Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Adaptive Patch Grid Strategy for Parallel Protein Folding using Atomic Burials with NAMD

Emerson de Araujo Macedo

Brasília
2025

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Adaptive Patch Grid Strategy for Parallel Protein Folding using Atomic Burials with NAMD

Emerson de Araujo Macedo

Thesis submitted as a partial requirement
for completion of the Ph.D. in Informatics

Advisor
Prof.ª Dr.ª Alba Cristina M. A. de Melo

Brasília
2025

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Doutorado em Informática

Coordenador: Prof. Dr. Rodrigo Bonifácio Almeida

Banca examinadora composta por:

Prof.ª Dr.ª Alba Cristina M. A. de Melo (Orientadora) — CIC/UnB
Prof. Dr. Márcio Dorn — INF/UFRGS
Prof. Dr. Mário Antônio Ribeiro Dantas — ICE/UFJF
Prof. Dr. Ricardo Pezzuol Jacobi — CIC/UnB

CIP — Catalogação Internacional na Publicação

de Araujo Macedo, Emerson.

Adaptive Patch Grid Strategy for Parallel Protein Folding using Atomic Burials with NAMD / Emerson de Araujo Macedo. Brasília : UnB, 2025.
167 p. : il. ; 29,5 cm.

Tese (Doutorado) — Universidade de Brasília, Brasília, 2025.

1. Protein Folding, 2. *ab initio* Molecular Dynamics, 3. Parallel Strategies, 4. High-Performance Computing, 5. Adaptive Patch Grid, 6. Atomic Burial, 7. NAMD, 8. Simulation Acceleration

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Adaptive Patch Grid Strategy for Parallel Protein Folding using Atomic Burials with NAMD

Emerson de Araujo Macedo

Tese apresentada como requisito parcial
para a conclusão do Doutorado em Informática

Prof.ª Dr.ª Alba Cristina M. A. de Melo (Orientadora)
CIC/UnB

Prof. Dr. Márcio Dorn
INF/UFRGS

Prof. Dr. Mário Antônio Ribeiro Dantas
ICE/UFJF

Prof. Dr. Ricardo Pezzuol Jacobi
CIC/UnB

Prof. Dr. Rodrigo Bonifácio Almeida
Coordenador do Programa em Pós-Graduação em Informática

Brasília, 11 de março de 2025

Dedication

I dedicate this work to my beautiful wife, Little Princess Bunny, and our two wonderful children, Malu and Pepê. Your unwavering love, joy, and patience have been my greatest sources of strength throughout this journey. This project demanded immense energy and focus, and I could not have completed it without knowing I could always count on your support. You inspire me every day, and I love you deeply.

Acknowledgments

I would like to express my heartfelt gratitude to my parents, who taught me the values of faith, hard work, and dedication, instilling in me a love for God and a deep appreciation for a fulfilling life. Their unwavering support and love have been a great source of strength.

To my advisor, Professor Dr. Alba Cristina Magalhães Alves de Melo from the Graduate Program in Informatics at the Department of Computer Science at the University of Brasília (CIC-UnB), I owe a profound debt of gratitude. Her insights, patience, and leadership have guided me through each phase of my research, pushing me to grow as a researcher and an individual.

I am also thankful to Associate Professor Gerson Henrique Pfitscher from the Department of Electrical Engineering (ENE). His collaboration in the initial stages of my research provided a valuable foundation, and his friendship has been a source of encouragement and intellectual exchange ever since.

I extend my appreciation to Professor Dr. Antônio Francisco Pereira de Araújo of the Laboratory of Theoretical and Computational Biology (LBTC) at the University of Brasília (UnB), Dr. Marx Gomes van der Linden, and Dr. Diogo César Ferreira. Their research on Atomic Burial has provided foundational insights that were invaluable to this study.

I am grateful to the Barcelona Supercomputing Center (BSC) for providing access to the Marenostrum 4 supercomputer and the Nord HPC cluster, which were essential for the simulations in this research.

To the Department of Computer Science at the University of Brasília (CIC-UnB), I am deeply grateful. Its resources, facilities, and academic environment have supported my growth as a scholar in every way.

Finally, my thanks to the University of Brasília (UnB) for providing a fertile ground for learning and discovery, and to all others who supported me on this journey. Your encouragement has made this achievement possible.

Abstract

Protein folding is a crucial process in molecular biology for understanding the structural and functional dynamics of proteins. Molecular dynamics simulations are important for providing atomic-level insights into the folding process, enabling the investigation of structural changes over time. However, these simulations face many challenges, such as high computational costs, inefficient load balancing, difficulties in scaling simulations for large systems, and limited accuracy in representing complex interactions like hydrogen bonding and solvation effects. This PhD Thesis aims to address these challenges by investigating parallel strategies to accelerate *ab initio* molecular dynamics simulations of protein folding while maintaining accuracy. To overcome load imbalance caused by static spatial decomposition methods, which becomes critical as the folding progresses, this work introduces a dynamic approach. Specifically, this work proposes an Adaptive Patch Grid (APG) strategy that dynamically adjusts spatial decomposition throughout the folding process to improve load balancing and efficiency in molecular dynamics simulations. As an ab initio approach, these simulations derive atomic interactions from first principles, demanding high computational resources and advanced parallelization strategies. Additionally, this PhD Thesis proposes the N2HB algorithm, which addresses the inadequacy of force fields in capturing key structural forces during folding by introducing atomic burial and hydrogen bonding potentials to enhance model realism. Both proposals (APG and N2HB) were implemented using the NAMD platform, a parallel molecular dynamics software. New components such as ComputeBurialForce and ComputeHBonds were added to enable parallel computation of burial and hydrogen bonding forces. Annealing weights were applied to optimize the energy minimization process during folding. The solutions were evaluated through extensive testing on various High-Performance Computing (HPC) systems, showing reduced execution times while maintaining good simulation accuracy. This PhD Thesis contributes to the field by providing parallelization strategies that enhance the performance of molecular dynamics simulations for protein folding, addressing computational limitations and offering methods to scale these simulations for more complex biological systems. This PhD Thesis advances the field by introducing scalable and biologically relevant enhancements to molecular dynamics simulations, supporting the accurate and efficient modeling of protein folding in complex systems.

Keywords: Protein Folding, *ab initio* Molecular Dynamics, Parallel Strategies, High-Performance Computing, Adaptive Patch Grid, Atomic Burial, NAMD, Simulation Acceleration

Resumo

O dobramento de proteínas é um processo crucial na biologia molecular para a compreensão da dinâmica estrutural e funcional das proteínas. As simulações de dinâmica molecular são importantes para fornecer insights em nível atômico sobre o processo de dobramento, permitindo a investigação das mudanças estruturais ao longo do tempo. No entanto, essas simulações enfrentam muitos desafios, como o alto custo computacional, desbalanceamento de carga durante a execução paralela, dificuldades de escalonamento para grandes sistemas e limitações na representação precisa de interações moleculares complexas, como as ligações de hidrogênio e os efeitos de solvatação. Esta Tese de Doutorado tem como objetivo abordar esses desafios por meio da investigação de estratégias paralelas para acelerar simulações de dinâmica molecular *ab initio* aplicadas ao dobramento de proteínas, sem comprometer a precisão dos resultados. Para enfrentar o problema do desbalanceamento de carga causado pela decomposição espacial estática, especialmente em simulações com estruturas em constante mudança, esta Tese apresenta uma abordagem dinâmica para reorganização espacial ao longo do tempo. Especificamente, este trabalho propõe uma estratégia Adaptive Patch Grid (APG) que ajusta dinamicamente a decomposição espacial ao longo do processo de dobramento para melhorar o balanceamento de carga e a eficiência nas simulações de dinâmica molecular. Como uma abordagem *ab initio*, essas simulações derivam as interações atômicas a partir de princípios físicos fundamentais, exigindo alto poder computacional e estratégias avançadas de paralelização. Além disso, esta Tese propõe o algoritmo N2HB, que visa melhorar a precisão estrutural das simulações ao incorporar potenciais de enterramento atômico e - forças essenciais que não são bem representadas por modelos de campo de força tradicionais. Ambas as propostas (APG e N2HB) foram implementadas usando a plataforma NAMD, um software paralelo de dinâmica molecular. Novos componentes, como ComputeBurialForce e ComputeHBonds, foram adicionados para permitir o cálculo paralelo das forças de enterramento e Pesos de anelamento foram aplicados para otimizar o processo de minimização de energia durante o dobramento. As soluções foram avaliadas por meio de extensivos testes em vários sistemas HPC, demonstrando redução nos tempos de execução enquanto mantinham boa precisão na simulação. Esta Tese contribui com soluções escaláveis e biologicamente relevantes para simulações de dinâmica molecular, ampliando a capacidade de modelar com precisão o dobramento de proteínas em sistemas complexos.

Palavras-chave: Protein Folding, *ab initio* Molecular Dynamics, Parallel Strategies, High-Performance Computing, Adaptive Patch Grid, Atomic Burial, NAMD, Simulation Acceleration

Estratégia Adaptativa de Grade em Blocos para Enovelamento Paralelo de Proteínas Utilizando Enterramentos Atômicos com o NAMD

Resumo Expandido

Introdução

As proteínas são componentes vitais dos sistemas biológicos, desempenhando funções essenciais como a catálise de reações metabólicas, o suporte estrutural celular e a comunicação entre células. A compreensão dos processos biológicos e das doenças depende, em grande parte, da elucidação de como as proteínas adquirem suas estruturas tridimensionais funcionais. Entretanto, o processo de enovelamento proteico é extremamente complexo, influenciado por interações intrincadas entre os aminoácidos e o ambiente molecular.

As simulações de dinâmica molecular (MD) surgem como uma abordagem promissora para investigar o enovelamento de proteínas, oferecendo detalhes em nível atômico sobre o comportamento conformacional ao longo do tempo. Essas simulações resolvem as equações de movimento de Newton para prever as mudanças estruturais, mas enfrentam desafios significativos, como o elevado custo computacional e as dificuldades de escalabilidade em sistemas de grande porte.

Modelos com solvente explícito aumentam a complexidade computacional, enquanto modelos implícitos oferecem alternativas mais leves, ainda que com alguma perda de precisão. Além disso, o nível de representação molecular impacta diretamente o custo e a fidelidade da simulação: modelos *Coarse-Grained* permitem simulações mais longas e amplas, ao passo que modelos totalmente atomísticos são essenciais para a captura de interações moleculares detalhadas, como ligações de hidrogênio e efeitos de solvatação.

Para enfrentar essas limitações, estratégias de decomposição espacial e técnicas de paralelização têm sido investigadas no contexto de computação de alto desempenho (HPC), buscando melhorar a distribuição da carga computacional ao longo da execução da simulação. A necessidade de adaptação dinâmica da decomposição espacial torna-se ainda mais crítica em processos como o enovelamento proteico, onde a distribuição atômica varia significativamente ao longo do tempo.

Contribuições

Nesta Tese, propõe-se duas contribuições principais para otimizar simulações de enovelamento de proteínas: a estratégia de decomposição espacial adaptativa chamada Adaptive Patch Grid (APG) e o algoritmo N2HB para a execução paralela da técnica de enterramento atômico (*Atomic Burials*). Ambas as soluções foram integradas ao simulador de dinâmica molecular NAMD, com o objetivo de reduzir o tempo de execução, melhorar a escalabilidade e preservar a precisão estrutural das simulações.

As duas contribuições da Tese atuam de forma complementar para aprimorar simulações de enovelamento de proteínas utilizando o simulador NAMD: a estratégia adaptativa de grade em blocos (APG) e a integração de novas forças de enterramento atômico e ligação de hidrogênio baseadas no algoritmo MDBury.

O NAMD foi escolhido como plataforma de implementação por permitir acesso ao seu código-fonte, apresentar escalabilidade para sistemas com mais de mil átomos e dispor de uma arquitetura modular compatível com a adição de novos componentes. O mecanismo de execução do NAMD foi estendido para incluir algoritmos relacionados à força de enterramento atômico e à força de ligação de hidrogênio definidas no algoritmo MDBury.

A primeira contribuição da Tese, chamada APG, consiste em uma estratégia que adapta dinamicamente a grade em blocos do NAMD ao formato da proteína ao longo da simulação. Essa adaptação visa produzir uma decomposição geométrica mais balanceada entre os núcleos de processamento, promovendo melhor aproveitamento da execução paralela. A abordagem foi implementada de forma a não exigir mudanças na configuração do sistema molecular ou nos parâmetros físicos definidos, utilizando critérios baseados na densidade de átomos simulados para reconfigurar a grade espacial.

O código da APG foi integrado ao fluxo de execução da ferramenta NAMD, respeitando os mecanismos de comunicação entre processos paralelos. Assim, tornou-se possível a adaptação dinâmica da grade em blocos gerada durante a execução, sem comprometer a continuidade da simulação, o que representa uma novidade em relação às estratégias tradicionais que utilizam decomposição fixa.

A segunda contribuição trata da inclusão de novas forças no NAMD, inspiradas no algoritmo MDBury, utilizado para simular o enovelamento de proteínas através de dinâmica molecular com base em energias potenciais de enterramento atômico. Foram incorporados três componentes distintos ao simulador NAMD: um para calcular a força de enterramento atômico, outro para calcular a força de ligação de hidrogênio associada aos átomos enterrados e outro para calcular os pesos de anelamento utilizados na técnica de enterramento atômico.

Esses componentes foram implementados como módulos compatíveis com a execução paralela e respeitando o modelo de distribuição de dados do NAMD. A principal dificuldade enfrentada foi garantir acesso eficiente aos dados dos átomos distribuídos na grade em blocos e necessários para o cálculo local das contribuições de energia e força.

Resultados

As simulações foram realizadas no supercomputador MareNostrum 4, usando entre 48 e 288 núcleos de processamento, e no supercomputador Nord III, utilizando entre 16 e 128 núcleos de processamento. Foram avaliadas quatro proteínas globulares obtidas da base de dados de proteínas PDB: 1ENH (947 átomos), 1IFR (1.746 átomos), 1OZ9 (2.346 átomos) e 4LNZ (5.714 átomos). Essas proteínas apresentam diferentes números de átomos, característica utilizada para avaliar o desempenho e o comportamento das estratégias propostas em contextos variados de simulação.

Os experimentos foram ajustados por meio de arquivos de configuração, que incluíam os parâmetros estruturais da simulação, o uso de solvente implícito e as opções de reinício. Foram conduzidos testes com a grade em blocos padrão, com número ampliado de blocos e com reinício manual. Os resultados mostraram que o aumento manual do número de blocos nem sempre levou à melhoria de desempenho, devido à criação de blocos vazios ou à má distribuição entre os núcleos. O reinício manual revelou que a decomposição podia ser melhorada entre fases, abrindo caminho para a adaptação dinâmica proposta pela APG.

A avaliação considerou tanto o impacto no tempo total de execução quanto a eficiência na distribuição da carga computacional. A Tese apresenta resultados que mostram a viabilidade da abordagem, evidenciando que a integração da grade em blocos adaptativa e da técnica de enterramento atômico ao NAMD pode ser feita sem prejuízo à execução da simulação, apresentando melhor escalabilidade em relação ao MDBury.

Os experimentos realizados nesta Tese permitiram avaliar o desempenho e a escalabilidade das estratégias propostas para simulações de enovelamento de proteínas no NAMD. Foram conduzidos testes com a grade em blocos padrão, ampliada, com reinício manual e com adaptação dinâmica via APG, além da integração das forças de enterramento atômico e ligações de hidrogênio.

Inicialmente, testes com a grade em blocos padrão foram realizados com as proteínas 1ENH, 1IFR e 1OZ9, obtidas da base PDB. As grades geradas pelo NAMD (por exemplo, configurações em 7x3x1 e 19x3x1 blocos) possuíam menos blocos que núcleos, resultando em ociosidade e baixo aproveitamento na execução paralela. Os tempos de execução não mostraram correlação evidente com o número de núcleos, e os tempos de execução foram inconsistentes.

Na sequência, foram realizados testes com grade em blocos ampliada, utilizando os parâmetros *twoAwayX*, *twoAwayY* e *twoAwayZ* no arquivo de configuração. Embora essa abordagem aumentasse o número de blocos, observou-se a criação de blocos "vazios" (sem átomos associados), por exemplo, 154 blocos vazios no teste com a proteína 1OZ9, mostrando uma distribuição espacial ineficiente. Com isso, algumas simulações não foram concluídas dentro das 48 horas reservadas para execução de cada simulação nos supercomputadores.

Os testes com reinício manual da simulação foram realizados com a proteína 4LNZ no supercomputador Nord III. A cada fase, a simulação era interrompida e reiniciada, gerando novas grades em blocos (por exemplo, de 20x7x1 blocos para 7x4x5 blocos). Isso mostrou que alterações na conformação da proteína ao longo

do tempo afetavam a decomposição espacial e que recalcular a grade após essas mudanças melhorava a eficiência.

Com base nessas observações, foi proposta a estratégia adaptativa de grade em blocos (APG). Em simulações com a proteína 4LNZ (20 milhões de iterações), executadas em quatro fases, a APG reconfigurou dinamicamente a grade de 47x3x1 para configurações mais compactas como 6x4x3 e 5x4x4. No supercomputador Nord III com 128 núcleos (8 nós computacionais), o tempo de execução foi reduzido de 34 horas e 18 minutos (1 nó) para 11 horas e 22 minutos com APG (8 nós). Em relação à grade padrão, houve redução de 16 horas e 57 minutos para 11 horas e 22 minutos, confirmando a redução significativa no tempo de execução.

Além disso, a integração do algoritmo N2HB ao NAMD com a APG gerou ganhos adicionais. Com 15 milhões de iterações no supercomputador Nord III, a versão NAMD+APG+AB reduziu o tempo de execução em 2 horas e atingiu até 34 ns/dia, superando os 24 ns/dia da versão padrão com enterramento, mantendo a estabilidade numérica e estrutural da simulação.

Adicionalmente, foram realizados testes com o NAMD+APG+AB utilizando 1,7 bilhão de iterações para simular o enovelamento completo da proteína 4LNZ, correspondendo a $1.7\,\mu s$ simulados. A simulação foi dividida em 17 fases de 100 milhões de iterações, cada uma com 20 subfases de 5 milhões, respeitando o limite de $48\,h$ por execução no supercomputador Nord III.

Durante o experimento, a grade em blocos evoluiu de uma configuração inicial 47x3x1 (141 blocos), condizente com o formato alongado da proteína, para grades mais compactas como 6x4x3, 5x4x4 e 5x8x3 (120 blocos), refletindo as mudanças estruturais do enovelamento ao longo da simulação.

Foram registrados 68.000 quadros com posições atômicas, visualizados com o VMD. A comparação entre o modelo final e a estrutura nativa indicou redução do RMSD de $306.5\,Å$ para $25.0\,Å$. O TM-score obtido foi de 0,1765 e o GDT-TS, de 9,07, refletindo a aproximação da proteína simulada a uma conformação realista. Esses resultados evidenciam que a estratégia NAMD+APG+AB é capaz de conduzir o enovelamento de forma biofisicamente plausível em larga escala.

Conclusões

Esta Tese apresentou duas contribuições para a melhoria do desempenho e da precisão de simulações de enovelamento de proteínas utilizando o simulador NAMD: a estratégia adaptativa de grade em blocos (APG) e o algoritmo N2HB, responsável pela execução paralela de forças de enterramento atômico e ligação de hidrogênio.

A estratégia APG mostrou comportamento escalável aprimorado ao permitir a adaptação dinâmica da grade em blocos ao longo da simulação, conforme as alterações conformacionais da proteína. Essa adaptação resultou em distribuições mais equilibradas entre os núcleos de processamento e reduções no tempo de execução. Os testes mostraram que a reorganização dinâmica da grade foi fundamental para melhorar a eficiência em simulações longas e com formatos moleculares variáveis.

A contribuição N2HB viabilizou a incorporação paralela das forças de enterramento atômico e ligações de hidrogênio ao NAMD. A implementação foi feita por

meio de novos módulos e componentes integrados ao mecanismo de execução do simulador, respeitando sua distribuição de dados. Os resultados confirmaram que a combinação entre APG e N2HB reduziu o tempo total de execução e permitiu a simulação de conformações mais compactas, com fidelidade estrutural biofisicamente realista.

As estratégias desenvolvidas possibilitam avanços significativos no estudo do enovelamento proteico em larga escala e podem ser aplicadas a pesquisas em biologia estrutural, desenvolvimento de fármacos e simulações de sistemas complexos. A metodologia proposta também é compatível com futuras extensões, incluindo simulações multiescalares, modelos híbridos e integração com técnicas de aprendizado de máquina.

Contents

List of Figures

List of Tables

List of Acronyms

AB Atomic Burial. 42, 76, 96–100, 108, 109, 116

AI Artificial Intelligence. 28

AOS Array of Structures. 55

APG Adaptive Patch Grid. vi, vii, ix, 79, 84, 92, 94, 98, 116

API Application Programming Interface. 48

AR Adaptive-Resolution. 69, 70

ARM Advanced RISC Machine. 61, 62

ASIC Application-Specific Integrated Circuit. 74, 75

AW Annealing Weights. 43, 97, 99

BC Bond Calculator. 64

BLAS Basic Linear Algebra Subprograms. 48

CABS C-Alpha, Beta and Side Chain. 23, 24

CASP14 14th Critical Assessment of Structure Prediction. 28

CG Coarce-Grained. 21–24, 72–74, 76

CMP Computational Molecular Physics. 19, 29

CPE Computing Processing Element. 58

CUDA Compute Unified Device Architecture. 48, 51

CVAE Convolutional Variational Encoder. 65

DDR3 Double Data Rate type 3. 58, 66

DTL Data Transport Layer. 68

FFT Fast Fourier Transformation. 63

FG Fine-Grained. 73, 74

FPGA Field Programmable Gate Array. 48

GaMD Gaussian accelerated Molecular Dynamics. 66, 67

GB Generalized Born. 36

GBIS Generalized Born Implicit Solvent. 36

GBSA Generalized Born Surface Area. 73

GC Geometry Core. 64

GP-DC General Purpose/Data Centric Module. 52

GPFS General Parallel File System. 53

GPGPU General Purpose Graphics Processing Unit. 48

GPU Graphics Processing Unit. 4, 38, 47, 50, 52–56, 61, 65, 74

GROMACS GROningen MAchine for Chemical Simulations. 56, 57, 74

HB Hydrogen Bonds. 10, 13, 14, 30, 42, 96–99, 116

HDD Hard Disk Drive. 52

HPC High-Performance Computing. vi, vii, 1–6, 46, 48–54, 56, 58–65, 67, 68, 71, 74–76, 80, 83, 84, 91, 108

HPE Hewlett Packard Enterprise. 47, 50

IDP Intrinsically Disordered Protein. 16

LAMMPS Large-scale Atomic/Molecular Massively Parallel Simulator. 58–60, 74–76

LAN Local Area Network. 51

LAPACK Linear Algebra PACKage. 48

MC Memory Controller. 58

MD Molecular Dynamics. 19, 21, 29, 31–33, 35, 36, 38, 39, 45, 47, 54–77, 91, 92, 103, 113

MELD Modeling Employing Limited Data. 19

MIC Many Integrated Core. 56–58

ML Machine Learning. 19, 65, 66

MPE Management Processing Element. 58

MPI Message Passsing Interface. 38, 48, 49, 51, 60, 62, 73, 74, 83

MPP Massively Parallel Processors. 47–49, 51, 58, 59, 63, 67

NAMD NAnoscale Molecular Dynamics. 6, 36, 38–41, 54, 55, 66–68, 74–76, 79, 81–94, 96–100, 102–111, 114, 116–119

NCAB NAMD Configuration File with AB/HB. 108

NCF NAMD Configuration File. 85, 86, 89–92

NMR Nuclear Magnetic Resonance. 19–21

NoC Network on Chip. 58

NVMe NonVolatile Memory express. 52

OpenCL Open Computing Language. 48

OpenMP OpenMP Open Multi-Processing. 51, 73, 74

ORNL Oak Ridge National Laboratory. 49, 54

PAMI Parallel Active Message Interface. 55

PDB Protein Data Bank. 11, 19, 26, 28, 84, 112, 113

PF Protein Folding. 12, 13, 17, 74, 75

PFLOPS Peta Floating-point Operations Per Second. 47, 59

PPIM Pairwise Point Interaction Modules. 64

PPM Parallel Particle Mesh. 69, 76

RDMA Remote Direct Memory Access. 48, 49, 67, 68, 76

Rmax Maximal LINPACK performance achieved. 47

RMSD Root Mean Square Deviation. 45, 112–114, 118

SBI Structural Bioinformatics. 18, 21, 24

SDRAM Synchronous Dynamic Random-Access Memory. 58

SERDES Serializer/Deserializer functional block. 63, 64

SI System Interface. 58

SIMD Single Instruction Multiple Data. 49, 55, 58, 59, 70

SLURM Simple Linux Utility for Resource Management. 83

SOA Structure of Arrays. 55

SRAM Synchronous Random-Access Memory. 64

SSD Solid State Drive. 52

UNRES UNited RESidue. 23, 72, 73, 76

VAE Variational AutoEncoder. 65

Chapter 1

Introduction

Proteins are vital molecular components of biological systems, responsible for several cellular functions, from catalyzing metabolic reactions to providing structural support and enabling communication among cells. Understanding biological processes and diseases depends heavily on deciphering how proteins fold into their functional three-dimensional structures [1]. However, protein folding is a complex process influenced by intricate interactions among amino acids and their environment, which presents significant challenges for both experimental and computational methods [2].

An *ab initio* method is one that derives its predictions directly from basic physical principles, without relying on empirical parameters. Molecular dynamics simulations, as *ab initio* approaches, are widely used to study protein folding, offering atomic-level detail on molecular behavior over time [3]. These simulations solve Newton's equations of motion for atoms and molecules to track time-dependent conformational changes. While molecular dynamics provides critical insights, it faces challenges such as computational cost and scaling issues, requiring optimization strategies that balance accuracy and computational efficiency to enable large-scale simulations [4–6].

To address these challenges, HPC strategies have been developed to run molecular dynamics simulations efficiently. Explicit solvent models, which account for individual solvent molecules, increase the simulation complexity and benefit from advanced HPC methods, such as those used in studies [7–15]. These approaches include techniques like parallel molecular dynamics simulations on various supercomputers (e.g., Summit, TianHe-2, BlueGene/Q) to speed up simulations without compromising accuracy. On the other hand, implicit solvent models, which approximate the solvent environment rather than simulating it atom by atom, offer a computationally cheaper alternative. These models are typically employed in combination with HPC techniques, as discussed in [16–19], optimizing performance by reducing the number of particles simulated while still capturing key effects.

In addition to the choice of solvent models, the level of molecular representation plays a crucial role in molecular dynamics simulations of protein folding. Coarse-grained models reduce system complexity by grouping atoms into larger pseudo-particles, enabling simulations to access larger spatial and temporal scales with significantly lower computational cost. This abstraction is useful for investigating

global folding patterns and long-timescale events, though it comes at the expense of atomic-level precision [5, 19]. Full-atom models, on the other hand, describe each atom individually, capturing detailed molecular interactions such as hydrogen bonding, electrostatics, and steric effects with high accuracy. These models are particularly valuable for analyzing fine-grained conformational transitions and assessing solvation effects [20, 21]. Despite their higher computational demands, full-atom representations remain the preferred approach when detailed structural insight is required, especially in contexts involving ligand interactions or folding intermediates.

This work addresses the gap between execution time and accuracy in molecular dynamics simulations by focusing on strategies for domain decomposition during protein folding simulations, which adjusts spatial decomposition throughout the folding process, aiming for a more efficient distribution of computational loads across HPC resources. The following sections are structured as follows: Section 1.1 discusses the limitations of molecular dynamics, including the challenges of force field accuracy and computational cost. Section 1.2 presents the motivation behind this Thesis, emphasizing the need for improved parallelization in molecular dynamics simulations. Section 1.3 outlines our objectives and Section 1.4 details the key contributions of this Thesis, respectively. Finally, Section 1.5 summarizes the organization of the rest of the document.

1.1 Problem: Limitations of Molecular Dynamics

Molecular dynamics simulations, while a useful tool for studying molecular systems, have several well-known limitations. One primary issue is the accuracy of some force field models. Such models are, many times, oversimplified approximations of atomic interactions, which makes them computationally manageable but leads to reduced accuracy in representing complex interactions like hydrogen bonding and solvent effects. This can result in low accuracy, particularly when simulating large or structurally changing systems like proteins [4, 6, 22–24]

Another limitation is the challenge of processing the large amount of data generated by molecular dynamics simulations. The data is high-dimensional and often contains noise, making it difficult to extract useful information about the system's behavior. This is especially problematic for large biomolecular systems where significant conformational changes occur over time, requiring sophisticated computational tools for analysis [6, 24].

The computational cost is another major drawback. Simulating even short time periods, such as nanoseconds or microseconds, demands significant computational power. As the size of the system increases or the duration of the simulation extends, the resource requirements grow considerably. In many cases, specialized hardware, like Anton supercomputer or HPC clusters, are required to handle the workloads, but they still fall short when simulating biological processes like protein folding, which occur on much longer timescales [22–25].

Finally, scalability remains a challenge in molecular dynamics simulations of protein folding. While advancements in HPC and parallel computing have im-

proved performance, efficient load balancing remains difficult due to the dynamic and unpredictable nature of molecular systems like proteins. This makes it challenging to fully utilize computational resources, limiting the scalability of these simulations as system size and complexity increase [23, 24, 26, 27].

1.2 Motivation

The complexity and scale of molecular dynamics simulations, particularly those involving protein folding, pose significant computational challenges. One major challenge is executing standard molecular dynamics algorithms in parallel, which is difficult due to the high time-dependency and data-dependency of molecular simulations [3, 28]. This generates the need for highly scalable parallel techniques to fully leverage HPC architectures, which are essential for tackling large-scale molecular simulations [3, 23, 24, 28].

Executing standard molecular dynamics algorithms in parallel is particularly challenging because of the inherent dependencies between data and over time. As the simulation evolves, atomic interactions are be recalculated, and these interdependent calculations make it challenging to achieve scalable parallelization without sacrificing performance or accuracy [23, 24, 26, 27].

Moreover, simulating dynamic biological processes like protein folding involves non-static atomic distributions, making traditional spatial decomposition methods inadequate for large-scale systems. The spatial configuration of atoms changes over time, and static decomposition methods struggle to adapt to this dynamics. Addressing these limitations requires innovative approaches that can dynamically adjust the distribution of computational tasks and improve load balancing, ensuring simulations remain efficient and scalable across different HPC platforms [7–19].

In this context, an important area of investigation is exploring techniques that dynamically adapt spatial decompositions to optimize load distribution across processors. These techniques must not only improve computational efficiency but also enable simulations to handle large and complex biological systems without sacrificing accuracy. Another aspect to investigate is the application of implicit solvent models, which reduces the computational cost by approximating solvent environments, allowing for simulations over biologically meaningful timescales without the overhead of explicit solvation [22, 22, 23, 25, 29].

Over the years, many techniques were developed that used HPC for simulating molecular dynamics, focusing on the parallelization strategies and computational efficiencies required to model complex protein folding processes. With implicit solvent models, these HPC-based simulations require less computing power than the explicit solvent models as they bypass the intensive computational demands of modeling each solvent molecule explicitly, relying instead on approximations that reduce processing costs while still maintaining the key solvent-protein interactions. It must be noted that the computing power for realistic protein folding simulations with implicit solvent models is still very high [16–19]. In an HPC environment, the ability to handle large-scale simulations is supported by tools like

NAMD, GROMACS, and LAMMPS, each optimized for massive parallelism across supercomputer architectures such as Summit, Tianhe, and Sunway [7–9]. These platforms leverage high core counts and advanced Graphics Processing Unit (GPU) architectures to partition the computational space and apply adaptive resolution or domain decomposition techniques, which allow efficient calculation of interactions within defined areas, thus optimizing memory usage and computational time [22, 22, 23, 25, 29].

Although recently proposed tools like AlphaFold [30] have made remarkable advances in protein structure prediction, their capabilities are limited to obtaining the final 3D conformation and do not generate the atom's trajectories over time that led to the final conformation. Unlike molecular dynamics simulations, which can depict the conformational changes of proteins through sequential frames, AlphaFold's predictions focus solely on finding a stable folded structure, not addressing the detailed dynamics of folding events [30]. Consequently, while AlphaFold provides valuable insights into protein structures, it does not offer the trajectory data required to observe real-time protein dynamics. This gap is filled by molecular dynamics in HPC environments, where extensive processing power supports the time-intensive calculations necessary for accurate dynamic and over time simulation of the folding process [12, 19, 30].

The main motivation of this PhD Thesis is to address the computational challenges faced by molecular dynamics simulations, particularly in protein folding. By investigating strategies that reduce execution time, enhance computational efficiency, and improve load balancing in HPC environments, we aim to advance the current state of protein folding simulations. These efforts will focus on achieving scalable performance while maintaining accuracy, enabling larger and more complex protein systems to be simulated within practical time frames.

1.3 Objectives

The main objective of this PhD Thesis is to investigate parallel strategies to reduce the execution time of molecular dynamics simulations of protein folding, while maintaining good accuracy.

Another objective of this work is to explore a flexible strategy for incorporating additional force calculations into parallel protein folding simulations. This includes the integration of complex interaction terms that impact molecular behavior during protein folding, such as presented in [31, 32]. By incorporating additional force computations and ensuring they can be efficiently parallelized, the proposed strategy enhances the scalability of simulations, enabling the study of larger systems while maintaining the desired accuracy.

1.4 Contributions

In order to achieve the objectives of this PhD Thesis, two strategies were developed and are listed below:

- A dynamic spatial decomposition strategy to handle non-static distribution of atoms and calculations among processors during the execution of a simulation routine using parallel protein folding simulation applications. In our strategy, the simulation box dynamically adapts to molecular structural changes during the simulation. In order to do so, a solution was created that modifies the workflow of a parallel protein folding simulation by incorporating three new steps: (i) saving partial trajectory data during the folding process, (ii) updating structural configuration data for executing new spatial decompositions during the simulation, and (iii) merging all partial results into a final trajectory output.

  A parallel execution framework for protein folding simulations of biomolecular systems, incorporating new interaction terms related to molecular stability and energy optimization. A new algorithm was devised to integrate calculations for various interaction potentials into an HPC molecular dynamics software. The NAMD parallel molecular dynamics software platform was chosen as the HPC tool for implementation because it is designed for large biomolecular systems using HPC parallel simulation, allows code modifications for research purposes, and has been recognized for its outstanding achievement in HPC. We opted to include atomic burial forces [31–33] into the simulation since very good results were obtained for globular proteins. Various functionalities were extended to include the necessary computing capabilities by (i) creating new compute objects for specific interactions and (ii) modifying key components to integrate these objects into the simulation engine. For energy optimization, modifications were primarily made to objects responsible for managing the global execution flow in the simulation platform.

1.5 Document Organization

This PhD Thesis is divided in three parts. The first part, Background/ Contextualization, provides first a foundation for understanding proteins, their structure, and the significance of protein folding. Then, we discuss parallel strategies for running molecular dynamics simulations, in particular protein folding, in HPC environments.

Chapter 2 explains the basics of proteins, covering the amino acids that form them and their structural hierarchy - primary, secondary, tertiary, and quaternary structures. The chapter concludes with a discussion of different types of proteins.

Chapter 3 presents an overview of the protein folding process and how it is viewed as an interdisciplinary problem, next it outlines two well-known experimental methods used for determining protein structures, and then it details computational methods for obtaining protein structures, how they are currently grouped by their main roles, ending with an overview of molecular dynamics in its role as an *ab initio* method for simulating the protein folding process.

In Chapter 4, we have a more detailed view about molecular dynamics simulation, starting by explaining its methodology and general algorithm, next presenting an overview about two typical models used for representing the simulated

system, called solvation models. Then, we present three well-known challenges that emerge when using molecular dynamics and, finally, we explain NAnoscale Molecular Dynamics (NAMD) [34] and Atomic Burial [31–33, 35], two different solutions used to deal with these challenges.

In Chapter 5, we present a review about parallel techniques for running molecular dynamics simulations on supercomputers, using different approaches to combine HPC architectures and specialized software for handling the high computational power requirements of molecular simulations. We start with an overview about HPC and a list of the fastest supercomputers in the world, followed by an explanation about HPC architectures and concluding with the most common HPC architectures currently, Massively Parallel Processing and Cluster Computing. Next, we discuss nine different techniques for running molecular simulations in HPC architectures and, then, we present a comparative analysis of these techniques and also, including, our own proposal strategy.

The second part of the PhD Thesis (Contributions) presents the two contributions developed throughout this work. Chapter 6 tackles the challenge of HPC molecular dynamics simulations with static domain decomposition and presents our Adaptive Patch Grid strategy. It describes the domain decomposition computation process, followed by an extensive report of experimental results, including tests with different patch grids (default, scaled, manual, automatic). The chapter concludes with a review of the contribution.

Chapter 7 focuses on the parallel execution of the atomic burial algorithm (MDBury) and its integration with NAMD, called N2HB. Our N2HB algorithm is explained, along with how atomic burial forces and hydrogen bond forces are computed. The chapter ends with experimental results showing the performance of NAMD with atomic burials and a contribution review.

In third and final part, Conclusion, Chapter 8 summarizes the findings of this research and discusses possible future work, outlining potential directions for further improving molecular simulations of protein folding. At the end of the document there are four Appendices: the Appendix A holds the first page of our published paper, Appendix B contains the basic NAMD configuration file used in our simulations, Appendix C contains the list of NAMD Components and files which were used in building the NAMD solution with MDBury and, finally, the Appendix D brings the Acknowledgment required by the group NAMD maintainer.

Part I

Background / Contextualization

Chapter 2

Proteins: An Overview

Proteins are at the center of action of biological processes, being responsible for almost all tasks of cellular life [1]. Among the many roles that proteins play are: protecting the body (antibodies), catalyzing chemical reactions and assisting in the formation of new molecules (enzymes), and transmitting signals to coordinate actions between different cells (hormones).

Much of the research in Biochemistry and Molecular Biology currently involves the definition of structures of individual proteins, given that there is a direct relationship between the specific function of the protein in the organism and the 3D structure that it adopts within the cellular environment [36].

In this chapter, some concepts used in the definition of protein structures are briefly reviewed. Section 2.1 covers the basic unit of a protein, the amino acid, and the chemical bonds commonly found in proteins. Section 2.2 discusses the different geometric configurations of protein structures and Section 2.3 presents the existing types of proteins.

2.1 Amino Acids

Amino acids serve as the building blocks of proteins, and they share a common structural framework. At the center of each amino acid is an α-carbon atom, which is covalently bonded to four distinct groups: a hydrogen atom, an amine group ($-NH_2$), a carboxyl group ($-COOH$), and a variable side chain denoted as R. In aqueous environments, the carboxyl group tends to donate a proton to the amine group, resulting in the molecular structure illustrated in Figure 2.1 [37, 38].

There are 20 different amino acids commonly found in proteins. Each amino acid possesses distinct properties - such as electric charge, polarity, and hydropathy index - outlined in Table 2.1.

Peptide bonds are formed through a condensation reaction between the amino group of one amino acid and the carboxyl group of another, with the release of one water molecule. This reaction links amino acids in a linear sequence, producing polypeptides. Figure 2.2 [42] illustrates this process, showing the formation of a peptide bond that joins two amino acids together into a dipeptide.

In this context, some physical-chemical interactions between molecules are commonly mentioned regarding the structure and function of proteins, namely: cova-

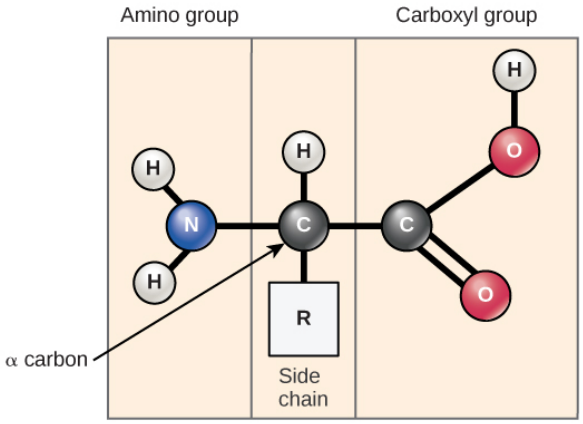


Figure 2.1: Basic structure of an amino acid. Image from [39].

Amino acid (aa)	Abbreviations		Properties		
	1 letter	3 letters	Hydropathy	Polarity	Charge
Alanine	A	Ala	hydrophobic	nonpolar	neutral
Cysteine	C	Cis, Cys	hydrophobic	nonpolar	neutral
Phenylalanine	F	Fen, Phe	hydrophobic	nonpolar	neutral
Glycine	G	Gli, Gly	neutral	nonpolar	neutral
Isoleucine	I	Ile	hydrophobic	nonpolar	neutral
Leucine	L	Leu	hydrophobic	nonpolar	neutral
Methionine	M	Met	hydrophobic	nonpolar	neutral
Proline	P	Pro	neutral	nonpolar	neutral
Tryptophan	W	Tri, Trp	hydrophobic	nonpolar	neutral
Valine	V	Val	hydrophobic	nonpolar	neutral
Asparagine	N	Asn	hydrophilic	polar	neutral
Glutamine	Q	Gln	hydrophilic	polar	neutral
Serine	S	Set	neutral	polar	neutral
Threonine	T	The, Thr	neutral	polar	neutral
Tyrosine	Y	Tir, Tyr	neutral	polar	neutral
Aspartate	D	Asp	hydrophilic	polar acid	negative
Glutamate	E	Glu	hydrophilic	polar acid	negative
Arginine	R	Arg	hydrophilic	polar basic	positive
Histidine	H	His	neutral	polar basic	positive (10%) neutral (90%)
Lysine	K	Lis, Lys	hydrophilic	polar basic	positive

Table 2.1: The 20 amino acids found in Nature [40, 41].

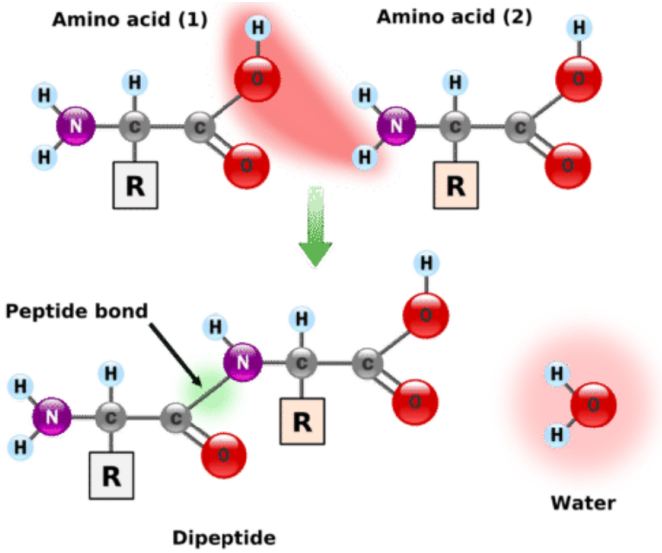


Figure 2.2: Formation of a peptide bond between two amino acids. Image from [42].

lent bonds, hydrogen bonds, *van der* Waals forces and the hydrophobic effect (or hydrophobic interactions [43]).

The most common interactions between proteins are covalent bonds. This is a type of chemical bond that is strong and difficult to break, as it occurs between atoms that are very close and that share electrons with each other or between other covalent bonds [5].

Hydrogen Bonds (HB) are also strong intermolecular interactions. In the context of protein structure, these bonds usually happen when the hydrogen of one molecule (positive pole) bonds with an oxygen or nitrogen atom of another molecule (highly negative pole). As there is no sharing of electrons, despite being a strong bond, HB are weaker than covalent bonds [1, 37].

Also known as London forces, *van der* Waals forces intermolecular interactions that can be easily broken, because they are weaker than HB. The *van der* Waals forces correspond to the result of all forces (attraction or repulsion) between molecules and atoms, when nearby particles are polarized [1, 37].

The hydrophobic effect is related to one of the properties of amino acid molecules, the hydropathy index [40], or hydrophobicity, i.e. the degree of affinity or interaction of the amino acid with water. The hydropathy index reveals whether an amino acid's interaction level is neutral, hydrophilic (high affinity) or hydrophobic (low affinity), as shown in Table 2.1. Considering that the cellular environment is basically aqueous, the hydrophobic effect plays an important role in the structural formation of proteins [43]. For example, a hydrophobic nonpolar solution when added to water will not dissolve, as it happens in the mixture of water and oil. This happens because of the hydrophobic effect as a hydrophobic non-polar amino acid molecule (H) tends to, predominantly, aggregate and concentrate when in contact with water.

Thus, examining the functional structure of a protein, hydrophobic amino acids (H) tend to concentrate in the inner part of the protein (core), while polar amino

acids (P) accumulate on the surface of the structure, establishing greater contact with the cell's aqueous environment.

2.2 Protein Structures

Every protein in an organism begins its existence in a ribosome, which synthesizes it, connecting the corresponding amino acids in an initial structure forming a polypeptide chain [44]. As soon as the synthesis ends, the protein undergoes a series of transformations in its structure until it finds a stable configuration that will determine its biological function [1, 37].

Considering the complexity of these transformations, proteins are organized and studied in terms of their structures [1], traditionally in a four-level hierarchy [1, 37], called: primary, secondary, tertiary and quaternary structures (Figure 2.3). The sections that follow present each of these levels.

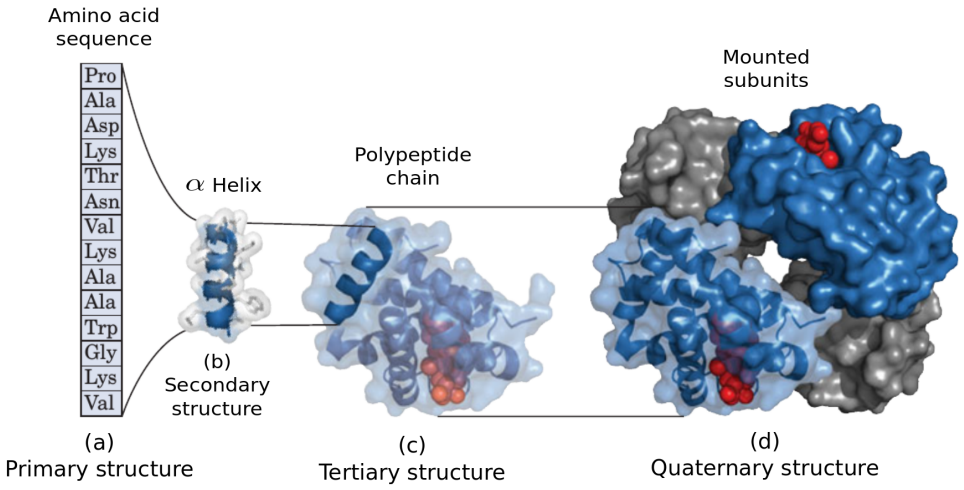


Figure 2.3: Hierarchy of protein structures in four levels: (a) primary structure; (b) secondary structure; (c) tertiary structure; and (d) quaternary structure. Image adapted from [44, 45].

2.2.1 Primary Structure

The first level of protein composition is called the primary structure, which has a 1D geometric configuration in the form of an amino acid sequence [37, 44, 46], as illustrated in Figure 2.3a.

The primary structure is determined by the gene that encodes the protein and its linear sequence of amino acids is unique [44].

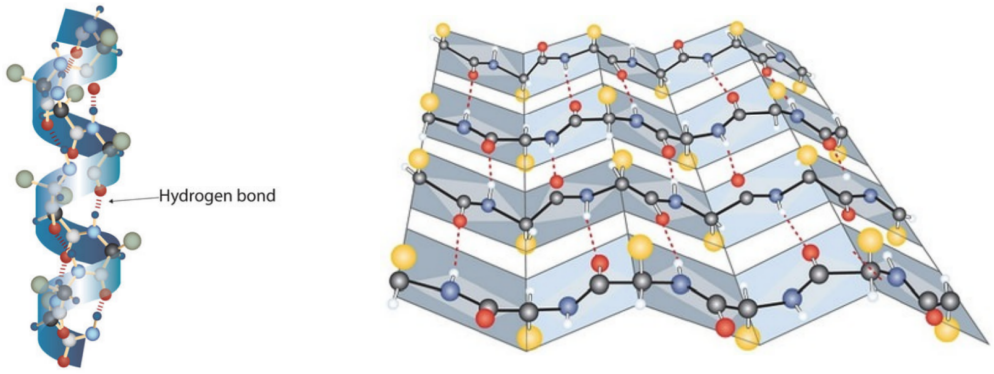
Figure 2.4 shows an extract of the primary structure of the human protein beta globin from the Protein Data Bank (Protein Data Bank (PDB)), named NP_000539 [47]. In this figure, it can be seen that the chain of amino acids is stored in the protein data format as an ordered string of characters that corresponds to known amino acid abbreviations, as per Table 2.1.

```
> MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQRFFESFGDLSTPDAVMGNPKVKAHGKKVLG
AFSDGLAHLDNLKGTFATLSELHCDKLHVDPENFRLLGNVLVCVLAHHFGKEFTPPVQAAYQKVVAGVAN
ALAHKYH
```

Figure 2.4: Primary structure of a human beta globin protein (NP_000539) [47].

2.2.2 Secondary Structure

The secondary structure of a protein is identified by the patterns found in the arrangements adopted by the chain of amino acids that compose it. The most common ones are known as α helix and β sheets, as shown in Figure 2.5.



(a) α helix structure

(b) β sheet structure

Figure 2.5: Illustrations of secondary structures: (a) α helix structure on the left; (b) β sheet structure on the right. Images from [48].

The α helices are spiraling strands or ribbons that can be composed of stretches of four to forty amino acid residues (for example, myoglobin) [47]. The side-chain (R) elements of amino acids extend to the outside of the structure. Sheets β usually contain adjacent β strands of five to fifteen residues generally, and their orientations can be parallel (chains in the same direction) or antiparallel (chains in opposite directions) [47].

2.2.3 Tertiary Structure

The tertiary structure of a protein refers to its 3D geometric configuration, shown on Figure 2.6. The tertiary structure is directly related to the biological function that the protein performs in an organism [44].

The process by which a protein undergoes a series of structural transformations, moving from its 1D primary structure to a stable 3D tertiary structure in order to perform its biological function, is called Protein Folding (PF).

During this process, each possible 3D arrangement that the protein adopts, until it reaches its 3D tertiary structure, is also called a conformation [1]; and the final stable conformation is also called native structure.

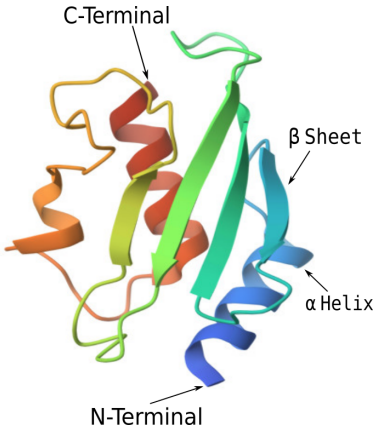


Figure 2.6: Tertiary structure with α helices and β sheets. Structure based on the protein 2FO3, adapted from the RCSB PDB [49].

At this point, a distinction must be made between the formats of secondary and tertiary structures. While secondary structure refers to the spatial arrangement of amino acid residues that are adjacent in a polypeptide segment, such as the α helices and β sheets, tertiary structure refers to the overall 3D configuration of all atoms in a protein [44]. This includes both elements that are close and distant in the amino acid sequence, or even located in different types of secondary structures, but that interact with each other, when the protein assumes its native 3D conformation [44].

Proteins are capable of interacting with their cellular environment in a variety of ways, assuming a different native conformation for each function they perform. Anfinsen [36] established that all the information necessary for a protein to find its native 3D structure is contained in its primary 1D structure, that is, its amino acid sequence [2].

Under physiological conditions, from a thermodynamic point of view, the tertiary native structure has a fragile stability [1], and, during the process of Protein Folding (PF), the free energy required to sustain this formation is influenced by several physical-chemical interactions between the amino acids that make up the protein [50], for example: (a) forces resulting from HB contribute to the the formation of α helices and β sheets of minor structures [2]; (b) the *van der* Waals forces that are present in interactions between atoms close to each other while the structure assumes a compact configuration, typical of globular proteins [2]; (c) the preferred angles that orient bonds between neighboring atoms in the backbone of the protein, usually in α helices and β sheets [2]; (d) the electrostatic interactions of attraction or repulsion that occur due to negative or positive charges of the amino acids involved (see charges in Table 2.1); (e) the disulfate bonds, which prevent the 3D structure from breaking down outside the cellular environment, are commonly found in proteins secreted by cells (e.g. immunoglobins) [50]; (f) the chain entropy, which drastically decreases as the chain of amino acids of the primary structure folds and coils quickly, and finds an energetically stable 3D configuration, without going through all possible conformations [2]; and, finally, (g) the hydrophobic interactions and their hydrophobic effect (Section 2.1), which make the non-polar

components minimize their contact with the aqueous environment, assuming configurations such as the hydrophobic nuclei of globular proteins [2, 37].

Figure 2.7 shows a schematic example of some physicochemical factors that contribute to the stability of the 3D configuration of the tertiary structure, such as those resulting from hydrophobic interactions, electrostatic attractions, side chain HB and elements of secondary structures (α helices and β sheets) [37].

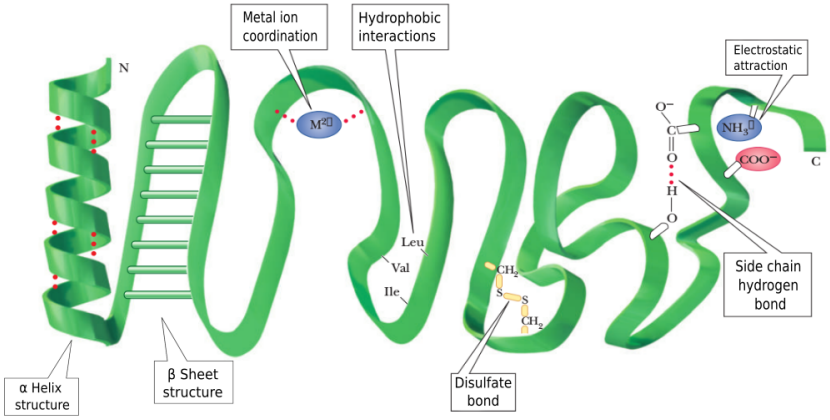


Figure 2.7: Examples of forces that stabilize the tertiary structure of proteins. Image adapted from [37].

2.2.4 Quaternary Structure

The last level of the hierarchy of protein structures is the quaternary structure. At this level all proteins are composed of two or more chains of amino acids that assemble into a unique 3D configuration [37, 44].

In a quaternary structure, each chain of amino acids is called a subunit. Subunits may be different or identical, and may vary in number from one protein to another. [37].

The connection of the subunits in the quaternary structure is maintained via non-covalent interactions (Section 2.1), such as HB, electrostatic attractions and hydrophobic interactions [37, 50].

Hemoglobin is an example of a protein with a quaternary structure (Figure 2.8), composed of two α globin chains, two β globin chains and four non-protein groups (*heme* groups) attached. In its stable configuration, each subunit of hemoglobin works cooperatively, so that during oxygen transport, each subunit attached to an oxygen increases the affinity of another subunit for oxygen. [50].

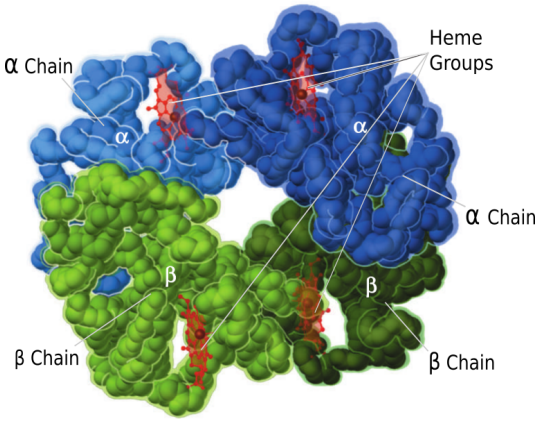


Figure 2.8: Quaternary structure of hemoglobin. Image adapted from [37].

2.3 Types of Proteins

Proteins can be grouped into three major classes by observing the 3D geometric configuration of their native structures, namely: globular, fibrous (or structural) and membrane [1, 37]. Figure 2.9 contains an illustration with an example of each of these types of proteins.

Globular proteins are formed by polypeptide chains that fold back on themselves and assume a spherical shape. This characteristic 3D shape is generally attributed to their interaction with the aqueous environment, as these proteins are water-soluble [1]. Globular proteins do not lie dormant in the body and generally play a role in dynamic biological functions, such as hormone and transport proteins, antibodies and most enzymes [37].

An example of globular protein structure can be seen in Figure 2.9a, which shows a representation of myoglobin. It is an oxygen transporting and storing protein commonly found in abundance in marine animals such as seals and whales.

(a) Globular (Myoglobin)

(b) Fibrous (Collagen)

(c) Membrane (Bacteriorhodopsin)

Figure 2.9: Illustration of the types of proteins: (a) globular, (b) fibrous, and (c) membrane. Image adapted from [37].

Fibrous proteins are formed by parallel polypeptide chains arranged in long fibers or sheets. These proteins are resistant and insoluble in water [1], due to their high composition of hydrophobic amino acids (Section 2.1). Fibrous proteins generally act in the structural maintenance of the organism, being able to confer

both elasticity and resistance. They are composed of units that are repeated and associated to allow the formation of large structures.

Examples of fibrous proteins are: collagen, which acts on bones and tendons (Figure 2.9b); keratin, present in skin, nails and hair; and elastin, an element of elastic connective tissue.

Membrane proteins are so named because performing their functions in an organism involves interacting with cell membranes, influencing communication between the inside of the cell and the outside environment. These proteins are important because they act as catalysts and regulators in many vital processes, such as photosynthesis and cell-to-cell communication.

Figure 2.9c shows an example of a membrane protein, called *bacteriorhodopsin*, usually found in membranes of halobacteria [37]. This protein acts as a proton pump, capturing light energy and using it to move protons across the cell membrane [1].

In addition to these well-known classes, a fourth type of protein has gained increasing attention in recent decades: the Intrinsically Disordered Protein (IDP), or proteins that contain Intrinsically Disordered Regions (IDRs). Unlike structured proteins that adopt a stable 3D conformation, IDPs do not fold into a single well-defined structure under physiological conditions [51]. Instead, they exist as dynamic ensembles of conformations, often shifting in response to environmental changes or binding partners.

Despite their lack of a stable fold, IDPs play crucial roles in cellular processes such as transcriptional regulation, signal transduction, and molecular recognition [52]. Their functional versatility is associated with structural plasticity, which allows them to interact with multiple targets through conformational adaptation. IDPs are typically rich in polar and charged amino acids and depleted in hydrophobic residues, contributing to their solubility and extended conformations. This makes them resistant to the hydrophobic collapse that drives folding in most globular proteins. Although this intrinsic disorder complicates structural characterization by traditional experimental methods, computational tools - especially disorder predictors and molecular dynamics simulations - have become essential in their study [53].

Furthermore, IDPs are increasingly recognized for their involvement in human diseases. Their flexible structures and multifunctionality make them key players in neurodegenerative disorders such as Alzheimer's and Parkinson's, as well as in cancer-related pathways [54]. Recent reviews reinforce the growing importance of IDPs in modern molecular biology, highlighting their expanding roles in signaling networks, molecular recognition, and disease mechanisms [55, 56]. Including IDPs in the classification of protein types provides a more complete view of the diversity of structural behavior observed in nature.

Although there are different classes of proteins, each with distinct structural and functional characteristics, this work will focus exclusively on globular proteins. This choice is justified by their fundamental role in various dynamic biological functions, given their fundamental role in dynamic biological functions and the relevance of their 3D structure to the computational challenges addressed in this Thesis.

Chapter 3

Protein Folding

3.1 Overview

As presented in Section 2.2.3, the function of a protein in an organism is directly related to its 3D shape, called tertiary structure or native structure.

Protein Folding (PF) is the spontaneous process by which the protein abandons its primary 1D structure and performs a series of geometric transformations until it assumes its stable 3D structure [2].

The study of protein folding as an interdisciplinary problem began, according to [2], with the publication of the results presented in [57]. Using the globular protein myoglobin, the first protein structure obtained experimentally was determined and, different from what was expected, it had a complex, asymmetrical and irregular 3D arrangement [57]. This result drew the attention of the scientific community to understand and explain protein folding.

In the last decades, the protein folding problem has encompassed three main issues [2]: (a) what is the physical code for folding? (b) what is the folding mechanism? (c) is it possible to develop a computational algorithm to predict the native structure of a protein from its primary structure?

The first question basically aims to understand and determine the native 3D structure from the physicochemical properties encoded in the primary structure. The second intends to understand the folding process and determine how a protein is capable of folding itself and finding its most stable 3D geometric configuration in such a short time. Considering that the chain of amino acids has, at each folding step, a large number of possible geometric configurations to adopt in the 3D space, the researchers try to understand how the protein "chooses" which one is the next conformation.

Figure 3.1 shows that there are methods for protein folding determination with the use of experimental techniques, however due to their high cost and time needed to perform them, the third question refers to the construction of computational methods capable of aiding the experimental methods by predicting the native structure of a protein using the information contained in its primary structure. The context of this Thesis is inserted in the third question, due to its focus on computational methods for protein folding.

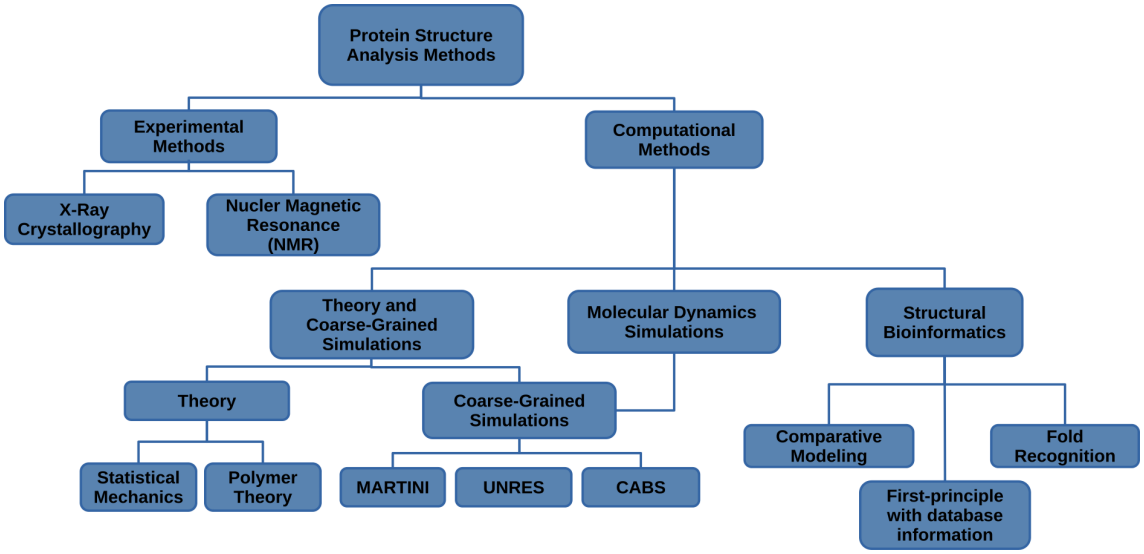


Figure 3.1: Hierarchical classification of methods used in protein structure analysis. At the top level, methods are divided into experimental techniques and computational approaches. Experimental methods include X-ray crystallography and Nuclear Magnetic Resonance (NMR) [37], which directly determine atomic coordinates from physical samples. Computational methods are further subdivided into three categories [3, 58]: (i) Theoretical models and Coarse-Grained simulations, which use simplified representations or statistical physics; (ii) Structural Bioinformatics, which includes techniques such as comparative modeling, fold recognition, and first-principles approaches with information from databases; and (iii) Molecular Dynamics simulations, which model atomic interactions over time using force fields.

In the classification presented in [5, 59], the computational methods used for predicting protein structures, also known as protein structure prediction, were organized into four groups: comparative modeling, fold recognition, first-principle methods using database information, and first-principle methods without such information. Although structure prediction is distinct from simulating the physical folding process, theory and computation have evolved together. As discussed in [3, 58], some of these methods also contribute to the investigation of how folding occurs - by revealing possible intermediate states and energy landscapes. This enables researchers to study the "stories" behind protein folding and gain insights into the underlying mechanisms.

1. Theory and Coarse-Grained simulations: the methods in this group focus on telling the protein folding macro-story; it uses statistical mechanics and polymer theories to answer general questions and principles through identifying differences and common features amidst proteins [58];

2. Structural Bioinformatics (SBI) [3]: also called Database Models by [58], this group of methods focuses on inferring the protein folding story by taking snapshots of potential native structures and guessing the process through the usage of large databases with information about known protein structures (e.g.

PDB), and inference approaches, that consider, for example, sequence homologies in order to predict unknown structures. One successful example of this approach is the AlphaFold deep learning algorithm [30, 60], which leverages extensive protein sequence and structure databases throughout its training and prediction processes, enabling it to predict a protein's static 3D structure and provide a likely model of its folded form; however, AlphaFold does not generate protein trajectories or simulate protein dynamics [30].

3. Molecular Dynamics (MD) simulations: is a technique of Computational Molecular Physics (CMP) applied to protein folding [3, 28] aiming to unveil microstories about the folding process that satisfy principles of chemistry and thermodynamics. Molecular dynamics simulations capture the movement of atoms in a protein under physical forces, simulating how proteins change shape over time in response to thermal motion, solvent interactions, and other factors. For that, it typically uses force fields of atomic interactions and a standard algorithm to sample the conformational space [22, 23, 27, 61]. Current advances to compensate MD's well known demand for high computing power and improve its accuracy include new techniques that apply experimental data in molecular dynamics execution flow while still guaranteeing its proper physics calculations, like ML×MELD×MD [28], a molecular dynamics method that integrates residue contacts predicted by Machine Learning (ML) servers into the Modeling Employing Limited Data (MELD) tool, a Bayesian MD-accelerator designed to maintain detailed-balance statistics [3].

Although the techniques that emerged from these three roles provided different methodologies for aiding experimental methods in solving the protein folding problem, a physical-based analysis is still considered preferable for investigating the protein folding process [3]; therefore, this PhD Thesis focuses on the third role, more specifically, in the molecular dynamics simulation method.

The following sections bring two well-known experimental methods and their limitations, as well as the computational methods most commonly used in these cases.

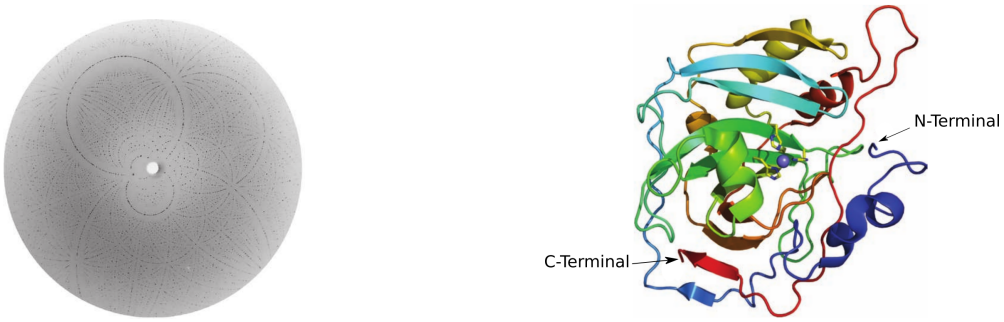
3.2 Experimental methods for determining protein structures

Once the primary amino acid sequence of a protein is obtained, one can begin the difficult task of determining its unique native 3D structure. Figure 3.1 shows two well-known experimental techniques for determining tertiary protein structures: X-ray Crystallography and Nuclear Magnetic Resonance (NMR).

3.2.1 X-Ray Crystallography

In X-ray crystallography, pure crystals are used to diffract X-ray beams which produces scattered patterns on a specific plate or counter as shown in Figure 3.2. The 3D structure is, then, determined by measuring the angles and intensities.

Crystals are grown under controlled conditions containing, in each crystal, protein molecules with the same orientation and the same 3D configuration [37]. Such crystals can only be formed if the proteins are of high purity and, furthermore, a structure is obtained only if the protein can be crystallized [37].



(a) X-ray diffraction. (b) 3D structure.

Figure 3.2: (a) X-ray diffraction photograph of the enzyme *glutathione synthetase*, reproduced from [37]; (b) 3D structure of the enzyme *anhydrase*, determined by X-ray crystallography, adapted from [1].

Using this technique, the pattern is created when the electrons in each atom of the molecule scatter X-rays. The number of electrons in the atom determines the amount of X-ray scattering: heavier atoms scatter more effectively than lighter ones.

X-rays scattered from individual atoms can reinforce or cancel each other out, giving rise to the characteristic pattern for each type of molecule. A series of diffraction patterns from different angles contains the information needed to determine the tertiary structure.

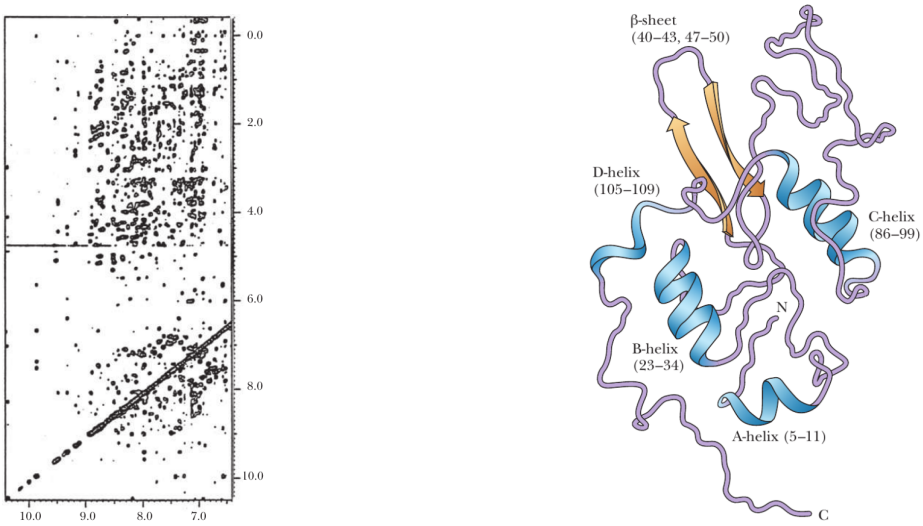
Information is extracted from the diffraction patterns through a mathematical analysis by Fourier series [62]. Thousands of these calculations are needed to determine the structure of a protein, and even when performed on a powerful computer, the process is quite time consuming [1, 37].

3.2.2 Nuclear Magnetic Resonance

Another experimental technique used to determine protein structures is the Nuclear Magnetic Resonance (NMR) [1, 37]. Among the NMR methods, the most widely used is NMR spectroscopy, also known as 2D NMR [37]. This technique complements X-ray diffraction results through computational analysis of large collections of data points, as shown in Figure 3.3a.

Similar to the X-ray diffraction technique, 2D NMR requires small amounts of protein (milligrams), uses Fourier series to analyze the results, and is a very time consuming and computationally powerful process [37].

One difference between 2D NMR and X-ray diffraction is in the sample: 2D NMR uses protein samples in aqueous solution instead of crystals. The environment used is one of the main advantages of the 2D NMR method, as it is the closer to proteins in cells [37].



(a) NMR data.

(b) 3D structure.

Figure 3.3: 2D NMR data for determining the 3D structure of α-lactalbumin. Images adapted from [37].

Ultimately, 2D NMR relies on distances between hydrogen atoms and the data it generates are independent of and commonly complement those obtained by X-ray crystallography [37].

In general, these are considered the best-known experimental methods used for determining 3D protein structures. Nevertheless, they are not enough and there is still a great number of protein structures that can not be determined by only these experimental methods alone [63], mainly due to their high cost in time and human resources, combined with the need of specific materials and equipments.

As more protein sequences are being obtained in the last decades at a close to exponential rate, there is a need to accelerate the prediction of protein structures using the help of computational methods [2]. This topic will be discussed in Section 3.3.

3.3 Computational methods for predicting protein structures

In order answer the third question of protein folding problem, the development of computational algorithms to predict tertiary structures of proteins has been one of the great challenges of Computational Biology in the last 50 years [2]. Currently, the knowledge acquired from polymer theory and computation advances in the last decades shows three main roles emerging in this scope (Figure 3.1): Theory and Coarce-Grained (CG) simulations, Structural Bioinformatics (SBI) and Molecular Dynamics (MD) simulations.

The following sections will briefly review each of the roles.

3.3.1 Theory and Coarse-Grained Simulations

Figure 3.1 shows that this type of simulation comprises of techniques that take advantage of Statistical Mechanics and Polymer Theory in conjunction with Coarce-Grained (CG) simulations aiming to address general questions, mainly concerning the differences and similar traits among proteins, based on global principles, for obtaining a macro comprehension of the whole folding process [58].

Polymer theory provides concepts and tools for describing the macro states of the folding process based on sets of different conformational ensembles of large numbers of microscopic states. CG models are applicable for studying large scale biological systems focusing on their mesoscopic or macroscopic properties [64, 65].

According to [58], the methods using this approach present four key points, as follows:

1. they define specific ensemble averages that can be used as descriptors for the sizes of conformational ensembles;

2. they use polymer theory to explain the role of the solvent in the geometric conformations present in obtained ensembles; for instance: proteins generally unfold in "good" solvents or denaturant agents such as heavy metal salts and alcohol, and potentially fold in "poor" solvents, like water; however, for some folding processes, water can behave both as poor or good solvent during the unfolded stages of the protein folding producing "unexpected" conformations;

3. they apply the acquired knowledge about the role of solvents to understand features of conformational ensembles, such as the impact of hydrophobicity and charge in the compactness of specific protein structures, for example: globular protein conformational ensembles;

4. since different experiments might give different weightings to the components of the ensembles, these may seem occasionally inconstant; thus, in this case, theory can be applied for calculating these different features and resolving apparent contrasts among distinct experiments, for example the tool presented in [66] created for scattering analysis of hydrophobic protein ensembles.

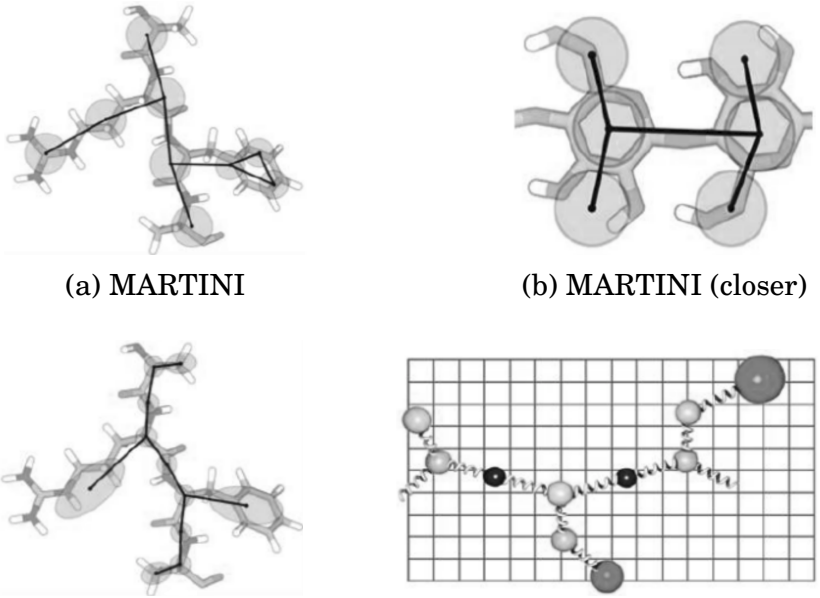
CG models are a reduced representation of all-atom models, however maintaining the essential molecular aspects for the original system. This representation of atoms allows simulating large-scale biological systems, with faster sampling due to reduced degrees of freedom and achieving longer time scales [64].

In general, a CG simulation model comprehends: (a) *pseudoatoms* sites which are defined for representing collections of atoms; (b) an energy function determined for simulating the interaction between *pseudoatoms*, typically reproducing the thermodynamic properties of the system; and (c) defining dynamical equations for describing and studying the evolution of the CG system over time.

These dynamical equations allow CG models not only to describe conformational ensembles, but also to be employed in time-resolved simulations. Although CG models are commonly associated with structural or thermodynamic studies,

they are also widely used in the context of molecular dynamics simulations, forming the basis of *Coarse-Grained Molecular Dynamics* (CGMD). This approach combines the reduced resolution of CG representations with the time-resolved nature of MD, enabling simulations of large-scale systems and longer time scales with reduced computational cost. Notably, CGMD approaches such as those based on MARTINI models are widely adopted for simulating membrane proteins and complex biological assemblies [64]. Recent studies have also demonstrated the successful application of CGMD to simulate the behavior of antibodies and their interactions in solution [65]. A comprehensive overview of CG models and their integration into molecular dynamics frameworks is provided in [67]. To reflect this conceptual overlap, Figure 3.1 includes a connecting link between the CG simulation branch and the molecular dynamics category.

This connection is also supported by the practical relevance and diversity of CG models used in such simulations. Figure 3.4 illustrates three widely adopted CG models: MARTINI, UNRES, and CABS [64].



(a) MARTINI (b) MARTINI (closer)

(c) UNRES (d) CABS

Figure 3.4: Illustrations of CG models: (a) and (b) MARTINI force field, with granularity of 5 (five) beads per group; (c) UNRES model with CG beads for peptides; and (d) CABS model with its beads upon the C-Alpha, Beta and Side chain. Image adapted from [64].

The MARTINI model was initially proposed for studying lipids and then extended for protein systems. Nowadays, is typically used as the CG model for studying membrane proteins [64]. It follows a mapping of one-to-four heavy atoms and a hydrogen associated to which corresponds to the one interaction site, as illustrated in Figure 3.4a and Figure 3.4b. The UNited RESidue (UNRES) is a physics-based CG model with a moderate resolution and a highly reduced depiction of a protein since each residue is represented by two sites of interactions: one for the united peptide group and one for the united side chain (Figure 3.4c). Finally, the C-Alpha,

Beta and Side Chain (CABS) CG model is an average resolution knowledge-based with reduced representation model for proteins. The amino acid residue is represented by four united groups, namely, the C-Alpha, the Beta, the center of mass of the Side chain and the center of the peptide bond [64] (Figure 3.4d).

For more details about CG models and their applications refer to [64, 67].

3.3.2 Structural Bioinformatics (SBI)

The techniques involved in the role of SBI take advantage of the increasing growth of databases with protein structures information in order to predict unknown structures using sequence homologies to known structures. Figure 3.1 shows three common groups that depend on these Database models in their techniques: Comparative Modeling, Fold Recognition and Threading, and First-principle methods with database information.

Comparative Modeling

Comparative modeling, or homology modeling, exploits the fact that evolutionarily related proteins with similar sequences have similar structures [68, 69]. Such similarity is measured by the percentage of identical residues at each position based on optimal structural overlap.

Structure similarity is very high in the so-called core regions, which are typically made up of a set of secondary structure elements, such as α helix and β leaves. As seen in Section 2.2.2, these elements are connected through loops that tend to vary even in pairs of homologous structures having a high degree of similarity between the sequences.

Figure 3.5 shows a generic representation of a Comparative Modeling workflow.

The process of building a comparative model typically proceeds as follows [68, 69]:

1. A sequence alignment is performed between the amino acid sequence of the primary structure to be modeled (target sequence) and the amino acid sequence of a protein whose structure is determined by experimental methods (parent sequence);

2. From this alignment, an initial model, called a *template*, is constructed by copying the coordinates of the side and main chains of the structure of the parent sequence based on similar residues of the alignment;

3. The side chain is constructed for both the residues of the target sequences that do not match an alignment identity for residues where the 3D configuration of the side chain of the target sequence is expected to be different from the configuration in the parent structure;

4. Finally, the main chain is built for cases of insertions and regions close to deletions and other areas that present potential variations of the main chain.

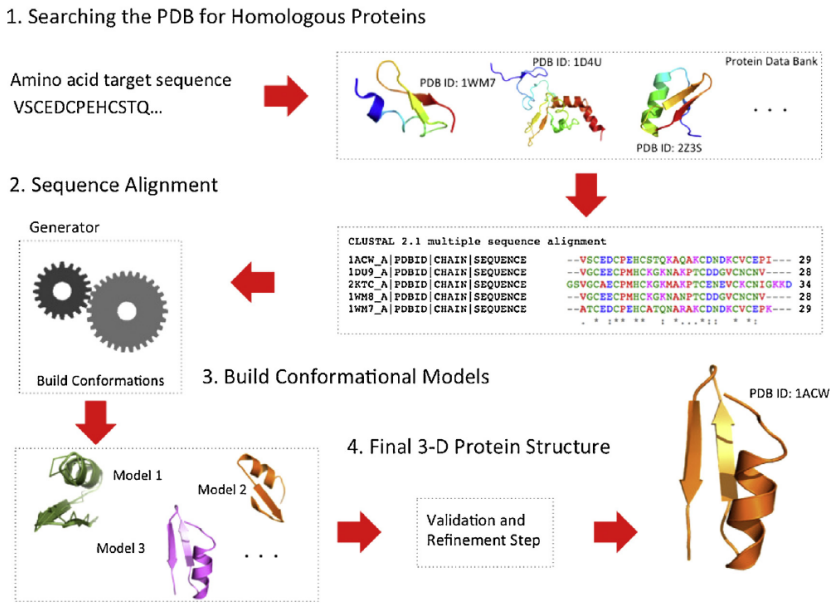


Figure 3.5: Comparative Modeling: representation of the generic workflow. Image from [5].

Comparative modeling typically uses primary sequence pairwise comparison methods to identify local alignments, as similarities between amino acid sequences usually occur between segments of the two sequences [68]. Another technique used is sequence profiling comparison, which considers the trends of each of the 20 types of amino acids for each position in a multiple sequence alignment [69].

Fold Recognition

Fold Recognition methods, or *Threading* methods, use a database of known 3D structures to identify amino acid sequences that do not yet have their folded structures defined [70–73]. The general goal is to fit a protein sequence correctly against a structural model, a *template* [5].

This is achieved with the aid of a scoring function, usually derived from a database of known structures, which evaluates whether a sequence, placed in a structural position of 3D *template*, corresponds to a given fold.

This way, these methods compare a target sequence against a library of structural *templates*, producing a list of scores. These scores are then ranked and the best-scoring fold is assumed to be the one adopted by the target sequence.

Figure 3.6 shows the generic workflow of these methods for predicting 3D structures:

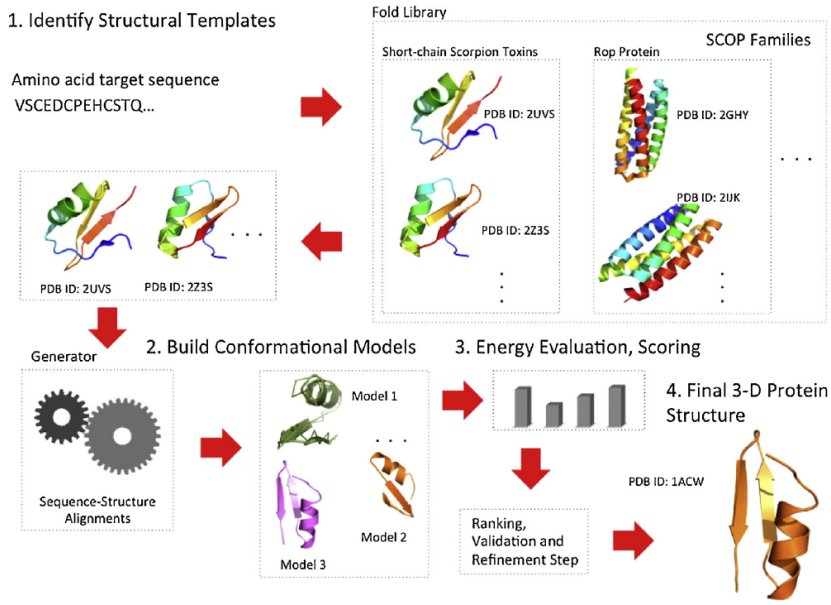


Figure 3.6: Fold Recognition: representation of the generic workflow. Image from [5].

The typical fold recognition approach for predicting 3D structures involves [5]:

1. A library of potential 3D *templates* is constructed using known native protein structures from the PDB, reducing their 3D coordinates to abstract representations;

2. A scoring system is used to evaluate the placement of any candidate sequence into the fold. In general, the scoring functions used are a list of statistical references of each amino acid residue to each structural or fold environment. Additionally, it calculates the potential energy of the structures and the models are scored, the structures are ranked and validated;

3. A strategy (algorithm) is used to search over the space of possible replacements to identify the optimal sequence-structure substitute; i.e, find the global best score and the optimal fitting/threading: two examples of approaches to this sequence-structure replacement are: (1) 3-D profile methods and (2) contact potentials [5].

To test a sequence against a library of folds, the chosen methods usually take advantage of sophisticated sequence alignment algorithms, such as hidden Markov models (*Hidden Markov Models*), to computationally elaborate techniques, such as dynamic double programming, dynamic programming with the so-called *frozen approximation* and Gibbs sampling using the *threading* cores [70–73].

First-principle methods with database information

These methods compare fragments of a target protein (i.e, amino acid subsequences) with fragments of other known structures available in databases (PDB).

Then, a structure is assembled from the identified fragments, using optimization algorithms and [59] scoring functions. These methods are called first-principle due to the similarity that exists between how their algorithms and functions are used and how free energy optimization algorithms and energy functions are used in first principle methods based exclusively on physical forces [59] (see Section 3.3.3).

Figure 3.7 shows a generic representation of this technique.

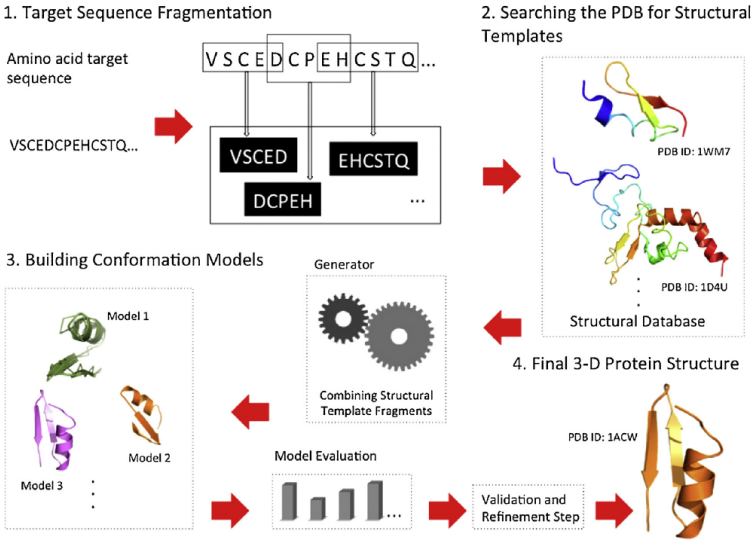


Figure 3.7: Fragment-based First-Principle methods: representation of the generic workflow. Image from [5].

According to [5], these methods are composed of five basic steps:

1. The target sequence is divided into fragments;
2. A search is performed for similar sequences from each fragment in a database of known structures;
3. The fragments are classified using a scoring system;
4. Using a combination method, a 3D configuration is constructed using the fragment *template* applying a combination technique;
5. The obtained 3D configuration is refined.

Fragment assembly methods are generally based on the idea that local interactions influence and direct, but do not uniquely define the local structure of the protein [59]. In other words, if there are fragments of proteins that fold into similar local structures, this information can be used to build 3D models of protein structures [5].

These assembly methods approximate this trend in structural formation by averaging between observed fragment geometries in known protein structures. Only when an appropriate set of observed fragments is identified, compact structures can be assembled by randomly combining fragments using a combination method, such as *simulated annealing* [74, 75].

The classification and suitability of a conformation is evaluated using a scoring function derived from statistical data of known protein structures [59].

These methods have some advantages, such as the ability to predict new folds, which is not the case with the comparative modeling methods [5], and the reduction of the conformational search space when compared to methods with no database information (Section 3.3.3).

However, the main limitations of these methods are the challenges of reducing the potential energy in regions where fragments are identified and dealing with the large conformational search spaces created by different combinations of these fragments [5].

In this case, the usage of Artificial Intelligence (AI) as another auxiliary method has also advanced in recent years, such as the agent-based parallel deep learning framework used to accelerate protein folding simulations presented in [13] and, most recently, the AlphaFold deep learning algorithm presented in [30, 60]. This method comprises of a deep-learning strategy for folding prediction divided in four stages as shown in Figure 3.8:

1. Homology-based features are derived from the protein's primary structure (from the PDB) and from Multiple Sequence Alignment (MSA);
2. A Neural network is trained on known structures from PDB in order to predict the distances between β-Carbon atoms of pairs of residues and create a set of distance distribution predictions based on probability distribution of pair of atoms in regions of 64 x 64 atoms;
3. A Potential function constructed using torsion, distance distributions and *van der* Waals terms;
4. A Gradient descent is used to find the specific sets of torsion angles that can be used to minimize the potential function throughout the process [30, 60].

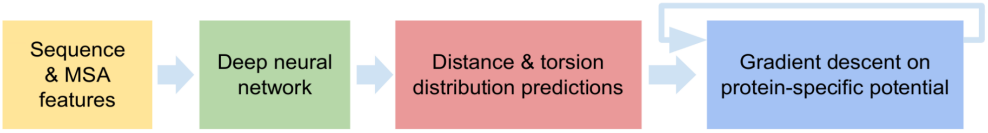


Figure 3.8: AlphaFold structure prediction stages: Sequence and MSA features, Deep neural network, Distance and torsion distribution predictions; and Gradient descent on protein-specific potential. Image from [60].

In the 14th Critical Assessment of Structure Prediction (CASP14), AlphaFold predicted the 3D structures of a group of proteins with a high accuracy compared only to models derived from experimental data. No longer after that, it was able to predict the structures of almost every protein in the human proteome [76]. However, AlphaFold has its limitations, as presented in [77]: (a) *low accuracy predictions* of intrinsically disordered proteins/regions and loops; (b) it predicts only a single conformer, failing to identify *apo* and *holo* forms; (c) it is not able to predict defects in the protein folding, caused by point mutations; and (d) it cannot predict new structures because its algorithm requires data of known structures.

3.3.3 Molecular Dynamics and Protein folding

Molecular dynamics simulation applied to protein folding is a set of methods based on Computational Molecular Physics (CMP) that use force fields of atom-atom interactions and explore the 3D geometric space in order to develop biological micro-stories that satisfy principles of chemistry and thermodynamics.

Unlike the techniques presented in Sections 3.3.1 and 3.3.2, Molecular Dynamics (MD) simulation in essence is a first-principle method without database information, also called *ab initio*, and, typically, do not use any data about structures of previously resolved proteins. Traditionally, these methods have a high computational cost and were limited to study only simple actions of small proteins.

However, recent advances [3, 6] were made on more accurate force fields and solvent models for capturing the molecular interactions, and on new faster methods for conformational searching and sampling [21, 78]. With this, MD is now able to model, with good accuracy, protein actions on time scales longer than microseconds, and sometimes milliseconds, with a great impact on protein folding [3]. Additionally, new methods that make use of (external) experimental data of determined structures are now used in order to accelerate the process, while preserving proper physics calculations [3, 6, 28].

Concerning the protein folding problem and its third question (Section 3.1), the biggest challenge regarding the usage of MD techniques for predicting 3D native structures can be organized into two steps:

1. Building a scoring function that can distinguish correct (or similar) native structures from incorrect (non-native) structures;

2. Choosing a search method for exploring the 3D geometric space looking for possible configurations.

In MD methods, components 1 and 2 are coupled in a way that the search method guides, and is guided by, the scoring function to find similar native structures. Typically, a potential energy function is chosen as a scoring function. Currently, there is no generic and reliable scoring function that can guide a search to always find a native structure. And, likewise, there is no search method that can traverse the space of conformations in order to guarantee a significant fraction of 3D configurations close to the native structure [6].

In this context, according to [5, 79, 80], a MD method usually consists of three elements: a geometric representation of the protein structure, a potential energy function and a surface energy search technique.

The first element of the MD method, the geometric representation, corresponds to the way the 3D structure of the protein will be represented computationally. The most detailed representations include all the atoms of the protein chain and the solvent molecules that surround the protein, such as water molecules (H_2O). These detailed atomic representations have a high computational cost [5].

The second element, the potential energy functions, is used in molecular mechanics simulations, protein design and prediction of protein structures [5]. These functions can be either knowledge-based or based on physical parameters, both of

which representing the forces that determine the macromolecular 3D conformation [5]

A potential energy function is typically made up of two types of terms: *bonded* and *non-bonded*. Bonded terms correspond to those potentials that emerge due to the forces among atoms that share a chemical bond (electrons) between them, for example: *covalent bonds*, *valence angles* and *torsion angles* [5, 81]. *Non-bonded* terms refer to those potentials that emerge due to the forces among atoms that present an attraction between them, although not sharing a chemical bond. These include ionic bonds, hydrophobic interactions, Hydrogen Bonds (HB), *van der* Waals forces, and dipole-dipole bonds (Section 2.1).

Chapter 4 presents a detailed view of this technique, its general algorithm, limitations, challenges and an example tool which implements a parallel version of its algorithm.

Chapter 4

Molecular Dynamics Simulation: A Detailed View

In order to carry out its biological function, a protein goes through structural transitions in its 3D configuration, as presented in Chapter 3. This causes changes both in the geometry of the protein and in the arrangement of molecules around the protein, e.g. the solvent. The energetic and thermodynamic parameters of these molecular systems are derived from realistic simulations which are devised, typically, for finding a 3D configuration with a minimum potential energy and exploring the energy space of the molecular system [61].

In this context, a well-known molecular technique is the Molecular Dynamics (MD) simulation used to study time-dependent behavior of molecular systems. It involves numerically solving Newton's equations of motion to track the positions and velocities of the atoms over time, considering their initial coordinates and velocities and a potential energy function of the system [27, 61].

These MD simulations use physical forces to provide a detailed picture of the atomic interactions in molecular systems and, when applied to protein folding, they depict their relation to the structure and function of proteins [82]. Nowadays, they are routinely employed to obtain insights of the atomic behavior of molecular systems [6].

MD simulation methods can be organized into two main groups according to the approach of the model chosen to represent the physical system [26], namely:

- Classical approach: starting with [83, 84], it refers to all techniques that deal with molecules using the Ball-and-stick model shown in Figure 2.1 (Section 2.1), where the atoms are represented as spheres (*balls*), the bonds or interactions between them are represented as lines connecting them (*sticks*) and the laws of classical mechanics are applied to define the system dynamics;

- Quantum approach: since the work in [85], it refers to methods that calculate the atomic interactions of the system considering the quantum nature of its chemical bonds, that is, quantum equations are applied to calculate a density function of electrons in order to estimate the valence electrons that determines atomic interactions, while still using the classical mechanics equations to calculate the dynamics of ions.

Although the quantum approach represents an advance in relation to the classical approach [26], its practical use in MD simulation continues to be limited by its very high computational cost.

In this context, classical MD simulation method has evolved since the publication of the first protein simulated with it [86]. It has become an established tool in the study of biomolecular systems, complementary to the experimental process, as it offers a way to follow processes that are difficult to understand with experimental techniques [87, 88].

According to [5], the computational methods for simulating the protein folding process can be classified into four groups. This Thesis focuses on one specific group, called *ab initio*. As shown in Section 3.3.3, in general, these methods do not use any data on previously resolved protein structure predictions and are based strictly on the principles of physics, seeking to predict the stable 3D configuration considering only the protein's 1D sequence.

MD simulation is an *ab initio* technique used to study the time-dependent behavior of molecular systems. It involves numerically solving Newton's equations of motion to track the positions and velocities of the atoms over time, considering their initial coordinates and velocities, and a potential energy function of the system. For modeling protein structures, canonical MD is the method of choice.

In general, the MD simulation technique consists of (i) a geometric representation of the 1D structure, (ii) a potential energy function, and (iii) an energy surface searching technique, as explained in [5, 80].

One known limitation of this method is that finding the optimal sequence of geometric transformations which leads to the stable 3D configuration is an NP-Complete problem [89]. Thus, the energy surface searching technique applies energy functions to explore the protein conformational space, describing its energy and atomic forces [5, 79, 80].

This PhD Thesis focuses on the classical MD simulation technique and, from this point on, will refer to it only as MD simulation. Section 4.1 introduces the methodology behind the MD simulation general approach and Section 4.2 addresses other known limitations of this technique.

4.1 Methodology

The MD simulation uses a methodology based on both Classical Mechanics equations of motion and Statistical Mechanics principles [90, 91].

Newton's equations of motion are used to describe the microscopic properties of a molecular system from information on the behavior of each atom of the system over time, i.e. positions and motion of the atoms, in order to generate a sequence of system arrangements as a function of time. From these microscopic properties, Statistical Mechanics is used to calculate observable macroscopic properties, such as number of atoms (N), pressure (P), temperature (T), volume (V) and total potential energy (E) of the system [90–94].

A software that implements a MD simulation, typically follows a sequence of steps that (a) calculates both microscopic and macroscopic properties of the atomic

system; and (b) records these data for subsequent analysis of their properties. Section 4.1.1 presents how this sequence of steps is described through a general algorithm typically used in MD simulation systems.

4.1.1 General Molecular Dynamics Simulation Algorithm

In an overview, the general method used in the MD simulation comprises of four steps [4, 23, 25, 95], namely: (1) inputs the geometric configuration and other initial conditions of the atomic system; (2) calculates the forces acting on the system; (3) updates the geometric configuration; (4) writes to the output all system data collected for intended analysis, such as variations in geometric configuration, temperature, pressure, etc; (5) goes back to step 2, until enough data has been collected.

These steps can be further detailed and structured as shown in Algorithm 1.

Algorithm 1: General MD Simulation Algorithm

1 **** Initial state of the atomic system ****
Input :
a) $\{a_1, ..., a_N\}$: initial coordinates (x,y,z) for atoms 1 to N;
b) $\{v_1, ..., v_N\}$: initial velocities for atoms 1 to N;
c) $Force\ field$: model parameters;
d) $Solvation$: model data (implicit or explicit);
f) $time\_step$: size of each time interval to be calculated;
g) num_steps: # of iterations for the numerical integration loop.

2 **** Numerical Integration Loop ****
3 **for** $ns \leftarrow 1$ **to** num_steps **do**
4 $U_{bonded} \leftarrow$ `ComputeBondedForces` $(a_1, ..., a_N)$;
5 $U_{non-bonded} \leftarrow$ `ComputeNonBondedForces` $(a_1, ..., a_N)$;
6 $U_{total} \leftarrow$ `ComputePotentialEnergy` $(U_{bonded}, U_{non-bonded})$;
7 **** move atoms and update coordinates and velocities ****
8 $\{a'_1, ..., a'_N\}, \{v'_1, ..., v'_N\} : \leftarrow$ `MoveAtoms` $(a_1, ..., a_N)$;
9 $\{a_1, ..., a_N\}: \leftarrow \{a'_1, ..., a'_N\}$;
10 $\{v_1, ..., v_N\}: \leftarrow \{v'_1, ..., v'_N\}$;
11 **** Writes output data to log files ****
12 **** at every predefined interval of iterations ****
Output:
a) $\{a'_1, ..., a'_N\}$: updated coordinates of each atom;
b) $\{v'_1, ..., v'_N\}$: updated velocities of each atom;
c) Potential energy (U_{total}), Temperature (T), Pressure (P), etc.
13 **end**

The initial state of the molecular system is defined in line 1. The initial coordinates of each atom (x, y, z) are provided in vector a for a total of N atoms (Input a). The initial velocities of those atoms (Input b) are normally set according to the desired temperature. The force field model is provided in two files: *parameters* and *topology* (Input c). The solvation model (Input d) is provided by the user before

starting the simulation and defines how the atoms of the solvent will be treated during the simulation. Section 4.1.2 will present more details about this input data. The time step (Input e) provides the order of magnitude of time, which is normally in femtoseconds (10^{-15}) and the number of steps (Input f) is normally set to tens of millions or billions of iterations.

Using the selected force field model (Input c), the force that is exerted on each atom at the position a_i is calculated from the total potential energy of the system, U (Equation 4.1) [96].

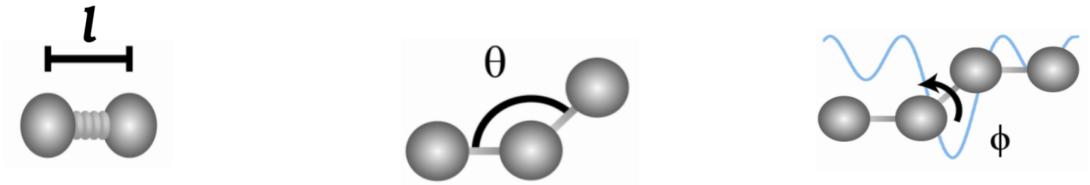
$$F_i = -\frac{\partial U}{\partial a_i}, \tag{4.1}$$

The empirical approach treats U as the sum of the potential energy related to chemical bonded interactions and the potential energy related to non-bonded interactions (Equation 4.2).

$$U_{total} = U_{bonded} + U_{non-bonded} \tag{4.2}$$

The potential energy for bonded interactions is typically calculated taking into account three components, as depicted by the U_{bonded} term in Equation 4.3: (i) harmonic and elastic forces for each covalent *bond* between two adjacent atoms (a_i, a_j), where each bond vector $a_i - a_j$ is separated by distance $d_{ij} = l = |a_i - a_j|$, with equilibrium distance d_{eq} (Figure 4.1a); (ii) three consecutive atoms (a_i, a_j, a_k) and the bending *angle* θ between each bond vector formed among them, such as $a_i - a_j$ and $a_j - a_k$ (Figure 4.1b); and (iii) four consecutive atoms (a_i, a_j, a_k, a_l) connected by three bond vectors with a torsion angle ϕ (*dihedral* angle at Figure 4.1c), due to rotations around the central bond vector.

$$U(a^N) = \underbrace{U_{bonds} + U_{angles} + U_{dihedrals}}_{U_{bonded}} + \underbrace{U_{vdW} + U_{Coulomb}}_{U_{non-bonded}} \tag{4.3}$$



(a) Bond length (distance)

(b) Bond angle

(c) Bond torsion angle

Figure 4.1: Schematic representation of bonded interactions in a molecular system. These include bonds, angles, and torsions, which are explicitly defined by the topology of the molecule. Image adapted from [4].

The potential energy term for non-bonded interactions is commonly estimated by considering *van der* Waals and electrostatics interactions, depicted in Figures 4.2a and 4.2b.

Typically, the potential of *van der* Waals interactions is calculated using the 6-12 Lennard-Jones potential and the potential of the electrostatic interactions is

calculated using the Coulomb law, represented by the terms U_{vdW} and $U_{Coulomb}$ in Equation 4.3, respectively [4].

(a) *van der* Waals forces (Lennard-Jones 6-12 Potential)

(b) Electrostatic forces (Coulomb)

Figure 4.2: Illustration of non-bonded interactions in a molecular system, including van der Waals forces and electrostatic interactions, which are computed based on atomic distances and charges: (a) *van der* Waals forces and (b) Electrostatics forces for a pair of atoms with a distance r_{ij} between them. Image adapted from [4].

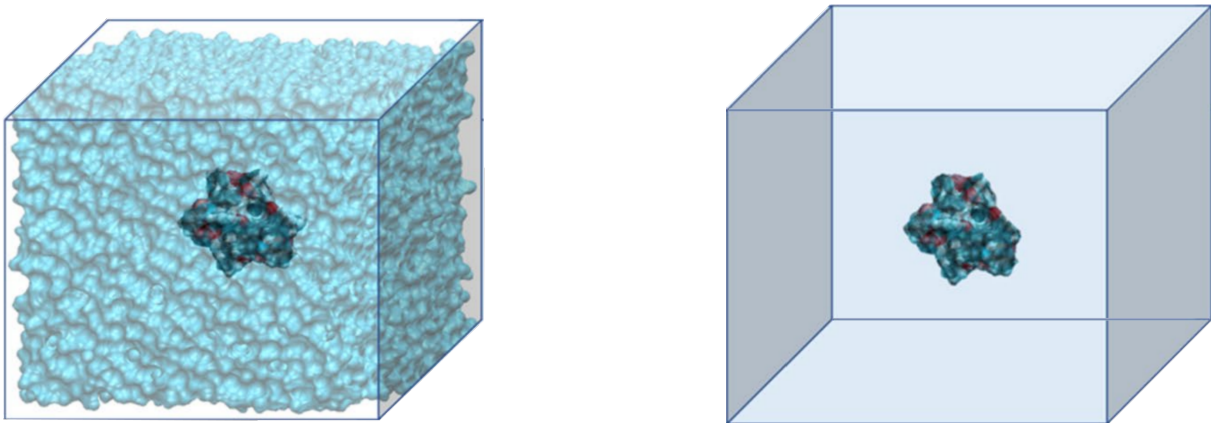
$$U_{total} = \underbrace{\sum_{bonds} k_d(d - d_{eq})^2 + \sum_{angles} k_\theta(\theta - \theta_{eq})^2 + \sum_{dihedrals} k_\phi(1 + cos(n\phi - \gamma_i))}_{U_{bonded}} + \underbrace{\sum_i \sum_{j>i} 4\varepsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] + \sum_i \sum_{j>i} \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}}}_{U_{non-bonded}} \tag{4.4}$$

Inside the main loop (Algorithm 1, lines 3 to 13), two functions are responsible for computing the bonded and non-bonded forces (lines 4 and 5), respectively, and their results are used to compute the total energy of the system in line 6 (Equation 4.4). The function *MoveAtoms* (line 8) is responsible for calculating each atom new position (a') and velocity (v') (lines 9 and 10).

As output (line 11), two files are written at predefined time steps: one with all atom coordinates (Output a) and one with all the velocities (Output b). These files are used to keep track of the trajectory of the molecular system. A third file is also written with properties such as potential energy, temperature and pressure (Output c).

4.1.2 Solvation model: Explicit and Implicit

Another characteristic of the simulated system usually defined before starting a MD simulation is the solvation model (Input d in Algorithm 1). In a solvation model, the solvent represents the environment where the molecular system is immersed (usually, water). Thus, the chosen model will influence how the MD simulation will treat the atoms of the solvent during the simulation. The two typical solvent models used in MD simulations are the explicit solvent and implicit solvent models, shown in Figure 4.3.



(a) Explicit solvent model.

(b) Implicit solvent model.

Figure 4.3: Illustration of explicit and implicit solvent models. Image adapted from [22].

Explicit models (Figure 4.3a) generally describe the system as a container filled with atoms that belong to both the solvent and the molecular system. The shapes of the container can also be defined in the initial model and the commonly used shapes are cubes, parallelepipeds, octahedrals and spheres [23, 25, 27, 90, 95].

Explicit modeling aims to increase the accuracy by including a higher level of detail, which means including in the model all the solvent atoms individually, analyzing them during the simulation and computing their interactions with both atoms of the molecular system and with those of the solvent itself. A well-known consequence of this rich detail approach is that the MD simulations using this solvent model demand an extremely high computing power. [22, 23, 25].

On the other hand, implicit modeling (Figure 4.3b) treats the solvent as a polarized continuous structure and uses a less detailed model, which ignores some characteristics of the solvent's molecular and atomic nature, in order to reduce the computational cost, when compared with the explicit modeling approach. At the core of the implicit modeling there is a potential energy function describing the changes in energy of the system during transformations in the protein 3D structure [27].

In MD simulations of proteins, a commonly used class of techniques for modeling implicit solvent is called Generalized Born (GB) [29] and one example is the Generalized Born Implicit Solvent (GBIS) approach, which is well known in MD simulation of relatively small biomolecular systems. The NAMD software (Section 4.3) implements a version of the GBIS approach, which allows the parallel execution of MD simulations of protein folding using an implicit solvent model, as presented in [97].

For more details about solvent models, please refer to [22, 29].

4.2 Limitations and Challenges

According to [6, 24], molecular dynamics simulations have three well-known major limitations: achieving reliable accuracy in force field models, analyzing and interpreting the resulting trajectories to provide a meaningful description of the system's behavior, and the high computational power required to explore the possible 3D configuration space that constitutes the system's trajectory.

The challenge of achieving reliable accuracy in force field models is mainly related to the intrinsic limitations of the mathematical approximations used to represent the complexity of molecular interactions, which contribute to the emergence of systematic errors, for instance [4, 6].

The challenge of analyzing and interpreting the trajectories obtained from a molecular dynamics simulation is related to the noise and high dimensionality of the resulting trajectories, which complicates the production of a meaningful description of molecular behavior [6, 21, 24].

The third and big challenge is the time constraint due to the high computational cost of space exploration for possible 3D configurations [6, 24]. The relevant timescales in important biological processes span many orders of magnitude, as shown in the Figure 4.4 [26].

In this figure, molecular dynamics simulations with atomic-level detail typically use *time steps* on the order of femtoseconds. However, the scales of other events of interest, such as α helix However, the timescales of other events of interest, such as α-helix formation in secondary protein structures (Section 2.2.2) and the overall folding process, are significantly larger. As a result, molecular dynamics simulations must be executed over billions (10^9) of iterations in order to capture the full trajectory of these events, which occur at time values t on the order of microseconds to milliseconds, respectively [98].

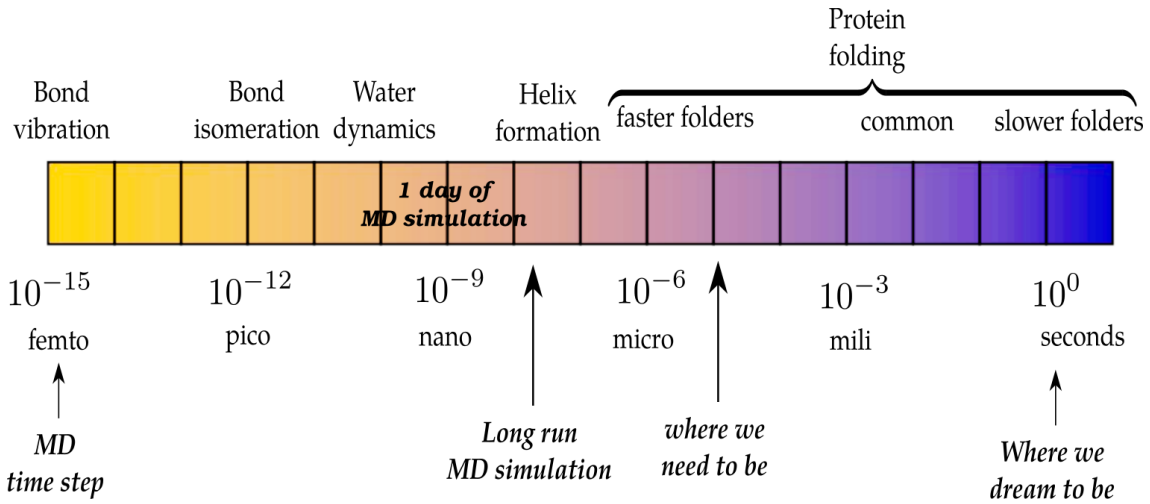


Figure 4.4: Molecular dynamics relevant time scales for protein folding. Image adapted from [98].

As an example of this timescale difference, consider the simulation of the process of oxygenation of the hemoglobin. This process may require tens of microsec-

onds (10^{-3}) [26]. Now, observe that in order to simulate the structural changes in this protein, each *simulated time step* must consider first the fastest atomic motions that can be calculated, and these motions occur in smaller time scales, such as 1 to 5 femtoseconds (10^{-15}). Thus, for simulating the structural changes in the hemoglobin oxygenation process, the numerical integration loop would have to iterate for tens of billions of steps for calculating, at each step, the forces and potentials of the molecular system [26].

To overcome this timescale limitation, computational techniques involving *hardware* and *software* are developed and applied to accelerate calculations performed in a molecular dynamics simulation. In this context, it can be mentioned:

- Building specialized hardware for molecular dynamics simulation, such as the Anton supercomputer [12] developed by the research group *D.E. Shaw Research*, capable of performance of the order of magnitude of microseconds per day of simulated time and therefore achieves longer simulated times (above milliseconds), capturing the complete folding of proteins on this time scale [4, 12, 99]; and,
- Using GPU and parallel programming techniques with a message passing protocol, such as Message Passsing Interface (MPI) [100] to adapt and run the molecular dynamics simulation code in different CPU and GPU *cores* [4, 101–104].

4.3 NAMD

4.3.1 Overview

NAnoscale Molecular Dynamics (NAMD) is a software that implements Algorithm 1 using parallel techniques for high-performance simulations. Figure 4.5 shows NAMD's strategy for a hybrid spatial and force decomposition which allows the parallel execution of the *numerical integration loop* present in Algorithm 1 (line 3) [7, 104–107].

The main elements of NAMD's parallel execution are *Home Patch*, *Proxy Patch*, *Compute*, *Sequencer*, and *Controller* objects, multiple threads and message exchanges [108, 109]. NAMD uses these elements to build its strategy aiming for reducing the execution time of MD simulations of huge molecular systems, in the order of billion of atoms [109].

For MD simulation, NAMD uses the first-principle methods with all atoms using empirical force fields and usually a resolution of time steps in the order of femtoseconds [104]. And for high-performance methods, the focus of NAMD's development is on strong scalability, because the biological systems of interest generally have fixed sizes and, therefore, require a fine-grained parallelization strategy for the MD simulation to be successfully completed in a reasonable time [110].

The next Section 4.3.2 presents an overview of the methodology used in NAMD, starting with its basic objects and followed by a description of its execution flow using these objects.

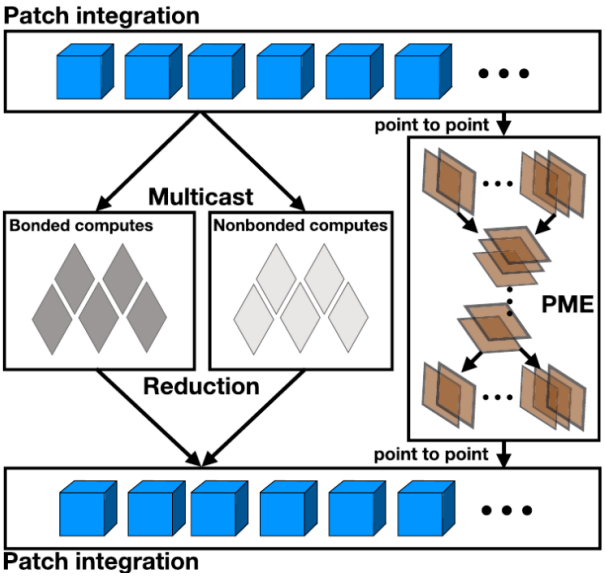


Figure 4.5: Illustration of NAMD's parallel decomposition. Spatial and force calculations represented with *patches* (cubes) and computes abstractions, respectively. Image adapted from [7, 104].

4.3.2 Methodology

NAMD's method comprises of a set of parallel techniques to perform the decomposition of the 3D space and the calculation of forces, aiming mainly to increase the scalability and reduce the execution time of simulations [109]. These techniques are built using Charm++, a *framework* for parallel programming based on object orientation and supported by a *runtime* system [109, 111, 112].

Basic Objects: *Patch*, *Compute* and *Sequencer*

As presented in [109], NAMD has three basic types of objects in its class hierarchy, called *Patch*, *Compute* and *Sequencer* and these objects are essential to NAMD's parallel execution flow of the Algorithm 1.

Patch objects are used to model the spatial decomposition process, as follows: a) the simulation space, i.e. the 3D geometric space containing the molecular system (and the solvent, if a explicit solvent model is chosen) is divided into cells, smaller blocks of that space, all of them with the same volume (usually cubes or parallelepipeds); b) each of these blocks is represented by a *patch* object; c) each *patch* is responsible for the atoms that are inside the block, storing both the coordinates of these atoms and the forces exerted on them; d) each *patch* related to a block in the simulation space is also named a *Home Patch* as a way of referring to a specific *patch* that contains only *local atoms*, which means these atoms are within the same block and, consequently, in the same computational node.

Figure 4.6 shows two *Home Patches* named *Patch A* and *Patch B*. In NAMD, the collection of all *Home Patches* created for mapping the MD simulation's 3D geometric space is called *Patch Grid*.

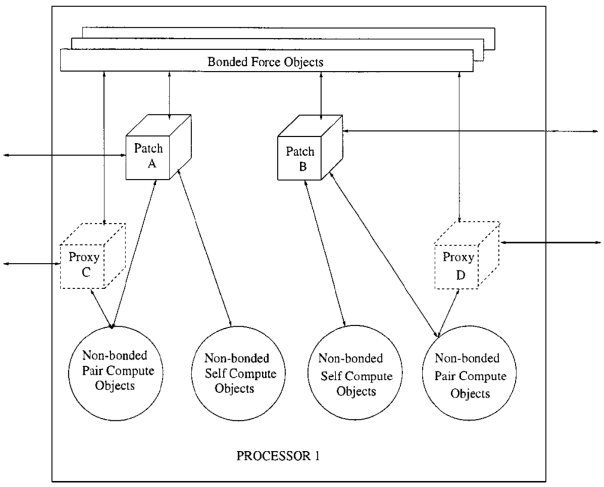


Figure 4.6: NAMD objects created inside a CPU, called *PROCESSOR 1*. The arrows represent the exchange of messages. There are two *patch* objects (cubes *Patch A* and *Patch B*), two *Proxy Patch* objects (dotted cubes called *Proxy C* and *Proxy D*), and four *Compute* objects (circles). Image originally from [109].

Compute objects model the decomposition of forces, as each *Compute* object is responsible for calculating a different type of force, *bonded* or *non-bonded* (see Section 4.1.1), acting on atoms belonging to one or more *Home Patches*. For example, Figure 4.6 shows four *Compute* objects, called *Non-bonded Pair Compute Objects* and *Non-bonded Self Compute Objects*.

There is another particular type of *Patch* object, called *Proxy Patch* (represented by *Proxies C* and *D* in Figure 4.6). This object has the specific function of preventing that unnecessary messages are exchanged between local *Compute* objects (within the same compute node) and a remote *Home Patch* object (inside another compute node), working as follows: if more than one local *Compute* object needs atom positions that are located in a *Home Patch* of another computational node, NAMD creates a local *Proxy Patch* object, which (a) receives the positions of the atoms that are in the remote *Home Patch*; and (b) sends back the resultant forces computed with them. In this scenario, each *Compute* object gets remote atomic positions of a *Proxy Patch*, calculates the corresponding forces, then puts them into the *Proxy Patch*, which is responsible for sending these resulting forces to remote *Home Patch* [109].

In Figure 4.6, four *Compute* objects were created to calculate *non-bonded* forces: the two objects *Self Compute* are responsible for calculating the forces between the atoms located only inside the *Home Patches* A and B (locals); and two other *Pair Compute* objects are responsible for calculating the forces between atoms residing within the local *Home Patches* and atoms that belong to remote *Home Patches*, but have had their coordinates sent to local *Proxy Patches*; so there is communication between the *Compute* and *Proxy Patches* objects C and D (represented by the arrows between the objects), in order to obtain the coordinates of atoms belonging to remote *Home Patches*.

Finally, a *Sequencer* object is created with a main function of managing the execution of a version of the Algorithm 1 in each computing node, using the forces cal-

culated by the *Compute* objects. And, as presented in [108], another function of the *Sequencer* object is to allow the NAMD user to adapt and extend the Algorithm 1 according to the needs of the experiment, through the necessary modifications of specific methods inside the *Sequencer* object. The behavior of the *Sequencer* object will be detailed below, within the context of the NAMD execution flow.

Execution Flow

In its basic operation, NAMD decomposes calculations into objects that interact by sending asynchronous, one-way messages to other objects on local or remote processors. Utilizing the Charm++ message-driven model, a method on an object is only activated when a message is received, designed specifically to minimize resource consumption associated with waiting for data [106, 111]. Figure 4.7 demonstrates how these objects are used to construct the basic flow of NAMD's operation through shared memory and message passing.

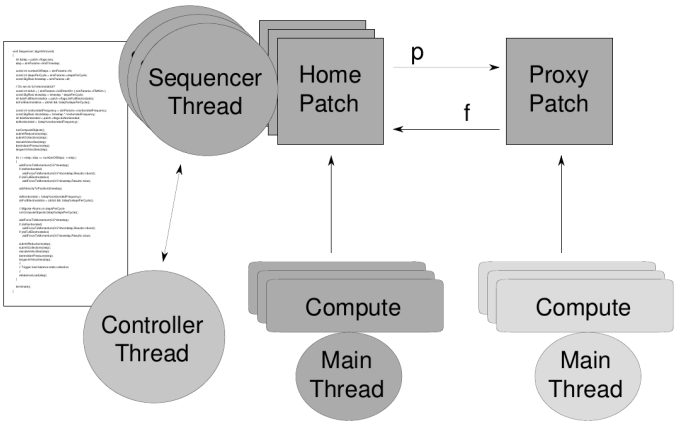


Figure 4.7: NAMD components: basic objects (computes and patches), multiple threads (controller and sequencers) and message exchanges. Image adapted from [108].

At the start of parallel execution, using Charm++'s *runtime* system, a NAMD process is created on each defined compute node. Then, inside the *main* node, the main *thread*: a) performs the domain decomposition, that is, calculates the *Patch Grid* and the number of *Patches*; and b) evenly distributes *Patches* among computational nodes.

Then, on all nodes, the main *thread*: a) creates a new Charm++ *thread* for each *Home Patch* defined at that node; the only method executed by this *thread* is defined in the *Sequencer* object (represented by the scrambled left white box of Figure 4.7); b) creates *Proxy Patch* objects to receive the atoms that are in remote *Home Patches* and to return the calculated forces; c) creates registered *Compute* objects that depend on atoms *contained* in both *Home Patches* and *Proxy Patches* [113].

Next, inside the main node, the main *thread* creates a new Charm++ *thread* to execute the methods defined in an unique type of object, called *Controller*. This

object has similar responsibilities to those present in the *Sequencer*, but with additional ones that must be carry out only inside the main node, which are: a) coordinating the overall execution and integrity of the *numerical integration loop*, by exchanging messages with local and remote *Sequencer* objects; and b) collecting all calculated system energies, computing global results and recording them throughout the whole simulation execution.

In the next step, in all computing nodes, each *Sequencer thread* runs the *numerical integration loop* of the Algorithm 1 only for atoms *contained* in the *Home Patch* associated with it. For each iteration, the *Sequencer*: a) sends the positions of atoms of each *Home Patch* to local *Compute* objects for calculating the corresponding forces; b) sends the positions of *Home Patch* atoms to the remote *Proxy Patch* objects for their corresponding *Compute* objects calculate the forces; c) suspends its *thread* execution, waiting for resulting forces; d) when the last message arrives containing the forces calculated by the last *Compute* object (local or remote), the *thread* is woken up; and e) using the values of these forces, the *Sequencer* calculates the corresponding potential energy; then f) updates the positions and velocities of the atoms, which corresponds to the action performed by the *MoveAtoms* method of the Algorithm 1 and, if necessary, migrates these atoms to new *Home Patches* due to their new coordinates.

Finally, the *Controller thread* gathers and writes (to *log* files) the updated coordinates and velocities, as well as temperature, pressure, potential energy and other predefined *macroscopic characteristics* (see Section 4.1).

For more details, see [106, 108, 109, 113, 114].

4.4 Atomic Burials

4.4.1 Overview

For an atomic system composed of a globular protein completely folded in its native structure (Figure 2.3a, Section 2.3), the distance between an atom and the center of the 3D geometric configuration of this system is called Atomic Burial (AB) [31]. The work presented in [31–33, 35, 115] shows that atomic burial has sufficient information to be used for predicting the stable 3D configuration of globular proteins, starting from the data contained in the protein original 1D chain of amino acids. Figure 4.8 shows the basic method for predicting protein structures using atomic burial and developed for the work presented in [32].

In order to apply the atomic burial concept to the protein folding problem explained in Section 3.1, two new energy terms were developed [32, 33] and added to U_{total}, the total potential energy in Equation 4.4, namely: U_{ab} in Equation 4.5 is the contribution of atomic burial and U_{hb} in Equation 4.6 is the contribution of the Hydrogen Bonds (HB).

$$U_{ab} = \sum_{atoms} B(r_i) \tag{4.5}$$

$$U_{hb} = \sum E_i(\Lambda_i, a_i) \tag{4.6}$$

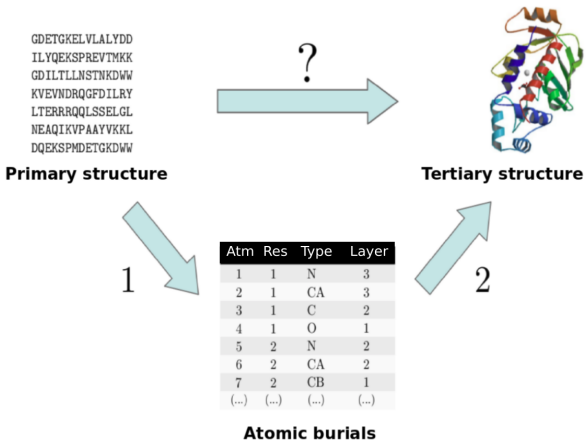


Figure 4.8: Prediction of protein structures with atomic burial. Image adapted from [32].

4.4.2 MDBury Algorithm

The potentials described by Equations 4.5 and 4.6 were applied in molecular dynamics simulations of the folding process of globular proteins. These simulations were carry out using a modified version of the Algorithm 1, called MDBury. Figure 4.9 shows the molecular dynamics simulation algorithm used in MDBury for the work presented in [32, 116].

Similar to the Algorithm 1, Figure 4.9 shows that the MDBury algorithm starts with three initialization steps. First, it reads topology data with the initial configuration of the molecular system. Second, it initializes the positions and velocities of the atoms in the system. And third, it initializes the Annealing Weights (AW) for each hydrogen bond using predefined *hydrogen bond annealing factors*. It is important to mention that, although it is not stated in [116], according to the available implementation of the MDBury, at this point: (i) the algorithm also initializes annealing weights for the atomic burial, using initial *atomic burial annealing factors*, and (ii) it reads *estimated* center distances used for atomic burial calculations.

Once the initialization is concluded, the MDBury algorithm proceeds to the *numerical integration loop* to perform typical molecular dynamics tasks, similar to Algorithm 1. The number of iterations in this *loop* is defined in a configuration file constructed and provided in the initialization process.

For the tasks inside the *integration loop*, it is important to note that MDBury applies a thermostatic control method for maintaining constant the average temperature of the system so that the simulation results are as compatible as possible with those of a biological system. The algorithm for this method is called Berendsen Thermostat [117], and it is applied to calculate a scaling factor used for updating the velocities of the atoms and preserve the desired temperature during the simulation.

So, the first task of the *loop* is the main computation performed at each step: the calculation of the potential energy of each atom in the system, as well as the corresponding forces acting on them, including those related to atomic burial and hydrogen bond. Using Newton's second law, the resulting force is used to update

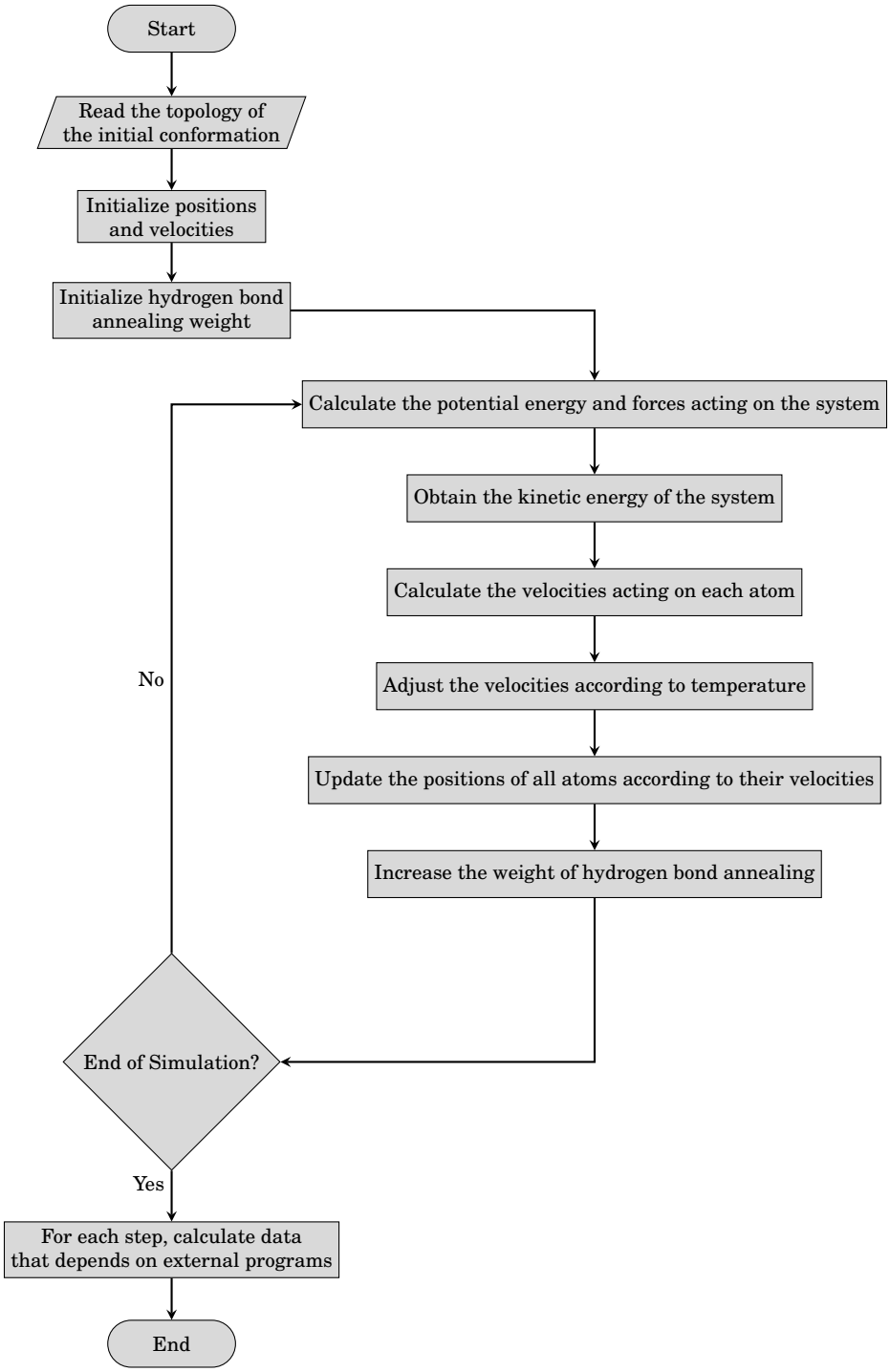


Figure 4.9: Molecular simulation algorithm adopted by MDBury. Image adapted from [116].

the speed of each atom, multiplied by a scaling factor, obtained through the thermostatic algorithm.

MDBury computes the atomic burial force for each atom a at a position i (a_i) using the expected burials (r_i^*), all interval tolerance (δ_i) and applying the atomic burial annealing factor (A_{ab}). The energy contribution of each atomic burial force

is calculated with $B(r_i)$ (Equation 4.7) [32],

$$B(r_i) = \begin{cases} -a_i r^2 + b_1, & \text{for} \quad r \leq \delta_q \\ -a_2 r^2 + b_2, & \text{for} \quad \delta_q < r \leq r_i^* - \delta_i - \delta_q \\ a_3(r - b_3)^2, & \text{for} \quad r_i^* - \delta_i - \delta_q \leq r < r_i^* - \delta_i \\ 0, & \text{for} \quad r_i^* - \delta_i \leq r < r_i^* + \delta_i \\ a_4(r - b_4)^2, & \text{for} \quad r_i^* + \delta_i < r \leq r_i^* + \delta_i + \delta_q \\ a_5 r - b_5, & \text{for} \quad r > \delta_q \end{cases} \tag{4.7}$$

For computing hydrogen bond force, MDBury considers five atoms and put them into a 5-tuple, containing the atom numbers in the following order: (a) one donor nitrogen atom (a_1) and its two adjacent atoms (a_2, a_3), simply called in this Thesis as "3-donor atoms"; and (b) one acceptor oxygen atom (a_4) and its adjacent carbon atom (a_5), also referenced here as "2-acceptor atoms" [32]. The resulting hydrogen bond formed between the donor a_1 and the acceptor a_4, and their adjacent atoms is described by the Equation 4.8:

$$\lambda_{a_1,a_4}(h, \eta, \theta) = F(h)F(\eta)F(\theta) \tag{4.8}$$

where the function $F(\alpha)$ is defined as $F(\alpha) = 1/(1+exp(\beta_\alpha(\alpha - \mu\alpha)))$ and applied for the values of h ($h = |\vec{v_1}|$), η (angle between vectors $\vec{v_1}$ and $\vec{v_2}$) and θ (angle between vectors $\vec{v_1}$ and $\vec{v_3}$). For more details about their definition and usage, refer to [32].

Next, for each donor a_i or acceptor a_j, the energy contribution is given by the Equation 4.9

$$E_i(\lambda_i, a_i) = \frac{1}{2}\epsilon_{hb} f(a_i, \Lambda) \tag{4.9}$$

where $\Lambda_i = \sum \lambda_{a_i,a_j}$ accounts for all possible hydrogen bond formed by a donor a_i or an acceptor a_j, $f(a_i, \Lambda)$ is defined as equal to $F(a_i)(1 - \Lambda)$ for values of $\Lambda \leq 0.95$, and to 0 for values of $\Lambda > 1.05$. Finally, the potential energy term due to all hydrogen bond tuples is calculated by the sum of all energetic contributions, $\sum E_i(\Lambda_i, a_i)$ [32].

In the second and third tasks, the MDBury obtains the instantaneous kinetic energy of the system and it calculates the instantaneous velocities acting on each atom, respectively. And, in the the fourth task, it adjusts the velocities of the atoms according to the desired temperature.

In the fifth task, the positions of the atoms are updated based on the new calculated atomic speed values, ending the Berendsen's algorithm. In the last task of the *loop*, MDBury increases the value of the hydrogen bond annealing weight, as part of the atomic burial solution.

Finally, for each step of the simulation, the molecular dynamics algorithm calculates and writes two output data, using external programs as follows: (i) it registers the Root Mean Square Deviation (RMSD) for each 3D configuration related to native structure; and, then, (ii) it registers the residue ratio of regular secondary structures, α helix and β leaves (Section 2.2.2), for each 3D configuration.

In the next Chapter, we will present parallel techniques used for executing MD simulations in high-performance computing environments.

Chapter 5

Parallel Techniques for MD Simulations using HPC Architectures

5.1 High-Performance Computing (HPC)

High-Performance Computing (HPC) refers to the use of both supercomputers and parallel processing techniques to solve complex computational problems. It focuses on the development of algorithms and processing systems, incorporating administration and parallel computing techniques. The terms High-Performance Computing and supercomputing are sometimes used interchangeably [118].

Using computational resources concurrently, HPC systems have the ability to provide continued performance, a characteristic that makes them suitable for solving advanced problems through tools such as computational modeling, simulations and analysis. Thus, HPC is used in several areas of knowledge, such as oil and gas modeling, automation of electronic design, climate modeling, media and entertainment and molecular biology [118, 119].

5.1.1 Top500 List

Since 1993, the Top500 project publishes twice a year a list of the 500 fastest computers in the world, called the Top500 list [120]. This list is compiled with data provided by HPC experts, computer scientists, manufacturers, and the Internet community at large [118]. So far, this is the only public document specifying the purpose, configuration and processing power of supercomputers on a worldwide scale and it is considered the most authoritative source of information on the capabilities of supercomputers [118].

The Top500 list ranks supercomputers based on their speed when solving a collection of subroutines for calculating systems of linear equations in Fortran. This package of subroutines is called LINPACK benchmark [121, 122] and was created specifically to measure the best performance of a target computer system while solving this system of equations, as it is possible to choose test combinations

between the size of the problem and the type of application in order to obtain the best performance of the target computer system [120].

The main metric used to list supercomputers is the Maximal LINPACK performance achieved (Rmax), given in PFLOPS in the LINPACK report [122] and which represents the best score obtained by the machine in the execution of the biggest problem of the LINPACK package [121].

Table 5.1 presents a reduced example of the Top500 list. The *Rank* column contains the positions in the Top500 list. The name of the supercomputer and the model of the system it is based on are represented in the *Name* and *System* columns, respectively. The column *#Cores* contains the total number of CPU and GPU cores, in each supercomputer and, finally, the column Rmax contains the result obtained after resolving the LINPACK problems, represented in PFLOPS [120]. Note that all supercomputers listed in Table 5.1 are HPC systems that run high-performance molecular dynamics simulations [7, 11, 123–132].

Table 5.1: Selected entries from the November 2024 Top500 list of the world's fastest supercomputers. All systems listed have been used in molecular dynamics simulations. Additionally, MareNostrum 5 ACC (ranked 11th) and MareNostrum 4 (ranked 174th) are both hosted at the Barcelona Supercomputing Center (BSC). [120]

Rank	Name	System Model	*#Cores* (CPU/GPU)	R_{max} (PFLOPS)	MD
2	Frontier	HPE Cray EX235a	9,066,176	1,353.00	Yes [123]
6	Fugaku	Supercomputer Fugaku	7,630,848	442.01	Yes [11, 124]
8	LUMI	HPE Cray EX235a	2,752,704	379.70	Yes [125]
9	Leonardo	BullSequana XH2000	1,824,768	238.70	Yes [126]
11	MareNostrum 5 ACC	BullSequana XH3000	663,040	175.30	Yes [133]
14	Sierra	IBM Power System AC922	1,572,480	94.64	Yes [127]
15	Sunway TaihuLight	Sunway MPP	10,649,600	93.01	Yes [128]
19	Perlmutter	HPE Cray EX 235n	761,856	70.87	Yes [129]
23	Selene	Nvidia DGX A100	555,520	63.46	Yes [130]
24	TianHe-2A	TH-IVB-FEP *Cluster*	4,981,760	61.44	Yes [131]
174	MareNostrum	ThinkSystem SD530	153.216	6.47	Yes [132]

In addition to the Top500 list, which ranks supercomputers based on performance measured by the LINPACK benchmark, there are other classifications such as the *HPCG (High Performance Conjugate Gradients)* and *Green500* lists. The HPCG benchmark evaluates systems using computational patterns more representative of certain applications, while the Green500 ranks supercomputers by energy efficiency, measuring performance per watt consumed. However, since this Thesis focuses on parallel molecular dynamics simulations, which align more closely with the performance metrics of the Top500 list, we have opted not to incorporate the HPCG and Green500 rankings into our analysis.

For the purpose of this document, from now on, the term Top500 List will be referring to the November 2024 edition, unless it is specified otherwise.

5.1.2 HPC Architectures

A survey published in 2022 brought to light a generic platform for running HPC application workloads [134] and it is showed in Figure 5.1. Note that typical HPC applications are usually iterative and tightly coupled, usually based on mathematical models, where is essential the usage of hardware acceleration (GPGPU/FPGA), API acceleration (CUDA/OpenCL) and numerical libraries (BLAS/LAPACK).

Since HPC workloads are sensitive to compute and interconnect types, data is generally shared by message exchanges over high-speed interconnects (InfiniBand/High Performance Ethernet) between computing nodes with high bandwidth and low latency. The widely used communication libraries are MPI and Remote Direct Memory Access (RDMA) [134] and are specifically designed to improve performance by taking advantage of user space communication paths that bypass kernel space, avoiding context switches, using a zero-copy buffer and leveraging high-speed, interconnected HPC infrastructure to reduce overall costs.

On top of that, typical HPC workloads are batch jobs with large datasets on large *Clusters* of distributed computers. In this case, the storage is dedicated to the HPC with distributed parallel file systems that allow simultaneous access to data and provide high speeds and bandwidths.

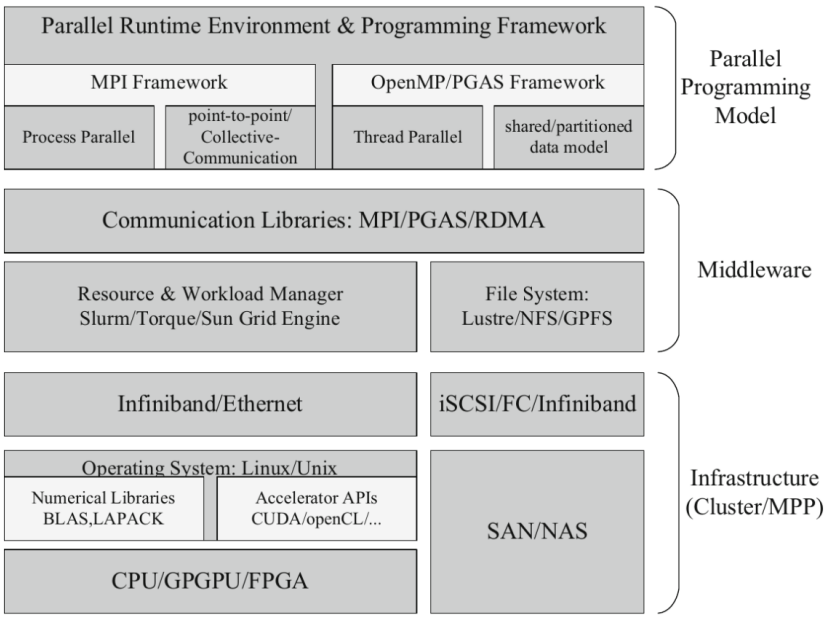


Figure 5.1: Illustration shows a generic HPC architecture in 2022. Image adapted from the original survey presented in [134].

HPC application workloads were executed primarily on specialized and highly expensive supercomputing platforms, such as the Massively Parallel Processors (MPP), which represented 69.2% of the Top500 "Architecture Share" statistics available in the November 2000 edition.

Since then, in the search for alternative HPC infrastructures, *Cluster* computing gained traction and has been moving away from traditional specialized supercomputing platforms towards general-purpose *Clusters* with high-speed intercon-

nections and analogous solutions stacks. Now, twenty three years later, the Top500 List showed that *Cluster* is the main architecture for HPC, with by 88.9% of the architecture share, followed by 11.1% of MPP [120].

5.1.3 Massively Parallel Processing

The term MPP, in its original usage, referred to a class of architectures characterized by a large number of small processors that typically were custom-built and had a Single Instruction Multiple Data (SIMD) architecture, called massively parallel processors, like the *Connection Machines CM-2* [135].

Over time, according to the trend pointed out in [135, 136], the term MPP has been used less accurately to refer to all large-scale multiprocessors, noting that most commercially available multiprocessors with massively parallel processing architecture is really an MPP, in its original sense.

However, this MPP nomenclature for HPC architectures has evolved [135], emphasizing its adoption in the construction of supercomputers that require intensive usage of computation and data [137]. Figure 5.2 shows a simplified example of an MPP architecture with two compute nodes. Each node has its own CPU, memory and network interfaces, and the interconnection between nodes is performed by a specialized network and the communication between different computational nodes is made possible by some message exchange protocol, namely MPI or RDMA (Figure 5.1).

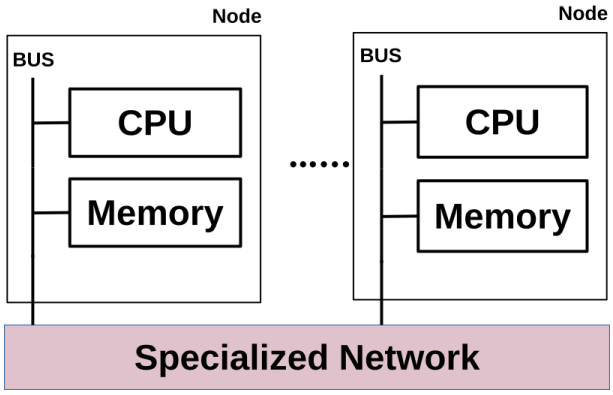


Figure 5.2: Simplified illustration of a generic MPP architecture. Image adapted from [138].

In this context, one of the features that typically distinguish a supercomputer with MPP architecture is the network that connects its processors and allows the machine to operate as one large coherent computational entity. For example, this is the case for several top-ranked systems in Table 5.1, such as the Frontier supercomputer [139, 140].

Frontier Supercomputer

The #2 place in the Top500 List is held by the Frontier supercomputer, located at Oak Ridge National Laboratory (ORNL) in the United States. It is a MPP ar-

chitecture based on the HPE Cray EX235a system and the Slingshot interconnect. It contains a total of 9,408 nodes organized in 74 cabinets: 73 cabinets with 128 nodes and one partially full cabinet with 64 nodes [139].

Figure 5.3 illustrates a Frontier compute node, where each node is equipped with: (a) one AMD EPYC 64C 2GHz processor; (b) four purpose built AMD Instinct MI250X GPUs; and (c) a CPU-GPU interconnect based on AMD Infinity fabric and coherent memory across the node, resulting in a total of 8,730,112 CPU cores, interconnected with a HPE Slingshot system focused in low latency and high throughput for HPC workloads [140].

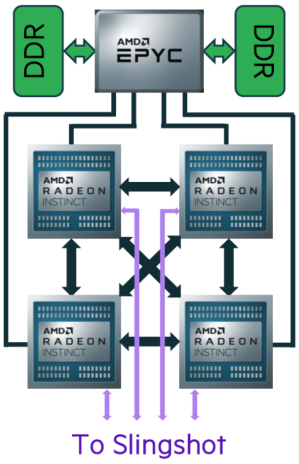


Figure 5.3: Illustration of a Frontier computer node. Image adapted from [139].

Frontier's Slingshot interconnect is configured with a three-hop dragonfly topology, depicted in Figure 5.4, however it supports other topologies such as flattened butterflies and fat trees [139, 141].

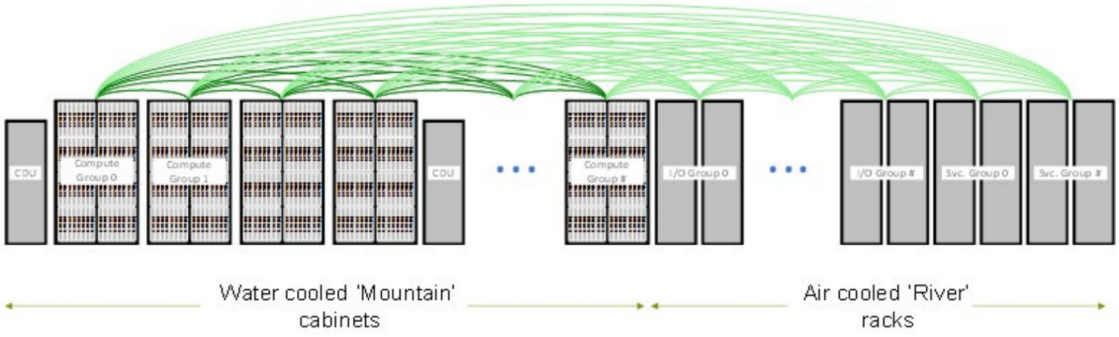


Figure 5.4: Illustration of a Frontier Dragonfly interconnect topology. Image adapted from [139].

5.1.4 Cluster Computing

In the *Cluster* Computing platform, the nodes are connected through dedicated standard network systems and protocols, usually operating with the same type of

operating system [134, 137]. Thus, the nodes are standalone computers interconnected to run together as a single computer. Nodes share resources with other nodes, such as computational modules and directories, typically having an implementation of a message exchange protocol (MPI).

A typical *Cluster* platform has at least two computing nodes in the same cabinet or separately connected to each other through from a dedicated local network [134, 137]. From the point of view of applications and users, this computational solution is seem as a single HPC platform. The main components of a *Cluster* are: network, computer nodes, *middleware* and the application workload [134, 137].

The general architecture of a *Cluster* computing platform is shown in Figure 5.5. The connection network provides the physical interconnection between nodes, either in the same cabinet or through a dedicated standard LAN

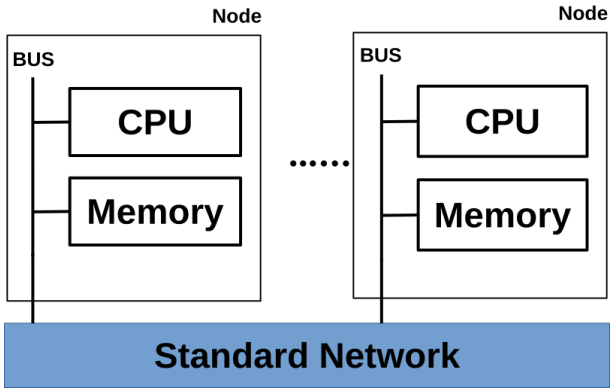


Figure 5.5: Simplified illustration of a generic *Cluster* computing architecture. Image adapted from [137, 138].

Cluster computing platforms are generally fault-tolerant in order to allow active continuous operation across compute nodes. The hardware of the network interface, connecting the nodes, acts as a dedicated communication processor that transmits and receives packets of data between nodes. This interconnection network is usually faster than traditional LAN and its communication protocols are configured for reducing the communication costs [134, 137].

Compute nodes operate as independent computers, however they are connected in way that allows them to handle an application workload collaboratively. The middleware interacts between nodes and applications, sequential or parallel, thus the users have the impression that they are working on a single computer.

As for programming environment for HPC systems, both MPP and *Clusters* platforms generally enlist variations of a set of tools composed of message exchange libraries and shared memory, including MPI, OpenMP and CUDA, *debuggers* and *profilers* for the development of both *software* and *middleware* HPC applications [134, 137]. Note that this configuration is primarily based on Linux and Unix operating system families, which together account for 100% of the systems listed in the Top500 [120].

Leonardo Supercomputer

The 9th place in Table 5.1 is the high-performance Cluster-based system known as the Leonardo supercomputer. Figures 5.6 and 5.7 show its overall system architecture and dedicated interconnect topology.

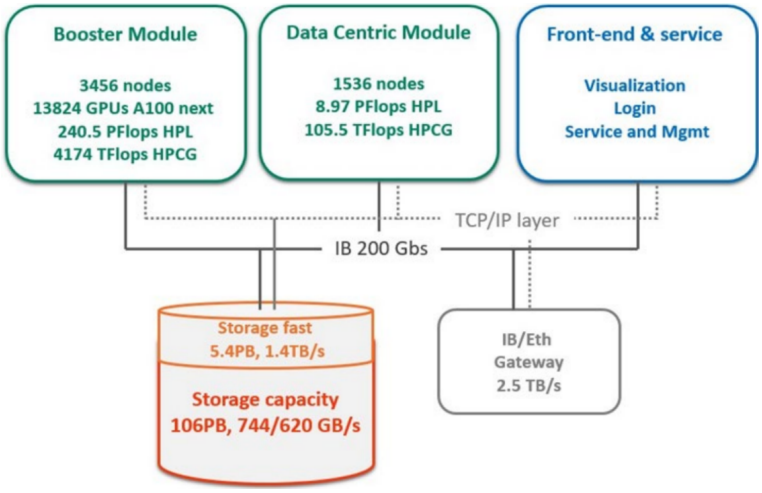


Figure 5.6: Illustration representing the Leonardo system architecture. Image adapted from [126].

Leonardo's *Cluster* architecture is composed of two main modules, a Booster Module and a General Purpose/Data Centric Module (GP-DC) [126]. The Booster module is built to handle maximum computational capacity. It has 3,456 computing nodes, equipped with four Nvidia Ampere based GPUs and with two 32-cores Intel Ice Lake CPUs. The GP-DC module is built for attend a wider range of HPC applications. It has 1,536 computing nodes equipped with two 56-core Intel Sapphire Rapids CPUs per node.

The whole *Cluster* infrastructure is assemble by connecting these two modules, using three other elements: (i) a Front-end and Service, (ii) a Storage area with two tiers, Fast and Capacity; and (iii) a standard high speed interconnect. The Front-end is comprised of 16 login nodes with 2 Ice Lake CPUs (32 cores each), 512 GB RAM and 6 TB disks in RAID-1 configuration; additionally, it has 16 additional nodes equipped with 6.4 TB NVMe disks and two Nvidia Quadro RTX8000 48GB to be used as visualization nodes. The Fast Storage area has net capacity of 105 PB with NVMe and HDD.

The Storage area architecture acts in conjunction with the booster module design by taking advantage of its GPUDirect capability for improving I/O bandwidth and reduce I/O latency towards the GPUs. The Fast storage tier is "full flash" based only in NVMe and SSD technologies for providing high performance especially for AI workloads; and it has net capacity of 5.4 PB with 1,400 GB/s aggregated bandwidth. The Capacity tier provides the parallel filesystem which is based on the Lustre HPC file system [126, 142]; based on NVMe and HDD technologies, it has net capacity of 106 PB and aggregated read/write performance of 744 GB/s and 620 GB/s, respectively.

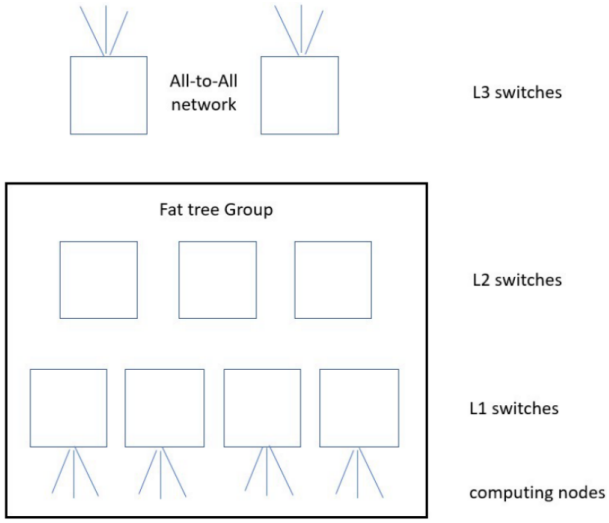


Figure 5.7: Illustration representing the Leonardo dedicated network. Image adapted from [126].

The high speed interconnect is an InfiniBand-based network designed around the Dragonfly+ and "Fat Tree" topologies, as depicted in Figure 5.7 [126, 143]. It aims to allow interconnection of a very large number of nodes with a moderate number of switches, while also keeping the network diameter very small. It features a fat-tree intra-group interconnection, with 2 layers of switches L1 and L2, and an all-to-all inter-group interconnection with a third layer of switches L3.

Marenostrum

The system used for running HPC molecular dynamics simulations in this Thesis - MareNostrum (ranked 174th in the November 2024 Top500 list) - is hosted at the Barcelona Supercomputing Center (BSC). In the same list, a more recent system from the same center, MareNostrum 5 ACC, reached the 11th position, highlighting the continuous evolution of HPC infrastructure in Spain.

Launched in 2004, the MareNostrum 4 cluster comprises 48 racks housing 3,456 compute nodes (Figure 5.8). Each node contains two Intel Xeon Platinum processors (Skylake generation) with 48 cores, totaling 165,888 CPU cores and 96 GB of main memory per node [144].

As illustrated in Figure 5.8, this facility has: (a) two transverse rows with 48 racks hosting the 3,456 *Computing nodes*; (b) one rack for *Computing nodes* with IBM POWER9 and ARMv8 64bit CPUs and Nvidia Volta and AMD Radeon Instinct MI50 GPUs; (c) three racks for *Storage nodes* with General Parallel File System (GPFS) and 14 PB capacity; (d) three *Interconnection racks* that accommodate the fiber optic system for glsopa and Ethernet networks; (e) one rack with *Management nodes* for handling the computing environment and equipped with Linux operating system.

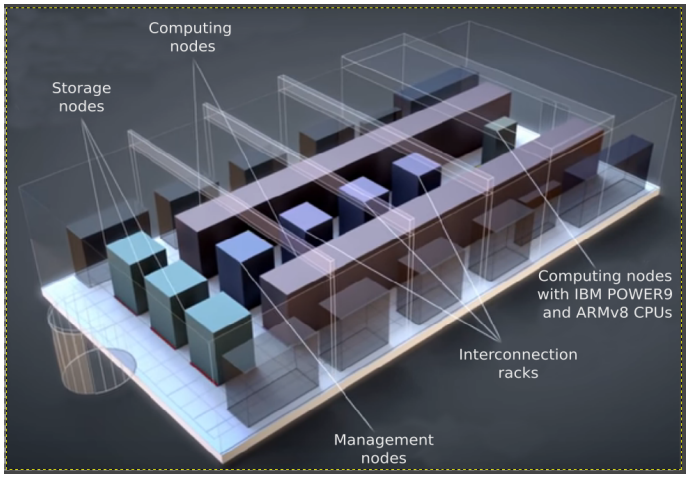


Figure 5.8: *Cluster* computing architecture of the MareNostrum 4 supercomputer.

5.2 Parallel HPC for MD Simulations

This section introduces parallel HPC techniques applied to molecular dynamics simulations, emphasizing the potential of HPC systems. It outlines various parallelization methods and tools used in molecular dynamics simulations. Specifically, Sections 5.2.1 to 5.2.9 detail simulations utilizing the explicit solvent model, discussing techniques and computational optimizations for handling the large-scale interactions between solvent molecules and proteins. In contrast, Sections 5.2.10 to 5.2.12 present simulations using the implicit solvent model, which simplifies the computational burden by approximating solvent effects without explicitly modeling individual solvent molecules. The shift between these approaches demonstrates the varied computational strategies needed for different types of molecular dynamics simulations. Hence, for the purpose of this document, we will refer to this type of molecular simulation as HPC MD simulation or just HPC MD, interchangeably. Similarly, we will reference the software used for executing this type of simulation as a HPC MD tool, package or software, interchangeably.

5.2.1 Parallel MD Simulations on Summit using NAMD

Summit was an IBM-built cluster system located at Oak Ridge National Laboratory (ORNL). Although not present in the Top500 List, Summit was previously one of the world's fastest supercomputers and played a prominent role in large-scale molecular dynamics simulations. Summit was decommissioned on November 15, 2024. Summit contains 4,608 computing nodes, where, each node houses: (a) two CPUs (POWER9) with 22 cores per CPU; and (b) six GPUs (Nvidia Tesla V100) with 5,120 cores per GPU. The nodes are interconnected with a high-speed standard InfiniBand network (Mellanox dual-rail EDR interconnect). Figure 5.9 shows Summit's computing node.

The Summit supports the execution of HPC MD simulations through its platform and diverse MD tools stack available, one of them being the NAMD MD package. HPC MD simulations using NAMD in this architecture were conducted for

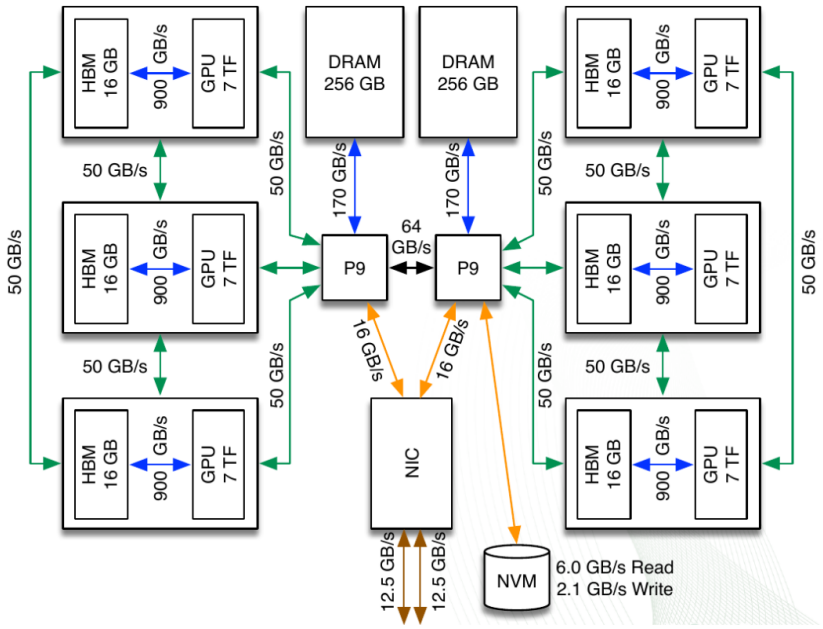


Figure 5.9: Illustration of Summit computing node architecture. Image adapted from [145].

the work presented in Acun et al. [7]. The overall configuration is shown in Figure 5.10.

In this case, Acun et al. introduced a set of algorithmic modifications and performance improvements with the intention of allowing NAMD to make complete use of the Summit architecture, while performing large-scale MD simulations. The modifications targeted the CPU and GPU architectures present on the platform.

Among the optimizations of [7], the main one was made to the data layout in order to boost GPU acceleration and CPU vectoring. This was applied to the expected layout of NAMD C++ objects, or in other words, instead of the expected Array of Structures (AOS) it used a Structure of Arrays (SOA) that is also found in the POWER9 SIMD architecture. A direct consequence of this approach is that critical data structures used during MD simulations were stored in consecutive memory locations, which can be exploited by vectorization loops and GPU accelerators [7].

Other modifications introduced by Acun et al. include (a) increased efficiency of calculations made only in GPU (*offload*); (b) addition of support for the IBM Parallel Active Message Interface (PAMI) interface, in order to increase performance with native code for low-level communication within the Summit platform; (c) optimization of long-distance electrostatic force calculations; (d) changes in the bonded forces calculation routine (*bonded*) to improve load balancing; (e) enabling CPU vectoring for new routines included in NAMD; and (f) the proposal of an alternative method for the Langevin thermostatic calculation used in NAMD, through the implementation of a different thermostatic algorithm, known as *stochastic velocity rescaling* [7].

The largest simulated atomic system had 224 million atoms, using an explicit solvent model in a MD simulation box with dimensions 7 x 6 x 5 and a time step of 2 fs. The parallel execution demanded 1,025 compute nodes with a total of 45,056

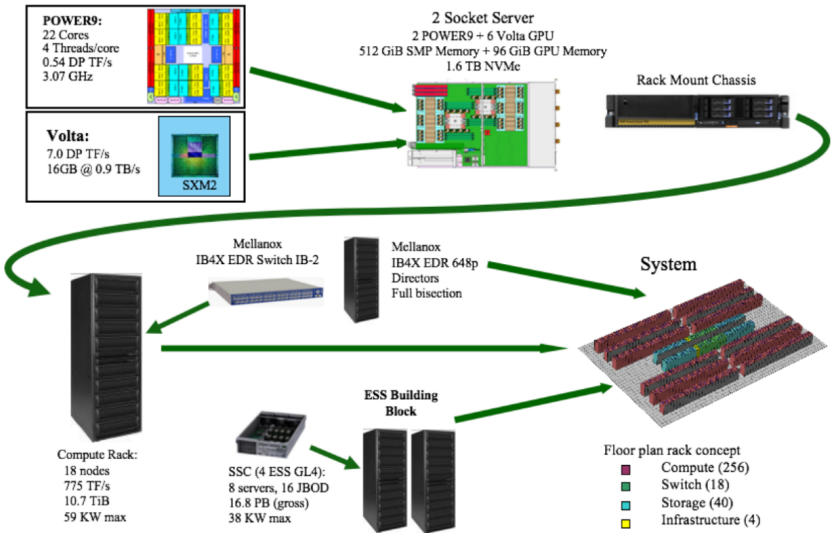


Figure 5.10: Illustration of Summit architecture in which HPC molecular dynamics simulations were performed. Image adapted from [7].

CPU cores and 31,457,280 GPU cores, achieving a routine of 32 ns of simulated time per day of uninterrupted parallel execution.

5.2.2 GROMACS Parallel MD Simulations on TianHe-2

Peng et al. [8] presented a parallel framework based on a HPC MD simulation software, called GROningen MAchine for Chemical Simulations (GROMACS) [23], for running MD simulation on TianHe-2 supercomputer (10th place in Table 5.1). Figure 5.11 depicts the TianHe-2A cluster architecture: (a) shows the logical structure of the computational node; and (b) brings the dedicated network topology, respectively [131, 146].

TianHe-2 first iteration had 16,000 computing nodes, each of them built with two Intel CPUs + three Intel Xeon Phi KNC, using a interconnection with 10 Gbps, 1.6 PB of main memory and 12.4 PB of storage at 512 GB/s. For the second iteration, called TianHe-2A, it has 17,702 compute nodes, with each node with two 12-core Intel Ivy Bridge CPUs and two 128-core Matrix-2000 accelerator, using a 14 Gbps interconnection, 3.4 PB of main memory and 19 PB of storage as 1 TB/s. This combination results in a compute system with 35,584 Ivy Bridge CPUs, 35,584 Matrix-2000 accelerators, and a total of 4,981,760 compute cores [131].

GROMACS implements a parallel execution strategy of the generic MD Algorithm 1, showed in Section 4.1.1. Thus, the framework of [8] explores the architecture of the CPU chips available on each computational nodes of the TianHe-2A. This solution aims the CPU cores and the accelerator cores with the Many Integrated Core (MIC) architecture, in order to reduce the execution time of MD simulations in large-scale runs on this HPC platform. Figure 5.12 shows the relationship among the operational steps needed to execute the strategy of [8].

Paper [8] presented a HPC MD technique that included a three-mode operation in order to accelerate the GROMACS MD execution, while taking advantage of

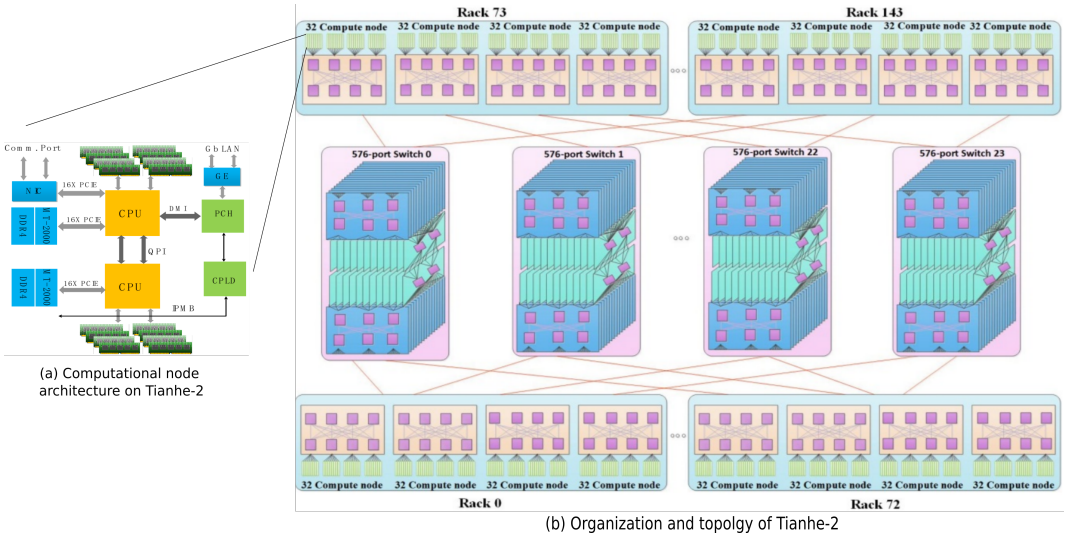


Figure 5.11: Simplified illustration of TianHe-2A Cluster architecture: (a) architecture of one heterogenous computing node, and (b) organization of the TH Express-2 network architecture and topology. Image adapted from [131, 146].

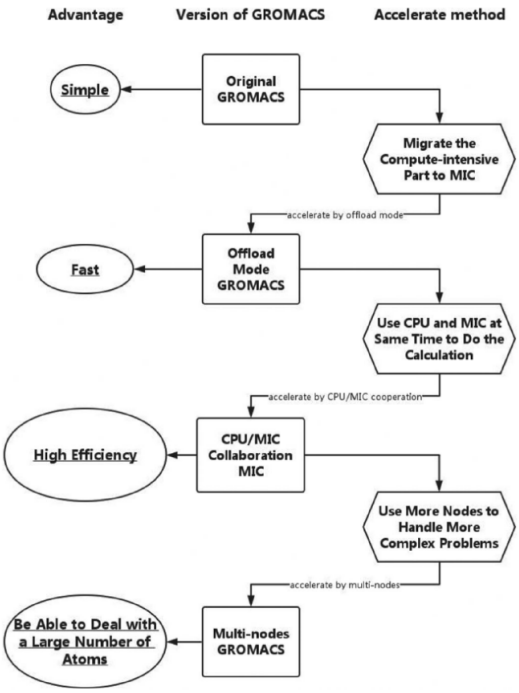


Figure 5.12: Illustration shows the three steps of the CPU/Many Integrated Core (MIC) collaborated parallel framework for accelerating GROMACS MD execution on TianHe-2 computational node. Image adapted from [8].

Tianhe-2A computational node architecture: (1) one mode called *offload*, running only on the MIC Intel Phi cores, responsible for calculating the *non-bonded* forces, and thus decreasing the CPU load; (2) one CPU/MIC collaboration mode, which uses a new data stream to avoid the *overhead* during the synchronization between CPU and MIC; and (3) one method with multiple nodes, called *multi-node*, which

connects both methods used in (1) and (2) to achieve better scalability.

Using the *offload* mode and CPU/MIC collaboration mode methods, the largest simulated atomic system was composed of 300,000 atoms, running on a compute node with 24 CPU cores and 513 MIC cores. And, using the *multi-node* method, the largest atomic system simulated was composed of 100,000 atoms, running on 4 compute nodes with a total of 96 CPU cores and 2,052 MIC cores.

5.2.3 Accelerating MD with LAMMPS on Sunway TaihuLight

Sunway TaihuLight holds the 7th place in Table 5.1 as a MPP system based on the SW26010 processor chip. Each processor is comprised of 4 Management Processing Element (MPE), 4 Computing Processing Element (CPE) (a total of 260 cores), 4 Memory Controller (MC), and a Network on Chip (NoC) connected to the System Interface (SI). Each MPE, CPE, and MC have access to 8GB of DDR3 SDRAM. The total system has 40,960 nodes for a total of 10,649,600 cores and 1.31 PB of memory [147]. Figure 5.13 shows the basic configuration of a compute node.

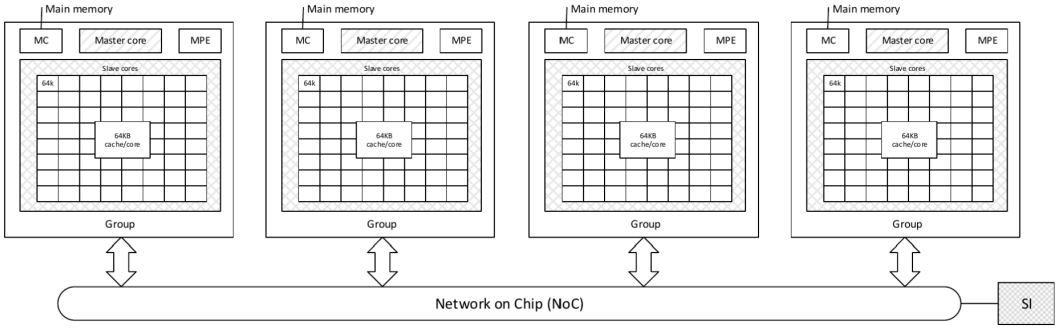


Figure 5.13: Illustration shows the basic compute node layout of the Sunway TaihuLight. Image adapted from [147].

Duan et al. [9] presented a set of optimization techniques to reduce the execution time of MD simulations in the Sunway TaihuLight supercomputer. These techniques deal with memory constraints identified in the *many-core* architecture of the SW26010 processor, for instance, low memory bandwidth, lack of a memory hierarchy and lack of SIMD instructions in the CPEs available [9].

Using HPC MD software, specifically Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) [148], and leveraging its methods for incorporating *Lennard-Jones* (L-J) and *Tersoff* potentials in short-range interaction calculations, [9] introduced a set of optimizations for performing *n-body* computations via a non-regular memory access method.

This way, Duan et al. introduced in his work: (1) a strategy for dealing with memory updates while computing three-body interactions; (2) a method for improving memory bandwidth usage by applying a *software cache*; (3) a custom set of math functions implemented to avoid the need for searches in lookup tables; and (4) a vectoring method to take advantage of the 256-bit SIMD vector registers and use them to improve the calculation of forces in pairs by calculating, in one pass, the forces between an atom and four of its interacting atoms (Figure 5.14).

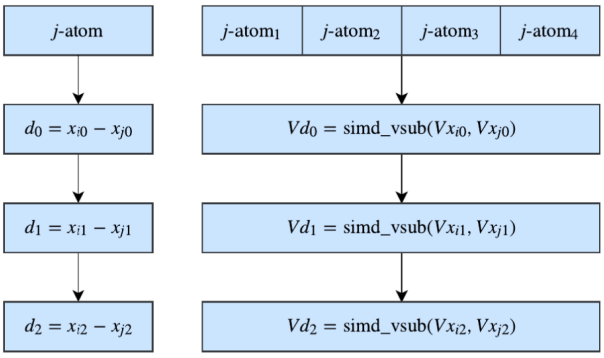


Figure 5.14: Illustration shows vectorization for the inter j-atom: using SIMD instructions, four operations can be executed at once, while running LAMMPS MD simulations on Sunway TaihuLight supercomputer architecture. Image adapted from [9].

During the experiments, these heuristics were applied in MD simulations of large atomic systems: (i) the largest system used was composed of approximately 275 billion atoms; (ii) the calculations were distributed among 16,384 computational nodes and reached a total of 4,194,304 CPU cores, which culminated in the Sunway achieving a peak performance of 2.43 PFLOPS.

5.2.4 High-Throughput MD on BlueGene/Q with LAMMPS

Malakar et al. exploited the high throughput of three supercomputers to reduce the overall running time of the simultaneous MD simulation and analysis workflow for large-scale atomic systems [10]. HPC MD simulations were configured for running with a time step of 4.0 fs and were conducted with simultaneous parallel execution and data analysis on three HPC MPP-platform supercomputers: Mira Blue Gene/Q, Theta and Cori (Table 5.2). Both Cori and Theta use an Aries interconnect, while Mira uses a 5D torus interconnect. Cori and Theta have 622,336 and 280,320 CPU cores, respectively, based on the Cray MPP architecture (Cray XC40) using the Intel Xeon Phi processor family; both with 14,01 and 6,92 PFLOPS Mira is a IBM-based MPP system with a total of 786,432 CPU cores, where each compute node is based on the Blue Gene/Q system Power BQC 16C processor chip depicted on Figure 5.15. By the time of [10] publication, Cori, Theta and Mira held the 12th, 24th and 21th places in the Top500 list, respectively.

Table 5.2: Supercomputers used for HPC MD: Cori, Theta and Mira [10].

Name	System Model	# *Cores* (CPU)	R_{max} (PFLOPS)	Architecture
Cori	Cray XC40	622,336	14,01	MPP
Theta	Cray XC40	280,320	6,92	MPP
Mira	Blue Gene/Q	786,432	8,58	MPP

For the experiments performed in [10], it is worth noting that each HPC MD simulation and MD data analysis were performed in the same application, which are submitted in the same job, taking the name of *spaced-share co-analysis*. However, simulation and MD analysis are performed in 2 (two) different parallel MPI processes, called *partitions*, with data being transferred from the MD simulation partition to the MD analysis. Malakar et al. [10] chose the LAMMPS software [148] for the experiments with simultaneous parallel simulation and analysis.

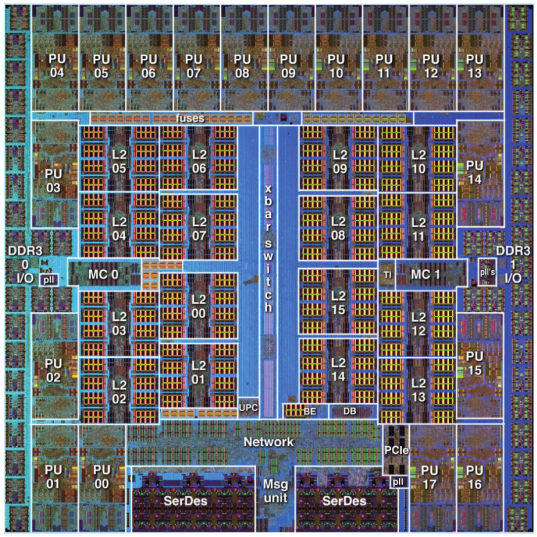


Figure 5.15: Illustration of the Blue Gene/Q Compute (BQC) chip die, used in the Mira Supercomputer. It shows a large Level-2 (L2) cache in the center of the chip, surrounded by 18 CPU units based on the PowerPC A2 CPU core. Memory controllers are integrated on the left and right sides, and, at the bottom, the Chip-to-Chip communication (network) logic Image adapted from [149].

First, the LAMMPS source code was modified for allowing different execution flows and improve the synchronization process between the MD simulation and analysis partitions. Second, considering the growing presence of the *multicore* architecture in computational nodes of its contemporary supercomputers, it was proposed to separate/reserve some CPU cores in each computational node in order to run the analysis in parallel and leaving the rest of the CPU cores available in each node for the intensive computation of the HPC MD simulation (Section 4.2).

Using an empirically determined ratio to choose the number of cores reserved for both analysis and MD simulation partitions, Malakar et al. [10] presented four process mapping heuristics for targeting different embedded interconnections and NUMA domains within a HPC computational node [10]. Third, a method called Mixed Integer Linear Program (MILP) was introduced for obtaining an "optimal" decomposition of processes for mapping both MD simulation and data analysis processes onto the interconnects available within the CPUs, depicted on Figure 5.16.

For the tests performed on the Mira Blue Gene/Q and Cori supercomputers, the largest simulated atomic system had 51 million atoms with calculations distributed among 128 computational nodes, which means that 2048 CPU cores were used in Mira and 4,096 cores in Cori. And, for the simulations carried out on the Theta

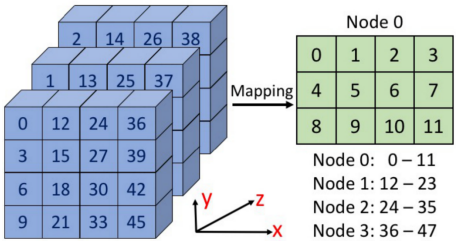


Figure 5.16: Illustration shows the process decomposition map for a 4 x 4 x 3 grid on 4 nodes with 12 CPU cores each. Image adapted from [10].

system, the largest atomic system was composed of 220 million atoms and the MD parallel execution was performed on 512 computational nodes, that is, 33,768 CPU cores.

5.2.5 GENESIS for Parallel MD on Fugaku Supercomputer

Fugaku supercomputer is the 2nd place in the Top500 list. It was designed and built by Fujitsu and RIKEN, following the legacy of its predecessor, the K Computer. The system is installed at the RIKEN Center for Computational Science (R-CCS) in Kobe, Japan, and it has a total of 158,976 nodes. Figure 5.17 shows Fugaku's system configuration and Table 5.3 presents a breakdown of Fugaku's system characteristics.

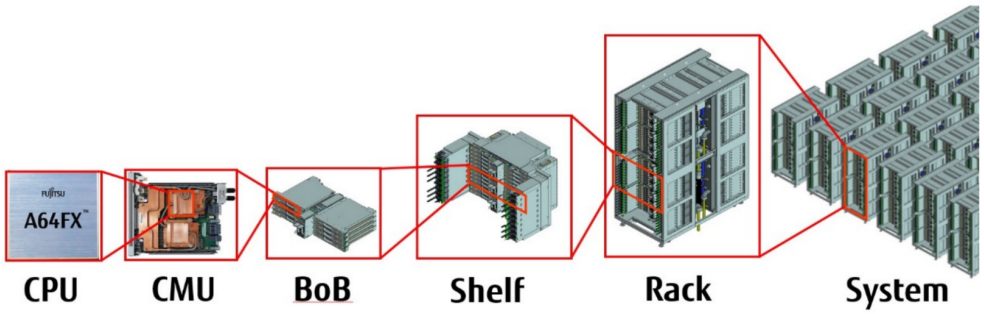


Figure 5.17: Illustration of Fugaku system configuration. Image adapted from [150].

Each computational node is equipped with 1 A64FX CPU (Figure 5.18), which is a many-core Advanced RISC Machine (ARM) CPU with 48 compute cores and 2 or 4 assistant cores used by the operating system. It uses a core design with the ARM V8, 64-bit ARM ecosystem, a Tofu-D interconnect and PCIe Gen3 x16 external connections. Instead of a GPU accelerator, it has the SVE 512-bit x 2 vector extensions. The on-package main memory is a 2nd generation High Bandwidth Memory (HBM2) with 31 GiB, capable of streaming memory, strided, and gather-scatter accesses.

The work present by Jung et al. in [11] and [124] showed the results for executing HPC MD simulations on the Fugaku supercomputer using a MD software called GENESIS. In [11], a molecular system with 1.6 billion atoms was simulated

Table 5.3: Fugaku System Characteristics [150].

Unit	# of Nodes	Description
CMU	2	CPU Memory Unit: 2x CPU
BoB	16	Bunch of Blades: 8x CMU
Shelf	48	3x BoB
Full Rack	384	8x Shelf
Half Rack	192	4x Shelf
System	158,976	As a Fugaku system

through a performance of 8.30 ns/day, using 16,384 nodes of Fugaku. Furthermore, [124] extended this work by performing a 57.72 ns/day MD simulation of a 1.5 million-atom molecular system. In both cases, the simulations were executed on 16,384 computing nodes of Fugaku.

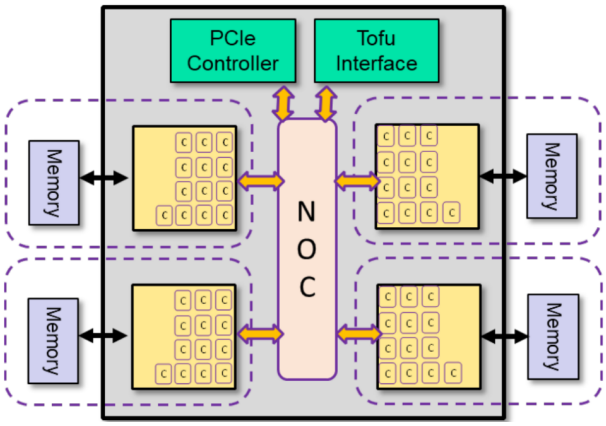


Figure 5.18: Illustration of Fugaku's node CPU: A64FX. Image adapted from [150].

GENESIS was designed for extreme-scale MD simulations on HPC platforms using parallelization and enhanced sampling methods aiming molecular system with million to a billion atoms. The source code is written in Fortran and the software is released to the community under the LGPLv3 license. For the parallelization method, GENESIS includes (i) an algorithm for calculating real-space non-bonded interactions, optimized to maximize performance on ARM CPU architecture; (ii) non-bonded interactions in reciprocal space optimized to minimize communication overhead (iii) evaluations of temperature and pressure which allows MD simulations with larger time steps; (iv) parallel file inputs/outputs (I/O) for MD simulations of extremely huge systems. It also applies a domain decomposition of the simulation space by dividing it into subdomains, and each subdomain into cells. Interactions between particles in different subdomains are computed based on a technique named midpoint cell. Each MPI process (P) has the data of the corresponding subdomain D_p and its margin B_p. Figure 5.19 depicts a 2D example using 16 MPI processes for calculating the long-range atomic interactions in

the MD simulation through running a parallel Fast Fourier Transformation (FFT) algorithm method considering its real and reciprocal spaces.

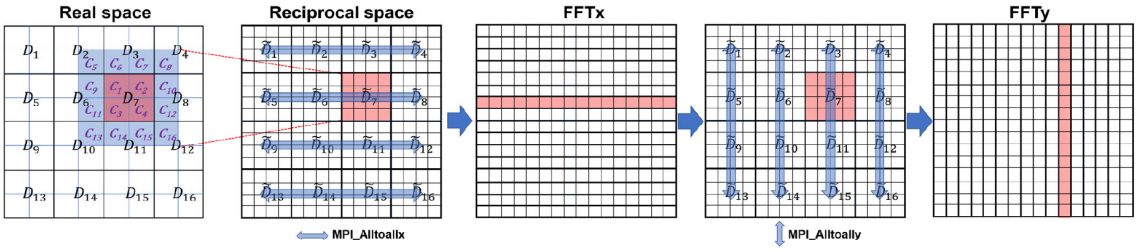


Figure 5.19: Illustration of Genesis domain decomposition method, using a 2D grid for 16 processes. First left grid refers to the real-space and the second left refers to reciprocal-space. Image adapted from [11].

For the real space, process $P = 7$ has the data of 4 (four) cells in subdomain D_7 (inner square colored in red) and its adjacent cells, B_7 (square from C_9 to C_{16} colored in blue). The charge grid data in $\tilde{D}_7$ is obtained from the charge data of atoms in D_7 and B_7. MPI_alltoall communications among four subdomains are done to obtain the global data for FFT direction. For the reciprocal-space interaction, the FFT calculations are perform using two methods: one for calculating the FFT in a forward and backward fashion using five 1D MPI_alltoall routines; and another that uses two 1D MPI_alltoall and one 2D MPI_alltoall routines, which offers less frequent MPI_alltoall calls, however it could involve a much larger number of processes.

For more details about algorithms and techniques implemented on GENESIS for executing MD simulations on HPC system Fugaku, refer to [11, 124].

5.2.6 Anton's Custom Hardware for Large-Scale MD Simulations

Anton is a special-purpose supercomputer built for increasing speed and scale of HPC MD simulations related to basic scientific research, protein folding and drug design. Anton basic platform uses a MPP architecture around specialized chips with an integrated 3D torus network (Figure 5.20a), aligned with a co-designed methodology for hardware, software and algorithms [12].

Figure 5.20b illustrates Anton's chip layout, designed to optimize computation and communication routines. The chip features Core Tiles and Edge Tiles. In the center, there is an array of 24 columns and 12 rows of Core Tiles, dedicated to perform MD computations using specialized pipelines and general-purpose CPUs. On the left and right sides of this array, there is a column of Edge Tiles arranged for handling communication between the Core Tiles and the inter-chip 3D torus network through a off-chip high-speed serial links. The chip has 96 off-chip serial lanes (SERDES transmit/receive pairs), each operating at 29 Gbps in both directions, providing a total bandwidth of 5.6 Tbps [12].

The components of the Core Tile are illustrated in Figure 5.20c. The Core Router facilitates networking by connecting the computing blocks within the tile to a network-on-chip. Two specialized buses traverse the tile, transporting atom

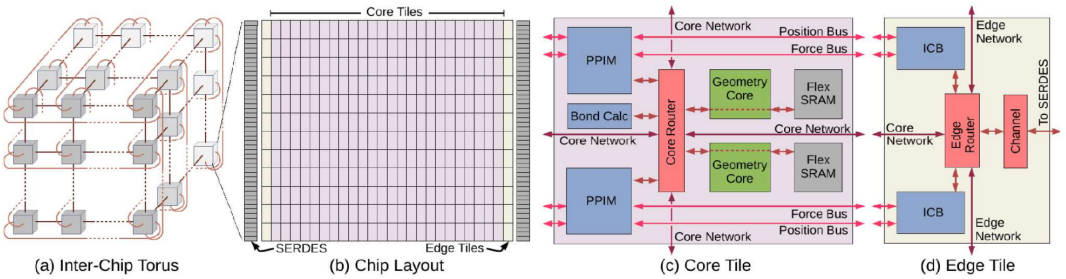


Figure 5.20: Illustration representing Anton 3 system elements: (a) Inter-Chip 3D torus network; (b) Tiled Chip layout with Core Tiles in purple and Edge Tiles in yellow; (c) Core Tile block diagram; and (d) Edge Tile block diagram.
Image adapted from [12].

positions and forces to and from the Pairwise Point Interaction Modules (PPIM), one for positions and another for forces. Each PPIM is equipped with dedicated pipelines for computing non-bonded interactions, while a specialized Bond Calculator (BC) manages the calculation of bonded forces. Additionally, two Geometry Core (GC) units and 128 KiB of Flex SRAM memory handle all time-step processing tasks not performed by the Bond Calculator (BC) or PPIM. The Geometry Core (GC)'s micro-architecture is optimized for 3D geometry and MD computations [12].

Figure 5.20d illustrates the Edge Tiles, which house the logic for off-chip links, called *Channels*. For each *Channel* connects to one of the chip's six neighboring nodes in the 3D torus via a set of SERDES. Each *Channel* is also connected to an Edge Router, which, along with other Edge Tiles on the same side of the die, forms an Edge Network that allows traffic to transition across dimensions in the inter-chip 3D torus network. Additionally, the Edge Router is linked to the Core Tile's 2D mesh network for traffic injection and ejection. Finally, the Interaction Control Blocks (ICBs) connect the Edge Router to the Force and Position Buses, which run throughout the Core Tile array [12]

In the Anton system, each chip is connected to a low-power host processor that provides node control, management, and two external interfaces for data I/O. Anton holds 128 nodes organized as four backplanes with 32 node boards each. Multiple backplane designs allow different torus configurations, starting with 8 nodes with a 2×2×2 configuration reaching to 512 nodes if using a 8×8×8 configuration [12].

For the HPC MD simulation, Anton's parallel execution of the Algorithm 1 operates as follows: (a) the molecular system's 3D space is partitioned into contiguous boxes, with each box allocated to a node such that neighboring boxes are assigned to adjacent nodes in the system's torus topology (Figure 5.20a); (b) each computing node calculates the forces between atoms within its designated box, known as the *Home Box*; (c) atom coordinates are transmitted to other nodes that require them to compute forces between atoms residing in different boxes; (d) finally, the resulting force data is returned to the original node, where it integrates the forces and updates the atom coordinates before initiating the next simulation time step.

Among the results presented in [12], Anton 3 successfully simulated the folding process of a small benchmark protein, DHFR, consisting of 24,000 atoms, using 64

compute nodes and achieving a simulation rate of 212 µs/day. For a larger system, Anton 3 required approximately 5 hours to simulate a ribosome composed of 2.2 million atoms, using 512 compute nodes and achieving a simulation performance of 20 µs of simulated time per day.

5.2.7 Integrating Machine Learning with OpenMM for MD on Summit

In [13], Lee et al. introduced DeepDriveMD, an agent-based, parallel deep learning framework designed to accelerate protein folding MD simulations. DeepDriveMD leverages a Convolutional Variational Encoder (CVAE) to learn features in an unsupervised manner while enabling the simultaneous execution of numerous simulations within its framework.

In this approach, Lee et al. incorporated a computational motif into the generalized workflow of enhanced sampling MD simulations for protein folding. This workflow can be structured into four steps, as outlined in Figure 5.21.

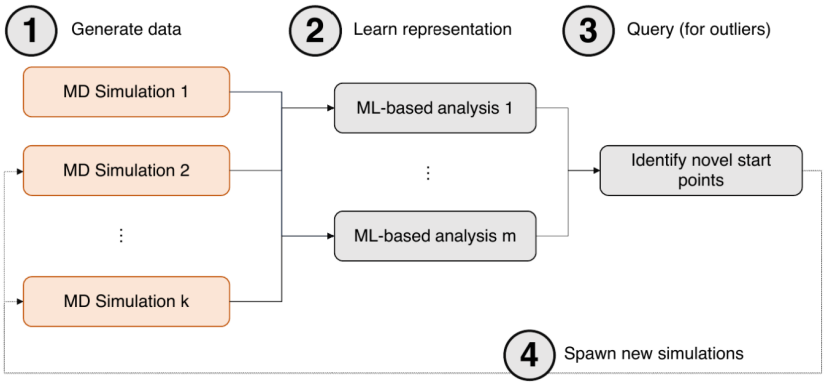


Figure 5.21: Illustration representing a DeepDriveMD's computational workflow in order to couple HPC MD simulations with its Machine Learning approach. Image adapted from [13].

In the first step (1), multiple MD simulations are run in parallel to generate a large set of MD data, which is then used as input for the second step (2). Here, a Machine Learning (ML) algorithm is responsible for the training phase of the CVAE model. In the third step (3), the CVAE performs an inference process to identify new starting points (i.e., conformations) for MD simulations. Finally, in the last step (4), new MD simulations are launched using the identified starting points to expand the initial simulations.

To deploy this workflow on the Summit supercomputer, Lee et al. used a modern HPC tool to manage the heterogeneous computational tasks (both MD and ML) in the DeepDriveMD framework. According to Lee et al., both MD simulations and ML tasks were accelerated using GPUs: the former using a HPC molecular dynamics toolkit called OpenMM [103], and the latter with a VAE framework based on Keras/TensorFlow. Figure 5.22 shows the heterogeneous tasks in green boxes. Items 1 and 2 handle the processes that manage task execution, while items 3, 4,

and 5 are responsible for creating and deploying the computational tasks (MD or ML).

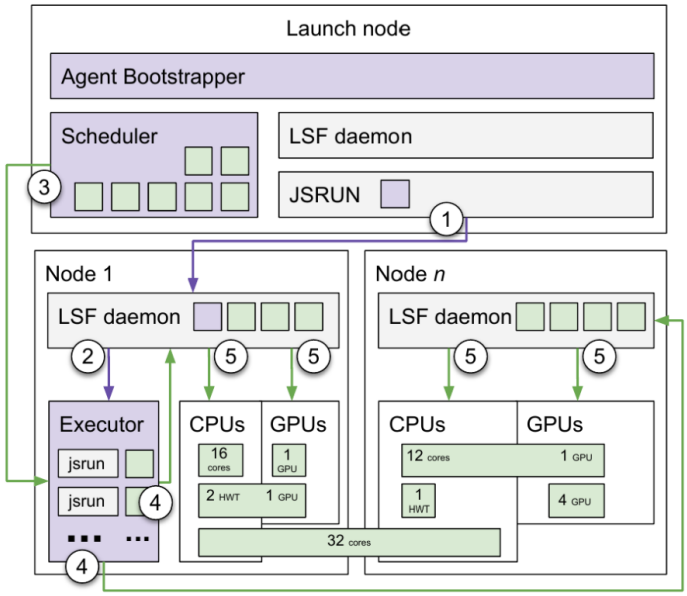


Figure 5.22: Illustration representing the deployment of the DeepDriveMD workflow on Summit supercomputer. Image adapted from [13].

The largest simulation was performed on the fast-folding variant of the villin headpiece protein (VHP), consisting of 35 amino acids, using an explicit solvent model. The simulation spanned 0.9 μs, but within this time frame, the native 3D structure did not form. The simulation used up to 140 nodes of the Summit supercomputer

5.2.8 GaMD-Accelerated Simulations on Gordon

The Gordon supercomputer uses a *cluster* architecture and has a theoretical peak of 341 TFlop/s. It is hosted at the San Diego Supercomputer Center (SDSC) in the United States. Figure 5.23 illustrates its overall architecture.

It comprises of 1024 compute nodes and 64 I/O nodes. Each compute node contains two Intel EM64T Xeon E5 (Sandy Bridge) CPUs with 8 cores, 2.6 GHz and 64 GB of memory (DDR3). Each I/O node consists of two CPUs Intel X5650 (Westmere) with 6 cores, 2.67 GHz, 48 GB memory (DDR3) and sixteen Intel 710 SSDs with 300 GB per unit (4.4 TB total) [151]. The system is interconnected via network topology of 4x4x4 dual 3D torus (Figure 5.23a), with adjacent switches connected by three 4x QDR InfiniBand links with 120 Gb/s. Each switch conects 16 compute nodes (Figure 5.23b) with one I/O node throught a 4x QDR with 40 Gb/s [151].

Pang et al. [14] integrated a method called Gaussian accelerated Molecular Dynamics (GaMD), described in [152], into NAMD with the goal of reducing the execution time of protein folding simulations. GaMD applies a harmonic boost potential to smooth the potential energy surface, thereby accelerating conformational transitions and ligand binding.

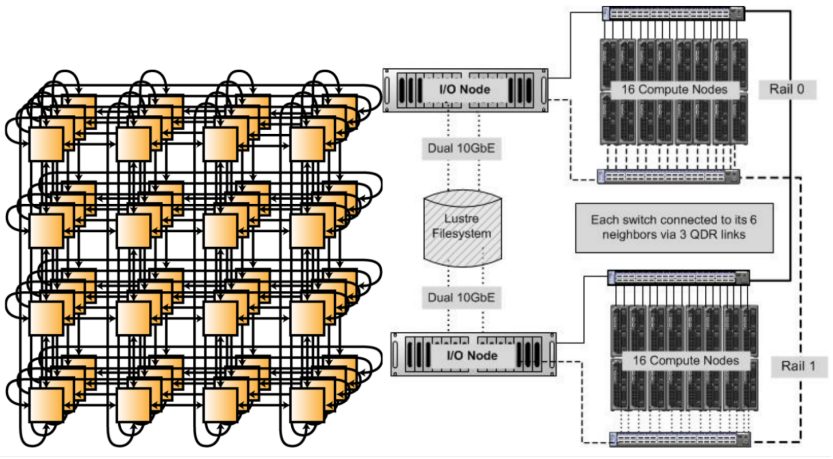


Figure 5.23: Illustration representing Gordon *Custer* architecture. (a) 3D torus topology of switches; (b) network architecture with I/O nodes, Lustre Filesystem and subracks (Rail 0 and Rail 1) for the I/O compute nodes. Image adapted from [151].

The NAMD code was modified to incorporate three new modes of operation when using GaMD: (a) in mode one, only the dihedral potential term from Equation 4.3 is boosted; (b) in mode two, only the total potential energy term is boosted; (c) in mode three, both the dihedral and total potential energy terms are boosted.

Using the dual-boost mode (mode three), Pang et al. [14] successfully simulated the folding of the chignolin mini-protein, consisting of 1,912 atoms, along with 630 solvent (water) molecules. A 2 fs time step was employed, and the simulations ran on the Gordon supercomputer using up to 640 CPU cores, achieving a simulation rate of 61 ns/day.

5.2.9 Real-Time MD Analysis on Cori using NAMD

As mentioned in Section 5.2.4, Cori is a supercomputer built with MPP architecture (Cray XC40). Figure 5.24 illustrates Cori's overall architecture, network and storage. It contains 9,688 Intel Xeon Phi and 2,388 Intel Haswell processors, amounting to 622,336 cores in the computing nodes interconected with a Aries High-Speed network, achieving a peak computational performance of 27 Pflop/s. For the storage it consists of a Lustre file system comprised of almost 10,000 disks organized as 248 Lustre Object Storage Targets (OST). Each OST is configured with GridRAID and has a corresponding Object Storage Server (OSS) for handling I/O requests. The total size of the file system is close to 30 PB with a peak I/O bandwidth of 744 GB/s.

Using this HPC platform, Taufer et al. [15] proposed a strategy to tackle protein folding with simultaneous MD simulation and analysis by evaluating the impact of dropping frames (snapshots of 3D configurations) in the accuracy of the result. As long as the frames are generated by the simulation task, they are sent to the analysis task using a Remote Direct Memory Access (RDMA) framework.

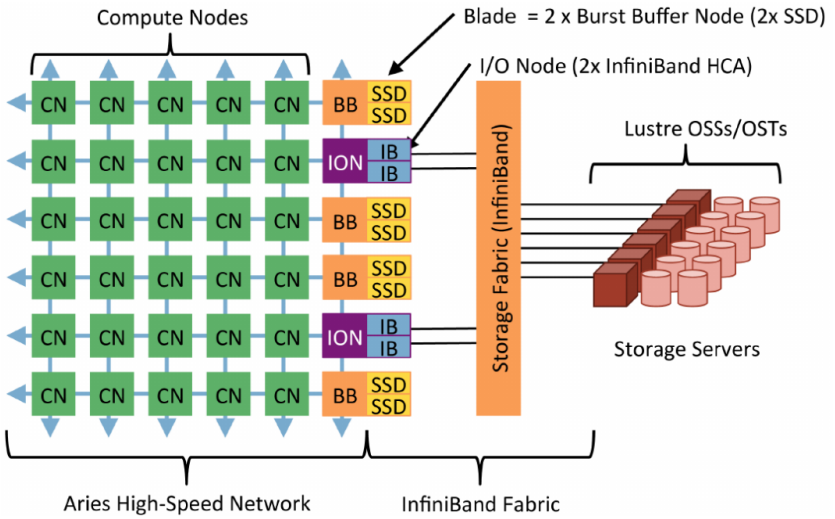


Figure 5.24: Illustration representing Cori system architecture. Image adapted from [153].

Figure 5.25 shows the workflow introduced in [15] based on a producer-consumer pattern with the producer being the Data Generator and the consumer responsible for the Data Analytics. It works as follows: for the data generation, MD simulations are executed by a HPC MD tool which produces fames at regular intervals, called strides. [15] chose NAMD as the software to perform the MD simulations. Then, a tool called *Plumed* was used to read each frame from the MD simulation and transfer to a shared memory area, called *DataSpaces*, through a software module called *Ingestor*. Here, *DataSpaces* works like a RDMA serving as a memory-to-memory framework for a Data Transport Layer (DTL). For the data analytics, a retriver module passes each frame from the *DataSpaces* to the modules responsible for the actual analysis, using collective variables (CVs) and A4MD analytics algorithms. For more details, see [15].

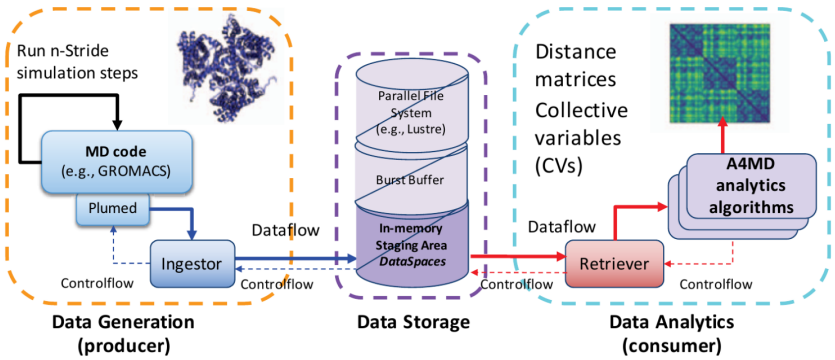
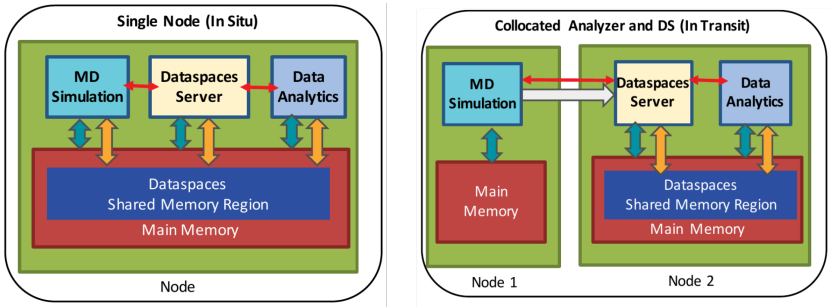


Figure 5.25: Illustration representing the workflow scheme for integrating a HPC MD simulation with a MD analytics with *DataSpaces*. Image adapted from [15].

For evaluating metrics of interest, such as lost frames and idle times of the analysis, or time spent while waiting for a I/O response, Taufer et al. devised two workflows for execution and data collection, called *In Situ* and *In transit* workflows, shown on Figures 5.26a and 5.26b, respectively. For the *In Situ* workflow (a),

both execution and analysis are perform in the same computing node of the Cori supercomputer, which *DataSpaces* server works as a manager of a shared region of the main memory. For the *In transit* workflow, the MD simulation is executed on a separate computing node from the one responsible for managing *DataSpaces* server and the Analytics module.



(a) in situ (b) in transit

Figure 5.26: Illustration representing the integrated analytic workflows on Cori supercomputer nodes using DataSpaces, that is: (a) *in situ* workflow with MD simulaton and analytics performed in a single node; and (b) *in transit* workflow with MD simulation and analytics performed in different nodes. Image adapted from [15].

The authors also proposed a model to predict which frames are lost in a given MD trajectory and with it tried to reduce the I/O waiting time by dropping specific frames that were predicted to have low impact on the accuracy of the folding states. For the MD simulations, 32-core nodes of Cori were used and the protein studied was 1BDD, with 478 atoms.

5.2.10 Adaptive-Resolution Parallel Particle Mesh for MD

In [16], Awile et al. ran HPC molecular dynamics simulations using a custom Fortran 90 implementation of Adaptive-Resolution (AR) neighbor lists, integrated into the Parallel Particle Mesh (PPM) library. The benchmarks were performed on a 2.8 GHz Intel Xeon E5462 CPU using the Intel Fortran compiler version 12.0 with the -O3 optimization flag. This hardware configuration was used to evaluate the efficiency of AR cell lists against conventional cell lists, specifically in handling varying cutoff radii in particle simulations.

Awile et al. core contribution is an AR cell list algorithm, which is based on the domain decomposition depicted in Figure 5.27. The AR cell list algorithm handles particle interactions in simulations with varying interaction cutoff radii. Instead of using uniform cells, the domain is subdivided into smaller or larger cells considering the particle local cutoff radii. This is done using a hierarchical tree structure, where particles with smaller cutoff radii are placed in finer cells and those with larger radii are placed in coarser cells. By adjusting the cell sizes to the particle interaction ranges, the algorithm reduces the number of unnecessary particle-pair checks, improving computational efficiency.

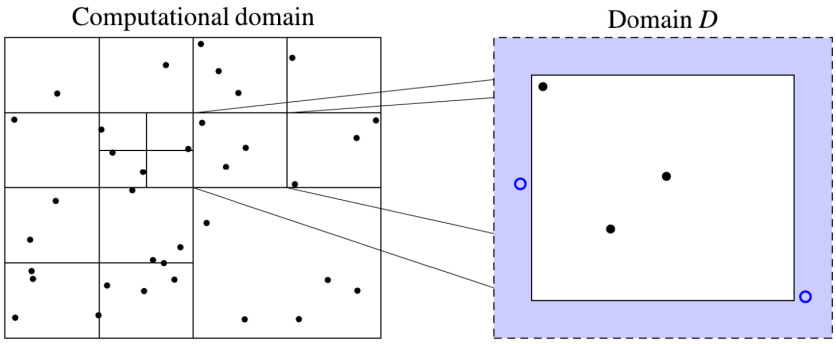


Figure 5.27: Illustration representing domain decomposition used in the AR cell list algorithm. Blue area representing halo layers with ghost particles. Image adapted from [16].

For the computational domain, it is decomposed into cuboidal subdomains, with each subdomain extended by halo layers (blue area) containing ghost particles. In Domain D, black dots represent real particles from the adjacent subdomains. This setup enables transparent implementation of boundary conditions and parallelism. The algorithm allows access to interaction partners even when the interaction cutoff radius varies spatially. This use of AR cell lists enables the construction of Verlet lists and more efficient particle interaction computations.

The results presented in [16] show AR cell list algorithm with better performance than conventional cell lists, especially as the number of atoms increases. The study benchmarks the algorithm over a range of particle distributions and resolution spans, with AR cell lists results showing increasingly efficient as the ratio between the largest and smallest interaction cutoffs grows. For large-scale simulations, where particle cutoff radii span several orders of magnitude, the AR cell list method handled up to 1,000,000 particles with varying cutoff radii, delivering a runtime improvements of nearly three orders of magnitude compared to conventional approaches.

5.2.11 Parallel Non-Bonded Force Computations with `mdcore`

In [17], Gonnet introduced a different variation of the Verlet list algorithm for speeding up the computation of non-bonded force interactions between atoms in molecular dynamics simulations: Pairwise Verlet list and Pseudo Verlet list. Basically, a Verlet list is a list of interacting atoms for a given domain (space).

MD simulations performed [17] and [18] used the *mdcore* library, which supports shared-memory parallel MD. The simulations were conducted on a 4x quad-core 2.5 GHz AMD Opteron 8380 node from the ETH Zürich *Brutus Cluster*. Single precision was used for particle positions, velocities, and forces to enhance memory efficiency and optimize SIMD parallelization.

The parallel implementation of the Pairwise Verlet lists, as depicted in Figure 5.28, works by dividing the computational domain into cells and creating pairwise Verlet lists for each pair of neighboring cells. In a shared-memory parallel system, multiple threads operate on these cell pairs independently, with each thread

selecting a pair of cells to compute particle interactions. To avoid concurrency issues, when one thread is processing a pair of cells, those cells are marked as "in use" preventing other threads from accessing them until the interaction computation is complete.

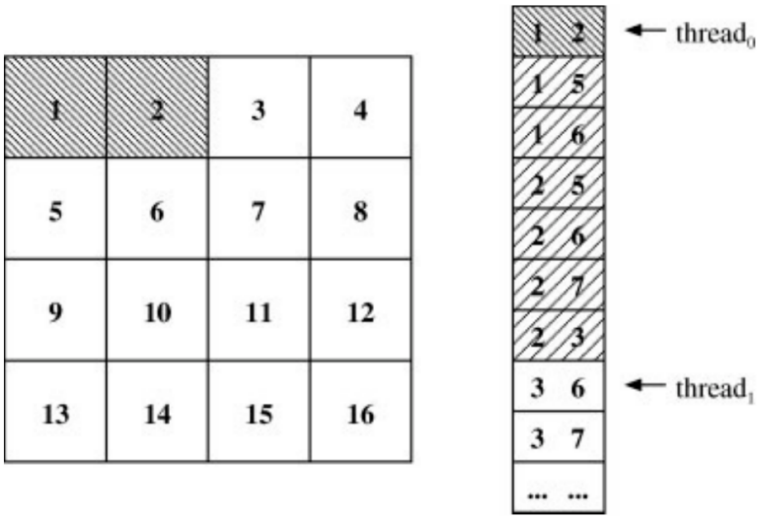


Figure 5.28: Illustration representing parallel execution of the pairwise Verlet list with two threads, using HPC MD `mdcore` tool. Image adapted from [17].

For $thread_0$ in Figure 5.28, processing the cell pair (1, 2), both cells are marked as being in use (dark shading). Consequently, any other thread, such as $thread_1$, is prevented from selecting any cell pairs that involve either of these cells (light shading). For example, $thread_1$ would then choose a different pair, such as (3, 6), to avoid conflicts

This approach simplifies parallelization by focusing on cell pairs rather than individual particles, reducing the need for fine-grained data locking. Threads continuously check for available cell pairs that are not in use, process them, and then mark the cells as available again. This design ensures efficient load balancing and minimizes memory access conflicts, improving both cache locality and overall parallel efficiency.

The study tested the pairwise Verlet lists method on simulations of bulk water and liquid argon with implicit solvent, varying particle densities and interaction cutoff radii. The results showed that for high-density systems, such as bulk water, the pairwise Verlet lists achieved up to 27% faster performance on a single core compared to traditional Verlet lists, and maintained parallel efficiency as the number of processor cores increased.

The work present in [18], introduced *pseudo-Verlet lists*, a more compact and memory-efficient alternative to traditional and pairwise Verlet lists. As shown in Figure 5.29 particles within neighboring cells are sorted along the axis joining the cell centers. The left and right cells are sorted in descending and ascending order, respectively, which optimizes interaction calculations by limiting the search space for particle interactions to particles within the cutoff distance along this axis. This significantly reduces memory use and improves cache performance, especially on multi-core systems.

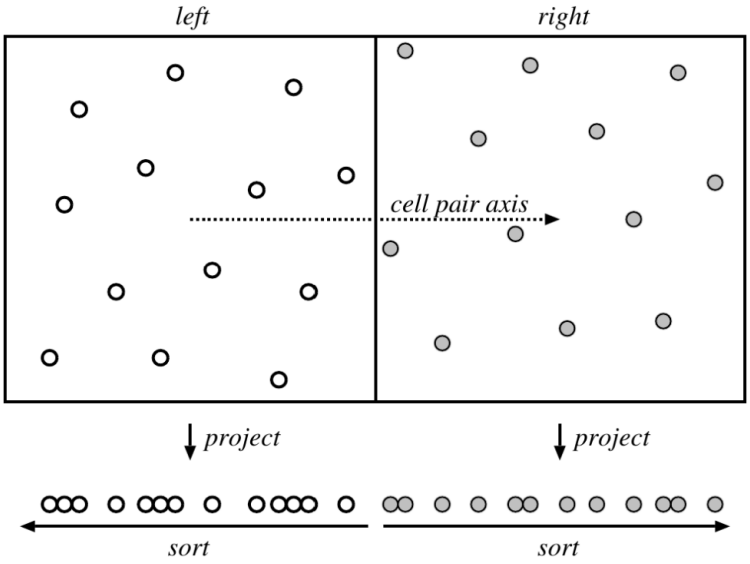


Figure 5.29: Illustration representing the algorithm for computing interactions between particles in two neighboring cells. Particles in the left cell are sorted in descending order, while particles in the right cell are sorted in ascending order along the axis connecting the cell centers. The algorithm efficiently computes interactions between particles in the left cell and those in the right cell that fall within the defined cutoff distance along this axis. Image adapted from [17].

The parallel algorithm divides the simulation space into cells, with each cell containing particles. Interactions are computed between neighboring cell pairs, with each pair assigned to a single thread at a time to avoid conflicts. Particles in each pair of cells are sorted along the axis between the cell centers to reduce unnecessary calculations by only considering particles within the interaction range. Self-interactions within a cell are handled directly with a simple double loop, eliminating the need for a Verlet list in these cases.

Tasks are dynamically distributed across available processors, with each thread selecting a cell pair that is not in use. After computing the interactions for that pair, the cells are marked as available for other threads. This dynamic task allocation ensures efficient load balancing and minimizes idle time, allowing the algorithm to scale well across multiple processors.

The results in [18] demonstrate that pseudo-Verlet lists reduce memory usage relative to traditional Verlet lists, with the effectiveness dependent on system density. The compact structure of the pseudo-Verlet lists enables simulations to scale efficiently with an increasing number of cores. For instance, the largest system simulated with pseudo-Verlet lists comprised 92,224 atoms, utilizing 384 domain-decomposed cells in an 8x8x6 grid configuration and a $2.5\,\mathrm{fs}$ time step.

5.2.12 Coarse-Grained MD with UNRES on Tryton Cluster

In [19], Sieradzan et al. presented the optimization of the UNited RESidue (UNRES) package for CG MD simulations, designed to treat large proteins. The performance tests were conducted on the Tryton Linux *Cluster*, containing Intel Xeon E5

processors (Haswell architecture) with 12 cores and 128/256 GB RAM per server. Parallelization was achieved through a hybrid MPI and OpenMP Open Multi-Processing (OpenMP) model, leveraging both distributed memory and shared memory to optimize scalability. The software was compiled with Intel Parallel Studio XE, and profiling tools such as Intel VTune were used to eliminate bottlenecks in the code.

Interaction lists were used to optimize computational efficiency in CG MD simulations by limiting the number of interactions that need to be calculated. It reduces the number of computations needed by only evaluating interactions between particles (e.g., amino-acid residues) that are within a predefined cut-off distance. This list is constructed using a combination of the Verlet neighbor list and cell index methods, allowing efficient management of particle pairs that must be evaluated during simulations.

The implicit solvent model used for simulations was the Generalized Born Surface Area (GBSA) model. This model was applied during the calculations with the AMBER package, using a 25 Å cut-off for all long-range interactions, including electrostatics. This setup was employed to simulate large protein systems efficiently while accounting for solvent effects implicitly, thus avoiding the explicit modeling of water molecules.

The construction and management of these interaction lists are highly parallelized. The parallelization is performed using both MPI (Message Passing Interface) and OpenMP in a two-grain parallelization scheme as depicted in Figure 5.30.

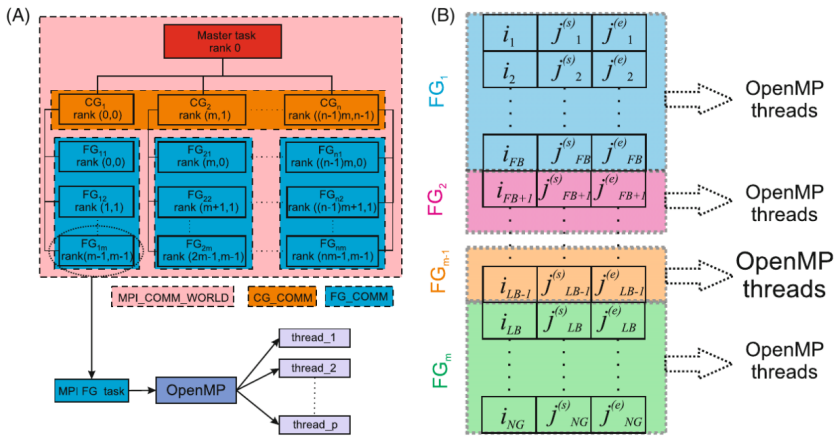


Figure 5.30: Illustration representing UNRES parallelization scheme. Part (A) illustrates the overall structure, highlighting MPI-level parallelization with CG tasks spanning Fine-Grained (FG) tasks, where CG tasks handle MD trajectory or energy evaluations. The master CG process coordinates tasks, and each FG task involves energy and energy-gradient evaluation. Part (B) shows how particle interactions are grouped and assigned to FG tasks to maintain load balance, with further divisions into threads. Image adapted from [19].

This approach divides tasks into CG and FG levels, where CG tasks handle independent MD trajectories, and FG tasks are responsible for computing energy and forces within each trajectory. At the CG level, MPI is used for handling multiple tasks in a multi-trajectory run, where each CG task corresponds to an inde-

pendent MD trajectory. Each CG task is assigned a dedicated MPI process, and synchronization occurs only at specific intervals during replica-exchange or after completing a full run. The FG level involves further division of these CG tasks into smaller FG tasks, which also use MPI to manage communication between processes responsible for evaluating energy and forces in the simulations. This two-level structure improves scalability, allowing multiple MD trajectories to be processed simultaneously across distributed computing environments.

Additionally, the implementation utilizes OpenMP threads for shared-memory parallelism within each FG task intended to minimize communication overhead in the energy and force evaluation steps, as well as when calculating energy-gradient components, for being memory intensive. OpenMP threads operate on a shared memory architecture, with each thread handling its own local copy of the relevant data to avoid synchronization penalties.

The best result using a implicit solvent model was achieved using the 5Y6P protein, which contains 153,243 residues and 2,283,236 atoms. The simulation was performed with a time step of $5.0\,\mathrm{fs}$ for folding simulations.

5.3 Comparative Analysis

This section presents a recapitulation of the results obtained by the HPC MD techniques described in the previous section, as they are organized and displayed on Table 5.4. The data from this table are arranged in columns as follows: (a) a column named *Paper* casting the referenced paper of the original publication; (b) *Contribution* column with a short description of the paper main contribution; (c) *atoms* column with the number of atoms of the simulated molecular system; (d) *cores*, representing the number of CPU, GPU or Application-Specific Integrated Circuit (ASIC) cores used in the referred simulation; (e) *Solvent*, casting the solvation model used in the paper; (f) *NAMD* indicating whether the corresponding technique used the NAMD software as the main MD tool for performing its simulations; (g) *PF* indicating whether the referenced technique were applied for protein folding simulation; and (h) *AG* showing whether the method used some adaptive grid strategy for performing is MD simulation.

Starting with Paper [7], the main contribution refers to a new data layout for boosting GPU acceleration and CPU vectoring in HPC MD simulation. The largest molecular system simulated was composed of 224,000,000 atoms and executing this simulation took 31,043,008 cores while using NAMD. However, in this case, the target simulation was not a protein folding process.

In [8], the main contribution was a three-mode operation, including a CPU+ IntelPhi mode for accelerating the GROMACS execution of MD simulations in the Tianhe-2A supercomputer, simulating a 300,000 molecular system behavior and also achieving a total usage of 2,148 cores.

For Paper [9], its contribution is a software cache method for improving memory bandwidth usage while computing three-body interactions. In this case, the HPC MD method were able to simulate a 275,000,000,000 atom molecular system (the largest in the table) using the MD tool LAMMPS for parallel execution using

Paper	Contribution	#atoms	#cores	Solvent	NAMD	PF	AG
[7]	Data layout	224,000,000	31,043,008	Explicit	Yes	No	No
[8]	CPU+IntelPhi mode	300,000	2,148	Explicit	No	No	No
[9]	Software cache	275,000,000,000	4,194,304	Explicit	No	No	No
[10]	Task mapping	220,000,000	33,768	Explicit	No	No	No
[11]	Real/Reciprocal-space decomposition	1,600,000,000	16,384 nodes	Explicit	No	No	No
[12]	Custom hardware	151,924	33,792	Explicit	No	Yes	No
[13]	Deep learning	359	140 nodes	Explicit	No	Yes	No
[14]	Gaussian acceleration on NAMD	1,912	640	Explicit	Yes	Yes	No
[15]	I/O with RDMA	478	NP	Explicit	Yes	Yes	No
[16]	AR cell list	1,000,000	NP	Implicit	No	No	No
[17]	Pairwise Verlet lists	NP	16	Implicit	No	No	No
[18]	Pseudo-Verlet lists	92,224	16	Implicit	No	No	No
[19]	UNRES interac. lists	2,283,236	48	Implicit	No	Yes	No
Our Work	**Adaptive Grid & NAMD with Atomic Burials**	**5,714**	**256**	**Implicit**	**Yes**	**Yes**	**Yes**

Table 5.4: Comparative table with state-of-the-art contributions in HPC MD. AG means "Adaptive Grid" and NP means "Not Provided".

4,194,304 cores. For this experiment, the simulated process did not carried out protein folding neither involved an adaptive grid solution.

Next, [10] showed as contribution a task mapping heuristic for executing MD simulations on HPC systems, running different experiments on three supercomputers using the LAMMPS MD tool. Again, no protein folding process was simulated nor adaptive grid method was mentioned in this paper. The largest simulated system has 220,000,000 atoms and 33,768 CPU cores were used.

Paper [11] proposed a HPC technique based on a MD Real/Reciprocal-space decomposition for large scale molecular systems, such as the testing system used with 1,600,000,000 atom (the second largest system in the table) that demanded a total of 16,384 computing nodes for executing its MD simulation using the GENESIS MD tool.

The next four papers [12–15] deal with MD simulation of protein folding, although none of them indicated any adaptive grid heuristic in the solutions. Thus, starting with [12], it brought the new version of a well-known custom hardware, special-purpose supercomputer, called Anton, built with a specif purpose of running mainly MD simulations of protein folding. In this paper, Anton's third iteration was able to simulate the folding process of a protein system with 151,924 atoms, using 33,792 ASIC cores.

For the next two papers [13, 14], the main focus was the HPC MD simulation general workflow, which means not only focusing on parallel execution of the MD simulation, but improving the analysis of the obtained partial results. For the paper [13], a Deep learning technique was presented for the analysis of intermediary conformations and using them for orient the protein folding simulation. The main experiment used a 359-atom protein and the MD simulation workflow took 140 nodes of the Summit supercomputer.

In [14] the main contribution was the introduction of a Gaussian acceleration method into the NAMD tool with the intention of reduce the overall execution time of MD simulations of protein folding. For testing this heuristic, a protein with 1,912

atoms and a total of 640 CPU cores were used in the Gordon supercomputer.

Next, the paper [15] built a strategy for performing simultaneous MD simulations and analysis in a HPC environment, proposing analysis of bottlenecks and contentions of using an I/O with a RDMA solution. The largest system used in this scenario was a 478-atom protein and the MD simulations were executed using the NAMD tool. The amount of cores used for executing the MD simulation were not provided in the paper.

These previous approaches primarily focus on MD simulations with explicit solvents, utilizing parallel techniques to reduce execution time and leveraging domain decomposition algorithms (e.g., Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) [9, 34]). Some methods even incorporate hybrid combinations, such as force-domain decomposition in NAMD [7, 104]. In case of simulations with implicit solvents, alternative techniques have been developed [16–19], which also begin with domain decomposition, despite the resulting domains often being non-uniformly populated with atoms [19, 34]. These methods implement additional algorithms to optimize the calculation of interatomic forces to further reduce computational costs.

Paper [16] proposes an Adaptive-Resolution (AR) cell-list algorithm for an efficient way to access potential neighbor particles for computing particle-particle force interactions. This method was implemented using a library called PPM [154], which allowed performing simulations using shared-memory techniques, with the largest simulated system consisting of 1,000,000 particles.

Papers [17] and [18] introduce different variations of the Verlet list algorithm for speeding up the computation of non-bonded force interactions between atoms in MD simulations: Pairwise Verlet list and Pseudo Verlet list. Basically, a Verlet list is a list of interacting atoms for a given domain (space) [18]. Both Pairwise and Pseudo-Verlet lists were implemented using a shared-memory library for parallel MD simulations, called `mdcore`. The MD simulations were executed on one node with 4 quad-core AMD Opteron 8380. The largest simulated system using Pseudo Verlet list consisted of 92,224 atoms, using 384 domain-decomposed cells in a grid configuration of 8x8x6 and time-step of $2.5\,\mathrm{fs}$.

Using CG protein models the time and scale of the MD simulation can be extended considerably [67, 155]. In this context, paper [19] proposes algorithmic improvements for a package called UNRES which performs physics-based CG MD simulations of proteins. Improvements added in [19] to optimize the computations were the use of interaction lists and a selected cut-off distance to a achieve better trade-off between computing cost and accuracy, typical for CG MD simulations with implicit solvents. The largest simulated system with implicit solvent consisted of 153,243 amino-acid residues (2,283,236 atoms in all-atom representation with implicit solvent), for folding 5Y6P using a $5.0\,\mathrm{fs}$ time-step. Simulations were conducted in a Intel Xeon Gold-6148 (2.4 GHz) processor with 4 MPI processes, 12 threads each.

Lastly, we included our work which introduces an adaptive grid technique using Atomic Burial (AB) as contribution (see Part II of this Thesis) for performing HPC MD simulations of protein folding, with the largest system simulated consisting of a protein with 5,714 atoms and the usage of 256 CPU cores.

From Table 5.4, it can be seen that each work tackles a different aspect of the MD simulation (column 2), aiming to accelerate it. It must be noted that, for protein folding simulations (column 7), the number of atoms is much smaller than the classic MD simulations. The exception is [12], which uses custom hardware and seeks the quaternary structure. Among the works that deal with protein folding, our work deals with more atoms than the other three [13–15].

Considering the context brought to light in these previous Sections, to our knowledge, there is no solution with a parallel execution of molecular dynamics simulation which presents an adaptive parallel strategy and that incorporates the calculations of atomic burial, hydrogen bonds and annealing (see Section 4.4).

Part II

Contributions

Chapter 6

Adaptive Patch Grid (APG)

The first contribution of this Thesis is a strategy that allows NAMD simulations to adapt the patch grid created (see Section 4.3.2) to the shape of the protein throughout the execution, producing a more balanced geometric decomposition among the processing elements, which is more appropriate for parallel execution.

In Section 6.1, we explain the challenge that motivates the APG strategy. Section 6.2 then describes how NAMD's workflow for molecular dynamics simulations was adapted to incorporate APG, followed by Section 6.3, which provides an overview of the strategy.

Section 6.4 presents the results obtained in molecular dynamics simulations of protein folding using NAMD with the APG strategy. Finally, Section 6.5 concludes with a review of the main points in the chapter.

To our knowledge, there is no solution in the literature for varying dynamically the shape of the simulation box in parallel protein folding simulations. All results of this section were published in [156].

6.1 Challenge: HPC MD Simulation of PF using Static Domain Decomposition

In order to reduce the execution time of a simulation with typical *workflow*, we chose the molecular dynamics simulation software called NAMD (Section 4.3), which implements a parallel version of the general molecular dynamics simulation algorithm (Algorithm 1). NAMD implements a hybrid parallel execution strategy that combines the force decomposition and domain decomposition methods,

For this strategy, domain refers to the 3D geometric space surrounding simulation system, both solvent and solute, e.g. water and the target protein, respectively. Initially, the dimensions of the domain are calculated based on the distribution of all the atoms in system. Then, the domain decomposition method is executed to divide the simulation space into cells containing the system's atoms, and these cells will be assigned to processors, maintaining this allocation until the end of the simulation.

Figure 6.1 shows a scenario identified in this Thesis. Initially, the globular protein structure has an extended shape, 1D geometric configuration. The simulation

space is defined for this 1D configuration, followed by the domain decomposition calculation, which generates the necessary cells containing the atoms that were distributed. Then, these cells are assigned to the processors available to execute the *job* on the HPC system (Figure 6.1a). Note that, during the molecular dynamics simulation, the 3D configuration of the protein will change, moving from the extended 1D shape to a more compact, in our case, a globular shape, as shown in Figures 6.1b and 6.1c.

This means that the atoms of the 1D structure migrate from their original position (and the cells they were originally in) to a 3D configuration concentrated around a geometric center, leaving the processors that were allocated to those cells with a smaller number of atoms and, also, communicating with other processors during the molecular dynamics simulation.

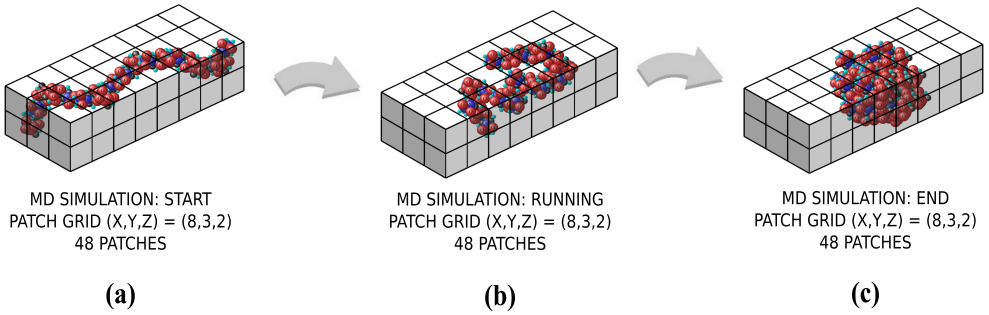


Figure 6.1: Illustration of a domain decomposition performed for a molecular dynamics simulation of a globular protein fold: (a) the simulation domain is divided into blocks, called cells, containing protein atoms; (b) the protein begins to fold and the atoms are migrating from their original positions - some cells are already empty and their processors are idle; (c) the globular protein folds into a stable configuration and the atoms are concentrated in "central" cells, leaving more "peripheral" processors in an idle state.

Thus, considering the previous scenario, a challenge identified in this Thesis was how to act on this distribution of cells, generated by domain decomposition, among the processors available for the *job* responsible for executing the simulation of protein folding on a supercomputer. Section 6.2 presents an overview of our adaptive domain decomposition approach to solve this challenge and Section 6.3 explains the design of our adaptive patch grid strategy.

6.2 Adaptive Domain Decomposition Computation

The method proposed to address the challenge presented in Section 6.1 was the adaptive parallel strategy shown in Figure 6.3b. In this technique, we added: (a) a new *loop* starting at the action *Patch Grid Generation*; and (b) at the end of this *loop*, a new step for writing the *Partial Folding Trajectory File* and a new action, named *Update Simulation Configuration File*.

For the *Patch Grid Generation* action, the modification added was reinserting the *partial* 3D configuration of the protein after a predefined number of simulation

steps. The simulation space will be updated as the geometric configuration of the protein has changed. From this *updated* partial 3D configuration, the subsequent domain decomposition will also recalculate the number of cells (*patched*) needed and distribute them among the processors, as shown in Figures 6.2a, 6.2b and 6.2c.

For the step at the *bottom* of the new *loop*, a *Partial Folding Trajectory File*, containing the changes in geometric configuration that have occurred up to that point, is used as an updated *Structure File* and is provided as re-input for the *Patch Grid Generation* action. Additionally, the *Simulation Configuration File* is also updated: (i) with a new number of steps to be executed in the next iteration of the simulation *workflow loop*; and (ii) with an indicator for the updated 3D configuration that is used by the *Simulation Domain Calculation* action, as described previously.

The number of steps required to complete this *loop* of the adaptive *workflow* is determined empirically for the first complete simulation run of protein folding. However, once defined, the same number of steps can be used in future simulations of folding for proteins from the same family.

After the end of the added *loop*, a new action was included to perform the *Merge of the Partial Trajectory Files*, generated during the execution of the adaptive patch grid strategy, into a single file containing the complete trajectory that will be presented in the output, as expected in a typical molecular dynamics simulation.

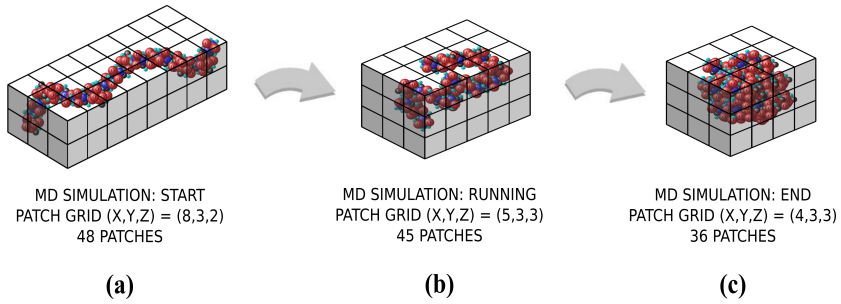


Figure 6.2: Illustration of our adaptive patch grid strategy for a NAMD simulation of protein folding using implicit solvent.

In our strategy, the dimensions of the *patch grid* do not change during the execution of the *loop* of numerical integration, however, they are updated whenever NAMD execution is stopped and restarted [157, 158]. Therefore, it is possible to divide the number of iterations of this *loop* into different phases by stopping the current simulation in a defined iteration and restarting the next phase in the subsequent *loop* iteration. For instance, if the original simulation has 100,000 iterations and 4 domain generation phases, a new patch grid will be generated for every 25,000 iterations and each one of them will be referenced in the corresponding phase of the molecular dynamics simulation execution. Thus, taking advantage of the *patch grid* construction process performed by NAMD, it is possible to redistribute the simulation domains (*patches*) between the available processors through the inclusion of our adaptive *loop* in the original *workflow*.

It is important to note that while the patch grid dynamically adapts during execution, the overall size of the simulation box enclosing the protein remains fixed

throughout the simulation. This design ensures consistency in spatial resolution and avoids introducing variability unrelated to the domain decomposition strategy.

It is important to note that while the patch grid dynamically adapts during execution, the overall size of the simulation box enclosing the protein remains fixed throughout the simulation. This design ensures consistency in spatial resolution and avoids introducing variability unrelated to the domain decomposition strategy.

6.3 Design of the APG strategy

Figure 6.3 shows the steps followed in a simulation workflow of protein folding, where (a) is a typical simulation workflow and, (b) our main contribution, an adaptive simulation workflow. The typical workflow of a simulation follows Algorithm 1.

In a typical simulation workflow (Figure 6.3a), NAMD receives three files as input (Configuration, Structure and Force Field). The simulation domain is generated using these files and the patch grid is created. Then, for the number of time steps configured, the simulation loop is executed, where the traditional forces are computed for each atom or set of atoms, which will cause a modification in the atoms' positions, generating an updated structure that is written in the trajectory file. With the new atoms' positions, the traditional forces are computed and so on, until the number of *time steps* is attained.

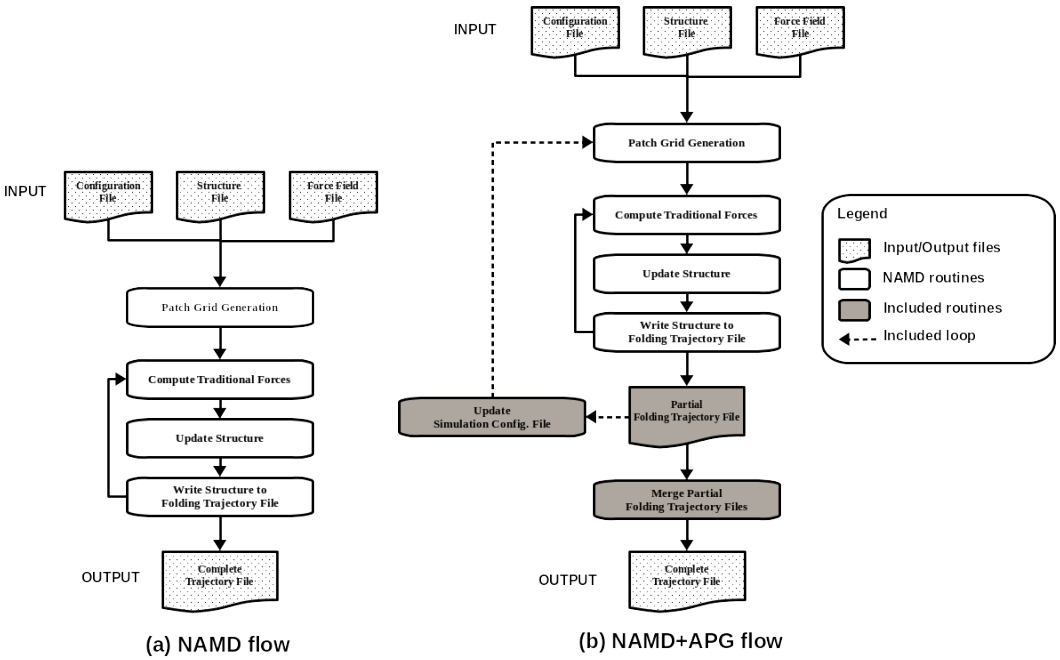


Figure 6.3: NAMD and NAMD+APG execution flows.

In our APG strategy (Figure 6.3b), at the beginning, the same three files are read and used to generate the simulation domain and the initial patch grid. Using NAMD's restart facility, we divide the number of NAMD iterations into different phases, stop the execution of a phase in a specified iteration and automatically

restart the next phase in the next iteration of the loop. This technique is able to use NAMD to redistribute the simulation domains (*patches*) among the available processors, creating *patch grids* and adapting to the structural protein changes that occurred during simulation. In this case, the simulation is divided into a set of domain generation phases, where each phase corresponds to a part of the whole original simulation. This corresponds to the included loop in Figure 6.3b.

Periodic boundary conditions (PBCs) were not applied in the simulations, as they are not required in the context of implicit solvation. Without explicit solvent molecules, the simulation box functions as a closed system, and the focus remains solely on the internal conformational dynamics of the protein. This modeling choice reduces complexity and aligns with the goals of the APG strategy.

Section 6.4 describes the experiments we performed using NAMD with and without our APG strategy and present the results we obtained.

6.4 Experimental Results

6.4.1 Description of the Computing Environment

Our tests were ran in one supercomputer - MareNostrum 4 (MN4) - and one HPC cluster - Nord III (Nord) - both installed at the Barcelona Supercomputing Center (BSC) [144, 159]. MN4 supercomputer was the 174th fastest supercomputer in the world according to the November 2024 Top500 list and was described in Section 5.1.4. MN4 is a *Cluster* consisting of 48 racks housing a total of 3,456 compute nodes. Each node is equipped with two Intel Xeon Platinum processors, providing 48 cores per node. This setup delivers a combined total of 165,888 CPU cores and 96 GB of main memory per node [144].

Nord [159] is one rack of the Marenostrum 3 supercomputer. It is CPU-based, where each node is composed of two CPUs Intel E5-2670 (16 cores). There are 84 nodes (1,344 cores) interconnected through InfiniBand Mellanox, with storage capacity of 15 PB and 2.6 TB of RAM.

The submission of jobs in MN4 was performed through the scheduling software called SLURM [144]. At Nord, this job submission is handled by the LSF [159] software. Among the different execution queues available via SLURM and LSF, the following were assigned for the tests of this Thesis: (a) *bsc_sc* (in SLURM): with a maximum number of 50 nodes, 2400 cores and execution time limit of 48 hours for each job; and (b) *bsc_cs* (in LSF), but with a maximum of 16 nodes available, that is, 256 cores and an execution time limit of 48 hours per job.

NAMD (version 2.11) was compiled with support for C and C++ languages, MPI-3.1 standard, using the Intel compiler, available to users of computational environments through the *intel/2017.4* modules, *impi/2017.4* in MN4 and the *impi* module in Nord.

These two computing systems were selected due to their suitability for parallel experiments in protein folding simulations and their availability through research collaboration with the Barcelona Supercomputing Center (BSC). MareNostrum 4, with its large number of cores and high-speed interconnect, enabled large-scale

executions and strong scalability studies. In contrast, Nord provided a more constrained and complementary environment, ideal for developing and validating the proposed strategies under limited-resource conditions. This combination allowed for a thorough evaluation of performance and adaptability across different HPC scenarios.

6.4.2 Description of Proteins

The proteins used during the evaluation of the simulation test environment with NAMD are listed in Table 6.1: 1ENH, 1IFR, 1OZ9 and 4LNZ. These are globular proteins with functions related to the immunological, motor and DNA association, with data available in the Protein Data Bank (PDB) [160–163].

Protein	# Atoms (PDB)	# Atoms (Hydrogenated)	Ligands	Disordered Regions
1ENH	466	947	None	1-4, 52-54
1IFR	878	1,746	SO4	103-105
1OZ9	1,151	2,346	CA, GOL	45-49
4LNZ	2,840	5,714	ZN, DTT	187-192

Table 6.1: Test proteins used during the evaluation of the molecular dynamics simulation environment with NAMD. For each protein, the table shows: (i) the number of atoms from the original PDB file [160–163]; (ii) the number of atoms after adding hydrogens during preprocessing; (iii) ligands identified in the original structure; (iv) and disordered regions, i.e., unresolved residues often linked to flexible loops or terminal segments.

The number of atoms of each protein in Table 6.1 was obtained as follows: a) column "Original PDB" presents the total number of atoms represented in the original PDB file, obtained directly from the PDB website [160–163]; b) the column "Updated PDB: with hydrogens" shows the total number of atoms represented in the PDB file generated while processing the force field input files (see Section 6.3).

For this Thesis, the force field model chosen was the CHARMM [164], according to the basic configuration suggested in [25]. Two CHARMM files are handled before running NAMD: a topology file and a force field parameter file. The topology file contains, among other data, the hydrogen atoms empirically defined for the CHARMM model, which are added to the generated structure file (PSF extension, *Protein Structure File*) and to the "updated PDB" file, both handled by NAMD when running the molecular dynamics simulation. As for the force field parameter file name, it is provided inside the molecular dynamics simulation configuration file, which is described in the following Section.

All proteins tested in this work were modeled without explicit ions or ligands. Disulfide bridges were also not enforced. These modeling decisions are consistent with the use of implicit solvation and allow for an isolated evaluation of folding behavior under the influence of the APG strategy alone, following the rationale discussed in Section 4.1.2.

Although all simulations were carried out with proteins modeled without explicit ions or ligands, additional structural details about the original PDB entries are included in Table 6.1. Specifically, the table reports which ligands were identified in the crystallographic structures and which residues were unresolved due to structural disorder. These characteristics, although not used in the simulations, are relevant for contextualizing each protein's experimental structure and understanding potential influences on their folding behavior in vivo or under different modeling conditions [51, 53].

6.4.3 NAMD Configuration File

Section 4.3 showed that NAMD implements a parallel version of general molecular dynamics algorithm (Algorithm 1) and that, it is capable of running molecular dynamics simulations of atomic systems containing millions of atoms [7, 114]. And, as stated in [25], it is through a configuration file that NAMD obtains all the data that defines the behavior of the molecular dynamics simulation, with the exception only of those related to parallel execution, as they are platform dependent.

In this context, the objective of this section is to present a basic configuration of NAMD that allows the execution of molecular dynamics simulations of the folding process for proteins with a number of atoms bigger than 5,000 atoms. Furthermore, in order to define the platform-dependent data required for the parallel execution of these molecular dynamics simulations, Section 6.4.4 will cover a evaluation of NAMD's behavior when running molecular dynamics simulations on MN4 and Nord supercomputers using this basic configuration as an input for each simulation, both with a default configuration or with our APG strategy. Thus, from this point on, we will refer as NCF to the basic NAMD Configuration File containing either the configuration parameters for running NAMD without any contribution of this Thesis (NAMD default) or containing additional parameters for running NAMD with our APG strategy (NAMD+APG).

NAMD provides an extensive list of parameters to create the molecular dynamics simulation that will be performed, which are detailed in the official NAMD's tutorial [25]. For experimental tests of this Thesis, the NCF adopted was obtained by changing parameter values available in NAMD. Appendix B presents an example of configuration with a typical parameters setup used in our experiments. The input parameters for NAMD to execute the general molecular dynamics simulation algorithm, akin to those in line 1 of Algorithm 1, are specified in an NCF as follows: a) *molecular system name*: this parameter serves as an identifier referenced throughout the simulation, such as when naming and writing to output files. In our case, the chosen protein's name was used as the identifier for this parameter in each simulation; b) *coordinates* and *structure* parameters indicating the names of input files containing data related to positions of the atoms and geometric configurations (structure) of the system, respectively; c) the name of the parameter file containing data for the chosen force field is entered in the *parameters* field; d) for the solvation model, it was adopted an implicit environment provided by NAMD's implementation of GBIS (see Section 4.1.2); this choice was defined by setting the the parameter *gbis* with the value "*yes*" in the NCF; e) the time interval which will

be simulated in each step of the numerical integration *loop* of the Algorithm 1 was defined in the *timestep* field with the value 1 fs; and, f) the *run* parameter was used to define the number of iterations used in numerical integration *loop*.

Finally, our NAMD basic configuration file (NCF) was also used to establish the values of parameters related to each restarting process of the molecular dynamics simulation which were used in tests with our adaptive strategy, namely: a) to restart a NAMD simulation, parameters *bincoordinates*, *binvelocities* and *extendedSystem* fields correspond to the names of the files that contain data related to the last recording point of the atomic system state and which allows that the molecular dynamics simulation execution starts from that point; b) the *firsttimestep* informs NAMD of the initial iteration from which the numerical integration *loop* must restart; and c) the *numsteps* parameter informs NAMD the number of iterations that must be executed by the numerical integration *loop* from that point forward.

6.4.4 Evaluation Tests of NAMD's Default Patch Grid

This section shows the tests that were carried out in order to evaluate a NAMD configuration that is suitable for running molecular dynamics simulations on Marenostrum 4 (MN4) and Nord supercomputers. These tests were called *evaluation tests* and are comprised of (i) parallel executions of NAMD, using a prepared NCF for each molecular dynamics simulation (Section 6.4.3) and (ii) analysis of NAMD's output, considering the domain decomposition and the execution times recorded during each simulation.

For each parallel execution of NAMD on MN4 and Nord, it was necessary to create and submit a job file to an execution queue on both supercomputers. Each job file contained, mainly, the number of cores to be allocated for that specific simulation, the chosen parallelization method (message exchange using MPI) and the name of NCF to be read by NAMD on each execution.

Tests on Marenostrum 4: Protein 1ENH

For this evaluation test, the atomic system chosen was the 1ENH protein (947 atoms, Section 6.4.3), initially extended (1D). We ran a *default NAMD* configuration with a *fixed patch grid* and a 20,000,000 iteration loop with a *time step* of 1 fs, which means simulating about 20 ns of the folding process. The results of these simulations are shown in Table 6.2. Six jobs of molecular dynamics simulations were executed. In the first job, 48 cores of CPU were allocated, corresponding to one MN4 computational node. Then, for each following job, the addition of a computational node was made, corresponding to the numbers of cores 96, 144, 192, 240 and 288 in the "#Cores" column.

Note that the domain decomposition performed at the beginning of the NAMD execution defined the same domain configuration (*patch grid*), consequently, same number of cells (patches): 7 x 3 x 1 and 21 patches, respectively, from start to finish of each simulation.

#Cores	#Nodes	Default NAMD - 1ENH				
		#Iterations	Runtime		Domain Decomposition	
		(million)	(seconds)	(d:h:m:s)	*Patch Grid*	#Patches
48	1	20M	130,834	1d 12h 20m 34s	7 x 3 x 1	21
96	2	20M	159,788	1d 20h 23m 08s	7 x 3 x 1	21
144	3	20M	158,835	1d 20h 07m 15s	7 x 3 x 1	21
192	4	20M	127,883	1d 11h 31m 23s	7 x 3 x 1	21
240	5	20M	99,756	1d 03h 42m 36s	7 x 3 x 1	21
288	6	20M	164,580	1d 21h 43m 00s	7 x 3 x 1	21

Table 6.2: Evaluation test: Molecular dynamics simulation on MN4 using NAMD with default patch grid parameters and protein 1ENH (947 atoms).

This is a *fixed* assignment of patches to the available cores, as this assignment is also carried out at the beginning of the simulation and remains unchanged throughout the execution of job. One consequence is that the number of patches generated is lower than the number of allocated processors, suggesting that there are potentially idle processors during execution. For example, in the first job with 48 cores, a patch was assigned to each processor, 27 cores will be idle during the simulation.

Finally, it is important to highlight the execution time values recorded in each job, which are not related to the number of nodes/cores (Table 6.2). This means that augmenting the number of nodes/cores does not reduce the execution time, making parallelism ineffective. For example, there was an increase in the execution time of the first job, with 48 cores for the second job. However, from the second to the third job the execution time remained almost the same. Then, the time decreased from the third to the fourth and the fifth job to, again on the same scale, increase in the sixth job. As a conclusion, we can say that, if the number of cores between jobs increased, there was no (significant) reduction in execution time, and in some jobs there was an increase in execution time. Interestingly, the job executed with 240 cores resulted in the lowest recorded execution time (99,756 seconds). Since the number of patches (21) remained unchanged, the amount of parallelism did not increase. This result likely reflects a more favorable allocation of system resources during that specific run - such as lower network contention or more efficient node scheduling - rather than a direct consequence of utilizing more cores. Therefore, this improvement should be interpreted as an effect of system-level variability rather than increased computational efficiency.

Tests on Marenostrum 4: Protein 1IFR

Given the previous results, the next step was to carry out a new evaluation test using a protein with more atoms, observing the decomposition of NAMD domains and execution times. The chosen protein was 1IFR with 1,746 atoms. The test results of this assessment are shown in Table 6.3.

#Cores	#Nodes	Default NAMD - 1IFR				
		#Iterations (million)	Runtime		Domain Decomposition	
			(seconds)	(d:h:m:s)	*Patch Grid*	#Patches
48	1	20M	152.668	1d 18h 24m 28s	7 x 6 x 1	42
96	2	20M	141.290	1d 15h 14m 50s	7 x 6 x 1	42
144	3	20M	150.781	1d 17h 53m 01s	7 x 6 x 1	42
192	4	20M	147.389	1h 16h 56m 29s	7 x 6 x 1	42
240	5	20M	122.832	1d 10h 07m 12s	7 x 6 x 1	42
288	6	20M	168.994	1d 22h 56m 34s	7 x 6 x 1	42

Table 6.3: Evaluation test: Molecular dynamics simulation on MN4 using NAMD with default patch grid parameters and protein 1IFR (1,746 atoms).

Similarly, for these tests, the number of cores defined was different for each simulation, ranging from 48 to 288 cores. In each job, the patch grid and the number of patches generated by NAMD were the same, 7 x 6 x 1 and 42, respectively.

It can be noted, again, that there is no relation between the execution time and the number of cores/jobs used in the experiments. In domain decomposition, these results draw attention once again to the fixed allocation of the patch grid and the number of patches generated (42), which is lower than the number of processors allocated in each job. For example, in the first job, NAMD allocates a patch to each core, thus 6 cores are potentially idle during the simulation run.

Tests on Marenostrum 4: Protein 1OZ9

A new evaluation of NAMD was carried out using a system with more atoms: protein 1OZ9 with 2,346 atoms. Table 6.4 shows the results obtained.

#Cores	#Nodes	Default NAMD - 1OZ9				
		#Iterations (million)	Runtime		Domain Decomposition	
			(seconds)	(d:h:m:s)	*Patch Grid*	#Patches
48	1	20M	145,812	1d 16h 30m 12s	19 x 3 x 1	57
96	2	20M	114,051	1d 07h 40m 51s	19 x 3 x 1	57
144	3	20M	83,213	23h 06m 53s	19 x 3 x 1	57
192	4	20M	98,462	1d 03h 21m 02s	19 x 3 x 1	57
240	5	20M	103,016	1d 04h 36m 56s	19 x 3 x 1	57
288	6	20M	170,608	1d 23h 23m 28s	19 x 3 x 1	57

Table 6.4: Evaluation test: Molecular dynamics simulation on MN4 using NAMD with default patch grid parameters and protein 1OZ9 (2,346 atoms).

For the jobs of this test, the patch grid generated was 19 x 3 x 1 with 57 patches to be distributed among the cores allocated. Note that, dividing the 2,346 atoms between the 57 patches generated, it obtains 48 patches with 41 atoms and 9 patches with 42 atoms each. In the first job, the number of patches is enough to distribute

among the 48 cores. However, from the second job onwards, the number of cores allocated (96) is greater than the number of patches generated by NAMD, potentially leaving processing cores in idle state. Table 6.4 also shows that the execution times obtained are still not related to the number of nodes/cores.

Tests on Nord: Protein 1OZ9

The next step was to evaluate the behavior of NAMD's simulation using the same 1OZ9 protein, a system with the largest number of atoms so far, the same NCF with the predefined domain decomposition (*Default*), however reducing the number of cores allocated to each job. For this, the tests were carried out on the Nord cluster [159], where each computational node has 16 cores, instead of the 48 available in the MN4. Table 6.5 shows the results obtained in this evaluation test.

#Cores	#Nodes	Default NAMD - 1OZ9				
		#Iterations (million)	Runtime		Domain Decomposition	
			(seconds)	(d:h:m:s)	*Patch Grid*	#Patches
16	1	20M	127,221	1d 11h 20m 21s	19 x 3 x 1	57
32	2	20M	128,099	1d 11h 34m 59s	19 x 3 x 1	57
48	3	20M	784.23	21h 47m 03s	19 x 3 x 1	57
64	4	20M	93,744	1d 02h 02m 24s	19 x 3 x 1	57
80	5	20M	103,653	1d 04h 47m 33s	19 x 3 x 1	57
96	6	20M	65,850	17h 17m 30s	19 x 3 x 1	57
112	7	20M	169,987	1d 23h 13m 07s	19 x 3 x 1	57
128	8	20M	60,980	16h 56m 20s	19 x 3 x 1	57

Table 6.5: Evaluation test: Molecular dynamics simulation on Nord using NAMD with default patch grid parameters and protein 1OZ9 (2,346 atoms).

Eight jobs were executed with parallel execution using 16, 32, 48, 64, 80, 96, 112 and 128 CPU cores. Again, the patch grid and the number of patches generated by NAMD, for the input protein 1OZ9, were equal to 19 x 3 x 1 and 57, respectively. Note that the execution times obtained did not decrease with the increase in the number of cores. For example, the execution times obtained with 64 cores was greater (93,744 seconds) than that obtained with 48 cores. The same was identified for 96 and 112 cores. Additionally, note that the execution times with 96 and 128 cores were the lowest among the jobs, however, this was not expected considering that the number of patches is fixed and smaller than the number of cores (57), that is, the lowest execution times were obtained with jobs with idle cores (without patches) associated.

Thus, in this test, even reducing the number of cores per computational node and obtaining lower execution times than in previous evaluations (jobs 3, 6 and 8), it was not possible to identify a coherent behavior of NAMD regarding domain decomposition, the number of patches and execution times.

6.4.5 Evaluation Test with Scaled Patch Grid

At this point, in order to better understand the relation between the number of cores and the execution time, we increased the number of patches generated by NAMD and evaluated the execution behavior with more patches to distribute among the cores. This was done by activating the *twoAwayX*, *twoAwayY* and *twoAwayZ* parameters in the NCF. In accordance with the recommendation contained in [157, 158, 165], attributing the value "*yes*" to these parameters will cause NAMD to approximately double the number of patches generated in each of the dimensions of the patch grid.

Three NAMD simulations were performed to evaluate this configuration, one for each of the 1ENH, 1IFR, and 1OZ9 proteins. For each simulation, 48 cores were allocated. None of the three simulations finished within the time slot of 48 hours reserved for each job, that is, the job was terminated, without the simulation having been completed. To investigate this, the domain decomposition was analyzed in each case. Table 6.6 shows the values obtained with the standard domain decomposition and the one generated by NAMD after activating the *twoAway* parameters.

Protein		Domain Decomposition			
		Default		*twoAway**	
Name	#Atoms	#*Patches*	#*Patches*($\frac{\#Atoms}{Patch}$)	#*Patches*	#*Patches*($\frac{\#Atoms}{Patch}$)
1ENH	947	21	19(45) + 2(47)	600	253(1) + 347(2)
1IFR	1,746	42	18(41) + 24(42)	1,092	438(1) + 654(2)
1OZ9	2,346	57	32(57) + 9(58)	2,500	2,346(1) + 154(0)

Table 6.6: Evaluation test: NAMD's generated patches on MN4 using *twoAway** parameter. The *twoAway** column shows the results obtained by activating the parameters *twoAwayX*, *twoAwayY* and *twoAwayZ*. The *Default* column shows the results of running without these parameters (they are disabled by default).

For the three proteins in Table 6.6, notice that: a) the data in the *Default* column shows the number of patches generated with NAMD's default domain decomposition and adopted in the first three evaluations so far, as well as an illustration of the distribution of atoms for each patch; b) the data in the *twoAway* column shows the number of patches after activating the parameters *twoAwayX*, *twoAwayY* and *twoAwayZ*; also illustrating a distribution of atoms for each patch.

In this case, based on the data in the *twoAway* column, two potential scenarios were identified involving the distribution of atoms and the cost of communication. In the first, even though NAMD generated a greater number of patches (600, 1,092 and 2,500) than in the first three evaluations, the number of atoms of each protein is very close to the number of patches generated for each simulation, which creates an inefficient distribution of atoms, as shown in the column #*Patches*($\frac{\#Atoms}{Patch}$), where the maximum number of atoms assigned to each core is equal to 2. It is evident that running NAMD with this configuration has low utilization of MN4 resources, that is, high number of idle cores. And, in this context, for the second scenario, it

was observed that there is also the possibility that CPU cores do not have atoms allocated, as is the case with the 1OZ9 protein (last column and last line of the Table 6.6); with 154 patches generated and all of them are *empty*, with no atoms allocated.

Again, it was identified in the same scale that, even activating the *twoAway* parameters in the NCF, the execution presents similar behavior to previous evaluations: runtime unpredictability and idle cores during simulation.

6.4.6 Evaluation Test with Manual Restart

For this evaluation, we decided to proceed using Nord as HPC platform, since it has less CPU cores per node and the results of Section 6.4.4 showed the lowest execution times. However, we increased the number of atoms by introducing the larger, named 4LNZ with 5,714 atoms (Table 6.1).

Initially, most of the executions with 20,000,000 iterations took more than 2 days in Nord and, since the maximum execution time allowed by the job scheduler was 48 hours, some simulations were interrupted. Thus, we decided to set NAMD's *restart* parameter as "true", and manually restart the simulation from the last point in time. The results of the corresponding tests are shown in Table 6.7.

#C	#N	NAMD						
		Phase 1			Phase 2 (*Restart*)			Exec. Time Total (MD) (d:h:m:s)
		#Iter.	Time (d:h:m:s)	*Patch Grid*	#Iter.	Time (d:h:m:s)	*Patch Grid*	
16	1	16,160,000	1d 23h 39m 53s	20 x 7 x 1	3,840,000	12h 08m 53s	7 x 4 x 5	2d 11h 48m 46s
32	2	15,520,000	1d 23h 43m 00s	20 x 7 x 1	4,480,000	11h 43m 16s	7 x 4 x 5	2d 11h 26m 16s
48	3	20,000,000	1d 23h 13m 34s	20 x 7 x 1	N.A.			1d 23h 13m 34s
64	4	17,760,000	1d 23h 57m 31s	20 x 7 x 1	2,240,000	02h 55m 32s	7 x 4 x 5	2d 02h 53m 04s
80	5	16,960,000	1d 23h 32m 54s	20 x 7 x 1	3,040,000	07h 55m 06s	10 x 2 x 7	2d 07h 28m 00s
96	6	17,920,000	1d 23h 36m 36s	20 x 7 x 1	2,080,000	5h 20m 58s	7 x 4 x 5	2d 04h 57m 34s
112	7	20,000,000	1d 23h 03m 05s	20 x 7 x 1	N.A.			1d 23h 03m 05s
128	8	16,800,000	1d 23h 40m 00s	20 x 7 x 1	3,200,000	10h 06m 18s	7 x 4 x 5	2d 09h 46m 18s

Table 6.7: Evaluation test: the simulation ran in two phases. In Phase 1, NAMD runs up to the maximum execution time set by the scheduler. If the simulation completes during Phase 1, Phase 2 is unnecessary, marked as N.A. (Not Applicable). If the simulation does not finish in Phase 1, then Phase 2 resumes NAMD from the last saved *timestep* to complete the remaining simulation steps. The atomic system used in the simulation contains 5,714 atoms, corresponding to the Extended 4LNZ protein.

In this evaluation, NAMD generated the patch grid configuration 20 x 7 x 1, a total of 140 patches to distribute the 5,714 atoms of the system. In Table 6.7, only two jobs (3 and 7), had their molecular dynamics simulations completed within the reserved runtime for each job (48 hours); for the remaining six jobs (1, 2, 4, 5, 6 and 8), the molecular dynamics simulation was interrupted, when the execution of job has reached the maximum execution time set by the SLURM scheduler. In the column *#Iter.*, the number of iterations that were performed by the numerical

integration *loop* of the molecular dynamics simulation, recorded in the *log* file. The results obtained up to that point of termination of jobs were annotated under the column named *Phase 1*. Due to the interruption of the molecular dynamics simulation, six jobs were restarted to continue the simulations that were incomplete, starting the numerical integration *loop* from the next iteration to the last recorded in the *#Iter.* column of *Phase 1* (for example, *16,160,000* iterations of job 1). The results obtained with the restart tests were recorded under the column *Phase 2 (Restart)* call. In the *#Iter.* and *Time* columns for *Phase 2*, the total iterations completed by the numerical integration *loop* and the execution times for the molecular dynamics simulations are shown. Since the simulations for jobs 3 and 7 concluded in *Phase 1*, no data was recorded for these jobs in *Phase 2*, and they are labeled as "N.A.", indicating "Not Applicable".

Note that in the *Patch Grid* column of *Phase 2*, the values obtained were different from those of *Phase 1*, identifying a new scenario in the execution flow of NAMD: during a simulation starting from a initial extended 1D configuration, when NAMD is restarted, a new domain decomposition is performed, generating a new patch grid configuration relative to the current geometric configuration of the protein. This means that by stopping and restarting a molecular dynamics simulation of NAMD, it is possible to change the generated *fixed patch grid*, in order to obtain a new patch grid with different dimensions and, potentially, obtain a new set of allocated cores, both *adapted* to the current 3D configuration of the atomic system.

Realizing that these changes in the patch grid, with each restart of the molecular dynamics simulation, was the key point in developing our adaptive solution. This allowed the domain decomposition of the parallel algorithm implemented by NAMD to be used flexibly throughout the molecular dynamics simulation, adapting to expected geometric transformations in the process of protein folding. This change in the simulation execution workflow has been activated in the NCF, via the reset parameters preset in NAMD (Section 6.4.3).

6.4.7 NAMD Test with Adaptive Patch Grid

This section presents the results of NAMD with our Adaptive Patch Grid (APG) strategy in the Nord cluster. We used the largest protein (4LNZ, 5,714 atoms) in Table 6.1 and split the total number of iterations (20 million) into 4 phases of 5 million iterations. The results are shown in Table 6.8.

#C	#N	#It.	NAMD with APG - 4LNZ								
			Phase 1		Phase 2 (Restart)		Phase 3 (Restart)		Phase 4 (Restart)		Run Time
			Time (sec.)	*Patch Grid*	Time (sec.)	*Patch Grid*	Time (sec.)	*Patch Grid*	Time (sec.)	*Patch Grid*	Total (MD) (d:h:m:s)
16	1	5M	41,688	20 x 7 x 1	52,224	10 x 2 x 7	52,654	10 x 2 x 7	55,022	7 x 2 x 10	2d 07h 59m 02s
32	2	5M	24,858	20 x 7 x 1	30,434	10 x 2 x 7	34,867	7 x 4 x 5	48,314	5 x 2 x 14	1d 14h 27m 53s
64	4	5M	16,787	20 x 7 x 1	26,306	14 x 5 x 2	43,755	7 x 4 x 5	34,095	7 x 4 x 5	1d 09h 35m 43s
128	8	5M	18,556	20 x 7 x 1	39,083	10 x 2 x 7	33,427	7 x 5 x 4	28,545	7 x 5 x 4	1d 09h 13m 31s
256	16	5M	23,856	20 x 7 x 1	25,123	14 x 2 x 5	32,312	7 x 4 x 5	50,824	7 x 5 x 4	1d 12h 41m 55s

Table 6.8: NAMD execution with APG. The total number of iterations per row is 20M.

NAMD+APG simulations were ran for 1, 2, 4, 8 and 16 nodes (16, 32, 64, 128 and 256 cores respectively). Please notice that, in Table 6.8, when the phase changes from 1 to 2, 2 to 3 and 3 to 4, the patch grid is adapted to the current shape of the protein in the simulation box. In addition, the runtime is now related to the number of cores, which decreased from 2 days and almost 8 hours (16 cores) to 1 day and about 9 hours (128 cores).

The execution time increases when 256 cores are used. This happens because there is not enough parallelism to surpass the amount of communication at the end of each iteration. Therefore, in Nord, using 4LNZ in simulation with 20 million iterations, the best choice would be 8 nodes (128 cores).

For comparing the execution times between simulations without APG, which means using a *fixed patch grid* (default), and with our APG, the number of iterations was reduced to 15 million. This was necessary because some simulations without APG did not end in less than 48 hours when 20 million iterations were applied, and the job was killed. Table 6.9 and Figure 6.4 show the execution times without and with APG for protein 4LNZ. It should be noted that, for all test cases, the execution time was reduced in at least 1 hour when APG was used. Also, the best execution times were obtained for 16 nodes (256 cores).

#C	#N	#It.	NAMD with and without APG - 4LNZ							
			Default Strategy			APG Strategy				
			Execution time		Patch Grid	Execution time		Patch Grid		
			seconds	(h:m:s)		seconds	(h:m:s)	Phase:1	Phase:2	Phase:3
16	1	15M	123,515	34h 18m 35s	47x3x1	119,169	33h 06m 09s	47x3x1	28x5x1	20x7x1
32	2	15M	81,148	22h 32m 28s	47x3x1	60,865	16h 54m 25s	47x3x1	28x5x1	28x5x1
64	4	15M	53,836	14h 57m 16s	47x3x1	42,918	11h 55m 18s	47x3x1	28x5x1	20x7x1
128	8	15M	61,063	16h 57m 43s	47x3x1	40,963	11h 22m 43s	47x3x1	28x5x1	20x7x1
256	16	15M	45,363	12h 36m 03s	47x3x1	40,098	11h 08m 18s	47x3x1	28x5x1	20x7x1

Table 6.9: NAMD (Default) vs APG for 4LNZ (5,714 atoms).

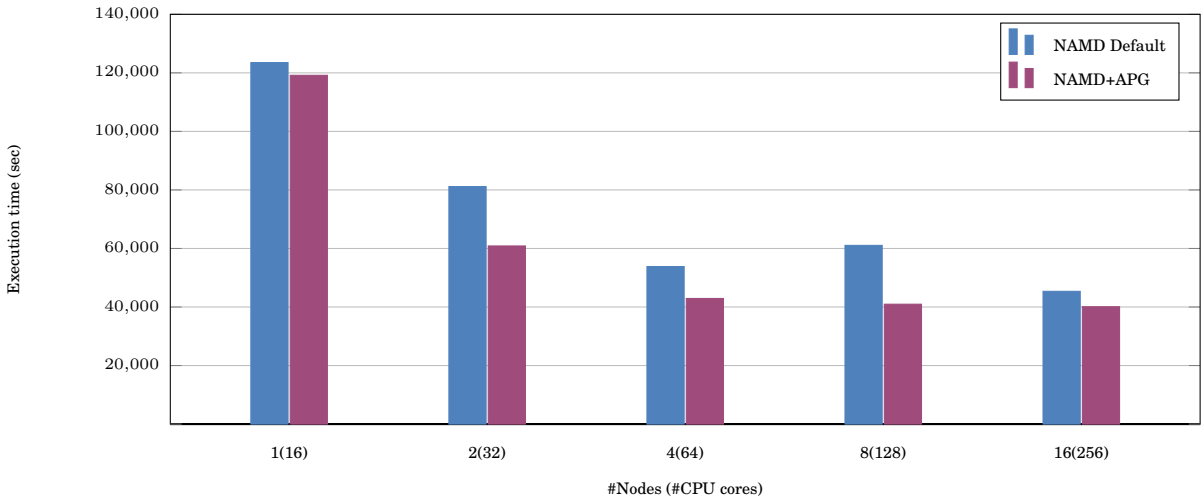


Figure 6.4: Execution times: NAMD (Default) vs NAMD with APG.

While the APG implementation is specific to the patch-based parallelism in NAMD, other MD engines such as GROMACS adopt different parallel models. GROMACS typically uses spatial domain decomposition combined with dynamic load balancing and SIMD-accelerated kernels for nonbonded interactions [23]. These strategies are tailored to its architecture and differ from NAMD's patch/thread-based design, emphasizing the uniqueness of the APG strategy in this context.

6.5 Contribution Review

In this chapter, we proposed and evaluated the Adaptive Patch Grid (APG) strategy which allows NAMD simulations to change its patch grid during the simulations execution flow and adapt to the protein geometric configuration. We presented the challenge of using the default static domain decomposition for molecular dynamics simulations of protein folding and our domain decomposition strategy to address the challenge and the design of our APG strategy proposal.

We then provide an initial description of the chosen computing environment for the simulation, followed by a description of test proteins and a basic configuration file for each simulation execution with NAMD.

Next, we showed our obtained results while evaluating NAMD simulation with the default patch grid using three different proteins in MN4 and Nord, with no evidence of scalability, i.e. reduction of execution time when the number of nodes/-cores is increased. Similarly, we introduced our evaluation with a scaled patch grid, which did not improve the running time or the allocation of cores.

We then presented a manual version of the adaptive patch grid with first results obtained after restating some partially executed molecular dynamics simulations with NAMD. These results were the evidence needed for us to pursue the execution of molecular dynamics simulation using NAMD with our strategy.

Finally, in Section 6.4.7, we show that, using the Nord cluster and the 4LNZ protein, our APG strategy is able to provide scalability up to 8 nodes (128 cores), reducing the simulation time from 34 hours and 18 minutes (1 node with 16 cores) to 11 hours and 22 minutes (8 nodes using APG). If we compare the 128-core execution without and with APG, the execution time decreases from 16 hours and 57 minutes to 11 hours and 22 minutes, which is a considerable reduction in execution time. Recall that in NAMD simulations, communication occurs at the end of each iteration, and our simulations had a total of 15 million iteration steps. So, the reduction of more than 5 hours in the 128-core simulation is remarkable.

Although not part of the APG strategy itself, the atomic burial potential plays an important role in encouraging compact configurations during the protein folding (Section 4.4). By favoring buried atomic arrangements, this potential contributes to realistic structural packing [31–33], complementing the spatial optimization provided by the APG mechanism. The integration of atomic burial into the parallel simulation process is addressed in detail in the next chapter.

It should also be emphasized that the current implementation of APG was developed specifically for simulations using implicit solvent models. The absence of

solvent particles simplifies the computational setup and enables the grid to be adjusted based solely on protein atom positions. While the underlying concept may be adapted for explicit solvent scenarios, such an extension would require additional considerations related to solvent density, diffusion, and interaction balancing, aspects that are beyond the scope of this Thesis.

The proposal and results presented in this chapter were published in [156].

Chapter 7

NAMD with Atomic Burials

The second contribution of this Thesis is the addition of two new forces and corresponding potentials to the classical protein folding simulation executed by NAMD: Atomic Burial (AB) and Hydrogen Bonds (HB) forces. These additional forces are appropriate for globular proteins and produce adequate results, as shown in [31–33, 35].

Section 4.4 showed how the MDBury algorithm applied atomic burial and hydrogen bond forces in molecular dynamics simulations of protein folding to determine the stable 3D configuration of globular proteins with less than 1000 atoms. These simulations were sequentially executed and demanded extremely high execution times. Thus, in order increase the size of the protein simulated with atomic burial forces, we present an additional proposal that consists of inserting the MDBury algorithm in the NAMD tool, which (a) allows the parallel execution of the generic molecular dynamics algorithm, used by MDBury; and (b) allows increasing the number of atoms of the simulated system above the aforementioned limit. To our knowledge, there is no proposal in the literature for using AB+HB in parallel protein folding simulations. All results of this section were published in [156].

Section 7.1 presents the challenge of adding the MDBury computation to NAMD. Section 7.2 presents an overview of this contribution. Section 7.3 describes the algorithm developed for the integration of atomic burials into NAMD. Section 7.4 presents in detail the NAMD components created or modified to achieve this integration. Sections 7.5, 7.6, 7.7 show how we use NAMD's components for adding atomic burial, hydrogen bond and annealing weights computations. Section 7.8 presents our experiments and discusses results obtained with NAMD+AB. Finally, Section 7.9 concludes the chapter with a review of the key takeaways from the chapter.

7.1 Challenge: Parallel execution of MDBury

In order to build this solution, NAMD (Section 4.3) was the parallel molecular dynamics simulation software of choice, because: (a) it provides its source code making possible changes into it; (b) it is capable of handling atomic systems with more than 1,000 atoms; (c) it performs the calculation of traditional molecular dynamics forces similar to those existing in the MDBury algorithm (Section 4.4.2);

and (d) it allows adding new components to calculate MDBury forces, in this case, one related to the atomic burial force, and another to the hydrogen bond force.

Additionally, NAMD provides support to address the following identified *challenges*:

1. How to include the calculation of the atomic burial force in a parallel execution of the molecular dynamics numerical integration, for computing each atomic burial energy contribution and using Equation 4.7?

2. How to include the hydrogen bond force calculation, as described in the MDBury Algorithm (Section 4.4.2)?

3. How to include the calculation and update of the annealing weights for atomic burial and hydrogen bond potentials, as described in MDBury?

Next, Section 7.2 presents an overview of the proposed solution. Sections 7.3, 7.4, 7.5, 7.6 and 7.7 describe in detail the strategies used to insert the MDBury algorithm calculations into the NAMD's execution flow and resolve challenges 1, 2 and 3.

7.2 Overview of the Solution

As stated in Section 4.4, the MDBury algorithm is sequential. It computes Atomic Burial (AB) and Hydrogen Bonds (HB) forces and potentials, and Annealing Weights (AW). It is important to note that the main obstacle identified for creating the parallel version of MDBury was to respect the data dependencies and, at the same time, achieve reasonable parallelism. More specifically, we needed to address these integration obstacles: (i) defining a method to compute atomic burial forces, which involves the geometric center of the current structure and requires global information, (ii) devising a strategy for parallel computation of hydrogen bonds, accounting for cases where one or more atoms necessary for the force computation are located on different nodes, and (iii) incorporating the calculation, update, and distribution of global annealing weight values, which are used in the atomic burial and hydrogen bond potential energy computations carried out on each node.

Figure 7.1 illustrates the default execution flow of NAMD's simulation engine. The process begins with the input of the configuration file, structure file, and force field parameters, which define the initial system setup and physical interactions. Following this, starts the patch grid generation, a domain decomposition step for partitioning simulation space into smaller, parallelizable regions, for large-scale computations (Section 6.2). Within the numerical integration loop, the algorithm computes, in parallel, traditional molecular forces, updates atomic coordinates and velocities, and writes the updated structure to the trajectory file. This iterative loop ensures the accurate simulation of molecular dynamics, according to the standard Algorithm 1, culminating in the generation of a complete trajectory file as the final output.

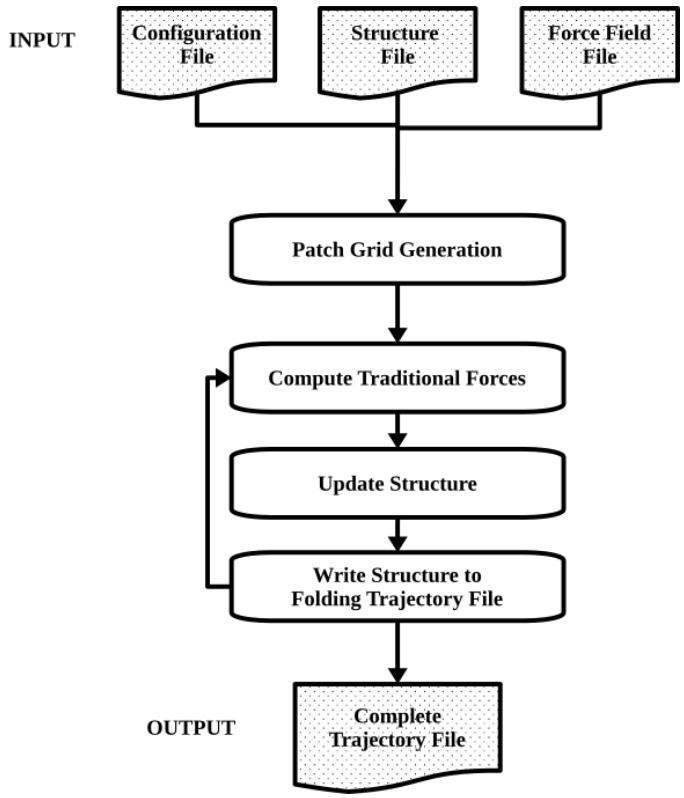


Figure 7.1: Illustration of NAMD's default execution flow.

To enable the parallel execution of MDBury potentials, we developed a strategy to integrate these calculations and their dependencies into NAMD's engine, as illustrated in black in Figure 7.2. This integration also ensures compatibility with our Adaptive Patch Grid (APG) strategy. As part of this effort, we introduced two new routines in NAMD, named *ComputeBurialForce* and *ComputeHBonds*, resulting in a modified version of the NAMD code.

Figure 7.2 showcases the updated execution flow of NAMD, enhanced with the parallelized MDBury algorithm for the computation of Atomic Burial (AB) and Hydrogen Bonds (HB) forces. Similar to the default workflow, the process begins with system initialization and domain decomposition. Within the numerical integration loop, the newly implemented routines — ComputeBurialForce and ComputeHBonds — are incorporated into the framework to compute atomic burial and hydrogen bond potentials at each timestep. These additions, highlighted in black, efficiently manage global data dependencies (such as annealing weights and geometric centers) while preserving parallel scalability.

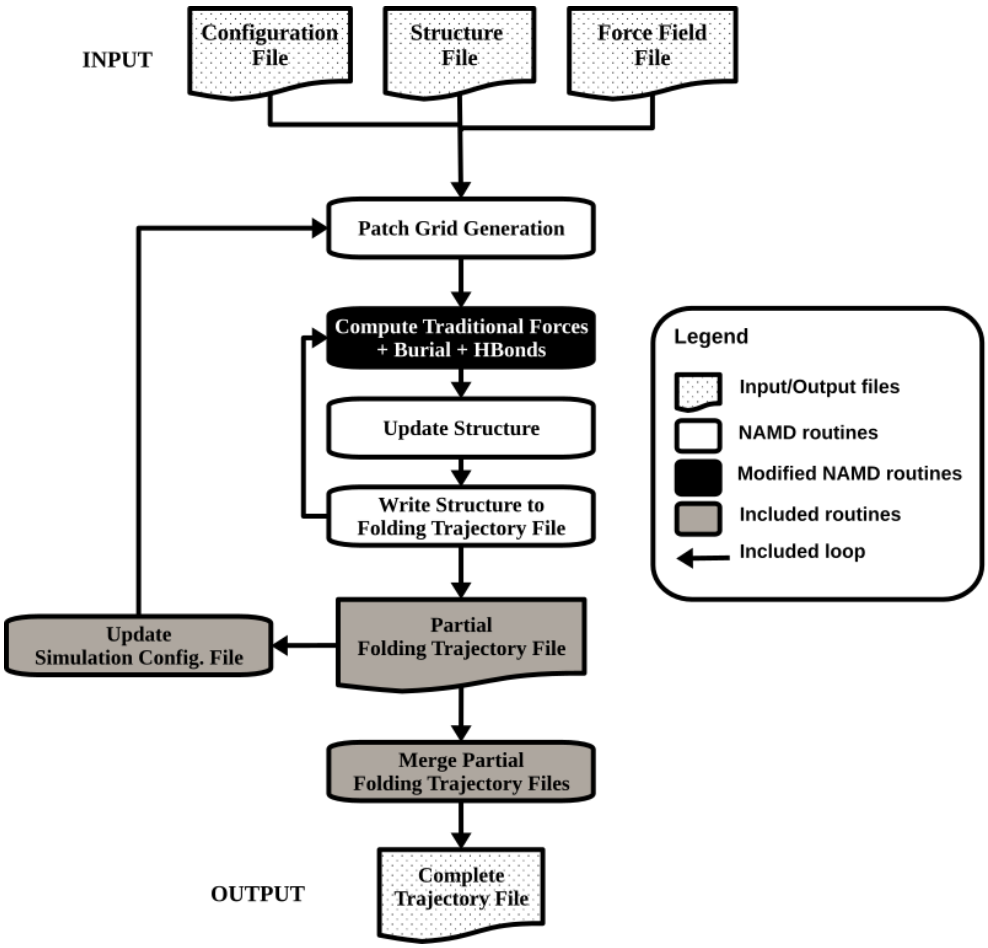


Figure 7.2: Illustration of NAMD+APG+AB execution flow.

Figure 7.3 illustrates the main components integrated into our strategy to extend NAMD with the Atomic Burial (AB) and Hydrogen Bonds (HB) computations. In dark gray, the two new compute objects, ComputeBurialForce and ComputeHBonds, handle the calculation of atomic burial and hydrogen bond forces, respectively. These objects operate within the worker nodes, where each compute object runs inside a dedicated Sequencer thread. The Sequencer thread coordinates the computations required for the assigned patches, ensuring that the atomic burial and hydrogen bond potentials are calculated during each simulation step.

In parallel, the Annealing Weights (AW), which influence the burial and hydrogen bond potentials, are dynamically updated throughout the simulation. These weights are computed centrally on the main node within the Controller thread (Figure 4.7), ensuring consistency across all worker nodes. The updated annealing weights are then distributed to the worker nodes, maintaining synchronization across the system. This efficient division of labor between the main and worker nodes allows the simulation to scale while preserving the integrity of the data dependencies required for accurate potential calculations.

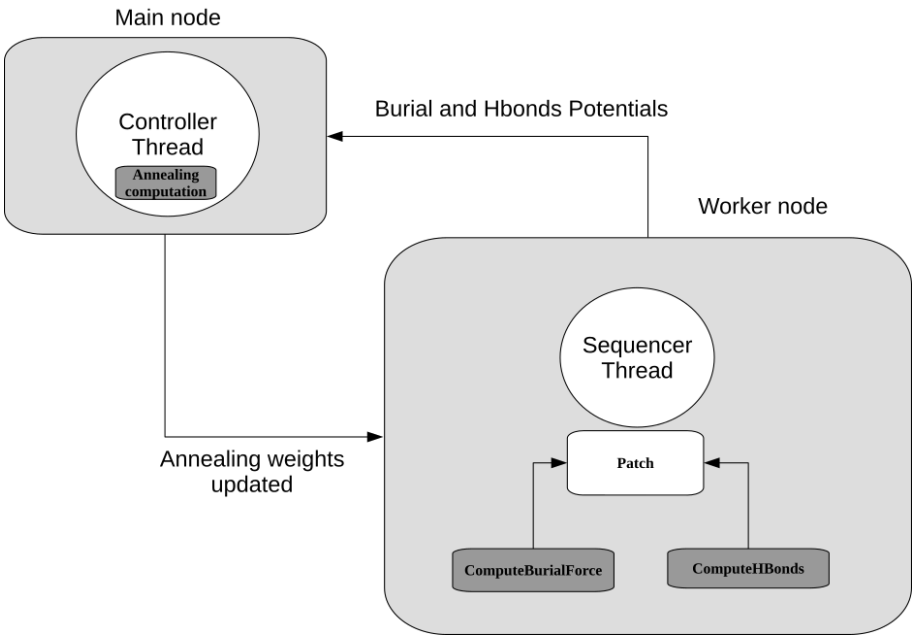


Figure 7.3: NAMD with Atomic Burial (AB).

Next, Section 7.3 introduces the N2HB algorithm, which integrates the computation of atomic burial and hydrogen bond forces into NAMD, addressing the aspects outlined in integration obstacles (i) and (ii). Section 7.4 focuses on the abstraction and modification of NAMD components required to enable parallel computation of atomic burial forces, hydrogen bond forces, and annealing weights, addressing integration obstacles (i), (ii), and (iii). Sections 7.5, 7.6 and 7.7 detail the development of two NAMD components for computing atomic burial forces (ComputeBurialForce) and hydrogen bond forces (ComputeHBonds), as well as the modifications to support annealing weights computations, effectively providing solutions to integration obstacles (i), (ii), and (iii), respectively.

7.3 N2HB Algorithm

7.3.1 Description

Considering the MDBury algorithm presented in Section 4.4, we developed the N2HB algorithm to integrate the necessary modifications for performing MDBury calculations within the general molecular dynamics algorithm. This integration allows MDBury to execute in parallel within the NAMD environment (Section 4.3), applying the potentials described by Equation 4.5 and Equation 4.6.

Similar to Algorithm 1, the N2HB algorithm (Algorithm 2) starts by loading the initial coordinates (a_i) and velocities (v_i) for each the atom i, from 1 to N atoms (line 3, inputs "a" and "b"); inherited from NAMD, it provides support for third-party force fields and access to configure explicit or implicit solvent models (inputs "c" and "d"). For the next inputs "e" and "f", Algorithm 2 reads the *time step* and *num_step* values applied in the numerical integration.

Algorithm 2: N2HB Algorithm

**** The lines in black correspond to the general MD algorithm ****
**** The lines in blue were added to computing the atomic burial method ****

**** Initial state of the atomic system ****
Input :
a) $\{a_1, ..., a_N\}$: initial coordinates (x,y,z) for atoms 1 to N;
b) $\{v_1, ..., v_N\}$: initial velocities for atoms 1 to N;
c) $ForceField$ model: parameters and topology;
d) $Solvation$ model: implicit or explicitly;
e) $time\_step$: size of each chunk of time to be calculated;
f) num_steps: # of iterations for the numerical integration loop.

**** Atomic Burial (AB) + Hydrogen Bond (HB) ****
g) $\{baf_1, baf_2, baf_3\}$: AB annealing factors;
h) $\{r_1^*, ..., r_N^*\}$: AB expected values;
i) $\{\delta_1, ..., \delta_N\}$: AB tolerance interval for the energy function;
j) $\{k_1, ..., k_N\}$: AB *slope* values for the energy function;
k) $\{haf_1, haf_2, haf_3\}$: HB annealing factors;
l) $\{\mu_h, \mu_\eta, \mu_\theta, \mu_r, \beta_h, \beta_\eta, \beta_\theta, \beta_r, \epsilon_{hb}^{max}\}$: parameters for $F(\alpha)$;
m) $\{hb_1, ..., hb_M\}$: HB list of 5-tuples;

**** Initializing Annealing Factors ****
$\{A_{ab}, S_{ab}\} : \leftarrow \{baf_1, baf_2, baf_3\}$;
$\{A_{hb}, S_{hb}, \epsilon_{hb}\} : \leftarrow \{haf_1, haf_2, haf_3, \epsilon_{hb}^{max}\}$;

**** Numerical Integration Loop ****
for $ns \leftarrow 1$ **to** num_steps **do**
    $U_{bonded} \leftarrow$ `ComputeBondedForces` $(a_1, ..., a_N)$;
    $U_{non-bonded} \leftarrow$ `ComputeNonBondedForces` $(a_1, ..., a_N)$;
    $U_{ab} \leftarrow$ `ComputeBurialForce` $(a_1, ..., a_N, r_1^*, ..., r_N^*, \delta_1, ..., \delta_N, k_1, ..., k_N, A_{ab})$;
    $U_{hb} \leftarrow$ `ComputeHBonds` $(a_1, ..., a_N, hb_1, ..., hb_M, A_{hb})$;
    $U_{total} \leftarrow$ `ComputePotentialEnergy` $(U_{bonded}, U_{non-bonded}, U_{ab}, U_{hb})$;
    **** Moving atoms: updating their positions and velocities ****
    $\{a'_1, ..., a'_N\}, \{v'_1, ..., v'_N\} : \leftarrow$ `MoveAtoms` $(a_1, ..., a_N)$;
    $\{a_1, ..., a_N\} : \leftarrow \{a'_1, ..., a'_N\}$;
    $\{v_1, ..., v_N\} : \leftarrow \{v'_1, ..., v'_N\}$;
    **** Updating Annealing weights ****
    $\{A_{ab}\} : \leftarrow \{A_{ab}, S_{ab}\}$;
    $\{A_{hb}, \epsilon_{hb}\} : \leftarrow \{A_{hb}, S_{hb}, \epsilon_{hb}^{max}\}$;
    **** Writing output to log files at predefined iterations****
    Output:
        a) $\{a'_1, ..., a'_N\}$: updates the coordinates of each atom;
        b) $\{v'_1, ..., v'_N\}$: updates the velocities of each atom;
        c) Potential Energy (U_{total}), Temperature (T), Pressure (P), etc.
end

The data of inputs "g", "h", "i" and "j" are used by atomic burial potential calculations. Input "g" contains atomic burial annealing factors ($baf_{1,2,3}$), input "h" deals with atomic burial expected central distances (r_i^*), input "i" holds the tolerance intervals (δ_i) and input "j" the atomic burial slope values for all atomic burials. Inputs "g", "h", "i" and "j" are read from a file, which must contain one line for $baf_{1,2,3}$, N lines with r_i^* and δ_i data for each atom i.

The next inputs "k", "l" and "m" are used to calculate the hydrogen bond potential. Input "k" takes the hydrogen bond annealing factors ($haf_{1,2,3}$), input "l" holds the parameters for function $F(\alpha)$ [32] and input "m" holds the hydrogen bond 5-tuples, where each 5-tuple is composed of the three donor atoms and two acceptor atoms. Inputs "k", "l" and "m" are read from a file containing at least $M + 2$ lines:

one for the annealing factors, one for the $F(\alpha)$ parameters and M lines for all hydrogen bond 5-tuples [32]. All atomic burial and hydrogen bond input parameters must be previously calculated by the applications HmmPred and MDTools. For details about these applications and their use, refer to [32].

Both atomic burial and hydrogen bond potential calculations include an annealing technique, which applies a factor (A_{ab}) to the atomic burial potential term and a penalty (ϵ_{hb}) to the hydrogen bond potential term, respectively. These factors are initialized at lines 5 and 6, applied to their respective potentials at lines 11 and 12 and updated throughout the simulation at lines 19 and 20. For more details about this annealing technique, refer to [31–33].

At the numerical integration loop, the function *ComputeBurialForce* (line 11) computes the atomic burial force (F_{ab}) for all atom positions a_i, using all expected burials (r_i^*) (Inputs "g" to "j"), all interval tolerances (δ_i) and applying the annealing factor (A_{ab}). The energy contribution of each atomic burial force is calculated as defined in Equation 4.7.

The function *ComputeHBonds* (line 12) calculates the resulting force due to hydrogen bond formations, using Inputs "k" to "m". Each hydrogen bond represents an attractive force between one oxygen atom and one nitrogen atom, both belonging to the protein backbone, as explained in Section 4.4.2.

For each 5-tuple hb_i in input "m", the hydrogen bond is evaluated using the coordinates of the 3-donor and 2-acceptor atoms with three defined vectors: $\vec{v_1} = \vec{a_1} - \vec{a_4}$, $\vec{v_2} = \vec{a_2} + \vec{a_3} - 2\vec{a_1}$, and $\vec{v_3} = \vec{a_4} - \vec{a_5}$. The resulting hydrogen bond formed between the donor a_1 and the acceptor a_4, and their adjacent atoms, for any tuple hb_i of input "m", is described by Equation 4.8 in Section 4.4.2.

The parameters used for calculating each function $F(h)$, $F(\eta)$ and $F(\theta)$ of Equation 4.8 are listed in the input "i" of the N2HB algorithm.

The values for the $F(a_i)$ function are also entered at input "l" of the N2HB algorithm and were defined for [32]. Finally, the potential energy term due to all hydrogen bond tuples of input "m", U_{hb}, as defined by Equation 4.6, is calculated by the sum of all energetic contributions, $\sum E_i(\Lambda_i, a_i)$ [32].

7.4 NAMD Components Modified

For NAMD to recognize and execute the atomic burial and hydrogen bond potential calculations proposed in the N2HB algorithm, it is essential to include these calculations, and their dependencies, into specific NAMD components. We first identified these components and listed them according to their functionality within the NAMD execution flow. This abstraction was then used as a reference in the sequence of changes necessary to include the calculations of atomic burial and hydrogen bond potentials and to construct the solution to the challenges listed in Section 7.1.

The list of components and a outline of their functionalities used by our strategy is shown in Table 7.1.

#N	Components (Resource*)	Functionality
1	*DataTypeRepository**	Repository for data types used in NAMD.
2	*Molecule*	Stores and manages structural data.
3	Parameters	Stores and communicates simulation parameters to atoms and different bonds.
4	SimParameters	Stores global simulation parameters, read from the NAMD configuration file (NCF).
5	ComputeMgr	Maps and manages Compute objects, used to create, record, execute and transfer data between them.
6	WorkDistrib	Calculates and defines the layout of Patches, manages the mapping and distribution of Compute objects and its association with the created Patches objects.
7	LdbCoordinator	Defines, executes and monitors NAMD's load balancing strategies.
8	ReductionMgr	Provides management functionality for reduction operations, such as the sum of all energy terms.
9	Broadcasts	Provides interface for exchanging broadcast messages.
10	Sequencer	Runs the numerical integration *loop* on each node. It receives messages from the Controller and responds with the calculated results.
11	Controller	Controls the progress of the overall MD algorithm on all nodes. It runs in the starting node broadcasting data and commands to Sequencers in all nodes, receiving the results and writing the output log files.

Table 7.1: Description of eleven NAMD elements modified in this PhD Thesis.

In Table 7.1, one resource and ten components are enumerated and named in the first and second columns, respectively. The third column contains a synopsis of each functionality used in our strategy. For a list with the names of the components and corresponding modified files, responsible for implementing each functionality, see the Table C.1 in Appendix C.

The first item in the list (N=1 in Table 7.1) contains the term *DataTypeRepository**. This is the only exception in the Table, because it does not represent a component, per se. This term was created exclusively for this work, just to facilitate understanding and organization of the solution. It is used to name a set of NAMD elements that provide the functionality of a repository for declaring basic data types used throughout the project. And, due to its comprehensive use by all the objects of this work, it was included in the list.

The second item in the list is the first component: *Molecule*. This component has the methods to store and communicate the structural data of the system, for example, the names of the atoms, the coordinates and the geometric configuration of the molecule. At the beginning of NAMD execution, data from this component is

sent to all compute nodes so that all created Compute objects have access to it.

The third item is the *Parameters* component. This component is responsible for storing the parameters associated with atoms and their interactions, *bonded* and *non-bonded*, within the molecular system. The parameters registered in this component are also sent to all nodes to be used by the Compute objects during the parallel execution of the simulation.

The fourth item is the *SimParameters* component. This is the component responsible for reading and processing the the global parameters of the simulation obtained from INPUT files (Figure 7.2) and are applied throughout the NAMD execution.

The fifth item is the *ComputeMgr* component. This component needs to be changed for the purpose of making NAMD recognize and execute the *Compute* objects created for our strategy. All *Compute* objects must be declared in the *ComputeMgr*, which allows them to be created, registered (for later mapping) and executed in NAMD's engine. In this component, there are methods for mapping and managing the behavior of *Compute* objects. Furthermore, it allows data to be transferred and updated.

The sixth item is the *WorkDistrib* component, which contains the methods to create the *layouts* of *Home Patches*, and will manage the mapping and distribution of calculations in parallel in NAMD, that is, associate the *Compute* objects with the corresponding *Patches* and include these objects in the execution queue of each computational node, as illustrated in Figure 4.6.

The seventh item is the *LdbCoordinator* component, responsible for running and monitoring the load balancing strategies that are executed by NAMD in each simulation. The two strategies available in NAMD are hierarchical and centralized balancing, the latter being *default*.

The eighth item is the *ReductionMgr* component, which is responsible for the reduction operations used by NAMD. These operations bring together the results of energy contribution calculations into a data structure, called *reduction structure*. For the reduction calculation mechanism to work, it needs at least: (a) to register, in *ReductionMgr*, an identifier for energy contribution; and (b) in the *Compute* object, to use the energy identifier responsible for inserting into the reduction structure the result of the energy contribution calculation.

The ninth item is the *Broadcasts* component. Its main functionality is to provide a message exchange interface for communication between *Controller* and *Sequencer* objects.

The last two items refer to the components *Sequencer* and *Controller*, mentioned in Section 4.3.2. The functionalities of these components that we identified as relevant to our strategy are:

(a) the *Sequencer* object is responsible for executing the molecular dynamics simulation algorithm within the computational node, it receives global data from the *Controller* to be used by the calculations of the *Compute* objects created in that node and then sends the results of these calculations to the *Controller* .

(b) the *Controller* runs on the main node and coordinates the global evolution of the molecular dynamics simulation algorithm (using message exchange between local and remote objects), it receives the calculated values from the *Sequencer* and

writes in *log* files the data related to both the microscopic characteristics of the system, such as the positions and velocities of the atoms, as well as macroscopic ones, such as the temperature, pressure and energy of the system throughout the simulation.

7.5 Computing Atomic Burial Forces

The atomic burial force is calculated in the *ComputeBurialForce* object and it is responsible for computing (a) the atomic burial force values for each atom i within the existing home patches in the worker node; and (b) the atomic burial energy contribution to the total potential energy, using function $B(r_i)$ (Equation 4.7). For NAMD to recognize this object and execute its methods, it was necessary to include the *ComputeBurialForce* object into NAMD's parallel execution flow. In order to do so, it was necessary to map the object's data dependencies and register them into NAMD's engine components.

Figure 7.4 shows NAMD's components that were modified in order to include the *ComputeBurialForce* object. First, we adapted the *DataTypeRepository* resource, which contains objects that host data dependencies. The BurialInfo type was mapped into this resource for handling the values of $r_i^*,\ \delta_i,\ k_i$, as well as $r_1, \ldots, r_5,\ m_1, \ldots, m_5,\ n_1,\ n_2,\ n_5$ (Equation 4.7), in order to calculate the energy contribution $B(r_i)$ (Equation 4.7) within *ComputeBurialForce*.

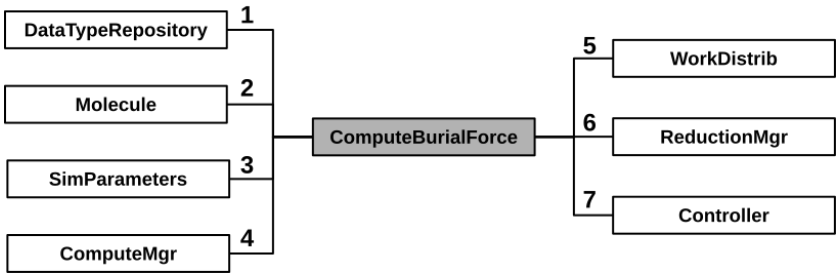


Figure 7.4: NAMD components used for adding *ComputeBurialForce*. Modified from NAMD's Class Hierarchy [166].

Then, we modified the component Molecule to load $r_i^*,\ \delta_i,\ k_i$, calculate $r_1, \ldots, r_5,\ m_1, \ldots, m_5,\ n_1,\ n_2,\ n_5$, and fill the BurialInfo element. Since the atomic burial force depends on the distance from the atom to a central position, this computation does not depend on atom interaction. For this reason, we added a method that creates a *Position* object with the geometric center (central position), and included it in *ComputeBurialForce*, to compute distances to the geometric center. We also included BurialInfo in the method that handles the messages sent by the main node and contains the structural data that are handled by the *ComputeBurialForce* object.

In addition, the components *SimParameters* and *ComputeMgr* were created and the option to perform the atomic burial calculation was added. Moreover, the object *ComputeBurialForce* was created and inserted into NAMD's Compute objects queue. We also modified the component *WorkDistrib* by adding an entry to the

method mapComputes, thus creating a binding between *ComputeBurialForce* and its corresponding Patch, which contains the atom data necessary for parallel computations.

In *ReductionMgr*, the identifier `REDUCTION_BURIAL_ENERGY` was registered, referring to atomic burial energy contribution. This identifier is also used by *ComputeBurialForce*, for adding the atomic burial contribution to the parallel reduction computation, calculated for the i atoms which belong to the associated Home Patch. Finally, we inserted global operations into the *Controller* for atomic burial forces calculations, in order to: (a) recover the sum of the atomic burial energy contributions calculated by all *ComputeBurialForce* objects (U_{ab}^{total}); (b) apply annealing weights to U_{ab}^{total} (Section 7.7); and (c) add the value of U_{ab}^{total} to the global potential energy.

7.6 Computing Hydrogen Bond Forces

NAMD computes non-bonded interactions, like hydrogen bond, through (a) *SelfCompute* objects, for atoms in the same home patch; and (b) *PairCompute*, for atoms in different ones. To perform computations in parallel, NAMD ensures that data from the atoms are available to Compute objects, including the patches.

Figure 7.5 presents the NAMD components that were adapted for adding hydrogen bond forces. In the first step (*DataTypeRepository*), the HBondsInfo type was included to handle the values of "k" and "l" (Algorithm 2) and haf_1, haf_2, haf_3, μ_h, μ_η, μ_θ, μ_r, β_h, β_η, β_θ, β_r and ϵ_{hb}^{max}. In addition, the type HBond was inserted to hold the values of the 5-tuples stated in input "m" of Algorithm 2, i.e. $hb_1, ..., hb_M$. Finally, the types HBondsAcc and HBondsDon were added to contain data from acceptors and donors, respectively.

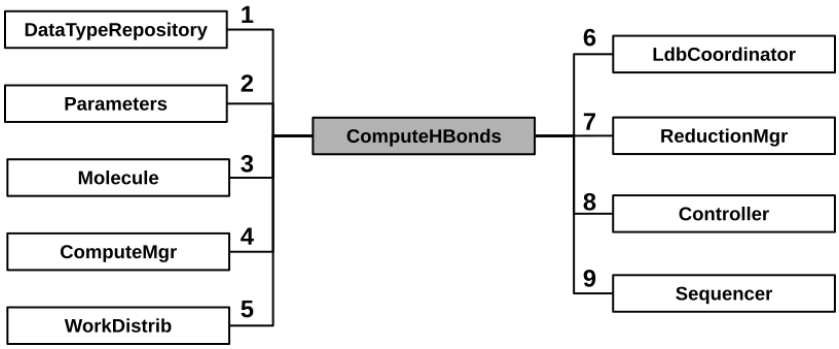


Figure 7.5: NAMD components used for adding *ComputeHBonds*. Modified from NAMD's Class Hierarchy [166]

The *Parameters* component was also modified to register the data structure that holds the *HBondValue* objects. With this, *HBondValue* can be sent by the main node to all workers, for parallel execution.

We then included the methods for reading input data and initial processing in the *Molecule* component. More specifically, object types were created for force and potential calculations. In this component, the *HBondsInfo* objects are filled, and a

mapping is created for all acceptor and donor atoms necessary for hydrogen bond interactions, using the *HBondsAcc* and *HBondsDon* components. Finally, these elements are also registered in the interface of the *Molecule* component for communication so that they are sent by the parent node to child nodes and are available to objects *ComputeHBonds* and *ComputeSelfHBonds* during parallel execution.

The objects *ComputeHBonds* and *ComputeSelfHBonds* were then registered in in components *ComputeMgr* and *WorkDistrib*. In *ComputeMgr*, those objects were mapped to the Compute object queue. Additionally, in *WorkDistrib*, instructions were included to map *ComputeHBonds* objects to home patches and proxy patches, and *ComputeSelfHBonds* objects were mapped to their home patches.

The new Compute objects were also included in NAMD's load balancing and registered in NAMD's reduction operations. For inclusion in load balancing, *ComputeHBonds* and *ComputeSelfHBonds* were registered in the *LdbCoordinator* component, which maps the Compute objects to patches [108].

After that, we handled global operations. The components involved were the *Sequencer* and the *Controller*. In the *Sequencer*, we added the functionality of receiving, from the *Controller*, the value of the annealing weight (Section 7.7) and updating it for the Compute objects in its worker node. In the *Controller*, a feature was added for receiving energy contributions (U_{hb}) from the *Sequencer* and adding them to the total potential energy calculation.

7.7 Computing Annealing Weights

For including calculations associated with the annealing weights applied in in Algorithm 2, we modified the NAMD components shown in Figure 7.6, in order to guarantee the correct values while performing a parallel execution.

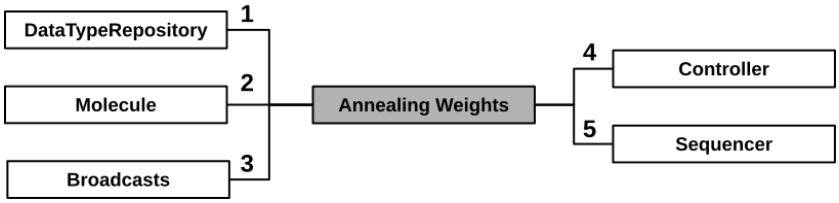


Figure 7.6: NAMD components used for adding *Annealing* weights computation. Modified from NAMD's Class Hierarchy.

Considering the problem listed in Challenge (1), the elements A_{ab}, S_{ab}, A_{hb}, S_{hb}, ϵ_{hb} are global variables in MDBury's sequential version. However, for the parallel execution, the usage of these elements represent potential points for race conditions. Our strategy for solving this challenge consists of five steps so that reading and updating of the annealing weights are performed correctly in parallel.

The first step was to add the data dependencies needed for computing the *Annealing* contribution (Algorithm 2), i.e.: the weights (A_{ab}, A_{hb} and ϵ_{hb}), the increment step (S_{ab} and S_{hb}) and the factors (baf_1, baf_2, baf_3, haf_1, haf_2 e haf_3). Similarly to atomic burial and hydrogen bond, we modified BurialInfo and HBondsInfo, adding the annealing fields.

The second step was the modification of the *Molecule* component and comprised: (a) reading the input factors baf_1, baf_2, baf_3, haf_1, haf_2 e haf_3, used in the calculations of *annealing weights* in atomic burial and hydrogen bond forces; (b) initialization of weights and increments for computing annealing in the atomic burial and hydrogen bond forces computations.

In the parallel execution, the values of weights, increments, and factors should be available to compute objects *ComputeBurialForce* and *ComputeHBonds* which will use them. As explained in Section 7.5, NAMD makes the *Molecule* component data available in all compute nodes. Therefore, we registered these values in the data type BurialInfo and HBondsInfo (see Sections 7.5 and 7.6) to make them available to *Compute* objects.

The last three steps are responsible for updating and using *annealing weights*. We used two objects *SimpleBroadcastObject*, one for each *annealing weight* (A_{ab} and ϵ_{hb}), which are updated globally and sent to objects *ComputeBurialForce* and *ComputeHBonds*. The *SimpleBroadcastObject* is used in the communication between the *Controller* and *Sequencer* objects, ensuring the correct update of values on nodes. In the *Controller*, we added the *UpdateAnnealing* method for getting the values of A_{ab} and ϵ_{hb}, calculating their new global values according to Algorithm 2, and sending them to *Sequencer* objects. Similarly, in the *Sequencer* object, the method *UpdateAnnealing* is used to receive the values of A_{ab} and ϵ_{hb}, update their values inside the worker node, which will be accessed by the parallel *threads* of the *ComputeBurialForce* and *ComputeHBonds* objects.

7.8 Experimental Results

In this section, we conducted tests using the proteins and execution environments detailed in Sections 6.4.1 and 6.4.2, respectively. The largest protein, 4LNZ, was selected for these tests, and the HPC environment utilized was Nord.

7.8.1 NAMD Configuration with Atomic Burials

Similar to was presented in Section 6.4.3 the objective of this Section is to present a basic configuration of NAMD that allows running a parallel molecular dynamics simulation of the folding process of proteins using the atomic burial method.

We will refer as NCAB to this basic NAMD configuration file containing either additional parameters for running NAMD with our Atomic Burial (AB) contribution (NAMD+AB) or containing additional parameters for running NAMD with our both contributions, the APG strategy and the Atomic Burial (AB) method (NAMD+APG+AB). An example of the contents of all parameters used in this work is available in Appendix B.

In order to maintain compatibility with the atomic burial technique, two parameters from Algorithm 2 were added to NCAB and their manipulation was inserted into NAMD's source code. The first parameter is called *burials* and defines the name of the file containing the input values used in atomic burial calculations,

such as *annealing* factors and *estimated* center distances, which are used in the initialization process (items "e" e "f" of line 3 in Algorithm 2). The second parameter is called *hbondsAF* and defines the name of the file containing the input values used in hydrogen bond potential energy calculations, such as hydrogen bond *annealing* factors, hydrogen bond parameters for the calculation of the function $F(\alpha)$ and the list of 5-tuple atoms used in hydrogen bond calculation. (items "i", "j" e "k" from the Algorithm 2).

Finally, the atomic burial and hydrogen bond calculations that were included in NAMD for executing our experimental tests with parallel molecular dynamics simulations using atomic burial will only work if both *burials* and *hbondsAF* parameters are defined in the NACB. Otherwise, NAMD will run without atomic burial and hydrogen bond calculations.

7.8.2 Tests in Nord: APG and Atomic Burial

Execution Times

Here we present the execution times, simulation performance, speedup and efficiency obtained with executions that used both APG and Atomic Burial (AB). In this experiment, we also used up to 256 cores and the largest protein (4LNZ) in its extended form (1D). The results are shown in Table 7.2.

#C	#N	#It.	NAMD with AB							
			Default Strategy			APG Strategy				
			Execution time		Patch Grid	Execution time		Patch Grid		
			seconds	(h:m:s)		seconds	(h:m:s)	Phase:1	Phase:2	Phase:3
16	1	15M	126,511	35h 08m 31s	47x3x1	125,904	34h 58m 24s	47x3x1	28x5x1	20x7x1
32	2	15M	88,454	24h 34m 14s	47x3x1	65,338	18h 08m 58s	47x3x1	28x5x1	20x7x1
64	4	15M	45,595	12h 39m 55s	47x3x1	39,812	11h 03m 32s	47x3x1	28x5x1	20x7x1
128	8	15M	40,172	11h 09m 32s	47x3x1	37,494	10h 24m 54s	47x3x1	28x5x1	20x7x1
256	16	15M	44,420	12h 20m 20s	47x3x1	37,318	10h 21m 58s	47x3x1	35x4x1	20x7x1

Table 7.2: NAMD+AB executions times: not using APG strategy (Default) vs using APG strategy. Protein used: 4LNZ with 5,714 atoms.

In Table 7.2, it is worth noticing that, also with Atomic Burial (AB), the execution times are reduced when APG is used. Figure 7.7 shows the comparison of execution times from Tables 6.9 and 7.2. In all executions, the runtime with atomic burial was lower than the runtime without it. This happens because, with Atomic Burial (AB), we added forces (atomic burial and hydrogen bonds) and this can potentially change the way the protein folds over the simulation.

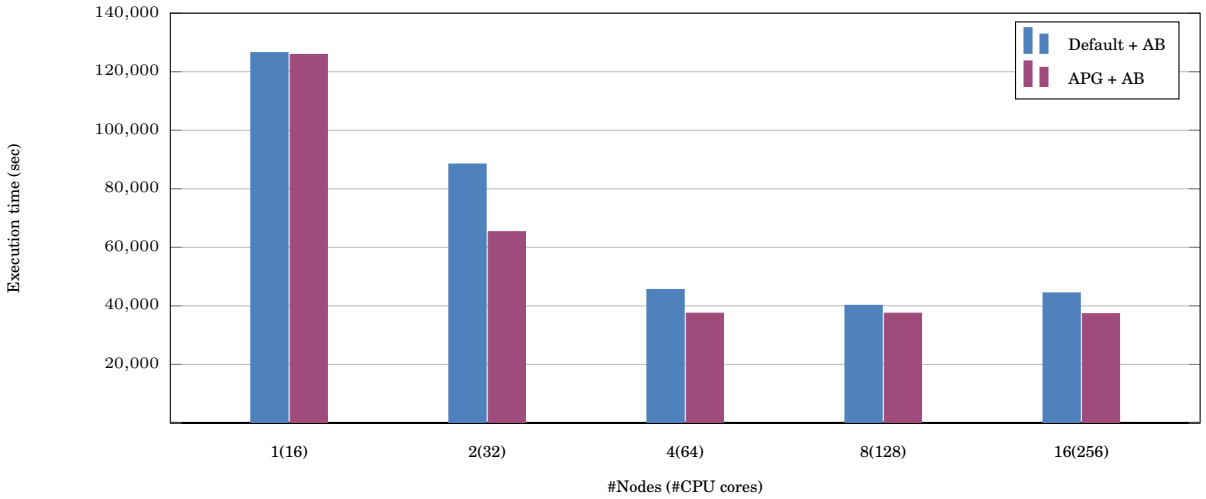
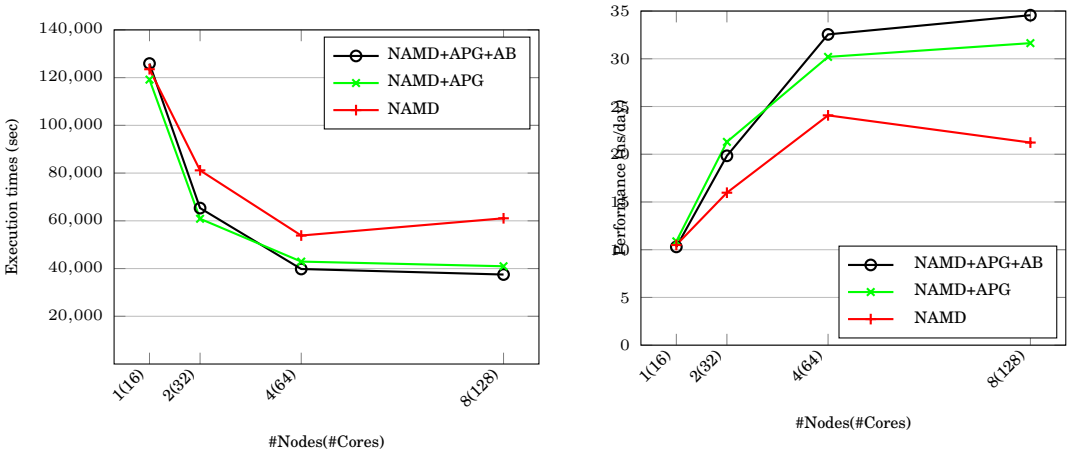


Figure 7.7: Execution times: default NAMD + AB vs APG + AB.

Figure 7.8a shows the execution times (in seconds) and Figure 7.8b the performance (in nanoseconds per day), both for running molecular dynamics simulations of the folding process for protein 4LNZ. These simulations were executed with three strategies: default NAMD, NAMD+APG and NAMD+APG+AB. All simulations were performed in Nord cluster, using in 1, 2, 4 and 8 nodes - 16, 32, 64 and 128 CPU cores, respectively. Note that, in Figures 7.7 and 7.8a, we were able to reduce the execution time with APG and APG+AB, and the best results were obtained running NAMD+APG+AB. This is also shown in Figure 7.8b where NAMD+APG+AB was able to achieve up to 34 ns/day whereas default NAMD achieved up to 24 ns/day. We only plotted the speedup and parallel efficiency for up to 4 nodes (64 cores) since it can be seen in Figure 7.8 that the gain in execution time with 8 and 16 nodes is marginal.



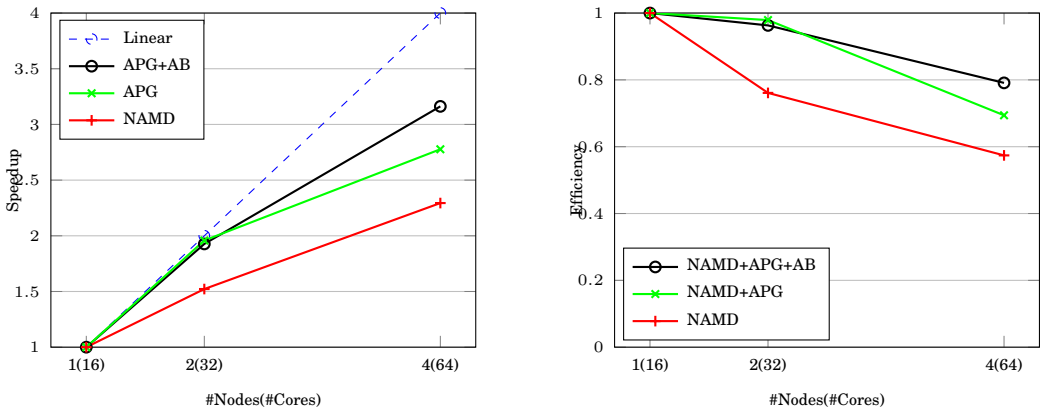
(a) Execution times × number of nodes

(b) Performance (ns/day) × number of nodes

Figure 7.8: Execution times and performance (ns/day) using up to 8 nodes.

Figure 7.9a shows the speedup and Figure 7.9b the parallel efficiency for these

simulations. In Figure 7.9a, the best speedup (3.162) was obtained with NAMD+APG+AB on 4 nodes (64 cores) and this corresponds to an 80% efficiency (Figure 7.9b). We can also see in Figure 7.9 that both NAMD+APG and NAMD+APG+AB are able to improve considerably the speedup of NAMD for protein 4LNZ. It is worth noticing that NAMD simulations have considerable message exchange and, in this scenario, an efficiency of 80% is very good.



(a) Speedup × number of nodes

(b) Efficiency × number of nodes

Figure 7.9: N2HB: Speedup and parallel efficiency using up to 4 nodes.

Realistic Protein Folding with NAMD+APG+AB

In this section, we discuss the results obtained with NAMD+APG+AB in a realistic simulation using the 4LNZ protein. This simulation was prepared using the same basic configuration described in Section 7.8.2, with *time step* set to 1 fs and the number of iterations was set to 1.7 billion, which corresponds to a folding time of 1.7 µs.

The first objective of this 1.7 µs simulation was to observe the behavior of NAMD+APG+AB in a realistic simulation. The execution was divided into 17 phases with 100 million iterations each, with 20 subphases of 5 million iterations. This was done to respect the 48 hour per job restriction in the Nord cluster. Overall, this simulation took more than 2 months.

NAMD with APG+AB						
Phase	Subphases	#Iter.	Runtime		*Patch Grid*	
			(seconds)	(d:h:m:s)	Config.	*#Patches*
1	1-20	100M	559,927	6d 11h 32m 07s	47 x 3 x 1	141
4	61-80	100M	628,306	7d 06h 31m 46s	6 x 4 x 3	72
8	141-140	100M	444,717	5d 03h 31m 57s	5 x 4 x 4	80
12	221-240	100M	354,569	4d 02h 29m 29s	8 x 5 x 3	120
16	301-320	100M	332,177	3d 20h 16m 17s	6 x 7 x 3	126
17	321-340	100M	371,293	4d 07h 08m 13s	5 x 8 x 3	120

Table 7.3: Runtime and patch grid values for the realistic simulation. Protein used: 4LNZ with 5,714 atoms

Table 7.3 presents the values of patch grids for some phases/subphases in this experiment. It is interesting to notice that, at the beginning, the simulation box had an elongated shape (patch grid 47 x 3 x 1). Then, the shape evolved to a more compact one in phases 4 and 8. After that, in phases 12, 16 and 17, the simulation box assumes an intermediary shape, showing that the simulation is evolving to a more stable configuration.

In order to visualize the folding of protein 4LNZ, its geometric configurations were generated using the VMD software [167]. For that, we provided to VMD the trajectories file, that contains, for this particular simulation, a sequence of 68,000 *frames* recorded during the execution. Each *frame* represents a snapshot of the protein's 3D configuration at that moment (atomic positions). Using our trajectory file, VMD generated a 3D graph consisting of 1,000 *frames* uniformly sampled along the trajectory, providing a detailed visualization of the folding process.

Additionally, VMD was used to generate a *frame* of the native 3D structure of the 4LNZ protein, obtained from PDB [163] using the atomic coordinates provided in the original PDB file. This *frame* served exclusively as a reference for visual comparison.

Figure 7.10 shows the starting point of the simulation, i.e., the 1D structure of 4LNZ.

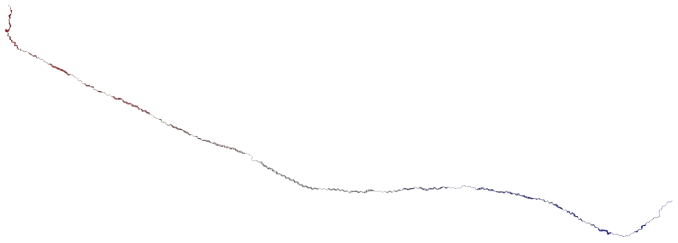


Figure 7.10: Geometric configuration of the 4LNZ protein at the beginning of the simulation.

Figure 7.11 illustrates the progress of the simulation and presents *frame* 35, generated from record 2,380 of atomic coordinates. At this point in the simulation, it can be seen that, at this moment of the simulation, the protein has started to fold.

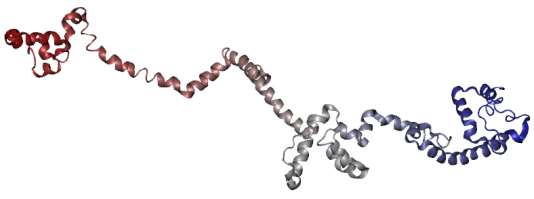
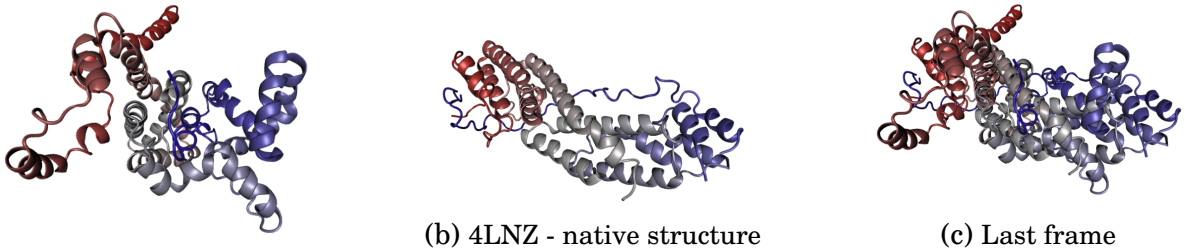


Figure 7.11: Geometric configuration of 4LNZ during the simulation (*frame* 35).

To quantitatively assess how close the simulated structure is to the native conformation, the Root Mean Square Deviation (RMSD) is used [168]. RMSD, expressed in Angstroms (Å), measures the average distance between atoms of superimposed protein structures. In this study, RMSD values were computed using VMD

by comparing the frames generated from our simulation to the native 3D structure of the 4LNZ protein, obtained from the original PDB file. This metric allows us to evaluate how the protein evolves toward its folded state during the simulation.

Figure 7.12 presents: (a) the *frame* corresponding to the last 3D configuration written to the trajectory file at the end of the NAMD+APG+AB simulation; (b) the native 3D structure of protein 4LNZ obtained from the original PDB file; and (c) the superposition of both structures for RMSD calculation. Although the simulated structure is close to the native conformation, some differences remain. This suggests that a simulation time longer than 1.7 µs may be required to complete the folding process.



(a) 4LNZ - last frame

(b) 4LNZ - native structure (PDB)

(c) Last frame superimposed on native

Figure 7.12: 3D configurations of 4LNZ. (a) On the left, the last *frame* obtained after running NAMD+APG+AB for 1.7 billion iterations.(b) In the center, the native structure generated from the original PDB file. (c) On the right, last frame superimposed onto the native structure for RMSD calculation.

The results of the RMSD calculation are presented in Figure 7.13. We report RMSD values only for this realistic simulation, since those shown in Section 7.8.2 corresponded to shorter runs (up to 20 million steps) that began from the protein's primary structure (unfolded state).

This is visually represented in Figure 7.13, which illustrates through the RMSD values, the evolution of the protein's transformations along its folding. The X axis shows the sample space composed of 1000 frames. Position $x = 0$ represents *frame* 0, i. e., the first *frame* generated from the 1D configuration of 4LNZ (Figure 7.10). The Y axis shows the RMSD values, where the value of y is equal to the RMSD calculated between the *frame* of the native structure and frame x. In this graph, the lower the RMSD, the better. At the beginning of the simulation, for the first *frame*, the calculated RMSD value is equal to 306.5 Å, confirming the great difference between the structures at the beginning of the simulation (Figure 7.10). For *frame* 35 (Figure 7.11), the value of RMSD is 52.1 Å. This reduction in RMSD indicates an increased similarity between the geometric configurations. For the last *frame* represented in Figure 7.12, the calculated RMSD value is 25.0 Å. Therefore, we can notice that, at the beginning, there is a great reduction in RMSD. However, after the first folding operations, the RMSD decreases slowly but constantly. We also calculated the TM-score (0.1765) and the GDT-TS-score (9.07) using the Yang Zhang's webserver at https://zhanggroup.org/TM-score/, and those values are consistent with the 25.0 Å RMSD. This shows how hard MD-based protein folding simulation is and suggests that, for this protein, we would need more than two months of execution in Nord to obtain the native structure. The 4LNZ protein was

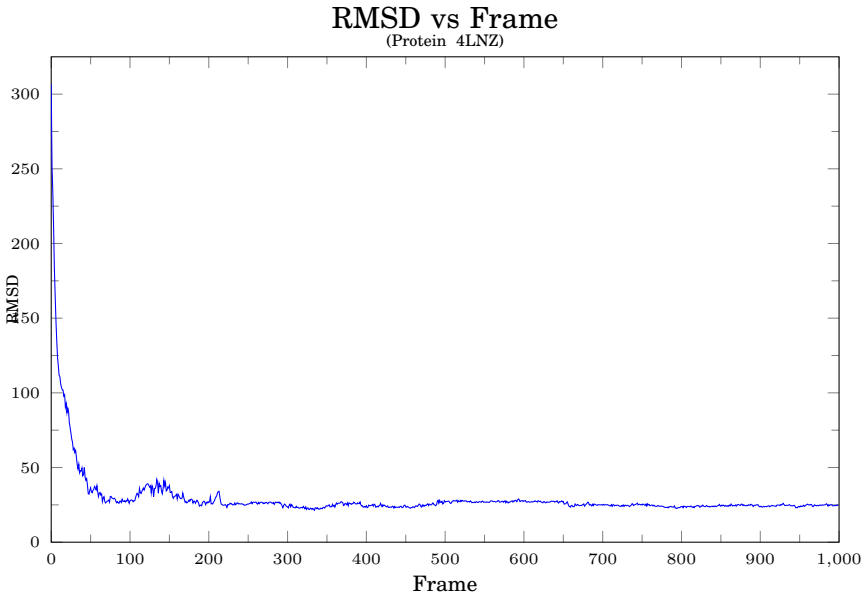


Figure 7.13: RMSD between the structure obtained by APG+AB and the native structure along the execution.

chosen for this realistic simulation because it has a greater number of atoms than previous proteins (Table 6.1).

7.9 Contribution Review

In this chapter, we proposed and evaluated our strategy for parallel execution of MDBury's algorithm using NAMD. In conjunction with NAMD and APG (Section 7.8), the atomic burials force was able to reduce the simulation running time, allowing it to work on proteins with a number of atoms bigger than 5,000 atoms, including with our APG strategy. As far as we know, this is the first time atomic burials are integrated to a parallel simulation framework such as NAMD.

We first showed how the atomic burial, hydrogen bond and annealing weights computation were added to NAMD's engine, by modifying its source code.

Next, we presented our results while running NAMD with both our strategies (APG+AB) for simulating the folding of the protein 4LNZ, showing the obtained execution time, performance, speedup and efficiency results for this experiment.

Then, we presented the result of a realistic simulation with 1.7 billion iteration for a total of 1.7 µs of simulated time. We showed the runtimes obtained in each phase of our APG strategy execution, then the RMSD calculated between the native structure and our simulated protein.

The proposal and results presented in this chapter were published in [156].

Part III

Conclusions

Chapter 8

Conclusions and Future Work

8.1 Conclusions

Ab initio Molecular Dynamics simulations of Protein Folding demand high computing power on account of mainly two factors: (i) the complex calculations for simulating molecular behavior are time-dependent and (ii) they often use an explicit solvation model for representing all the atoms in the target system (solvent and solute).

In this PhD Thesis we investigated two parallel strategies to accelerate *ab initio* molecular dynamics protein folding simulations, while maintaining good accuracy. Our first contribution is a strategy called Adaptive Patch Grid (APG), performed in the simulation workflow of the NAMD parallel tool for molecular dynamics simulations. Here, we added a new *loop* to the execution workflow of the molecular dynamics simulation performed with NAMD, starting at the beginning of NAMD's domain decomposition at a point that we identified as *Patch Grid Generation*. Next, we added new actions for writing a *Partial Folding Trajectory File*, *Update the Simulation Configuration File* and *Merge the Partial Trajectories*.

In the *Patch Grid Generation* loop, after a predefined number of molecular dynamics simulation steps, we reinsert the *partial* 3D geometric configuration of the protein as input to NAMD's domain decomposition method, which generates a new *patch grid* and the corresponding set of *patches* are reassigned to CPU cores. Next, for the *Partial Folding Trajectory File*, we write the changes in 3D geometric configuration that are used as input to generate the new *patch grid*. For the *Configuration File Update*, we added intermediary simulation data to be used in the next execution, such as the first loop step and the number of steps to be performed in the next iteration. Finally, we *Merge All The Partial Trajectory Files* into a single file containing the complete trajectory that is presented in the output, as expected in a typical molecular dynamics simulation.

Our second contribution is a parallel execution strategy for a method called Atomic Burial (AB) applied in *ab initio* protein folding simulations, as defined in [31, 32]. We incorporated the corresponding Atomic Burial (AB) and Hydrogen Bonds (HB) forces and potentials, as well as the annealing weights, into NAMD's classical force computation engine, which allowed the parallel execution of atomic burial and hydrogen bond calculations in a molecular dynamics simulation of pro-

tein folding, potentially increasing the sizes of the simulated globular proteins and reducing the execution time of these simulations. We constructed an algorithm, called N2HB, and incorporated it to NAMD. We used NAMD's component abstraction to add the necessary calculations of AB+HB forces and potentials into NAMD's engine for parallel execution. We analyzed and modified 11 components in NAMD and added 2 new objects to compute the corresponding AB+HB forces and potentials, along with the annealing weights. To the best of our knowledge, there was no existing solution that integrates atomic burials into a molecular dynamics simulation framework such as NAMD.

In order to evaluate our proposals, our experiments were conducted in two high-performance computing environments: the Marenostrum 4 supercomputer [144] and the Nord III cluster [159]. The proteins selected for our experiments were globular and of different sizes to compose the target molecular systems. We presented evaluation results showing that there was no evidence of scalability or reduction of execution time while running simulations using the default NAMD tool for simulating protein folding in a implicit solvent model in Marenostrum 4 and in Nord.

We performed tests on our first contribution, the APG strategy, in the Nord Cluster, using with 1, 2, 4, 8 and 16 computing nodes, with 16 CPU cores on each node. Each test consisted of molecular dynamics simulations of the folding process of protein 4LNZ with 5,714 atoms, running NAMD using our APG strategy. We did not use Atomic Burials in these tests. In the molecular dynamics numerical integration loop, we executed a total of 20 million iterations. With parallel execution, these iterations were divided into 4 phases of 5 million iterations each. For each test with different number of computing nodes, we recorded simulation data consisting of (i) the execution time of each phase, (ii) the simulation box configuration, called *patch grid*, generated by NAMD's domain decomposition method for each phase, and (iii) the running time of the complete 20 million iteration molecular dynamics simulation. Next, we conducted another set of tests using a numerical integration loop with 15 million iterations to compare the parallel execution of molecular dynamics simulations performed with a fixed patch grid against those using our adaptive patch grid. Each test was executed twice: (a) once with default NAMD without APG, and (b) once with NAMD and our APG strategy. We compared all these results and noted that the parallel execution using NAMD and our APG strategy for 4LZN protein reduced the execution time in at least 1 hour compared to default NAMD simulation, considering that the best results were obtained while using 16 nodes (256 cores).

To evaluate our second contribution, running NAMD with the Atomic Burial method and APG (NAMD+APG+AB), we performed molecular dynamics simulations in the Nord cluster, using 1, 2, 4, 8 and 16 nodes, each node with 16 CPU cores. For each simulation test, we utilized the 4LNZ protein, consisting of 5,714 atoms, and ran a numerical integration loop for 15 million iterations. Each test was executed twice: (i) once using default NAMD with AB+HB forces, however without our APG strategy (we called default NAMD+AB); (ii) once using NAMD with AB+HB applying our APG strategy with 3 phases of 5 million iterations each; we named this strategy APG+AB.

We compared the results in both cases and observed a reduction of 2 hours in the execution times when using the NAMD+APG+AB strategy with 256 CPU cores. In all cases, the runtime with atomic burial was consistently shorter than without it, indicating that incorporating the two forces and potentials from the atomic burial method can potentially influence the protein's folding during the simulation. We also compared the performance in nanoseconds per day across three strategies: default NAMD, NAMD+APG, and NAMD+APG+AB. The NAMD+APG+AB strategy achieved up to 34 ns/day, while the default NAMD reached up to 24 ns/day. Additionally, we evaluated the speedup and parallel efficiency on up to 4 nodes (64 cores). Once again, our proposed strategies, NAMD+APG and NAMD+APG+AB improved the speedup of NAMD for simulating the folding process of the globular protein 4LNZ: the best speedups obtained were 2.77x with NAMD+APG and 3.16x with NAMD+APG+AB, both running on 4 nodes using a total of 64 cores.

Lastly, we performed a realistic protein folding simulation with 1.7 billion iterations, simulating a 1.7 μs folding process of the 4LNZ protein. Using NAMD+APG+AB, this simulation was executed in 17 phases, each consisting of 100 million iterations, further subdivided into 20 subphases of 5 million iterations each. The total execution spanned over two months. We recorded the runtime and the patch grid generated for each phase. Using these data, we observed the evolution of the simulation box configuration over time. Unlike the fixed domain decomposition approach of default NAMD, our APG strategy allowed the simulation box configuration to adapt to the protein's structural changes throughout the simulation, evolving from an elongated shape around the protein in its 1D primary structure, with a patch grid of 47x3x1 (total of 141 patches), to more compact 3D configurations such as 5x4x4 and 6x4x3 (80 and 72 patches, respectively).

We recorded 68,000 atomic positions, referred to as *frames*, to visualize the realistic folding process. Using the VMD software, we: (i) generated a 3D graphic with 1,000 frames along the protein trajectory; and (ii) calculated the Root Mean Square Deviation (RMSD) for each frame compared to the native 3D structure. The RMSD data revealed a consistent reduction in the structural difference between the initial 1D 4LNZ structure and the final 3D structure over the course of the simulation, decreasing from an expected 306.5 Å to 25.0 Å. We confirmed these results through additional measures, calculating a TM-score of 0.1765 and a GDT-TS score of 9.07, both of which align with the 25.0 Å RMSD.

To summarize, we conclude that our contributions advance the state-of-the-art in parallel protein folding simulations. We successfully achieved the proposed objectives of this PhD Thesis, which were: (i) to investigate parallel strategies that reduce the execution time of molecular dynamics simulations of protein folding while maintaining good accuracy, and (ii) to explore a flexible approach to incorporate additional force computations into such simulations, especially those related to molecular stability. These objectives were met through the development of two complementary strategies: the Adaptive Patch Grid (APG) and the Parallel Execution Strategy for Atomic Burial computation (N2HB algorithm), both of which demonstrated significant improvements in simulation performance and scalability. These contributions, along with the obtained results, were published in [156].

8.2 Future work

In recent years, artificial intelligence techniques have been successfully applied to the protein folding problem [30], although they rely on prior information that is not available in *ab initio* simulations. With this in mind, we plan to incorporate new modules that leverage artificial intelligence strategies to improve *ab initio* molecular dynamics simulations using both of our contributions.

For our APG strategy, it was observed during the experiments that finding the appropriate relationship between the number of processing units (nodes or cores) and the generated patch grids requires numerous executions of the domain decomposition algorithm and subsequent numerical integration loops. This can lead to the decision to discard a simulation if the generated patch grid does not produce a usable ensemble. To avoid this scenario, we consider as future work the addition of a module assisted by a supervised learning algorithm that could be trained on previously obtained ensembles for a family of globular proteins of interest. With this, we hope that a model can suggest better patch grids for new simulations and improve further the molecular dynamics simulation workflow when using our APG strategy.

In our experiments, we noticed that the geometric positions of the atoms remain quite the same in most iterations, and the geometric positions change quite rapidly in other specific iterations. Thus, we consider using artificial intelligence to accelerate the parts of the simulation where the geometric positions remain almost the same. We also intend to use artificial intelligence to detect when the geometric positions are changing quickly and, in such cases, switch to the traditional molecular dynamics, where we will run molecular dynamics simulations using both our strategies.

Given the large dataset generated with the *Partial Folding Trajectory Files* when using our APG strategy, we plan to incorporate a module that leverages artificial intelligence models. These models will be trained to predict intermediate folding states based on our partial trajectory data, allowing us to bypass unnecessary intermediate steps or prioritize the exploration of specific folding trajectories. This approach aims to reduce the computational cost of simulating the entire folding process.

The largest globular protein analyzed in our experiments (4LNZ) consists of 5,714 atoms. In future work, we aim to extend our simulations to larger proteins, which will benefit from utilizing more computational nodes and cores. To facilitate this, we will implement a new module designed to evaluate and allocate the optimal number of nodes or cores for each simulation, improving the usage of available resources.

Building on the above improvements, we believe that our simulations can be enhanced by adding a new module to the N2HB algorithm. This module would harness the various implicit solvent models supported by NAMD, as well as other resources available in NAMD, such as the Collective Variables. These features could also be used by an artificial intelligence module to guide the sampling of specific conformations relevant to simulating the protein folding process.

Finally, although RMSD was adopted as the primary structural descriptor in this work due to its robustness and wide usage in molecular dynamics studies, it is important to acknowledge that other parameters can provide valuable complementary insights into the folding process. Solvent Accessible Surface Area (SASA), radius of gyration (Rg), the number of intramolecular hydrogen bonds, and side-chain contact formation are examples of structural features that reflect additional aspects of protein compaction, packing, and stability [20, 21]. These descriptors are particularly relevant for identifying folding intermediates and evaluating native-like structural patterns. Their inclusion, however, would have required a broader analytical pipeline and additional post-processing that were beyond the scope of this work. Therefore, their omission reflects a methodological decision aligned with the core objectives of this study, which focused on evaluating folding dynamics under atomic burial forces using parallel simulation strategies. Future work may benefit from integrating these additional metrics to deepen the structural interpretation of folding trajectories.

References

[1] Donald Voet and Judith G. Voet. *Biochemistry*. John Wiley & Sons, Inc., 4 edition, 2021. ISBN 9781119770640. International Adaptation; Last access: 2025-03-01. 1, 8, 10, 11, 12, 13, 15, 16, 20

[2] K. A. Dill and J. L. MacCallum. The protein-folding problem, 50 years on. *Science*, 338:1042–1046, 2012. doi: 10.1126/science.1219021. Last access: 2025-03-01. 1, 13, 14, 17, 21

[3] Emiliano Brini, Carlos Simmerling, and Ken A. Dill. Protein storytelling through physics. *Science*, 370, Nov 2020. doi: 10.1126/science.aaz3041. Last access: 2025-03-01. 1, 3, 18, 19, 29

[4] Jacob D. Durrant and J. Andrew McCammon. Molecular dynamics simulations and drug discovery. *BMC Biology*, 9(71):992–1023, oct 2011. doi: 10.1186/1741-7007-9-71. Last access: 2025-03-01. 1, 2, 33, 34, 35, 37, 38

[5] Márcio Dorn, Mariel Barbachan e Silva, Luciana S. Buriol, and Luís C. Lamb. Three-dimensional protein structure prediction: Methods and computational strategies. *Computational Biology and Chemistry*, 53:251–276, 2014. doi: 10.1016/j.compbiolchem.2014.10.001. Last access: 2025-03-01. 1, 2, 10, 18, 25, 26, 27, 28, 29, 30, 32

[6] Jérôme Hénin, Tony Lelièvre, Michael R Shirts, Omar Valsson, and Lucie Delemotte. Enhanced sampling methods for molecular dynamics simulations. *arXiv.org*, 2022. doi: 10.1016/j.bbagen.2014.10.019. Last access: 2025-03-01. 1, 2, 29, 31, 37

[7] B. Acun, D. J. Hardy, L. V. Kale, K. Li, J. C. Phillips, and J. E. Stone. Scalable Molecular Dynamics with NAMD on the Summit System. *IBM Journal of Research and Development*, 62(6):4:1–4:9, Dez 2018. doi: 10.1147/JRD.2018.2888986. Last access: 2025-03-01. 1, 3, 4, 38, 39, 47, 55, 56, 74, 75, 76, 85

[8] S. Peng, Y. Cui, S. Yang, W. Su, X. Zhang, T. Zhang, W. Liu, and X. Zhao. A CPU/MIC Collaborated Parallel Framework for GROMACS on Tianhe-2 Supercomputer. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(2):425–433, March 2019. ISSN 1545-5963. doi: 10.1109/TCBB.2017.2713362. Last access: 2025-03-01. 1, 3, 4, 56, 57, 74, 75

[9] X. Duan, P. Gao, T. Zhang, M. Zhang, W. Liu, W. Zhang, W. Xue, H. Fu, L. Gan, D. Chen, X. Meng, and G. Yang. Redesigning LAMMPS for Peta-Scale and Hundred-Billion-Atom Simulation on Sunway TaihuLight. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 148–159. IEEE, Nov 2018. doi: 10.1109/SC.2018.00015. Last access: 2025-03-01. 1, 3, 4, 58, 59, 74, 75, 76

[10] P. Malakar, T. Munson, C. Knight, V. Vishwanath, and M. E. Papka. Topology-Aware Space-Shared Co-Analysis of Large-Scale Molecular Dynamics Simulations. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 305–319. IEEE, Nov 2018. doi: 10.1109/SC.2018.00027. Last access: 2025-03-01. 1, 3, 59, 60, 61, 75

[11] Jaewoon Jung, Chigusa Kobayashi, Kento Kasahara, Cheng Tan, Akiyoshi Kuroda, Kazuo Minami, Shigeru Ishiduki, Tatsuo Nishiki, Hikaru Inoue, Yutaka Ishikawa, Michael Feig, and Yuji Sugita. New parallel computing algorithm of molecular dynamics for extremely huge scale biological systems. *Journal of Computational Chemistry*, 42(4):231–241, 2021. doi: 10.1002/jcc.26450. Last access: 2025-03-01. 1, 3, 47, 61, 63, 75

[12] David E. Shaw, Peter J. Adams, Asaph Azaria, and 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*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384421. doi: 10.1145/3458817.3487397. Last access: 2025-03-01. 1, 3, 4, 38, 63, 64, 75, 77

[13] Hyungro Lee, Heng Ma, Matteo Turilli, Debsindhu Bhowmik, Shantenu Jha, and Arvind Ramanathan. DeepDriveMD: Deep-Learning Driven Adaptive Molecular Simulations for Protein Folding. *2019 IEEE/ACM Third Workshop on Deep Learning on Supercomputers (DLS)*, pages 12–19, 2019. doi: 10.1109/DLS49591.2019.00007. Last access: 2025-03-01. 1, 3, 28, 65, 66, 75, 77

[14] Yui Tik Pang, Yinglong Miao, and Yi Wang et al. Gaussian Accelerated Molecular Dynamics in NAMD. *Journal of Chemistry Theory and Computation*, 13:9–19, 2017. doi: 10.1021/acs.jctc.6b00931. Last access: 2025-03-01. 1, 3, 66, 67, 75, 77

[15] Michela Taufer, Stephen Thomas, and Michael Wyatt et al. Characterizing In Situ and In Transit Analytics of Molecular Dynamics Simulations for Next-Generation Supercomputers. In *2019 15th International Conference on eScience*, pages 188–198, 09 2019. doi: 10.1109/eScience.2019.00027. Last access: 2025-03-01. 1, 3, 67, 68, 69, 75, 76, 77

[16] Omar Awile, Ferit Büyükkeçeci, Sylvain Reboux, and Ivo F. Sbalzarini. Fast neighbor lists for adaptive-resolution particle simulations. *Computer Physics*

Communications, 183(5):1073–1081, 2012. doi: 10.1016/j.cpc.2012.01.003. Last access: 2025-03-01. 1, 3, 69, 70, 75, 76

[17] Pedro Gonnet. Pairwise verlet lists: Combining cell lists and verlet lists to improve memory locality and parallelism. *J. Comput. Chem.*, 33(1):76–81, 2012. doi: 10.1002/jcc.21945. Last access: 2025-03-01. 1, 3, 70, 71, 72, 75, 76

[18] Pedro Gonnet. Pseudo-Verlet lists: a new, compact neighbour list representation. *Molecular Simulation*, 39(9):721–727, 2013. doi: 10.1080/08927022.2012.762097. Last access: 2025-03-01. 1, 3, 70, 71, 72, 75, 76

[19] Adam K. Sieradzan, Jordi Sans-Duñó, and et al. Emilia A. Lubecka. Optimization of parallel implementation of UNRES package for coarse-grained simulations to treat large proteins. *Journal of Computational Chemistry*, 44 (4):602–625, 2023. doi: 10.1002/jcc.27026. Last access: 2025-03-01. 1, 2, 3, 4, 72, 73, 75, 76

[20] Pedro R. Arantes, Lucas T. Costa, Guilherme M. A. Ferreira, Rodrigo A. C. A. Lima, Renata B. Araujo, Wendel A. Santos, Rafael R. C. Pereira, and Rodrigo Ligabue-Braun. Development of GROMOS-compatible parameter set for simulations of chalcones and flavonoids. *The Journal of Physical Chemistry B*, 123(5):994–1008, 2019. doi: 10.1021/acs.jpcb.8b10503. Last access: 2025-03-01. 2, 120

[21] Marcelo Depólo Polêto, Bruno Iochins Grisci, Márcio Dorn, and Hugo Verli. ConfID: an analytical method for conformational characterization of small molecules using molecular dynamics trajectories. *Bioinformatics*, 36(11): 3576–3577, 2020. doi: 10.1093/bioinformatics/btaa130. Last access: 2025-03-01. 2, 29, 37, 120

[22] Gantulga Norjmaa, Gregori Ujaque, and Agustí Lledós. Beyond Continuum Solvent Models in Computational Homogeneous Catalysis. *Topics in catalysis*, 65(1-4):118–140, 2022. doi: 10.1007/s11244-021-01520-2. Last access: 2025-03-01. 2, 3, 4, 19, 36

[23] Mark Abraham, Andrey Alekseenko, Cathrine Bergh, and et al. GROMACS 2023.1 Manual, apr 2023. Last access: 2025-03-01. 2, 3, 4, 19, 33, 36, 56, 94

[24] Eva Prašnikar, Martin Ljubič, Andrej Perdih, and Jure Borišek. Machine learning heralding a new development phase in molecular dynamics simulations. *Artificial intelligence review*, 57(4, 102), 2024. ISSN 1573-7462. doi: 10.1007/s10462-024-10731-4. Last access: 2025-03-01. 2, 3, 37

[25] Tim Isgro, James Phillips, Marcos Sotomayor, and et al. NAMD TUTORIAL. Unix/MacOSX Version. Tutorial, Beckman Institute, University of Illinois at Urbana-Champaign, February 2012. Computational Biophysics Workshop; Last access: 2025-03-01. 2, 3, 4, 33, 36, 84, 85

[26] Jaroslaw Meller. Molecular Dynamics. In *Encyclopedia of Life Sciences*. John Wiley & Sons, Inc., 2001. doi: 10.1038/npg.els.0003048. Last access: 2025-03-01. 3, 31, 32, 37, 38

[27] Harold A. Scheraga, Mey Khalili, and Adam Liwo. Protein-Folding Dynamics: Overview of Molecular Simulation Techniques. *Annual Review of Physical Chemistry*, 58:57–83, 2007. doi: 10.1146/annurev.physchem.58.032806.104614. Last access: 2025-03-01. 3, 19, 31, 36

[28] Roy Nassar, Emiliano Brini, Sridip Parui, Cong Liu, Gregory L. Dignon, and Ken A. Dill. Accelerating Protein Folding Molecular Dynamics Using Inter-Residue Distances from Machine Learning Servers. *Journal of Chemical Theory and Computation*, 18(3):1929–1935, 2022. doi: 10.1021/acs.jctc.1c00916. Last access: 2025-03-01. 3, 19, 29

[29] Alexey V. Onufriev and David A. Case. Generalized Born Implicit Solvent Models for Biomolecules. *Annual Review of Biophysics*, 48(1):275–296, 2019. doi: 10.1146/annurev-biophys-052118-115325. Last access: 2025-03-01. 3, 4, 36

[30] John Jumper, Richard Evans, and et al Alexander Pritzel. Highly accurate protein structure prediction with AlphaFold. *Nature (London)*, 596(7873):583–589, 2021. doi: 10.1038/s41586-021-03819-2. Last access: 2025-03-01. 4, 19, 28, 119

[31] A. F. P. Araújo, A. L. C. Gomes, A. A. Bursztyn, and E. I. Shakhnovich. Native atomic burials supplemented by physically motivated hydrogen bond constraints, contain sufficient information to determine the tertiary structure of small globular proteins. *Proteins: Structure, Function, and Bioinformatics*, 70(3):971–983, 2008. doi: 10.1002/prot.21571. Last access: 2025-03-01. 4, 5, 6, 42, 94, 96, 102, 116

[32] M. G. van der Linden, D. C. Ferreira, and L. C. de Oliveira et al. Ab initio protein folding simulations using atomic burials as informational intermediates between sequence and structure. *Proteins: Structure, Function, and Bioinformatics*, 82(7):1186–1199, 2014. doi: 10.1002/prot.24483. Last access: 2025-03-01. 4, 5, 6, 42, 43, 45, 94, 96, 101, 102, 116

[33] A. F. P. de Araújo and J. N. Onuchic. A sequence-compatible amount of native burial information is sufficient for determining the structure of small globular proteins. *Proceedings of the National Academy of Sciences*, 106(45):19001, 2009. doi: 10.1073/pnas.0910851106. Last access: 2025-03-01. 5, 6, 42, 94, 96, 102

[34] Aidan P. Thompson, H. Metin Aktulga, Richard Berger, and et al. LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer Physics Communications*, 271:108171, 2022. doi: 10.1016/j.cpc.2021.108171. Last access: 2025-03-01. 6, 76

[35] Marx G. van der Linden, Diogo C. Ferreira, and Antônio F. Pereira de Araújo. Constrained Layer Assignment for the Protein Burial Folding Code Accounting for Chain Connectivity. *The journal of physical chemistry. B*, 126(33): 6159–6170, 2022. doi: 10.1021/acs.jpcb.2c03931. Last access: 2025-03-01. 6, 42, 96

[36] C. B. Anfinsen. Principles that Govern the Folding of Protein Chains. *Science*, 181:223–230, 1973. doi: 10.1126/science.181.4096.223. Last access: 2025-03-01. 8, 13

[37] Mary. K. Campbell, Shawn O. Farrell, and Owen McDougal. *Biochemistry*. Cengage Learning, 7 edition, 2016. ISBN 9781118918401. Last access: 2025-03-01. 8, 10, 11, 14, 15, 16, 18, 20, 21

[38] F. Bettelheim, W. Brown, and M. Campbell et al. *Introduction to General, Organic and Biochemistry*. Cengage Learning, 2015. ISBN 9789354243820. Bookmarks at pages: 498, 499; Last access: 2025-03-01. 8

[39] Boundless Learning. Proteins - Amino Acids. https://bio.libretexts.org/@go/page/12699, 2025. Figure 3.8.1. Licensed under CC BY-SA 4.0. Last access: 2025-03-01. 9

[40] J. Kyte and R. F. Doolittle. A simple method for displaying the hydropathic character of a protein. *Journal of Molecular Biology*, 157(1):105–132, 1982. doi: 10.1016/0022-2836(82)90515-0. Last access: 2025-03-01. 9, 10

[41] S. Kamtekar, J. M. Schiffer, and H. Xiong et al. Protein Design by Binary Patterning of Polar and Nonpolar Amino Acids. In *Protein Engineering Protocols*, volume 352 of *Methods in Molecular Biology*, pages 155–166. Humana Press, 1993. doi: 10.1126/science.8259512. Last access: 2025-03-01. 9

[42] Allison Soult and CK-12 Foundation. 13.2: Peptides. https://chem.libretexts.org/@go/page/58853, 2025. Figure 13.2.1. Licensed under the CK-12 Curriculum Materials License. Last access: 2025-03-01. 8, 10

[43] Ken A. Dill. Dominant forces in protein folding. *Biochemistry*, 29(31):7133–7155, 1990. doi: 10.1021/bi00483a001. Last access: 2025-03-01. 10

[44] David L. Nelson and Michael M. Cox. *Lehninger Principles of Biochemistry*, volume 1. W. H. Freeman and Company, 7 edition, 2017. ISBN 9781319114671. Last access: 2025-03-01. 11, 12, 13, 14

[45] Mercy Akinwale, Jerry Emmanuel, Itunuoluwa Isewon, and Jelili Oyelade. Application of Deep learning Algorithms On Protein Function Prediction: A Systematic Review. In *2024 International Conference on Science, Engineering and Business for Driving Sustainable Development Goals (SEB4SDG)*, pages 1–9, 2024. doi: 10.1109/SEB4SDG60871.2024.10629655. Last access: 2025-03-01. 11

[46] F. Sanger and H. Tuppy. The amino-acid sequence in the phenylalanyl chain of insulin. I. The identification of lower peptides from partial hydrolysates. *The Biochemical journal*, 49(4):463–481, sep 1951. doi: 10.1042/bj0490463. Last access: 2025-03-01. 11

[47] Jonathan Pevsner. *Bioinformatics and functional genomics (3rd. ed.)*. John Wiley & Sons Inc, 3 edition, 2015. ISBN 1118581784. Last access: 2025-03-01. 11, 12

[48] Allison Soult and CK-12 Foundation. 13.3: Protein Structure. https://chem.libretexts.org/@go/page/58854, 2025. Figures 13.3.1 and 13.3.2. Licensed under the CK-12 Curriculum Materials License. Last access: 2025-03-01. 12

[49] RCSB Protein Data Bank. Structure of Human Carbonic Anhydrase II (2FO3). https://www.rcsb.org/structure/2FO3, 2025. Adapted visualization from RCSB PDB. Last access: 2025-03-01. 13

[50] Richard Harvey and Denise Ferrier. *Biochemistry (5th. ed.)*. Lippincott Williams & Wilkins, 2011. ISBN 160831412X. Last access: 2025-03-01. 13, 14

[51] H. Jane Dyson and Peter E. Wright. Intrinsically unstructured proteins and their functions. *Nature Reviews Molecular Cell Biology*, 6(3):197–208, 2005. doi: 10.1038/nrm1589. Last access: 2025-03-01. 16, 85

[52] Peter E. Wright and H. Jane Dyson. Intrinsically unstructured proteins: re-assessing the protein structure-function paradigm. *Journal of Molecular Biology*, 293(2):321–331, 1999. doi: 10.1006/jmbi.1999.3110. Last access: 2025-03-01. 16

[53] Vladimir N. Uversky. Introduction to intrinsically disordered proteins (IDPs). *Chemical Reviews*, 114(13):6557–6560, 2014. doi: 10.1021/cr500391x. Last access: 2025-03-01. 16, 85

[54] M. Madan Babu, R. van der Lee, N. S. de Groot, and J. Gsponer. Intrinsically disordered proteins: regulation and disease. *Current Opinion in Structural Biology*, 21(3):432–440, 2011. doi: 10.1016/j.sbi.2011.03.006. Last access: 2025-03-01. 16

[55] Rakesh Trivedi and Hampapathalu Adimurthy Nagarajaram. Intrinsically Disordered Proteins: An Overview. *International Journal of Molecular Sciences*, 23(22):14050, 2022. doi: 10.3390/ijms232214050. Last access: 2025-03-01. 16

[56] Sarah E. Bondos, A. Keith Dunker, and Vladimir N. Uversky. Intrinsically Disordered Proteins Play Diverse Roles in Cell Signaling. *Cell Communication and Signaling*, 20(1):20, 2022. doi: 10.1186/s12964-022-00821-7. Last access: 2025-03-01. 16

[57] J. C. Kendrew, G. Bodo, and et al H. M. Dintzis. A three-dimensional model of the myoglobin molecule obtained by x-ray analysis. *Nature*, 181:662–666, 1958. doi: 10.1038/181662a0. Last access: 2025-03-01. 17

[58] Roy Nassar, Gregory L Dignon, Rostam M Razban, and Ken A Dill. The Protein Folding Problem: The Role of Theory. *Journal of Molecular Biology*, 433(20):167126, 2021. doi: 10.1016/j.jmb.2021.167126. Last access: 2025-03-01. 18, 22

[59] C. A. Floudas, H. K. Fung, and S. R. McAllister et al. Advances in protein structure prediction and De Novo protein design: A review. *Chemical Engineering Science*, 61:966–988, 2006. doi: 10.1016/j.ces.2005.04.009. Last access: 2025-03-01. 18, 27, 28

[60] Andrew W. Senior, Richard Evans, John M. Jumper, and et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577:706 – 710, 2020. doi: 10.1038/s41586-019-1923-7. Last access: 2025-03-01. 19, 28

[61] Jan K. Labanowski. Molecular Modeling. Web site, December 1996. URL http://www.ccl.net/cca/documents/molecular-modeling/node9.html. Last access: 2025-03-01. 19, 31

[62] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comput.*, 19:297–301, 1965. doi: 10.2307/2003354. Last access: 2025-03-01. 20

[63] Xiao-Chen Bai, Tamir Gonen, Angela Gronenborn, Anastassis Perrakis, Andrea Thorn, and Jianyi Yang. Challenges and opportunities in macromolecular structure determination. *Nature Reviews Molecular Cell Biology*, 25, 10 2023. doi: 10.1038/s41580-023-00659-y. Last access: 2025-03-01. 21

[64] Nidhi Singh and Wenjin Li. Recent Advances in Coarse-Grained Models for Biomolecules and Their Applications. *International Journal of Molecular Sciences*, 20(15):3774, Aug 2019. ISSN 1422-0067. doi: 10.3390/ijms20153774. Last access: 2025-03-01. 22, 23, 24

[65] Amjad Chowdhury, Jonathan A. Bollinger, Barton J. Dear, Jason K. Cheung, Keith P. Johnston, and Thomas M. Truskett. Coarse-Grained Molecular Dynamics Simulations for Understanding the Impact of Short-Range Anisotropic Attractions on Structure and Viscosity of Concentrated Monoclonal Antibody Solutions. *Molecular Pharmaceutics*, 17(5):1748–1756, 2020. doi: 10.1021/acs.molpharmaceut.9b00960. Last access: 2025-03-01. 22, 23

[66] Joshua A. Riback, Micayla A. Bowman, Adam M. Zmyslowski, Catherine R. Knoverek, John M. Jumper, James R. Hinshaw, Emily B. Kaye, Karl F. Freed, Patricia L. Clark, and Tobin R. Sosnick. Innovative scattering analysis shows that hydrophobic disordered proteins are expanded in water. *Science*, 358 (6360):238–241, 2017. doi: 10.1126/science.aan5774. Last access: 2025-03-01. 22

[67] Sebastian Kmiecik, Dominik Gront, Michal Kolinski, and et al. Coarse-Grained Protein Models and Their Applications. *Chemical Reviews*, 116(14): 7898–7936, 2016. doi: 10.1021/acs.chemrev.6b00163. Last access: 2025-03-01. 23, 24, 76

[68] R. Sánchez and A. Sali. Advances in comparative protein-structure modelling. *Current Opinion in Structural Biology*, 7(2):206–214, 1997. doi: 10.1016/s0959-440x(97)80027-9. Last access: 2025-03-01. 24, 25

[69] M. A. Marti-Renom, M. S. Madhusudhan, and A. Sali. Alignment of protein sequences by their profiles. *Protein Science*, 13(4):1071–1087, 2004. ISSN 1469-896X. doi: 10.1110/ps.03379804. Last access: 2025-03-01. 24, 25

[70] J. U. Bowie, R. Lüthy, and D. Eisenberg. A Method to Identify Protein Sequences That Fold into a Known Three-Dimensional Structure. *Science (New York, N.Y.)*, 253(5016):164–170, jul 1991. ISSN 0036-8075. doi: 10.1126/science.1853201. Last access: 2025-03-01. 25, 26

[71] D. T. Jones, W. R. Taylor, and J. M. Thornton. A new approach to protein fold recognition. *Nature*, 358(6381):86–89, jul 1992. ISSN 0028-0836. doi: 10.1038/358086a0. Last access: 2025-03-01. 25, 26

[72] S. H. Bryant and S. F. Altschul. Statistics of sequence-structure threading. *Current Opinion in Structural Biology*, 5(2):236–244, 1995. ISSN 0959-440. doi: 10.1016/0959-440x(95)80082-4. Last access: 2025-03-01. 25, 26

[73] M. Turcotte, S. Muggleton, and M. J. E. Sternberg. Application of Inductive Logic Programming to Discover Rules Governing the Three-Dimensional Topology of Protein Structure. In *Proceedings of the 8th International Workshop on Inductive Logic Programming*, ILP 1998, pages 53–64, London, UK, UK, 1998. Springer-Verlag. doi: 10.1007/BFb0027310. First Online: 2025-01-01; Last access: 2025-03-01. 25, 26

[74] K. T. Simons, C. Kooperberg, E. H., and D. Baker. Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and Bayesian scoring functions. *Journal of molecular biology*, 268 (1):209–225, 1997. doi: 10.1006/jmbi.1997.0959. Last access: 2025-03-01. 27

[75] C. A. Rohl, C. E. M. Strauss, D. Chivian, and D. Baker. Modeling structurally variable regions in homologous proteins with rosetta. *Proteins Structure Function and Bioinformatics*, 55(3):656–677, 2004. doi: 10.1002/prot.10629. Last access: 2025-03-01. 27

[76] Evan T. Powers and Lila M. Gierasch. The Proteome Folding Problem and Cellular Proteostasis. *Journal of molecular biology*, 433(20):167197–167197, 2021. doi: 10.1016/j.jmb.2021.167197. Last access: 2025-03-01. 28

[77] Letícia M. F. Bertoline, Angélica N. Lima, Jose E. Krieger, and Samantha K. Teixeira. Before and after AlphaFold2: An overview of protein structure

prediction. *Frontiers in Bioinformatics*, 3, 2023. ISSN 2673-7647. doi: 10.3389/fbinf.2023.1120370. Last access: 2025-03-01. 28

[78] Sebastián Aliaga-Rojas, Manuel Villalobos-Cid, Márcio Dorn, and Mario Inostroza-Ponta. A multi-objective approach for the protein structure prediction problem. In *40th International Conference of the Chilean Computer Science Society, SCCC 2021, La Serena, Chile, November 15-19, 2021*, pages 1–8. IEEE, 2021. doi: 10.1109/SCCC54552.2021.9650383. Last access: 2025-03-01. 29

[79] D. Chyvian, T. Robertson, R. Bonneua, and D. Baker. *Ab initio* methods. *Structural Bioinformatics*, 44:547, 2003. doi: 10.1002/0471721204.ch27. Last access: 2025-03-01. 29, 32

[80] D. Osguthorpe. *Ab initio* protein folding. *Current Opinion in Structutal Biology*, 10(2):146–152, 2000. doi: 10.1016/S0959-440X(00)00067-1. Last access: 2025-03-01. 29, 32

[81] Bernard Monasse and Frédéric Boussinot. Determination of Forces from a Potential in Molecular Dynamics. Technical report, Mines Paris - PSL - HAL, Jan 2014. URL https://minesparis-psl.hal.science/hal-00924263. 11 pages, Last access: 2025-03-01. 30

[82] D.C. Rapaport. *The Art of Molecular Dynamics Simulation*. Cambridge University Press, cambridge, 2nd edition, 2004. doi: 10.1017/CBO9780511816581. Last access: 2025-03-01. 31

[83] B. J. Alder and T. E. Wainwright. Phase Transition for a Hard Sphere System. *The Journal of Chemical Physics*, 27(5):1208–1209, 1957. URL http://aip.scitation.org/doi/abs/10.1063/1.1743957. Last access: 2025-03-01. 31

[84] B. J. Alder and T. E. Wainwright. Studies in Molecular Dynamics. I. General Method. *The Journal of Chemical Physics*, 31(2):459–466, 1959. doi: 10.1063/1.1730376. Last access: 2025-03-01. 31

[85] Roberto Car and Michele Parrinello. Unified Approach for Molecular Dynamics and Density-Functional Theory. *Physical Review Letters*, 55:2471–2474, Nov 1985. doi: 10.1103/PhysRevLett.55.2471. Last access: 2025-03-01. 31

[86] J. Andrew McCammon, Bruce R. Gelin, and Martin Karplus. Dynamics of folded proteins. *Nature*, 267(5612):585–590, jun 1977. doi: 10.1038/267585a0. Last access: 2025-03-01. 32

[87] Tomas Hansson, Chris Oostenbrink, and Wilfred F. van Gunsteren. Molecular dynamics simulations. *Current Opinion in Structural Biology*, 12:190–196, 2002. doi: 10.1016/s0959-440x(02)00308-1. Last access: 2025-03-01. 32

[88] R. Salomon-Ferrer, D. A. Case, and R. C. Walker. An overview of the Amber biomolecular simulation package. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 3(2):198–210, 2012. doi: doi.org/10.1002/wcms.1121. Last access: 2025-03-01. 32

[89] Aviezri S. Fraenkel. Complexity of protein folding. *Bulletin of Mathematical Biology*, 55(6):1199–1210, 1993. 32

[90] A. M. Namba, V. B. da Silva, and C. H. T. P. da Silva. Dinâmica: teoria e aplicações em planejamento de fármacos. *Eclética Química*, 33:13–23, 12 2008. doi: 10.1590/S0100-46702008000400002. Last access: 2025-03-01. 32, 36

[91] Pramod C. Nair and John O. Miners. Molecular dynamics simulations: from structure function relationships to drug discovery. *In Silico Pharmacology*, 2 (1):1–4, November 2014. doi: 10.1186/s40203-014-0004-8. Last access: 2025-03-01. 32

[92] Mike P Allen and Dominic J Tildesley. *Computer simulation of liquids*. Oxford university press, 1987. doi: 10.1093/oso/9780198803195.001.0001. Last access: 2025-03-01. 32

[93] Wilfred F. van Gunsteren and Herman J. C. Berendsen. Computational Simulation of Molecular Dynamics: Methodology, Applications and Perspectives in Chemistry. *Angewandte Chemie International Edition in English*, 29(09): 992–1023, sep 1990. doi: 10.1002/anie.199009921. Last access: 2025-03-01. 32

[94] Roland Stote, Annick Dejaegere, Dmitry Kuznetsov, and Laurent Falquet. Theory of Molecular Dynamics. Website, October 1999. URL http://www.ch.embnet.org/MD_tutorial/. ExPASy Bioinformatics Resource Portal. Last access: 2025-03-01. 32

[95] B. D. Madej and R. Walker. An Introduction to Molecular Dynamics Simulations using AMBER. http://www.ambermd.org, 2014. URL https://ambermd.org/tutorials/basic/tutorial0/index.php. Last access: 2023-04-20. 33, 36

[96] Michael P. Allen. Introduction to Molecular Dynamics Simulation. In Nobert Attig, Kurt Bindeer, Helmut Grubmuller, and Kurt Kremer, editors, *Computational Soft Matter: From Synthetic Polymers to Proteins, Lectures Notes*, volume 23 of *NIC Series*, pages 1–28. John von Neumann Institute for Computing, Jülich, February 2004. ISBN 3-00-012641-4. Last access: 2025-03-01. 34

[97] Tanner, David E. and Chan, Kwok-Yan and Phillips, James C. and Schulten, Klaus. Parallel Generalized Born Implicit Solvent Calculations with NAMD. *Journal of Chemical Theory and Computation*, 7(11):3635–3642, 2011. doi: 10.1021/ct200563j. Last access: 2025-03-01. 36

[98] Vijay S. Pande, Eric J. Sorin, Christopher D. Snow, and Young Min Rhee. Computer Simulations of Protein Folding. In *Protein Folding, Misfolding and Aggregation: Classical Themes and Novel Approaches*. The Royal Society of Chemistry, 06 2008. ISBN 978-0-85404-257-9. doi: 10.1039/9781847558282-00161. Last access: 2025-03-01. 37

[99] David E. Shaw, Paul Maragakis, Kresten Lindorff-Larsen, Stefano Piana, Ron O. Dror, Michael P. Eastwood, Joseph A. Bank, John M. Jumper, John K. Salmon, Yibing Shan, and Willy Wriggers. Atomic-Level Characterization of the Structural Dynamics of Proteins. *Science*, 330(6002):341–346, 2010. doi: 10.1126/science.1187409. Last access: 2025-03-01. 38

[100] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, Version 3.0. Technical report, MPI Forum, September 2012. Last access: 2025-03-01. 38

[101] Juekuan Yang, Yujuan Wang, and Yunfei Chen. GPU accelerated molecular dynamics simulation of thermal conductivities. *Journal of Computational Physics*, 221(2):799–804, 2007. doi: 10.1016/j.jcp.2006.06.039. Last access: 2025-03-01. 38

[102] Weiguo Liu, Bertil Schmidt, Gerrit Voss, and Wolfgang Müller-Wittig. Accelerating molecular dynamics simulations using Graphics Processing Units with CUDA. *Computer Physics Communications*, 179(9):634–641, 2008. doi: 10.1016/j.cpc.2008.05.008. Last access: 2025-03-01. 38

[103] Peter Eastman, Jason Swails, and John D. Chodera et al. OpenMM 7: Rapid development of high performance algorithms for molecular dynamics. *PLOS Computational Biology*, 13(7):1–17, 07 2017. doi: 10.1371/journal.pcbi.1005659. Last access: 2025-03-01. 38, 65

[104] James C. Phillips, David J. Hardy, and Julio D. C. Maia et al. Scalable molecular dynamics on CPU and GPU architectures with NAMD. *The Journal of Chemical Physics*, 153(4), 2020. doi: 10.1063/5.0014475. Last access: 2025-03-01. 38, 39, 76

[105] Abhinav Bhatelé and Sameer Kumar and Chao Mei and James C. Phillips and Gengbin Zheng and Laxmikant V. Kale. Overcoming Scaling Challenges in Biomolecular Simulations across Multiple Platforms. In *IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, pages 1–12. IEEE, 2008. doi: 10.1109/IPDPS.2008.4536317. Last access: 2025-03-01. 38

[106] James C. Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D. Skeel, Laxmikant Kalé, and Klaus Schulten. Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry*, 26(16):1781–1802, 2005. doi: 10.1002/jcc.20289. Last access: 2025-03-01. 38, 41, 42

[107] Mark Nelson, William Humphrey, Attila Gursoy, Andrew Dalke, Laxmikant Kalé, Robert D. Skeel, and Klaus Schulten. NAMD: A parallel, object-oriented molecular dynamics program. *The International Journal of Supercomputer Applications and High Performance Computing*, 10(4):251–268, 1996. doi: 10.1177/109434209601000401. Last access: 2025-03-01. 38

[108] James Phillips, Robert Brunner, Aritomo Shinozaki, Milind Bh, Neal Krawetz, Robert Skeel, and Klaus Schulten. Avoiding Algorithmic Obfuscation in a Message-Driven Parallel MD Code. In *Computational Molecular Dynamics: Challenges, Methods, Ideas*, pages 472–482, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. doi: 10.1007/978-3-642-58360-5_28. Last access: 2025-03-01. 38, 41, 42, 107

[109] Laxmikant V Kale, Robert Skeel, Milind Bhandarkar, Robert Kraemer Brunner, Attila Gursoy, Neal Krawetz, James Phillips, Aritomo Shinozaki, Krishnan Varadarajan, and Klaus J Schulten. NAMD2: Greater Scalability for Parallel Molecular Dynamics. *Journal of Computational Physics*, 151(1): 283–312, 5 1999. ISSN 0021-9991. doi: 10.1006/jcph.1999.6201. Last access: 2025-03-01. 38, 39, 40, 42

[110] James C. Phillips. Attacking HIV with Petascale Molecular Dynamics Simulations on Titan and Blue Waters. http://on-demand.gputechconf.com/gtc/2015/presentation/S5226-Ross-Walker.pdf, march 2015. Last access: 2025-03-01. 38

[111] Laxmikant V. Kalé and Sanjeev Krishnan. Charm++: Parallel programming with message-driven objects. *Parallel Programming using C++*, pages 175–213, 1996. Last access: 2025-03-01. 39, 41

[112] Laxmikant V. Kale, Eric Bohm, Celso L. Mendes, Terry Wilmarth, and Gengbin Zheng. Programming Petascale Applications with Charm++ and AMPI. In D. Bader, editor, *Petascale Computing: Algorithms and Applications*, pages 421–441. Chapman & Hall / CRC Press, 2008. ISBN 9780367387891. Last access: 2025-03-01. 39

[113] James C. Phillips, John E. Stone, and Klaus Schulten. Adapting a Message-driven Parallel Application to GPU-accelerated Clusters. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 8:1–8:9, Piscataway, NJ, USA, 2008. IEEE Press. ISBN 978-1-4244-2835-9. doi: 10.1109/SC.2008.5214716. Last access: 2025-03-01. 41, 42

[114] James C. Phillips, Yanhua Sun, Nikhil Jain, Eric J. Bohm, and Laximant V. Kale. Mapping to Irregular Torus Topologies and Other Techniques for Petascale Biomolecular Simulation. In *Proceedings of ACM/IEEE SC 2014*, pages 81–91, New Orleans, Louisiana, November 2014. doi: 10.1109/SC.2014.12. Last access: 2025-03-01. 42, 85

[115] Diogo C. Ferreira, Marx G. van der Linden, Leandro C. de Oliveira, José N. Onuchic, and Antônio F. Pereira de Araújo. Information and redundancy in

the burial folding code of globular proteins within a wide range of shapes and sizes. *Proteins: Structure, Function, and Bioinformatics*, 84(4):515–531, 2016. doi: doi.org/10.1002/prot.24998. Last access: 2025-03-01. 42

[116] Marx Gomes van der Linden. *Simulação do enovelamento de proteínas com potenciais de enterramentos atômicos dependentes da sequência*. Phd. thesis, University of Brasília - UnB, Brasilia, Brazil, December 2013. Last access: 2025-03-01. 43, 44

[117] 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*, 81(8):3684–3690, 10 1984. ISSN 0021-9606. doi: 10.1063/1.448118. Last access: 2025-03-01. 43

[118] Daniele Archibugi and Andrea Filippetti. *The Handbook of Global Science, Technology, and Innovation*. HGP - Handbooks of Global Policy. John Wiley & Sons, jun 2015. ISBN 9781118738962. URL https://books.google.com.br/books?id=fLIOCgAAQBAJ. Last access: 2025-03-01. 46

[119] Richard S. Segall, Jeffey S. Cook, and Qingyu Zhang. *Research and Applications in Global Supercomputing*. Advances in Systems Analysis, Software Engineering, and High Performance Computing. IGI Global, jan 2015. ISBN 9781466674622. URL httpsb//books.google.com.br/books?id=5CvhBgAAQBAJ. Last access: 2025-03-01. 46

[120] Jack Dongarra and Top500 Team. TOP500 Supercomputer Sites - November 2024. https://top500.org/lists/top500/2024/11, 2024. Last access: 2025-03-01. 46, 47, 49, 51

[121] Jack Dongarra, Piotr Luszczek, and Antoine Petitet. The LINPACK Benchmark: past, present and future. *Concurrency and Computation: Practice and Experience*, 15(9):803–820, 2003. doi: doi.org/10.1002/cpe.728. Last access: 2025-03-01. 46, 47

[122] Jack J. Dongarra. Performance of Various Computers Using Standard Linear Equations Software. Computer Science Technical Report Number CS - 89 - 85, Oak Ridge National Laboratory, University of Tennessee, Knoxville TN, 37996, Jun 2014. Last access: 2025-03-01. 46, 47

[123] Ada Sedova, Russ Davidson, Mathieu Taillefumier, and Wael Elwasif. HPC Molecular Simulation Tries Out a New GPU: Experiences on Early AMD Test Systems for the Frontier Supercomputer. *CUG2022 Conference*, 2(1), 6 2022. URL https://www.osti.gov/biblio/1883870. Last access: 2025-03-01. 47

[124] Jaewoon Jung, Chigusa Kobayashi, and Yuji Sugita. Acceleration of generalized replica exchange with solute tempering simulations of large biological systems on massively parallel supercomputer. *Journal of Computational Chemistry*, 44(20):1740–1749, 2023. doi: 10.1002/jcc.27124. Last access: 2025-03-01. 47, 61, 62, 63

[125] Gianluca Palermo, Gianmarco Accordi, Davide Gadioli, and et al. Tunable and Portable Extreme-Scale Drug Discovery Platform at Exascale:the LIGATE Approach. *20th ACM International Conference on Computing Frontiers - CF'23*, abs/2304.09953, Apr 2023. doi: 10.48550/arXiv.2304.09953. Last access: 2025-03-01. 47

[126] EuroHPC Joint Undertaking. The EuroHPC Ju Supercomputers Analysis of the Petascale and Pre exascale systems. Technical report, EuroHPC JU, Sep 2021. URL https://eurohpc-ju.europa.eu/system/files/2023-07/EuroHPC%20Systems%20Report-Sep2021.pdf. Last access: 2025-03-01. 47, 52, 53

[127] Justin S. Smith, Benjamin Nebgen, Nithin Mathew, Jie Chen, and et al. Automated discovery of a robust interatomic potential for aluminum. *Nature communications*, 12(1):1257–1257, 2021. doi: 10.48550/arXiv.2003.04934. Last access: 2025-03-01. 47

[128] Xiaohui Duan, Qi Shao, Junben Weng, Bertil Schmidt, Lin Gan, Guohui Li, Haohuan Fu, Wei Xue, Weiguo Liu, and Guangwen Yang. Bio-ESMD: A Data Centric Implementation for Large-Scale Biological System Simulation on Sunway TaihuLight Supercomputer. *IEEE transactions on parallel and distributed systems*, 34(3):881–893, 2023. doi: 10.1109/TPDS.2022.3220559. Last access: 2025-03-01. 47

[129] Albert Musaelian, Anders Johansson, Simon Batzner, and Boris Kozinsky. Scaling the leading accuracy of deep equivariant models to biomolecular simulations of realistic size. *preprint arXiv:2304.10061*, 2023. doi: 10.1145/3581784.3627041. Last access: 2025-03-01. 47

[130] Olivier Adjoua, Louis Lagardère, Luc-Henri Jolly, Arnaud Durocher, Thibaut Very, Isabelle Dupays, Zhi Wang, Théo Jaffrelot Inizan, Frédéric Célerse, Pengyu Ren, Jay W. Ponder, and Jean-Philip Piquemal. Tinker-HP: Accelerating Molecular Dynamics Simulations of Large Complex Systems with Advanced Point Dipole Polarizable Force Fields Using GPUs and Multi-GPU Systems. *Journal of Chemical Theory and Computation*, 17(4):2034–2053, 2021. doi: 10.1021/acs.jctc.0c01164. PMID: 33755446, Last access: 2025-03-01. 47

[131] Jack J. Dongarra. Report on the Tianhe-2A System. Tech Report No. ICL-UT-17-04, Oak Ridge National Laboratory, University of Tennessee, Knoxville, September 2017. Last access: 2025-03-01. 47, 56, 57

[132] Javier García-Marín, Diego Rodríguez-Puyol, and Juan J. Vaquero. Insight into the mechanism of molecular recognition between human Integrin-Linked Kinase and Cpd22 and its implication at atomic level. *Journal of computer-aided molecular design*, 36(8):575–589, 2022. doi: 10.1007/s10822-022-00466-1. Last access: 2025-03-01. 47

[133] Chenle Yu, Sara Royuela, and Eduardo Qui nones. Enhancing Heterogeneous Computing Through OpenMP and GPU Graph. In *Proceedings of the 53rd International Conference on Parallel Processing*, ICPP '24, page 534–543, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400717932. doi: 10.1145/3673038.3673050. Last access: 2025-03-01. 47

[134] Fei Yin and Feng Shi. A Comparative Survey of Big Data Computing and HPC: From a Parallel Programming Model to a Cluster Architecture. *International journal of parallel programming*, 50(1):27–64, 2022. doi: 10.1007/s10766-021-00717-y. Last access: 2025-03-01. 48, 51

[135] John L. Hennessy and David A. Patterson. *Computer Architecture, Sixth Edition: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 6th edition, 2017. ISBN 0128119055. Last access: 2025-03-01. 49

[136] Jack Dongarra, Thomas Sterling, Horst Simon, and Erich Strohmaier. High-performance computing: clusters, constellations, MPPs, and future directions. *Computing in Science and Engineering*, 7(2):51–59, 2005. doi: 10.1109/MCSE.2005.34. Last access: 2025-03-01. 49

[137] Hassan A. Karimi. *Big Data: Techniques and Technologies in Geoinformatics*. CRC Press, Taylor & Francis Group, feb 2014. ISBN 9781466586512. URL `httpse//books.google.com.br/books?id=BJGlAgAAQBAJ`. Last access: 2025-03-01. 49, 51

[138] Li Bo, Zhou Zhenliu, and Wang Xiangfeng. A Survey of HPC Development. In *2012 International Conference on Computer Science and Electronics Engineering*, pages 103–106. IEEE, 2012. doi: 10.1109/ICCSEE.2012.130. Last access: 2025-03-01. 49, 51

[139] Jack J. Dongarra and Al Geist. Report On The Oak Ridge National Laboratory's Frontier System. Tech Report No. ICL-UT-22-05, Oak Ridge National Laboratory, University of Tennessee, Knoxville, may 2022. Last access: 2025-03-01. 49, 50

[140] Kawthar Shafie Khorassani, Chen-Chun Chen, Bharath Ramesh, Aamir Shafi, Hari Subramoni, and Dhabaleswar K. Panda. High Performance MPI over the Slingshot Interconnect. *Journal of computer science and technology*, 38(1):128–145, 2023. doi: 10.1145/3491418.3530773. Last access: 2025-03-01. 49, 50

[141] Ping-Jing Lu, Ming-Che Lai, and Jun-Sheng Chang. A Survey of High-Performance Interconnection Networks in High-Performance Computer Systems. *Electronics (Basel)*, 11(9):1369, 2022. doi: 10.3390/electronics11091369. Last access: 2025-03-01. 50

[142] Cluster File Systems Inc. Lustre : A Scalable , High-Performance File System Cluster. In *Available at https://www.lustre.org*, 2023. URL https://cse.buffalo.edu/faculty/tkosar/cse710/papers/lustre-whitepaper.pdf. Last access: 2025-03-01. 52

[143] John Kim, Wiliam J. Dally, Steve Scott, and Dennis Abts. Technology-Driven, Highly-Scalable Dragonfly Topology. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ISCA '08, page 77–88, USA, 2008. IEEE Computer Society. doi: 10.1109/ISCA.2008.19. Last access: 2025-03-01. 53

[144] Barcelona Supercomputing Center. Marenostrum4 User's Guide. Tutorial, Barcelona SuperComputing Center, Barcelona Supercomputing Center, november 2020. Last access: 2025-03-01. 53, 83, 117

[145] D. E. Womble, M. Shankar, W. Joubert, J. T. Johnston, J. C. Wells, and J. A. Nichols. Early experiences on Summit: Data analytics and AI applications. *IBM Journal of Research and Development*, 63(6):2:1–2:9, 2019. doi: 10.1147/JRD.2019.2944146. Last access: 2025-03-01. 55

[146] Xiangke Liao, Liquan Xiao, Canqun Yang, and Yutong Lu. MilkyWay-2 supercomputer: system and application. *Frontiers of Computer Science*, 8(3): 345–356, 2014. ISSN 2095-2228. doi: 10.1007/s11704-014-3501-3. Last access: 2025-03-01. 56, 57

[147] Jack J. Dongarra and Al Geist. Report on the Sunway TaihuLight System. Tech Report UT-EECS-16-742, Oak Ridge National Laboratory, University of Tennessee, Knoxville, june 2016. URL https://www.netlib.org/utk/people/JackDongarra/PAPERS/sunway-report-2016.pdf. Last access: 2023-07-06. 58

[148] Steve Plimpton. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *Journal of Computational Physics*, 117(1):1–19, 1995. ISSN 0021-9991. doi: 10.1006/jcph.1995.1039. Last access: 2025-03-01. 58, 60

[149] Ruud A. Haring, Martin Ohmacht, Thomas W. Fox, Michael K. Gschwind, David L. Satterfield, Krishnan Sugavanam, Paul W. Coteus, Philip Heidelberger, Matthias A. Blumrich, Robert W. Wisniewski, Alan Gara, George Liang-Tai Chiu, Peter A. Boyle, Norman H. Chist, and Changhoan Kim. The IBM Blue Gene/Q Compute Chip. *IEEE MICRO*, 32(2):48–60, 2012. doi: 10.1109/MM.2011.108. Last access: 2025-03-01. 60

[150] Jack J. Dongarra. Report on the Fujitsu Fugaku System. Tech Report No. ICL-UT-20-06 ICL-UT-20-06, Oak Ridge National Laboratory, University of Tennessee, Knoxville, June 2020. Last access: 2025-03-01. 61, 62

[151] Dong Choi, Glenn Lockwood, Robert Sinkovits, and Mahidhar Tatineni. Performance of Applications using Dual-Rail InfiniBand 3D Torus network on the Gordon Supercomputer. In *Proceedings of the 2014 Annual Conference*

on extreme science and engineering discovery environment, pages 1–6. ACM, 2014. ISBN 9781450328937. 66, 67

[152] Y. Miao, V. A. Feher, and J. A. McCammon. Gaussian Accelerated Molecular Dynamics: Unconstrained Enhanced Sampling and Free Energy Calculation. *Journal of Chemical Theory and Computation*, 11(8):3584–3595, 2015. doi: 10.1021/acs.jctc.5b00436. Last access: 2025-03-01. 66

[153] Tirthak Patel, Suren Byna, Glenn K Lockwood, and et al. Uncovering access, reuse, and sharing characteristics of {I/O-Intensive} files on {Large-Scale} production {HPC} systems. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 91–101. USENIX Association, 2020. Last access: 2025-03-01. 68

[154] I.F. Sbalzarini and J.H. Walther and M. Bergdorf and et al. PPM – A highly efficient parallel particle–mesh library for the simulation of continuum systems. *Journal of Computational Physics*, 215(2):566–588, 2006. ISSN 0021-9991. doi: 10.1016/j.jcp.2005.11.017. Last access: 2025-03-01. 76

[155] Adam Liwo, Cezary Czaplewski, Adam K. Sieradzan, and et al. Theory and Practice of Coarse-Grained Molecular Dynamics of Biologically Important Systems. *Biomolecules*, 11(9), 2021. doi: 10.3390/biom11091347. Last access: 2025-03-01. 76

[156] Emerson A. Macedo and Alba C.M.A. Melo. Adaptive patch grid strategy for parallel protein folding using atomic burials with NAMD. *Journal of Parallel and Distributed Computing*, 189:104868, 2024. ISSN 0743-7315. doi: 10.1016/j.jpdc.2024.104868. Last access: 2025-03-01. 79, 95, 96, 114, 118

[157] M. Bhandarkar, A. Bhatele, and E. Bohm et al. NAMD User's Guide (Version 2.11). Tutorial, University of Illinois and Beckman Institute, University of Illinois at Urbana-Champaign, december 2015. Theoretical and Computational Biophysics Group; Last access: 2025-03-01. 81, 90

[158] A. Bhatele et al R. Bernardi, M. Bhandarkar. NAMD User's Guide (Version 2.14). Tutorial, University of Illinois and Beckman Institute, University of Illinois at Urbana-Champaign, august 2020. Theoretical and Computational Biophysics Group; Last access: 2025-03-01. 81, 90

[159] Barcelona Supercomputing Center. Nord III User's Guide. Tutorial, Barcelona SuperComputing Center, Barcelona Supercomputing Center, february 2021. Last access: 2025-03-01. 83, 89, 117

[160] RCSB Protein Data Bank. 1ENH: Engrailed Homeodomain. https://www.rcsb.org/structure/1ENH, 1998. Last access: 2025-03-01. 84

[161] RCSB Protein Data Bank. 1IFR: Lamin A/C Globular Domain. https://www.rcsb.org/structure/1IFR, 1997. Last access: 2025-03-01. 84

[162] RCSB Protein Data Bank. 1OZ9: AQ_1354, a hypothetical protein from Aquifex aeolicus. https://www.rcsb.org/structure/1OZ9, 2004. Last access: 2025-03-01. 84

[163] RCSB Protein Data Bank. 4LNZ: Human Myosin 5b globular domain. https://www.rcsb.org/structure/4LNZ, 2013. Last access: 2025-03-01. 84, 112

[164] Jing Huang, Sarah Rauscher, Grzegorz Nawrocki, and et al. CHARMM36m: an improved force field for folded and intrinsically disordered proteins. *Nature Methods*, 14:71, Nov 2016. URL https://doi.org/10.1038/nmeth.4067. Last access: 2025-03-01. 84

[165] NAMD Wiki: NamdPerformanceTuning. NAMD - Web site, 2015. URL https://www.ks.uiuc.edu/Research/namd/wiki/?NamdPerformanceTuning. Last access: 2025-03-01. 90

[166] NAMD: Class Hierarchy. https://www.ks.uiuc. edu/Research/namd/doxygen/hierarchy.html, 2020. Last access: 2025-03-01. 105, 106

[167] W. Humphrey, A. Dalke, and K. Schulten. VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics*, 14:33–38, 1996. doi: 10.1016/0263-7855(96)00018-5. Last access: 2025-03-01. 112

[168] I. Kufareva and R. Abagyan. Methods of protein structure comparison. In *Homology Modeling*, pages 231–257. Springer, 2011. doi: 10.1007/978-1-61779-588-6_10. Last access: 2025-03-01. 112

Appendix A

Article Derived from This Thesis

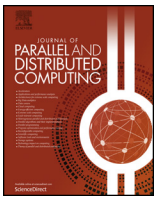
Emerson A. Macedo and Alba C.M.A. Melo. Adaptive patch grid strategy for parallel protein folding using atomic burials with NAMD. Journal of Parallel and Distributed Computing, 189:104868, 2024 (first page).

Journal of Parallel and Distributed Computing 189 (2024) 104868

Contents lists available at ScienceDirect

Journal of Parallel and Distributed Computing

journal homepage: www.elsevier.com/locate/jpdc



Adaptive patch grid strategy for parallel protein folding using atomic burials with NAMD

Emerson A. Macedo *, Alba C.M.A. Melo

Department of Computer Science, Campus UnB, Predio CIC/EST, University of Brasilia (UnB), 70910-00, Brasilia, DF, Brazil

ARTICLE INFO

Keywords:
Protein folding
Parallel simulation
Atomic burials

ABSTRACT

The definition of protein structures is an important research topic in molecular biology currently, since there is a direct relationship between the function of the protein in the organism and the 3D geometric configuration it adopts. The transformations that occur in the protein structure from the 1D configuration to the 3D form are called protein folding. *Ab initio* protein folding methods use physical forces to model the interactions among the atoms that compose the protein. In order to accelerate those methods, parallel tools such as NAMD were proposed. In this paper, we propose two contributions for parallel protein folding simulations: (a) adaptive patch grid (APG) and (b) the addition of atomic burials (AB) to the traditional forces used in the simulation. With APG, we are able to adapt the simulation box (patch grid) to the current shape of the protein during the folding process. AB forces relate the 3D protein structure to its geometric center and are adequate for modeling globular proteins. Thus, adding AB to the forces used in parallel protein folding potentially increases the quality of the result for this class of proteins. APG and AB were implemented in NAMD and tested in supercomputer environments. Our results show that, with APG, we are able to reduce the execution time of the folding simulation of protein 4LNZ (5,714 atoms, 15 million time steps) from 12 hours and 36 minutes to 11 hours and 8 minutes, using 16 nodes (256 CPU cores). We also show that our APG+AB strategy was successfully used in a realistic protein folding simulation (1.7 billion time steps).

1. Introduction

Once the synthesis ends, protein folding begins: the 1D sequence starts to fold on itself following a series of 3D transformations, changing its structure each time, until it finds a stable 3D configuration that determines its function in the organism [10]. Despite the advancements in experimental methods, there is a huge gap between the number of known 3D structures and the number of 1D sequences determined routinely [13]. This issue is addressed by simulations that predict the 3D structures.

The computational methods for protein folding can be classified into four groups. This paper focuses on the *ab initio* group, which does not use any data on previously resolved protein structures, i.e., predictions are based strictly on the principles of physics, seeking to predict the stable 3D configuration considering only the protein's 1D sequence. Molecular Dynamics (MD) simulation is a technique used to study the time-dependent behavior of molecular systems. It involves numerically solving Newton's equations of motion to track the positions and velocities of the atoms over time, considering their initial coordinates and velocities and a potential energy function of the system. For modeling protein structures, canonical MD is the method of choice. MD simulations are *ab initio* methods which, in general, consist of (i) a geometric representation of the 1D structure, (ii) a potential energy function, and (iii) an energy surface search technique [14], with energy surface search technique defined in [14]. One limitation of the *ab initio* methods is that finding the optimal sequence of geometric transformations which leads to the stable 3D configuration is an NP-Complete problem [17]. In this context, the energy surface search technique uses energy functions to simulate the protein conformational space, describing its energy and atomic interactions.

MD simulations are difficult to parallelize, thus the number of computing elements in parallel must be very well scaled to obtain some gain. NAMD (NAnoscale Molecular Dynamics program) [8,41] parallelizes the MD simulations using a hybrid decomposition method which has two components. The first component is responsible for spatial decomposition, i.e., for decomposing the simulation geometric space into

* Corresponding author.
E-mail addresses: 180061712@aluno.unb.br (E.A. Macedo), alves@unb.br (A.C.M.A. Melo).

https://doi.org/10.1016/j.jpdc.2024.104868
Received 12 January 2023; Received in revised form 17 February 2024; Accepted 25 February 2024
Available online 4 March 2024

Appendix B

NAMD Configuration file

```
#############################################################
## DATE    : 2025-02-11 07:11:04
## UPDATED BY: EMERSON DE A. MACEDO
#############################################################

#############################################################
## REFERENCES USED
# ref.1: www.ks.uiuc.edu/Training/Tutorials/namd/namd-tutorial-
#      unix-html/node10.html
#
# ref.2: www.ks.uiuc.edu/Research/namd/wiki/?NamdPerformanceTuning
#
# ref.3: www.ks.uiuc.edu/Research/2.11/ug/node92.html
#
#############################################################


#############################################################
## JOB DESCRIPTION
#############################################################

# Minimization and Equilibration of
# Protein 4lnz (Extended) in Generalized Born implicit solvent

# MOLECULAR SYSTEM
set mol_sys_name 4lnz_extendedA


#############################################################
## ADJUSTABLE PARAMETERS
#############################################################

structure     ${mol_sys_name}.psf
coordinates   ${mol_sys_name}.pdb

set temperature 310
#set outputname simulation

# restart: step 1
set previous  simulation1
set current   simulation2

# restart: step 2
set outputname ${current}

# restart: step 3
bincoordinates ${previous}.restart.coor
binvelocities ${previous}.restart.vel
extendedSystem ${previous}.restart.xsc

```

```
#coordinates ${mol_sys_name}.pdb ; # ignored & only "bincoordinates" is used
                          ; # for initializing the coordinates for the
                          ; # present configuration.

# restart: step 4
firsttimestep 5000000 ; # taken from the ${previous}.restart.xsc , which is the
                      # number of last saved restart configuration of the
                      # EARLIER SIMULATION

numsteps    10000000 ; # run stops when this step is reached


#############################################################
## BURIALS PARAMETERS
#############################################################

# central distances file
burials       rcntr_namd.dat


# hbons file (AF version)
hbondsAF      hbonds_namd.dat


#############################################################
## SIMULATION PARAMETERS
#############################################################

# Input
paraTypeCharmm on
parameters     par_all27_prot_lipid.prm

# restart: step 5
#temperature    ${temperature} ;# commented, because the option
                           ;# "binvelocities" is already specified

# Implicit Solvent [ref.1]
gbis           yes
alphaCutoff    12.0
ionConcentration 0.3 ; # default: 0.2

# Force-Field Parameters
exclude        scaled1-4
1-4scaling     1.0
cutoff         12.0
switching      on
switchdist     10.0
pairlistdist   14.0


# Integrator Parameters
timestep       1.0 ; # 2fs/step = default
rigidBonds     all ; # needed for 2fs steps
nonbondedFreq  1
fullElectFrequency 2 ; # irreleant if PME = no
stepspercycle  20  ; # default (ref.2)

# Constant Temperature Control
langevin       on   ; # do langevin dynamics
langevinDamping 1   ; # damping coefficient (gamma) of 1/ps
langevinTemp   ${temperature}
langevinHydrogen off ; # don't couple langevin bath to hydrogens

# Output
outputName     $outputname
#binaryoutput   no


restartfreq    2500000   ; # 10.000.000 / 2.500.000 = 4 restart points writen
dcdfreq        25000     ; # 10.000.000 / 25.000 = 400 frames
```

```
xstFreq        25000     ; # 10.000.000 / 25.000 = 400 frames
outputEnergies 50000     ; # 10.000.000 / 50.000 = 200 outputs -> every 2 frames, 1 output
outputPressure 50000     ; # 10.000.000 / 50.000 = 200 outputs -> every 2 frames, 1 output
outputTiming   50000     ; # 10.000.000 / 50.000 = 200 outputs -> every 2 frames, 1 output


# Improving Parallel Scaling # ref.3
# twoAwayX yes # "roughly doubles the number of patches"
# twoAwayY yes #
# twoAwayZ yes #



#############################################################
## EXECUTION SCRIPT
#############################################################

#############################################################
# restart: step 6: comment lines below
#############################################################
# :RMK: We don't want to run the 'minimization' process again
# (unless the earlier simulation run did not do it!)
#############################################################
# # Minimization
# minimize       100
# reinitvels     ${temperature}
#############################################################

run 5000000 ; # run = numsteps - firsttimestep
             # nr. steps to run this time (it overrides 'numsteps')
```

Appendix C

NAMD: Components and Files

#N	Components (Resource*)	Files
1	*DataTypeRepository**	structures.h, common.h
2	*Molecule*	Molecule.C, Molecule.h
3	*Parameters*	Parameters.C, Parameters.h
4	*SimParameters*	SimParameters.C, SimParameters.h
5	*ComputeMgr*	ComputeMgr.C, ComputeMap.h
6	*WorkDistrib*	WorkDistrib.C, WorkDistrib.h, WorkDistrib.ci
7	*LdbCoordinator*	LdbCoordinator.C, LdbCoordinator.decl.h, LdbCoordinator.h
8	*ReductionMgr*	ReductionMgr.h ReductionMgr.decl.h, ReductionMgr.C
09	*Broadcasts*	Broadcasts.h, BroadcastObject.h
10	*Sequencer*	Sequencer.C, Sequencer.h
9	*Controller*	Controller.C, Controller.h

Table C.1: List of NAMD components and their files used in this Thesis. The *Components* column contains the names of the components identified to build our strategy. The only exception is in the first line, *DataTypeRepository**. This name was created exclusively for this work, just to represent a resource, rather than a component.

Appendix D

NAMD Acknowledgment

"NAMD was developed by the Theoretical and Computational Biophysics Group in the Beckman Institute for Advanced Science and Technology at the University of Illinois at Urbana-Champaign."