitectural standpoint, the A64FX implements an out-of-order pipeline with a 7 $\mu$OP issue width and a 4 $\mu$OP commit width. It implements 96 general-purpose physical registers, 128 SIMD/FP registers, a 128-entry reorder buffer, 40 load-buffer entries, and 24 store-buffer entries. Reservation stations are distributed across integer, floating-point/SIMD, address-generation, and branch pipelines.

Regarding SIMD support, the A64FX is the first commercial processor to implement the Armv8.2-A Scalable Vector Extension (SVE) [256]. SVE is vector-length agnostic, allowing binaries to run on implementations ranging from 128 to 2048 bits. The A64FX implements 512-bit SVE registers in addition to 128-bit NEON.

#### Amazon AWS Graviton3 (Graviton3)

The Graviton3 platform features one Amazon Graviton3 processor, which implements the Arm Neoverse V1 microarchitecture and targets both cloud and HPC workloads. The evaluated configuration provides 64 out-of-order cores organized within a single NUMA domain and operates at frequencies of up to 2.6 GHz.

Each core integrates private 64 KB L1 instruction and 64 KB L1 data caches together with a private 1 MB L2 cache, while all cores share a 32 MB last-level cache.

The system is equipped with $8 \times 16$ GB DDR5-4800 memory channels, providing 128 GB of main memory and a peak theoretical bandwidth of 300 GB/s.

Memory latencies were measured using the LMbench benchmark [255], ranging from approximately 1.5 ns at the L1 level to 153.5 ns for main memory accesses.

Graviton3 implements a wide out-of-order pipeline with up to 15 $\mu$OP issue width. Public documentation of several internal resources is limited; when unavailable, Neoverse V1 default values are assumed [257], including a 256-entry reorder buffer, 85 load-buffer entries, and 90 store-buffer entries.

SIMD capabilities include both 128-bit NEON and 256-bit SVE vector extensions.

### 3.4.2. RISC-V Systems

#### HiFive Unmatched (Unmatched)

The Unmatched system is a RISC-V Linux development board powered by the SiFive Freedom U740 (FU740) system-on-chip, primarily intended for development and prototyping purposes. The FU740 integrates four SiFive U74 compute cores and one SiFive S71 auxiliary core designed for real-time and system tasks. The SoC operates at 1.2 GHz.

Each compute core provides a 32 KB instruction cache and a 32 KB data cache and implements the RV64GC instruction set architecture with a dual-issue in-order 64-bit pipeline. The auxiliary core integrates a 16 KB instruction cache and an 8 KB tightly integrated data memory and implements the RV64IMAC ISA. The FU740 also includes a coherent banked 2 MB shared LLC cache.

The platform is equipped with 16 GB of DDR4 main memory operating at 1866 MT/s.

Publicly available documentation describing deeper microarchitectural resources, such as reorder buffers, issue queues, load/store buffers, or precise latency figures, is limited. Consequently, these parameters are reported as not available in the corresponding tables.

The Unmatched platform does not implement the RISC-V vector extension (RVV).

#### Allwinner D1 (Allwinner)

The Allwinner system is based on the Allwinner D1 (D1-H) system-on-chip, which targets low-power embedded and edge devices. The SoC integrates a single Alibaba T-Head XuanTie C906 core running at 1 GHz and follows an in-order single-issue 64-bit pipeline design.

The processor implements a single cache level consisting of private 32 KB L1 instruction and 32 KB L1 data caches, without additional shared cache levels.

The platform is equipped with 2 GB of DDR3 main memory.

As with the Unmatched platform, publicly available documentation regarding detailed microarchitectural resources and precise memory-access latencies is scarce, and these parameters are therefore reported as not available in the corresponding tables.

The core implements the RV64GCV instruction set architecture, including the RISC-V vector extension v0.7.1 with support for vector lengths of up to 128 bits.

### 3.4.3. x86 Systems

#### Intel Xeon Skylake Platinum 8160 (SKX)

The SKX platform [258] is a dual-socket x86_64 server system, each socket featuring an Intel Xeon Skylake Platinum 8160 processor with 24 out-of-order cores, for a total of 48 physical cores in the evaluated configuration.

The processor operates within a dynamic frequency range between 1 GHz and 2.1 GHz, with frequency scaling determined by workload characteristics. When executing AVX or AVX512 instructions, the operating frequency is reduced [259]. Simultaneous multi-threading (SMT) is supported by the architecture but was disabled in all experiments.

Each core provides private 32 KB L1 instruction and 32 KB L1 data caches together with a private 1 MB L2 cache. At the last level, every socket integrates a 33 MB shared LLC, yielding two LLC domains in the dual-socket configuration.

The system is equipped with $2 \times 6 \times 8$ GB of DDR4-2667 memory organized as two NUMA nodes, one per socket. The theoretical peak memory bandwidth reaches up to 120 GB/s per socket, or 240 GB/s in total.

Memory latencies were measured using Intel Memory Latency Checker, ranging from 1.9 ns for L1 accesses to 86.2 ns for local main memory and 144 ns for remote-socket memory.

From a microarchitectural standpoint, SKX implements an out-of-order pipeline with an issue width of up to 8 $\mu$OPs and a commit width between 4 and 8 $\mu$OPs. The processor can fuse pairs of micro-operations into a single fused micro-operation, increasing the theoretical commit rate from four to eight micro-operations. It provides 180 general-purpose physical

registers, 168 SIMD/FP registers, a 224-entry reorder buffer, 72 load-buffer entries, 56 store-buffer entries, and a unified 97-entry reservation station.

In terms of SIMD capabilities, SKX supports SSE, AVX2, and AVX512 instruction sets with vector widths of 128, 256, and 512 bits, respectively.

### AMD EPYC 7742 Rome (Rome)

The Rome system [260], [261] is based on an AMD EPYC 7742 processor organized into eight core chiplets, referred to as core complex dies (CCDs), which are connected to a central I/O die responsible for memory and I/O management. Each CCD contains two core complexes (CCXs), and each CCX integrates four cores, providing 64 physical cores.

The operating frequency ranges between 1.5 GHz and 2.25 GHz depending on workload conditions. As in the SKX system, simultaneous multi-threading (SMT) is supported but was disabled in this evaluation.

Each core includes private 32 KB L1 instruction and 32 KB L1 data caches together with a private 512 KB L2 cache. At the last level, each 4-core CCX shares a 16 MB LLC slice, resulting in multiple LLC domains across the processor. The evaluated node is equipped with $16 \times 64$ GB of DDR4-3200 memory, providing 1 TB of main memory and a theoretical peak bandwidth of 204.8 GB/s.

Memory latencies were measured using the LMbench benchmark [255], ranging from 1.8 ns at the L1 level to approximately 121.5 ns for main memory accesses.

Rome follows an out-of-order design with an issue width of up to 11 $\mu$OPs and an 8-macro-operation commit width. It provides 180 general-purpose physical registers, 160 SIMD/FP registers, a 224-entry reorder buffer, 44 load-buffer entries, and 48 store-buffer entries. Reservation stations are partitioned into multiple structures, including $4 \times 16$ integer entries, 28 address-generation entries, and 36 floating-point/SIMD entries.

SIMD support includes SSE and AVX2 instruction sets with vector widths of 128 and 256 bits.

## 3.5. Porting GenArchBench Kernels to Arm

Most GenArchBench kernels target x86 architectures and have not been extensively tested or optimized for Arm machines. Thus, it was expected that some kernels could run into failures and even generate incorrect results. To verify the execution of the kernels, we used the SKX system to compute the correct output for all kernels and inputs (i.e., ground truth).

For our experiments, we used the GNU compiler (GCC) on Graviton3 (v11.2.0), SKX (v10.1.0), and Rome (v10.2.0). On the A64FX, we used GCC (v10.2.0) and the Fujitsu Compiler (FCC) (v4.2.0b). For most kernels, FCC-compiled binaries exhibited better performance. The Fujitsu Compiler implements two compilation modes: a traditional mode (Trad) based on compilers for earlier systems and a Clang mode based on Clang/LLVM. In all cases, we obtained better execution times when compiling with FCC's Clang mode. We lacked FCC-compiled versions of key-optimized Python libraries. For these reasons, all the results presented in this document for the A64FX have been obtained using the

Clang mode of FCC, excluding the two Python kernels (NN-BASE and NN-VARIANT), whose libraries were compiled using GCC.

We compile all kernels with at least `-O2` optimization level and enable CPU-specific optimizations: `-march=armv8-a+sve` on the A64FX, `-mcpu=native` on Graviton3, and `-march=native` on SKX and Rome. Enabling CPU-specific optimizations in ABEA and POA resulted in incorrect executions, probably due to programming errors in the original source code. Therefore, such optimizations are not used for these two kernels.

After performing the appropriate modifications to the kernels so that all of them successfully execute on Arm, we applied further optimizations to some kernels to improve the performance obtained in this architecture. Such optimizations are described in the following subsections.

### 3.5.1. Exploiting Vectorization

Some kernels implement x86-vectorized versions of their most time-consuming parts. In particular, BSW and FAST-CHAIN include AVX2 and AVX512 versions of their critical functions using intrinsics. Similarly, POA implements SIMD versions of its code using AVX2 intrinsics and SIMD Everywhere (SIMDe) [262]. We have implemented SVE-intrinsics versions of FAST-CHAIN, BSW, and WFA, and a Neon-intrinsics version of BPM. At the time of writing this text, SIMDe does not fully support SVE, so we could not leverage POA's SIMDe version. Figure 3.2 shows the speedup of vectorized kernels over their scalar version on the experimental Arm and x86 systems using the large input of the kernels. Note that the SVE vector length of Graviton3 (256 bits) is half that of A64FX (512 bits), and therefore, the performance speedups of SVE kernels over their scalar versions are more modest in Graviton3.

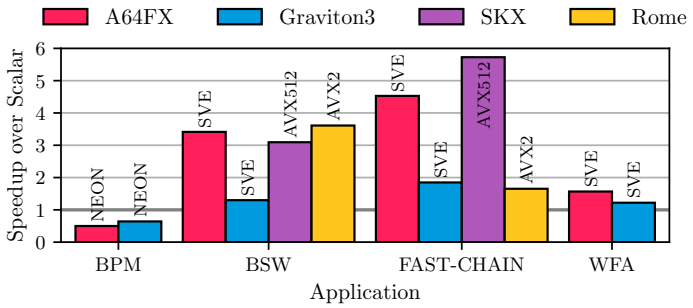



Figure 3.2.: Speedup of SIMD kernels over their scalar version on the Arm and x86 systems using the large inputs.

**Bit-Parallel Myers (BPM).** The core idea behind vectorizing BPM is to transform the alignment operations used to fill the dynamic programming table into simple machine-word operations. These simple operations are integer additions, bit shifts, and bitwise `ORs` and `ANDs`. This way, various dynamic programming cells are bit-packed within a machine word and its dependencies are encoded using bit-wise operations. In packed SIMD, vector

operations are performed in independent packets with a maximum width equal to the machine's maximum word width, rather than a whole bit vector (i.e., it is not possible to perform a 128-bit width operation in a 64-bit double word machine). For example, when performing a left-shift operation, the leftmost bit of each word is lost. However, in order to vectorize BPM, we would want this bit to be appended to the closest-left word, effectively performing a vector-width left-shift operation. To circumvent this problem, we must perform additional operations to manually carry that bit to the correct position. The number of additional operations required by this approach to work scales with the vector length. Thus, we decided to evaluate the potential of the vector version of BPM using the Neon vector extension (128-bit vectors). The vectorized loop executes 1.7× more instructions than the original but performs 2× fewer iterations.

On the A64FX, SIMD versions of simple instructions, like integer addition, were much more expensive than scalar ones. For example, a simple 64-bit addition takes one cycle, while a vector addition of two 64-bit words takes four cycles. This difference in latencies leads to a slowdown of 2×. Graviton3 has lower SIMD latencies. However, the increase in the number of instructions in the loop leads to a 30% performance loss. Since we did not gain any performance using the Neon version, it was discarded in favor of the original scalar code. We believe that an inter-sequence or coarse-grain approach (i.e., perform the sequence alignment of several sequences simultaneously) will deliver better performance since it simplifies the vectorization.

**Banded Smith-Waterman (BSW).** The SVE version of BSW [263] is a translation to Arm SVE-intrinsics of the x86-vector version found in BWA-MEM2, which groups the sequence alignment of multiple equal-length sequences via SIMD instructions (i.e., inter-sequence vectorization). The x86-intrinsics version of BSW relies on masks and blend operations to select valid entries from the vector registers. The SVE version takes advantage of SVE's predicate instructions to avoid the need for blend operations, effectively reducing the number of total instructions executed. BSW uses integers of 16 bits, allowing to process 32 elements per iteration using SVE-512 (A64FX) and 16 using SVE-256 (Graviton3).

The SVE version of BSW performs 3.4× and 1.3× faster than its scalar version on the A64FX and Graviton3, respectively.

**FAST-CHAIN.** Our SVE implementation of FAST-CHAIN is a translation to SVE intrinsics of the x86 version. The original x86 implementation of FAST-CHAIN executes its main loop scalar version (i.e., avoids executing the vectorized loop) when the number of iterations to perform is small. Additionally, as usual in x86 vector loops, it implements a loop-tail to process the remaining elements. Since SVE is vector-length agnostic, we could avoid most of the logic of the x86 version, reducing the number of performed instructions.

The x86 vectorized version of FAST-CHAIN uses 32-bit anchors. In some cases, 32-bit anchors are not sufficient, and this kernel generates incorrect results. To solve this, we have implemented 64 and 32-bit SVE versions of FAST-CHAIN. The 64-bit version always outputs correct results, but we have used the 32-bit implementation to compare against

the 32-bit x86 implementation.

Our SVE version of FAST-CHAIN runs 4.5× and 1.8× faster than its scalar version (CHAIN without heuristics) on the A64FX and Graviton3, respectively. Experimental results show that the performance of FAST-CHAIN compared to regular CHAIN greatly depends on the input used—the usage of heuristics may lead to performance variations based on the characteristics of the input. For instance, using GenArchBench large input, our SVE version of FAST-CHAIN is 2.2× faster than regular CHAIN on the A64FX, but it presents a 1.4× slowdown on Graviton3.

**WFA.** The Wavefront Alignment Algorithm consists of two operations: compute the next wavefront (`next` operation) and extend all the farthest-reaching points of a wavefront by exact matching characters from two strings (`extend` operation). The `next` operation can be automatically vectorized by the compiler due to its simple computational pattern. In contrast, the `extend` operator cannot be automatically vectorized as each diagonal requires an irregular amount of computations. To this end, we have vectorized the `extend` operation using a custom implementation relying on SVE intrinsics. Each vector lane extends a different diagonal, comparing four bases per lane until a mismatch is found. Because each diagonal requires a different number of character comparisons, some lanes can require more iterations than others. We tackle this problem by masking the lanes as they finish the extension process. This way, several diagonals are extended in parallel.

The SVE version of WFA delivers a 1.6× and 1.25× speedup over its scalar version on the A64FX and Graviton3, respectively.

### 3.5.2. Optimized Libraries

Many HPC kernels and tools rely on frequently used libraries. It is common for vendors, such as Arm, Fujitsu, or Intel, to develop optimized versions of widely used functions and libraries targeting their systems and architectures. For genome data analysis, some tools exploit neural networks (NNs) to improve the quality of their analysis and results. For the porting of GenArchBench kernels to Arm, we have tested different implementations of the libraries used by the NN-BASE and NN-VARIANT kernels.

**NN-BASE.** The NN-BASE kernel builds upon the PyTorch library [242]. On the A64FX we have used an optimized version of PyTorch for this specific CPU provided by Fujitsu. On Graviton3, we tried two different PyTorch backends: PyTorch compiled with OpenBLAS (recommended by Arm) and PyTorch compiled with oneDNN optimized with ACL (labelled as experimental). The Docker images with the two backends are available in [264]. The oneDNN-ACL backend performed 6.5× better than the OpenBLAS one, and therefore, we used it for our experiments. On SKX, we used an optimized version of PyTorch that exploits the AVX512 vector extension. On Rome, we used an optimized PyTorch version that supports the AVX2 vector extension available on the machine.

**NN-VARIANT.** The original NN-VARIANT kernel from GenomicsBench is based on Clair [210] variant caller. In turn, this variant caller relies on TensorFlow [243]. Clair

uses TensorFlow 1 while Fujitsu provides an optimized version of TensorFlow 2 for the A64FX. For that reason, we decided to use Clair3 [211] instead, an updated version of Clair that relies on TensorFlow 2. To execute using GenArchBench inputs, we used the Oxford Nanopore `r941_prom_hac_g360+g422` [245] pre-trained model from Clair3. On Graviton3, we tested three different TensorFlow backends: TensorFlow compiled with oneDNN optimized with ACL, using TensorFlow's Eigen thread-pool for parallelism (recommended by Arm); TensorFlow compiled with oneDNN optimized with ACL, using ACL's scheduler; and TensorFlow compiled with the Eigen backend. The Docker images with the three backends are available in [265]. The Eigen backend performed more than 1.6× better than the other, and therefore it was the one used to run our experiments. We used optimized TensorFlow versions on SKX and Rome capable of exploiting the AVX512 and AVX2 vector extensions.

### 3.5.3. Algorithmic and Code Optimizations

This section presents the algorithmic and code optimization we performed to improve the performance of FMI and KMER-CNT in Arm.

**FMI.** Our optimized version of FMI implements three optimizations proposed by Langarita et al. [263]. One of the most called functions in this kernel is `backwardExt`. To reduce the overhead of the calls, this function is always forced to be inlined. FMI uses the `builtin_popcount` function. This function counts the number of bits set to one in an integer. None of the tested compilers translates this function to SVE's population count instruction. Instead, they use bitwise operations and masks. To force exploiting SVE capabilities, all calls to `builtin_popcount` are replaced by SVE intrinsics. FMI performance is heavily affected by memory access latencies. To hide these latencies, the optimized version of FMI interleaves the execution of several sequences, effectively performing several memory accesses in parallel. By applying the three presented optimizations, we improved the kernel performance on both Arm machines by roughly 35%.

**KMER-CNT.** Our experimental evaluation shows that the performance of this kernel is heavily affected by thread migrations. To avoid thread migrations, we ported KMER-CNT from the Pthreads library to OpenMP and set `OMP_PROC_BIND` clause to `true` before executions. This change led to more than 4× speedups on both x86 machines when using all available cores. However, the performance of the Arm machines remained the same.

KMER-CNT relies on two global data structures to store the number of individual k-mers: an array of 4-bit counters and libcuckoo's [240] multi-threaded hash-map, which stores 64-bit counters. Each entry of the global array is an 8-bit atomic integer, which is split in half to create two 4-bit counters. The array counters are updated using an atomic compare-and-swap operation. Once the 4-bit counter of a k-mer saturates, the following increments are performed in the global hash map, also relying on atomic compare-and-swaps to update its counters.

Even by avoiding thread migrations, the scalability of the original kernel was poor on all the machines. It achieved a maximum of 7× and 5× vs. serial execution on the

A64FX (48 threads) and Graviton3 (64 threads), respectively. We decided to implement two new approaches to try to improve parallel performance.

The application divides the input between the available threads. Each thread iterates through the k-mers of its part of the input and increments the counter of the read k-mers in the global array or hash map. Since the input is read sequentially and there is almost no computation to perform, most of the execution time is spent accessing the global counters in mutual exclusion. To reduce contention and improve data locality, our first approach assigns part of the input to each thread, and all threads read the full input but only count part of the k-mers. This way, instead of having a single global array and hash map, each thread can have a smaller instance of the data structures and access them without contention.

The original version of the kernel uses compare-and-swap instead of fetch-and-add to update the counters because each 8-bit entry of the array stores two 4-bit counters. In order to limit memory usage, when the k-mer size is greater than 17, the kernel does not instantiate the global array, and all the counting takes place on the hash map. For the maximum allowed k-mer size (17), we require $2^{17}$ 4-bit entries in the global array, resulting in 8 GB of memory. Since we have more than enough memory in all systems, our second approach uses 8-bit instead of 4-bit counters, doubling the memory requirements. This enables fetch-and-add usage and reduces the number of accesses to the hash map.

The single-threaded execution time of the kernel did not change with any of the new versions. Our first approach (private structures) equally divides the possible k-mers between threads. However, some k-mers are more common in the input, causing load imbalance between threads resulting in even poorer scalability than the original kernel. The second approach (fetch-and-add) improves the kernel's scalability on the Arm and x86 machines: it runs 2.5×, 1.4×, 3× and 2.3× faster than the original version on the A64FX (48 threads), Graviton3 (64 threads), SKX (48 threads), and Rome (64 threads), respectively. Consequently, we used the fetch-and-add approach for the rest of the experiments on Arm and x86.

## 3.6. Porting GenArchBench Kernels to RISC-V

From the initial 13 genomic kernels included in the GenArchBench, we have successfully ported and executed 10 of them. Unfortunately, the lack of RISC-V support from the toolchain, libraries, or hardware has prevented the porting of some kernels. In particular, the lack of RISC-V support from the widely-used Pytorch and Tensorflow libraries prevented the porting of the DL-based kernels (NN-Base and NN-Variant). As will be discussed later, the FAST-CHAIN kernel has been successfully ported to RISC-V by emulating certain vector instructions. However, we have been unsuccessful in executing the vectorized version of the kernel on Allwinner, the only RISC-V machine with support for the vector extension. FAST-CHAIN is a variant of the CHAIN kernel that has been modified to eliminate all heuristics in order to enable vectorization. Accordingly, assessing its scalar performance is not relevant or informative. Thus, the performance evaluation does not include data on FAST-CHAIN.

As a first step, we focused on the scalar adaptation of the kernels with particular emphasis on validating the results using the same ground truth outputs generated for Arm. Then, we proceeded to perform executions on scalar cores and gather profiling information. Afterwards, we worked on extending the porting to vector instructions (RVV) to accelerate the kernels' execution on RVV-enabled processors. Compared to the scalar adaptation, the RVV extensions have little software and hardware support, limiting the development to simulated environments using experimental tools.

The RISC-V port inherits the algorithmic and code-level optimizations previously introduced for FMI and KMER-CNT for the Arm port of GenArchBench.

The remainder of this section discusses the main challenges and limitations encountered throughout the porting effort, including the current state of RISC-V software and toolchain support, the suitability of the RISC-V ISA for genomics workloads, and additional system-level and hardware-related constraints.

### 3.6.1. Software and Libraries Support

Due to its recent introduction, RISC-V-based platforms still lack widespread application and library support. As a result, most applications and libraries require recompilation and careful ad-hoc optimizations to exploit the capabilities of current RISC-V processors. In some cases, these libraries lack specific RISC-V support, hindering the efficient porting of high-performance applications. Notable examples in genome sequence data analysis include libraries for common file-format management (like HDF5 and HTSlib libraries) and Deep Learning (DL) frameworks (like Pytorch and TensorFlow).

Nowadays, Deep Learning (DL) libraries and tools have gained widespread adoption in multiple research areas and industrial applications. In particular, DL-based applications for genome analysis rely on the popular Pytorch and TensorFlow libraries. These DL frameworks depict many libraries' dependencies and lack precompiled versions for RISC-V. Moreover, due to the computationally intensive nature of these libraries, straightforward and non-optimized porting to RISC-V could result in significant execution inefficiencies. For that, many efforts from the HPC community aim to enable high-performance execution of DL-libraries on RISC-V.

### 3.6.2. Development Toolchain Support

To address the challenges of porting and optimising widely-used applications and libraries, it is paramount to have a mature and robust toolchain support. We have utilised several development tools for the genomics porting to RISC-V; some are stable and upstream available, others experimental and under active development.

For the porting development and evaluation, we performed multiple executions on commercial RISC-V scalar cores running standard Linux distributions; i.e., Ubuntu 20.04 (Unmatched), Fedora 33 (Allwinner), SUSE Linux Enterprise Server 12 SP2 (SKX). In addition, we utilised a QEMU virtual machine [266] to emulate a generic 64-bit RISC-V processor for development, profiling, and RVV emulation purposes. Note that the selected QEMU virtual machine does not natively support RVV extension. We utilised the

Vehave emulator [267] for vectorial kernels, which enabled us to debug and functionally validate the ported kernels and perform performance profiling and analysis on an emulated environment.

We mainly used GCC v10.3.0 and LLVM-based v12.0.0 compilers to compile libraries and tools. Both compilers have shown no difficulties in generating scalar binaries. However, RVV and auto-vectorization support are still under development for mainstream compilers. As a result, compiling vector code has been challenging and error-prone in many situations. For that reason, we resort to BSC-Clang[1], an extended Clang compiler from the EPI project supporting RVV extensions via auto-vectorization, pragma annotations, and C-Style intrinsics.

Currently, commonly used tools for profiling and debugging (Valgrind[2], GNU Debugger (GDB)[3], and perf[4]) offer limited support for RISC-V. For instance, the RISC-V support for disassembling instructions (using RISC-V mnemonics) is a crucial feature when developing applications for RISC-V platforms. Therefore, fully-compliant RISC-V disassembling in the GDB debugger is paramount, including instructions on the extended ISA. In that regard, we observe that the llvm-mc tool (Clang toolchain) is more mature, being able to convert opcodes to mnemonics and vice versa. Moreover, we noticed that the widely used Valgrind does not support RISC-V. Due to its importance and usefulness, there is an ongoing effort to port to RV64GC (https://github.com/petrpavlu/valgrind-riscv64).

Regarding profilers, like perf, we found that current RISC-V cores lack numerous standard counters mapped on other architectures. In the case of the RISC-V cores studied in this work, we were only able to measure cycles and instructions in Unmatched. The Allwinner does not have counters.

To provide a better understanding of the kernel's performance and guide the development and optimization on RISC-V platforms, profiling utilities like Paraver [268] have been promptly upgraded. Paraver can be used in conjunction with software emulators like Vehave to produce performance traces and analyse them using a versatile GUI. This tool allows analyzing performance metrics (like instructions and register usage), studying the memory access pattern, displaying annotated timelines with events, and more. These tools allowed us to inspect the critical regions and identify potential performance overheads. Moreover, we were able to investigate vectorial kernels without the need for a real vector machine.

### 3.6.3. ISA Support

Ratified elemental extensions of the RISC-V architecture, such as the multiplication and division (M), floating-point (F), and double-precision (D) extensions, have been available for more than a decade and are generally adopted by commercial hardware and mainstream compilers. Most kernels ported in this project rely solely on these basic extensions, allowing their straightforward compilation and execution on any compliant

[1] https://repo.hca.bsc.es/gitlab/rferrer/llvm-epi
[2] https://github.com/ivosh/valgrind
[3] https://github.com/riscvarchive/riscv-binutils-gdb
[4] https://perfwiki.github.io/main/

RISC-V core. However, some RISC-V cores lack support for specific instructions from the specification. For example, the Allwinner machine does not support the optional `fence.tso` instruction used by the POA kernel.

**RVV Extension.** Porting the genomic kernels to the RISC-V vector (RVV) extension has proven challenging in many cases. We found that only one kernel could be partially vectorized automatically by the compiler (i.e., WFA). For the kernels BSW and CHAIN, we develop a vectorial implementation based on RVV-intrinsics. For the remaining kernels, we found that the current RVV extension lacks essential instructions used by genomic kernels.

In particular, we found that, unlike AVX-512 (x86_64) or SVE (Arm) ISAs, the well-known count leading zeros (`clz`) and count trailing zeros (`ctz`) instructions lack a vector counterpart in the RVV specification, useful for the CHAIN and WFA kernels. The scalar versions of `clz` and `ctz` are part of the ratified bit manipulation extension (B), and their vector versions are currently part of the cryptography extension proposal. Beyond genomic kernels, `clz` and `ctz` are useful for multiple applications, including square root computation, Huffman decoding, binary logarithm computation, and hash table indexing. Although their functionality can be emulated by executing other arithmetic instructions, the overall instruction overhead has a non-negligible toll on performance.

Similarly, the RVV specification lacks specific support to reverse the order of the elements within a vector register, a helpful operation for some genomic kernels. Vector reverse can be conveniently implemented using the `vid` instruction to compute an index vector and the `vrgather` instruction to generate the final result. However, some additional instructions are needed to rearrange the indices produced by `vid`. We claim that a more powerful version of the `vid` instruction is possible. If RVV supported an instruction similar to `vid`, allowing to specify the starting, ending, and increment values (akin to the functionality of Linux's `seq` command), we could reverse a vector using fewer instructions. Additionally, this new instruction would be useful in a wide range of scenarios, since `vid` usually requires being paired with additional instructions to compute the desired vector of indices.

In practice, we have encountered issues executing vector kernels on Allwinner, the only RVV-enabled core. Although this processor implements the RVV extension v0.7, it may not execute correctly in certain scenarios. For example, it throws an illegal instruction error when executing the `vmv` instruction, which is used on the kernels BSW, CHAIN, and WFA. Upon further investigation, we found that these kernels used a 64-bit element width (SEW=64), and Allwinner does not support RVV instructions for 64-bit element widths, resulting in an illegal instruction exception. Although the kernels could be adapted to use a smaller element width, the compiler emits a 64-bit element width by default. Another example is the execution of RVV gather instructions, required by some kernels like WFA, which also result in a run-time error when executed on Allwinner.

At the time of this porting effort, the RVV specification and cores implementing it were still in the process of being refined and made more robust with future developments. A clear example of this is the handling of masked instructions: in RVV v0.7, the mask

operand must reside in register v0, even though a mask can logically be stored in any vector register. Moving a mask from an arbitrary register to v0 required multiple instructions in this version. With the introduction of RVV 1.0, however, the same operation can be performed in a single instruction. While the scalar RISC-V specification and cores are relatively mature and stable, the RVV specification and hardware continue to evolve rapidly to reach the performance and usability levels of established vector extensions such as AVX and SVE.

**Platform Limitations.** In the exploration of the maturity and limitations of RISC-V processors to execute genomic applications, we have encountered other technical challenges. Most notably, genomic kernels usually process large amounts of data, requiring large memory footprints. Considering the memory available on the evaluated RISC-V platforms, we found that real executions at the genome scale are not feasible as of today. Specifically, Allwinner has only 2 GB of memory. Considering that representative genomics datasets and databases are usually of the order of tens of gigabytes, we were forced to use reduced inputs for testing the execution of the ported kernels. As a result, for the most memory-demanding kernels—FMI, KMER-CNT, and POA—we created custom "tiny inputs" with reduced memory footprints to ensure successful execution on the RISC-V platforms.

## 3.7. Evaluation in Arm

This section presents a detailed performance characterization of the kernels in the Arm and x86 systems. For all of the studies presented, we have annotated the code of the kernels to define their region of interest, i.e., we only study the part of the kernels dedicated to meaningful computation. All the results shown in this section have been computed using the large input of each kernel. We noted minimal variation between executions of the kernels, with a maximum relative standard deviation of 5% observed across 10 repetitions of the experiments. Consequently, we showcase the results based on a single execution in the figures. While executing DBG with high thread counts in Graviton3, outlier execution times occurred approximately 10% of the time. In the case of DBG in Graviton3, we selectively present results from an inlier execution.

### 3.7.1. Single-Threaded Performance

The top plot of Figure 3.3 shows the single-threaded execution time of each kernel on Arm and x86 machines. The results are normalized to the performance on the A64FX.

The A64FX features significantly fewer out-of-order resources, a smaller memory hierarchy, and higher memory latencies than the rest of the systems. On average, the former is 2.4×, 1.8×, and 1.7× slower than Graviton3, SKX, and Rome on single-threaded executions, respectively. Exploiting the SVE capabilities of the A64FX helps to reduce this slowdown. SVE vectorized kernels (BSW, FAST-CHAIN, and WFA) present better-than-average performance on the A64FX: BSW performance is similar to that exhibited on Graviton3 and only 17% worse than the performance on the x86 machines, FAST-CHAIN

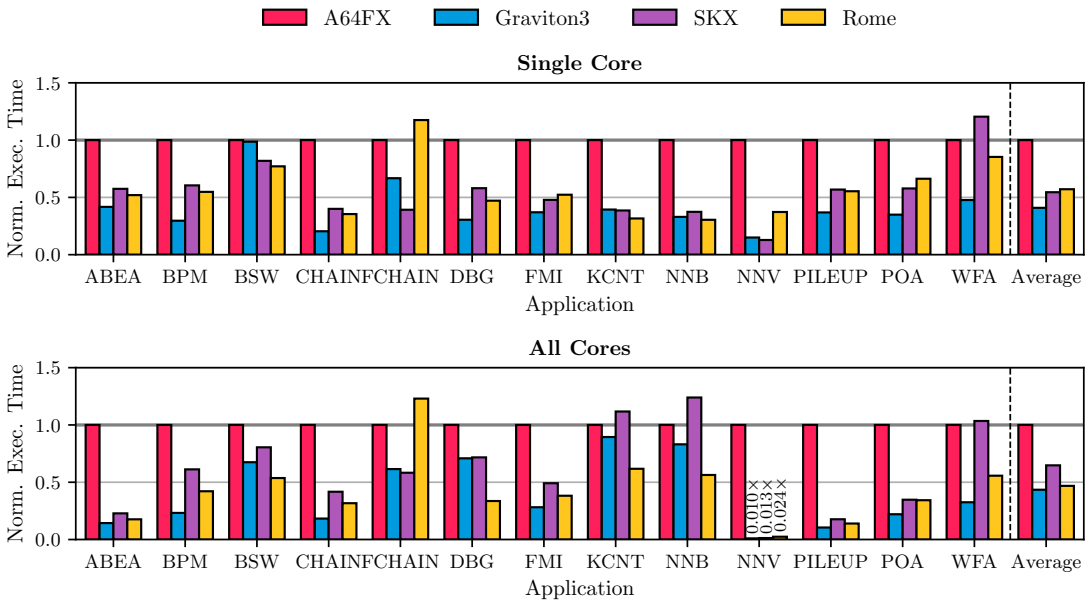



Figure 3.3.: Single-core (top) and multi-core (bottom) execution time of GenArchBench kernels on the Arm and x86 systems. Multi-core results correspond to executions using all available cores on each machine: 48 threads on the A64FX and SKX and 64 threads on Graviton3 and Rome. The results are normalized to the performance on the A64FX using one core (top) and 48 cores (bottom). FCHAIN, KCNT, NNB, and NNV are the abbreviations of FAST-CHAIN, KMER-CNT, NN-BASE, and NN-VARIANT, respectively. NN-VARIANT is not taken into consideration for the average in the multi-core plot.

performs better than on Rome, and WFA performs better than on SKX. Note that BSW and FAST-CHAIN exploit AVX512 on SKX while they leverage AVX2 on Rome, and that WFA is not vectorized on the x86 machines. The deep-learning kernels (NN-BASE and NN-VARIANT) are the worst-performing on the A64FX.

Graviton3 performs exceptionally well in single-threaded executions. On average, it presents 2.44×, 1.33×, and 1.39× performance speedups with respect to the A64FX, SKX, and Rome, respectively. FAST-CHAIN performance on Graviton3 is 1.8× better than on Rome (AVX2) but 70% worse than on SKX since it exploits AVX512 (512 bits) on that machine. WFA runs 2.5× and 1.8× faster on Graviton3 than on SKX and Rome, respectively. In contrast to the A64FX, the deep-learning kernels (NN-BASE and NN-VARIANT) deliver good performance on Graviton3, showing speedups of between 3.1-6.2× compared to the A64FX.

### 3.7.2. Multi-Threaded Performance

We evaluate the parallel performance of GenArchBench kernels using different thread counts: 2, 8, 24, 48, and 64. The A64FX and SKX implement 48 cores. Hence, executions with more than 48 threads have only been performed on Graviton3 and Rome. Controlling thread affinity was mandatory in our experiments to achieve good parallel performance on the machines, especially on the A64FX. For most kernels, all the executions were performed by binding threads to cores. ABEA, NN-BASE, and NN-VARIANT do not allow full thread affinity control. Therefore, thread migrations can occur in these three kernels.

Figure 3.4 shows the speedup over serial execution achieved by the kernels on the experimental setup using the previously presented thread counts. Additionally, the bottom plot of Figure 3.3 compares the performance obtained using all available cores on each machine: 48 threads on the A64FX and SKX and 64 threads on Graviton3 and Rome. The parallel performance of NN-VARIANT on the A64FX is extremely poor; therefore, it is not considered for the average calculation. The results are normalized to the performance on the A64FX using 48 threads.

All GenArchBench kernels exploit coarse-grain parallelism. Most of them, except for KMER-CNT, present little to no interaction between threads. It can be seen that some kernels achieve near-perfect scaling on all machines. This is the case for BPM, BSW, CHAIN, FMI, and WFA. For this set of kernels, the normalized plots using 1 thread and all available cores are similar. It is important to note that Graviton3 and Rome show some performance gains compared to the other two machines since the number of available cores is higher.

In ABEA and PILEUP, the primary thread reads the full input and splits it into smaller chunks that are dynamically assigned to idle threads. This is the same scheduling implemented by other kernels, such as BSW. However, the chunks used in ABEA and PILEUP are significantly bigger, leading to load imbalance. For ABEA, the smallest grain size assigned to a thread is a whole read. Therefore, to improve scalability, we would need to change the granularity used by the kernel. PILEUP implements a default chunk size of 100 kbp. Dynamically choosing the chunk size based on the input would reduce load imbalance. The parallel performance of both kernels should also improve by using larger inputs. In both kernels, the A64FX presents poorer scalability than the other three machines, further increasing their performance difference compared to single-threaded executions.

DBG also implements dynamic scheduling and shows good scalability and load balance on the A64FX, SKX, and Rome. Runs of DBG on Graviton3 using more than 8 threads present high variability in contrast to the other machines, resulting in poor scalability in most cases. Due to this behavior, DBG performance on Graviton3, when using all cores, is similar to that of SKX.

In KMER-CNT, all threads continuously perform random memory write accesses using atomic operations, resulting in high memory contention and poor scalability. For this kernel, the A64FX presents the best parallel scalability: a maximum speedup of 16× with respect to single-threaded executions vs. a maximum of 5× on the other machines. This

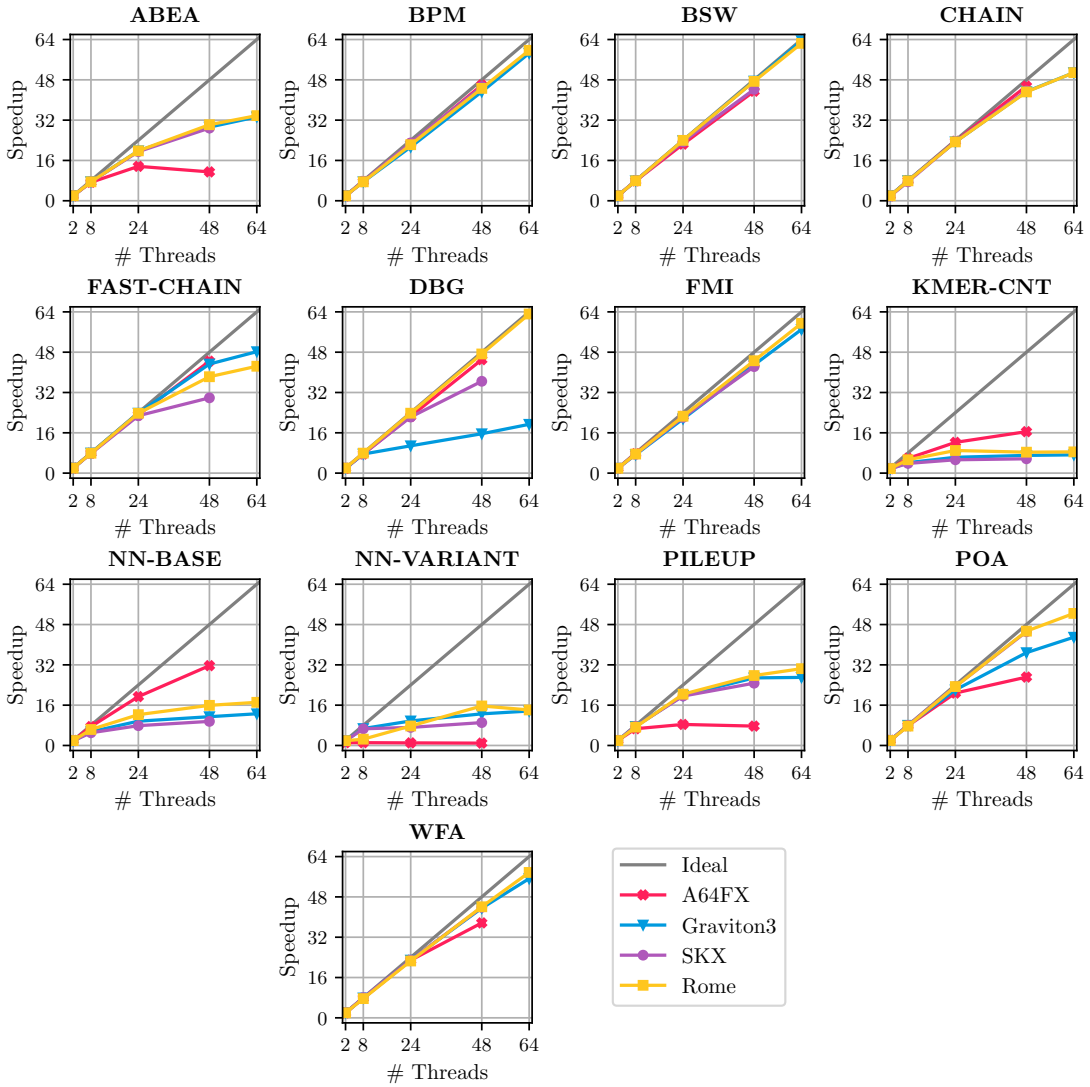



Figure 3.4.: Speedup over serial execution of GenArchBench kernels on the Arm and x86 systems. We show the achieved speedup using different thread counts: 2, 8, 24, 48, and 64. The A64FX and SKX 64-thread points are not shown in the figure, since those machines only implement 48 cores.

results in similar performance between the A64FX, Graviton3, and SKX when using all available cores.

NN-BASE does not implement any high-level parallelism. It relies on PyTorch multi-threading, which allows using intra-op parallelism (via math libraries like Intel MKL) and inter-op parallelism. This approach works relatively well on the A64FX but offers poor scalability on the rest of the systems.

NN-VARIANT presents a significant load imbalance even with low thread counts. In order to improve this, we tried two different scheduling policies: to assign each core a similar-sized chunk of the input and to dynamically assign small chunks to available threads. In both cases, the time needed to process the chunks was unpredictable. Additionally, NN-VARIANT relies on TensorFlow, making it difficult to control the number of threads used. Besides the kernel's high-level parallelism, TensorFlow uses between 1 and 4 threads during the model inference step, degrading parallel performance when NN-VARIANT uses more than 1/4 of the available threads. The performance of NN-VARIANT on the A64FX does not improve with any number of threads, resulting in extremely poor parallel performance compared to the other systems.

### 3.7.3. Instruction Mix Comparison

Table 3.3.: Instruction mix mapping from Fujitsu Advanced Performance Profiler (FAPP) to our DynamoRIO `opcode_mix` tool categories.

| **FAPP Category** | **DynamoRIO Category** |
|---|---|
| Load (memory) | Load |
| Store (memory) | Store |
| Floating point move<br>Element manipulated<br>Register manipulated | Register Move/Manipulation |
| Floating-point (arithmetic)<br>Floating-point conversion | Floating-point |
| SIMD integer<br>Predicated SIMD integer<br>Other instructions | Integer/Logical |
| Branch | Branch |
| Prefetch<br>DCZVA<br>Cryptographic<br>MOVPRFX<br>Math functional | Other |

An application's instruction mix determines which processor pipelines and functional units are the most used during its execution. To obtain it, we used the instruction mix

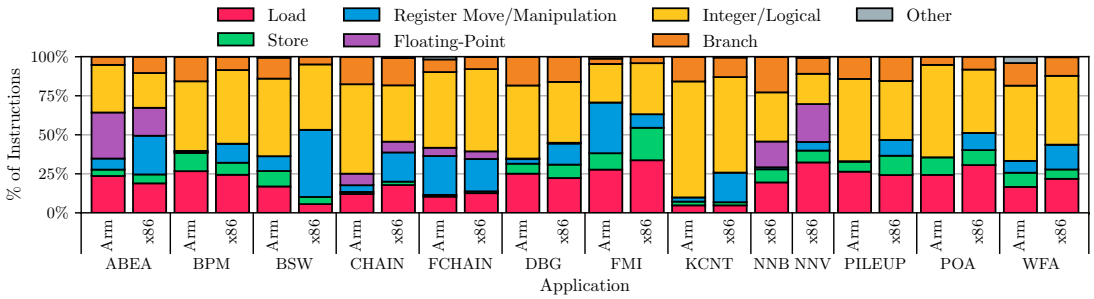



Figure 3.5.: Instruction mix of GenArchBench kernels on Arm (A64FX) and x86 (SKX). FCHAIN, KCNT, NNB, and NNV are the abbreviations of FAST-CHAIN, KMER-CNT, NN-BASE, and NN-VARIANT, respectively.

report offered by the Fujitsu Advanced Performance Profiler (FAPP) on Arm (A64FX) and a modified version of DynamoRIO's `opcode_mix` tool [269] on x86 (SKX) that divides the executed instructions into different categories[5]. The instruction mix offered by both tools is significantly different, so we designed a mapping from FAPP categories to the categories defined in our modified DynamoRIO (see Table 3.3).

Figure 3.5 shows the instruction mix of GenArchBench kernels in Arm (A64FX) and in x86 (SKX). The Fujitsu Advanced Performance Profiler does not allow the creation of child processes. For this reason, we have not been able to compute the Arm instruction mix of the Python kernels (NN-BASE and NN-VARIANT). The `Register Move/Manipulation` category includes any data movement between registers or manipulation of the contents of a register without performing any arithmetic operation (like the `setz` instruction of x86). The `Other` category includes prefetching, cryptographic, string, and special instructions (such as the `DCZVA` and `MOVPRFX` instructions of Arm or the `RDRAND` instruction of x86).

ABEA and the deep-learning kernels (NN-BASE and NN-VARIANT) are the only kernels that perform a significant number of floating-point operations. As explained before, we lack the tools to compute the instruction mix of NN-BASE and NN-VARIANT on Arm, but as for the rest of the kernels, we expect it to be similar on both architectures. CHAIN and FAST-CHAIN also perform floating-point operations but are mainly dominated by integer, logical, and register move/manipulation instructions. FMI mainly performs memory operations and is heavy on register move/manipulation instructions on Arm. On the other hand, KMER-CNT performs nearly no data movements (although memory accesses in this kernel are expensive, as shown in Section 3.7.4). The rest of the kernels mostly execute integer and logical instructions, and between 30% and 40% of data movement operations.

### 3.7.4. Microarchitecture Bottleneck Analysis

We have studied the microarchitecture bottlenecks of each application using FAPP on the A64FX, Perf on Graviton3 and Rome, and Intel VTune Profiler on SKX. We have not been

[5] https://github.com/LorienLV/dynamorio

Table 3.4.: Micro-architecture bottlenecks mappings from each system's profiler categories to our aggregated categories.

| **Aggregated Category** | **A64FX (FAPP)** | **Graviton3 (Perf)** | **SKX (VTune)** | **Rome (Perf)** |
|---|---|---|---|---|
| Back-End Stalls | Memory Stalls + Core Stalls | stall_backend / cpu_cycles | Memory Stalls + Core Stalls | stalled-cycles-backend / cycles |
| Memory Stalls | (Prefetch port busy wait + Operation wait + Store port busy wait) / Total | stall_backend_mem / cpu_cycles | (CYCLE_ACTIVITY.STALLS_MEM_ANY + EXE_ACTIVITY.BOUND_ON_STORES) / CPU_CLK_UNHALTED.THREAD | — |
| Core Stalls | Memory & cache access wait / Total Cycles | — | 1 - Front-End Stalls - Memory Stalls - (UOPS_ISSUED / SLOTS + INT_MISC.RECOVERY_CYCLES / CPU_CLK_UNHALTED.THREAD) | — |
| Front-End Stalls | Instruction fetch wait / Total Cycles | stall_frontend / cpu_cycles | IDQ_UOPS_NOT_DELIVERED.CORE / SLOTS | stalled-cycles-frontend / cycles |
| Bad Speculation Stalls | Branch instruction wait | 1 - (Back-End Stalls + Front-End Stalls + Useful Work) | (UOPS_ISSUED.ANY - UOPS_RETIRED.RETIRE_SLOTS) / SLOTS + INT_MISC.RECOVERY_CYCLES / CPU_CLK_UNHALTED.THREAD | (uops dispatched - uops retired) / SLOTS |
| Useful Work | (1/2/3/4 instruction commit + Other instruction commit) / Total Cycles | (cpu_cycles - stall) / cpu_cycles | (CPU_CLK_UNHALTED.THREAD - UOPS_RETIRED.STALL_CYCLES) / CPU_CLK_UNHALTED.THREAD | 1 - (Back-End Stalls + Front-End Stalls + Bad Speculation Stalls) |
| Other | (Other wait + Barrier synchronization wait) / Total Cycles | — | — | — |

able to compute the microarchitectural bottlenecks of Python kernels (NN-BASE and NN-VARIANT) on the A64FX, as FAPP does not allow the creation of child processes.

Analogous to the instruction mix, we have designed a mapping from the different profilers and systems metrics to our microarchitecture bottleneck categories: `Back-End stalls`, `Front-End Stalls`, `Bad Speculation Stalls`, `Useful Work`, and `Other`. We further split the category `Back-End stalls` into `Memory Stalls` and `Core stalls`. The `Memory Stalls` category includes stalls due to main memory and caches. Unfortunately, we could not compute `Memory Stalls` or `Core Stalls` in Rome because it does not implement the required performance counters. The same problem occurs on Graviton3, where we could not measure `Core stalls`, and the `Memory Stalls` category only includes stalls due to main memory.

Table 3.4 shows the mappings, utilizing the names of the metrics as defined in the profiler of each machine: FAPP detailed report categories on the A64FX, Perf events on Graviton3 and Rome, as shown by the `perf list` command, and Intel VTune hardware events on SKX. The `SLOTS` metric in SKX and Rome corresponds to the number of cycles executed multiplied by the commit width of the processor, that is, `CPU_CLK_UNHALTED.THREAD` $\times$ 4 in SKX and `cycles` $\times$ 8 in Rome. In order to measure `Useful Work` in SKX similarly to the other machines, we use the `UOPS_RETIRED.STALL CYCLES` hardware counter, instead of `UOPS_RETIRED.RETIRE_SLOTS`, which makes the total percentage of executed cycles greater than 100%. To circumvent this, we normalize the other categories (`Memory Stalls`, `Core Stalls`, `Front-End Stalls`, `Bad Speculation Stalls`) with respect to `Useful Work`.

Most hardware events in SKX measure slots instead of cycles. Although we have homogenized the formulas used in each machine as much as possible, it is important to note that performance counters are not standardized between machines, let alone architectures, so similar metrics may count moderately different events on different machines. Comparisons between counters of different machines must be seen as rough estimations of reality.

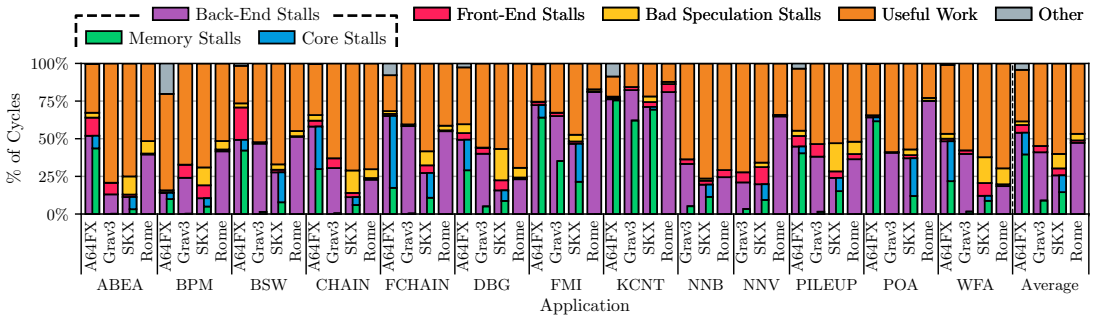



Figure 3.6.: Microarchitecture bottlenecks of GenArchBench kernels on the Arm and x86 systems. FCHAIN, KCNT, NNB, and NNV are the abbreviations of FAST-CHAIN, KMER-CNT, NN-BASE, and NN-VARIANT, respectively. Grav3 is the abbreviation of Graviton3.

Figure 3.6 shows the microarchitecture bottlenecks of GenArchBench kernels on the experimental setup. On average, there are significantly more memory stalls on the A64FX than on the rest of the machines. While the A64FX has the highest memory bandwidth, it implements a small memory hierarchy and suffers from high memory latencies. On the other hand, the bottlenecks on Graviton3 are more similar to those shown by the x86 machines. ABEA, BSW, DBG, and POA suffer from a high percentage of memory stalls on the A64FX compared to the other machines. On the other hand, FMI and KMER-CNT are mainly memory-bound on all machines. These two kernels mainly perform random memory accesses and do not exploit temporal or spatial locality. Thus, they are highly impacted by memory latencies. Kernels such as DBG, PILEUP, or WFA suffer from a high count of stalls due to bad speculation on x86 systems, especially on SKX, while the bottleneck on Arm is much smaller. Although this can very well be due to how these stalls are counted on different machines, we believe that Arm-predicted instructions play an important role in this metric. The number of cycles NN-VARIANT dedicates to useful work on Rome is small compared to the other machines, explaining its poor performance compared to other kernels on this machine. On the contrary, the `Useful Work` metric percentage of NN-BASE is considerable on all machines.

### 3.7.5. Energy Consumption

Figure 3.7 shows the energy-to-solution of GenArchBench kernels on the A64FX, SKX, and Rome using one (top plot) and all available cores (bottom plot) on each machine. Graviton3 is not included in the figure as it does not expose its energy consumption. Additionally, we can not measure the energy consumption of Rome's DRAM. However, since Rome and SKX use the same DRAM technology, we have added a 12% extra energy consumption to Rome's measurements based on the energy consumption of SKX's DRAM (expressed as error bars in Figure 3.7). Energy consumption was measured using the Fujitsu power API [270] on the A64FX, and an in-house library[6] based on the Running Average Power Limit (RAPL) [271] on SKX and Rome. Since there is a large difference between machines, the results of NN-VARIANT were not taken into consideration for the average calculation.

The maximum power consumption of the A64FX (120 W) is substantially lower than that of the x86 systems: 2×150 W on SKX and 225 W on Rome. However, SKX and Rome are capable of dynamically scaling their frequency depending on the load of the system (CPU throttling), while the A64FX constantly consumes power near its peak, even on low usage.

The results shown in the top plot of Figure 3.7 are highly similar to those presented in the top plot of Figure 3.3. On average, the A64FX consumes 1.7× more than SKX and 2.6× more than Rome in single-threaded executions. Kernels with good single-threaded performance, such as BSW or WFA, show better-than-average energy-to-solution results on the A64FX.

The previous picture changes when using all available cores on each machine (bottom plot

[6] https://github.com/LorienLV/rapl_stopwatch

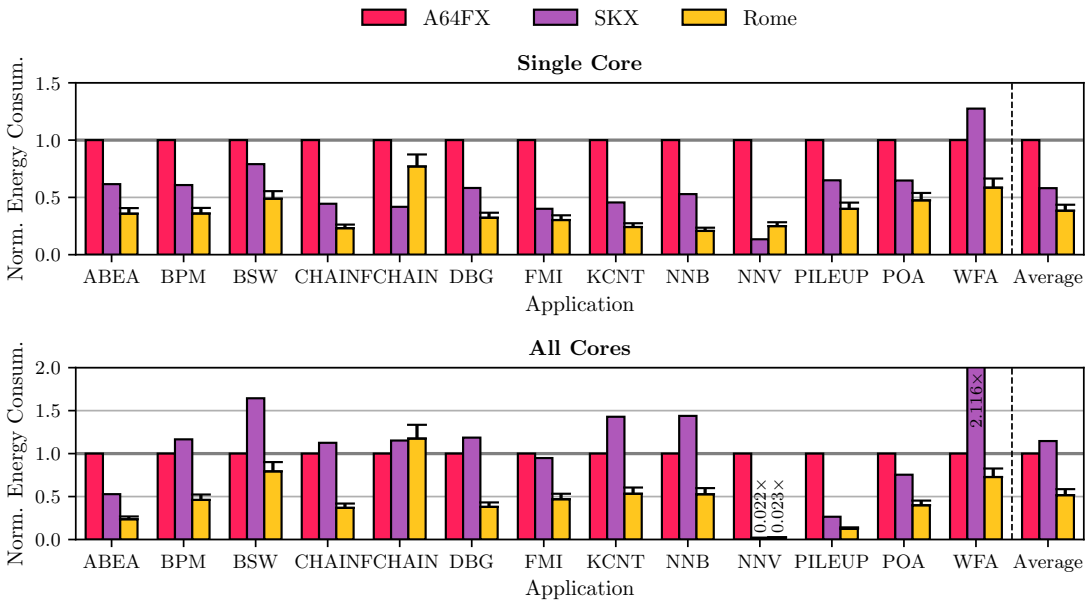



Figure 3.7.: Single-core (top) and multi-core (bottom) energy-to-solution of GenArchBench kernels on the Arm and x86 machines. Multi-core results correspond to executions using all available cores on each machine: 48 threads on the A64FX and SKX and 64 threads on Rome. The results are normalized to the performance on the A64FX using one core (top) and 48 cores (bottom). FCHAIN, KCNT, NNB, and NNV are the abbreviations of FAST-CHAIN, KMER-CNT, NN-BASE, and NN-VARIANT, respectively. NN-VARIANT is not taken into consideration for the average in the multi-core plot. Graviton3 is not included in this plot since it does not expose its energy consumption.

of Figure 3.7). In this scenario, all the machines are near their peak power consumption. The A64FX consumes less energy than SKX in 8 out of 13 kernels (12% less energy consumption on average), while Rome is again the most energy efficient when executing most kernels (1.9× less energy consumption than the A64FX).

### 3.7.6. Evaluation of a Real Genomics Tool

Finally, we evaluate the performance of a real genomics tool that uses some of the kernels ported to Arm presented in this chapter. For this matter, we use BWA-MEM2 [197], a read mapping tool (Figure 3.1-3.a.1) that employs the FMI kernel for the seed stage (Figure 3.1-3.a.1.1) and the BSW kernel for the extend stage (Figure 3.1-3.a.1.3). We use the optimized versions of the kernels, which have been presented and evaluated in this work. In our tests, BWA and FMI represent 45% and 34% of the total execution time of BWA-MEM2, respectively.

For our study, we have used three input datasets, each one with 1.25M reads of different

lengths obtained from real sequencing machines: D3 [272], D4 [273], and D5 [274]. These reads are aligned against the human genome [275].

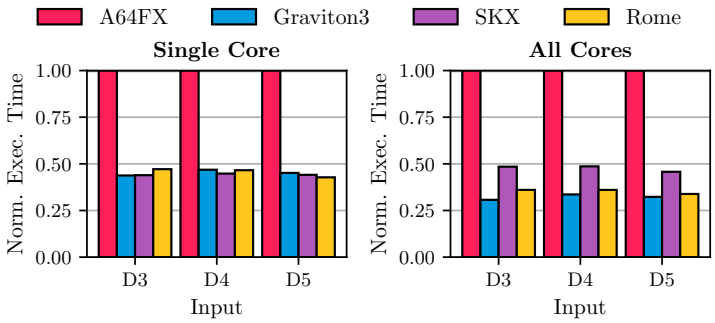



Figure 3.8.: Execution time of BWA-MEM2 using one core (left) and all available cores in the Arm and x86 machines (right). We show results using three inputs: D3, D4, and D5. The results are normalized to the performance on the A64FX using one core (left) and 48 cores (right).

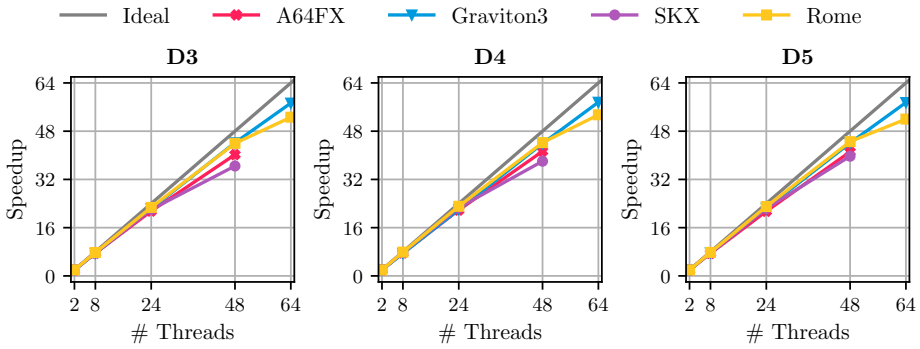



Figure 3.9.: Speedup over serial execution of BWA-MEM2 on the Arm and x86 machines for three inputs: D3, D4 and D5. We show the achieved speedup using different thread counts: 2, 8, 24, 48, and 64. The figure does not show the A64FX and SKX 64-thread points since those machines only implement 48 cores.

Figure 3.8 shows the performance of BWA-MEM2 using one (left) and all available cores (right) on the Arm and x86 machines. On single-threaded executions, Graviton3, SKX, and Rome perform similarly, showing over 2× speedups over the A64FX. When using all available cores, Graviton3 performs 10% better than Rome, and more than 30% better than SKX (as can be expected due to the difference in cores). The A64FX, on the other hand, shows 2× slowdowns compared to SKX. Both single-threaded and multi-threaded results can be easily correlated with the ones shown in Figure 3.3. In the results from GenArchBench, BSW on Graviton3 showed slowdowns with respect to the x86 systems. However, FMI performed better on Graviton3, and the kernel represents

a higher percentage of the total execution time of BWA-MEM2. Similarly, the A64FX delivered good performance when executing BSW, but severe slowdowns with respect to the other machines when executing FMI.

We also evaluate the parallel scalability of BWA-MEM2 using 2, 8, 24, 48, and 64 threads. As presented previously (see Figure 3.4), BSW and FMI achieved perfect scaling when executed standalone as part of GenArchBench. As we could anticipate, BWA-MEM2 also showed excellent parallel scalability, as presented in Figure 3.9.

## 3.8. Evaluation in RISC-V

In this section, we evaluate the performance of the two RISC-V machines, Unmatched and Allwinner, compared to the SKX system when executing GenArchBench. First, we present the execution time results for scalar and single-threaded executions performed in the RISC-V and SKX machines. Next, we present a scalability analysis for scalar multi-threaded executions on the SiFive Unmatched machine. Due to the challenges and limitations discussed in previous sections, we have not included results for the RVV versions of the kernels.

This evaluation uses the small input datasets provided with GenArchBench. The only exceptions are FMI, KCNT, and POA, for which we use the generated tiny inputs to reduce memory footprint and enable execution on RISC-V machines.

### 3.8.1. Single-Threaded Scalar Execution

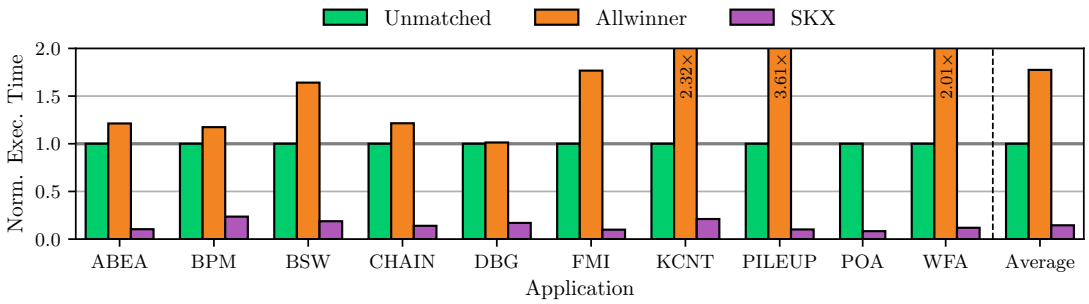



Figure 3.10.: Single-threaded and scalar execution time of GenArchBench kernels in the RISC-V machines and SKX. The results are normalized to the performance on the Unmatched machine.

Figure 3.10 shows the execution time of GenArchBench kernels when performing scalar and single-threaded executions on the two RISC-V machines, Unmatched and Allwinner, and the reference x86_64 machine, SKX. The results have been normalized to the performance on Unmatched. We selected the Unmatched for normalization reference as it is the only RISC-V machine that can successfully execute the scalar version of all the kernels.

We observe that, compared to the SKX, Unmatched shows slowdowns between 3.8× and 12×, for KCNT and POA, respectively. The average slowdown of Unmatched compared to SKX is 6.6×. This is an expected result considering the characteristics of both processors. In particular, SKX works at a higher frequency (1.75×), has a higher superscalar degree (2×), and implements an out-of-order pipeline while Unmatched implements an in-order core. In addition, SKX has significantly larger L2 and L3 caches (16×) and includes hardware prefetchers in L1 and L2, while Unmatched does not implement any prefetcher. These features of their memory hierarchies can make a big difference in some applications.

Moreover, we noticed that Allwinner is consistently slower than Unmatched for all kernels, showing slowdowns ranging between 1.01× and 3.6×. These results are consistent with the characteristics of the machines, as Unmatched operates at a higher frequency (×1.2) and implements a superscalar core (×2). Furthermore, the memory hierarchy may significantly affect the performance of some kernels, considering that Allwinner has only one cache level, divided into L1I and L1D (32 KB each).

It would be noteworthy to perform a more detailed analysis to understand the influence of the memory hierarchy on the results. However, this analysis is not possible due to the lack of the necessary hardware counters in the RISC-V systems.

### 3.8.2. Multi-Threaded Scalar Execution

The Unmatched processor includes 4 cores, allowing multi-threaded executions using up to 4 threads. Figure 3.11 shows the speedup of each kernel executing with 2 and 4 threads compared to single-threaded execution in the Unmatched machine. Most kernels exhibit near-perfect scalability as the number of threads increases. In particular, 9 of the 10 kernels achieve speedups above 1.85×, using two threads, and above 3.5×, using four threads. The WFA kernel presents the worst scaling behavior, showing speedups of 1.7× and 2.8× for 2 and 4 threads, respectively.

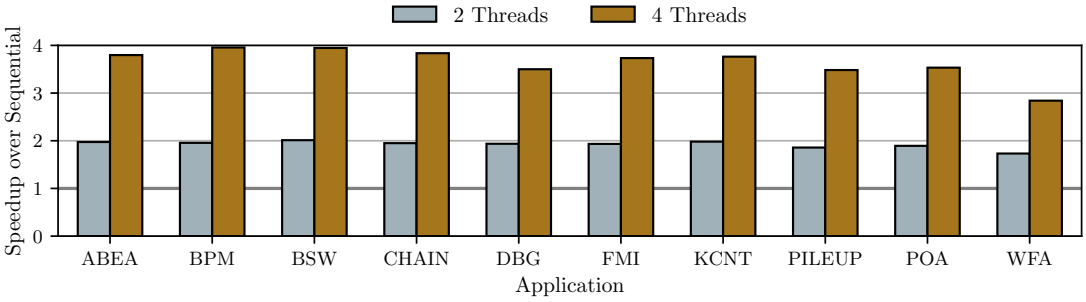



Figure 3.11.: Speedup over serial execution of GenArchBench kernels on the Unmatched machine. We show the achieved speedup using 2 and 4 threads.

## 3.9. Related Work

Outside the field of genomics and bioinformatics, there are many examples of domain-specific benchmark suites. Some widely known examples include the LINPACK benchmarks [276] for linear algebra, the GAP benchmark suite [277] for graph processing, and BigDataBench [278] for big data.

Focused on genomics, GenomicsBench [224], on which this work is based, is a benchmark suite that includes 12 computationally demanding kernels from common steps in genome data analysis. GenomicsBench includes CPU and GPU kernels, targeting the x86 HPC and NVIDIA GPU ecosystem. GenarchBench includes 10 CPU kernels of this suite and three additional ones. The GenArchBench kernels have been ported to the Arm and RISC-V architectures, and most of them implement optimizations targeting Arm. Additionally, GenArchBench includes automatic regression tests to verify the correctness of the execution of its kernels.

The BioPerf [279] benchmark suite, released in 2005, compiles DNA and protein analysis applications, such as Blast [203] or FASTA [280], two of the most widely used aligners at the time. Moreover, it includes three inputs per kernel; pre-compiled binaries for x86, PowerPC, and Alpha; execution scripts; and simulation points (Simpoints) to simulate the execution of the kernels. Similarly, BioBench [281], also released in 2005, offers some of BioPerf's benchmarks and presents a performance characterization. Recently, some of the kernels included in BioBench were updated in BioBench2 [282].

Furthermore, many publications analyze state-of-the-art genomics workloads, algorithms, and tools. Jason et al. [283] present a review of the steps involved in genome assembly. Similarly, Mohammed et al. [45] focus on the genome resequencing pipeline. Most notably, one of the main bottlenecks in genome sequence analysis is read mapping. As a result, there are many works discussing the algorithms used for this process and its acceleration on HPC processors [227], [284]–[286].

Recently, there have been multiple efforts to accelerate widely-used genome analysis kernels exploiting novel hardware solutions [287]–[292]. Tony et al. [293] provide a comprehensive review of state-of-the-art hardware acceleration techniques for genomics.

Regarding the RISC-V toolchain, we can find studies comparing different RISC-V compilers, such as the work by Poorhosseini et al. [294], which examines the performance and size of binaries generated by GCC and LLVM, and compares the performance of the compilers themselves. Other studies, such as the one by Adit and Sampson [295], have focused on compiler auto-vectorization. We observe there is an increasing interest in extending the RISC-V support to compilers, such as the JIT compiler [296].

Moreover, we find surveys on the RISC-V specifications [297] and many proposals to speed up critical kernels in the genomics field. For instance, Sargantana [298] is a RISC-V processor with support for a subset of the instructions part of the RISC-V vector extension. Additionally, it includes custom vector instructions to speed up the WFA kernel studied in this project. Furthermore, we find other accelerator proposals for genomics based on the RISC-V ISA [299], [300]. Similarly, we can find works leveraging the RVV extensions to speed up critical kernels in different domains, such as the post-quantum cryptography [301] or deep-learning [302]. Ramírez et al. [303] present a vector benchmark

suite for RISC-V. Further, we find several studies benchmarking the performance of HPC applications across different RISC-V cores [304], [305] and different architectures [306].

## 3.10. Conclusion

In this chapter, we presented the GenArch Benchmark suite (GenArchBench), comprising 13 computationally demanding kernels targeting HPC CPUs. These kernels originate from widely used genomics tools and exploit coarse-grained thread-level parallelism. We ported and optimized them to leverage the capabilities of two Arm processors (A64FX and Graviton3) and two RISC-V processors (HiFive Unmatched and Allwinner), including the Arm Scalable Vector Extension (SVE) and the RISC-V Vector Extension (RVV). In addition, we introduced algorithmic and code-level optimizations and adapted the kernels to exploit Arm-optimized libraries. The work also provides a detailed performance characterization across these platforms, together with two x86-64 reference systems (Intel Xeon Skylake Platinum 8160 and AMD EPYC 7742 Rome).

**SIMD Programming.** A key lesson learned during this project is that developing and maintaining intrinsic-based SIMD implementations is both labor-intensive and error-prone. Many genomics kernels are tightly coupled to x86-specific intrinsics such as SSE, AVX2, or AVX-512, and in several cases, no portable scalar implementation is available. Their structural complexity also hinders effective auto-vectorization, often forcing developers to rely on handwritten intrinsic code to obtain competitive performance. This approach, however, negatively impacts maintainability and readability, as it requires multiple architecture-specific code paths instead of a single portable implementation.

Vector-agnostic SIMD extensions, such as SVE and RVV, facilitate the development and maintenance of intrinsic code that works across different vector lengths. However, a higher-level solution, between intrinsics and auto-vectorized code, similar to the SIMDe library, may be the best way to enable applications to exploit the vector capabilities of current and future vector extensions and architectures.

**Microarchitectural Bottlenecks.** Another relevant observation concerns the difficulty of measuring and comparing microarchitectural bottlenecks across platforms. Hardware performance counters vary substantially across processors, and even counters with similar names may capture distinct events. Some systems expose only a minimal subset of counters, severely limiting performance analysis. In the two RISC-V systems evaluated, only instruction and cycle counters were available, preventing a detailed microarchitectural study. These limitations highlight the importance of standardizing hardware performance counters across architectures, which would facilitate deeper analysis of application bottlenecks and enable more direct and reliable cross-platform comparisons.

**A64FX.** When working with the A64FX, it is crucial to understand its micro-architecture in order to achieve optimal performance. The processor provides more limited out-of-order resources than many contemporary HPC CPUs, which motivates the use of aggressive

techniques such as loop fissioning to reduce pressure on execution resources [307]; the Fujitsu compiler, for instance, provides specific hints in this direction. Furthermore, its NUMA organization and memory subsystem require careful data placement and minimization of remote memory accesses in order to reduce latency and fully exploit the available bandwidth. The A64FX is primarily optimized for memory bandwidth, a design choice that often results in higher memory latencies and can make it less suitable for latency-sensitive workloads. In the context of GenArchBench, although only two kernels are strictly memory-bound, more than half exhibit a large fraction of stall cycles while waiting for data from memory on this architecture. Overall, the A64FX shows slowdowns of approximately 2× with respect to the reference SKX system, highlighting these latency and resource constraints

The peak power consumption of the evaluated x86 systems is considerably higher than that of the A64FX. However, x86 processors typically rely on dynamic voltage and frequency scaling to adapt energy usage to system load, whereas the A64FX tends to operate close to its maximum power level. Despite this, the A64FX achieves energy-to-solution figures comparable to the Skylake system when all cores are utilized. Future iterations of the A64FX would likely benefit from increased out-of-order resources and dynamic frequency scaling to broaden their competitiveness across diverse HPC workloads.

**Graviton3.** Graviton3 delivers strong out-of-the-box performance across all GenArchBench kernels without requiring extensive microarchitectural tuning. It achieves speedups over the reference x86 systems in both single-threaded and multi-threaded configurations. Nevertheless, the platform remains highly closed, exposing only a limited set of hardware counters and concealing frequency and power information, which restricts detailed performance and energy analyses.

**Whole Application Performance in Arm.** To contextualize the Arm evaluation results, we integrated the optimized Arm implementations of FMI and BSW into BWA-MEM2, a production-ready genomics application that relies on these kernels. In single-threaded execution, BWA-MEM2 exhibited comparable performance on Graviton3 and the x86 systems, while in fully multi-threaded runs, Graviton3 achieved speedups of 1.1–1.5× over the x86 processors. Conversely, the A64FX delivered roughly half the performance of the other platforms for this application.

Overall, all of our results demonstrate that Arm processors are fully capable of competing with x86 architectures for genomics workloads, and that GenArchBench provides a solid foundation for continued optimization of genome analysis tools on Arm.

**Unmatched and Allwinner.** The evaluated RISC-V platforms, Allwinner and HiFive Unmatched, are well-suited as development boards or for low-power scenarios, but they lack the computational resources typically associated with HPC systems, such as wide superscalar pipelines, powerful out-of-order execution engines, large and high-bandwidth memory hierarchies, and high-capacity main memory. These limitations significantly constrain their ability to process genomics workloads at scale. Nonetheless, these machines

have helped us study the current state and maturity of the RISC-V ISA and toolchain. Despite the rapid advancement of the RISC-V ecosystem, it still lacks critical support in compilers, libraries, tools, and mature hardware implementations. Compiler support for recently ratified extensions, including RVV, remains incomplete; debugging and profiling tools offer limited functionality; and key software libraries such as PyTorch and TensorFlow do not yet provide efficient RISC-V ports. In addition, hardware support for certain ISA extensions and performance counters is often limited or inconsistent. In our evaluation, these combined factors make the RISC-V platforms less competitive than the evaluated x86-64 and Arm processors for demanding genomics workloads.

Nevertheless, the openness and extensibility of RISC-V offer substantial opportunities for innovation, particularly through the development of domain-specific extensions and accelerators. The adoption of the recently ratified vector extension represents a crucial step toward improving both performance and energy efficiency in future HPC-oriented RISC-V processors. Although several technical and ecosystem challenges remain, RISC-V holds significant potential to evolve into a competitive platform for HPC in general, and for large-scale genome data analysis in particular.

**Insights for Processor Designers.** The cross-platform characterization of GenArchBench yields several actionable insights for architects of future HPC processors.

Memory stalls constitute a recurring major bottleneck across kernels and platforms. This underscores the importance of a well-designed memory hierarchy that correctly balances bandwidth and latency, as well as the value of mechanisms that actively reduce memory stalls, such as hardware prefetchers and sufficient Miss Status Holding Registers (MSHRs) to sustain high memory-level parallelism.

The RVV evaluation exposes gaps in the RISC-V vector ISA: the absence of vectorized count-leading-zeros and count-trailing-zeros instructions forces costly sequences of instructions to emulate them, negating the benefits of vectorization. To be competitive with other ISAs, future RISC-V vector extensions should prioritize the inclusion of instructions that are broadly useful across domains, not only genomics.

The scarcity of hardware performance counters and the unavailability of energy consumption metrics across several evaluated platforms severely limit a developer's ability to understand application behavior on a given machine. Without this information, optimizing an application is extremely difficult, since, as demonstrated in this work, performance bottlenecks can differ substantially across systems. Among the platforms evaluated, Intel provides the most comprehensive set of profiling tools, enabling developers to obtain detailed insight into application behavior on its processors. We encourage hardware manufacturers to invest more heavily in this area: without adequate observability tools, the full performance potential of a processor design cannot be realized in practice.

## 3.11. Future Work

Although this work introduces a broad set of optimizations for the GenArchBench kernels, there remains significant room for further improvement across all evaluated platforms,

not only Arm and RISC-V but also x86. In particular, several kernels implemented using the SIMDe library are only partially optimized, as full support for vector-length-agnostic extensions such as SVE and RVV is not yet available.

We have also observed that deep-learning and Python-based kernels exhibit poor parallel performance on Arm platforms, while their support on RISC-V remains limited or largely experimental. Given the rapidly growing importance of machine learning and data-driven workloads in bioinformatics and scientific computing, improving the scalability, efficiency, and architectural support of these software stacks is particularly critical.

We envision extending this study to newer generations of processors across all major architectures. Evaluating recent Arm CPUs, such as AWS Graviton4 and NVIDIA Grace, alongside more mature and powerful RISC-V systems, such as the 64-core MILK-V Pioneer, and contemporary x86 microarchitectures, such as Intel Sapphire Rapids and AMD Genoa, would provide valuable insights into the ever-evolving landscape of processor architectures. Furthermore, it is important to assess how software and toolchain support for each architecture is developing, which is particularly critical for RISC-V.

# Chapter 4.

# Singletrack: Reducing Memory Consumption and Improving Performance of Gap-Affine Sequence Alignment

*This chapter presents Singletrack, a novel backtrace algorithm that mitigates the memory requirements of gap-affine and dual gap-affine sequence alignment. By reconstructing optimal alignment paths using only one dynamic programming matrix, Singletrack reduces the memory footprint by 2× to 5× compared to classical algorithms. Additionally, this reduction in memory pressure translates into direct performance gains and improved scalability on many-core systems. We demonstrate the gains in memory footprint and performance by integrating Singletrack into the widely used KSW2 and WFA2-lib libraries.*

## 4.1. Introduction

The rapid advancements in DNA sequencing technologies over the past years have enabled the generation of vast amounts of sequencing data at unprecedented speeds [308]. The massive surge in sequencing data production has introduced a computational bottleneck in large-scale sequence analyses [45], [284], [309]. Notably, sequence alignment remains one of the most computationally demanding tasks in modern bioinformatics pipelines, such as read mapping [310], [311], de novo assembly [312], [313], and pangenome construction [314], [315].

The goal of pairwise sequence alignment is to find a sequence of operations (i.e., match, mismatch, insertion, and deletion) that maps one sequence to another while minimizing a given cost function. In practice, the gap-affine and dual gap-affine cost functions are preferred due to their ability to model biologically accurate alignments.

Classical dynamic programming (DP) algorithms are commonly used to compute gap-affine and dual gap-affine alignments. In their basic formulation, these algorithms require computing and storing multiple DP matrices (i.e., 3 for gap-affine and 5 for dual gap-affine), each of size $(n+1) \times (m+1)$, where $n$ and $m$ are the input sequence lengths. These matrices are later traced back to recover the optimal alignment.

Building upon this classical formulation, several modern alignment algorithms have been proposed to improve execution time and reduce memory usage. One notable example is the Suzuki-Kasahara (SK) formulation [228], which reformulates the gap-affine scoring equations so that the matrices store score differences instead of absolute scores. This

formulation increases the number of matrices to store, but reduces the size of each cell, resulting in overall memory savings. This approach is used by KSW2 [198], at the core of Minimap2 [198], and by the libgaba [228] library. More recently, the Wavefront Alignment Algorithm (WFA) [226] was proposed, which exploits homologous regions between sequences to substantially reduce the number of DP matrix cells computed and lower the memory requirements, particularly when aligning highly similar sequences.

Nevertheless, these implementations store all the computed cells of the DP matrices. In both the classical formulation and WFA, this involves storing three matrices for gap-affine and five for dual gap-affine alignment. In the case of the SK formulation, it requires storing four matrices for gap-affine alignment and six for dual gap-affine alignment. Compared to simpler cost functions, such as edit distance, using multiple matrices significantly increases memory requirements, making these algorithms particularly memory-intensive and degrading overall performance, especially when scaling to long sequences or when processing large alignment batches.

To address this challenge, practical sequence alignment implementations for gap-affine and dual gap-affine scoring adopt a variety of techniques to reduce memory usage. These include using heuristics at the expense of losing accuracy, storing compact backtrace representations instead of full matrices, and space-time trade-offs, such as divide-and-conquer strategies that recompute partial results to avoid storing the DP matrices entirely.

Heuristic-based pruning is a common technique to reduce memory usage by restricting the number of DP cells computed and stored. For instance, static banding restricts DP computations to a fixed diagonal band around the main diagonal, where the optimal alignment is expected to be found [316]. Alternatively, the computation region can be adjusted dynamically during alignment, pruning cells that fall too far from the most promising paths [204], [317]. Such heuristics may drastically reduce memory usage but risk missing the optimal alignment if it lies outside the explored region. Notwithstanding, they are standard (usually optional) features in many libraries, including KSW2 [198], Edlib [232], Parasail [318], and WFA2-lib [226].

Another memory-saving technique involves storing a backtrace matrix instead of the full DP matrices during alignment. That is, the aligner keeps only those DP cells required to compute subsequent DP cells, while the backtrace information (i.e., the direction of the next DP cell during the backtrace) is stored in a compact auxiliary matrix using small-width elements (e.g., 8-bit values). This approach is implemented in widely-used libraries such as KSW2 [198] and Parasail [318]. While effective at reducing memory consumption, this method introduces significant overhead from additional data manipulation and memory accesses during the alignment process.

Another common approach is to trade additional computation for reduced memory usage using a divide-and-conquer approach. The divide-and-conquer paradigm can be leveraged to recursively break the alignment process into smaller pieces, effectively transforming the quadratic memory consumption of classical algorithms to linear memory consumption. This technique is used by Hirschberg's algorithm [319], FORAlign [320], which accelerates Hirschberg by using the Four Russians method [321], and BiWFA [322], a version of WFA capable of exploiting divide-and-conquer. However, these linear-memory approaches, while extremely memory-efficient, tend to be slower than their quadratic-memory counterparts

for moderate sequence lengths. For instance, the BiWFA paper reports that WFA outperforms BiWFA for sequences of up to 100 Kbp.

This work presents Singletrack, an efficient backtracing algorithm that requires storing only a single DP matrix, even when computing gap-affine and dual gap-affine alignments. We demonstrate that storing a single DP matrix is enough to recover the optimal alignment while maintaining both the linear time complexity of the backtrace and the optimality of the alignments. Moreover, we show that Singletrack is broadly applicable with state-of-the-art DP-based methods, such as SK formulation, WFA, and commonly used heuristic strategies. To demonstrate applicability and performance, we integrated Singletrack into KSW2 (which implements SK and uses the backtrace matrix approach to reduce memory consumption) and WFA2-lib (which implements WFA), yielding the KSW2+Singletrack and WFA+Singletrack optimized implementations.

We show that Singletrack significantly reduces memory consumption in both SK and WFA, thereby mitigating memory pressure and often translating into performance gains. In the SK formulation, memory consumption is reduced by 2× and 4× for gap-affine and dual gap-affine alignments, respectively. Although Singletrack increases memory usage by 2× compared to the memory reduction approach used in KSW2, it reduces execution time by up to 1.4×. In WFA, Singletrack lowers memory usage by 3× and 5× for gap-affine and dual gap-affine, respectively, and delivers speedups of 1.2× to 2.1× compared to the implementation in WFA2-lib. Compared to the linear-space BiWFA algorithm implemented in WFA2-lib, WFA+Singletrack achieves speedups of up to 5.2× at the cost of increased memory usage, making it a compelling alternative for moderately long sequences.

## 4.2. Background

### 4.2.1. Pairwise Sequence Alignment

Let the query $q = q_0, q_1, \ldots, q_{n-1}$ and the target $t = t_0, t_1, \ldots, t_{m-1}$ be two sequences of length $n$ and $m$ characters, respectively. The optimal sequence alignment is the sequence of four basic operations (i.e., match, mismatch, insertion, and deletion) that transform $q$ into $t$ while minimizing a given cost function.

Naturally, the choice of cost function strongly affects the alignment result and its biological relevance, as well as the computational and memory requirements. The simplest cost functions are the edit-distance and gap-linear (a.k.a. weighted edit-distance). These functions use three values to score matches ($a$), mismatches ($x$), and gap extensions ($e$) for both insertions and deletions. In the case of the edit-distance, these values are fixed to $a = 0$, $x = 1$, $e = 1$, while gap-linear functions allow customizable values to match biological or empirical scoring needs. In the following, for simplicity, we define a substitution function $S(i, j)$ that evaluates to $a$ if $q_{i-1} = t_{j-1}$ and $x$ if $q_{i-1} \neq t_{j-1}$.

Edit-distance and gap-linear alignments are traditionally computed using DP algorithms that require computing and storing a single DP matrix $M$ of size $(n+1)\cdot(m+1)$, where each cell $M_{i,j}$ is filled using a simple recurrence, $M_{i,j} = \min\{M_{i,j-1} + e, M_{i-1,j} + e, M_{i-1,j-1} + S(i, j)\}$. As a result, the optimal score of the alignment is found in $M_{n,m}$, and the optimal

alignment can be recovered by tracing back from this cell the sequence of minimum-cost operations that led to it, using the backtrace algorithm.

### 4.2.2. Gap-Affine and Dual Gap-Affine Alignment

Despite their simplicity, gap-linear cost functions have notable limitations. These functions model long gaps as multiple independent insertions or deletions, whereas in reality, they often stem from a single biological event. This can lead to fragmented alignments and misleading interpretations, making linear models unsuitable in some cases. To reflect this phenomenon, the gap-affine function [207], [323] introduces a gap-opening cost ($o_1$). This function models cases where a gap follows a different type of operation, such as a mismatch or match. For example, two insertions separated by a non-insertion costs $2 \cdot o_1 + 2 \cdot e_1$, while a single gap of two insertions costs $o_1 + 2 \cdot e_1$.

However, gaps in sequence alignments arise from two distinct mechanisms: evolutionary events, which can introduce long gaps through a single event (e.g., transposon insertion), and sequencing errors, which tend to generate multiple short gaps. To model this distinction and produce more biologically accurate alignments, the dual gap-affine function [324] is often preferred. To that end, the dual gap-affine function introduces a second set of gap penalties ($o_2, e_2$), allowing for two types of gaps. One type of gap has a low opening cost but a high extension cost (to model short gaps), while the other has a high opening cost but a low extension cost (to model large gaps). For each gap of length $l$, the alignment algorithm chooses the lower-cost option: $\min\{o_1 + l \cdot e_1, o_2 + l \cdot e_2\}$.

Similar to gap-linear models, gap-affine and dual gap-affine alignments are typically computed using traditional DP algorithms. However, computing gap-affine alignments requires computing and storing three DP matrices ($M$,$I1$, and $D1$). More demanding still, dual gap-affine alignment requires five matrices ($M$,$I1$,$D1$,$I2$, and $D2$). In particular, the $I$ matrices ($I1$ and $I2$) and $D$ matrices ($D1$ and $D2$) track optimal scores for alignments ending in insertions and deletions, respectively. Then, the $M$ matrix summarizes the minimum alignment score between prefixes $q_{0,\ldots,i-1}$ and $t_{0,\ldots,j-1}$ for each cell $M_{i,j}$.

As in gap-linear models, the optimal alignment score is located in cell $M_{n,m}$. To reconstruct the alignment, a backtrace procedure traverses the $M$, $I$, and $D$ matrices, following the path of minimum-cost operations that led to this cell (i.e., optimal alignment).

The recurrence equations used to compute the gap-affine and dual gap-affine DP matrices are presented in Equation 4.1 and Equation 4.2.

$$
\begin{aligned}
I1_{i,j} &= \begin{cases} \infty & (i \geq 0, j = 0) \\ o_1 + j \cdot e_1 & (i = 0, j > 0) \\ \min\{I1_{i,j-1} + e_1, M_{i,j-1} + o_1 + e_1\} & (i > 0, j > 0) \end{cases} \\
D1_{i,j} &= \begin{cases} o_1 + i \cdot e_1 & (i > 0, j = 0) \\ \infty & (i = 0, j \geq 0) \\ \min\{D1_{i-1,j} + e_1, M_{i-1,j} + o_1 + e_1\} & (i > 0, j > 0) \end{cases} \\
M_{i,j} &= \begin{cases} 0 & (i = 0, j = 0) \\ o_1 + i \cdot e_1 & (i > 0, j = 0) \\ o_1 + j \cdot e_1 & (i = 0, j > 0) \\ \min\{I1_{i,j}, D1_{i,j}, M_{i-1,j-1} + S(i,j)\} & (i > 0, j > 0) \end{cases}
\end{aligned} \tag{4.1}
$$

$$
\begin{aligned}
I1_{i,j} &= \begin{cases} \infty & (i \geq 0, j = 0) \\ o_1 + j \cdot e_1 & (i = 0, j > 0) \\ \min\{I1_{i,j-1} + e_1, M_{i,j-1} + o_1 + e_1\} & (i > 0, j > 0) \end{cases} \\
D1_{i,j} &= \begin{cases} o_1 + i \cdot e_1 & (i > 0, j = 0) \\ \infty & (i = 0, j \geq 0) \\ \min\{D1_{i-1,j} + e_1, M_{i-1,j} + o_1 + e_1\} & (i > 0, j > 0) \end{cases} \\
I2_{i,j} &= \begin{cases} \infty & (i \geq 0, j = 0) \\ o_2 + j \cdot e_2 & (i = 0, j > 0) \\ \min\{I2_{i,j-1} + e_2, M_{i,j-1} + o_2 + e_2\} & (i > 0, j > 0) \end{cases} \\
D2_{i,j} &= \begin{cases} o_2 + i \cdot e_2 & (i > 0, j = 0) \\ \infty & (i = 0, j \geq 0) \\ \min\{D2_{i-1,j} + e_2, M_{i-1,j} + o_2 + e_2\} & (i > 0, j > 0) \end{cases} \\
M_{i,j} &= \begin{cases} 0 & (i = 0, j = 0) \\ \min\{o_1 + i \cdot e_1, o_2 + i \cdot e_2\} & (i > 0, j = 0) \\ \min\{o_1 + j \cdot e_1, o_2 + j \cdot e_2\} & (i = 0, j > 0) \\ \min\{I1_{i,j}, D1_{i,j}, I2_{i,j}, D2_{i,j}, M_{i-1,j-1} + S(i,j)\} & (i > 0, j > 0) \end{cases}
\end{aligned} \tag{4.2}
$$

### 4.2.3. Classical Backtrace Algorithm

The classical backtrace algorithm reconstructs an optimal alignment path by traversing the $M$, $I$, and $D$ matrices. More in detail, the backtrace procedure starts at the last cell, $M_{n,m}$. At each step, the algorithm identifies the predecessor cell that originated the score of the current cell by evaluating the transitions defined in Equation 4.1. This transition corresponds to an alignment operation, belonging to the optimal alignment which is

commonly represented using the CIGAR string format: `M` for match, `X` for mismatch, `I` for insertion, and `D` for deletion. The algorithm then moves to the identified predecessor cell and repeats the process until it reaches the starting cell $M_{0,0}$. The pseudocode of the classical backtrace is shown in Algorithm 4.1. It is important to note that multiple transitions may lead to the same minimum-cost cell, resulting in different equally optimal alignment paths. In the case of the dual gap-affine function, the backtrace algorithm is analogous.

**Algorithm 4.1:** Classical gap-affine backtrace

**Input:** Sequences $q_{0,\ldots,n-1}$, $t_{0,\ldots,m-1}$, matrices $M, I1, D1$, penalties $\{a, x, o_1, e_1\}$
**Output:** CIGAR string that represents the optimal alignment

```
1  Function BACKTRACE(q, t, M, I1, D1, p = {a, x, o1, e1}) begin
2      Definition: push_op(cigar, op, len) appends op len times to cigar.
3      cigar ← ϵ, i ← n, j ← m, state ← M
4      while i > 0 ∨ j > 0 do
5          if state = M then
6              if M_{i,j} = I1_{i,j} then
7                  state ← I1
8              else if M_{i,j} = D1_{i,j} then
9                  state ← D1
10             else
11                 if q_{i−1} = t_{j−1} then push_op(cigar, 'M', 1) ;
12                 else push_op(cigar, 'X', 1) ;
13                 i ← i − 1, j ← j − 1
14         else if state = I1 ∧ j > 0 then
15             if I1_{i,j} ≠ I1_{i,j−1} + e1 then
16                 state ← M
17             push_op(cigar, 'I', 1)
18             j ← j − 1
19         else if state = D1 ∧ i > 0 then
20             if D1_{i,j} ≠ D1_{i−1,j} + e1 then
21                 state ← M
22             push_op(cigar, 'D', 1)
23             i ← i − 1
24         else
25             state ← M
26     return cigar
```

For example, Figure 4.1 shows how the classical backtrace algorithm operates over the computed DP matrices when aligning two sequences ($q$ = `GCA` and $t$ = `GCCAA`) using a gap-affine cost function with penalties $a = 0$, $x = 4$, $o_1 = 6$, and $e_1 = 2$. The dark-shaded

**I1**

| | - j=0 | G j=1 | C j=2 | C j=3 | A j=4 | A j=5 |
|---|---|---|---|---|---|---|
| - i=0 | ∞ | 8 | 10 | 12 | 14 | 16 |
| G i=1 | ∞ | 16 | 8 | 10 | 12 | 14 |
| C i=2 | ∞ | 18 | 16 | 8 | 10 | 12 |
| A i=3 | ∞ | 20 | 18 | 16 | 12 | 14 |

**M**

| | - j=0 | G j=1 | C j=2 | C j=3 | A j=4 | A j=5 |
|---|---|---|---|---|---|---|
| - i=0 | 0 | 8 | 10 | 12 | 14 | 16 |
| G i=1 | 8 | 0 | 8 | 10 | 12 | 14 |
| C i=2 | 10 | 8 | 0 | 8 | 10 | 12 |
| A i=3 | 12 | 10 | 8 | 4 | 8 | 10 |

**D1**

| | - j=0 | G j=1 | C j=2 | C j=3 | A j=4 | A j=5 |
|---|---|---|---|---|---|---|
| - i=0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| G i=1 | 8 | 16 | 18 | 20 | 22 | 24 |
| C i=2 | 10 | 8 | 16 | 18 | 20 | 22 |
| A i=3 | 12 | 10 | 8 | 16 | 18 | 20 |

Figure 4.1.: Classical backtrace for the gap-affine alignment of $q =$ `GCA` and $t =$ `GCCAA` using penalties $a = 0$, $x = 4$, $o_1 = 6$, and $e_1 = 2$. Dark-shaded cells indicate the optimal path over matrices $M$, $I1$, and $D1$.

cells in the figure indicate the optimal alignment path.

The backtrace begins at cell $M_{n,m} = M_{3,5} = 10$. The only valid transition is from $M_{2,4}$, since $M_{3,5} = M_{2,4} + S(3,5)$. The current cell is updated to $M_{2,4}$, and the CIGAR string is set to `M`. From $M_{2,4}$, the algorithm moves to $I1_{2,3}$, as $M_{2,4} = I1_{2,3} + e_1$, updating the CIGAR string to `IM`. Then, since $I1_{2,3} = M_{2,2} + o_1 + e_1$, the path returns to $M_{2,2}$, resulting in `IIM`. Next, $M_{2,2}$ traces back to $M_{1,1}$ via $M_{2,2} = M_{1,1} + S(2,2)$, giving `MIIM`. Finally, $M_{1,1} = M_{0,0} + S(1,1)$, and the completed CIGAR string is `MMIIM`. This example illustrates how the presence of a gap in the alignment requires transitioning into $I1$ or $D1$ during backtrace and returning to $M$ once the gap is closed.

## 4.3. The Singletrack Algorithm

We now describe Singletrack, a method for performing the backtrace of gap-affine and dual gap-affine alignments in linear execution time using only the $M$ matrix. As a result, the alignment process no longer needs to store the indel matrices, significantly reducing memory usage. For simplicity, the description focuses on the gap-affine, noting that the method extends naturally to dual gap-affine alignments.

We observe that Equation 4.1 shows that while computing $M$ requires values from all three matrices, each indel matrix ($I1$, $D1$) depends only on $M$ and itself. Thus, any cell in an indel matrix traces back either to a cell in $M$ or to another cell within the same indel matrix. Hence, during backtrace, if the current cell is in an indel matrix such as $I1$ with score $s$, we can avoid accessing $I1$ by checking first whether $s = M_{i,j-1} - o_1 - e_1$. If this holds, the gap was opened from $M_{i,j-1}$; otherwise, the gap continues, and the predecessor is necessarily $I1_{i,j-1}$ with score $s' = s - e_1$. As a result, backtracing through $I1$ can be performed iteratively without accessing its scores, until the trace returns to a cell in $M$. Singletrack generalizes this idea to avoid accessing $I1$ and $D1$ scores. When the current cell is in $M$, say $M_{i,j}$, we first check if the predecessor is in $M$ by checking if $M_{i,j} = M_{i-1,j-1} + S(i,j)$. If not, the predecessor cell is either $I1_{i,j-1}$ or $D1_{i-1,j}$. Although we do not know which one, from Equation 4.1, we know that the score of the predecessor cell must be $s' = M_{i,j} - e_1$. At this point, we explore two tentative backtrace paths simultaneously: one assuming the predecessor lies in $I1$, and the other in $D1$.

Then, we extend both paths step by step, increasing the gap length $l$ and computing the expected score to return to $M$ as $s(l) = M_{i,j} - o_1 - l \cdot e_1$. Eventually, either $M_{i,j-l} = s(l)$ or $M_{i-l,j} = s(l)$ holds, indicating that the correct backtrace path goes from $M_{i,j}$ to $M_{i,j-l}$ through $I1$ (insertion) or to $M_{i-l,j}$ through $D1$ (deletion), respectively. Once a consistent path back to $M$ is found, the other path is discarded, and the algorithm continues the backtrace in $M$. The pseudocode of Singletrack's backtrace algorithm is shown in Algorithm 4.2 for gap-affine and in Algorithm 4.3 for dual gap-affine. Note that Singletrack always finds an optimal path, just as the classical backtrace algorithm does. If multiple optimal paths exist, it will select any of them, ensuring optimality in all cases.

**Algorithm 4.2:** Singletrack gap-affine backtrace

**Input:** Sequences $q_{0,\ldots,n-1}$, $t_{0,\ldots,m-1}$, matrix $M$, penalties $\{a, x, o_1, e_1\}$
**Output:** CIGAR string that represents the optimal alignment

```
1  Function BACKTRACE(q, t, M, p = {a, x, o1, e1}) begin
2      Definition: push_op(cigar, op, len) appends op len times to cigar.
3      cigar ← ∅
4      state ← M
5      i ← n, j ← m, l ← 0
6      while i > 0 ∨ j > 0 do
7          if state = M then
8              if i > 0 ∧ j > 0 ∧ M_{i,j} = M_{i-1,j-1} + S(i,j) then
9                  if q_{i-1} = t_{j-1} then push_op(cigar, 'M', 1) ;
10                 else push_op(cigar, 'X', 1) ;
11                 i ← i − 1, j ← j − 1
12             else
13                 l ← 0, state ← M̄
14         else
15             l ← l + 1
16             s' ← M_{i,j} − o1 − l · e1
17             if j − l ≥ 0 ∧ s' = M_{i,j-l} then
18                 push_op(cigar, 'I', l)
19                 j ← j − l, state ← M
20             else if i − l ≥ 0 ∧ s' = M_{i-l,j} then
21                 push_op(cigar, 'D', l)
22                 i ← i − l, state ← M
23     return cigar
```

Following the previous example, Figure 4.2 shows the behavior of the Singletrack backtrace algorithm. For that, we show both the stored and recomputed cells: scores stored in memory (corresponding to the $M$ matrix) are displayed in black, while tentative scores, computed on the fly during the backtrace but not stored, are shown in dark grey.

**Algorithm 4.3:** Singletrack dual gap-affine backtrace

**Input:** Sequences $q_{0,\ldots,n-1}$, $t_{0,\ldots,m-1}$, matrix $M$, penalties $\{a, x, o_1, e_1, o_2, e_2\}$
**Output:** CIGAR string that represents the optimal alignment

```
1  Function BACKTRACE(q, t, M, p = {a, x, o1, e1, o2, e2}) begin
2      Definition: push_op(cigar, op, len) appends op len times to cigar.
3      cigar ← ϵ
4      state ← M
5      i ← n, j ← m, l ← 0
6      while i > 0 ∨ j > 0 do
7          if state = M then
8              if i > 0 ∧ j > 0 ∧ M_{i,j} = M_{i−1,j−1} + S(i, j) then
9                  if q_{i−1} = t_{j−1} then push_op(cigar, 'M', 1) ;
10                 else push_op(cigar, 'X', 1) ;
11                 i ← i − 1, j ← j − 1
12             else
13                 l ← 0, state ← M̄
14         else
15             l ← l + 1
16             s'_1 ← M_{i,j} − o_1 − l · e_1
17             s'_2 ← M_{i,j} − o_2 − l · e_2
18             if j − l ≥ 0 ∧ s'_1 = M_{i,j−l} then
19                 push_op(cigar, 'I', l)
20                 j ← j − l, state ← M
21             else if i − l ≥ 0 ∧ s'_1 = M_{i−l,j} then
22                 push_op(cigar, 'D', l)
23                 i ← i − l, state ← M
24             else if j − l ≥ 0 ∧ s'_2 = M_{i,j−l} then
25                 push_op(cigar, 'I', l)
26                 j ← j − l, state ← M
27             else if i − l ≥ 0 ∧ s'_2 = M_{i−l,j} then
28                 push_op(cigar, 'D', l)
29                 i ← i − l, state ← M
30     return cigar
```

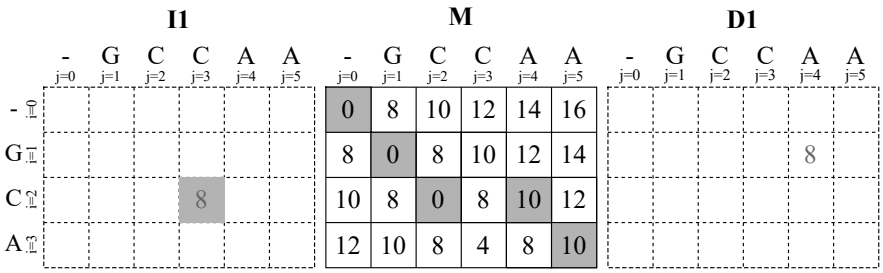



Figure 4.2.: Singletrack backtrace for the gap-affine alignment of $q = \texttt{GCA}$ and $t = \texttt{GCCAA}$ using penalties $a = 0$, $x = 4$, $o_1 = 6$, and $e_1 = 2$. Only the matrix $M$ is stored. Strictly required $I1$ and $D1$ values are recomputed on the fly. Dark-shaded cells indicate the optimal path.

In the first step, the behavior matches the classical algorithm since $M_{3,5} = M_{2,4} + S(3,5)$ and the CIGAR string is set to `M`. At $M_{2,4}$, since $M_{2,4} \neq M_{1,3} + S(2,4)$, the predecessor cell must be either $I1_{2,4}$ or $D1_{2,4}$, both having a tentative score $s' = M_{2,4} = 10$. For a tentative gap length of $l = 1$, the expected score back to $M$ should be $s' = M_{2,4} - o_1 - l \cdot e_1 = 2$, which does not correspond to $M_{2,3}$ or $M_{1,4}$. Note that the tentative score of $I1_{2,3}$ and $D1_{1,4}$ is $s' = M_{2,4} - e_1 = 8$, corresponding to the true score at $I1_{2,3}$ (correct backtrace path) but not of $D1_{1,4}$ (see Figure 4.1). Increasing the gap length to $l = 2$, the algorithm extends both paths and checks whether the updated score back to $M$, $s' = 10 - 6 - 2 \cdot 2 = 0$, matches either $M_{2,2}$ or $M_{1,3}$. Since $s' = M_{2,2}$, the correct path goes through $I1$, and two insertions are added to the CIGAR string, which becomes `IIM`. From this point, the algorithm proceeds as in the classical case, and the final CIGAR string is `MMIIM`, matching the result produced by the original backtrace algorithm.

The time complexity of the Singletrack algorithm is linear $\mathcal{O}(n + m)$, the same as the classical backtrace. In the worst-case scenario, Singletrack may explore up to twice as many cells as the classical backtrace algorithm. However, the backtrace step generally represents only a small fraction of the total alignment time, which is dominated by the quadratic $\mathcal{O}(nm)$ time complexity of the alignment step. In the case of dual gap-affine alignments, the approach is identical, but with four tentative paths corresponding to $I1$, $D1$, $I2$, and $D2$, potentially increasing the worst-case number of explored cells by up to four times compared to the classical algorithm.

The key advantage of Singletrack lies in its ability to compute optimal alignments while storing only the $M$ matrix and a minimal set of values from the other matrices (i.e., $I1$, $D1$, $I2$, $D2$). In particular, to compute the DP cells using Equation 4.1 column-wise, the horizontal recurrences of $M$ and $I1$ depend on the left neighbour $I1_{i,j-1}$, which requires storing the full previous column of $I1$. In contrast, the vertical recurrences of $M$ and $D1$ depend on the top neighbour $D1_{i-1,j}$. Under column-wise iteration, this value corresponds to a single scalar that can be carried forward, so only one cell from $D1$ needs to be stored. Exploiting this property allows the alignment to be computed without any additional overhead while reducing memory usage by approximately $3\times$ for gap-affine alignments. The same reasoning extends naturally to the dual gap-affine case

by additionally storing a full column of $I2$ and a single cell of $D2$, resulting in a 5× reduction in memory consumption.

Moreover, requiring only a small set of the indel matrices enables software optimizations. For instance, the single element needed from $D1$ (and $D2$) can be stored directly in a CPU register rather than in memory, significantly reducing memory accesses and latency, and improving performance. Similarly, the memory of the single column required from $I1$ (and $I2$) can be reused continuously, which improves the cache locality and reduces pressure on lower levels of the memory hierarchy, which is particularly advantageous when aligning long sequences or processing large alignment batches.

More importantly, Singletrack can be applied to any DP-based sequence alignment algorithm, provided that the required scores of the $M$ matrix are available during the backtrace. As shown in the following sections, Singletrack integrates seamlessly with the Suzuki-Kasahara formulation [228] and with the Wavefront Alignment Algorithm [226]. Additionally, it is compatible with heuristic alignment approaches, such as bands, early termination, drops, or score filtering, requiring only additional checks for boundary conditions during the backtrace. For simplicity, we focus on global alignment in this work, noting that the same idea extends naturally to semi-global, ends-free, and local alignments.

We have integrated the Singletrack backtrace into the classical dynamic programming algorithm. The implementation is available on Zenodo (DOI:10.5281/zenodo.18770585) and GitHub (https://github.com/LorienLV/singletrack).

## 4.4. Singletrack in Suzuki-Kasahara Formulation

The Suzuki-Kasahara (SK) formulation [228] extends the difference encoding idea from the BPM algorithm [225] to the gap-affine cost function, storing only the differences between adjacent DP cells rather than full scores. Since these differences and alignment penalties are usually small integers, they can be efficiently represented with reduced data types, such as 8-bit integers. The standard recurrence relations (Equation 4.1) are reformulated in terms of score differences for the $M$, $I1$, and $D1$ matrices. Specifically, we define the horizontal, vertical, and diagonal differences in the $M$ matrix as $\Delta H_{i,j} = M_{i,j} - M_{i,j-1}$, $\Delta V_{i,j} = M_{i,j} - M_{i-1,j}$, and $A_{i,j} = M_{i,j} - M_{i-1,j-1}$, respectively. Additionally, the differences between the $I1$ and $D1$ matrices and the $M$ matrix are given by $\Delta E1_{i,j} = I1_{i,j} - M_{i,j}$ and $\Delta F1_{i,j} = D1_{i,j} - M_{i,j}$. The resulting Suzuki-Kasahara gap-affine recurrence relations are shown in Equation 4.3.

$$
\begin{aligned}
A_{i,j} &= \min \begin{cases} S(i,j) \\ \Delta E1_{i,j-1} + \Delta V_{i,j-1} + e_1 \\ \Delta F1_{i-1,j} + \Delta H_{i-1,j} + e_1 \end{cases} \\
\Delta H_{i,j} &= A_{i,j} - \Delta V_{i,j-1} \\
\Delta V_{i,j} &= A_{i,j} - \Delta H_{i-1,j} \\
\Delta E1_{i,j} &= \min\{o_1, \Delta E1_{i,j-1} - \Delta H_{i,j} + e_1\} \\
\Delta F1_{i,j} &= \min\{o_1, \Delta F1_{i-1,j} - \Delta V_{i,j} + e_1\}
\end{aligned}
\tag{4.3}
$$

With these new recurrence relations, four matrices are stored: $\Delta H_{i,j}$, $\Delta V_{i,j}$, $\Delta E1_{i,j}$, and $\Delta F1_{i-1,j}$, compared to only three in the original formulation (where $A_{i,j}$ is a temporary variable and not stored in memory). However, since the new matrices store narrow-integer values (e.g., 8-bit integers), the overall memory footprint is actually reduced compared to the original approach (assuming 32-bit integer encoding baseline). Furthermore, this reformulation increases the number of elements that can be processed simultaneously through SIMD (Single Instruction, Multiple Data) operations available in modern CPUs, enhancing computational performance.

To apply the Singletrack backtrace under the SK formulation, it is sufficient to retain only the $\Delta V$ and $\Delta H$ matrices, as they provide all the information required to reconstruct the original $M$ matrix (the remaining $\Delta E_1$ and $\Delta F_1$ matrices can be discarded). However, the Singletrack algorithm requires integer values from the $M$ matrix to trace back the optimal alignment from $M_{n,m}$ to $M_{0,0}$. To address this, we define a new matrix $M'$ such that $M'_{i,j} = M_{i,j} - M_{n,m}$ for all $i \in \{0, \ldots, n\}$ and $j \in \{0, \ldots, m\}$. By construction, $M'_{n,m} = 0$, providing a known integer value to start the backtrace. During the backtrace, the relevant entries of $M'$ are reconstructed on the fly using the stored differential matrices $\Delta V$ and $\Delta H$ as $M'_{i,j} = M'_{i-1,j-1} + \Delta H_{i-1,j} + \Delta V_{i,j}$. Since $M'$ is simply a shifted version of $M$, the original alignment path is preserved, and the general Singletrack algorithm remains unchanged. These ideas naturally extend to the SK formulation of the dual gap-affine cost function.

We have integrated Singletrack into KSW2[1], which provides a SIMD-accelerated implementation of the Suzuki-Kasahara formulation and serves as a core component of Minimap2 [198], a widely adopted read mapper and a foundational tool in many bioinformatics pipelines. KSW2 implements the previously described backtrace matrix approach to reduce memory consumption, in which the backtrace information for all DP cells is generated on the fly during alignment and stored in a single compressed matrix of 8-bit elements. This approach requires 4× and 6× less memory than the original Suzuki-Kasahara implementation for gap-affine and dual gap-affine, respectively. However, it introduces additional computation in the alignment step by forcing the precomputation of backtrace information for all DP cells, including those that are not accessed during traceback because they do not lie on the optimal alignment path.

[1] https://github.com/lh3/ksw2

We produced a minimal version of KSW2 that utilizes Singletrack for backtrace, referred to as KSW2+Singletrack. Instead of computing the backtrace matrix, the matrices $\Delta H$ and $\Delta V$ are stored. This approach requires approximately twice the memory compared to the original KSW2; however, it eliminates all data manipulation associated with the backtrace matrix. Note that KSW2+Singletrack uses 2× and 4× less memory than the classical approach that stores all matrices for gap-affine and dual gap-affine, respectively. The KSW2+Singletrack implementation is available at https://github.com/LorienLV/singletrack.

## 4.5. Singletrack in Wavefront Alignment Algorithm

The Wavefront Alignment Algorithm [226] (WFA) proposes alternative recurrence relations to compute the optimal gap-affine alignment. For each matrix ($M$, $I1$, $D1$) and diagonal $k$ of the DP matrix, WFA computes and stores the offset (column) of the most advanced DP cell with score $z$, known as the furthest reaching point (f.r.p.). The vector of offsets for a given score $z$ is called the $z$-wavefront. In the gap-affine case, WFA computes three wavefronts ($\widetilde{M}_z$, $\widetilde{I1}_z$, $\widetilde{D1}_z$) per each score value following Equation 4.4.

$$\begin{aligned}
\widetilde{I1}_{z,k} &= \max\{\widetilde{M}_{z-o_1-e_1,k-1}, \widetilde{I1}_{z-e_1,k-1}\} + 1 \\
\widetilde{D1}_{z,k} &= \max\{\widetilde{M}_{z-o_1-e_1,k+1}, \widetilde{D1}_{z-e_1,k+1}\} \\
\widetilde{M}_{z,k} &= \max\{\widetilde{M}_{z-x,k} + 1, \widetilde{I1}_{z,k}, \widetilde{D1}_{z,k}\}
\end{aligned} \tag{4.4}$$

Once the wavefronts for the optimal score $s$ have been computed, the alignment path is recovered through a backtrace procedure that follows the same principles as the classical backtrace algorithm. That is, the WFA backtrace reconstructs the optimal alignment path following the origin of each furthest-reaching point in Equation 4.4 from $\widetilde{M}_s$ to $\widetilde{M}_0$, accessing all computed wavefronts in between (i.e., $\widetilde{M}_z$, $\widetilde{I1}_z$, and $\widetilde{D1}_z$)

Unlike classical DP-based alignment algorithms, which require quadratic time and memory, WFA has a time complexity of $\mathcal{O}(ns)$ and a space complexity of $\mathcal{O}(s^2)$, where $n$ is the length of the sequences (assuming $n \sim m$) and $s$ is the optimal alignment score. In practice, WFA outperforms most sequence alignment algorithms, especially when the sequences aligned are highly similar.

Analogous to the general Singletrack strategy, we assume that the WFA backtrace algorithm only has access to the $\widetilde{M}_{0,\ldots,s}$ wavefronts. As in other sections, we focus on gap-affine, noting that the described ideas extend naturally to dual gap-affine. It is important to note that WFA does not explicitly store all computed DP cells due to its use of the `extend` operation. After generating a DP cell with score $z$, diagonal $k$, and offset $off'$ using the `next` operation, the `extend` operation increases the offset as long as matches are found along the diagonal, ultimately storing only the offset $off$ corresponding to the furthest-reaching cell for each diagonal and score. Consequently, when considering a current cell $off = \widetilde{M}_{z,k}$, the original offset $off'$ before extension is not directly available. To address this, the Singletrack algorithm performs a reverse extension (`rev_extend`)

**Algorithm 4.4:** Singletrack gap-affine backtrace in WFA

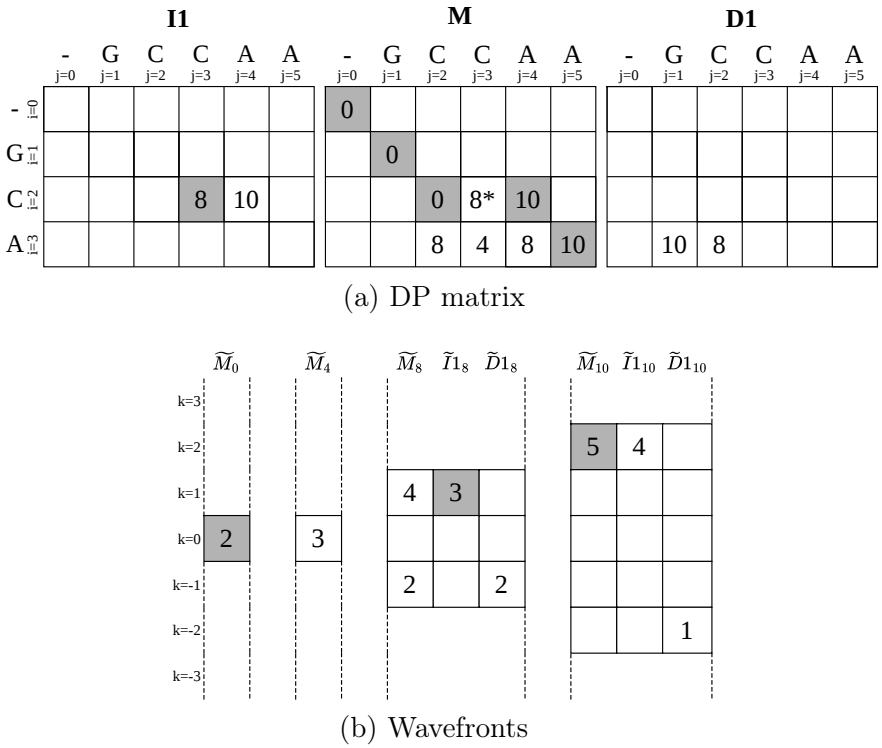
**Input:** Sequences $q_{0,\ldots,n-1}$, $t_{0,\ldots,m-1}$, optimal score $s$, wavefronts $\widetilde{M}_{0,\ldots,s}$, penalties $\{a = 0, x, o_1, e_1\}$

**Output:** CIGAR string that represents the optimal alignment

**Function** BACKTRACE($q, t, s, \widetilde{M}, p$) **begin**
  **Definition:** `push_op(cigar, op, len)` appends `op` `len` times to `cigar`.
  `// Assume invalid accesses to` $\widetilde{M}$ `returns` $-\infty$
  cigar $\leftarrow \emptyset, z \leftarrow s, l \leftarrow 0, k \leftarrow m - n, \text{off} \leftarrow m, \text{state} \leftarrow M$
  **while** $z > 0$ **do**
    **if** state $= M$ **then**
      $i \leftarrow \text{off} - k, j \leftarrow \text{off}, \text{off}' \leftarrow \text{off}$
      **while** $i > 0 \wedge j > 0 \wedge q_{i-1} = t_{j-1}$ **do**
        $i \leftarrow i - 1, j \leftarrow j - 1, \text{off}' \leftarrow \text{off}' - 1$
      $z' \leftarrow z - x$
      **if** $\widetilde{M}_{z',k} + 1 = \text{off}'$ **then**
        push_op(cigar, 'M', off $-$ off$'$)
        push_op(cigar, 'X', 1)
        $z \leftarrow z', \text{off} \leftarrow \text{off}' - 1$
      **else**
        $l \leftarrow 0, \text{state} \leftarrow \bar{M}$
    **else**
      $l \leftarrow l + 1, z' \leftarrow z - o_1 - l \cdot e_1$
      **if** $\text{off}' \leq \widetilde{M}_{z',k-l} + l \leq \text{off}$ **then**
        push_op(cigar, 'M', off $-$ $(\widetilde{M}_{z',k-l} + l)$)
        push_op(cigar, 'I', $l$)
        $\text{off} \leftarrow \widetilde{M}_{z',k-l}, k \leftarrow k - l, z \leftarrow z', \text{state} \leftarrow M$
      **else if** $\text{off}' \leq \widetilde{M}_{z',k+l} \leq \text{off}$ **then**
        push_op(cigar, 'M', off $-$ $\widetilde{M}_{z',k+l}$)
        push_op(cigar, 'D', $l$)
        $\text{off} \leftarrow \widetilde{M}_{z',k+l}, k \leftarrow k + l, z \leftarrow z', \text{state} \leftarrow M$
  push_op(cigar, 'M', off)
  **return** cigar

operation. Starting from the current cell $off = \widetilde{M}_{z,k}$, `rev_extend` decrements the offset while matches are found along the diagonal in reverse, recovering the tentative original offset $off'$. Once this reverse extension is complete, we determine the origin of the predecessor cell by checking if $off'$ is consistent with the expected predecessor in the $M$ matrix, specifically whether $off'$ equals $\widetilde{M}_{z-x,k} + 1$ for a substitution. If this condition is satisfied, the predecessor cell is in the $M$ matrix (i.e., in a $\widetilde{M}$ wavefront), and the algorithm continues the backtrace from this position. If the condition is not true, the predecessor must be in one of the indel wavefronts, either $\widetilde{I}1$ or $\widetilde{D}1$. At this stage, we open two tentative backtrace paths: one assuming the current cell originated from $\widetilde{I}1$, and the other from $\widetilde{D}1$. For these tentative paths, the offset $off'$ obtained from `rev_extend` may not correspond to the original offset of $\widetilde{M}z - o_1 - e_1, k$. The real original offset could be any value between $off'$ and $off$, since the insertion or deletion may have occurred in the middle of a chain of consecutive matches along the diagonal. Knowing this, the algorithm increases the gap length $l$ iteratively, computing the expected score $z'$ and checking for a consistent path back to $M$. In the case of insertions, this is done by verifying if $off'$ is less than or equal to $\widetilde{M}_{z',k-l} + l$ and less than or equal to $off$. For deletions, the check is if $off'$ is less than or equal to $\widetilde{M}_{z',k+l}$ and less than or equal to $off$. When one of these criteria is satisfied, the corresponding path is confirmed as the correct backtrace path, and the other path is discarded. The algorithm then resumes the backtrace from the identified $M$ cell. The pseudocode for the Singletrack backtrace algorithm in WFA for gap-affine is presented in Algorithm 4.4.

We reuse the example from the main text to demonstrate the Singletrack backtrace in WFA. Figure 4.3 shows the DP cells and wavefronts computed by WFA for the alignment of sequences $q =$ `GCA` and $t =$ `GCCAA`. The $\widetilde{I}1$ and $\widetilde{D}1$ wavefronts are shown to provide full context for the example, although they are never used by the Singletrack backtrace algorithm itself. The alignment begins at cell $M_{3,5}$ (the current cell), which corresponds to $\widetilde{M}_{10,2} = 5$. Here, $z = s = 10$, and the diagonal and offset are $k = 2$ and $off = 5$, respectively. Since we start in $M$, the Singletrack algorithm performs the `rev_extend` operation. In this case, there is one match (the letter `A`) on diagonal $k = 2$ starting at $off = 5$, so $off' = 4$. However, the cell preceding $\widetilde{M}_{10,2}$ before executing the `extend` operation was not generated from a substitution, since $off' \neq \widetilde{M}_{z',2}+1$, with $z' = z-x = 6$. Therefore, the predecessor cell must be in either $I1$ or $D1$. At this point, the gap length $l$ is initialized to 0 and then increased to 1, and two tentative paths have been opened. The following conditions are then checked: $off' \leq \widetilde{M}_{z',k-l}+l \leq off$ for the insertion path, and $off' \leq \widetilde{M}_{z',k+l} \leq off$ for the deletion path, where $z' = z - o_1 - l \cdot e_1 = 10 - 6 - 1 \cdot 2 = 2$. None of the conditions is true, so the gap length $l$ is increased to 2. In this case, the condition $off' \leq \widetilde{M}_{z',k-l} + l \leq off$, where $z' = z - o_1 - l \cdot e_1 = 10 - 6 - 2 \cdot 2 = 0$ is satisfied, indicating that the optimal path to $M_{3,5}$ goes through $M_{2,2}$, $I1_{2,3}$, and $M_{2,4}$. At this stage, a match and two insertions are added to the CIGAR string, which is initialized as `IIM`. Upon reaching cell $M_{2,2}$ with score $z = 0$, the condition of the `while` loop is no longer satisfied, and the loop exits. Since $off > 0$, there remain $off$ matches to be added to the CIGAR string, which are appended before completing the backtrace process. The final CIGAR string is thus `MMIIM`.




(a) DP matrix

(b) Wavefronts

Figure 4.3.: Computed cells of the DP matrix (a) and wavefronts (b) generated by WFA during the alignment of sequences $q =$ `GCA` and $t =$ `GCCAA` using the gap-affine cost function with penalties $a = 0$, $x = 4$, $o_1 = 6$, and $e_1 = 2$. DP matrix cells marked with an asterisk (*) indicate cells that are computed but not stored due to the `extend` operation. Empty cells in the wavefronts represent non-existent DP cells; that is, for a given score $z$ and diagonal $k$, no DP cell exists with that score on that diagonal. Note that only a subset of the $\widetilde{I}1$ and $\widetilde{D}1$ wavefronts needs to be stored during alignment, as the Singletrack backtrace algorithm requires only the $\widetilde{M}$ wavefronts. Dark-shaded cells indicate matrix entries that are part of the optimal alignment path.

We have integrated Singletrack into WFA2-lib[2], which implements the Wavefront Alignment Algorithm and is utilized in several state-of-the-art genome analysis tools, including Anchorwave [251] and wfmash [314]. WFA2-lib explicitly stores all the wavefronts to perform the backtrace (i.e., for each score $z$, three wavefronts in the case of gap-affine alignment and five wavefronts in the case of dual gap-affine alignment).

In our version of WFA2-lib, WFA+Singletrack, only the $\widetilde{M}$ wavefronts are stored and used for the backtrace. For the indel matrices, we allocate a minimal scope of $N$ wavefronts at the start of the alignment, each wavefront with the maximum possible size that may be required during the alignment. These $N$ wavefronts are initially assigned to scores $z = 0, \ldots, N-1$. As the alignment progresses and a wavefront's score moves out of scope, its memory is recycled for higher scores; i.e., the wavefronts within the scope are cycled, ensuring that no additional memory is required for the indel matrices. For sufficiently large sequences, WFA+Singletrack reduces memory usage by a factor of 3× for gap-affine and 5× for dual gap-affine alignments compared to the original WFA implementation in WFA2-lib. The modified version of WFA2-lib supporting Singletrack is available at https://github.com/LorienLV/singletrack.

## 4.6. Experimental Setup

We evaluate the performance and memory consumption of KSW2+Singletrack and WFA+Singletrack, introduced in previous sections, and compare them with their original implementations. Additionally, we evaluate the divide-and-conquer version of WFA (BiWFA) as a state-of-the-art reference alignment algorithm with linear memory complexity. The classical dynamic programming implementation (with and without Singletrack) is omitted from the results due to its substantially higher computational cost compared to the optimized KSW2 and WFA algorithms. All algorithms are executed without heuristics and therefore always produce optimal alignments.

For the evaluation, we configure all implementations to compute global alignments using the default penalties of BWA-MEM [196] and WFA2-lib. When computing gap-affine alignments, penalty values are set to $a = 0$, $x = 4$, $o_1 = 6$, and $e_1 = 2$. For dual gap-affine alignments, penalties are set to $a = 0$, $x = 4$, $o_1 = 6$, $e_1 = 2$, $o_2 = 24$, and $e_2 = 1$.

It is important to note that the chosen penalties do not affect the alignment time of KSW2 and KSW2+Singletrack, since the full DP matrices are computed in both cases. While the penalties can slightly increase traceback time, this overhead is negligible for both algorithms. In contrast, the chosen penalty values affect the execution time and memory usage of the WFA and BiWFA algorithms and, consequently, WFA+Singletrack. However, the relative increase in the number of computed cells, and therefore in memory consumption and execution time, remains the same across all these algorithms.

We evaluate our method on three datasets obtained from real sequencing experiments. For each dataset, the error rate is defined as the proportion of edit operations required to optimally align the query sequence into the target sequence. For example, an error

[2] https://github.com/smarco/WFA2-lib

rate of 15% for a query and target of length 100 indicates that 15 edit operations (i.e., mismatches, insertions, and deletions) are required.

The first, Illumina WGS, was obtained from NIST's Genome in a Bottle (GIAB) project[3] and comprises 100 million sequence pairs with lengths ranging from 140 bp to 280 bp (average 248.38 bp) and an average error rate of 1%. The second, PacBio HiFi, was obtained from the Precision FDA Truth Challenge V2[4] and contains 10 million sequence pairs ranging from 0.20 Kbp to 24.64 Kbp (average 12.87 Kbp) and an average error rate of 0.6%. The third dataset, ONT PromethION, was obtained from the Human Pangenome Reference Consortium [325] and comprises 1,312 sequence pairs with lengths of up to 309.28 Kbp (average 173.27 Kbp). For our experiments, we select only the pairs up to 50 Kbp in length, resulting in 18 sequences with an average length of 27.60 Kbp and an average error rate of 12%. While the full ONT dataset was preliminarily tested, only BiWFA could complete the alignments, as the other algorithms exceeded the compute node's memory capacity. Consequently, our evaluation focuses on the subset of sequence pairs that can be aligned by all methods under comparison, ensuring a fair and informative comparison. It is important to note that aligning such long sequences (i.e., $>$ 50 Kbp) is relatively uncommon in practice, as most mappers, such as Minimap2 [198], typically use a seed-and-extend approach that requires aligning much smaller sequence chunks (e.g., anchors or seeds) rather than full-length sequences. The curated datasets are available in Zenodo (DOI:10.5281/zenodo.17525720).

All the experiments were conducted on a computing node equipped with two Intel Xeon Platinum 8480+ processors, each featuring 56 cores and 112 threads, and 256 GiB of main memory. The main specifications of the node are summarised in Table 4.1.

Table 4.1.: Main features of the computing nodes used to execute the experiments.

| | |
|---|---|
| Processor | 2 × Intel Xeon Platinum 8480+ |
| Cores | 2 × 56 |
| Frequency | 2 GHz |
| L1 cache (I + D) | 32 KiB + 48 KiB (per core) |
| L2 cache | 2 MiB (per core) |
| LLC | 2 × 105 MiB (shared) |
| Main Memory | 256 GiB DDR5 (16×16 GiB@4800 MHz) |

## 4.7. Evaluation

### 4.7.1. Single-Threaded Results

In this section, we evaluate the single-threaded performance of the original KSW2, WFA, and BiWFA implementations, compared with the Singletrack-improved variants introduced in this thesis, KSW2+Singletrack and WFA+Singletrack, in terms of time and memory usage. Performance is evaluated by measuring the total execution time required

[3] https://github.com/genome-in-a-bottle/giab_data_indexes
[4] https://precision.fda.gov/challenges/10/intro

to align complete datasets (including the backtrace phase), while memory consumption is measured as the peak resident set size (RSS) recorded during the alignment process using the GNU `/usr/bin/time` command. Table 4.2 summarizes the execution times and memory footprints for all five implementations using both gap-affine (aff) and dual gap-affine (2aff) cost functions. To enable comparison across datasets of varying sizes, the alignment times are normalized by the number of base pairs. For each dataset and cost function, the best-performing algorithm for each metric (performance or memory) is highlighted in bold.

Table 4.2.: Single-threaded runtime and memory usage of KSW2, KSW2+ST, WFA, BiWFA, and WFA+ST (aff = gap-affine; 2aff = dual gap-affine; ST = Singletrack)

| | | Time ($\mu$s) / Kbp | | | Mem (MiB) | | |
|---|---|---|---|---|---|---|---|
| | | Illumina | PacBio HF | PromethION | Illumina | PacBio HF | PromethION |
| aff | KSW2 | 65.8 | 2,980.6 | 7,849.8 | 3.8 | 10,463.6 | 4,534.5 |
| | KSW2+ST | 60.8 | 3,634.5 | 10,063.8 | 4.0 | 21,864.9 | 9,065.2 |
| | WFA | 4.8 | 13.3 | 7,447.2 | 5.3 | 427.9 | 6,337.0 |
| | BiWFA | 8.9 | 18.6 | **4,428.1** | **3.2** | **15.1** | **17.3** |
| | WFA+ST | **3.4** | **10.9** | 5,233.2 | 26.0 | 179.5 | 2,123.8 |
| 2aff | KSW2 | 110.5 | 4,887.9 | 12,479.2 | 3.8 | 10,463.6 | 4,537.9 |
| | KSW2+ST | 77.7 | 4,108.7 | **11,472.8** | 4.0 | 21,864.9 | 9,066.2 |
| | WFA | 11.3 | 50.9 | 44,884.7 | 6.8 | 2,557.0 | 37,653.1 |
| | BiWFA | 28.5 | 172.8 | 35,223.5 | **6.0** | **58.8** | **115.4** |
| | WFA+ST | **5.4** | **39.7** | 28,782.6 | 62.4 | 543.8 | 7,546.1 |

KSW2+Singletrack requires twice the memory of the original KSW2 for both gap-affine and dual gap-affine implementations, as it stores the $\Delta H$ and $\Delta V$ matrices instead of the backtrace matrix used in KSW. However, KSW2+Singletrack eliminates all computations and memory accesses related to constructing this backtrace matrix. This change leads to performance improvements on the Illumina dataset when using the gap-affine cost function. When aligning short sequences, both implementations have a similar memory footprint, as the total memory usage is dominated by fixed overheads such as auxiliary structures and allocation thresholds. In contrast, aligning the PacBio HF and PromethION datasets results in a slowdown of 1.2× on average due to increased memory pressure, highlighting the critical importance of memory efficiency when scaling to long sequences. However, when computing dual gap-affine (2aff), which is the primary alignment mode in Minimap2 [198], the memory requirements for KSW2+Singletrack remain the same as in the case of gap-affine (aff). For the baseline KSW2, constructing the backtrace matrix requires more instructions due to the additional matrices involved in dual gap-affine. As

a result, KSW2+Singletrack achieves consistent speedups on all three datasets, ranging from 1.1× to 1.4×, while presenting a simpler implementation and minimal integration overhead.

Among all the studied algorithms, KSW2+Singletrack outperforms KSW2 in four of the six experiments in terms of execution time, consuming roughly twice as much memory in the runs with the PacBioHF and PromethION datasets. Furthermore, KSW2+Singletrack demonstrates performance advantages over BiWFA, WFA, and WFA+Singletrack on the PromethION dataset under dual gap-affine cost function, while also exhibiting lower memory usage than WFA in that same experiment.

Regarding WFA+Singletrack, it outperforms the original WFA, achieving 1.2–2.1× speedups and reaching the theoretical 3× and 5× memory reductions for gap-affine (aff) and dual gap-affine (2aff), respectively, on the dataset with the longest sequences (PromethION). It should be emphasized that WFA+Singletrack does not use the same amount of memory in gap-affine and dual gap-affine, since in the latter it explores a larger number of cells in the dynamic programming (DP) matrix, i.e., the $\widetilde{M}$ wavefronts require more memory.

In terms of execution time improvement, note that WFA+Singletrack repeatedly reuses the memory allocated to the indel matrices, reducing cache misses compared to WFA, especially for larger sequences. By examining the memory usage reported in the tables, it is clear that the alignment data exceeds the capacity of both the L1D cache (48 KiB) and the L2 cache (2 MiB) across all datasets and algorithm variants. Therefore, reducing cache misses has a significant positive impact on performance.

In terms of memory consumption, WFA+Singletrack uses less memory than WFA, except for the Illumina dataset. The difference in memory usage increases as the sequence length grows, eventually reaching theoretical maximum reductions of 3× and 5×. For Illumina reads, WFA+Singletrack over-allocates memory for indel matrices assuming worst-case usage, while most of it remains unused. Despite this, it outperforms WFA 1.3× for gap-affine and 2.1× for dual gap-affine, justifying the trade-off given the low absolute memory cost.

The linear-memory BiWFA algorithm also benefits from efficient memory reuse, leading to fewer cache misses than the original WFA. However, BiWFA's divide-and-conquer approach introduces additional overhead by recursively recomputing already-processed alignment regions. This overhead translates to WFA+Singletrack outperforming BiWFA on the Illumina and PacBio HiFi datasets for gap-affine and in all three datasets for dual gap-affine. Across the reported results, WFA+Singletrack is up to 5.2× faster than BiWFA. Although BiWFA consistently exhibits the lowest memory consumption across all datasets, the substantial performance improvements offered by WFA+Singletrack make it a compelling choice for moderately long sequences, where memory usage remains manageable.

Among all the studied algorithms, WFA+Singletrack demonstrates better performance and memory efficiency than WFA in all experiments. It also outperforms BiWFA in four of the six experiments, though at the cost of higher memory usage. Compared to KSW2 and KSW2+Singletrack, WFA+Singletrack delivers better performance in five of the six experiments, only being outperformed by KSW2+Singletrack on the PromethION dataset

under gap-affine, while still using less memory.

Overall, in five of the six experiments, the top-performing algorithm is one that uses Singletrack (either KSW2+Singletrack or WFA+Singletrack). Notably, Singletrack consistently improves the memory efficiency of WFA, while in KSW2 it strikes a balance between the aggressive backtrace matrix approach of KSW2 and the original Suzuki-Kasahara implementation, providing performance benefits for short to moderately large sequences.

### 4.7.2. Multi-Threaded Results

Modern sequence alignment tools are frequently deployed in multi-threaded settings to take advantage of today's multi-core architectures. This is especially true in large-scale applications such as read mapping. In these contexts, scalability across multi-core computing nodes becomes a key performance metric. In this section, we evaluate the parallel performance of KSW2, KSW2+Singletrack, WFA, BiWFA, and WFA+Singletrack when executed across multiple cores on our computing node.

To mitigate the effects of load imbalance, which are not the focus of this study and specifically affect the PromethION dataset, each core independently aligns the entire dataset. Execution time is calculated as the average runtime across all threads and normalized by dividing by the number of threads, yielding the scaled time. To maintain a consistent system load, once a core completes its alignment, it immediately begins a new alignment of the dataset if other cores are still processing their first alignment. This process continues until every core has completed at least one full alignment. Only the execution time for the first alignment on each core is included in the analysis. Multi-threaded speedup is then computed relative to single-threaded BiWFA performance by dividing the single-threaded BiWFA execution time (reported in the previous section) by the scaled time observed for each algorithm using $N$ threads. BiWFA serves as the baseline for speedup calculations because it consistently completes all test cases without exhausting memory, whereas KSW2, KSW2+Singletrack, WFA, and WFA+Singletrack run out of memory in certain scenarios.

Figure 4.4 shows the speedup of each algorithm relative to BiWFA single-threaded execution for thread counts of 1, 14, 28, 42, 56, 70, 84, 98, and 112. In this figure, rows correspond to datasets, and columns to cost functions. That is, each subplot corresponds to a dataset-function pair. Note that the y-axis range varies across subplots. The computing node consists of two chips: thread counts up to 56 utilize a single chip, while higher thread counts engage both chips. This architectural transition is indicated by a thick vertical line at 56 threads. In some cases, the speedup curves end before reaching 112 threads because the algorithm exhausts the node's available memory. All algorithms, however, always find the optimal alignment as long as they do not run out of memory.

Since each core performs the same workload as in the single-threaded benchmarks, total memory usage at a given thread count can be precisely computed by multiplying the per-thread memory reported in Table 4.2 by the number of threads. For completeness, Figure 4.5 reports the resulting total memory footprint for each algorithm and experiment as a function of thread count.

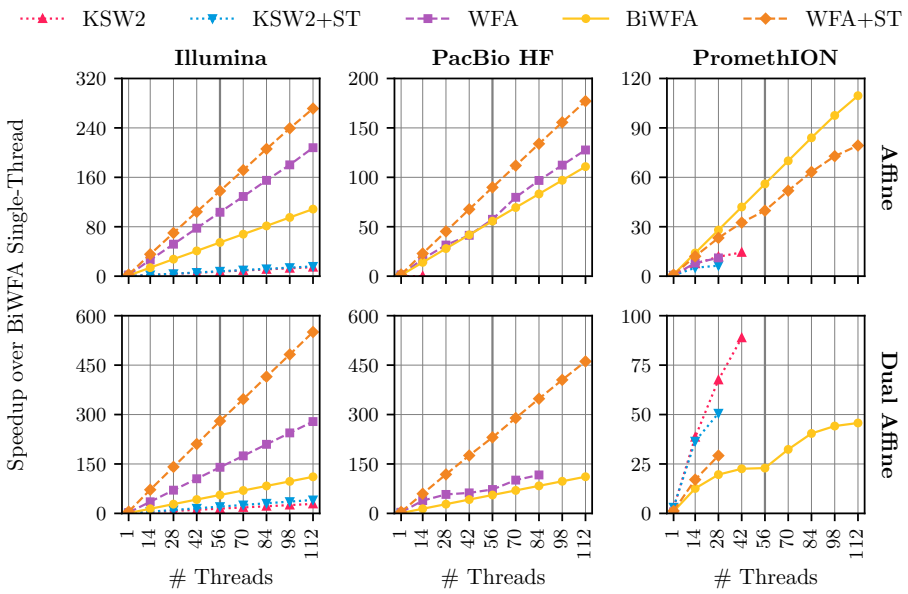



Figure 4.4.: Multi-threaded speedups of KSW2, KSW2+ST, WFA, BiWFA, and WFA+ST relative to single-threaded BiWFA (ST = Singletrack).

The parallel performance of each algorithm is influenced by the use of shared resources, such as the last-level cache (LLC) and main memory. When alignment tasks fit entirely within the private L1 and L2 caches of each core, the algorithms achieve ideal speedups: the scaled execution time with $N$ threads matches that of single-threaded execution. This is the case for all algorithms on the Illumina dataset, which comprises short sequences.

As memory requirements increase, particularly with the dual gap-affine cost function, WFA, KSW2, and KSW2+Singletrack increasingly rely on shared resources, causing deviations from ideal performance. In contrast, both BiWFA and WFA+Singletrack maintain ideal or near-ideal scaling in most cases due to their more efficient utilization of the memory hierarchy. Consequently, and consistent with the single-threaded performance results discussed previously, WFA+Singletrack outperforms both BiWFA and WFA in the majority of executions in which it does not exhaust the available memory. Notably, WFA+Singletrack's efficient memory usage enables higher thread counts on large datasets than WFA. For example, when aligning the PacBio HiFi dataset using the dual-gap affine scoring, WFA+Singletrack can utilize all 112 cores on the node, whereas WFA exhausts available memory beyond 84 threads.

KSW2+Singletrack achieves better parallel performance than KSW2 for the Illumina experiments in both gap-affine and dual gap-affine alignments, as well as for the PacBio HiFi experiments under dual gap-affine (before exhausting memory), consistent with the single-threaded results. It also outperforms KSW2 in the PromethION experiments under dual gap-affine at low thread counts. In contrast, KSW2 shows better parallel performance than KSW2+Singletrack for the PacBio HiFi and PromethION datasets under gap-affine and for the PromethION dataset under dual gap-affine, across 14–42 threads. Notably,

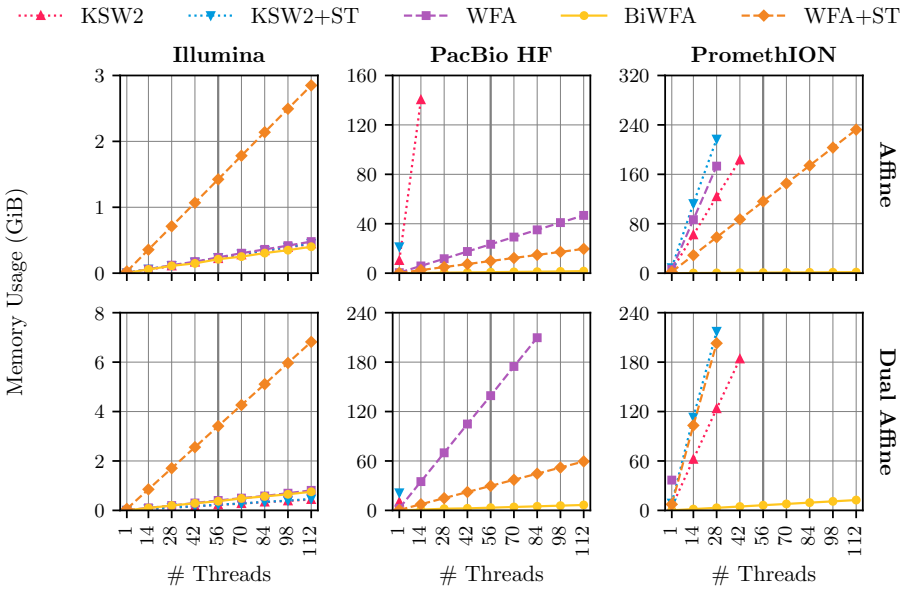



Figure 4.5.: Multi-threaded memory usage in GiB of KSW2, KSW2+ST, WFA, BiWFA, and WFA+ST (ST = Singletrack).

either KSW2 or KSW2+Singletrack is the best-performing algorithm for PromethION under dual gap-affine before reaching the node's memory limit. While KSW2's lower memory requirements allow it to reach higher thread counts than KSW2+Singletrack, the memory demands of both algorithms ultimately constrain their scalability, leading to memory exhaustion at low thread counts for the PacBio and PromethION datasets.

For the largest datasets, the speedup curves display two distinct plateaus: the first occurs when the shared resources of a single chip are fully utilized (up to 56 cores), and the second occurs when the resources of both chips are fully utilized (beyond 56 cores). This behavior is particularly evident for BiWFA with the dual gap-affine cost function on the PromethION dataset, where frequent access to shared resources constrains speedups.

At equivalent thread counts, WFA+Singletrack achieves up to a 3.2× speedup over WFA and a maximum speedup of 5.2× over BiWFA. Meanwhile, KSW2+Singletrack reaches speedups of up to 1.4× over KSW2 at low thread counts, while KSW2 exhibits better scalability and is thus preferable for fully loaded nodes. On the other hand, BiWFA shows better parallel performance than all other algorithms on the PromethION dataset under gap-affine scoring and can utilize all available cores in dual gap-affine mode. Consequently, algorithms implementing Singletrack are best suited for short to moderately large sequences.

## 4.8. Conclusion

In this chapter, we present Singletrack, a general backtrace algorithm that enables the computation of optimal gap-affine and dual gap-affine alignments with low memory

requirements (comparable to gap-linear alignments), without adding alignment overheads, and with a simple, straightforward integration into existing tools.

We demonstrate that Singletrack reduces memory consumption in both the Suzuki-Kasahara (SK) and Wavefront Alignment (WFA) algorithms. For SK, memory usage is reduced by 2× for gap-affine and 4× for dual gap-affine alignments. For WFA, the reductions are 3× and 5×, respectively.

We implement Singletrack in the KSW2 and WFA2-lib libraries. In KSW2, replacing the backtrace matrix approach with Singletrack results in up to 1.4× speedups at the cost of roughly doubling memory usage; this trade-off is consistently beneficial under dual gap-affine across all sequence lengths, and under gap-affine for short sequences, whereas the original KSW2 is preferable for sequences where the additional memory would otherwise prevent alignment or saturate memory bandwidth (either due to long sequences or multiple alignments in parallel). In WFA2-lib, Singletrack provides speedups of 1.2–2.1× over WFA with reduced memory in all scenarios, making it the preferred choice. Compared to BiWFA, WFA with Singletrack achieves up to 5.2× higher performance and is the better option for short to moderately long sequences; BiWFA remains preferable only when memory is severely constrained.

In summary, our method offers a practical solution to overcome the memory limitations of gap-affine and dual gap-affine sequence alignment, while also providing additional performance improvements during alignment. We expect Singletrack to contribute to the development of more efficient and scalable bioinformatics tools that can handle large-scale sequencing datasets in the future.

## 4.9. Future Work

While this chapter demonstrates the successful integration of Singletrack into the widely used KSW2 and WFA2-lib libraries, its minimally invasive design allows for straightforward integration into other alignment frameworks with minimal engineering effort. A natural continuation of this work would be to integrate and evaluate it in additional high-performance libraries, such as Parasail [318], or to extend its application to Multiple Sequence Alignment (MSA), sequence-to-graph, and sequence-to-tree aligners. Notably, the Singletrack traceback has already been incorporated into Theseus [326], a sequence-to-graph aligner targeting pangenomes where memory requirements are typically more demanding due to the complexity of graph-structured references.

Beyond CPU-based aligners, Singletrack is also suitable for hardware accelerators. In the KSW2 integration, Singletrack removes the control logic required to compute the backtrace matrix during the alignment step, resulting in simpler and more regular compute kernels that are generally easier to implement and optimize on accelerators such as GPUs and FPGAs. In the WFA integration, Singletrack significantly reduces memory usage in all cases without modifying the alignment step, which is directly beneficial on memory-constrained accelerators both in terms of capacity and bandwidth. Since Singletrack does not alter the alignment algorithm itself, it can be integrated into existing accelerator designs without redesigning the compute kernels.

# Chapter 5.

# Conclusion and Future Work

## 5.1. Conclusion

In this thesis, we propose approaches to improve the efficiency of molecular dynamics (MD) and genomic data analysis by designing new hardware-aware algorithms and optimizing existing algorithms for the underlying hardware.

**First Contribution.** We introduce the ILVES family of parallel algorithms for enforcing constraints in molecular dynamics simulations. The ILVES algorithms achieve high accuracy at a lower cost than state-of-the-art methods such as SHAKE and (P-)LINCS. We demonstrate that high accuracy in MD is essential to mitigate the accumulation of numerical errors over time, which otherwise introduce non-negligible drifts in physical quantities and compromise simulation stability. Furthermore, ILVES extends the scope of constraint solvers by enabling the efficient constraint of additional degrees of freedom, such as bond angles, which were previously computationally prohibitive.

The ILVES family consists of three algorithms: ILVES-PC, ILVES-M, and ILVES-F. All three employ Newton's method and custom linear solvers to achieve rapid convergence. ILVES-PC is tailored specifically for peptides and proteins; it exploits repeating structural patterns to generate optimized, loop-free code that solves the linear systems associated with these recurring molecular structures. ILVES-M generalizes this approach to handle arbitrary molecular structures, providing a fully generic solver at the cost of slightly lower performance than the PC variant. Finally, ILVES-F redefines the underlying equations, achieving rapid linear convergence instead of the quadratic convergence of ILVES-PC and ILVES-M. It exploits the fact that the maximum accuracy is bounded by hardware precision, thereby keeping the required number of iterations comparable to those of ILVES-PC and ILVES-M. By leveraging these modified equations, ILVES-F can utilize a faster custom linear solver, ultimately achieving roughly twice the performance of ILVES-M.

Our results show that ILVES delivers performance comparable to state-of-the-art methods for low-accuracy bond constraints, while providing significant speedups for high-accuracy requirements. Most importantly, by leveraging the rapid convergence of ILVES to constrain additional degrees of freedom, we achieve a throughput increase of 1.65× within Gromacs, an improvement previously unattainable with existing methods.

**Second Contribution.** We introduce the GenArch benchmark suite, comprising 13 computationally demanding kernels extracted from widely used genomic tools. This suite serves as a framework to evaluate the maturity and readiness of the Arm and RISC-V architectures for genomic workloads.

We port and optimize GenArchBench for two Arm-based processors: the Fujitsu A64FX and Amazon's Graviton3. These optimizations leverage Arm's Scalable Vector Extension (SVE), architecture-specific HPC libraries, and microarchitectural tuning. Similarly, we port the kernels to RISC-V, targeting the Allwinner D1 and HiFive Unmatched CPUs using the RISC-V Vector extension. Through this process, we characterize the current limitations of the RISC-V ecosystem in handling HPC-scale workloads.

We perform a deep performance analysis of GenArchBench across Arm, RISC-V, and x86 reference systems. This analysis reveals that the A64FX requires aggressive optimization to exploit its unique microarchitecture and delivers a 2× slowdown relative to x86. However, it remains competitive in energy efficiency under full-core utilization. In contrast, Graviton3 delivers competitive out-of-the-box performance, establishing itself as a viable alternative to consolidated x86 platforms. While the RISC-V ecosystem is evolving rapidly, our evaluation indicates that current hardware and toolchains are not yet competitive in the HPC landscape. Furthermore, we find that performance and energy monitoring support for both Arm and RISC-V lags behind that of x86 competitors, particularly Intel, which provides the most transparent telemetry to users.

**Third Contribution.** We present Singletrack, a novel backtrace algorithm for gap-affine and dual gap-affine sequence alignment. While standard formulations require storing multiple matrices to reconstruct the optimal alignment, Singletrack enables the alignment step to store a single matrix at the cost of increasing the backtrace time. However, since the backtrace step has linear time complexity $\mathcal{O}(n)$ rather than the quadratic complexity $\mathcal{O}(n^2)$ of the alignment phase, the total execution time is largely unaffected by the increase in backtrace time.

By integrating Singletrack into the KSW2 and WFA2-lib libraries, we demonstrate that the algorithm reduces memory consumption and improves performance by alleviating pressure on the memory hierarchy. Our Singletrack-enabled version of KSW2 achieves speedups of up to 1.4×, while WFA+Singletrack delivers speedups between 1.2–2.1× alongside up to a 5× reduction in memory usage.

All the contributions of this thesis are grounded in the same hardware-aware design philosophy, in which we design solutions that take into account the underlying hardware, rather than algorithms that ignore the machine on which they ultimately run. On one side lies the problem to be solved, together with its specific characteristics, such as the linear topology of a molecule or the asymmetry between a costly alignment phase and an inexpensive backtrace. On the other side lie the features of the hardware, such as its finite numerical precision, its memory hierarchy, or its vector units, which determine how each algorithmic strategy translates into performance. By considering both sides together, we are able to exploit trade-offs that are not apparent when either is considered in isolation:

a method with slower convergence than another can be faster once we account for the maximum precision attainable by the hardware; a fully vectorized exact solution can outperform a scalar heuristic one; and reducing memory consumption relieves pressure on the memory hierarchy, which in turn improves performance.

This co-design relies on support from both software and hardware. On the software side, it requires a mature toolchain: compilers and optimized libraries that exploit the hardware's capabilities, and profiling tools that help developers understand how those capabilities are being used. On the hardware side, it requires architectures whose design is guided by the demands of the real applications they are meant to run, and which expose performance and energy counters that make their behavior measurable.

As each new processor generation increasingly demands software tailored to make full use of its resources, hardware-aware design becomes essential for high-performance scientific computing.

## 5.2. Future Work

All three contributions presented in this thesis offer potential avenues for future research.

**First Contribution.** A promising direction is to generalize the ILVES-PC approach by developing a framework that dynamically detects repeated subblocks in the linear system of constraints and generates optimized code at runtime. This would increase applicability while improving maintainability.

All ILVES variants presented in this work target CPU architectures. In recent years, molecular dynamics packages have increasingly prioritized GPU execution, making a GPU port of ILVES a natural next step. A straightforward strategy would be to follow an approach similar to that currently used by the P-LINCS GPU implementation in GROMACS. We expect ILVES to keep gains comparable to those observed on CPUs. Moreover, existing GPU constraint solvers typically work only on bond-length constraints involving hydrogen atoms, since constraining additional degrees of freedom requires substantial inter-thread communication. Researching alternative approaches to efficiently support additional degrees of freedom on GPUs remains an open and relevant line of research.

Concluding the first contribution, we have shown that ILVES enables the efficient constraining of additional degrees of freedom beyond bond lengths. However, current molecular dynamics packages do not yet support this capability, largely because it was previously impractical. Identifying which additional degrees of freedom are most advantageous to constrain to enhance performance without compromising stability remains an important direction for future research.

**Second Contribution.** We have evaluated the behavior of GenArchBench across several Arm, RISC-V, and x86 systems and identified clear opportunities for further improvement on all three architectures. For instance, some kernels are only partially vectorized or remain entirely scalar.

The toolchains for both Arm and RISC-V offer significant opportunities to improve support for the efficient porting and optimization of general HPC workloads. These improvements include support from compilers, profiling tools, and external libraries. On the hardware side, extending the range of available performance counters and exposing energy-consumption data to users would further strengthen the ecosystem.

At the time of writing this thesis, new systems are emerging across all three studied architectures. Analyzing these platforms would provide valuable insight into how each architecture ecosystem is evolving and how well they adapt to the demands of modern workloads.

**Third Contribution.** Finally, we have demonstrated that Singletrack is an effective method for reducing memory consumption and achieving performance gains by alleviating pressure on the memory hierarchy, all while requiring only minimal code changes. While in this thesis we have integrated Singletrack into two widely used libraries, KSW2 and WFA2-lib, extending this to additional high-performance sequence aligners or even sequence-to-graph aligners remains a clear next step. Furthermore, integrating Singletrack on hardware accelerators, such as GPUs or FPGAs, and evaluating its synergy with existing memory-reduction techniques, such as banding, represents a promising direction for future research.

## 5.3. Conclusiones

En esta tesis, hemos propuesto distintos métodos para mejorar la eficiencia de las simulaciones de dinámica molecular y el análisis de datos genómicos siguiendo una estrategia de diseño y optimización consciente del *hardware* subyacente.

**Primera contribución.** Hemos presentado la familia de algoritmos paralelos ILVES para la imposición de restricciones en simulaciones de dinámica molecular. Los algoritmos ILVES alcanzan una alta precisión a un coste computacional inferior al de los métodos del estado del arte, como SHAKE y (P-)LINCS. Hemos demostrado que alcanzar una alta precisión en simulaciones de dinámica molecular es esencial para mitigar la acumulación de errores numéricos a lo largo del tiempo, que, de otro modo, introducen desviaciones no despreciables en las magnitudes físicas y comprometen la estabilidad de las simulaciones. Además, ILVES es capaz de imponer restricciones en grados de libertad adicionales que los algoritmos actuales no pueden manejar de forma eficiente.

La familia ILVES comprende tres algoritmos: ILVES-PC, ILVES-M e ILVES-F. Los tres emplean el método de Newton y solucionadores de sistemas lineales de ecuaciones *ad hoc* para lograr una convergencia rápida y eficiente. ILVES-PC está diseñado específicamente para péptidos y proteínas, aprovechando los patrones estructurales repetitivos de estas estructuras moleculares para generar código optimizado y sin bucles para resolver sus sistemas lineales asociados. ILVES-M generaliza esta idea para soportar estructuras moleculares arbitrarias, implementando un solucionador completamente genérico a costa de un rendimiento ligeramente inferior al de ILVES-PC. Por último, ILVES-F reformula las

ecuaciones subyacentes, sacrificando la convergencia cuadrática del método de Newton a favor de la convergencia lineal de un método quasi-Newton, pero permitiendo implementar un solucionador de sistemas lineales más rápido. ILVES-F explota el hecho de que la precisión máxima alcanzable está acotada por la precisión del *hardware*, de modo que una convergencia lineal rápida mantiene un número de iteraciones comparable al de ILVES-PC e ILVES-M, lo que se traduce en un rendimiento de aproximadamente el doble que el de ILVES-M.

En nuestros experimentos, hemos mostrado que ILVES ofrece un rendimiento comparable al de los métodos del estado del arte al imponer restricciones en la longitud de los enlaces con baja precisión, y mejora significativamente el rendimiento al hacer lo mismo con alta precisión. Por otra parte, la rápida convergencia de ILVES permite imponer restricciones a grados de libertad adicionales, lo que incrementa el rendimiento de las simulaciones ejecutadas con GROMACS hasta en $1,65\times$, un resultado que los métodos existentes no pueden alcanzar.

**Segunda contribución.** Hemos presentado GenArchBench, una batería de *benchmarks* que comprende 13 núcleos de ejecución computacionalmente exigentes extraídos de aplicaciones ampliamente utilizadas en genómica. GenArchBench sirve como marco de evaluación para determinar la madurez y la competitividad de las arquitecturas Arm y RISC-V para el análisis de datos genómicos.

Hemos portado y optimizado GenArchBench para dos procesadores Arm: el Fujitsu A64FX y el Graviton3 de Amazon. Para ello, explotamos la extensión vectorial escalable (SVE) de Arm, utilizamos librerías de alto rendimiento específicas para la arquitectura y aplicamos optimizaciones basadas en la microarquitectura de cada máquina. Asimismo, portamos los núcleos a RISC-V, con el Allwinner D1 y el HiFive Unmatched como plataformas objetivo, aprovechando la extensión vectorial de RISC-V cuando es posible. A través de este trabajo, evaluamos las limitaciones actuales del ecosistema RISC-V para manejar cargas de trabajo de alto rendimiento.

Hemos realizado un análisis exhaustivo del rendimiento de GenArchBench en los sistemas mencionados anteriormente, junto con dos máquinas x86 de referencia. Este análisis revela que el A64FX requiere optimizaciones agresivas para aprovechar su microarquitectura y presenta una pérdida de rendimiento de $2\times$ con respecto a las máquinas x86. No obstante, el A64FX resulta competitivo en consumo energético cuando se utiliza con carga máxima. Por otro lado, el Graviton3 ofrece un rendimiento competitivo sin necesidad de optimizaciones específicas, consolidándose como una alternativa viable a las plataformas x86 tradicionalmente empleadas para ejecutar aplicaciones genómicas. Si bien RISC-V es una arquitectura considerablemente más reciente que Arm y x86, y ha evolucionado con una rapidez reseñable en los últimos años, nuestra evaluación indica que su *hardware* y su ecosistema de herramientas no son todavía competitivos en el entorno de alto rendimiento. Por último, cabe destacar que el soporte para la monitorización del rendimiento y del consumo energético en Arm y RISC-V está por detrás del de sus competidores x86, especialmente el de Intel, que ofrece la telemetría más transparente y accesible para el usuario.

**Tercera contribución.** Hemos presentado Singletrack, un nuevo algoritmo de *backtrace* para el alineamiento de secuencias con las funciones de coste *gap-affine* y *dual gap-affine*. Mientras que los algoritmos tradicionales de alineamiento de secuencias requieren almacenar múltiples matrices para reconstruir el alineamiento óptimo, Singletrack permite que la fase de alineamiento almacene una única matriz, a costa de incrementar el tiempo del *backtrace*. Sin embargo, dado que la fase de *backtrace* tiene una complejidad temporal lineal $\mathcal{O}(n)$ en lugar de la cuadrática $\mathcal{O}(n^2)$ de la fase de alineamiento, el tiempo total de ejecución permanece en gran medida inalterado por el aumento del tiempo de *backtrace*.

Hemos integrado Singletrack en las librerías KSW2 y WFA2-lib, demostrando que el algoritmo reduce el consumo de memoria y mejora el rendimiento al aliviar la presión sobre la jerarquía de memoria. Nuestra versión de KSW2 con Singletrack alcanza mejoras de rendimiento de hasta 1,4×, mientras que WFA+Singletrack consigue acelerar la implementación original entre 1,2× y 2,1×, junto con una reducción del consumo de memoria de hasta 5×.

Todas las contribuciones de esta tesis se fundamentan en la filosofía de diseño consciente del *hardware*, en la que diseñamos soluciones que tienen en cuenta el *hardware* subyacente, en lugar de algoritmos que ignoran la máquina sobre la que finalmente se ejecutan. Por un lado, se encuentra el problema a resolver junto con sus particularidades, como la topología lineal de una molécula o la asimetría en exigencia computacional entre la fase de alineamiento y la de *backtrace*. Por otro lado, se encuentran las características del *hardware*, como su precisión numérica finita, su jerarquía de memoria y sus unidades vectoriales, que determinan cómo se traduce cada estrategia algorítmica en rendimiento. Al considerar ambas partes de forma conjunta, somos capaces de aprovechar compromisos que no resultan evidentes cuando cada una se analiza de forma aislada: un método con una convergencia más lenta que otro puede ser más rápido una vez se tiene en cuenta la precisión máxima alcanzable por el *hardware*; una solución exacta vectorizada puede ser más rápida que una solución heurística escalar; y reducir el consumo de memoria alivia la presión sobre la jerarquía de memoria, lo que a su vez mejora el rendimiento.

Este codiseño requiere soporte tanto del *software* como del *hardware*. Por el lado del *software*, se requiere un ecosistema de herramientas maduro (compiladores, librerías optimizadas y herramientas de perfilado) que permita a los desarrolladores comprender el *hardware* y explotar sus capacidades. Por el lado del *hardware*, se requieren arquitecturas cuyo diseño esté guiado por las exigencias de las aplicaciones reales que van a ejecutar, y que expongan contadores de rendimiento y de energía que hagan medible su comportamiento.

A medida que cada nueva generación de procesadores exige cada vez más que el *software* se adapte para aprovechar plenamente sus recursos, el diseño consciente del *hardware* se vuelve esencial para la computación científica de alto rendimiento

## 5.4. Trabajo Futuro

Las tres contribuciones presentadas en esta tesis ofrecen posibles vías para futuros proyectos.

**Primera contribución.** Un camino prometedor es generalizar el enfoque de ILVES-PC para que, en lugar de funcionar únicamente con algunas moléculas (péptidos y proteínas, en este caso), detecte programáticamente los bloques que se repiten en un sistema lineal arbitrario y genere código optimizado para cada bloque en tiempo de ejecución. Esto aumentaría su aplicabilidad y facilitaría el mantenimiento del código.

Todas las variantes de ILVES presentadas en este trabajo tienen como objetivo la ejecución en CPU. En los últimos años, los paquetes de dinámica molecular han priorizado cada vez más la ejecución en GPU, lo que hace que portar ILVES a GPU sea el paso natural a seguir. Una estrategia directa sería adoptar un enfoque similar al que utiliza actualmente la implementación de P-LINCS en GPU dentro de Gromacs. Esperamos que ILVES mantenga ganancias con respecto al estado del arte comparables a las observadas en CPU en esta tesis. Asimismo, los algoritmos para imponer restricciones en GPU existentes trabajan únicamente con restricciones en la longitud de enlaces que involucran átomos de hidrógeno, ya que imponer restricciones sobre grados de libertad adicionales requiere comunicación entre hilos, lo cual penaliza gravemente el rendimiento en GPU. Investigar estrategias alternativas para soportar grados de libertad adicionales de forma eficiente en GPU constituye una línea de investigación abierta.

Para concluir con la primera contribución, hemos demostrado que ILVES abre la puerta a imponer restricciones sobre grados de libertad más allá de las longitudes de enlace. Sin embargo, los paquetes de dinámica molecular actuales no incorporan en gran medida esta funcionalidad, ya que los métodos del estado del arte no son capaces de aprovecharla de manera eficiente. Determinar qué grados de libertad adicionales resultan más ventajosos restringir para mejorar el rendimiento sin comprometer la estabilidad de las simulaciones es una dirección importante para la investigación futura.

**Segunda contribución.** Hemos evaluado el comportamiento de GenArchBench en varios sistemas Arm, RISC-V y x86 e identificado oportunidades claras de mejora en las tres arquitecturas. Por ejemplo, algunos núcleos están solo parcialmente vectorizados o son completamente escalares.

El ecosistema de herramientas tanto para Arm como para RISC-V presenta oportunidades de mejora para facilitar la portabilidad y la optimización de cualquier carga de trabajo de alto rendimiento, no solo las genómicas. Estas mejoras incluyen el aumento del soporte por parte de compiladores, herramientas de perfilado y librerías externas. En el plano del *hardware*, ampliar el rango de contadores de rendimiento disponibles y exponer los datos de consumo energético a los usuarios reforzarían aún más estos ecosistemas.

Las tres arquitecturas estudiadas en esta tesis se actualizan constantemente con nuevas generaciones de procesadores. Reevaluar los núcleos de GenArchBench en estas nuevas plataformas proporcionaría una visión valiosa de cómo evoluciona el ecosistema de cada arquitectura y de su capacidad de adaptación a las exigencias de las cargas de trabajo modernas.

**Tercera contribución.** Por último, hemos demostrado que Singletrack es un método eficaz para reducir el consumo de memoria y obtener ganancias de rendimiento al aliviar

la presión sobre la jerarquía de memoria, todo ello con cambios mínimos en el código. En esta tesis hemos integrado Singletrack en dos bibliotecas ampliamente utilizadas, KSW2 y WFA2-lib; integrar este método en otros alineadores de secuencias de alto rendimiento, o incluso en alineadores de secuencia contra grafo, constituye un siguiente paso natural. El uso de Singletrack en aceleradores *hardware*, como GPUs o FPGAs, y la evaluación de su sinergia con técnicas existentes de reducción de memoria, como el uso de bandas, también representan direcciones prometedoras para futuros proyectos.

# Appendix A.

# Further Discussion on the Need to Solve the Constraint Equations Accurately

*This appendix provides additional justification for solving constraint equations to the limit of machine precision. Through numerical experiments with a pendulum model, we demonstrate how insufficient solver tolerance leads to energy conservation violations and unphysical results. We argue that minimizing computational error to the unit roundoff is a prerequisite for reliable model validation and the accurate estimation of discretization errors.*

The main text has established the motivation for accurately solving the constraint equations. In this appendix, we present additional evidence that supports and complements those arguments.

First, let us note that if the constraint equations are solved exactly, then the SHAKE algorithm is second-order accurate in the time step. In practice, it is impossible to solve the constraint equations exactly and we accept an approximation if the relative error is less than the tolerance Tol passed to the constraint solver. In this section, we shall explain why the only safe value of Tol is tiny and why we should run the constraint solver until stagnation. We shall present arguments that should appeal to both the numerical analyst and the experimental scientist who seeks to validate a model of physical reality.

It is important to distinguish between the exact numbers that are defined by the SHAKE algorithm and the numbers that are computed when the algorithm is executed using finite precision arithmetic and tolerance $\text{Tol} > 0$.

Table A.1.: Parameters defining a simple pendulum.

| Parameter | Symbol | Value |
|---|---|---|
| Mass | $m$ | $1\ kg$ |
| Length | $l$ | $1\ m$ |
| Gravitational const. | $g$ | $9.82\ m/s^2$ |
| Initial velocity | $v_0$ | $(1, 0)\ m/s$ |
| Initial position | $r_0$ | $(0, 0)$ |

We begin by demonstrating that if the tolerance parameter Tol is small enough, then the computed numbers behave in a manner that is indistinguishable from the numbers that are defined by the SHAKE algorithm. To this end, we shall study a simple pendulum. This is a standard test problem. The necessary files to execute the calculations and

regenerate our figures from scratch can be found in the public GitHub repository at https://github.com/spockcc/_PAPER_ILVES_pendulum

We shall demonstrate that if the tolerance in solving is small enough, then the total energy is very nearly preserved and the error is $\mathcal{O}(h^2)$. Our pendulum is defined by Table A.1.

Table A.2.: The parameters for the first set of simulations of the pendulum given by Table A.1. The evolution of the total energy is shown in Figure A.1.

| Description | Name | Value |
|---|---|---|
| Number of periods | `np` | 10,000 |
| Number of recorded state vectors per period | `nsp` | 10 |
| Number of recorded state vectors | `N1` | $\texttt{N1} = \texttt{np} \cdot \texttt{nsp}$ |
| Number of steps between recorded state vectors | `N2` | $\{10, 20, 40\}$ |
| Tolerance passed to constraint solver | Tol | $10^{-12}$ |

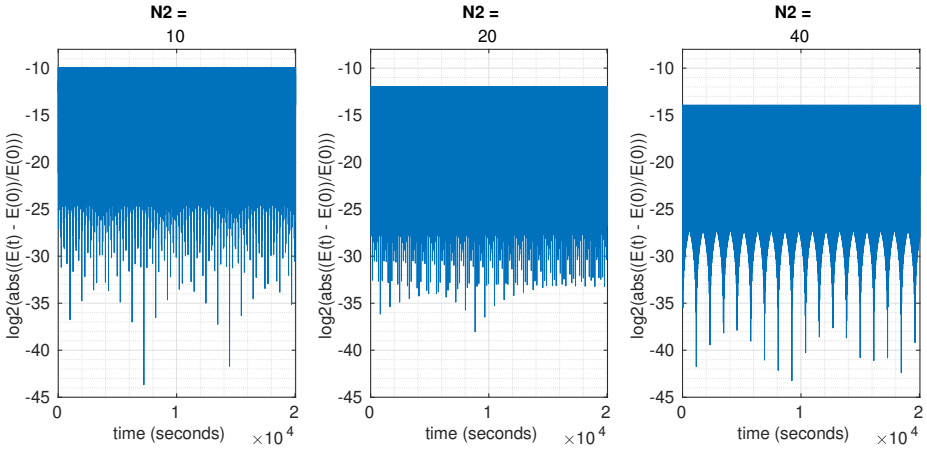



Figure A.1.: Total energy of a simple pendulum as a function of time for three different values of timestep.

The MATLAB function `shake_pendulum` is included in the GitHub mentioned above. It does a sequence of 3 simulations of the same pendulum using different values of the time step and generates Figure A.1. The exact parameters of these simulations are given by Table A.2.

We note that the simulations do $\texttt{nsp} \cdot \texttt{N2} \in \{100, 200, 400\}$ steps per period, which is far above the rule of thumb that suggests at least 5 steps per period. In Figure A.1, we see that the relative error of the total energy, i.e.,

$$R(t) = \frac{E(t) - E(0)}{E(0)} \tag{A.1}$$

is bounded as a function of the time $t$ and that the maximum value of the absolute value of relative error is approximately $\{2^{-10}, 2^{-12}, 2^{-14}\}$ for $\texttt{N2} = \{10, 20, 40\}$. This is

experimental evidence in support of the hypothesis that the maximum relative error is $\mathcal{O}(h^2)$. This is the behavior that we would observe if the algorithm were executed using exact arithmetic and the constraint equations were solved exactly.

Table A.3.: The parameters for the second set of simulations of a pendulum given by Table A.1. The evolution of the total energy is displayed in Figure A.2.

| Description | Name | Value |
|---|---|---|
| Number of periods | `np` | 10,000 |
| Number of recorded state vectors per period | `nsp` | 10 |
| Number of recorded state vectors | `N1` | $\texttt{N1} = \texttt{np} \cdot \texttt{nsp}$ |
| Number of steps between recorded state vectors | `N2` | 10 |
| Tolerance passed to constraint solver | Tol | $\{10^{-2}, 10^{-3}, \ldots, 10^{-6}\}$ |
| Constraint solver | `method` | `quasi` |

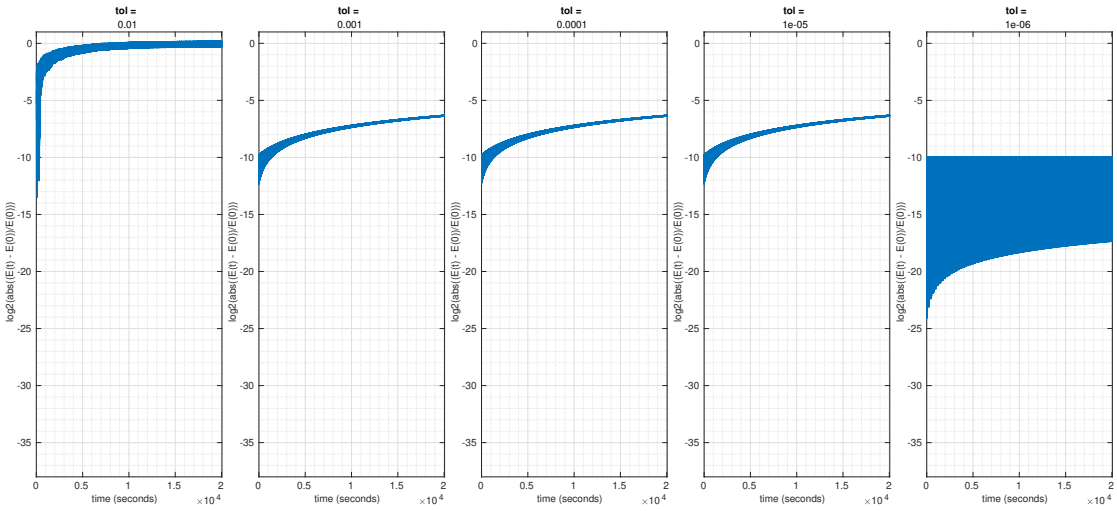



Figure A.2.: The evolution of the total energy of a pendulum over time for different values of the tolerance Tol passed to the constraint solver.

The MATLAB function `shake_pendulum` does a second sequence of 10 simulations of the same pendulum using different values of the tolerance Tol. The exact parameters are given by Table A.3. The constraint solver is a quasi-Newton's method, specifically the simplified Newton's method that uses a fixed value of the Jacobian for each step. The linear convergence of this method ensures that the constraints are solved with a relative error that is essentially equal to the tolerance passed to the solver. Figure A.2 shows how the relative error $R(t)$ of the energy evolves over time for different values of the tolerance Tol. In particular, we see that the relative error is a strictly increasing function of $t$ for Tol$\in \{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$. For these values of the tolerance Tol, we cannot claim that the simulation corresponds to an isolated system because the fundamental principle of conservation of energy is clearly violated. However, for $\tau = 10^{-6}$, the maximum relative error is bounded as a function of $t$. Again, this is exactly the behavior we would observe if

the algorithm were executed in exact arithmetic and the constraint equations were solved exactly.

These two numerical experiments show that if the tolerance parameter Tol is sufficiently small, then the total energy of the pendulum is very nearly preserved, and the computed numbers behave in a manner that is consistent with a method that is second-order accurate in the time step. If the tolerance is not sufficiently small, then the absolute value of the relative error of the energy is an increasing function of time.

In the absence of any deep understanding of the relationship between any system, the time step, the tolerance, and the evolution of the relative error of the energy, the only safe policy is to run the constraint solver until it stagnates due to the limitation of finite precision arithmetic. Naturally, this process is not practical unless the constraint solver converges rapidly.

The MATLAB function `shake_pendulum` also conducts a third sequence of numerical experiments. Here, the focus is on the sign of the relative error rather than on its absolute value. The two experiments reveal that the sign of the relative error can depend on the choice of the constraint solver. The exact parameters of these simulations are given by Table A.4.

Table A.4.: The parameters of a third set of simulations of the pendulum given by Table A.1. The results are shown in Figure A.3.

| Parameter | Name | Value (s) |
|---|---|---|
| Number of periods | `np` | 10,000 |
| Number of recorded state vectors per period | `nsp` | 10 |
| Number of recorded state vectors | `N1` | $\texttt{N1} = \texttt{np} \cdot \texttt{nsp}$ |
| Number of steps between recorded state vectors | `N2` | 10 |
| Tolerance passed to constraint solver | Tol | $10^{-4}$ |
| Constraint solver | `method` | {`newton`,`quasi`} |

Figure A.3 shows the evolution of the sign of the relative error of the energy. The scientist who uses Newton's method to probe the inner mysteries of the pendulum will (almost certainly) conclude that the system loses energy over time, whereas the scientist who uses the simplified Newton's method will conclude that the system gains energy over time. We find it intolerably dangerous that the physical conclusions drawn from a numerical experiment might depend on the algorithm used to solve the central equations.

We shall now argue further in favor of solving the constraint equations as accurately as the hardware allows. To this end, we ask the reader to consider a fundamental question: What is the purpose of an MD simulation? In general, we seek to obtain numbers that cannot be measured in the laboratory either because we lack the equipment or because the procedure is too expensive. However, before these experiments can be conducted, it is necessary to validate our model and determine if it accurately captures the critical aspects of physical reality. In general, we do not care for the solution of a specific initial value problem for the system of differential algebraic equations (DAEs); rather, we seek the value of a physical quantity that can be computed in terms of several such trajectories. Let $T$ denote the value that would be obtained if we could solve the relevant systems

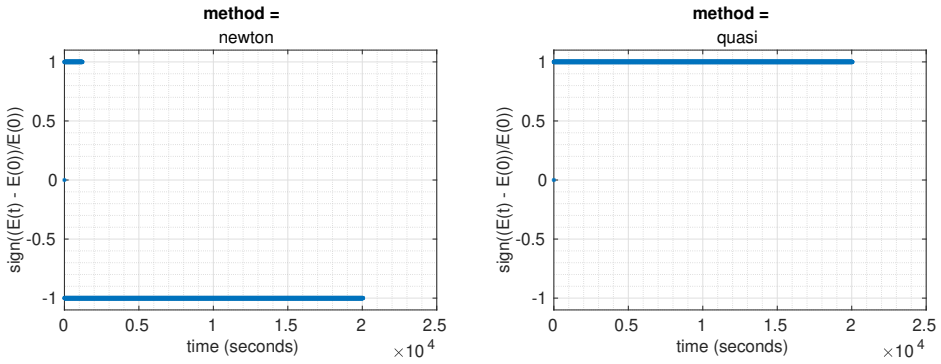



Figure A.3.: Sign of the relative error of the total energy for the pendulum for two different constraint solvers as a function of the time: Newton's method (left) and the simplified Newton's method (right).

of DAEs exactly. Let $A_h$ be the value that would be obtained if we could solve the corresponding discrete-time equations exactly, and let $\hat{A}_h$ denote the value of $A_h$ that is returned by the computer. It is important to recognize that the three numbers $T$, $A_h$, and $\hat{A}_h$ are almost certainly different. The discretization error $T - A_h$ is not zero because we have replaced the system of DAEs with a set of discrete equations. Moreover, the computational error $A_h - \hat{A}_h$ is not zero because we cannot solve the constraint equations exactly (truncation error) and because we are forced to use finite precision arithmetic (rounding error). Why should we care about these errors? We validate our model by comparing $T$ against the physical reality $P$. We cannot compute $T$ directly, and a small value of the observed error $P - \hat{A}_h$ does not necessarily imply that the modeling error $P - T$ is small. Why is that? This issue can be understood using the elementary identity

$$\underbrace{(P - T)}_{\text{modeling error}} = \underbrace{(P - \hat{A}_h)}_{\text{observed error}} - \underbrace{(A_h - \hat{A}_h)}_{\text{computational error}} - \underbrace{(T - A_h)}_{\text{discretization error}} . \tag{A.2}$$

From this identity, we conclude that $T$ is a good approximation of $P$ if the computational error and the discretization error are both negligible compared with a small value of the observed error $P - \hat{A}_h$. It is, therefore, critical to control the size of the computational error and the discretization error. Frequently, but not universally, it is possible to use Richardson extrapolation [50], [51], [327] to estimate the discretization error $T - A_h$ by repeating the same simulation using different values of the time step $h$. In particular, Richardson extrapolation cannot be applied unless the force fields are sufficiently smooth [50], [51]. Moreover, Richardson's error estimate is not reliable unless the computational error $A_h - \hat{A}_h$ is sufficiently small [50], [51]. The size of the computational error $A_h - \hat{A}_h$ is controlled by the size of the unit roundoff $u$ and the tolerance Tol that is passed to the constraint solver. In the absence of any information about the size of the discretization error $T - A_h$, the safest course of action is, therefore, to use the smallest practical value of the unit roundoff $u$ and the tolerance Tol. This choice minimizes the computational error and maximizes the chance that the computational error is so small that we can

reliably estimate the discretization error $T - A_h$. Therefore, it is prudent to favor double precision over single precision and tiny values of Tol over larger values of Tol. In ILVES, we measure the relative constraint violation and iterate until

$$\frac{1}{2}\frac{\|\boldsymbol{q}_{a_i} - \boldsymbol{q}_{b_i}\|_2^2 - d_i^2}{d_i^2} \leq \text{Tol}$$

for all bonds, or when the maximum number of iterations is reached. If the relative constraint violation is small, then

$$\frac{1}{2}\frac{\|\boldsymbol{q}_{a_i} - \boldsymbol{q}_{b_i}\|_2^2 - d_i^2}{d_i^2} = \frac{1}{2}\frac{(\|\boldsymbol{q}_{a_i} - \boldsymbol{q}_{b_i}\|_2 - d_i)(\|\boldsymbol{q}_{a_i} - \boldsymbol{q}_{b_i}\|_2 + d_i)}{d_i^2} \approx \frac{\|\boldsymbol{q}_{a_i} - \boldsymbol{q}_{b_i}\|_2 - d_i}{d_i}. \tag{A.3}$$

On a machine with unit roundoff $u$ we cannot hope to reliably reduce the right-hand side below $u$. Therefore, the smallest value of Tol that should be considered is literally the unit roundoff $\text{Tol} = u$.

In conclusion, we can say the following: If we do not solve the constraint equations accurately, then we cannot claim that our calculations are $m$-th order accurate in the time step, we cannot assume that the fundamental principle of conversation of energy is respected, and we cannot reliably estimate the discretization error and perform the validation needed to develop faith in our model of the physical reality. This strong reason supports solving the constraints as accurately as possible.

# Bibliography


[1] T. Hey, "The fourth paradigm - data-intensive scientific discovery", in *E-Science and Information Management*, Springer Berlin Heidelberg, 2012, pp. 1–1, ISBN: 9783642332999. DOI: 10.1007/978-3-642-33299-9_1.

[2] P. Bauer, A. Thorpe, and G. Brunet, "The quiet revolution of numerical weather prediction", *Nature*, vol. 525, no. 7567, pp. 47–55, Sep. 2015, ISSN: 1476-4687. DOI: 10.1038/nature14956.

[3] E. van der Giessen et al., "Roadmap on multiscale materials modeling", *Modelling and Simulation in Materials Science and Engineering*, vol. 28, no. 4, p. 043 001, Mar. 2020, ISSN: 1361-651X. DOI: 10.1088/1361-651x/ab7150.

[4] H. Kitano, "Computational systems biology", *Nature*, vol. 420, no. 6912, pp. 206–210, Nov. 2002, ISSN: 1476-4687. DOI: 10.1038/nature01254.

[5] Z. D. Stephens et al., "Big data: Astronomical or genomical?", *PLOS Biology*, vol. 13, no. 7, e1002195, Jul. 2015, ISSN: 1545-7885. DOI: 10.1371/journal.pbio.1002195.

[6] D. A. Reed and J. Dongarra, "Exascale computing and big data", *Communications of the ACM*, vol. 58, no. 7, pp. 56–68, Jun. 2015, ISSN: 1557-7317. DOI: 10.1145/2699414.

[7] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions", *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, pp. 256–268, Oct. 1974, ISSN: 1558-173X. DOI: 10.1109/jssc.1974.1050511.

[8] G. E. Moore, "Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp.114 ff.", *IEEE Solid-State Circuits Society Newsletter*, vol. 11, no. 3, pp. 33–35, Sep. 2006, ISSN: 1098-4232. DOI: 10.1109/n-ssc.2006.4785860.

[9] J. Alastruey, J. Briz, P. Ibanez, and V. Vinals, "Software demand, hardware supply", *IEEE Micro*, vol. 26, no. 4, pp. 72–82, 2006. DOI: 10.1109/MM.2006.80.

[10] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture", *Communications of the ACM*, vol. 62, no. 2, pp. 48–60, Jan. 2019, ISSN: 1557-7317. DOI: 10.1145/3282307.

[11] J. L. Hennessy, D. A. Patterson, and C. Kozyrakis, *Computer architecture: A Quantitative Approach*, en. San Francisco, CA: Morgan Kaufmann, Dec. 2025.

[12] H. Sutter et al., "The free lunch is over: A fundamental turn toward concurrency in software", *Dr. Dobb's journal*, vol. 30, no. 3, pp. 202–210, 2005.

[13] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling", *SIGARCH Comput. Archit. News*, vol. 39, no. 3, pp. 365–376, Jun. 2011, ISSN: 0163-5964. DOI: 10.1145/2024723.2000108. [Online]. Available: https://doi.org/10.1145/2024723.2000108.

[14] T. N. Theis and H.-S. P. Wong, "The end of Moore's law: A new beginning for information technology", *Computing in Science & Engineering*, vol. 19, no. 2, pp. 41–50, 2017. DOI: 10.1109/MCSE.2017.29.

[15] K. Asanovic et al., "A view of the parallel computing landscape", *Communications of the ACM*, vol. 52, no. 10, pp. 56–67, Oct. 2009, ISSN: 1557-7317. DOI: 10.1145/1562764.1562783.

[16] S. Mittal and J. S. Vetter, "A survey of CPU-GPU heterogeneous computing techniques", *ACM Computing Surveys*, vol. 47, no. 4, pp. 1–35, Jul. 2015, ISSN: 1557-7341. DOI: 10.1145/2788396.

[17] N. N. Islam et al., "Dynamicasome—a molecular dynamics-guided and ai-driven pathogenicity prediction catalogue for all genetic mutations", *Communications Biology*, vol. 8, no. 1, Jul. 2025, ISSN: 2399-3642. DOI: 10.1038/s42003-025-08334-y. [Online]. Available: http://dx.doi.org/10.1038/s42003-025-08334-y.

[18] L. Casalino et al., "Beyond shielding: The roles of glycans in the sars-cov-2 spike protein", *ACS Central Science*, vol. 6, no. 10, pp. 1722–1734, Sep. 2020, ISSN: 2374-7951. DOI: 10.1021/acscentsci.0c01056. [Online]. Available: http://dx.doi.org/10.1021/acscentsci.0c01056.

[19] S. Portelli et al., "Exploring the structural distribution of genetic variation in sars-cov-2 with the covid-3d online resource", *Nature Genetics*, vol. 52, no. 10, pp. 999–1001, Sep. 2020, ISSN: 1546-1718. DOI: 10.1038/s41588-020-0693-3. [Online]. Available: http://dx.doi.org/10.1038/s41588-020-0693-3.

[20] M. Karplus and J. A. McCammon, "Molecular dynamics simulations of biomolecules", *Nature Structural Biology*, vol. 9, no. 9, pp. 646–652, Sep. 2002, ISSN: 1072-8368. DOI: 10.1038/nsb0902-646.

[21] J. Gelpi, A. Hospital, R. Goñi, and M. Orozco, "Molecular dynamics simulations: Advances and applications", *Advances and Applications in Bioinformatics and Chemistry*, p. 37, Nov. 2015, ISSN: 1178-6949. DOI: 10.2147/aabc.s70333.

[22] G. Hlawacek, P. Puschnig, P. Frank, A. Winkler, C. Ambrosch-Draxl, and C. Teichert, "Characterization of step-edge barriers in organic thin-film", *Science*, vol. 321, no. 5885, pp. 108–111, 2008.

[23] X. Shan et al., "Rational strategies for improving the efficiency of design and discovery of nanomedicines", *Nature Communications*, vol. 15, no. 1, Nov. 2024, ISSN: 2041-1723. DOI: 10.1038/s41467-024-54265-3.

[24] E. Andreano et al., "SARS-CoV-2 escape a highly neutralizing COVID-19 convalescent plasma", *Proceedings of the National Academy of Sciences*, vol. 118, no. 36, 2021.

[25] V. K. Bhardwaj, R. Singh, J. Sharma, V. Rajendran, R. Purohit, and S. Kumar, "Identification of bioactive molecules from tea plant as SARS-CoV-2 main protease inhibitors", *Journal of Biomolecular Structure and Dynamics*, vol. 39, no. 10, pp. 3449–3458, 2021.

[26] W. Yu, S. Kumar, M. Zhao, D. J. Weber, and A. D. MacKerell, "High-throughput ligand dissociation kinetics predictions using site identification by ligand competitive saturation", *Journal of Chemical Theory and Computation*, vol. 21, no. 9, pp. 4964–4978, Apr. 2025, ISSN: 1549-9626. DOI: 10.1021/acs.jctc.5c00265.

[27] D. Lu, D. Luo, Y. Zhang, and B. Wang, "A robust induced fit docking approach with the combination of the hybrid all-atom/united-atom/coarse-grained model and simulated annealing", *Journal of Chemical Theory and Computation*, vol. 20, no. 14, pp. 6414–6423, Jul. 2024, ISSN: 1549-9626. DOI: 10.1021/acs.jctc.4c00653.

[28] J. J. Galano-Frutos, H. García-Cebollada, and J. Sancho, "Molecular dynamics simulations for genetic interpretation in protein coding regions: Where we are, where to go and when", *Briefings in Bioinformatics*, vol. 22, no. 1, pp. 3–19, Dec. 2019, ISSN: 1477-4054. DOI: 10.1093/bib/bbz146.

[29] S. A. Hollingsworth and R. O. Dror, "Molecular dynamics simulation for all", *Neuron*, vol. 99, no. 6, pp. 1129–1143, Sep. 2018, ISSN: 0896-6273. DOI: 10.1016/j.neuron.2018.08.011.

[30] C. N. Kim, D. Shin, A. Wang, and T. J. Nowakowski, "Spatiotemporal molecular dynamics of the developing human thalamus", *Science*, vol. 382, no. 6667, Oct. 2023, ISSN: 1095-9203. DOI: 10.1126/science.adf9941.

[31] C. Gossens, I. Tavernelli, and U. Rothlisberger, "Rational design of organo-ruthenium anticancer compounds", *CHIMIA*, vol. 59, no. 3, p. 81, Mar. 2005, ISSN: 0009-4293. DOI: 10.2533/000942905777676795.

[32] B. Leimkuhler and S. Reich, *Simulating Hamiltonian Dynamics*. Cambridge University Press, Feb. 2005, ISBN: 9780511614118. DOI: 10.1017/cbo9780511614118.

[33] M. Karplus, "Dynamics of folded proteins", *Abstracts of papers of the American Chemical Society*, vol. 175, no. Mar, p. 70, 1978.

[34] E. Brini, C. Simmerling, and K. Dill, "Protein storytelling through physics", *Science*, vol. 370, no. 6520, Nov. 2020, ISSN: 1095-9203. DOI: 10.1126/science.aaz3041.

[35] J. Jumper et al., "Highly accurate protein structure prediction with AlphaFold", *Nature*, vol. 596, no. 7873, pp. 583–589, Jul. 2021, ISSN: 1476-4687. DOI: 10.1038/s41586-021-03819-2.

[36] H. K. Brittain, R. Scott, and E. Thomas, "The rise of the genome and personalised medicine", *Clinical Medicine*, vol. 17, no. 6, pp. 545–551, Dec. 2017, ISSN: 1470-2118. DOI: 10.7861/clinmedicine.17-6-545.

[37] M. Flores, G. Glusman, K. Brogaard, N. D. Price, and L. Hood, "P4 medicine: How systems medicine will transform the healthcare sector and society", *Personalized Medicine*, vol. 10, no. 6, pp. 565–576, Aug. 2013, ISSN: 1744-828X. DOI: 10.2217/pme.13.57.

[38] R. F. Green et al., "Evaluating the role of public health in implementation of genomics-related recommendations: A case study of hereditary cancers using the CDC science impact framework", *Genetics in Medicine*, vol. 21, no. 1, pp. 28–37, Jan. 2019, ISSN: 1098-3600. DOI: 10.1038/s41436-018-0028-2.

[39] L. Chin, J. N. Andersen, and P. A. Futreal, "Cancer genomics: From discovery science to personalized medicine", *Nature Medicine*, vol. 17, no. 3, pp. 297–303, Mar. 2011, ISSN: 1546-170X. DOI: 10.1038/nm.2323.

[40] D. P. Germain et al., "The benefits and challenges of family genetic testing in rare genetic diseases-lessons from Fabry disease", *Molecular Genetics & Genomic Medicine*, vol. 9, no. 5, Apr. 2021, ISSN: 2324-9269. DOI: 10.1002/mgg3.1666.

[41] R. Spreafico, L. B. Soriaga, J. Grosse, H. W. Virgin, and A. Telenti, "Advances in genomics for drug development", *Genes*, vol. 11, no. 8, p. 942, Aug. 2020, ISSN: 2073-4425. DOI: 10.3390/genes11080942.

[42] M. E. K. Niemi, M. J. Daly, and A. Ganna, "The human genetic epidemiology of COVID-19", *Nature Reviews Genetics*, vol. 23, no. 9, pp. 533–546, May 2022, ISSN: 1471-0064. DOI: 10.1038/s41576-022-00478-5.

[43] S. Knyazev et al., "Unlocking capacities of genomics for the COVID-19 response and future pandemics", *Nature Methods*, vol. 19, no. 4, pp. 374–380, Apr. 2022, ISSN: 1548-7105. DOI: 10.1038/s41592-022-01444-z.

[44] G. Lightbody et al., "Review of applications of high-throughput sequencing in personalized medicine: Barriers and facilitators of future progress in research and clinical application", *Briefings in Bioinformatics*, vol. 20, no. 5, pp. 1795–1811, Jun. 2019, ISSN: 1477-4054. DOI: 10.1093/bib/bby051.

[45] M. Alser et al., "From molecules to genomic variations: Accelerating genome analysis via intelligent algorithms and architectures", *Computational and Structural Biotechnology Journal*, vol. 20, pp. 4579–4599, 2022, ISSN: 2001-0370. DOI: 10.1016/j.csbj.2022.08.019.

[46] L. López-Villellas et al., "Accurate and efficient constrained molecular dynamics of polymers using Newton's method and special purpose code", *Computer Physics Communications*, vol. 288, p. 108 742, Jul. 2023, ISSN: 0010-4655. DOI: 10.1016/j.cpc.2023.108742.

[47] L. López-Villellas et al., "ILVES: Accurate and efficient bond length and angle constraints in molecular dynamics", *Journal of Chemical Theory and Computation*, vol. 21, no. 18, pp. 8711–8719, Sep. 2025, ISSN: 1549-9626. DOI: 10.1021/acs.jctc.5c01376.

[48] C. C. Kjelgaard Mikkelsen, L. López-Villellas, and P. García-Risueño, "How accurate does Newton have to be?", in *Parallel Processing and Applied Mathematics*, Springer International Publishing, 2023, pp. 3–15, ISBN: 9783031304422. DOI: 10.1007/978-3-031-30442-2_1.

[49] C. C. Kjelgaard Mikkelsen, L. López-Villellas, and P. García-Risueño, "Newton's method revisited: How accurate do we have to be?", *Concurrency and Computation: Practice and Experience*, vol. 36, no. 10, Jul. 2023, ISSN: 1532-0634. DOI: 10.1002/cpe.7853.

[50] C. C. Kjelgaard Mikkelsen and L. López-Villellas, "The need for accuracy and smoothness in numerical simulations", in *Parallel Processing and Applied Mathematics*, Springer Nature Switzerland, 2025, pp. 3–16, ISBN: 9783031856976. DOI: 10.1007/978-3-031-85697-6_1.

[51] C. C. Kjelgaard Mikkelsen and L. López-Villellas, "How accurate is Richardson's error estimate?", *Concurrency and Computation: Practice and Experience*, vol. 37, no. 27-28, Nov. 2025, ISSN: 1532-0634. DOI: 10.1002/cpe.70305.

[52] N. Gupta, R. Ashiwal, B. Brank, S. K. Peddoju, and D. Pleiter, "Performance evaluation of ParalleX execution model on Arm-based platforms", in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, IEEE, Sep. 2020, pp. 567–575. DOI: 10.1109/cluster49012.2020.00080.

[53] S. Sreepathi and M. Taylor, "Early evaluation of Fugaku A64FX architecture using climate workloads", in *2021 IEEE International Conference on Cluster Computing (CLUSTER)*, IEEE, Sep. 2021, pp. 719–727. DOI: 10.1109/cluster48925.2021.00107.

[54] L. López-Villellas et al., "GenArchBench: A genomics benchmark suite for Arm HPC processors", *Future Generation Computer Systems*, vol. 157, pp. 313–329, Aug. 2024, ISSN: 0167-739X. DOI: 10.1016/j.future.2024.03.050.

[55] L. López-Villellas et al., "RISC-V for genome data analysis: Opportunities and challenges", in *2023 38th Conference on Design of Circuits and Integrated Systems (DCIS)*, IEEE, Nov. 2023, pp. 1–6. DOI: 10.1109/dcis58620.2023.10335997.

[56] L. López-Villellas et al., "Singletrack: An algorithm for improving memory consumption and performance of gap-affine sequence alignment", *Bioinformatics*, vol. 42, no. 5, Apr. 2026, ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btag183.

[57] J. L. Alonso, G. A. Chass, I. G. Csizmadia, P. Echenique, and A. Tarancón, "Do theoretical physicists care about the protein folding problem?", in *Meeting on Fundamental Physics 'Alberto Galindo'*, Madrid: Aula Documental de Investigación, 2004.

[58] J. J. Galano-Frutos and J. Sancho, "Energy, water, and protein folding: A molecular dynamics-based quantitative inventory of molecular interactions and forces that make proteins stable", *Protein Science*, vol. 33, no. 2, Jan. 2024, ISSN: 1469-896X. DOI: 10.1002/pro.4905.

[59] Y. Li et al., "Single-molecule imaging and molecular dynamics simulations reveal early activation of the MET receptor in cells", *Nature Communications*, vol. 15, no. 1, Nov. 2024, ISSN: 2041-1723. DOI: 10.1038/s41467-024-53772-7.

[60] S. Lehrer and P. H. Rheinstein, "Predicted configuration and stability of the ATAD2/SOX10 complex using molecular dynamics simulations", *Cancer Diagnosis & Prognosis*, vol. 3, no. 3, pp. 398–402, May 2023, ISSN: 2732-7787. DOI: 10.21873/cdp.10231.

[61] N. Zheng et al., "Protein-glutaminase engineering based on isothermal compressibility perturbation for enhanced modification of soy protein isolate", *Journal of Agricultural and Food Chemistry*, vol. 70, no. 43, pp. 13 969–13 978, Oct. 2022, ISSN: 1520-5118. DOI: 10.1021/acs.jafc.2c06063.

[62] E. A. Osifalujo, B. N. Rutkowski, L. R. Satterwhite, P. C. Betts, A. K. Nkosi, and J. T. Froese, "Production of novel Rieske dioxygenase metabolites enabled by enzyme engineering", *Catalysis Science & Technology*, vol. 13, no. 13, pp. 3784–3790, 2023, ISSN: 2044-4761. DOI: 10.1039/d3cy00262d.

[63] B. Meng et al., "Altered TMPRSS2 usage by SARS-CoV-2 Omicron impacts infectivity and fusogenicity", *Nature*, vol. 603, no. 7902, pp. 706–714, 2022.

[64] S. Mosalaganti et al., "AI-based structure prediction empowers integrative structural analysis of human nuclear pores", *Science*, vol. 376, no. 6598, Jun. 2022, ISSN: 1095-9203. DOI: 10.1126/science.abm9506.

[65] J. Min, X. Rong, J. Zhang, R. Su, Y. Wang, and W. Qi, "Computational design of peptide assemblies", *Journal of Chemical Theory and Computation*, vol. 20, no. 2, pp. 532–550, Jan. 2024, ISSN: 1549-9626. DOI: 10.1021/acs.jctc.3c01054.

[66] U. Stocker, D. Juchli, and W. F. van Gunsteren, "Increasing the time step and efficiency of molecular dynamics simulations: Optimal solutions for equilibrium simulations or structure refinement of large biomolecules", *Molecular Simulation*, vol. 29, no. 2, pp. 123–138, Jan. 2003, ISSN: 1029-0435. DOI: 10.1080/0892702031000065791.

[67] J.-P. Ryckaert, G. Ciccotti, and H. J. C. Berendsen, "Numerical integration of the cartesian equations of motion of a system with constraints: Molecular dynamics of n-alkanes", *Journal of Computational Physics*, vol. 23, no. 3, pp. 327–341, Mar. 1977, ISSN: 0021-9991. DOI: 10.1016/0021-9991(77)90098-5.

[68] B. Hess, H. Bekker, H. J. C. Berendsen, and J. G. E. M. Fraaije, "LINCS: A linear constraint solver for molecular simulations", *Journal of Computational Chemistry*, vol. 18, no. 12, pp. 1463–1472, Sep. 1997, ISSN: 1096-987X. DOI: 10.1002/(sici)1096-987x(199709)18:12<1463::aid-jcc4>3.0.co;2-h.

[69] B. Hess, "P-LINCS: A parallel linear constraint solver for molecular simulation", *Journal of Chemical Theory and Computation*, vol. 4, no. 1, pp. 116–122, Dec. 2007, ISSN: 1549-9626. DOI: 10.1021/ct700200b.

[70] S. Thallmair, M. Javanainen, B. Fábián, H. Martinez-Seara, and S. J. Marrink, "Nonconverged constraints cause artificial temperature gradients in lipid bilayer simulations", *The Journal of Physical Chemistry B*, vol. 125, no. 33, pp. 9537–9546, Aug. 2021, ISSN: 1520-5207. DOI: 10.1021/acs.jpcb.1c03665.

[71] P. Eastman and V. S. Pande, "Energy conservation as a measure of simulation accuracy", Oct. 2016. DOI: 10.1101/083055.

[72] M. Pechlaner and W. F. van Gunsteren, "On the use of intra-molecular distance and angle constraints to lengthen the time step in molecular and stochastic dynamics simulations of proteins", *Proteins: Structure, Function, and Bioinformatics*, vol. 90, no. 2, pp. 543–559, 2022.

[73] E. Krieger and G. Vriend, "New ways to boost molecular dynamics simulations", *Journal of Computational Chemistry*, vol. 36, no. 13, pp. 996–1007, Mar. 2015, ISSN: 1096-987X. DOI: 10.1002/jcc.23899.

[74] L. Verlet, "Computer "experiments" on classical fluids. I. thermodynamical properties of Lennard-Jones molecules", *Physical Review*, vol. 159, no. 1, pp. 98–103, Jul. 1967, ISSN: 0031-899X. DOI: 10.1103/physrev.159.98.

[75] E. Hairer, M. Hochbruck, A. Iserles, and C. Lubich, "Geometric numerical integration", *Oberwolfach Reports*, vol. 8, no. 1, pp. 825–900, Sep. 2011, ISSN: 1660-8941. DOI: 10.4171/owr/2011/16.

[76] J. Huang and A. D. MacKerell, "CHARMM36 all-atom additive protein force field: Validation based on comparison to NMR data", *Journal of Computational Chemistry*, vol. 34, no. 25, pp. 2135–2145, Jul. 2013, ISSN: 0192-8651. DOI: 10.1002/jcc.23354.

[77] J. Huang et al., "CHARMM36: An improved force field for folded and intrinsically disordered proteins", *Biophysical Journal*, vol. 112, no. 3, 175a–176a, Feb. 2017, ISSN: 0006-3495. DOI: 10.1016/j.bpj.2016.11.971.

[78] V. Hornak, R. Abel, A. Okur, B. Strockbine, A. Roitberg, and C. Simmerling, "Comparison of multiple Amber force fields and development of improved protein backbone parameters", *Proteins: Structure, Function, and Bioinformatics*, vol. 65, no. 3, pp. 712–725, Sep. 2006, ISSN: 1097-0134. DOI: 10.1002/prot.21123.

[79] W. L. Jorgensen, D. S. Maxwell, and J. Tirado-Rives, "Development and testing of the OPLS all-atom force field on conformational energetics and properties of organic liquids", *Journal of the American Chemical Society*, vol. 118, no. 45, pp. 11 225–11 236, Nov. 1996, ISSN: 1520-5126. DOI: 10.1021/ja9621760.

[80] M. Abraham et al., "GROMACS 2024.1 manual", 2024. DOI: 10.5281/ZENODO.10721192.

[81] B. Fábián, S. Thallmair, and G. Hummer, "Optimal bond constraint topology for molecular dynamics simulations of cholesterol", *Journal of Chemical Theory and Computation*, vol. 19, no. 5, pp. 1592–1601, Feb. 2023, ISSN: 1549-9626. DOI: 10.1021/acs.jctc.2c01032.

[82] C. W. Hopkins, S. Le Grand, R. C. Walker, and A. E. Roitberg, "Long-time-step molecular dynamics through hydrogen mass repartitioning", *Journal of Chemical Theory and Computation*, vol. 11, no. 4, pp. 1864–1874, Mar. 2015, ISSN: 1549-9626. DOI: 10.1021/ct5010406.

[83] S. Pal, O. Jahanmahin, S. Zhang, and T.-S. Lee, "Impact of timestep on energy drift and accuracy of alchemical free energy calculations", Feb. 2025. DOI: 10.26434/chemrxiv-2025-jwkz1.

[84] C. Balusek et al., "Accelerating membrane simulations with hydrogen mass repartitioning", *Journal of Chemical Theory and Computation*, vol. 15, no. 8, pp. 4673–4686, Jun. 2019, ISSN: 1549-9626. DOI: 10.1021/acs.jctc.9b00160.

[85] M. Sahil, S. Sarkar, and J. Mondal, "Long-time-step molecular dynamics can retard simulation of protein-ligand recognition process", *Biophysical Journal*, vol. 122, no. 5, pp. 802–816, Mar. 2023, ISSN: 0006-3495. DOI: 10.1016/j.bpj.2023.01.036.

[86] A. K. Mazur, "Hierarchy of fast motions in protein dynamics", *The Journal of Physical Chemistry B*, vol. 102, no. 2, pp. 473–479, Jan. 1998, ISSN: 1520-5207. DOI: 10.1021/jp972381h.

[87] K. A. Feenstra, B. Hess, and H. J. C. Berendsen, "Improving efficiency of large time-scale molecular dynamics simulations of hydrogen-rich systems", *Journal of Computational Chemistry*, vol. 20, no. 8, pp. 786–798, Jun. 1999, ISSN: 1096-987X. DOI: 10.1002/(sici)1096-987x(199906)20:8<786::aid-jcc5>3.0.co;2-b.

[88] P. Gonnet, J. H. Walther, and P. Koumoutsakos, "Theta SHAKE: An extension to SHAKE for the explicit treatment of angular constraints", *Computer Physics Communications*, vol. 180, pp. 360–364, 2009.

[89] M. Pechlaner, A. P. Dorta, Z. Lin, V. H. Rusu, and W. F. van Gunsteren, "A method to apply bond-angle constraints in molecular dynamics simulations", *Journal of Computational Chemistry*, vol. 42, no. 6, pp. 418–434, 2021.

[90] Y. Hayashi, J. Shiomi, J. Morikawa, and R. Yoshida, "RadonPy: Automated physical property calculation using all-atom classical molecular dynamics simulations for polymer informatics", *npj Computational Materials*, vol. 8, no. 1, Nov. 2022, ISSN: 2057-3960. DOI: 10.1038/s41524-022-00906-4.

[91] M. M. Huda and N. Rai, "Probing early-stage aggregation of low molecular weight gelator in an organic solvent", *The Journal of Physical Chemistry B*, vol. 124, no. 11, pp. 2277–2288, Feb. 2020, ISSN: 1520-5207. DOI: 10.1021/acs.jpcb.9b11865.

[92] M. G. Ciulla et al., "Novel self-assembling cyclic peptides with reversible supramolecular nanostructures", *Materials Chemistry Frontiers*, vol. 7, no. 17, pp. 3680–3692, 2023, ISSN: 2052-1537. DOI: 10.1039/d3qm00198a.

[93] A. Wacha, Z. Varga, and T. Beke-Somfai, "Comparative study of molecular mechanics force fields for $\beta$-peptidic foldamers: Folding and self-association", *Journal of Chemical Information and Modeling*, vol. 63, no. 12, pp. 3799–3813, Jun. 2023, ISSN: 1549-960X. DOI: 10.1021/acs.jcim.3c00175.

[94] S. J. Marrink, H. J. Risselada, S. Yefimov, D. P. Tieleman, and A. H. de Vries, "The MARTINI force field: Coarse grained model for biomolecular simulations", *The Journal of Physical Chemistry B*, vol. 111, no. 27, pp. 7812–7824, Jun. 2007, ISSN: 1520-5207. DOI: 10.1021/jp071097f.

[95] K. D. Hammonds and D. M. Heyes, "Shadow hamiltonian in classical NVE molecular dynamics simulations: A path to long time stability", *The Journal of Chemical Physics*, vol. 152, no. 2, Jan. 2020, ISSN: 1089-7690. DOI: 10.1063/1.5139708.

[96] M. Pechlaner and W. F. van Gunsteren, "Algorithms to apply dihedral-angle constraints in molecular or stochastic dynamics simulations", *The Journal of Chemical Physics*, vol. 152, no. 2, Jan. 2020, ISSN: 1089-7690. DOI: 10.1063/1.5124923.

[97] J. J. Moré, "Nonlinear generalizations of matrix diagonal dominance with application to Gauss-Seidel iterations", *SIAM Journal on Numerical Analysis*, vol. 9, no. 2, pp. 357–378, Jun. 1972, ISSN: 1095-7170. DOI: 10.1137/0709035.

[98] T. A. Porshing, "On the Rates of Convergence of Jacobi and Gauss-Seidel Methods for M-functions", *SIAM Journal on Numerical Analysis*, vol. 8, no. 3, pp. 575–582, 1971.

[99] R. Elber, A. P. Ruymgaart, and B. Hess, "SHAKE parallelization", *The European Physical Journal Special Topics*, vol. 200, no. 1, pp. 211–223, Nov. 2011, ISSN: 1951-6401. DOI: 10.1140/epjst/e2011-01525-9.

[100] Y. Weinbach and R. Elber, "Revisiting and parallelizing SHAKE", *Journal of Computational Physics*, vol. 209, no. 1, pp. 193–206, Oct. 2005, ISSN: 0021-9991. DOI: 10.1016/j.jcp.2005.03.015.

[101] J. C. Phillips et al., "Scalable molecular dynamics with NAMD", *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1781–1802, Oct. 2005, ISSN: 1096-987X. DOI: 10.1002/jcc.20289.

[102] J. C. Phillips et al., "Scalable molecular dynamics on CPU and GPU architectures with NAMD", *The Journal of Chemical Physics*, vol. 153, no. 4, Jul. 2020, ISSN: 1089-7690. DOI: 10.1063/5.0014475.

[103] R. J. Broadbent, J. S. Spencer, A. A. Mostofi, and A. P. Sutton, "Accelerated simulations of aromatic polymers: Application to polyether ether ketone (PEEK)", *Molecular Physics*, vol. 112, no. 20, pp. 2672–2680, Aug. 2014, ISSN: 1362-3028. DOI: 10.1080/00268976.2014.905719.

[104] H. C. Andersen, "Rattle: A "velocity" version of the shake algorithm for molecular dynamics calculations", *Journal of Computational Physics*, vol. 52, no. 1, pp. 24–34, Oct. 1983, ISSN: 0021-9991. DOI: 10.1016/0021-9991(83)90014-1.

[105] S. Miyamoto and P. A. Kollman, "Settle: An analytical version of the SHAKE and RATTLE algorithm for rigid water models", *Journal of Computational Chemistry*, vol. 13, no. 8, pp. 952–962, Oct. 1992, ISSN: 1096-987X. DOI: 10.1002/jcc.540130805.

[106] S. G. Lambrakos, J. P. Boris, E. S. Oran, I. Chandrasekhar, and M. Nagumo, "A modified SHAKE algorithm for maintaining rigid bonds in molecular dynamics simulations of large molecules", *Journal of Computational Physics*, vol. 85, no. 2, pp. 473–486, Dec. 1989, ISSN: 0021-9991. DOI: 10.1016/0021-9991(89)90160-5.

[107] S.-H. Lee, K. Palmo, and S. Krimm, "WIGGLE: A new constrained molecular dynamics algorithm in cartesian coordinates", *Journal of Computational Physics*, vol. 210, no. 1, pp. 171–182, Nov. 2005, ISSN: 0021-9991. DOI: 10.1016/j.jcp.2005.04.006.

[108] P. Gonnet, "P-SHAKE: A quadratically convergent SHAKE in $O(n^2)$", *Journal of Computational Physics*, vol. 220, no. 2, pp. 740–750, Jan. 2007, ISSN: 0021-9991. DOI: 10.1016/j.jcp.2006.05.032.

[109] G. Ciccotti and J. P. Ryckaert, "Molecular dynamics simulation of rigid molecules", *Computer Physics Reports*, vol. 4, no. 6, pp. 346–392, Sep. 1986, ISSN: 0167-7977. DOI: 10.1016/0167-7977(86)90022-5.

[110] J.-P. Ryckaert, G. Arialdi, and S. Melchionna, "Molecular dynamics of polymers with explicit but frozen hydrogens", *Molecular Physics*, vol. 99, no. 3, pp. 155–165, Feb. 2001, ISSN: 1362-3028. DOI: 10.1080/00268970010007280.

[111] P. Eastman and V. S. Pande, "Constant constraint matrix approximation: A robust, parallelizable constraint method for molecular simulations", *Journal of Chemical Theory and Computation*, vol. 6, no. 2, pp. 434–437, Jan. 2010, ISSN: 1549-9626. DOI: 10.1021/ct900463w.

[112] V. Krautler, W. F. van Gunsteren, and P. H. Hunenberger, "A fast SHAKE algorithm to solve distance constraint equations for small molecules in molecular dynamics simulations", *Journal of Computational Chemistry*, vol. 22, no. 5, pp. 501–508, 2001, ISSN: 1096-987X. DOI: 10.1002/1096-987x(20010415)22:5<501::aid-jcc1021>3.0.co;2-v.

[113] M. Christen and W. F. van Gunsteren, "An approximate but fast method to impose flexible distance constraints in molecular dynamics simulations", *The Journal of Chemical Physics*, vol. 122, no. 14, Apr. 2005, ISSN: 1089-7690. DOI: 10.1063/1.1872792.

[114] A. G. Bailey, C. P. Lowe, and A. P. Sutton, "Efficient constraint dynamics using MILC SHAKE", *Journal of Computational Physics*, vol. 227, no. 20, pp. 8949–8959, Oct. 2008, ISSN: 0021-9991. DOI: 10.1016/j.jcp.2008.07.002.

[115] A. G. Bailey and C. P. Lowe, "MILCH SHAKE: An efficient method for constraint dynamics applied to alkanes", *Journal of Computational Chemistry*, vol. 30, no. 15, pp. 2485–2493, Apr. 2009, ISSN: 1096-987X. DOI: 10.1002/jcc.21237.

[116] M. Mazars, "Holonomic constraints: An analytical result", *Journal of Physics A: Mathematical and Theoretical*, vol. 40, no. 8, pp. 1747–1755, Feb. 2007, ISSN: 1751-8121. DOI: 10.1088/1751-8113/40/8/004.

[117] E. Barth, K. Kuczera, B. Leimkuhler, and R. D. Skeel, "Algorithms for constrained molecular dynamics", *Journal of Computational Chemistry*, vol. 16, no. 10, pp. 1192–1209, Oct. 1995, ISSN: 1096-987X. DOI: 10.1002/jcc.540161003.

[118] D. Dubbeldam, G. A. E. Oxford, R. Krishna, L. J. Broadbelt, and R. Q. Snurr, "Distance and angular holonomic constraints in molecular simulations", *The Journal of Chemical Physics*, vol. 133, no. 3, Jul. 2010, ISSN: 1089-7690. DOI: 10.1063/1.3429610.

[119] W. F. van Gunsteren et al., "On the effect of the various assumptions and approximations used in molecular simulations on the properties of bio-molecular systems: Overview and perspective on issues", *ChemPhysChem*, vol. 22, no. 3, pp. 264–282, 2021.

[120] D. A. Pearlman et al., "AMBER, a computer program for applying molecular mechanics, normal mode analysis, Molecular Dynamics and free energy calculations to elucidate the structures and energies of molecules", *Computer Physics Communications*, vol. 91, pp. 1–41, 1995.

[121] Q. N. Vo, C. A. Hawkins, L. X. Dang, M. Nilsson, and H. D. Nguyen, "Computational study of molecular structure and self-association of tri-n-butyl phosphates in n-dodecane", *The Journal of Physical Chemistry B*, vol. 119, no. 4, pp. 1588–1597, 2015.

[122] S.-W. Chiu, M. Clark, S. Subramaniam, and E. Jakobsson, "Collective motion artifacts arising in long-duration molecular dynamics simulations", *Journal of Computational Chemistry*, vol. 21, no. 2, pp. 121–131, Jan. 2000, ISSN: 1096-987X. DOI: 10.1002/(sici)1096-987x(20000130)21:2<121::aid-jcc4>3.0.co;2-w.

[123] S. Riniker and W. F. van Gunsteren, "A simple, efficient polarizable coarse-grained water model for molecular dynamics simulations", *The Journal of Chemical Physics*, vol. 134, no. 8, Feb. 2011, ISSN: 1089-7690. DOI: 10.1063/1.3553378.

[124] A. P. Ruymgaart, A. E. Cardenas, and R. Elber, "MOIL-opt: Energy-conserving molecular dynamics on a GPU/CPU system", *Journal of Chemical Theory and Computation*, vol. 7, no. 10, pp. 3072–3082, Sep. 2011, ISSN: 1549-9626. DOI: 10.1021/ct200360f.

[125] J. D. Chodera, W. C. Swope, J. W. Pitera, C. Seok, and K. A. Dill, "Use of the weighted histogram analysis method for the analysis of simulated and parallel tempering simulations", *Journal of Chemical Theory and Computation*, vol. 3, no. 1, pp. 26–41, 2007.

[126] J. D. Chodera, W. C. Swope, J. W. Pitera, and K. A. Dill, "Long-time protein folding dynamics from short-time MD", *Multiscale Modeling & Simulation*, vol. 5, no. 4, pp. 1214–1226, 2006.

[127] M. T. Panteva, G. M. Giambasu, and D. M. York, "Comparison of structural, thermodynamic, kinetic and mass transport properties of Mg(2+) ion models commonly used in biomolecular simulations", *Journal of Computational Chemistry*, vol. 15, no. 36, pp. 970–82, 13 2015.

[128] F. Chen, R. Kerr, and M. Forsyth, "Cation effect on small phosphonium based ionic liquid electrolytes with high concentrations of lithium salt", *The Journal of Chemical Physics*, vol. 148, no. 19, Jan. 2018, ISSN: 1089-7690. DOI: 10.1063/1.5016460.

[129] S. Das, V. S. Baghel, S. Roy, and R. Kumar, "A molecular dynamics study of model SI clathrate hydrates: The effect of guest size and guest-water interaction on decomposition kinetics", *Physical Chemistry Chemical Physics*, vol. 17, no. 14, pp. 9509–9518, 2015, ISSN: 1463-9084. DOI: 10.1039/c5cp00678c.

[130] D. L. Roest, P. Ballone, D. Bedeaux, and S. Kjelstrup, "MD simulations of metal/molten alkali carbonate interfaces", *The Journal of Physical Chemistry C*, vol. 121, no. 33, pp. 17 827–17 847, 2017.

[131] A. Ottochian, C. Ricca, F. Labat, and C. Adamo, "Molecular dynamics simulations of a lithium/sodium carbonate mixture", *Journal of Molecular Modeling*, vol. 22, no. 3, Feb. 2016, ISSN: 0948-5023. DOI: 10.1007/s00894-016-2921-4.

[132] H. Hu and F. Wang, "The liquid-vapor equilibria of TIP4P/2005 and BLYPSP-4F water models determined through direct simulations of the liquid-vapor interface", *The Journal of Chemical Physics*, vol. 142, no. 21, Jun. 2015, ISSN: 1089-7690. DOI: 10.1063/1.4922166.

[133] M. Turner, S. T. Mutter, R. J. Deeth, and J. A. Platts, "Ligand field molecular dynamics simulation of Pt(II)-phenanthroline binding to N-terminal fragment of amyloid-$\beta$ peptide", *PLOS ONE*, vol. 13, no. 3, M. K. Lakshmana, Ed., e0193668, Mar. 2018, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0193668.

[134] I. I. I. Thomas E. Cheatham, P. Cieplak, and P. A.Kollman, "A modified version of the Cornell et al. force field with improved sugar pucker phases and helical repeat", *Journal of Biomolecular Structure and Dynamics*, vol. 16, no. 4, pp. 845–862, 1999.

[135] J. M. Martínez, S. K. C. Elmroth, and L. Kloo, "Influence of sodium ions on the dynamics and structure of single-stranded DNA oligomers: A molecular dynamics study", *Journal of the American Chemical Society*, vol. 123, no. 49, pp. 12 279–12 289, Nov. 2001, ISSN: 1520-5126. DOI: 10.1021/ja0108786.

[136] R. Galindo-Murillo et al., "Assessing the current state of Amber force field modifications for DNA", *Journal of Chemical Theory and Computation*, vol. 12, no. 8, pp. 4114–4127, Jul. 2016, ISSN: 1549-9626. DOI: 10.1021/acs.jctc.6b00186.

[137] S. P. Hancock, T. Ghane, D. Cascio, R. Rohs, R. Di Felice, and R. C. Johnson, "Control of DNA minor groove width and Fis protein binding by the purine 2-amino group", *Nucleic Acids Research*, vol. 41, no. 13, pp. 6750–6760, May 2013, ISSN: 0305-1048. DOI: 10.1093/nar/gkt357.

[138] S. Toxvaerd, O. J. Heilmann, T. Ingebrigtsen, T. B. Schrøder, and J. C. Dyre, "Time-reversible molecular dynamics algorithms with bond constraints", *The Journal of Chemical Physics*, vol. 131, no. 6, Aug. 2009, ISSN: 1089-7690. DOI: 10.1063/1.3194785.

[139] H. Okumura, S. G. Itoh, A. M. Ito, H. Nakamura, and T. Fukushima, "Manifold correction method for the Nosé-Hoover and Nosé-Poincaré molecular dynamics simulations", *Journal of the Physical Society of Japan*, vol. 83, no. 2, p. 024 003, Feb. 2014, ISSN: 1347-4073. DOI: 10.7566/jpsj.83.024003.

[140] G. Bussi, T. Zykova-Timan, and M. Parrinello, "Isothermal-isobaric molecular dynamics using stochastic velocity rescaling", *The Journal of Chemical Physics*, vol. 130, no. 7, Feb. 2009, ISSN: 1089-7690. DOI: 10.1063/1.3073889.

[141] B. Hess, D. van der Spoel, M. J. Abraham, and E. Lindahl, "On the importance of accurate algorithms for reliable molecular dynamics simulations", Dec. 2019. DOI: 10.26434/chemrxiv.11474583.v1.

[142] P. García-Risueño, P. Echenique, and J. L. Alonso, "Exact and efficient calculation of Lagrange multipliers in constrained biological polymers: Proteins and nucleic acids as example cases", *Journal of Computational Chemistry*, vol. 32, no. 14, pp. 3039–3046, 2011.

[143] P. García-Risueño and P. Echenique, "Linearly scaling direct method for accurately inverting sparse banded matrices", *Journal of Physics A: Mathematical and Theoretical*, vol. 45, no. 6, p. 065 204, Jan. 2012, ISSN: 1751-8121. DOI: 10.1088/1751-8113/45/6/065204.

[144] C. C. K. Mikkelsen, J. Alastruey-Benedé, P. Ibáñez-Marín, and P. G. Risueño, "Accelerating sparse arithmetic in the context of newton's method for small molecules with bond constraints", in *Parallel Processing and Applied Mathematics*, Springer International Publishing, 2016, pp. 160–171, ISBN: 9783319321493. DOI: 10.1007/978-3-319-32149-3_16.

[145] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU press, 2013.

[146] G. Lou, "Parallel methods for solving linear systems via overlapping decomposition", M.S. thesis, University of Illinois at Urbana-Champaign, 1989.

[147] B. R. Brooks et al., "CHARMM: The biomolecular simulation program", *Journal of Computational Chemistry*, vol. 30, no. 10, pp. 1545–1614, May 2009, ISSN: 1096-987X. DOI: 10.1002/jcc.21287.

[148] A. P. Thompson et al., "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales", *Computer Physics Communications*, vol. 271, p. 108 171, 2022, ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2021.108171. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010465521002836.

[149] K. J. Bowers et al., "Scalable algorithms for molecular dynamics simulations on commodity clusters", in *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, 2006, 84–es.

[150] D. E. Shaw et al., "Anton 3: Twenty microseconds of molecular dynamics simulation before lunch", in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21, ACM, Nov. 2021, pp. 1–11. DOI: 10.1145/3458817.3487397.

[151] J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*. Society for Industrial and Applied Mathematics, Jan. 2000, ISBN: 9780898719468. DOI: 10.1137/1.9780898719468.

[152] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*. Society for Industrial and Applied Mathematics, Jan. 1995, ISBN: 9781611970944. DOI: 10.1137/1.9781611970944.

[153] G. Karypis and V. Kumar, "Multilevelk-way partitioning scheme for irregular graphs", *Journal of Parallel and Distributed Computing*, vol. 48, no. 1, pp. 96–129, Jan. 1998, ISSN: 0743-7315. DOI: 10.1006/jpdc.1997.1404.

[154] P. R. Amestoy, T. A. Davis, and I. S. Duff, "An approximate minimum degree ordering algorithm", *SIAM Journal on Matrix Analysis and Applications*, vol. 17, no. 4, pp. 886–905, Oct. 1996, ISSN: 1095-7162. DOI: 10.1137/s0895479894278952.

[155] J. Kim, J. H. Oh, and D. Kim, "Recent advances in single-benzene-based fluorophores: Physicochemical properties and applications", *Organic & Biomolecular Chemistry*, vol. 19, no. 5, pp. 933–946, 2021, ISSN: 1477-0539. DOI: 10.1039/d0ob02387f.

[156] I. Levental and E. Lyman, "Regulation of membrane protein structure and function by their lipid nano-environment", *Nature Reviews Molecular Cell Biology*, vol. 24, no. 2, pp. 107–122, 2023.

[157] C. Martin, V. Richard, M. Salem, R. Hartley, and Y. Mauguen, "Refinement and structural analysis of barnase at 1.5 å resolution", *Acta Crystallographica Section D Biological Crystallography*, vol. 55, no. 2, pp. 386–398, Feb. 1999, ISSN: 0907-4449. DOI: 10.1107/s0907444998010865.

[158] A. D. Mackerell, M. Feig, and C. L. Brooks, "Extending the treatment of backbone energetics in protein force fields: Limitations of gas-phase quantum mechanics in reproducing protein conformational distributions in molecular dynamics simulations", *Journal of Computational Chemistry*, vol. 25, no. 11, pp. 1400–1415, Jun. 2004, ISSN: 1096-987X. DOI: 10.1002/jcc.20065.

[159] W. L. Jorgensen, J. Chandrasekhar, J. D. Madura, R. W. Impey, and M. L. Klein, "Comparison of simple potential functions for simulating liquid water", *The Journal of Chemical Physics*, vol. 79, no. 2, pp. 926–935, Jul. 1983, ISSN: 1089-7690. DOI: 10.1063/1.445869.

[160] F. A. Cotton, E. E. Hazen, and M. J. Legg, *Staphylococcal nuclease: Proposed mechanism of action based on structure of enzyme-thymidine 3',5'-bisphosphate-calcium ion complex at 1.5-å resolution*, Jun. 1979. DOI: 10.1073/pnas.76.6.2551.

[161] A. Douangamath et al., "Crystallographic and electrophilic fragment screening of the SARS-CoV-2 main protease", *Nature Communications*, vol. 11, no. 1, pp. 1–11, 2020.

[162] E. Núñez-Rojas, H. M. Flores-Ruiz, and J. Alejandre, "Molecular dynamics simulations to separate benzene from hydrocarbons using polar and ionic liquid solvents", *Journal of Molecular Liquids*, vol. 249, pp. 591–599, Jan. 2018, ISSN: 0167-7322. DOI: 10.1016/j.molliq.2017.10.147.

[163] N. Schmid et al., "Definition and testing of the GROMOS force-field versions 54A7 and 54B7", *European Biophysics Journal*, vol. 40, no. 7, pp. 843–856, Apr. 2011, ISSN: 1432-1017. DOI: 10.1007/s00249-011-0700-9.

[164] B. Wu, K. Mohideen, D. Vasudevan, and C. A. Davey, "Structural insight into the sequence dependence of nucleosome positioning", *Structure*, vol. 18, no. 4, pp. 528–536, Mar. 2010, ISSN: 0969-2126. DOI: 10.1016/j.str.2010.01.015.

[165] C. Tian et al., "ff19SB: Amino-acid-specific protein backbone parameters trained against quantum mechanics energy surfaces in solution", *Journal of Chemical Theory and Computation*, vol. 16, no. 1, pp. 528–552, 2020.

[166] M. Zgarbová, J. Šponer, and P. Jurečka, "Z-DNA as a touchstone for additive empirical force fields and a refinement of the alpha/gamma DNA torsions for AMBER", *Journal of Chemical Theory and Computation*, vol. 17, no. 10, pp. 6292–6301, Sep. 2021, ISSN: 1549-9626. DOI: 10.1021/acs.jctc.1c00697.

[167] S. Izadi, R. Anandakrishnan, and A. V. Onufriev, "Building water models: A different approach", *The Journal of Physical Chemistry Letters*, vol. 5, no. 21, pp. 3863–3871, Oct. 2014, ISSN: 1948-7185. DOI: 10.1021/jz501780a.

[168] P. Li, L. F. Song, and K. M. Merz, "Systematic parameterization of monovalent ions employing the nonbonded model", *Journal of Chemical Theory and Computation*, vol. 11, no. 4, pp. 1645–1657, Mar. 2015, ISSN: 1549-9626. DOI: 10.1021/ct500918t.

[169] A. Sengupta, Z. Li, L. F. Song, P. Li, and K. M. Merz, "Parameterization of monovalent ions for the OPC3, OPC, TIP3P-FB, and TIP4P-FB water models", *Journal of Chemical Information and Modeling*, vol. 61, no. 2, pp. 869–880, Feb. 2021, ISSN: 1549-960X. DOI: 10.1021/acs.jcim.0c01390.

[170] H. Sui, B.-G. Han, J. K. Lee, P. Walian, and B. K. Jap, "Structural basis of water-specific transport through the AQP1 water channel", *Nature*, vol. 414, no. 6866, pp. 872–878, Dec. 2001, ISSN: 1476-4687. DOI: 10.1038/414872a.

[171] S. Jo, T. Kim, V. G. Iyer, and W. Im, "CHARMM-GUI: A web-based graphical user interface for CHARMM", *Journal of Computational Chemistry*, vol. 29, no. 11, pp. 1859–1865, Jun. 2008, ISSN: 1096-987X. DOI: 10.1002/jcc.20945.

[172] J. Lee, X. Cheng, S. Jo, A. D. MacKerell, J. B. Klauda, and W. Im, "CHARMM-GUI input generator for NAMD, Gromacs, Amber, Openmm, and CHARMM/OpenMM simulations using the CHARMM36 additive force field", *Biophysical Journal*, vol. 110, no. 3, 641a, Feb. 2016, ISSN: 0006-3495. DOI: 10.1016/j.bpj.2015.11.3431.

[173] E. L. Wu et al., "CHARMM-GUI *Membrane Builder* toward realistic biological membrane simulations", *Journal of Computational Chemistry*, vol. 35, no. 27, pp. 1997–2004, 2014.

[174] S. Jo, T. Kim, and W. Im, "Automated builder and database of protein/membrane complexes for molecular dynamics simulations", *PLoS ONE*, vol. 2, no. 9, A. Yuan, Ed., e880, Sep. 2007, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0000880.

[175] S. Páll and B. Hess, "A flexible algorithm for calculating pair interactions on SIMD architectures", *Computer Physics Communications*, vol. 184, no. 12, pp. 2641–2650, Dec. 2013, ISSN: 0010-4655. DOI: 10.1016/j.cpc.2013.06.003.

[176] U. Essmann, L. Perera, M. L. Berkowitz, T. Darden, H. Lee, and L. G. Pedersen, "A smooth particle mesh Ewald method", *The Journal of Chemical Physics*, vol. 103, no. 19, pp. 8577–8593, Nov. 1995, ISSN: 1089-7690. DOI: 10.1063/1.470117.

[177] E. J. Haug, J. S. Arora, and K. Matsui, "A steepest-descent method for optimization of mechanical systems", *Journal of Optimization Theory and Applications*, vol. 19, no. 3, pp. 401–424, Jul. 1976, ISSN: 1573-2878. DOI: 10.1007/bf00941484.

[178] H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, and J. R. Haak, "Molecular dynamics with coupling to an external bath", *The Journal of Chemical Physics*, vol. 81, no. 8, pp. 3684–3690, Oct. 1984, ISSN: 1089-7690. DOI: 10.1063/1.448118.

[179] M. Parrinello and A. Rahman, "Polymorphic transitions in single crystals: A new molecular dynamics method", *Journal of Applied Physics*, vol. 52, no. 12, pp. 7182–7190, Dec. 1981, ISSN: 1089-7550. DOI: 10.1063/1.328693.

[180] J. J. Galano-Frutos and J. Sancho, "Accurate calculation of Barnase and SNase folding energetics using short molecular dynamics simulations and an atomistic model of the unfolded ensemble: Evaluation of force fields and water models", *Journal of Chemical Information and Modeling*, vol. 59, no. 10, pp. 4350–4360, Sep. 2019, ISSN: 1549-960X. DOI: 10.1021/acs.jcim.9b00430.

[181] J. J. Galano-Frutos, F. Nerín-Fonz, and J. Sancho, "Calculation of protein folding thermodynamics using molecular dynamics simulations", *Journal of Chemical Information and Modeling*, vol. 63, no. 24, pp. 7791–7806, Nov. 2023, ISSN: 1549-960X. DOI: 10.1021/acs.jcim.3c01107.

[182] J. Estrada, P. Bernadó, M. Blackledge, and J. Sancho, "ProtSA: A web application for calculating sequence specific protein solvent accessibilities in the unfolded ensemble", *BMC Bioinformatics*, vol. 10, no. 1, Apr. 2009, ISSN: 1471-2105. DOI: 10.1186/1471-2105-10-104.

[183] W. J. Becktel and J. A. Schellman, "Protein stability curves", *Biopolymers*, vol. 26, no. 11, pp. 1859–1877, Nov. 1987, ISSN: 1097-0282. DOI: 10.1002/bip.360261104.

[184] K. D. Hammonds and J.-P. Ryckaert, "On the convergence of the SHAKE algorithm", *Computer Physics Communications*, vol. 62, no. 2-3, pp. 336–351, Mar. 1991, ISSN: 0010-4655. DOI: 10.1016/0010-4655(91)90105-t.

[185] C. Kutzner, S. Páll, M. Fechner, A. Esztermann, B. L. de Groot, and H. Grubmüller, "More bang for your buck: Improved use of GPU nodes for GROMACS 2018", *Journal of Computational Chemistry*, vol. 40, no. 27, pp. 2418–2431, Jul. 2019, ISSN: 1096-987X. DOI: 10.1002/jcc.26011.

[186] Y. Wei et al., "9.3 nvlink-c2c: A coherent off package chip-to-chip interconnect with 40gbps/pin single-ended signaling", in *2023 IEEE International Solid-State Circuits Conference (ISSCC)*, 2023, pp. 160–162. DOI: 10.1109/ISSCC42615.2023.10067395.

[187] W. M. Washington, L. Buja, and A. Craig, "The computational future for climate and earth system models: On the path to petaflop and beyond", *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 367, no. 1890, pp. 833–846, 2009.

[188] G. Martínez-Rosell, T. Giorgino, M. J. Harvey, and G. de Fabritiis, "Drug discovery and molecular dynamics: Methods, applications and perspective beyond the second timescale", *Current Topics in Medicinal Chemistry*, vol. 17, no. 23, Aug. 2017, ISSN: 1568-0266. DOI: 10.2174/1568026617666170414142549.

[189] M. S. Louis et al., "Towards deep learning using TensorFlow lite on RISC-V", in *Third Workshop on Computer Architecture Research with RISC-V (CARRV)*, vol. 1, 2019, p. 6.

[190] F. Sanger, S. Nicklen, and A. R. Coulson, "DNA sequencing with chain-terminating inhibitors", *Proceedings of the National Academy of Sciences*, vol. 74, no. 12, pp. 5463–5467, Dec. 1977, ISSN: 1091-6490. DOI: 10.1073/pnas.74.12.5463.

[191] A. M. Maxam and W. Gilbert, "A new method for sequencing DNA", *Proceedings of the National Academy of Sciences*, vol. 74, no. 2, pp. 560–564, Feb. 1977, ISSN: 1091-6490. DOI: 10.1073/pnas.74.2.560.

[192] E. S. Lander et al., "Initial sequencing and analysis of the human genome", *Nature*, vol. 409, no. 6822, pp. 860–921, Feb. 2001, ISSN: 1476-4687. DOI: 10.1038/35057062.

[193] J. A. Reuter, D. V. Spacek, and M. P. Snyder, "High-throughput sequencing technologies", *Molecular Cell*, vol. 58, no. 4, pp. 586–597, May 2015, ISSN: 1097-2765. DOI: 10.1016/j.molcel.2015.05.004.

[194] Chris Seymour, Oxford Nanopore Technologies Ltd., *Bonito: A pytorch basecaller for Oxford nanopore reads*, Oxford Nanopore Technologies, Ltd. Public License, v. 1.0, 2019. [Online]. Available: https://github.com/nanoporetech/bonito.

[195] R. R. Wick, L. M. Judd, and K. E. Holt, "Performance of neural network basecalling tools for Oxford nanopore sequencing", *Genome Biology*, vol. 20, no. 1, Jun. 2019, ISSN: 1474-760X. DOI: 10.1186/s13059-019-1727-y.

[196] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM", 2013.

[197] M. Vasimuddin, S. Misra, H. Li, and S. Aluru, "Efficient architecture-aware acceleration of BWA-MEM for multicore systems", in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, May 2019, pp. 314–324. DOI: 10.1109/ipdps.2019.00041.

[198] H. Li, "Minimap2: Pairwise alignment for nucleotide sequences", *Bioinformatics*, vol. 34, no. 18, I. Birol, Ed., pp. 3094–3100, May 2018, ISSN: 1367-4811. DOI: 10.1093/bioinformatics/bty191.

[199] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome", *Genome Biology*, vol. 10, no. 3, Mar. 2009, ISSN: 1474-760X. DOI: 10.1186/gb-2009-10-3-r25.

[200] B. Langmead and S. L. Salzberg, "Fast gapped-read alignment with Bowtie 2", *Nature Methods*, vol. 9, no. 4, pp. 357–359, Mar. 2012, ISSN: 1548-7105. DOI: 10.1038/nmeth.1923.

[201] S. Marco-Sola, M. Sammeth, R. Guigó, and P. Ribeca, "The GEM mapper: Fast, accurate and versatile alignment by filtration", *Nature Methods*, vol. 9, no. 12, pp. 1185–1188, Oct. 2012, ISSN: 1548-7105. DOI: 10.1038/nmeth.2221.

[202] P. Ferragina and G. Manzini, "Opportunistic data structures with applications", in *Proceedings 41st Annual Symposium on Foundations of Computer Science*, ser. SFCS-00, IEEE Comput. Soc, pp. 390–398. DOI: 10.1109/sfcs.2000.892127.

[203] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool", *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403–410, Oct. 1990, ISSN: 0022-2836. DOI: 10.1016/s0022-2836(05)80360-2.

[204] S. Altschul, "Gapped BLAST and PSI-BLAST: A new generation of protein database search programs", *Nucleic Acids Research*, vol. 25, no. 17, pp. 3389–3402, Sep. 1997, ISSN: 1362-4962. DOI: 10.1093/nar/25.17.3389.

[205] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins", *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, Mar. 1970, ISSN: 0022-2836. DOI: 10.1016/0022-2836(70)90057-4.

[206] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences", *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, Mar. 1981, ISSN: 0022-2836. DOI: 10.1016/0022-2836(81)90087-5.

[207] O. Gotoh, "An improved algorithm for matching biological sequences", *Journal of Molecular Biology*, vol. 162, no. 3, pp. 705–708, Dec. 1982, ISSN: 0022-2836. DOI: 10.1016/0022-2836(82)90398-9.

[208] A. McKenna et al., "The genome analysis toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data", *Genome Research*, vol. 20, no. 9, pp. 1297–1303, Jul. 2010, ISSN: 1088-9051. DOI: 10.1101/gr.107524.110.

[209] A. Rimmer et al., "Integrating mapping-, assembly- and haplotype-based approaches for calling variants in clinical sequencing applications", *Nature Genetics*, vol. 46, no. 8, pp. 912–918, Jul. 2014, ISSN: 1546-1718. DOI: 10.1038/ng.3036.

[210] R. Luo et al., "Exploring the limit of using a deep neural network on pileup data for germline variant calling", *Nature Machine Intelligence*, vol. 2, no. 4, pp. 220–227, Apr. 2020, ISSN: 2522-5839. DOI: 10.1038/s42256-020-0167-4.

[211] Z. Zheng, S. Li, J. Su, A. W.-S. Leung, T.-W. Lam, and R. Luo, "Symphonizing pileup and full-alignment for deep learning-based long-read variant calling", *Nature Computational Science*, vol. 2, no. 12, pp. 797–803, Dec. 2022, ISSN: 2662-8457. DOI: 10.1038/s43588-022-00387-x.

[212] R. Poplin et al., "A universal SNP and small-indel variant caller using deep neural networks", *Nature Biotechnology*, vol. 36, no. 10, pp. 983–987, Sep. 2018, ISSN: 1546-1696. DOI: 10.1038/nbt.4235.

[213] Oxford Nanopore Technologies, *Medaka: Neural network sequence correction*, Oxford Nanopore Technologies, 2018. [Online]. Available: https://github.com/nanoporetech/medaka.

[214] M. Kolmogorov, J. Yuan, Y. Lin, and P. A. Pevzner, "Assembly of long, error-prone reads using repeat graphs", *Nature Biotechnology*, vol. 37, no. 5, pp. 540–546, Apr. 2019, ISSN: 1546-1696. DOI: 10.1038/s41587-019-0072-8.

[215] S. Koren, B. P. Walenz, K. Berlin, J. R. Miller, N. H. Bergman, and A. M. Phillippy, "Canu: Scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation", *Genome Research*, vol. 27, no. 5, pp. 722–736, Mar. 2017, ISSN: 1549-5469. DOI: 10.1101/gr.215087.116.

[216] R. Vaser, I. Sović, N. Nagarajan, and M. Šikić, "Fast and accurate de novo genome assembly from long uncorrected reads", *Genome Research*, vol. 27, no. 5, pp. 737–746, Jan. 2017, ISSN: 1549-5469. DOI: 10.1101/gr.214270.116.

[217] D. Kim, L. Song, F. P. Breitwieser, and S. L. Salzberg, "Centrifuge: Rapid and sensitive classification of metagenomic sequences", *Genome Research*, vol. 26, no. 12, pp. 1721–1729, Oct. 2016, ISSN: 1549-5469. DOI: 10.1101/gr.210641.116.

[218] H. Sadasivan et al., "Rapid real-time squiggle classification for read until using RawMap", *Archives of Clinical and Biomedical Research*, vol. 07, no. 01, 2023, ISSN: 2572-5017. DOI: 10.26502/acbr.50170318.

[219] S. Kovaka, Y. Fan, B. Ni, W. Timp, and M. C. Schatz, "Targeted nanopore sequencing by real-time mapping of raw electrical signal with UNCALLED", *Nature Biotechnology*, vol. 39, no. 4, pp. 431–441, Nov. 2020, ISSN: 1546-1696. DOI: 10.1038/s41587-020-0731-9.

[220] A. Payne, N. Holmes, T. Clarke, R. Munro, B. J. Debebe, and M. Loose, "Readfish enables targeted nanopore sequencing of gigabase-sized genomes", *Nature Biotechnology*, vol. 39, no. 4, pp. 442–450, Nov. 2020, ISSN: 1546-1696. DOI: 10.1038/s41587-020-00746-x.

[221] D. E. Wood, J. Lu, and B. Langmead, "Improved metagenomic analysis with Kraken 2", *Genome Biology*, vol. 20, no. 1, Nov. 2019, ISSN: 1474-760X. DOI: 10.1186/s13059-019-1891-0.

[222] R. Ounit, S. Wanamaker, T. J. Close, and S. Lonardi, "CLARK: Fast and accurate classification of metagenomic and genomic sequences using discriminative k-mers", *BMC Genomics*, vol. 16, no. 1, Mar. 2015, ISSN: 1471-2164. DOI: 10.1186/s12864-015-1419-2.

[223] J. T. Simpson and R. Durbin, "Efficient de novo assembly of large genomes using compressed data structures", *Genome Research*, vol. 22, no. 3, pp. 549–556, Dec. 2011, ISSN: 1088-9051. DOI: 10.1101/gr.126953.111.

[224] A. Subramaniyan et al., "GenomicsBench: A benchmark suite for genomics", in *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, IEEE, Mar. 2021, pp. 1–12. DOI: 10.1109/ispass51385.2021.00012.

[225] G. Myers, "A fast bit-vector algorithm for approximate string matching based on dynamic programming", *Journal of the ACM*, vol. 46, no. 3, pp. 395–415, May 1999, ISSN: 1557-735X. DOI: 10.1145/316542.316550.

[226] S. Marco-Sola, J. C. Moure, M. Moreto, and A. Espinosa, "Fast gap-affine pairwise alignment using the wavefront algorithm", *Bioinformatics*, vol. 37, no. 4, P. Robinson, Ed., pp. 456–463, Sep. 2020, ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btaa777.

[227] S. Kalikar, C. Jain, M. Vasimuddin, and S. Misra, "Accelerating minimap2 for long-read sequencing applications on modern CPUs", *Nature Computational Science*, vol. 2, no. 2, pp. 78–83, Feb. 2022, ISSN: 2662-8457. DOI: 10.1038/s43588-022-00201-8.

[228] H. Suzuki and M. Kasahara, "Introducing difference recurrence relations for faster semi-global alignment of long sequences", *BMC Bioinformatics*, vol. 19, no. S1, Feb. 2018, ISSN: 1471-2105. DOI: 10.1186/s12859-018-2014-8.

[229] N. J. Loman, J. Quick, and J. T. Simpson, "A complete bacterial genome assembled de novo using only nanopore sequencing data", *Nature Methods*, vol. 12, no. 8, pp. 733–735, Jun. 2015, ISSN: 1548-7105. DOI: 10.1038/nmeth.3444.

[230] H. Gamaarachchi et al., "GPU accelerated adaptive banded event alignment for rapid comparative nanopore signal analysis", *BMC Bioinformatics*, vol. 21, no. 1, Aug. 2020, ISSN: 1471-2105. DOI: 10.1186/s12859-020-03697-x.

[231] Nanopore WGS Consortium, *Nanopore whole genome sequencing of human reference genome NA12878*, https://github.com/nanopore-wgs-consortium/NA12878.

[232] M. Šošić and M. Šikić, "Edlib: A C/C++ library for fast, exact sequence alignment using edit distance", *Bioinformatics*, vol. 33, no. 9, J. Hancock, Ed., pp. 1394–1395, Jan. 2017, ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btw753.

[233] M. Rautiainen and T. Marschall, "GraphAligner: Rapid and versatile sequence-to-graph alignment", *Genome Biology*, vol. 21, no. 1, Sep. 2020, ISSN: 1474-760X. DOI: 10.1186/s13059-020-02157-2.

[234] A. Ahmadi, A. Behm, N. Honnalli, C. Li, L. Weng, and X. Xie, "Hobbes: Optimized gram-based methods for efficient read alignment", *Nucleic Acids Research*, vol. 40, no. 6, e41–e41, Dec. 2011, ISSN: 0305-1048. DOI: 10.1093/nar/gkr1246.

[235] National Center for Biotechnology Information, *NCBI sequence read archive (SRA)*, https://www.ncbi.nlm.nih.gov/sra.

[236] K.-M. Chao, W. R. Pearson, and W. Miller, "Aligning two sequences within a specified diagonal band", *Bioinformatics*, vol. 8, no. 5, pp. 481–487, 1992, ISSN: 1460-2059. DOI: 10.1093/bioinformatics/8.5.481.

[237] Pacific Biosciences, *Caenorhabditis elegans 40× coverage dataset*, http://datasets.pacb.com.s3.amazonaws.com/2014/c_elegans/list.html.

[238] M. A. Eberle et al., "A reference data set of 5.4 million phased human variants validated by genetic inheritance from sequencing a three-generation 17-member pedigree", *Genome Research*, vol. 27, no. 1, pp. 157–164, Nov. 2016, ISSN: 1549-5469. DOI: 10.1101/gr.210500.116.

[239] M. Burrows, "A block-sorting lossless data compression algorithm", Systems Research Center (SRC), Tech. Rep. 124, 1994.

[240] M. Goyal, B. Fan, X. Li, D. G. Andersen, and M. Kaminsky, *Libcuckoo: A high-performance, concurrent hash table*, Carnegie Mellon University and Intel Corporation, 2013. [Online]. Available: https://github.com/efficient/libcuckoo.

[241] M. Kolmogorov, *Example datasets for the Flye assembler*, 2018. DOI: 10.5281/ZENODO.1172816.

[242] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library", in *Advances in Neural Information Processing Systems 32*, H. Wallach et al., Eds., Curran Associates, Inc., 2019, pp. 8024–8035.

[243] Martín Abadi et al., *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: https://www.tensorflow.org/.

[244] J. M. Zook et al., "Extensive sequencing of seven human genomes to characterize benchmark reference materials", *Scientific Data*, vol. 3, no. 1, Jun. 2016, ISSN: 2052-4463. DOI: 10.1038/sdata.2016.25.

[245] HKU-BAL, *Clair3 ONT model: R941_prom_hac_g360+g422*, http://www.bio8.cs.hku.hk/clair3/clair3_models/r941_prom_hac_g360+g422.tar.gz, University of Hong Kong.

[246] H. Li et al., "The sequence alignment/map format and SAMtools", *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, Jun. 2009, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btp352.

[247] C. Lee, C. Grasso, and M. F. Sharlow, "Multiple sequence alignment using partial order graphs", *Bioinformatics*, vol. 18, no. 3, pp. 452–464, Mar. 2002, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/18.3.452.

[248] C. Lee, "Generating consensus sequences from partial order multiple sequence alignment graphs", *Bioinformatics*, vol. 19, no. 8, pp. 999–1008, May 2003, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btg109.

[249] *Spoa: SIMD partial order alignment tool/library*. [Online]. Available: https://github.com/rvaser/spoa.

[250] A. Guarracino, N. Mwaniki, S. Marco-Sola, and E. Garrison, *Wfmash: Whole-chromosome pairwise alignment using the hierarchical wavefront algorithm*, 2021. [Online]. Available: https://github.com/ekg/wfmash.

[251] B. Song, S. Marco-Sola, M. Moreto, L. Johnson, E. S. Buckler, and M. C. Stitzer, "AnchorWave: Sensitive alignment of genomes with high sequence diversity, extensive structural polymorphism, and whole-genome duplication", *Proceedings of the National Academy of Sciences*, vol. 119, no. 1, Dec. 2021, ISSN: 1091-6490. DOI: 10.1073/pnas.2113075119.

[252] L. Pipes and R. Nielsen, "AncestralClust: Clustering of divergent nucleotide sequences by ancestral sequence reconstruction using phylogenetic trees", *Bioinformatics*, vol. 38, no. 3, I. Birol, Ed., pp. 663–670, Oct. 2021, ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btab723.

[253] R. Okazaki et al., "Supercomputer fugaku CPU A64FX realizing high performance, high-density packaging, and low power consumption", *Fujitsu Technical Review*, pp. 2020–03, 2020.

[254] *A64FX microarchitecture manual*, https://github.com/fujitsu/A64FX/tree/master/doc, Tokyo, Japan: Fujitsu, 2020.

[255] L. McVoy et al., "Lmbench: Portable tools for performance analysis", in *Proceedings of the 1996 Annual Conference on USENIX Annual Technical Conference*, ser. ATEC '96, San Diego, CA: USENIX Association, 1996, p. 23.

[256] N. Stephens et al., "The ARM scalable vector extension", *IEEE Micro*, vol. 37, no. 2, pp. 26–39, Mar. 2017, ISSN: 0272-1732. DOI: 10.1109/mm.2017.35.

[257] *Arm Neoverse V1 core technical reference manual*, https://developer.arm.com/documentation/101427/0102/, Cambridge, UK: Arm, 2019.

[258] J. Doweck et al., "Inside 6th-generation Intel core: New microarchitecture code-named Skylake", *IEEE Micro*, vol. 37, no. 2, pp. 52–62, Mar. 2017, ISSN: 1937-4143. DOI: 10.1109/mm.2017.38.

[259] G. Lento, "Optimizing performance with Intel advanced vector extensions", *Intel White Paper*, vol. 1, p. 2, 2014.

[260] *High performance computing: Tuning guide for AMD EPYC 7002 series processors*, https://developer.amd.com/wp-content/resources/56827-1-0.pdf, Santa Clara, California, USA: Advanced Micro Devices, 2020.

[261] D. Suggs, M. Subramony, and D. Bouvier, "The AMD "zen 2" processor", *IEEE Micro*, vol. 40, no. 2, pp. 45–52, Mar. 2020, ISSN: 1937-4143. DOI: 10.1109/mm.2020.2974217.

[262] *SIMD Everywhere (SIMDe)*, https://github.com/simd-everywhere/simde.

[263] R. Langarita, A. Armejach, P. Ibáñez, J. Alastruey-Benedé, and M. Moretó, "Porting and optimizing BWA-MEM2 using the Fujitsu A64FX processor", *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 20, no. 5, pp. 3139–3153, Sep. 2023, ISSN: 2374-0043. DOI: 10.1109/tcbb.2023.3264514.

[264] *PyTorch for Arm Neoverse Graviton3*, https://hub.docker.com/r/armswdev/pytorch-arm-neoverse, Arm Software Development.

[265] *TensorFlow Arm neoverse*, https://hub.docker.com/r/armswdev/tensorflow-arm-neoverse, Arm Software Development.

[266] F. Bellard et al., "QEMU, a fast and portable dynamic translator", in *Usenix ATC, Freenix Track*, 2005, pp. 41–46.

[267] F. Mantovani et al., "Software development vehicles to enable extended and early co-design: A RISC-V and HPC case of study", in *High Performance Computing*. Springer Nature Switzerland, 2023, pp. 526–537, ISBN: 9783031408434. DOI: 10.1007/978-3-031-40843-4_39.

[268] V. Pillet, J. Labarta, T. Cortes, and S. Girona, "Paraver: A tool to visualize and analyze parallel code", in *Proceedings of WoTUG-18: transputer and occam developments*, vol. 44, 1995, pp. 17–31.

[269] D. Bruening, "Efficient, transparent, and comprehensive runtime code manipulation", Ph.D. dissertation, Massachusetts Institute of Technology, Department of Electrical Engineering ..., 2004.

[270] R. E. Grant, M. Levenhagen, S. L. Olivier, D. DeBonis, K. T. Pedretti, and J. H. Laros III, "Standardizing power monitoring and control at exascale", *Computer*, vol. 49, no. 10, pp. 38–46, Oct. 2016, ISSN: 0018-9162. DOI: 10.1109/mc.2016.308.

[271] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "RAPL: Memory power estimation and capping", in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, ser. ISLPED'10, ACM, Aug. 2010, pp. 189–194. DOI: 10.1145/1840845.1840883.

[272] Broad Institute (BI), *Sequence read archive experiment SRX020470*, https://www.ncbi.nlm.nih.gov/sra/SRX020470.

[273] NCBI, *Sequence read archive experiment SRX207170*, https://www.ncbi.nlm.nih.gov/sra/SRX207170.

[274] NCBI, *Sequence read archive experiment SRX206890*, https://www.ncbi.nlm.nih.gov/sra/SRX206890.

[275] University of California Santa Cruz, *Genome Reference Consortium Human Reference 38*, http://hgdownload.cse.ucsc.edu/goldenPath/hg38/bigZips/.

[276] J. J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK benchmark: Past, present and future", *Concurrency and Computation: Practice and Experience*, vol. 15, no. 9, pp. 803–820, Jul. 2003, ISSN: 1532-0634. DOI: 10.1002/cpe.728.

[277] S. Beamer et al., "The GAP benchmark suite", 2015.

[278] L. Wang et al., "BigDataBench: A big data benchmark suite from internet services", in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, Feb. 2014, pp. 488–499. DOI: 10.1109/hpca.2014.6835958.

[279] D. A. Bader, Y. Li, T. Li, and V. Sachdeva, "BioPerf: A benchmark suite to evaluate high-performance computer architecture on bioinformatics applications", in *IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005.*, IEEE, pp. 163–173. DOI: 10.1109/iiswc.2005.1526013.

[280] W. R. Pearson and D. J. Lipman, "Improved tools for biological sequence comparison.", *Proceedings of the National Academy of Sciences*, vol. 85, no. 8, pp. 2444–2448, Apr. 1988, ISSN: 1091-6490. DOI: 10.1073/pnas.85.8.2444.

[281] K. Albayraktaroglu et al., "BioBench: A benchmark suite of bioinformatics applications", in *IEEE International Symposium on Performance Analysis of Systems and Software, 2005. ISPASS 2005.*, IEEE, 2005, pp. 2–9. DOI: 10.1109/ispass.2005.1430554.

[282] *BioBench2*, https://github.com/reiverjohn/biobench2.

[283] J. R. Miller, S. Koren, and G. Sutton, "Assembly algorithms for next-generation sequencing data", *Genomics*, vol. 95, no. 6, pp. 315–327, Jun. 2010, ISSN: 0888-7543. DOI: 10.1016/j.ygeno.2010.03.001.

[284] M. Alser et al., "Accelerating genome analysis: A primer on an ongoing journey", *IEEE Micro*, vol. 40, no. 5, pp. 65–75, Sep. 2020, ISSN: 1937-4143. DOI: 10.1109/mm.2020.3013728.

[285] H. Li and N. Homer, "A survey of sequence alignment algorithms for next-generation sequencing", *Briefings in Bioinformatics*, vol. 11, no. 5, pp. 473–483, May 2010, ISSN: 1477-4054. DOI: 10.1093/bib/bbq015.

[286] A. Zielezinski, S. Vinga, J. Almeida, and W. M. Karlowski, "Alignment-free sequence comparison: Benefits, applications, and tools", *Genome Biology*, vol. 18, no. 1, Oct. 2017, ISSN: 1474-760X. DOI: 10.1186/s13059-017-1319-7.

[287] Y. Turakhia, G. Bejerano, and W. J. Dally, "Darwin: A genomics co-processor provides up to 15,000x acceleration on long read assembly", 2, vol. 53, Association for Computing Machinery (ACM), Mar. 2018, pp. 199–213. DOI: 10.1145/3296957.3173193.

[288] A. Nag et al., "GenCache: Leveraging in-cache operators for efficient sequence alignment", in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52, ACM, Oct. 2019, pp. 334–346. DOI: 10.1145/3352460.3358308.

[289] D. Fujiki et al., "GenAx: A genome sequencing accelerator", in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, IEEE, Jun. 2018, pp. 69–82. DOI: 10.1109/isca.2018.00017.

[290] H. Sadasivan, D. Stiffler, A. Tirumala, J. Israeli, and S. Narayanasamy, "Accelerated dynamic time warping on GPU for selective nanopore sequencing", *Journal of Biotechnology and Biomedicine*, vol. 07, no. 01, 2024, ISSN: 2642-9128. DOI: 10.26502/jbb.2642-91280134.

[291] T. Dunn et al., "SquiggleFilter: An accelerator for portable virus detection", in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21, ACM, Oct. 2021, pp. 535–549. DOI: 10.1145/3466752.3480117.

[292] P. J. Shih, H. Saadat, S. Parameswaran, and H. Gamaarachchi, "Efficient real-time selective genome sequencing on resource-constrained devices", *GigaScience*, vol. 12, Dec. 2022, ISSN: 2047-217X. DOI: 10.1093/gigascience/giad046.

[293] T. Robinson, J. Harkin, and P. Shukla, "Hardware acceleration of genomics data analysis: Challenges and opportunities", *Bioinformatics*, vol. 37, no. 13, J. Wren, Ed., pp. 1785–1795, May 2021, ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btab017.

[294] M. Poorhosseini, W. Nebel, and K. Gruttner, "A compiler comparison in the RISC-V ecosystem", in *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, IEEE, Aug. 2020, pp. 1–6. DOI: 10.1109/coins49042.2020.9191411.

[295] N. Adit and A. Sampson, "Performance left on the table: An evaluation of compiler autovectorization for RISC-V", *IEEE Micro*, vol. 42, no. 5, pp. 41–48, Sep. 2022, ISSN: 1937-4143. DOI: 10.1109/mm.2022.3184867.

[296] Q. Ducasse, G. Polito, P. Tesone, P. Cotret, and L. Lagadec, "Porting a JIT compiler to RISC-V: Challenges and opportunities", in *Proceedings of the 19th International Conference on Managed Programming Languages and Runtimes*, ser. MPLR '22, ACM, Sep. 2022, pp. 112–118. DOI: 10.1145/3546918.3546924.

[297] E. Cui, T. Li, and Q. Wei, "RISC-V instruction set architecture extensions: A survey", *IEEE Access*, vol. 11, pp. 24 696–24 711, 2023, ISSN: 2169-3536. DOI: 10.1109/access.2023.3246491.

[298] V. Soria-Pardos et al., "Sargantana: A 1 ghz+ in-order RISC-V processor with SIMD vector extensions in 22nm FD-SOI", in *2022 25th Euromicro Conference on Digital System Design (DSD)*, IEEE, Aug. 2022, pp. 254–261. DOI: 10.1109/dsd57027.2022.00042.

[299] Z. Wu, K. Hammad, A. Beyene, Y. Dawji, E. Ghafar-Zadeh, and S. Magierowski, "An FPGA implementation of a portable DNA sequencing device based on RISC-V", in *2022 20th IEEE Interregional NEWCAS Conference (NEWCAS)*, IEEE, Jun. 2022, pp. 417–420. DOI: 10.1109/newcas52662.2022.9842014.

[300] L. D. Tucci, R. Baghdadi, S. Amarasinghe, and M. D. Santambrogio, "SALSA: A domain specific architecture for sequence alignment", in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, IEEE, May 2020, pp. 147–150. DOI: 10.1109/ipdpsw50202.2020.00033.

[301] S. Pircher, J. Geier, A. Zeh, and D. Mueller-Gritschneder, "Exploring the RISC-V vector extension for the classic McEliece post-quantum cryptosystem", in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, IEEE, Apr. 2021, pp. 401–407. DOI: 10.1109/isqed51717.2021.9424273.

[302] M. Cococcioni, F. Rossi, E. Ruffaldi, and S. Saponara, "Vectorizing posit operations on RISC-V for faster deep neural networks: Experiments and comparison with ARM SVE", *Neural Computing and Applications*, vol. 33, no. 16, pp. 10 575–10 585, Feb. 2021, ISSN: 1433-3058. DOI: 10.1007/s00521-021-05814-0.

[303] C. Ramírez, C. A. Hernández, O. Palomar, O. Unsal, M. A. Ramírez, and A. Cristal, "A RISC-V simulator and benchmark suite for designing and evaluating vector architectures", *ACM Transactions on Architecture and Code Optimization*, vol. 17, no. 4, pp. 1–30, Nov. 2020, ISSN: 1544-3973. DOI: 10.1145/3422667.

[304] A. Dörflinger et al., "A comparative survey of open-source application-class RISC-V processor implementations", in *Proceedings of the 18th ACM International Conference on Computing Frontiers*, ser. CF '21, ACM, May 2021, pp. 12–20. DOI: 10.1145/3457388.3458657.

[305] A. Sarihi, M. A. Schoenfelder, and A.-H. A. Badawy, "Performance evaluation of an out-of-order RISC-V CPU: A SPEC INT 2017 study", in *2022 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, IEEE, Nov. 2022, pp. 418–419. DOI: 10.1109/ipccc55026.2022.9894297.

[306] Y. Liu, K. Ye, and C.-Z. Xu, "Performance evaluation of various RISC processor systems: A case study on ARM, MIPS and RISC-V", in *Cloud Computing - CLOUD 2021*, Springer International Publishing, 2022, pp. 61–74, ISBN: 9783030963262. DOI: 10.1007/978-3-030-96326-2_5.

[307] T. Odajima, Y. Kodama, M. Tsuji, M. Matsuda, Y. Maruyama, and M. Sato, "Preliminary performance evaluation of the Fujitsu A64FX using HPC applications", in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, IEEE, Sep. 2020, pp. 523–530. DOI: 10.1109/cluster49012.2020.00075.

[308] S. Goodwin, J. D. McPherson, and W. R. McCombie, "Coming of age: Ten years of next-generation sequencing technologies", *Nature Reviews Genetics*, vol. 17, no. 6, pp. 333–351, May 2016, ISSN: 1471-0064. DOI: 10.1038/nrg.2016.49.

[309] S. P. Pfeifer, "From next-generation resequencing reads to a high-quality variant data set", *Heredity*, vol. 118, no. 2, pp. 111–124, Oct. 2016, ISSN: 1365-2540. DOI: 10.1038/hdy.2016.102.

[310] M. Alser et al., "Technology dictates algorithms: Recent developments in read alignment", *Genome Biology*, vol. 22, no. 1, Aug. 2021, ISSN: 1474-760X. DOI: 10.1186/s13059-021-02443-7.

[311] S. Schbath, V. Martin, M. Zytnicki, J. Fayolle, V. Loux, and J.-F. Gibrat, "Mapping reads on a genomic sequence: An algorithmic overview and a practical comparative analysis", *Journal of Computational Biology*, vol. 19, no. 6, pp. 796–813, Jun. 2012, ISSN: 1557-8666. DOI: 10.1089/cmb.2012.0022.

[312] X. Liao, M. Li, Y. Zou, F.-X. Wu, Yi-Pan, and J. Wang, "Current challenges and solutions of de novo assembly", *Quantitative Biology*, vol. 7, no. 2, pp. 90–109, Jun. 2019, ISSN: 2095-4697. DOI: 10.1007/s40484-019-0166-9.

[313] M. Jain et al., "Nanopore sequencing and assembly of a human genome with ultra-long reads", *Nature Biotechnology*, vol. 36, no. 4, pp. 338–345, Jan. 2018, ISSN: 1546-1696. DOI: 10.1038/nbt.4060.

[314] E. Garrison et al., "Building pangenome graphs", *Nature Methods*, vol. 21, no. 11, pp. 2008–2012, Oct. 2024, ISSN: 1548-7105. DOI: 10.1038/s41592-024-02430-3.

[315] T. Wang et al., "The human pangenome project: A global resource to map genomic diversity", *Nature*, vol. 604, no. 7906, pp. 437–446, Apr. 2022, ISSN: 1476-4687. DOI: 10.1038/s41586-022-04601-8.

[316] E. Ukkonen, "Algorithms for approximate string matching", *Information and Control*, vol. 64, no. 1-3, pp. 100–118, Jan. 1985, ISSN: 0019-9958. DOI: 10.1016/s0019-9958(85)80046-2.

[317] H. Li, "Minimap and miniasm: Fast mapping and de novo assembly for noisy long sequences", *Bioinformatics*, vol. 32, no. 14, pp. 2103–2110, Mar. 2016, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btw152.

[318] J. Daily, "Parasail: SIMD C library for global, semi-global, and local pairwise sequence alignments", *BMC Bioinformatics*, vol. 17, no. 1, Feb. 2016, ISSN: 1471-2105. DOI: 10.1186/s12859-016-0930-z.

[319] D. S. Hirschberg, "A linear space algorithm for computing maximal common subsequences", *Communications of the ACM*, vol. 18, no. 6, pp. 341–343, Jun. 1975, ISSN: 1557-7317. DOI: 10.1145/360825.360861.

[320] Y. Wei, T. Zhou, Y. Zhai, L. Yu, and Q. Zou, "FORAlign: Accelerating gap-affine DNA pairwise sequence alignment using FOR-blocks based on Four Russians approach with linear space complexity", *Briefings in Bioinformatics*, vol. 26, no. 1, Nov. 2024, ISSN: 1477-4054. DOI: 10.1093/bib/bbaf061.

[321] V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradzev, "On economical construction of the transitive closure of a directed graph", in *Dokl. Akad. Nauk SSSR*, vol. 194, 1970, pp. 1209–1210.

[322] S. Marco-Sola, J. M. Eizenga, A. Guarracino, B. Paten, E. Garrison, and M. Moreto, "Optimal gap-affine alignment in O(s) space", *Bioinformatics*, vol. 39, no. 2, btad074, Feb. 2023, ISSN: 1367-4811. DOI: 10.1093/bioinformatics/btad074.

[323] T. F. Smith, M. S. Waterman, and W. M. Fitch, "Comparative biosequence metrics", *Journal of Molecular Evolution*, vol. 18, no. 1, pp. 38–46, Jan. 1981, ISSN: 1432-1432. DOI: 10.1007/bf01733210.

[324] O. Gotoh, "Optimal sequence alignment allowing for long gaps", *Bulletin of Mathematical Biology*, vol. 52, no. 3, pp. 359–373, May 1990, ISSN: 1522-9602. DOI: 10.1007/bf02458577.

[325] K. H. Miga and T. Wang, "The need for a human pangenome reference sequence", *Annual Review of Genomics and Human Genetics*, vol. 22, no. 1, pp. 81–102, Aug. 2021, ISSN: 1545-293X. DOI: 10.1146/annurev-genom-120120-081921.

[326] A. Jiménez-Blanco, L. López-Villellas, J. C. Moure, M. Moreto, and S. Marco-Sola, "Theseus: Fast and optimal affine-gap sequence-to-graph alignment", Feb. 2026. DOI: 10.64898/2026.02.12.705572.

[327] Z. Zlatev, I. Dimov, I. Faragó, and Á. Havasi, *Richardson Extrapolation: Practical Aspects and Applications*. De Gruyter, Nov. 2017, pp. v–xii, ISBN: 9783110533002. DOI: 10.1515/9783110533002-001.