2023

FPGA-based range-limited molecular dynamics acceleration

BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Dissertation

FPGA-BASED RANGE-LIMITED MOLECULAR DYNAMICS ACCELERATION

by

CHUNSHU WU

B.S., Dalian University of Technology, 2016
M.S., Brown University, 2018

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2023

Approved by

First Reader

Martin C. Herbordt, Ph.D.
Professor of Electrical and Computer Engineering

Second Reader

Richard C. Brower, Ph.D.
Professor of Electrical and Computer Engineering

Third Reader

Tali Moreshet, Ph.D.
Senior Lecturer and Research Assistant Professor of Electrical and Computer Engineering

Fourth Reader

Tong Geng, Ph.D.
Assistant Professor of Electrical and Computer Engineering and Computer Science
University of Rochester

Everybody is a genius. But if you judge a fish by its ability to climb a tree, it will live its whole life believing that it is stupid. Albert Einstein

Acknowledgments

I would like to express my deepest gratitude to all those who have supported and guided me throughout my journey in completing this Ph.D. thesis. Their contributions and encouragement have been invaluable, and I am truly grateful for their unwavering support.

First and foremost, I am indebted to my supervisor, Prof. Martin C. Herbordt, for his guidance, expertise, and patience. His insightful feedback, constant motivation, and unwavering belief in my abilities have been instrumental in shaping this research work. I am immensely grateful for his mentorship and for challenging me to reach new heights.

I extend my heartfelt appreciation to the members of my thesis committee, Prof. Richard C. Brower, Prof. Tali Moreshet, and Prof. Tong Geng, for their valuable insights, constructive criticism, and suggestions. Their expertise in their respective fields has immensely enriched this work and broadened my horizons.

I would like to express my special thanks to Prof. Tong Geng at University of Rochester and his family. Their substantial support in both research and life means a lot to me, especially in my early PhD career when I had little knowledge of high performance computing.

I am grateful to Dr. Ang Li at Pacific Northwest National Laboratory, for providing necessary resources and valuable advice to carry out this research. His commitment to excellence and dedication to fostering knowledge have been truly inspiring.

I am indebted to all my colleagues and friends, both within and outside our research group. For my current CAAD lab mates, Pouya Haghi, Anqi Guo, Sahan Bandara, Robert Mufano, Reza Sajjadinasab, Hafsah, Shahzad, and Zaid Tahir, thank you all for creating a harmonic and joyful research environment. For those who already graduated, Zihao Yuan, Chen Yang, Tianqi Wang, Anthony Ducimo, and Pierre-François Wolfe, they generously dedicated their time and efforts to help me conduct this research. Their camaraderie and encouragement have made this journey more fulfilling and enjoyable.

I am grateful to my family for their unwavering love, understanding, and constant support throughout this journey. Their belief in me and their sacrifices have been the driving force behind my accomplishments. To my parents Ying Zhang and Qinghong Wu, I owe my deepest gratitude for instilling in me a love for learning and for always being my pillars of strength.

I would like to take a special moment to express my deepest gratitude and appreciation to my wife, Yuqing Wang. Her patience, sacrifices, and belief in me have been the driving force behind my perseverance. Thank you, my dear wife, this achievement would not have been possible without you.

Lastly, I would like to express my heartfelt gratitude to all the authors, researchers, and scholars whose works have been referenced in this thesis. Their contributions to the field have been instrumental in shaping my research and expanding my knowledge.

To all those who have played a part, big or small, in the completion of this Ph.D. thesis, please accept my heartfelt appreciation and gratitude. Your support has been invaluable, and I am truly honored to have had the opportunity to work with and learn from each and every one of you.

The research that forms the basis of this dissertation has been partially funded by the NSF through Awards CCF-1618303/7960, CCF-1618303, and CCF-1919130; by the NIH through Award R44GM128533; by grants from Microsoft and Red Hat; by Xilinx and by Intel through donated FPGAs, tools, and IP; by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, ComPort: Rigorous Testing Methods to Safeguard Software Porting, under Award Number 78284. The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under Contract DE-AC05-76RL01830.

FPGA-BASED RANGE-LIMITED MOLECULAR DYNAMICS ACCELERATION

CHUNSHU WU

Boston University, College of Engineering, 2023

Major Professor: Martin C. Herbordt, PhD
Professor of Electrical and Computer Engineering

ABSTRACT

Molecular Dynamics (MD) is a computer simulation technique that executes iteratively over discrete, infinitesimal time intervals. It has been a widely utilized application in the fields of material sciences and computer-aided drug design for many years, serving as a crucial benchmark in high-performance computing (HPC). Numerous MD packages have been developed and effectively accelerated using GPUs. However, as the limits of Moore's Law are reached, the performance of an individual computing node has reached its bottleneck, while the performance of multiple nodes is primarily hindered by scalability issues, particularly when dealing with small datasets.

In this thesis, the acceleration with respect to small datasets is the main focus. With the recent COVID-19 pandemic, drug discovery has gained significant attention, and Molecular Dynamics (MD) has emerged as a crucial tool in this process. Particularly, in the critical domain of drug discovery, small simulations involving approximately $\sim$50K particles are frequently employed. However, it is important to note that small simulations do not necessarily translate to faster results, as long-term simulations comprising billions of MD iterations and more are essential in this context.

In addition to dataset size, the problem of interest is further constrained. Referred to as the most computationally demanding aspect of MD, the evaluation of range-limited (RL) forces not only accounts for 90% of the MD computation workload but also involves irregular mapping patterns of 3-D data onto 2-D processor networks. To emphasize, this thesis centers around the acceleration of RL MD specifically for small datasets.

In order to address the single-node bottleneck and multi-node scaling challenges, the thesis is organized into two progressive stages of investigation. The first stage delves extensively into enhancing single-node efficiency by examining various factors such as workload mapping from 3-D to 2-D, data routing, and data locality. The second stage focuses on studying multi-node scalability, with a particular emphasis on strong scaling, bandwidth demands, and the synchronization mechanisms between nodes.

Through our study, the results show our design on a Xilinx U280 FPGA achieves 51.72× and 4.17× speedups with respect to an Intel Xeon Gold 6226R CPU, and a Quadro RTX 8000 GPU. Our research towards strong scaling also demonstrates that 8 Xilinx U280 FPGAs connected to a switch achieves 4.67× speedup compared to an Nvidia V100 GPU[1].

[1] A100 is outperformed by V100 because it runs at lower frequency (1410 MHz vs. 1530 MHz); more GPUs result in even worse performance compared to 1 GPU.

Contents

1 Introduction 1

2 Background 10

2.1 The Physical Model of Range-Limited Forces 10

2.1.1 RL Force Features 10

2.1.2 Force Integration 12

2.1.3 Periodic Boundary Condition 13

2.2 MD Data Structure 13

2.2.1 Neighbor List vs. Cell List 14

2.2.2 Filtering 15

2.2.3 Neighbor Data Importing Layout 16

2.3 FPGA Background 17

2.3.1 FPGA Architecture 17

2.3.2 FPGAs in Molecular Dynamics 19

2.3.3 FPGAs in High Performance Computing 21

3 High-Level Methodologies 22

3.1 Space Decomposition 22

3.1.1 Assumptions and Parameter Declaration 23

3.1.2 The Model 24

3.1.3 Quantitative Analysis for the Import Volume 26

3.1.4 Latency Estimation 27

3.2 Memory to PE Mapping Schemes 29

3.2.1 All PEs Work on the Same Reference Particle 30
3.2.2 All PEs Work on the Same Home Cell 32
3.2.3 Each PE Works on a Different Cell 33
3.3 Data Format 34
3.3.1 Position Data Format 34
3.3.2 Force-Related Data Format 37
3.4 Summary 38
4 Single-FPGA Architectures 40
4.1 Introduction 40
4.2 Baseline Architectures 42
4.2.1 Design 1: Particle Centric 42
4.2.2 Design 2: Cell Centric 44
4.2.3 Design 3: Uniform Spread 46
4.2.4 Motion Update and Particle Migration 48
4.3 Optimized Designs 49
4.3.1 Optimized Design 1: Transposed Memory Blocks 50
4.3.2 Optimized Design 2: On-chip Ring Network 54
4.3.3 General Summary of the Optimized Designs 66
4.4 Evaluation 67
4.4.1 Experiment Setup 67
4.4.2 Performance and Comparison 67
4.4.3 Evaluating Design Options 71
4.4.4 Benchmarking 73
4.4.5 Hardware Utilization 76
4.5 Related Work 76
4.6 Conclusion 77

5 Bandwidth Analysis of FPGAs on a 3-D Torus 78

5.1 Introduction 78

5.2 Multi-chip baseline design 80

5.2.1 Prior-art Single-chip Design 80

5.2.2 Baseline Multi-chip Design 80

5.2.3 Communication Bottlenecks of Baseline 82

5.3 Optimized Multi-chip Design 83

5.3.1 Neighbor Data Caching 84

5.3.2 Routing Configuration 86

5.3.3 Motion Update Conflict 90

5.4 Evaluation 91

5.4.1 Evaluation of Routing Configuration 92

5.4.2 Evaluation of Neighbor Data Caching 94

5.4.3 Overall Performance Evaluation 95

5.5 Related Work 96

5.6 Conclusion 96

6 Optimized Mappings for Symmetric Force Calculations on FPGAs 98

6.1 Introduction 98

6.2 Design 101

6.2.1 Logical Topology 101

6.2.2 Corner Caches and Overlapping Position Caches 103

6.2.3 The Modified Manhattan Method 103

6.2.4 Cell Cache Partitioning and Corner Particle Pre-checking 104

6.2.5 Architecture 105

6.2.6 Memory Misalignment 108

6.2.7 Multi-chip Solution 109

6.3 Evaluation . . . 110
6.3.1 Performance . . . 111
6.3.2 Filtering Rates . . . 112
6.3.3 Position Input Ring Latency . . . 112
6.3.4 Motion Update Ring Latency . . . 113
6.3.5 Multi-FPGA Data Transfer . . . 114
6.3.6 Hardware Resource Usage . . . 116
6.4 Related Work . . . 116
6.5 Conclusion . . . 117
7 FASDA: An FPGA-Aided, Scalable and Distributed Accelerator for Range-Limited Molecular Dynamics 118
7.1 Introduction . . . 118
7.2 Scalable Architecture . . . 122
7.2.1 Hyperring-like Communication Topology . . . 122
7.2.2 Cell ID Conversion . . . 124
7.2.3 Communication Interface . . . 125
7.2.4 Chained Synchronization . . . 126
7.2.5 PE Scaling . . . 128
7.2.6 CBB Scaling . . . 130
7.3 Evaluation . . . 132
7.3.1 Experimental Setup . . . 132
7.3.2 Overall Performance . . . 132
7.3.3 Utilization Breakdown . . . 135
7.3.4 Communication Intensity . . . 136
7.3.5 Resources Consumption . . . 138
7.3.6 Energy Conservation . . . 139

7.4 Related Work 139

7.5 Conclusion 140

8 Conclusions and Future Work 142

8.1 Conclusions 142

8.2 Future Work 143

References 145

Curriculum Vitae 162

List of Tables

4.1 FPGA-O2 performance compared to FPGA-O1, GPU, and CPU with up to 32 threads. 74

4.2 FPGA-O2 performance of large simulation spaces compared to FPGA-O1, GPU, and 32-thread CPU. 75

4.3 Hardware Utilization of FPGA-O2 with spatial configurations. 76

5.1 Bandwidth Demand - Pillar (Unit: $w_d f$) 87

5.2 Bandwidth Demand - Block (Unit: $w_d f$) 88

5.3 Throughput of a 2×2×2 FPGA cluster. The design runs at 350 MHz frequency, with 50 particles in each cell. Particle: Liquid Argon. Cutoff radius: 8.5 Å. Number of ports: 6. Bandwidth of each port: 100 Gbps. Data size per packet: 120 bits 94

6.1 Hardware Costs 116

7.1 Hardware Utilization of All Design Variations 138

List of Figures

2·1 Fundamentals of R_c and the cell list method. (a) The cutoff regions of four particles. Only A and B interact with each other. (b) A simulation space divided into 3×4×4 cells with side length the same as R_c. (c) The home cell of particle A and 26 adjacent cells unfolded from (b). The green cells are regarded as neighbor cells. 11

2·2 The 2-D illustration of two filtering methods and the criterion. 16

2·3 The mechanism of the traditional Manhattan method shown in 2-D. The particle pairs are evaluated by the processor located in orange cells for 3 particle pair scenarios: (a) both in same cell; (b) in neighboring cells sharing a boundary; (c) in neighboring cells sharing a corner. 17

3·1 Neighbor cells in 2-D. 25

3·2 Import volume in 3-D. A/B/C: face/edge/corner. 25

3·3 A variation of Gauss Circle Problem. 26

3·4 Latency analysis with respect to cell granularity. 28

3·5 High level memory-to-PE mapping schemes. (a): particle-centric, all PEs work on a single reference particle at a time. (b):cell-centric, all PEs work on a same cell at a time. (c): uniformly-spreaded, each PE works on a separate cell. 31

3·6 The data format and precision analysis. Top: The fixed-point position format. (a) and (b): Two methods of computing the displacements of particles. (c) and (d): Error analysis of the two methods with different numbers of bits. 36

3·7 The interpolation method. α is a positive integer too large for $r^{-\alpha}$ to be computed. In our case, α is 8 or 14. The small r region is excluded due to non-physical high energy. 38

4·1 The baseline designs and detailed distributor architectures. (a)-(c): baseline design layouts. (d)-(g): details of position and force distributors. 45

4·2 Particle migration handling methods. (a): directly searching for empty slots. (b): the one-pointer method. (c): double buffering. 49

4·3 The optimized architecture for the first mapping scheme. (a): the data flow in force evaluation. For simplicity, arbiters and force computational units are abstracted as entire blocks, and the forces read from force caches for accumulation are omitted. (b): the new memory layout compared to baseline. (c): particle migration handling in motion update. P is the particle being migrated from cell 2 to cell 3. 51

4·4 RL dataflow in brief with rings. Yellow: Memory blocks; orange: Routing rings; blue: computing units. 55

4·5 The ring routing path. Position input ring: from position caches to PEs. Force output ring: from PEs to force caches. Motion update ring: from motion update units to position caches and velocity caches. 55

4·6 Example cases of (a) high filter pass rate, (b) low filter pass rate in 2D illustration. 58

4·7 The optimized design for the third mapping scheme. (a): The data path of a single force computation pipeline. (b): Particle dispatching layout with duo registers. AR: Active Register; BR: Backup Register. (c): The partial force caches and force aggregation, where the two PEs work on a same home cell, and the purple paths are to resolve the conflict when two force output rings are used. 59

4·8 (a): The 1st level filtering in a position input ring node. Dashed line: planar filter criterion. (b): The flowchart of a position input ring node. 61

4·9 The out-of-order broadcast method. Gray boxes: Empty slots. Boxes with "x": Marked as used. (a) The 1st cycle. (b) The 5th cycle. (c) The 8th cycle. (d) the 9th cycle. Red arrows: home particle position; blue arrows: neighbor particle position. 62

4·10 Particle migration on a linked list with a table showing the pointer values. PT points at the last particle, LT points at the last slot of the list, and NT points at the next unused slot to be allocated. 63

4·11 The solution to the problems emerge in large scale simulations. We intuitively use 4×4 2-D cells with eight PEs for example. (a): blue cells: the cells currently being processed; orange cells: the extra cells whose data are required to process the blue cells. In practice, particles from cell A are broadcast to all B cells through a position input ring for particle pairing. (b): the memory structure used to achieve higher memory utilization. The two red cells are sharing the same memory block at different address domains. (c): the double buffers are restored to avoid unnecessary memory traversal. Two neighbor particles (NP1 and NP2) require data from different cells. 65

4·12 The performance in cycles per iteration of three baseline and two optimized designs. We show results for four different cubic cell structures in the simulation space to show scalability. The y-axis is scaled to x^3 for a better illustration of linearity. The dashed line is to split the designs with substantial scalability differences. 69

4·13 The PE utilizations of three baseline and two optimized designs. The results for four different cubic cell structures in the simulation space are provided. B1$\sim$B3: baseline designs; O1 and O2: optimized designs. The dashed lines are used to split the designs with substantial scalability differences. 70

4·14 The performance comparison of the original O2 and O2 with hierarchical filters for four representative cell spaces. 73

4·15 The performance of the original and hierarchical O2 for four representative cell spaces with different numbers of rings equipped. 74

5·1 Architecture of the single-chip design. (a): The general architecture of the iterative force evaluation and motion update. (b): Architecture of force evaluation pipelines 81

5·2 Force output buffers are re-located to prevent high latency caused by data dependency. The scheduling buffer is added in case the bandwidth bottleneck is met. 82

5·3 Position reading and force writing architecture (only 2 position caches in 2 nodes are shown for demonstration). (a): Baseline position reading. (b): New position reading. (c): Baseline force writing. (d): New force writing. Red arrows: Reference data paths. Blue arrows: Neighbor data paths. . . . 85

5·4 Workload distribution patterns in a cuboid simulation space. Grey: The cells evaluated on other nodes. Blue: Cells in which particle pairs are formed without external memory involved. Yellow: Cells in which data from a neighboring FPGA are needed or delivered (1 hop). Orange: 2 hops. Red: 3 hops. (a): Slab distribution. (b): Pillar. (c): Block. 87

5·5 Reconfigurable force data forwarding pattern examples for the pillar distribution. Yellow and orange arrows represent the 1st hop and 2nd hop, separately. Blue node: Home FPGA node. Grey node: Neighbor FPGA node. 88

5·6 Reconfigurable force data forwarding pattern examples for the block distribution. Yellow, orange and red represent the 1st hop, 2nd hop and 3rd hop separately. Blue node: Home FPGA node. Grey node: Neighbor FPGA node. 89

5·7 The port utilization due to the best routing configuration (Max Util) and the average utilization (Avg Util) of all routing configurations for 3 typical pillar distribution cases. Error bar: Standard deviation 92

5·8 The port utilization due to the best routing configuration (Max Util) and the average utilization (Avg Util) of all routing configurations for 4 typical block distribution cases.Error bar: Standard deviation 93

6·1 Cells, spatial partitioning methods, and filtering. R_c: Cutoff radius. (a): the cutoff radius of particles. The force A-B is valid, and A-C and B-C forces are too small and neglected. (b): the import volume of the half-shell method. (c): the spread layers of (b), where the orange cell at the center interacts with 13 blue cells and itself; the middle slice also shows the 2-D import volume. (d): 2-D import volume of the Manhattan method. (e): planar filtering method. dx and dy are the distance components between two particles; particles outside the octagon are not paired with the particle at the center. 99

6·2 The overall layout of the design. (a): cell ID numbers are calculated from their x, y, and z coordinates in space. (b): the topology of 8 ring cores (RC) and 4 motion update units (MUs) for example. 101

6·3 The overall layout of the design with more details. (a): the internals of a ring core. (b): the internals of a MU. 101

6·4 The rules of position cache overlapping and the modified Manhattan method in 2-D. Yellow cells: cell caches. Green: corner caches. The yellow/green particles are cell particles/corner particles separately, where the yellow particles are outlined in black. (a) and (b): the pair is evaluated if the neighbor particle has greater Manhattan distance. (c): the brown SRs should be included a yellow cache, otherwise the two particles shown will never be paired. (d): SR overview. Dark brown: the SRs potentially need to be accessed by green particles. (e): Only the green particles in the dark green region can may be paired with the SPs in the yellow cell. 105

6·5 The complete design with the Manhattan method. (a): the data flow with respect to a single PE. (b): the green particles enter the position input ring and are rotated to several destination PEs (c): two SR handling cases. Red arrow: data traversal order; left: all SRs are requested; right: SR 1 is not requested and is skipped. COP: corner particle. (d): the motion update ring. Each MU on the ring directly updates several yellow/green caches. (e): corner BRAMs store particle positions, cell IDs and the particle IDs for force write back and memory misalignment handling. The same mechanism is applied on SPs. 107

6·6 2-D illustration of the data transfer pattern for 4 FPGA nodes. (a): half-shell method; (b): the proposed position cache overlapping Manhattan method. Yellow: cell region (SR omitted); Blue: data to be transferred; Green: corner region. The arrows indicate the transfer directions of the position data. For small data transfers at corners, the arrows are lightened. . 110

6·7 The performance and PE utilization versus the number of filters for 4 cell geometries. 3 motion update rings and 3 force rings are used. For consistency, we assume each cell is processed by 1 PE, and all 4 cases share the same y-axis. Bar: cycle. Curve: utilization 111

6·8 Filtering rate and the number of cycles required for filtering for the two methods. For 3^3, 4^3, 5^3 and 6^3 cell geometries, the filtering rates remain the same. We assume each cell is only processed by 1 PE for consistency against the scaling of the simulation space. 112

6·9 The latency of the position input ring of a single MD iteration. HS: half-shell; CO: cache overlapping. 80, 100, 120: number of particles per cell. The line plot indicates the percentage of latency reduced using the cache overlapping method. 113

6·10 MU ring latency compared with ideal. Each MU is in charge of updating 8 yellow caches and 8 green caches. A cell contains 80 particles. 114

6·11 Data transfer per FPGA using the two methods. Each iteration (from force evaluation to the next force evaluation phase) takes 100 μs, with 120 bits per packet and 80 particles per cell. (a): the number of cells to be sent to remote FPGAs. (b1): the estimated ideal bandwidth demand per FPGA for a 3-D torus FPGA cluster. (b2): the estimated ideal bandwidth demand per FPGA for 8 FPGAs connected as a ring. 115

7·1 Overview of FASDA. 120

7·2 The topology of inter-node connection and an example of cell-to-FPGA mapping. 124

7·3 Two levels of cell ID conversion with examples. 124

7·4 Arriving/departing data processing. P2R/F2R: position/force to remote. . . 125

7·5 Details of data processing sub-modules. 125

7·6 Two synchronization methods for 4 FPGAs. In this example, each FPGA only communicates with its neighbors. 127

7·7 The behavior of the chained synchronization between two FPGA nodes as an example. Frc: force. Pos: position. 128

7·8 The architecture of a CBB with multiple PEs. 129

7·9 The architecture of an SCBB with 2 SPEs, where an SPE consists of 2 PEs for example. AT: Adder Tree. HPC: Home Position Cache. 130

7·10 The weak scalability comparison. 1-F: 1× FPGA board. 1-SPE: Each SCBB contains 1× SPE. 1-PE: Each SPE contains 1 PE. 133

7·11 The strong scalability comparison. 1-F: 1× FPGA board. 1-SPE: Each SCBB contains 1× SPE. 1-PE: Each SPE contains 1 PE. 134

7·12 The utilization of key components for the design varieties. A: 1-SPE, 1-PE. B: 1-SPE, 3-PE. C: 2-SPE, 3-PE. 137

7·13 The communication bandwidth demand for different design configurations. 137

7·14 The communication bandwidth demand breakdown for different design configurations. 138

7·15 Energy Relative error with respect to OpenMM. 139

List of Abbreviations

ACC	Accumulation operation
ADD	Addition operation
ALM	Adaptive Logic Module on Intel FPGAs
ASIC	Application-Specific Integrated Circuit
AT	Adder Tree
BRAM	Block Random Access Memory
BSP	Bulk Synchronization Parallel
BUF	Buffer
CBB	Cell Building Block
CNN	Convolution Neural Network
CONV	Convolution layer
COTS	Commercial Off the Shelf
CPU	Central Processing Unit
F2R	Force to Ring
FC	Force Cache
FCU	Force Computation Units
FF	Flip-Flop
FIFO	First In, First Out
FLOPS	Floating Point Operations Per Second
FMC	FPGA Mezzanine Card
FP	Floating-Point
FPGA	Field-Programmable Gate Array
FR	Force Ring
FRN	Force Ring Node
GAN	Generative Adversarial Networks
GCID	Global Cell ID
GCN	Graph Convolutional Network
GNN	Graph Neural Network

GPU	Graphics Processing Unit
HBM	High Bandwidth Memory
HDL	Hardware Description Language
HPC	High-Performance Computing / Home Position Cache
HW	Hardware
LCID	Local Cell ID
LR	Long Range
LUT	Look Up Table
MD	Molecular Dynamics
ML	Machine Learning
MU	Motion Update
MUR	Motion Update Ring
MURN	Motion Update Ring Node
N3L	Newton’s 3rd Law
NN	Neural Network
P2R	Position to Ring
PC	Position Cache
PE	Processing Element
PR	Position Ring
PRN	Position Ring Node
RAW	Read After Write
RCID	Relative Cell ID
REG	Register
RL	Range Limited
RNN	Recurrent Neural Network
RTL	Register Transfer Language
SCBB	Scalable Cell Building Block
SoC	System on Chip
SPE	Scalable PE
SUB	Subtraction operation
URAM	Ultra RAM
VAE	Variational Autoencoder
VC	Velocity Cache

Chapter 1

Introduction

Molecular Dynamics (MD) is a computational technique that employs physical laws to simulate atomic movements and interactions. Its use is widespread throughout science and engineering. The field of drug discovery has gained significant attention, particularly in light of the COVID-19 pandemic, and MD has emerged as a vital tool for predicting drug-target interactions and optimizing drug properties [Ganesan et al., 2017, Zhao and Caflisch, 2015, Salo-Ahen et al., 2020, Liu et al., 2018]. However, the process of discovering new drugs is typically time-consuming and expensive, requiring substantial investments and years of work [Mullard, 2014]. To mitigate these challenges, accelerating the drug discovery process through MD simulations, especially for small particle sets ($\sim$50K), is crucial [Mortier et al., 2015, Aminpour et al., 2019, Salo-Ahen et al., 2020].

Despite the small dataset size, the execution of MD simulations is not necessarily simple. To ensure accuracy, MD simulations iterate over discrete time intervals, often on the order of femtoseconds (10^{-15} seconds), resulting in billions of iterations for *long timescales* (10^{-6} and beyond). However, due to data dependencies, parallelizing sequential iterations poses a significant challenge, particularly for achieving strong scaling and accelerating limited workloads within a single iteration. Consequently, an approach that can maintain high efficiency with numerous powerful computing nodes, each handling a small portion of data, is necessary.

While there are several MD software packages available [Case et al., 2005, Phillips et al., 2005, Eastman and Pande, 2010, Abraham et al., 2015, Thompson et al., 2022, Bowers et al., 2006a], some of which support both CPUs and GPUs, CPUs generally exhibit lower computational capabilities, resulting in lower performance compared to GPUs.

The real difficulty, however, is strong scaling: i.e., scaling performance (length of time simulated per wall-clock time) while keeping the problem size constant. As the time simulated per day increases into the microseconds and beyond, the wall-clock time per iteration reduces through milliseconds down to 100s of microseconds and beyond. The (well-known) challenge of strong scaling is that more nodes are required to more quickly perform the computations, but these additional increase the complexity of the communication. Communication eventually dominates: it clearly makes no sense to simulate a few thousand particles with a similar number of nodes. Any solution requires:

1. Maximally powerful nodes,

2. Minimum internode communication latency, and

3. Minimal overhead in interaction between computation and communication.

Their widespread availability, and (comparative) ease of use, makes GPUs the technology of choice for *throughput* and *weak scaling* scenarios [Páll et al., 2020, Glaser et al., 2015]. A typical *throughput* scenario is the testing of a large number of small molecules, say, drug candidates, for a limited timescale, say, for a few hundred nanoseconds. In *weak scaling* scenarios the problem size is large, say, many millions of particles, and computation remains dominant over communication. In the *throughput* scenario, an economical computational alternative is a generic cluster with node-attached GPU accelerators. These resources can also be accessed in commercial clouds such as AWS. In the *weak scaling* scenario, a more powerful communication network is likely to be needed, but one that is typically provided in a High Performance Computing (HPC) cluster.

However, in *strong scaling* scenarios, the second and third parts of the solution must be upgraded. For small simulations GPUs configurations have so far failed to scale beyond one to two processors. In fact the problem is regarded as being so difficult that the only successful alternative (prior to this work) is created a fully custom supercomputer including the building of Application Specific Integrated Circuits (ASICs). These systems are remarkable achievements, offering exceptional performance exactly because they satisfy all three conditions. For instance, Anton 3 can attain a simulation rate of $\sim$200 μs-per-day for 10K particles with 512 nodes [Shaw et al., 2021]. Meanwhile, its predecessor, Anton 2, can achieve around 100 μs-per-day with the same number of nodes [Shaw et al., 2014]. However, their high cost, limited accessibility, and maintenance and upgrade issues hinder their widespread adoption, thus offering limited contribution to the community.

The focus of this dissertation is FPGAs. Similar to ASICs, FPGAs provide hardware flexibility without the same availability and maintenance concerns. Additionally, FPGAs possess low-latency communication capabilities, making them well-suited for MD applications that involve frequent data exchange between multiple FPGA nodes. We note that this low-latency is two parts. First, FPGAs are built so that they can be interconnects pin-to-pin in low latency clusters with point-to-point latency of 100ns or less. And second, within the FPGA the computation and communication logic can be integrated seamlessly so that only a few cycles are needed to get from the application to the communication interface. This combination of flexibility, low latency, and competitive computing power makes FPGAs an ideal choice for achieving strong scaling with commercial off-the-shelf (COTS) components.

FPGAs have also evolved significantly over the years, with on-chip resources increasing from hundreds to nearly a million logic blocks, enabling researchers and engineers to program them for complex applications such as MD with high performance. Recent FPGA-based MD studies have also demonstrated superior performance to GPUs of similar generations, even on a single node [Yuan et al., 2022, Yang et al., 2019a, Wu et al., 2021b].

In MD simulations, a significant portion of the computation revolves around the evaluation of non-bonded forces, which can be divided into two components: N-body range-limited (RL) forces with a cutoff and long-range (LR) force evaluation. RL forces constitute the majority (approximately 90%) of the computation and are highly compute-intensive, while LR forces are more focused on memory and communication aspects. These two components can be treated as separate tasks due to their relatively independent data flow. This thesis primarily focuses on the RL component, which has been the main subject of initial studies on FPGA-based MD simulations [Azizi et al., 2004, Hamada and Nakasato, 2005, Gu et al., 2005a, Kindratenko and Pointer, 2006, Scrofano et al., 2008]. Specifically, computation, communication, and memory access of RL are all studied in depth.

Having identified the operating platform and the specific problem at hand, our focus shifts towards achieving strong scaling in Molecular Dynamics (MD). To fulfill our goal, we must address two primary challenges in a sequential manner. Firstly, we need to determine how to **maximize the potential of a single FPGA**, ensuring high efficiency in MD computations. Subsequently, we must tackle the task of **constructing a multi-FPGA system** that not only maintains the efficiency achieved on a single FPGA but also exhibits strong scalability. By approaching these two high-level problems in a progressive order, we can effectively advance towards our objective of achieving both high efficiency and strong scalability in MD simulations.

To tackle the first challenge, an extensive examination is undertaken to thoroughly investigate three high-level 3-D to 2-D workload mapping schemes. These mapping schemes, characterized by different levels of granularity in workload partitioning, offer distinct advantages in various scenarios. Among these schemes, two design variations are proposed, both leveraging data locality while employing distinct data routing mechanisms. Through this comprehensive exploration, we establish the methodologies necessary to attain high efficiency on a single FPGA node.

The second challenge encompasses not only the need for an efficient design at the single-node level but also demands efficient inter-node communication, synchronization, and strong scaling. In terms of communication, the partitioning of workloads and the interaction patterns among nodes play a crucial role. Thus, we conduct an analysis of the bandwidth requirements and propose a method for balancing the bandwidth specifically tailored for FPGAs. Additionally, recognizing the potential reduction in overall bandwidth requirements through exploiting the force symmetry property, we introduce a modified Manhattan method that significantly reduces data transfer by over 40%.

Addressing synchronization and strong scaling, we propose a completely distributed and decentralized computing system with scalable processing elements (PEs). By eliminating the need for a central authority, we simplify inter-node communication and reduce latency. Furthermore, expanding the system by adding more nodes requires minimal additional effort. To ensure efficient synchronization, we incorporate a localized synchronization mechanism that avoids the overhead associated with global synchronization methods such as Bulk Synchronization Parallel (BSP). This localized approach effectively maintains a fully distributed system, enabling optimal performance and scalability.

The establishment of the system owes much to the emergence of FPGAs in cloud computing. Prominent cloud platforms such as Microsoft Azure [Zhang et al., 2017], Amazon EC2 F1 [ama,], FAbRIC [uta,], Chameleon Cloud [Keahey et al., 2020], and Open Cloud Testbed [Handagala et al., 2022] now provide accessible FPGA clusters. These platforms allow for seamless access to FPGA resources, facilitating the deployment of FPGA-based designs. As a result, we anticipate a rising trend in the development of FPGA applications in the near future, driven by the availability and accessibility of FPGA clusters offered by these cloud platforms.

Next, the specific work is listed with more detailed description.

1. Single-FPGA Design 1: Transposed Memory Blocks. This design stems from the finest-grained among the three 3-D to 2-D workload mapping schemes, where all PEs process the same particle at the same time. In conventional approaches, particles within a region are grouped together and stored sequentially in a memory block. However, in this design, the particles within a group are distributed across different memory blocks, enabling parallel access. As a result, concurrent processing of a single particle becomes possible, enabling high parallelism even for the most fine-grained workload mapping scheme.

2. Single-FPGA Design 2: Cell-based Workload Partitioning. This design is derived from the coarsest-grained mapping scheme among the three 3-D to 2-D workload mapping schemes. In this scheme, each processing element (PE) is responsible for processing a group of particles within a specific region. While this approach necessitates the incorporation of a daisy chain data routing mechanism to mitigate fan-in and fan-out challenges, it offers remarkable flexibility in parallelism and structured scaling in space. To further enhance efficiency, technologies such as hierarchical filters and out-of-order position broadcast are employed.

3. FPGAs on a 3-D torus: Bandwidth analysis and communication balancing. The 3-D torus topology is a commonly employed network structure for 3-D applications, including MD. This topology, utilized in systems like the Anton series, is well-suited for FPGAs, given the availability of sufficient communication ports. In this study, a comprehensive analysis is conducted on three different data partitioning patterns for FPGAs: slabs, pillars, and blocks. However, it is observed that the latter two patterns result in imbalanced communication intensity along different directions, leading to underutilized communication ports. To address this imbalance issue, we propose a static communication balancing mechanism. This mechanism aims to optimize overall communication intensity by effectively redistributing the communication load, leveraging the available bandwidth to its fullest potential. By implementing this approach, we achieve higher overall communication intensity while utilizing the given bandwidth efficiently.

4. Multi-FPGA bandwidth demand reduction: The exploration of force symmetry. Determining the optimal mapping of particles and computations to memories and processors is a complex task when parallelizing the computation of pairwise forces. However, achieving this optimal mapping can lead to significant reductions in data movement and computation. For many years, the exploration of mappings on FPGAs has been relatively limited, with the half-shell method being the preferred approach in prior studies. In this research, we discover that the Manhattan method surprisingly aligns well with FPGA hardware. By introducing the cache overlapping technique, we effectively address the requirement for ultra-fine-grained data access imposed by the Manhattan method, even though the memory blocks on FPGAs may not appear to be fine-grained enough. Eventually, our results show that we achieve balanced communication and significantly reduce data transfer by 40% to 75% in typical multi-FPGA scenarios. This finding is particularly valuable in the cases where communication bandwidth is limited, offering notable benefits in data transfer.

5. FASDA: An FPGA-Aided, Scalable and Distributed Accelerator for Range-Limited Molecular Dynamics. Our research in MD RL is driven by the overarching goal of achieving high efficiency on a single FPGA node while also enabling strong scaling across multiple FPGA nodes. To achieve this, we recognize the significant advantages of employing a completely distributed and decentralized computing system. By eliminating the need for a central authority, our system simplifies inter-node communication, reduces latency, and allows for seamless expansion of the system by adding more nodes with minimal additional effort. In line with these objectives, we introduce FASDA, which, to the best of our knowledge, is the first scalable and fully distributed RL N-body simulation system with a cutoff that leverages the customizable nature and communication efficiency of FPGAs. Moreover, FASDA is designed with a modular structure, enabling the integration of easily pluggable components that can be adjusted based on user requirements and available resources with minimal modifications.

In summary, this dissertation chronicles progress made in accelerating MD RL on FP-

GAs, beginning with the construction of single-node designs based on various 3-D to 2-D mapping schemes, and culminating in a multi-FPGA design that leverages the efficiency achieved in the single-FPGA designs, while enabling strong scaling in the meantime. Along this journey, our research not only accelerates MD RL on FPGAs but also offers valuable insights applicable to other applications that exhibit a localized data interaction pattern. By documenting these advancements, we contribute to the broader understanding of FPGA-based acceleration and its potential in optimizing performance for applications with similar characteristics.

The overall contributions of this dissertation are as follows:

- Problem Definition and Methodology: The objective of our research is to accelerate MD RL through a two-step approach. Firstly, we focus on optimizing the efficiency of single-FPGA designs. Secondly, we expand our efforts to encompass multi-FPGA configurations, aiming to achieve strong scaling.

- New Architectures: Our research has resulted in the development of a set of innovative hardware architectures that effectively enhance performance and minimize bandwidth requirements in communication.

- Performance: Our research successfully demonstrates the efficacy of the proposed methods, showcasing performance improvements of 4 to 5 times when compared to relevant prior art.

- Advancing Computing Technology for MD: In our research, we leverage the emerging FPGA cluster platforms to map and implement our designs. Through our experimentation and analysis, we provide compelling evidence that FPGA clusters are not only competitive but also demonstrate superiority over other computing technologies in terms of high-performance computing and strong scaling.

The subsequent sections of this dissertation are structured as follows. Chapter 2 provides the necessary background information and covers preliminary work in the field. Moving forward, Chapter 3 describes the high-level methodologies including space partitioning and workload mapping. Chapter 4 delves into the details of the single-FPGA designs. The subsequent three chapters focus on multi-FPGA solutions, each addressing a different aspect. Chapter 5 examines bandwidth analysis and communication balancing of FPGAs within a 3-D torus topology. Chapter 6 explores the utilization of force symmetry to reduce the demand for bandwidth. In Chapter 7, we present a scalable and fully distributed MD RL accelerator implemented on an FPGA cluster. Finally, Chapter 8 concludes the dissertation, summarizing the key findings and discussing potential future research directions.

Chapter 2

Background

This chapter serves to provide a comprehensive background on MD RL theories and algorithms, as well as the fundamentals of FPGAs. Firstly, we delve into the underlying physical model and algorithm that govern the calculation of forces and their relationship to particle distances. Next, we discuss common physics-based optimizations and the corresponding data structures that arise from them. Lastly, we introduce the architecture of FPGAs and their utilization in accelerating MD RL computations.

2.1 The Physical Model of Range-Limited Forces

This section briefly introduces the MD RL algorithm and the physical principles within.

2.1.1 RL Force Features

The RL forces in MD simulations comprise of two distinct components. Firstly, there is the short-range term of electrostatic force, which is commonly computed using methods such as Particle Mesh Ewald (PME) or similar techniques [Darden et al., 1993]. Secondly, there is the force derived from the Lennard-Jones (LJ) potential, which contributes to the overall RL force calculation. In this work, we refer to RL forces as only the latter to avoid unnecessary LR discussions.

The LJ potential is a mathematical representation of the Van der Waals interaction observed between electrically neutral particles. It describes the potential energy between two particles, labeled as i and j, based on their distance r_{ij}. The LJ potential equation is ex-

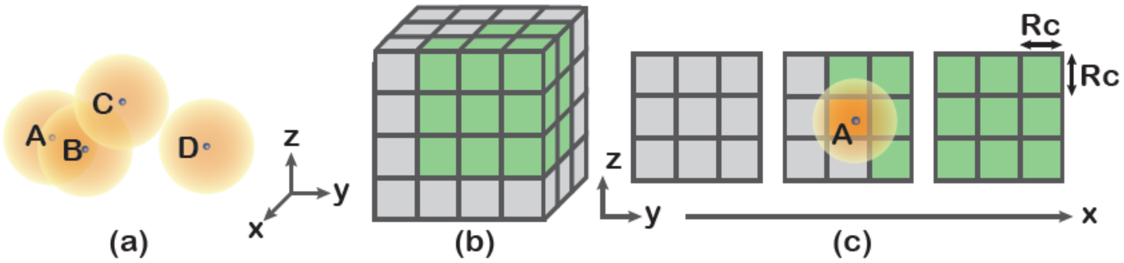


Figure 2·1: Fundamentals of R_c and the cell list method. (a) The cutoff regions of four particles. Only A and B interact with each other. (b) A simulation space divided into 3×4×4 cells with side length the same as R_c. (c) The home cell of particle A and 26 adjacent cells unfolded from (b). The green cells are regarded as neighbor cells.

pressed as follows:

$$V_{ij}^{LJ} = 4\varepsilon_{ij}[(\frac{\sigma_{ij}}{r_{ij}})^{12} - (\frac{\sigma_{ij}}{r_{ij}})^{6}] \tag{2.1}$$

The potential is influenced by two parameters: ε, which represents the dispersion energy and determines the amplitude of the potential, and σ, which represents the characteristic distance between particles where the LJ potential is zero. By taking the gradient of the potential, we can calculate the range-limited (RL) force between particle i and j:

$$\mathbf{F}_{ij}^{LJ} = \frac{\varepsilon_{ij}}{\sigma_{ij}^{2}}[48(\frac{\sigma_{ij}}{r_{ij}})^{14} - 24(\frac{\sigma_{ij}}{r_{ij}})^{8}]\mathbf{r}_{ij} \tag{2.2}$$

Equation 2.2 reveals that the computational complexity of range-limited (RL) forces in MD RL is O(N^2) since forces need to be calculated between every pair of particles. However, it can be observed that the force diminishes rapidly as the distance r_{ij} increases, indicating that the contribution of distant particles to the force is negligible. To address this, a cutoff radius (R_c) is introduced. For particle pairs with $r_{ij} > R_c$, the force is considered to be zero, allowing us to ignore the computation of pairwise forces beyond this distance. This cutoff radius is visually represented by halos in Figure 2·1(a). For instance, Particle A interacts only with Particle B because A is within B's halo, and vice versa. Particle C and D have minimal impact on the resultant force of A, and thus their contributions can be disregarded in A's computation. As a result, the overall computational complexity reduces to O(*mN*), where *m* represents the average number of neighboring particles for each particle. Typically, *m* is much smaller than *N*.

Moreover, the total computation can be effectively reduced by half since the pairwise forces between particles have an equal and opposite effect on each participating particle, in accordance with Newton's third law (N3L).

2.1.2 Force Integration

The MD procedure consists of two main phases: **force evaluation** and **motion update**, which are iteratively executed. In the force evaluation phase, the pair-wise forces between particles are computed using Equation 2.2 based on their current positions. These forces are then accumulated to determine the resultant force acting on each particle. Once all the forces are computed, the motion update phase begins. The resultant forces are utilized to calculate the changes in velocity and position for each particle, following a specific integration method. In this study, the Verlet Integrator is adopted due to its low hardware cost and favorable numerical stability. The Verlet Integration method with an error of O(Δt^2) is introduced:

$$\mathbf{a}(t) = \frac{\mathbf{F}(t)}{m} \tag{2.3}$$

$$\mathbf{v}(t+\Delta t) = \mathbf{v}(t) + \frac{\mathbf{a}(t)+\mathbf{a}(t+\Delta t)}{2}t \tag{2.4}$$

$$\mathbf{r}(t+\Delta t) = \mathbf{r}(t) + \mathbf{v}(t+\Delta t)\Delta t \tag{2.5}$$

Compared to force evaluation, motion update is much less computationally intensive as it is applied only once per particle.

2.1.3 Periodic Boundary Condition

Periodic boundary conditions are a commonly employed technique in solid-state physics and computational physics when studying large or infinite homogeneous systems. This approach allows for the modeling of a small system that is replicated to represent the entire system. With periodic boundary conditions, particles that exit the small system on one side re-enter from the opposite side with the same velocity, creating a seamless periodicity in the simulation.

By applying periodic boundary conditions, important conservation properties of the system are maintained. The number of particles, denoted by N, remains constant throughout the simulation. The volume of the simulation space, represented by V, also remains unchanged. Additionally, the total energy of the system, denoted by E, is conserved in the absence of external energy input. These conservation properties make the simulation compatible with a microcanonical ensemble (NVE ensemble), as if the system is isolated from its surroundings.

In addition to the conservation of particle number, volume, and energy, periodic boundary conditions allow for the determination of environmental parameters such as temperature. By performing simulations using periodic boundary conditions, it becomes possible to study the thermodynamic behavior of the system and obtain information about its temperature distribution. This is achieved by analyzing the kinetic energies and velocities of particles within the simulation.

2.2 MD Data Structure

In this section, we first compare and analyze two commonly used data structures, and analyze their pros and cons specifically for FPGAs. Second, we discuss a methodology for optimizing memory efficiency on FPGAs by partitioning the simulation space. Finally, we examine the communication patterns between FPGA nodes, particularly focusing on the import volume associated with different spatial decomposition methods. In our case, only on-chip memory is used to avoid the latency in off-chip data access.

2.2.1 Neighbor List vs. Cell List

Neighbor List. Conventionally, particle data is organized using neighbor lists and cell lists. In the case of neighbor lists, a list of neighboring particles is created for each individual particle in the simulation space. While this approach allows for fast data retrieval when evaluating a particle, it has two main limitations. Firstly, it does not effectively exploit data locality. A particle can appear as a neighbor in multiple neighbor lists with different addresses, resulting in duplicated data and disordered data access. This can consume significant memory capacity and bandwidth, particularly on FPGAs where on-chip memory resources are limited. Secondly, maintaining neighbor lists can be challenging. When a particle moves during an MD iteration, it needs to be removed from certain neighbor lists and added to others. Identifying and updating these neighbor lists can be complex and resource-intensive. Although the neighbor list method may be efficient in scenarios where memory resources are abundant, it may not be ideal for FPGAs where memory can easily become a bottleneck.

Cell List. In practice, the neighbor list method can be replaced by the cell list method with hardware-based filtering in implementations. The cell list approach involves spatially grouping particles before evaluation. As depicted in Figure 2·1(b), the simulation space is divided into cubic cells, such as a 3×4×4 configuration, with a specific side length denoted as R_c. Each cell contains a separate memory domain to store the particles assigned to it. For instance, in the case of a 3×4×4 setup, a total of 48 memory regions are allocated for particle storage, corresponding to each individual cell.

Suppose particle A is located within cell (1, 2, 2). In this scenario, only the particles in the green neighboring cells (including the home cell itself) need to be checked for valid particle pairs with particle A. By applying N3L, only 14 out of the 27 surrounding cells need to be considered as neighbor cells, as illustrated in Figure 2·1(c) using the half-shell method. Additional layouts for importing neighbor data are discussed in section 2.2.3. The cell lists effectively categorize particles in a geometric manner, promoting high data locality. Consequently, a particle can be broadcast to its neighboring cells for pairing. However, it should be noted that only approximately 15% of the neighbor particles are expected to form valid pairs with the reference particle, as indicated by the equation below:

$$P = \frac{\frac{4}{3}\pi R_c^3}{27R_c^3} = 15.5\% \tag{2.6}$$

The denominator in the equation represents the volume of the 3×3×3 cells surrounding a home cell, which has a side length equal to the cutoff radius R_c. This justification will be discussed in more detail in section 3.1. The numerator, on the other hand, corresponds to the volume of a sphere with a radius of R_c. As a result, it is desirable to perform preliminary particle pair filtering, where a force computing unit receives particle pairs from multiple filters to ensure efficient utilization and avoid idleness.

2.2.2 Filtering

In hardware implementations, locality out-weighs resources. That is to say, rather than constructing precise neighbor lists, we leverage the locality introduced by the cell-list method to minimize the number of data transactions per particle at the expense of deploying filters. Fortunately, the filtering process itself requires only a small amount of on-chip resources [Chiu and Herbordt, 2009], particularly when employing planar filtering, which utilizes low-precision integer comparisons.

Figure 2·2 provides an illustration of the two filter mechanisms. The full precision filter involves 3 multiplications and 5 additions, while the planar filter requires only 5 additions. Due to the absence of multiplications, the planar filter can be implemented using, for example, 8-bit fixed-point data, taking advantage of the linearity of the constraints. In

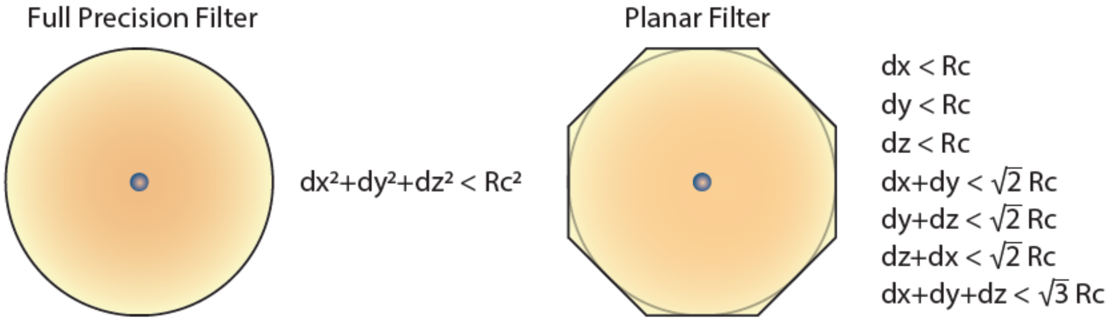


Figure 2·2: The 2-D illustration of two filtering methods and the criterion.

contrast, multiplications incur much higher hardware costs and necessitate more bits for accuracy. Considering the resource consumption, the planar filter mechanism is chosen throughout all our work, as hundreds of, or even a thousand filters are to be deployed, despite introducing 13% more pairs to be filtered.

2.2.3 Neighbor Data Importing Layout

The layout for importing neighbor data is fundamentally determined by the interpretation of N3L. One of the viable approaches for handling neighbor data import is the half-shell method (depicted in Figure 2·1(c)). In this method, 14 out of 27 neighbor cells are directly imported for a home cell without any additional cell partitioning. Other methods include the Neutral Territory (NT) method used in Anton 1 and 2 [Bowers et al., 2007, Bowers et al., 2006b], and the Hybrid Manhattan method used in Anton 3 [Shaw et al., 2021]. However, unlike ASICs, FPGAs rely on block RAMs (BRAMs) for data storage. Each BRAM typically has a depth of hundreds of entries and a width of tens of bits, making it suitable for storing data chunks rather than fine-grained data fragments. In comparison to the NT and Manhattan methods, the half-shell method provides easy organization of data by cells and allows for efficient memory access, but at the cost of a higher import volume. In chapter 6, a modified manhattan method will be demonstrated to overcome the problem in fine-grained memory access and reduce the overall import volume.

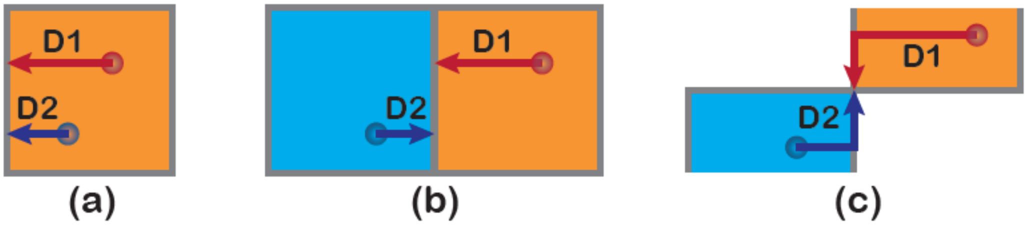


Figure 2·3: The mechanism of the traditional Manhattan method shown in 2-D. The particle pairs are evaluated by the processor located in orange cells for 3 particle pair scenarios: (a) both in same cell; (b) in neighboring cells sharing a boundary; (c) in neighboring cells sharing a corner.

To address the neighbor data importing problem in a general sense, we need to determine whether to compute the force $F_{a,\,b}$ when particle a or particle b is chosen as the reference particle, in order to avoid redundant force calculations. Figure 2·3 showcases the Manhattan method, which encompasses three scenarios for determining the reference particle. In the case where both particles reside in the same cell, an arbitrary tie-breaker is used (such as "furthest to the right" as shown). However, if the particles are in different cells, the reference particle is selected based on its greater Manhattan distance to the boundary of the two cells. The Manhattan method differs significantly from the half-shell method as it leads to a symmetric import volume that is also reduced in size.

2.3 FPGA Background

2.3.1 FPGA Architecture

FPGAs are highly versatile integrated circuits that can be configured by users after they are manufactured. They are renowned for their exceptional performance, low power consumption, and support for reconfigurability. FPGAs are widely recognized as the most commonly used reconfigurable devices globally. They offer not only the ability to customize their functionality but also seamless integration into the communication stack, including applications in routers [Bolaria and Byrne, 2009] and network-facing components [Caulfield et al., 2016, Eran et al., 2019].

FPGA normally consists of five types of hardware resources for efficient computation and communication [Hauck and DeHon, 2008].

1. **DSP units** are generally used to perform high-precision and high-performance multiplication, addition, and multiply-accumulation operations.

2. **Look-Up-Tables (LUTs)** can be used to provide both flexible computation and high-concurrency data storage.

3. **Flip-Flops (FFs)** are used as registers.

4. **Block RAMs (BRAMs)** provide tens of GByte on-chip storage.

5. **Multi Gigabit Transceivers (MGTs)** provide efficient inter-FPGA communication. Each FPGA chip normally has hundreds of high-bandwidth (over 20Gb/s for each MGT) and low-latency MGTs as I/Os. These I/Os can be directed connected to other on-chip resources.

The hardware resources of FPGAs are organized in a hierarchical and programmable on-chip interconnect network. This allows users to freely integrate these resources and tailor them to their specific problem requirements by programming the interconnect network. The inherent flexibility of FPGAs has positioned them as a highly competitive platform widely utilized in High-Performance Computing (HPC) and machine learning acceleration [Gokhale and Graham, 2005, Herbordt et al., 2007b, Herbordt et al., 2008a, VanCourt and Herbordt, 2009, Benkrid and Vanderbauwhede, 2013].

FPGAs are often compared to GPUs, which are widely used for acceleration purposes. While there are conceptual differences between the two, in practical usage, FPGAs often exhibit similarities to GPUs. FPGAs typically incorporate a large number of parallel computing units, similar to the streaming multiprocessors in GPUs. Each computing unit in an FPGA consists of computation pipelines implemented using LUTs, DSPs, and FFs, as well as local memories utilizing BRAMs and LUTs. BRAMs in FPGAs can also serve as global scratchpad memory, similar to caches in GPUs, which are shared by all computing units.

However, FPGAs also possess distinct advantages and differ from GPUs in several aspects. The computing units in FPGAs can be customized to precisely match the specific requirements of the target problem, enabling them to achieve near-optimal efficiency. Moreover, FPGAs offer a flexible and customizable interconnect, allowing the computing units to be interconnected in a versatile manner. There are no restrictions on inter-computing unit communication, apart from the physical limitations of the interconnect. This flexibility makes FPGAs well-suited for handling irregularity problems and provides them with unique capabilities compared to GPUs.

Considering the exceptional communication capabilities of FPGAs, it is logical to configure them in FPGA-centric clusters. Extensive research has been conducted to model and construct such clusters, which involves determining the interconnect types used, such as direct FPGA-to-FPGA communication or indirect communication through a router [Sheng et al., 2015,Sheng et al., 2016,Sheng et al., 2017b,Sheng et al., 2017a,Putnam, 2014,Plessl, 2018,Boku et al., 2019].

2.3.2 FPGAs in Molecular Dynamics

The roots of using FPGAs in Molecular Dynamics lies perhaps in the Gravity Pipe (GRAPE) project whose goal was to accelerate N-Body computations in cosmology through ASIC-based systems [Ebisuzaki et al., 1993, Ito et al., 1991, Okumura et al., 1992, Makino et al., 1994, Makino et al., 2003, Makino and Daisaka, 2012]. The work was extended to molecular dynamics, [Fukushige et al., 1996,Komeiji et al., 1997,Narumi et al., 2000,Sumanth et al., 2003,Narumi et al., 2006,Ohmura et al., 2014,Morimoto et al., 2021] and later extended to FPGAs.

From 2003-2011 there was much work investigating use of CPU-attached FPGAs to accelerate MD. The primary direction was to offload onto the FPGA the RL computation while control, motion update, and the rest of the computations were performed on the CPU [Azizi et al., 2004,Gu et al., 2005a,Gu et al., 2005b,Hamada and Nakasato, 2005,Gu et al., 2006a, Gu et al., 2006c, Gu et al., 2006b, Scrofano et al., 2006, Kindratenko and

Pointer, 2006, Alam et al., 2007, Gu and Herbordt, 2007b, Gu et al., 2008, Chiu et al., 2008, Chiu and Herbordt, 2009, Chiu and Herbordt, 2010a, Chiu and Herbordt, 2010b, Chiu et al., 2011].

There were also investigations related to the LR force [Sasaki et al., 2005, VanCourt and Herbordt, 2006a, Gu and Herbordt, 2007a] and a number of studies integrating MD into applications [VanCourt et al., 2004, VanCourt and Herbordt, 2006b, Sukhwani and Herbordt, 2008, Sukhwani and Herbordt,]. The next generation concentrated on demonstrating the viability of FPGA clusters in LR and strong scaling [Humphries et al., 2014, Sheng et al., 2014, Lawande et al., 2016, Sanaullah et al., 2016a, Sanaullah et al., 2016b, Sheng et al., 2017b].

Moreover, the last few years have seen the first studies on the bonded force [Xiong and Herbordt, 2017], integrated single FPGA solutions [Yang et al., 2019a], and studies related to the current work [Yang et al., 2017, Yang et al., 2019b, Pascoe et al., 2020, Stewart et al., 2021, Wu et al., 2020, Wu et al., 2021b, Wu et al., 2021a, Wu et al., 2022, Wu et al., 2023]. Surveys can be found in [Herbordt, 2013, Khan et al., 2013, Schaffner and Benini, 2018, Jones et al., 2022]. Also of potential interest is work on MD using Discrete Event Simulation [Model and Herbordt, 2007, Herbordt et al., 2008b, Herbordt et al., 2009, Khan and Herbordt, 2011].

Other MD models and MD-related simulations studied on FPGA platforms are as follows. Based on the Dynamic Lattice Liquid (DLL) model, ARUZ, a massive FPGA cluster consisting ∼26000 FPGAs, including Xilinx Artix and Zynq FPGAs, [Kiełbik et al., 2018] has been developed to simulate particle diffusion. In [Yuan et al., 2022], a Tersoff Potential, a three-body potential, has been applied, rather than the conventional pairwise potential. In terms of MD-related simulations, researchers have implemented a cosmological-like MD accelerator on multiple Intel Stratix 10 FPGAs with OpenCL [Menzel et al., 2021].

More related work is described in the later chapters with respect to specific studies.

2.3.3 FPGAs in High Performance Computing

Overall FPGAs have gained significant importance in HPC acceleration due to their aforementioned advantages [VanCourt and Herbordt, 2004, VanCourt and Herbordt, 2005b, VanCourt and Herbordt, 2007, Sanaullah et al., 2018c, Sanaullah et al., 2018a, Jamieson et al., 2018]. While GPUs currently dominate the HPC landscape, FPGAs have demonstrated their potential to become a key component of next-generation HPC systems. Researchers have successfully showcased the efficiency and benefits of FPGAs in various scientific computing applications beyond MD, including Adaptive Mesh Refinement [Wang et al., 2019a, Wang et al., 2019b], Algebraic Multigrid [Haghi et al., 2020a], Bioinformatics [Herbordt et al., 2006, Herbordt et al., 2007a] and security-related tasks [Wolfe et al., 2020, Patel et al., , Patel et al., 2022a, Patel et al., 2022b]. The programmability of FPGAs has often been considered a hurdle to their wider adoption in practical HPC systems. However, researchers have demonstrated that this challenge can be addressed through improved design tools [VanCourt and Herbordt, 2005a, VanCourt and Herbordt, 2006c, Sanaullah and Herbordt, 2018b, Sanaullah et al., 2018b, Sanaullah and Herbordt, 2018a, Sanaullah and Herbordt, 2018c, Herbordt, 2019, Shahzad et al., 2022] and Middleware [Haghi et al., 2020c, Haghi et al., 2020b, Xiong et al., 2020, Bandara et al., 2022]. In recent years, FPGAs have emerged as promising substrates, such as SmartNICs and Smart Switches, for coordinating communication and processing offloaded data [Haghi et al., 2022, Guo et al., 2022b, Guo et al., 2022a, Haghi et al., 2023, Guo et al., 2023]. FPGAs also find extensive use in neural network acceleration, e.g., for training [Geng et al., 2018b, Geng et al., 2018a, Wang et al., 2020], using CGRAs for QNNs [Geng et al., 2020b], GNNs [Geng et al., 2020a, Geng et al., 2021c], and binarized NNs [Geng et al., 2019b, Geng et al., 2019a, Geng et al., 2021a]. Researchers have proposed various model regularization approaches for CNNs and RNNs, achieving efficient acceleration of regularized models with FPGAs [Shi et al., 2020]. See also [Geng et al., 2021b] for a survey.

Chapter 3

High-Level Methodologies

This chapter focuses on three key aspects of the dissertation: A space decomposition method, workload mapping schemes, and data format choices.

Firstly, a thorough quantitative analysis is conducted to justify the selection of the cell size as the cutoff radius (R_c). The analysis demonstrates that choosing the cell size as the cutoff radius is balanced in terms of cost-effectiveness.

Next, the mapping of cells to processing elements (PEs) is discussed in detail, following the determination of the cell size. Three mapping schemes, varying in workload granularity, are comprehensively explored, providing options for efficient workload distribution.

Finally, the fundamental data format is determined. By utilizing fixed-point position and single-precision floating-point force data formats, the dissertation achieves several advantages. Firstly, it reduces the usage of Block RAMs (BRAMs) on FPGAs by 33%. Additionally, it avoids the need for complex and expensive force computations, replacing them with a convenient force look-up mechanism. These data format choices contribute to improved performance and resource utilization in the dissertation's framework.

3.1 Space Decomposition

As mentioned in Chapter 2, the cell-list method is adopted in our work. Specifically, the simulation space is partitioned into cubic cells with R_c as the side length. In order to justify this selection, an analytical model is established.

3.1.1 Assumptions and Parameter Declaration

This model is based on several assumptions to form an analytical standard. The assumptions are listed below:

Assumption 1. Particles are uniformly distributed. Physically speaking, particles in a simulation space tend to move around to fill empty spaces, especially in water environment. This assumption is to avoid tedious error terms involved.

Assumption 2. Perfect scheduling. We assume no bubble exists in the process, all filters start and finish at the same time.

Assumption 3. No data duplication or caching. In other words, for the evaluation of each pair of particles, the position of each particle needs to be read from the memory, and the positions read out are discarded after the evaluation.

Assumption 4. Each cell is mapped to a memory block. In this model, m cells are instantiated, and a cell only supports one read operation per cycle.

The following parameters and variables are only used in this section.

m	Number of cells
p	Number of filters
ρ	Particle density of the simulation space
V	Volume of the simulation space
N_n	Number of neighboring cells with respect to a cell
N_p	Number of particle pairs to filter in total
T	Time required for position reading and filtering
T_p	Time required for filtering
T_m	Time required for position reading
R_c	Cutoff radius
ω	Cutoff radius normalized to cell edge length, $\omega = R_c \sqrt[3]{\frac{m}{V}}$
μ_{mp}	min(m, $2p$)

3.1.2 The Model

In a simulation space, there are a total of ρV particles, and each particle is paired with all the particles in its neighboring cells. The number of neighbor particles associated with a single particle is $\frac{N_n \rho V}{m}$. Therefore, the total number of particle pairs that need to be filtered can be calculated as follows:

$$N_p = \frac{N_n \rho^2 V^2}{2m} \tag{3.1}$$

Note that the factor 2 in the denominator is from the force symmetry in N3L. Because perfect scheduling is assumed, it can be inferred the total time required for filtering is:

$$T_p(m,\ p) = \frac{N_p}{p} = \frac{N_n \rho^2 V^2}{2mp} \tag{3.2}$$

Following assumption 3, the total time required for position reading is:

$$T_m(m) = \frac{2N_p}{m} = \frac{N_n \rho^2 V^2}{m^2} \tag{3.3}$$

Consequently, the total time for both position reading and filtering can be represented as the maximum of T_p and T_m:

$$T(m,\ p) = \frac{N_n \rho^2 V^2}{m} \max(\frac{1}{m},\ \frac{1}{2p}) \tag{3.4}$$

We can observe that the number of cells and filters match at $m = 2p$. Let $\mu_{mp} = \min(m,\ 2p)$, T is translated as:

$$T(m,\ p) = \frac{N_n \rho^2 V^2}{m \mu_{mp}} \tag{3.5}$$

It can be observed that all the variables needed to calculate T can be determined, except for the number of neighbor cells N_n. As discussed in Chapter 2, when the side length of a cell is equal to the cutoff radius, N_n is equal to 27. However, the situation becomes more complex when dealing with varying cell sizes. For example, a slight decrease in cell size may introduce a lot more cells to evaluate, drastically increasing the import volume. This

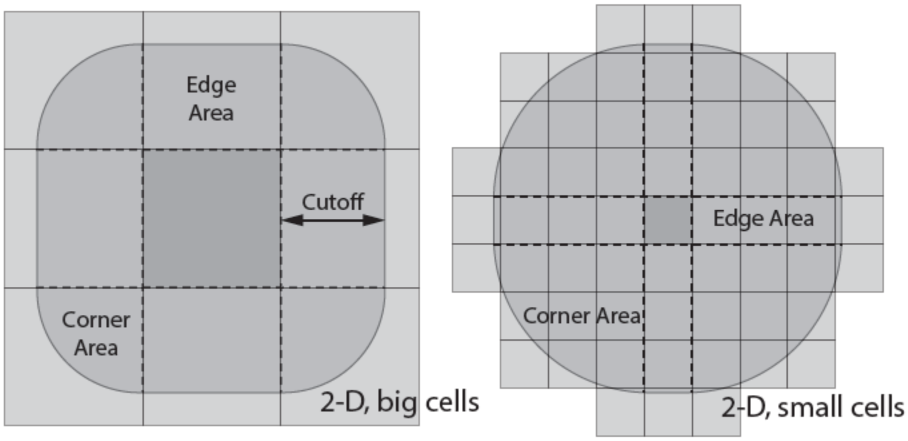


Figure 3·1: Neighbor cells in 2-D.

phenomenon can be observed in Figure 3·1 in 2-D illustration.

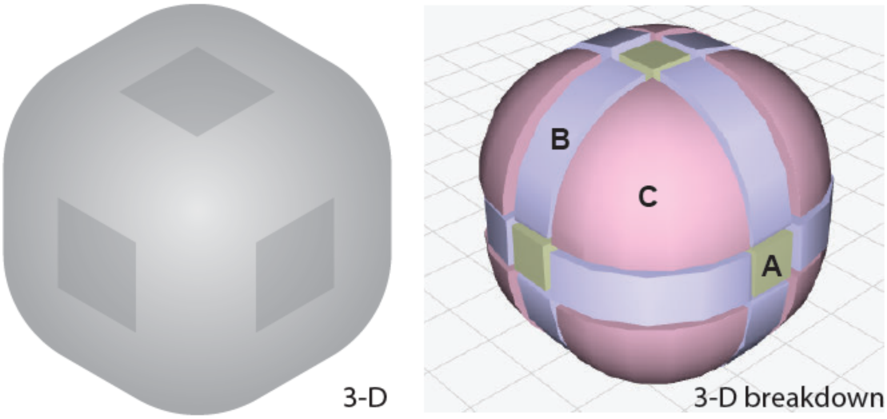


Figure 3·2: Import volume in 3-D. A/B/C: face/edge/corner.

Figure 3·2 illustrates the import volume of a cell located at the center. Our objective is to determine the number of cubic cells that are either inside or have overlapping regions with the round-cornered cube depicted. However, due to the irregularity of region B and region C, obtaining the import volume or the number of neighbor cells is not a straightforward task. In the next section, we demonstrate that through exploring the Gauss Circle Problem, we are able to obtain the **exact** number of neighbor cells with given cell side length and cutoff radius.

3.1.3 Quantitative Analysis for the Import Volume

We first present the solution. The number of neighbor cells satisfies:

$$N_n = 2\sum_{l=0}^{\lfloor\omega\rfloor} G_c(\sqrt{\omega^2 - l^2}) + 3G_c(\omega) + 6\lfloor\omega\rfloor + 7 \tag{3.6}$$

where $\omega = R_c \sqrt[3]{\frac{m}{V}}$ is the cutoff normalized to the edge length of a cell, and functions $G_c(\omega)$ and $N_c(\omega)$ are defined as

$$G_c(\omega) = 3 + 4\lfloor\omega\rfloor + N_c(\omega) \tag{3.7}$$

$$N_c(\omega) = 1 + 4\sum_{l=0}^{\infty}(\lfloor\frac{\omega^2}{4l+1}\rfloor - \lfloor\frac{\omega^2}{4l+3}\rfloor) \tag{3.8}$$

Proof. We aim to determine the number of cubic cells contained within a round-cornered cube, as depicted in Figure 3·2. To achieve this, we divide the cube into four distinct components: faces (A), edges (B), corners (C), and the center (not shown). The faces can be represented as simple cuboids, while the edges collectively form three cylinders with a radius of R_c in three different directions. The corners form a sphere with a radius of R_c. The calculation of the number of cells present in the center and on the faces is straightforward, resulting in the term $6\lfloor\omega\rfloor + 7$. To ensure consistency, we consider a cell as part of the count if its boundary lies within the import region.

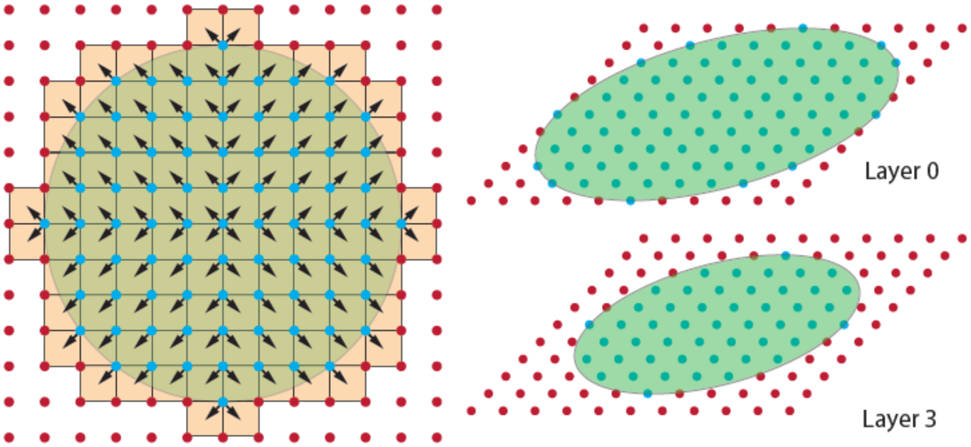


Figure 3·3: A variation of Gauss Circle Problem.

To obtain N_n, we harness the existing solution to the Gauss circle problem (GCP). The original GCP is about finding the number of integer lattice points (see the points in figure 3·3) inside a circle centered at a grid point. Equation 3.8 is one of the analytical solutions [Hilbert, 1952] to the original GCP.

Considering the simulation space is filled with the lattice of cells, we first convert GCP into the problem of finding the number of cells in a circle as figure 3·3 shows:

Let C be a circle in $\mathbb{R}^2$ with radius $r \in \mathbb{R}^+$ centered at $(x_0,\ y_0) \in \mathbb{Z}^2$, for all points $(x,\ y) \in \mathbb{Z}^2$ satisfying $(x-x_0)^2+(y-y_0)^2 \leq r^2$, the point $(x=x_0,\ y=y_0)$ contributes 4× to the number of cells, and points $(x=x_0,\ y \neq y_0)$ or $(x \neq x_0,\ y=y_0)$ contribute 2×, and points $(x \neq x_0,\ y \neq y_0)$ contribute 1×. The contributions are depicted by the black arrows. Here we skip the rigorous proof as it is obvious in the figure.

With the above analysis of grid point contribution, we can easily derive that the number of cells in a 2-D circle is the edges is $3+4\lfloor\omega\rfloor+N_c(\omega)$, where the $3+4\lfloor\omega\rfloor$ term represents the extra contribution done by the center grid point and . It is also clear that all edge regions in Figure 3·2 contain $3(3+4\lfloor\omega\rfloor+N_c(\omega))$ cells, as the edges are essentially three cylinders with thickness of 1 cell. This results in the $3G_c(\omega)$ term in Equation 3.6.

For the remaining corners, we observe that the number of cells in a sphere is equivalent to the number of cells in several layers of circles (see figure 3·3) with radius $r=\sqrt{\omega^2-l^2}$, where l is the layer ID with $l \in \mathbb{N}_0,\ l \leq \omega$. This corresponds to the sum of $G_c(\sqrt{\omega^2-l^2})$ term. If we only consider the layers of a hemisphere, each layer contributes 2× to the number of cells, resulting in the leading factor 2 in Equation 3.6. The final form of N_n shows that N_n monotonically non-decreases with ω and m.

3.1.4 Latency Estimation

Latency Analysis. This analysis aims to discuss the relation between T and $m=\frac{\omega^3 V}{R_c^3}$ under several scenarios, where m reflects the cell granularity with given V and R_c. We notice $T \propto \frac{N_n}{m\mu_{mp}}$ in Equation 3.5, and we therefore plot the curves of $f(m)=\frac{N_n}{m^2}$ and $f(m)=\frac{N_n}{2mp}$

that dominate T in figure 3·4.

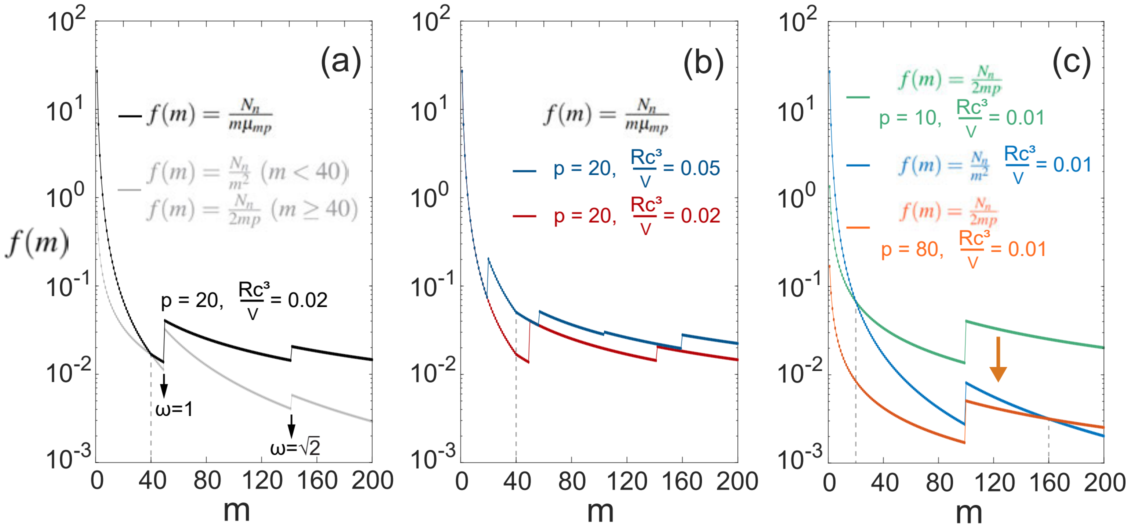


Figure 3·4: Latency analysis with respect to cell granularity.

Figure 3·4(a) shows that $f(m) = \frac{N_n}{m\mu_{mp}}$, the key factor to T, is a splicing of two curves. We take $p = 20$ and $\frac{R_c^3}{V} = 0.02$ for example, and two sharp leaps are observed on the curve. The splicing occurs at $m = 2p$, where the number of cells matches the number of filters. The gray curve below the black curve represent the discarded portions of function $f(m) = \frac{N_n}{m^2}$ and $f(m) = \frac{N_n}{2mp}$. We can easily notice the two leaps on the curves. These leaps correspond to $m = 49$ and $m = 141$, where the former leads to $\omega = 0.993 \approx 1$, right before reaching the edge length of a cell, and the latter leads to $\omega = 1.413 \approx \sqrt{2}$. This observation reveals that the boundary effect (drastic increase in N_n with slight ω increase) results in a significant amount of performance loss.

Figure 3·4(a) depicts the function $f(m) = \frac{N_n}{m\mu_{mp}}$ in black, which is a splicing of two curves. The splice point occurs at $m = 2p$, where the number of cells aligns with the number of filters. The gray curve, located below the black curve, represents the discarded portions of the functions $f(m) = \frac{N_n}{m^2}$ and $f(m) = \frac{N_n}{2mp}$. For example, we consider the case of $p = 20$ and $\frac{R_c^3}{V} = 0.02$. We can clearly observe that the $f(m) = \frac{N_n}{m\mu_{mp}}$ curve exhibits two distinct sharp leaps. Specifically, the first leap occurs at $m = 49$, which corresponds to

$\omega = 0.993 \approx 1$, just before reaching the edge length of a cell. The second leap occurs at $m = 141$, leading to $\omega = 1.413 \approx \sqrt{2}$. This observation highlights the impact of boundary effects, where a slight increase in ω results in a significant rise in N_n, leading to a notable performance loss.

Figure 3·4(b) compares two cases with different cutoff-to-volume ($\frac{R_c^3}{V}$) ratios. The ratio reflects the relative size of a simulation space with respect to cutoff. In the case of a smaller space ($\frac{R_c^3}{V} = 0.05$), it is clear that the leap occurs when there is a smaller amount of cells, and the performance loss is much greater at the first leap compared to the $\frac{R_c^3}{V} = 0.02$ case (note the log scale of y axis). We also observe that the first leaps are the most crucial among all the leaps with the greatest performance loss.

Figure 3·4(c) shows increasing p only leads to the vertical shift of $f(m) = \frac{N_n}{2mp}$ without affecting the leap properties. The shift is also obvious from its equation form.

To summarize, to conserve the on-chip memory resources while maintaining high performance, a cost-effective way is to let the normalized cell size right above the cutoff, i.e.,

$$\omega = \lim_{\delta \to 0^+} (1 - \delta) \tag{3.9}$$

such that any leap in the latency T is avoided. In the remaining sections of this dissertation, R_c is always chosen as the cell side length unless further noticed.

3.2 Memory to PE Mapping Schemes

In this section, we explore three distribution schemes that enable the mapping of work from cells to processing elements (PEs). A PE refers to a force computing unit along with its associated logic, which includes filters and accumulators. The particle data within cells is stored in various memory components, namely, HC (position caches in home cells), NC (position caches in neighbor cells), and Force Caches (containing forces of particles within individual cells). For the sake of simplicity, we disregard the data paths related to motion update units, as they solely read from force caches in cells and write to position and velocity caches within the same cells, contributing minimal complexity.

3.2.1 All PEs Work on the Same Reference Particle

Algorithm 1 High Level Mapping 1

```
for each cell $hcell$ in all cells as the home cell do                      ▷ Home cell loop
    for each particle $p$ in $hcell$ do                                     ▷ Home particle loop
        for each cell $ncell$ in neighbor cells do parallel   ▷ Neighbor cell loop, parallel
            for each particle $q$ in $ncell$ do parallel   ▷ Neighbor particle loop, parallel
                $f_{pq} = ComputeForce(r_p, r_q)$
            end for
        end for
    end for
end for
```

In Figure 3·5(a), one single reference particle from the home cell is broadcast to all filters in all PEs. The workflow is abstracted in Algorithm 1. In the meantime, the particles from neighbor cells are sent to filters to pair with the reference particle.

This mapping approach enables the accumulation of partial forces of the reference particle using an adder tree before storing them in the force cache. This reduces the data traffic involved in storing the force back to the force cache. Additionally, the partial forces of the neighbor particles are accumulated individually into the force caches using the adders associated with their respective cells. This method ensures workload balance at the particle pair level.

However, the disadvantages are obvious. First, the cost for maintaining data locality is high. Although the neighbor particles fetched can be reused for all the particles broadcast from the home cell, the number of neighbor particles is too high ($\sim$1000, assuming we have enough PEs, $\sim$150) for the particles to be cached in registers for concurrent reading. Second, all the neighbor force fragments are returned to only 14 neighbor cells, which requires tremendous bandwidth per cell to perform the accumulation without conflict.

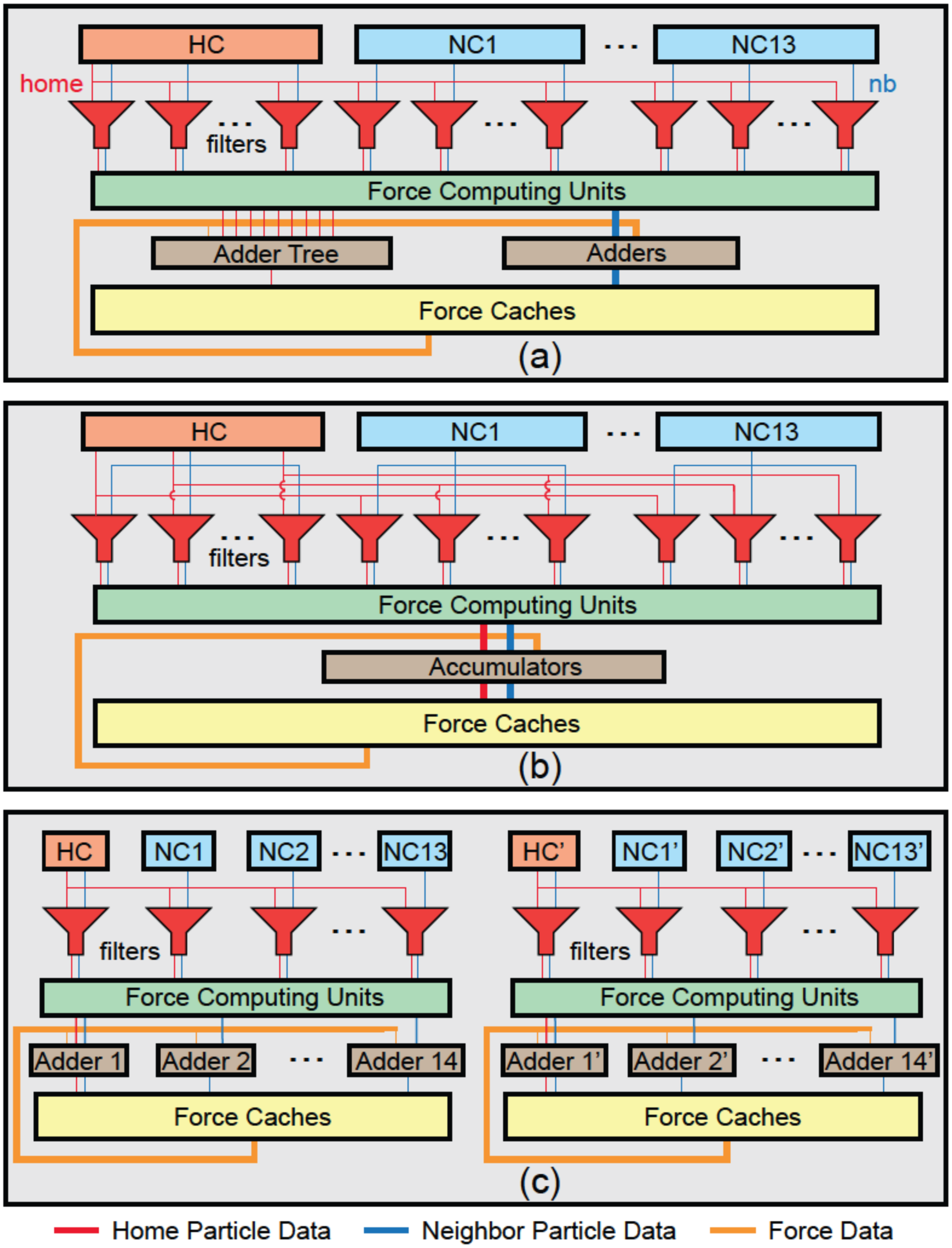


Figure 3·5: High level memory-to-PE mapping schemes. (a): particle-centric, all PEs work on a single reference particle at a time. (b):cell-centric, all PEs work on a same cell at a time. (c): uniformly-spreaded, each PE works on a separate cell.

3.2.2 All PEs Work on the Same Home Cell

Figure 3·5(b) shows that the home cell broadcasts multiple particles to the filters. Meanwhile, each neighbor cell only needs to broadcast one particle to the filters.

Algorithm 2 High Level Mapping 2 Original

```
for each cell hcell in all cells as the home cell do                    ▷ Home cell loop
    for each particle p in hcell do parallel                ▷ Home particle loop, parallel
        for each cell ncell in neighbor cells do parallel ▷ Neighbor cell loop, parallel
            for each particle q in ncell do                        ▷ Neighbor particle loop
                $f_{pq} = ComputeForce(r_p, r_q)$
            end for
        end for
    end for
end for
```

In this mapping scheme, instead of evaluating neighbor particles concurrently with a single reference particle at the same time, we evaluate home particles concurrently with a neighbor particle from each neighbor cell instead as Algorithm 2 suggests. As a result, the spatial locality is achieved for both home and neighbor cells as every particle is used in multiple filters, and the demand for force write-back bandwidth is drastically decreased.

Algorithm 3 High Level Mapping 2 Reordered

```
for each cell ncell in neighbor cells do parallel           ▷ Neighbor cell loop, parallel
    for each particle q in ncell do                                ▷ Neighbor particle loop
        for each cell hcell in all cells as the home cell do            ▷ Home cell loop
            for Home particle p in hcell do parallel       ▷ Home particle loop, parallel
                $f_{pq} = ComputeForce(r_p, r_q)$
            end for
        end for
    end for
end for
```

In practical scenarios, the locality of neighbor particles is more significant than that of home particles since a home cell typically involves more neighbor particles in its evaluation. By improving the locality of neighbor particles, we can increase the opportunities for accumulation before force write-back, resulting in bandwidth savings. One simple approach to enhance the locality of neighbor particles is by reordering the loops, as demonstrated in Algorithm 3. This algorithm caches the neighbor particles in registers and does not require fetching any new neighbor particles until the current set of neighbor particles has been processed with all home cells. This allows for the accumulation of neighbor forces, reducing the number of fine-grained neighbor force fragments and further conserving bandwidth during force write-back.

Still, only 14 cells are evaluated and the rest of the cell storage remains idle, unless an expensive memory interleaving method is applied for dynamically changing memory contents. Also, a heavy bandwidth requirement persists in the broadcast of the home cell position.

3.2.3 Each PE Works on a Different Cell

Algorithm 4 High Level Mapping 3 original

```
for each cell hcell in all cells as the home cell do parallel   ▷ Home cell loop, parallel
    for each cell ncell in neighbor cells do parallel           ▷ Neighbor cell loop, parallel
        for each particle p in hcell do                         ▷ Home particle loop
            for each particle q in ncell do                     ▷ Neighbor particle loop
                $f_{pq} = ComputeForce(r_p, r_q)$
            end for
        end for
    end for
end for
```

Figure 3·5(c) shows two individual PEs, unlike the former cases where the PEs are blended. In this case, a PE only works on one single home cell, which broadcasts only one particle to all the filters in one PE at a time, while each of the 14 neighbor cells sends one particle to a filter in the PE. In fact, a neighbor cell broadcasts the single particle to 13 other home cells as well. For instance, NC1 and NC2' in Figure 3·5(c) may correspond to the same cell in the simulation space.

Algorithm 5 High Level Mapping 3 reordered

```
for each home cell $hcell$ in all cells do $parallel$          ▷ Home cell loop, parallel
    for each cell $ncell$ in neighbor cells do $parallel$      ▷ Neighbor cell loop, parallel
        for each particle $q$ in $ncell$ do                    ▷ Neighbor particle loop
            for each particle $p$ in $hcell$ do                ▷ Home particle loop
                $f_{pq} = ComputeForce(r_p, r_q)$
            end for
        end for
    end for
end for
```

This mapping method addresses both the bandwidth and the memory idle problems. First, at most one particle is broadcast from any cell, allowing concurrent position reading without requiring a large number of BRAMs per cell. Second, now that multiple home cells can be evaluated, the PEs can operate simultaneously across the simulation space for optimal memory utilization. Third, similar to the second mapping scheme, the force write-back data transfer rate can be further reduced by switching the inner and outer loops to enhance neighbor particle locality, as shown in Algorithms 4 and 5.

3.3 Data Format

In this section, we explain the the fundamental data formats used in our work, as well as the merits brought by the choices of data formats. In this dissertation, a 23-bit fixed-point data format is adopted for position storage, while the 32-bit single precision floating point format is applied to velocity and force data throughout the designs.

3.3.1 Position Data Format

The 23-bit fixed-point position in our system is normalized to the cell edge width with the cell edge width defined as 1. The normalized position represents the offset from the boundary of a cell. This normalization ensures that the decimal point is fixed right before the 23 bits. Through this normalization process, the 23 bits achieve the similar level of accuracy as single-precision floating-point numbers. The 23 bits correspond to the 23-bit mantissa in floating-point numbers (with the ghost bit set as 0 instead of 1, potentially losing 1-bit information), equivalent to fixing the 1-bit sign and the 8-bit exponent to constants in floating-point.

As position is a three-dimensional quantity, the fixed-point format requires 69 bits, while the floating-point format demands 96 bits. This difference in data width is significant, especially when considering the storage requirements on FPGAs. For example, an Intel M20K Block RAM has a width of 40 bits, while a modern Xilinx FPGA Block RAM typically has a width of 36 bits. With the fixed-point format, only 2 BRAMs side-by-side are needed for storage, whereas the floating-point format requires 3 BRAMs. Here 23 instead of 24 bits are used because the remaining 3 bits for Xilinx BRAMs can be used to encode particle type.

Apart from storage savings, the fixed-point format also presents opportunities for filtering. As discussed earlier, planar filters take advantage of the reduced number of bits. With 75% fewer bits (if from 32 to 8) involved, the planar filters allow 3% more pairs to pass through. In the fixed-point format, obtaining the 8 leading bits is straightforward. In practice, the cell position information is padded before the 8 bits, resulting in approximately 10 bits in the planar filters, as described in the subsequent discussion.

While the leading bits can also be obtained from a single-precision floating-point number normalized to 1, the floating-point format has a major disadvantage in terms of particle interactions across cells. In the fixed-point format, when two particles are from different cells, their relative positions can be directly padded before the fixed-point bits. For instance, if particle A has a position of 1.5 along the x-axis and particle B has a position of 2.4, where 1 and 2 before the decimal points represent the padded cell locations along the x-axis. However, when padding cell information to a floating-point number, the exponent component needs to change, making the application of planar filters more challenging.

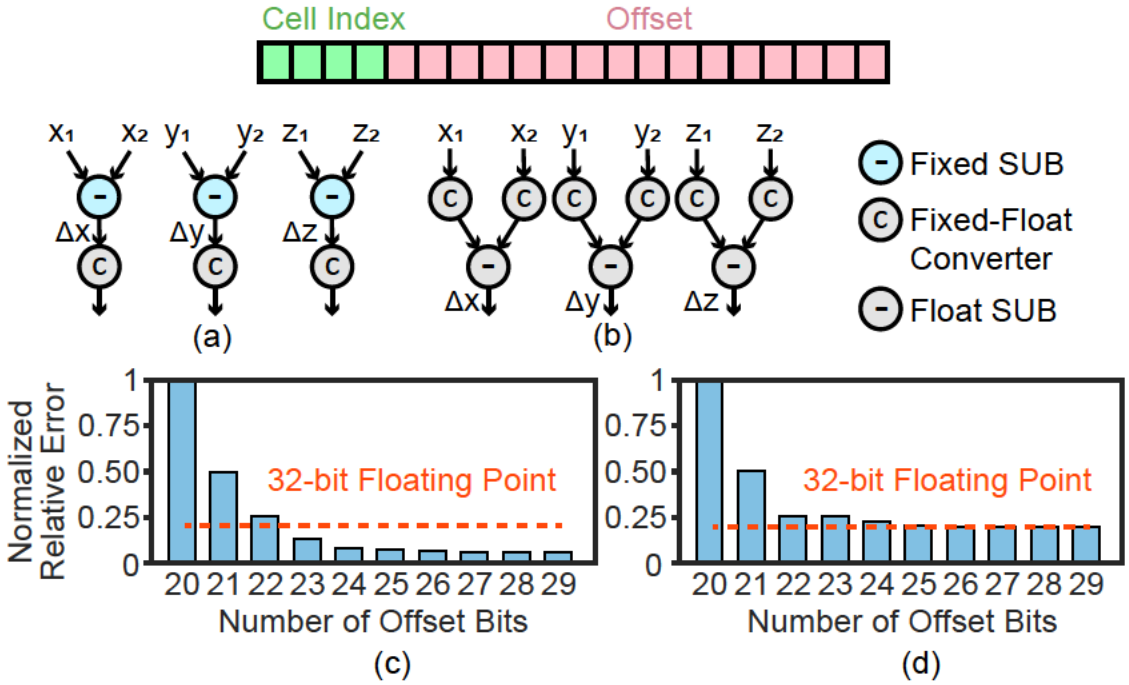


Figure 3·6: The data format and precision analysis. Top: The fixed-point position format. (a) and (b): Two methods of computing the displacements of particles. (c) and (d): Error analysis of the two methods with different numbers of bits.

During force evaluation, the fixed-point positions need to be converted to single-precision floating-point numbers to compute r^2. Figure 3·6 illustrates two scenarios of data type conversion. In Figure 3·6(a), the fixed-point positions are subtracted before being converted to 32-bit floating-point format. This approach saves floating-point subtraction operations but presents a challenge when the displacement, or position difference, is small. In such cases, the conversion becomes costly due to the difficulty in finding the leading "1" bit, which determines the location of the decimal point.

The alternative method performs the fixed-to-float conversion before subtraction, introducing complexity of floating-point operations. However, this conversion is nearly cost-free, as the challenge of finding the leading "1" bit can be easily resolved. By starting the cell index from 1 instead of 0 and padding the cell position information, the decimal point is guaranteed to be before the 23 bits. Consequently, the leading 1 bit can be easily obtained among the cell index bits in the front, which are considerably fewer in number compared to the 23 bits. After the conversion, the position differences dx, dy, and dz are obtained through floating-point additions, and r^2 can be computed accordingly. Importantly, this conversion is only performed for particle pairs that pass the filters, and each force evaluation unit requires only one converter.

On an Intel Stratix 10 FPGA, the converter in the second method consumes approximately 15 ALMs, while the first method requires around 170 ALMs. The converter in the second method also improves latency, taking only 1 cycle compared to approximately 5 cycles in the first method. However, the second method requires three additional DSPs per force evaluation unit.

In Figure 3·6(c) and (d), the normalized relative error results for computed forces with interpolation (the interpolation method will be discussed in the following section) are compared to 64-bit floating-point forces directly computed, utilizing 3 bits of cell indices. The absolute relative error typically varies with interpolation parameters and system parameters such as σ and is on the order of 10^{-5}. The 32-bit floating-point result, starting with a cell index of 0 instead of 1, is slightly more accurate than using the 24-bit offset since the mantissa actually represents 24-bit precision with the ghost bit.

3.3.2 Force-Related Data Format

Single-precision floating-point data format is adopted for forces and r^2. This comes with two main reasons. First, the format naturally emphasizes the precision of small r^2, which contributes to large forces that are significant. Second, the force exhibits a wide data range, making it hard for fixed-point numbers to represent.

First, during force computation, instead of computing the r^{14} and r^8 terms directly at high hardware cost, we utilize a force table-lookup technique as described in equation 3.10-3.12 and figure 3·7. In order to get the value of $r^{-\alpha}$, we use linear interpolation method as Equation 3.10 shows, where a and b are lists of parameters indexed by two indices s and b introduced in Equation 3.11 and 3.12. Since r^2 is a floating point number, the range of data

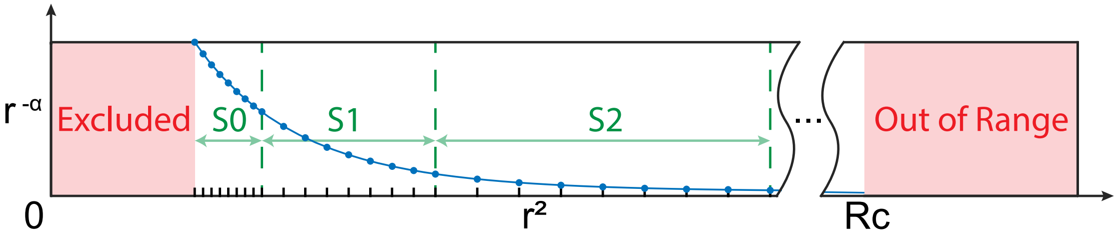


Figure 3·7: The interpolation method. α is a positive integer too large for $r^{-\alpha}$ to be computed. In our case, α is 8 or 14. The small r region is excluded due to non-physical high energy.

covered by r^2 is divided into n_s sections denoted as S0, S1, etc. based on the exponent bits of r^2. Furthermore, each section is split into n_b regular-sized bins based on the mantissa bits of r^2. That being said, the indices s (section) and b (bin) are obtained accordingly. It is also obvious in the figure that the precision is adaptive to the force significance, i.e., the resolution is higher for smaller r^2, which is subsequently translated to a higher resolution in larger forces.

$$r^{-\alpha} = a_\alpha(s,b)r^2 + b_\alpha(s,b) \tag{3.10}$$

$$s = \lfloor \log_2(r^2) \rfloor + n_s \tag{3.11}$$

$$b = \lfloor (2^{n_s - s} r^2 - 1) n_b \rfloor \tag{3.12}$$

Second, the r^{14} and r^8 terms directly lead to a much wider range of forces compared to r^2. In order not to miss either large forces or small forces, floating-point is considered a better choice compared to fixed-point, due to the use of exponent bits which significantly extends the data range it expresses.

3.4 Summary

The above theoretical analysis of cell size, and the high-level memory-to-PE mapping schemes establish the foundation of the hardware architectures to be discussed in the following chapters.

Except for the architecture discussed in Chapter 6, where a modified Manhattan method is employed instead of the previously discussed half-shell method, the cell edge length of R_c is utilized in all other architectures. In Chapter 6, although the cell shape becomes irregular, it is still based on the R_c cell edge length.

The mapping schemes presented in this section will be further explored in the subsequent chapter along with their corresponding architectures. Specifically, the first and third mapping schemes will be emphasized as two contrasting approaches. The first scheme is particle-centric and focused on depth, while the third scheme is characterized by a more uniform distribution and breadth. The second scheme serves as an intermediate method, between the first and third schemes.

Chapter 4

Single-FPGA Architectures

This chapter begins by presenting the baseline architectures, which are derived from the three mapping schemes discussed in Section 3.2. Subsequently, we introduce two optimized architectures that align with the first and third mapping schemes, namely, the particle-centric and uniformly-spread architectures.

4.1 Introduction

The mapping schemes brings us to the essence of the single-FPGA (or ASIC) scalability problem: after the high-level mapping scheme is determined, how can a scalable data path from memory to PEs be constructed. Directly connecting memory to PEs in a 3-D application that is mapped to a 2-D FPGA fabric leads to a complex interconnect between PEs and memory, resulting in a significant decrease in frequency. For instance, in the 512-node ($8\times8\times8$) Anton 2 machine, even with its direct 3-D torus topology to deploy such a 3-D application, the longest connection between two nodes is still almost two meters [Towles et al., 2014]. On the other hand, in a single FPGA, it is not practical to directly map the 3-D application to the 2-D FPGA structure due to space and wiring constraints. Therefore, a transposed memory block method and ring routing are proposed and proven adequate for RL force evaluations (as shown in the following discussions section 4.3).

However, the routing method in the ring architecture requires more cycles and can lead to network congestion, resulting in reduced throughput. To address this issue, we have developed several techniques. Firstly, a data caching method has been implemented to minimize data transfer demands and improve data locality (see Section 4.3.2). Additionally, instead of relying on bulk synchronization, we propose a novel dynamic data dispatching method to allow the processing elements (PEs) to work continuously without frequent synchronizations (see Section 4.3.2). Moreover, an out-of-order data broadcast mechanism has been devised to dynamically fill empty slots in the ring and eliminate bubbles in the network (see Section 4.3.2). These techniques collectively aim to enhance the performance and efficiency of the system.

An important consideration in these designs is to ensure compatibility with a third standard optimization technique: prefiltering particle pairs within neighboring cells to eliminate pairs where the mutual force is negligible, similar to the concept of neighbor lists in CPU implementations [Chiu and Herbordt, 2010a]. In theory, for uniformly distributed particles, the average pass rate for a filter is approximately 15.5%, and around 17% for a planar filter [Chiu and Herbordt, 2009]. In our work, we enhance the conventional filters by introducing hierarchical filters (see Section 4.3.2). This approach not only increases the pass rate to 25%, but also helps alleviate the data transfer pressure.

To summarize this introduction:

a new generation FPGA-based MD RL accelerator is proposed with the following major contributions.

- An optimized FPGA-based MD RL accelerator with several design variations that can process 50K particles without off-chip memory.

- On system-level, three particle/cell to PE mapping schemes are proposed and evaluated in depth.

- The ring routing method is studied to match the 3-D to 2-D mapping of the up-scaling number of PEs on a single FPGA chip. With an out-of-order data broadcast mechanism and a data caching mechanism applied, the latency and concurrency problem introduced by the ring are solved.

- Other optimizations, for example, the hierarchical filtering method and memory recycle mechanism are proposed; they reduce hardware consumption and improve hardware efficiency.

4.2 Baseline Architectures

In this Section, we introduce the detailed baseline designs corresponding to the high level memory-to-PE mapping schemes discussed in section 3.2.

4.2.1 Design 1: Particle Centric

The particle-centric design follows the first high level mapping scheme, where all PEs focus on a single reference particle at a time.

The design layout is illustrated in Figure 4·1(a). We assume there are *m* cells in the simulation space, each equipped with a set of block RAM-based caches, including a position cache, a force cache, and a velocity cache. These caches store particle information at consecutive addresses based on particle IDs.

The force evaluation phase begins with the position caches, where neighbor particle positions are distributed to individual filters for pairing with a reference particle. This distribution is carried out by a position distributor (Figure 4·1(d)), which selects particle position data from 14 specific position caches corresponding to the neighbor cells. The filters are partitioned into 14 groups, with each group only receiving neighbor particles from a single neighbor cell. Upon receiving a neighbor particle, it is stored in a chain of registers as indicated by the blue arrows. Once all the neighbor particles are pre-loaded into the registers, the filtering process begins. The reference particles in a cell are streamed and broadcast to all the filters. The streaming reference particles are then paired with the registered neighbor particles. If a reference-neighbor particle pair falls within the cutoff radius, it passes the filter and is buffered for force computation.

As explained in Section 3.2.1, the system is equipped with approximately 1000 filters, enabling the evaluation of a reference particle in a single cycle. Typically, a cell contains fewer than 80 particles, such as hydrogen with a Van der Waals radius of 0.12 nm, while the cutoff radius is typically around 1 nm. Consequently, the 14 neighboring cells collectively contain approximately 1000 particles, aligning with the number of filters available. Based on equation (2.6), a force computing unit can process particle pairs from seven filters simultaneously to maintain a consistent workload. In cases where the number of filters is insufficient to accommodate all the neighbor particles, it may require two or more cycles to evaluate a reference particle.

The force outputs from the force computing units are fed into two sets of adders. Firstly, the partial forces of a reference particle (reference forces) are accumulated using an adder tree and then stored in a force cache. The adder tree processes one reference particle at a time, resulting in bubbles between the evaluations of two reference particles. In the worst-case scenario, it requires seven cycles to evaluate a reference particle when all seven upstream filters of a force computing unit have the reference-neighbor particle pairs that pass the filtering process. Secondly, the forces applied to neighbor particles (neighbor forces) are registered and accumulated in separate adder groups (Figure 4·1(e)) during the evaluation of all the reference particles in a cell, and then accumulated into their respective force caches. In other words, the neighbor forces are accumulated into larger force chunks before being written to the force caches. Nonetheless, there are still approximately 1000 neighbor force chunks in total that need to be accumulated into 14 force caches. To address the mismatch between the force throughput and the force cache bandwidth, additional force caches can be employed if there are available block RAM resources. For example, a cell may be equipped with multiple force caches to achieve higher bandwidth. However, the key problem lies in the fact that there are *m* force caches available, but only 14 of them are utilized at any given time.

The motion update mechanism remains the same for all three baseline architectures and is therefore to be discussed after the descriptions of the baselines.

4.2.2 Design 2: Cell Centric

The cell-centric design corresponds to the second high-level mapping scheme, where all PEs are dedicated to a reference cell as the home cell. The design is illustrated in Figure 4·1(b), which shares a similar architectural layout with Figure 4·1(a), but demonstrates notable differences in its operation. We will describe the workflow by emphasizing the contrasts between this design and the previously discussed particle-centric design.

Firstly, the exploitation of position data locality differs between the two designs. In Design 1, neighbor particles are cached in a sequence of registers to optimize neighbor position locality, while reference particles naturally exhibit good locality as they are broadcast to all filters. However, in the current baseline design, the locality of reference particles is compromised, as the focus is on all reference particles within a cell rather than a single particle. As a result, neighbor locality is enhanced, as a neighbor particle is concurrently utilized in multiple filters.

Secondly, a difference lies in the filter grouping method. In Design 1, all filters are divided into 14 groups, with each group potentially containing tens of filters. However, in the current baseline design, a filter group consists of precisely 14 filters. Starting from a position cache, reference particles are sequentially broadcast to the filter groups and stored in registers. Each filter group is specifically designed to receive particle position data from 14 neighbor cells. Unlike Design 1, where neighbor particle positions are processed within each filter group, in the current design, the neighbor particle positions flow through the filters and are paired with the registered reference particles. More specifically, a neighbor particle is utilized in all filter groups to be evaluated with multiple reference particles from the home cell.

Thirdly, the force accumulation is modified accordingly. Since multiple reference par-

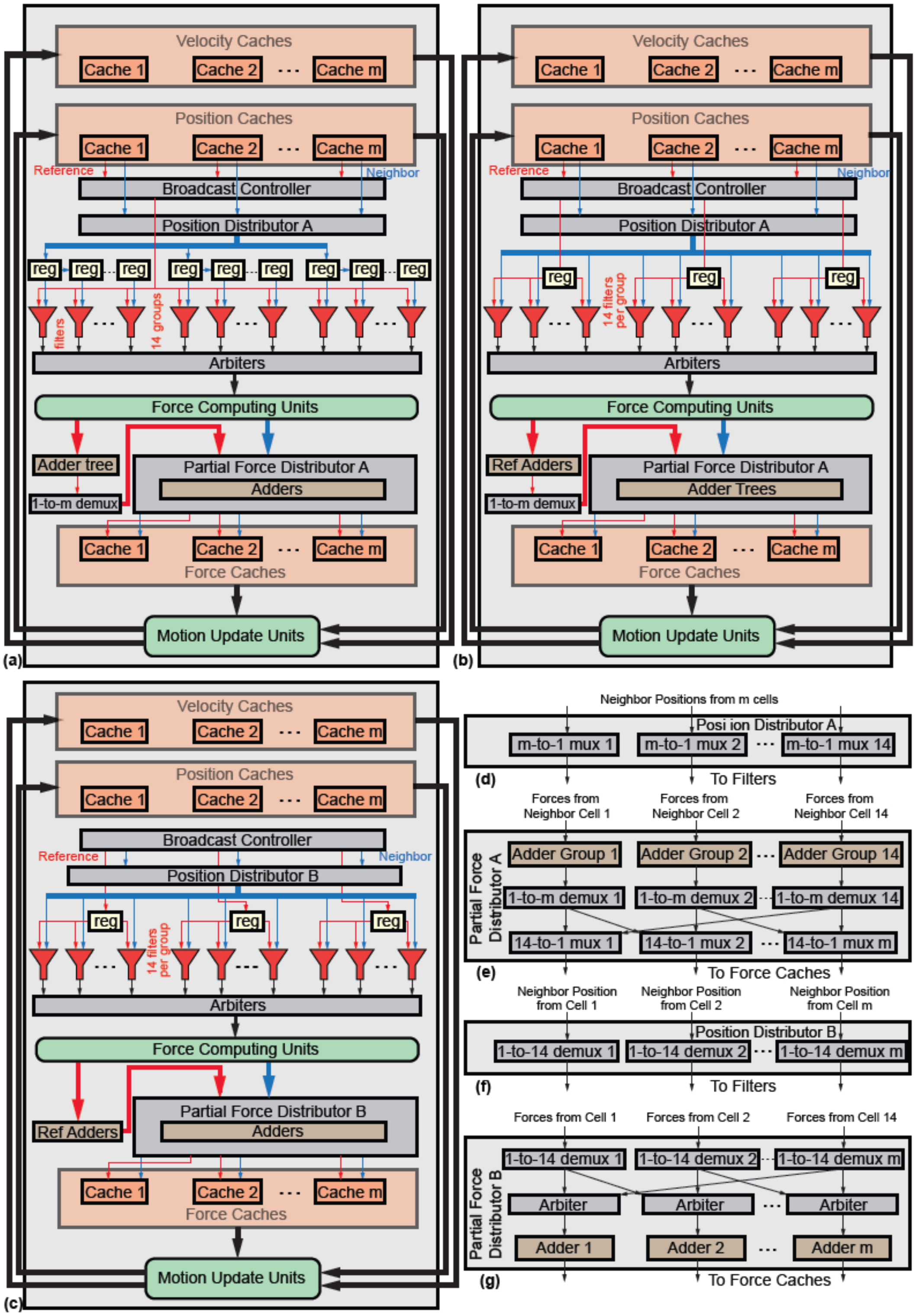


Figure 4·1: The baseline designs and detailed distributor architectures. (a)-(c): baseline design layouts. (d)-(g): details of position and force distributors.

ticles are now being processed simultaneously, the adder tree used in Design 1 is replaced with individual regular adders for each reference particle. Specifically, each adder is dedicated to accumulating the forces of a single reference particle in a filter group. The reference forces are first accumulated, and then directed and integrated into the force caches through the partial force distributor. In terms of the neighbor forces, a neighbor particle is shared among multiple filters at the same time, allowing for the accumulation of neighbor forces before being written to their respective force caches. Consequently, adder trees are required to manage the high throughput of the neighbor forces in coordination with the limited number of force caches.

Lastly, synchronization is demanded for neighbor partial forces in the adder trees. To ensure that the input forces to an adder tree come from the same neighbor particle, stalls are introduced to prevent the dispatch of neighbor forces from other particles before the previous forces are fully processed. This means that the neighbor particles are organized into batches, with each batch consisting of 14 particles corresponding to the 14 neighbor cells. It is essential to complete the evaluation of a batch before allowing the partial forces of the next batch to enter the adder trees. This mechanism is similar to the data handling in the adder tree in Design 1, as the goal is to prevent forces of different particles entering the adder tree at the same time.

4.2.3 Design 3: Uniform Spread

In this design, our focus shifts from processing a single reference particle or reference cell at a time to simultaneously handling all cells. The key distinctions lie in the specific aspects we will discuss in this section.

Firstly, there are significant differences in data locality compared to the previous two designs. In Figure 4·1(c), we assume m cells, and each of the m position caches broadcasts a reference particle from a home cell to its designated filter group. Unlike the previous designs, this uniform spread design eliminates the need for particle pre-loading, effectively reducing latency overhead. Each reference home cell particle serves a single filter group, similar to Design 2, but now with multiple home cells involved. On the other hand, the locality of neighbor particles is reduced compared to Design 2. A neighbor particle is only used in 14 filters within different filter groups corresponding to 14 home cells, whereas in Design 2, a neighbor particle is used in all filter groups.

Secondly, the position distribution method differs from the previous two designs. After the reference particles are registered, the position caches start distributing neighbor particles to the filters in streams. Figure 4·1(f) illustrates the position distribution mechanism, where the key difference lies in cell usage. In Design 1 and 2, only 14 cells are utilized at a time, while in the current design, all cells are utilized. Once all the current reference particles are evaluated, the position caches dispatch the next reference particles to the processing elements.

Thirdly, synchronization methods vary. Synchronization in Design 1 and 2 primarily stems from the adder trees, whereas in the current design, no adder tree is required. However, synchronization is necessary in the filters. Firstly, the position broadcast mechanism requires all processing elements to receive particles with the same ID, which facilitates starting with the same particle ID for all processing elements, thus necessitating global synchronization in case the position caches have different numbers of particles. Secondly, the reference position is not stored in filter buffers (between filters and arbiters, not shown in the figures), but in a register shared by all the filters and force computing units. Consequently, the reference position cannot be discarded until all related operations are completed.

Lastly, the force accumulation mechanism varies. The reference particles exhibit good temporal locality, allowing for direct accumulation of reference forces into their respective force caches using adders. On the other hand, the neighbor forces are highly fragmented and flood into force caches for accumulation. Adder trees are not utilized due to limited resources, as all cells are processed simultaneously. The force distribution mechanism (Figure 4·1(g)) is also adjusted from the previous designs.

4.2.4 Motion Update and Particle Migration

All three baseline designs utilize a cell-based cache data structure and employ the same motion update method.

After the force evaluation phase, the motion update units simultaneously request position, velocity, and force data from individual caches. Once requested, the force data is transmitted to the motion update units while the original entries in the force caches are erased to prepare for the next force evaluation phase.

However, when dealing with particle migration, simply sending the updated position and velocity data back to overwrite the cache contents is not feasible. This limitation is illustrated in Figure 4·2(a), where a particle being migrated out from Cache 1 leaves an empty slot in the cache. Without further optimization, a migrating particle would need to search for an empty slot over multiple cycles.

A more sophisticated approach involves using a pointer to track the previously evaluated particle, as depicted in Figure 4·2(b). In this method, the local particles are updated first, while the migrating particles are temporarily stored in a FIFO buffer. As the 3rd particle migrates away, the pointer points to the vacant slot, allowing the 4th particle to be updated in the 3rd slot, with the pointer shifting to the 4th. Consequently, the non-migrating local particles are compactly stored in the cache. Subsequently, the migrated particles are read from the FIFO buffer and integrated into the cache in the correct order.

However, the advanced method using a FIFO buffer incurs additional hardware costs that are comparable to implementing another cache itself. Therefore, a simpler and more efficient approach is to employ two caches to address this issue, as demonstrated in Figure 4·2(c). When a particle migrates in, the local update process is paused, and the new particle is written to Cache 2 in a sequential manner. Eventually, all updated particles are stored

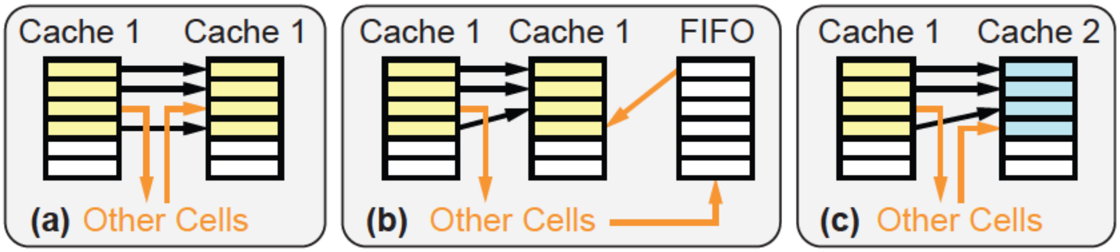


Figure 4·2: Particle migration handling methods. (a): directly searching for empty slots. (b): the one-pointer method. (c): double buffering.

in Cache 2 while Cache 1 becomes outdated. In the next molecular dynamics (MD) iteration, the data from Cache 2 are utilized, and the particle statistics are updated in Cache 1. This double-buffering technique allows for seamless integration of migrated particles and maintains data consistency for the motion update process.

4.3 Optimized Designs

The baseline designs mentioned above exhibit certain drawbacks that need to be addressed. Firstly, all three designs encounter challenges related to complex data routing paths, as illustrated by the position and force distributors depicted in Figure 4·1. The high fan-in and fan-out characteristics of these designs not only negatively impact the operating frequency but also tend to exhaust the available wiring resources.

Secondly, the baseline designs fail to fully exploit the potential benefits of data locality. For instance, in Design 1, approximately 1000 neighbor forces are accumulated independently, leading to a significant strain on the computing resources. This inefficiency hampers the overall performance of the system.

To tackle these fundamental issues inherent in the baseline designs, we propose two optimized design alternatives that build upon the concepts of Design 1 and Design 3. These optimized designs aim to address the challenges associated with complex data routing paths and underutilized data locality. In addition to resolving these fundamental problems, these optimized designs offer additional benefits that enhance the overall efficiency and effectiveness of the system.

4.3.1 Optimized Design 1: Transposed Memory Blocks

The Optimized Design 1, is derived from Design 1, where a single reference particle is paired with all its neighbor particles. The complete architecture of Optimized Design 1 is presented in Figure 4·3(a). While storing particles by cell, as employed in Baseline Design 1, is an effective approach for organizing data, it is not well-suited for the workload mapping scheme adopted in this optimization. In the particle-centric scheme, only 14 cells are active at any given time, and the performance is constrained by the throughput of the particle data caches. Therefore, the objective of this optimization is twofold: maximizing the utilization of available caches to meet the data bandwidth requirements, while maintaining the organization of particles.

The key solution lies in the transpose of the original block RAM arrays, as depicted in Figure 4·3(b). Instead of storing particles by cells, the block RAMs now store particles based on their particle ID, with the entries representing different cell IDs (assuming there are m cells). This transpose allows for improved data access patterns, enabling better utilization of the caches while preserving the organization of particles.

Optimized Data Paths

One key improvement lies in the reorganization of the position data distribution path to filters. In the physical sense, the number of particles per cell is typically fewer than 80, which means that the required number of block RAMs is limited. With each cache now responsible for storing particles with the same particle ID from different cells, it becomes possible to load all neighbor particles related to a reference particle into the registers above the filters in just 14 cycles. This is a significant improvement compared to the baseline design, which requires approximately x cycles due to the limited active position caches (x is the number of particles per cell, typically $\sim$60 in water environment). Moreover, these neighbor particles can be reused for other reference particles from the same cell, further optimizing the data flow.

Figure 4·3(a) illustrates the architecture of Optimized Design 1, highlighting the high localization of the position caches and filters. In this optimized design, each filter group is exclusively served by neighbor particles from the same position cache. This design choice eliminates the need for extensively overlapping position data paths and greatly reduces the fan-in/fan-out complexity observed in the baseline design. In Design 1, in contrast, the position caches and filters are interconnected almost in an all-to-all manner, leading to inefficient data routing and management.

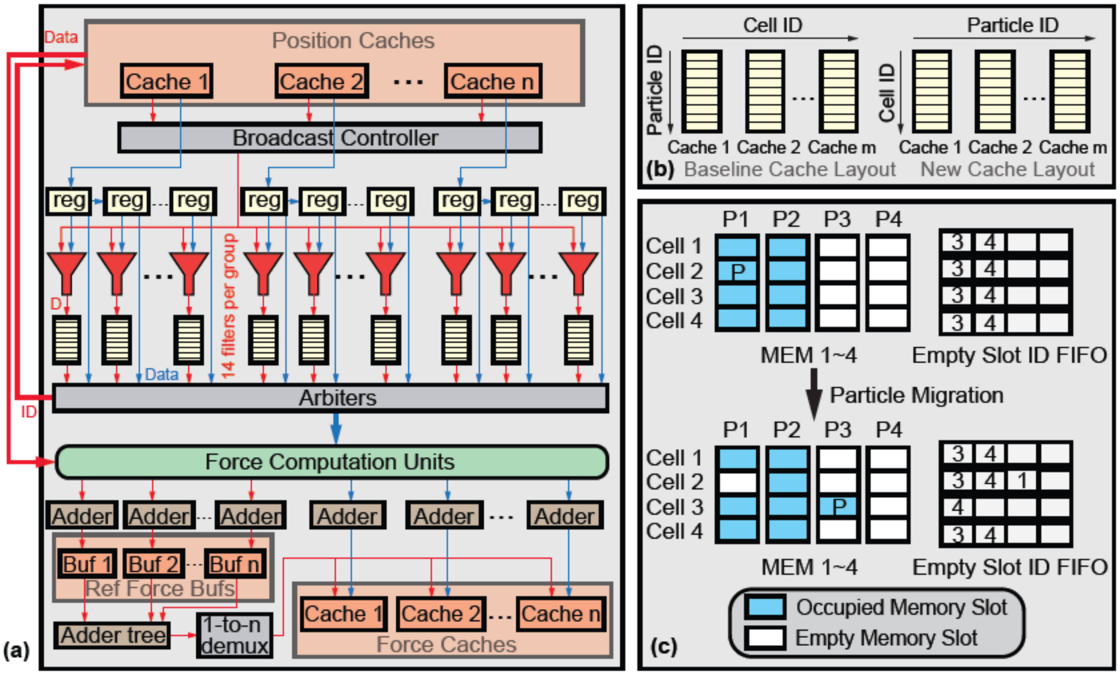


Figure 4·3: The optimized architecture for the first mapping scheme. (a): the data flow in force evaluation. For simplicity, arbiters and force computational units are abstracted as entire blocks, and the forces read from force caches for accumulation are omitted. (b): the new memory layout compared to baseline. (c): particle migration handling in motion update. P is the particle being migrated from cell 2 to cell 3.

Optimized Design 1 not only simplifies the data paths of neighbor forces but also leverages the locality of reference particles for improved performance. In the baseline design, each filter group evaluates neighbor particles from various particle IDs, leading to complex data routing. However, in Optimized Design 1, a filter group only evaluates neighbor particles with the same particle ID. As a result, each neighbor force produced by a filter group can be directly accumulated into its designated force cache, eliminating the need for intricate data routing. It is worth noting that the mapping ratio of seven filters to one force computation unit is maintained, ensuring efficient utilization of resources.

Furthermore, the design exploits the locality of reference particles to enhance performance. In baseline 1, the adder tree used for force accumulation could introduce bubbles and negatively impact performance. To mitigate this issue, a buffer array is inserted between the adder tree and the force computation units in Optimized Design 1. Unlike the force caches, all n buffers store reference forces from the same cell, identified by particle ID. Once a reference particle is evaluated and its reference force fragments are fully accumulated into the buffers, the accumulated reference force chunks are sent to the adder tree for an all-reduce operation. The resulting reference force is then accumulated into a force cache. By using buffers, the force computation process for a single reference cell can be pipelined without stalls. During the evaluation of a reference cell, the buffers are gradually cleared with the all-reduce operations conducted in the adder tree. The evaluation of the next reference cell begins once all the buffers have been cleared.

Another optimization is employed during data referencing. In Optimized Design 1, the buffers attached to the filters now only store neighbor cell IDs, which are much lighter in weight compared to particle positions. When a pair is selected by an arbiter, the corresponding neighbor cell ID is sent to the position cache to retrieve the required position data. This data is then transmitted to the force computation unit for further processing. This optimization reduces the amount of data transmitted and improves the overall efficiency of data referencing in the design.

Particle Migration Handling

The transposed caches in Optimized Design 1 offer new possibilities compared to the double-buffering migration handling method. Let's consider the example of four transposed caches shown in Figure 4·3(c). Initially, the last two caches are empty to ensure that no particles are lost due to migration. In practice, a slightly higher number of caches (around 25% more) are used for reservation purposes.

Suppose particle *P* is moving from Cell 2 to Cell 3. It checks the empty slot ID available from the empty slot ID FIFO of Cell 3 and moves to the corresponding slot. Simultaneously, the vacant slot ID is pushed into the empty slot ID FIFO. It is important to note that particle *P* may undergo further updates in Cell 3 during parallel motion update, but the data is not harmed because the force acting on particle *P* is 0, in other words, the corresponding force in the force cache is 0. The empty slot ID FIFOs required for this process are relatively small and can be implemented using a few registers.

As multiple iterations of the MD simulation progress, vacancies similar to the one created by particle *P* begin to populate the caches and are scattered among them. Surprisingly, these vacancies turn out to be advantageous rather than detrimental. Initially, the reserved empty caches have no workload to distribute to their filters or force pipelines downstream, leading to idle computing units. However, as the vacancies spread and migrate, the empty caches gradually fill up with migrating particles. Consequently, the workload becomes more balanced as the simulation proceeds, ensuring better utilization of computing resources.

High Spatial Capacity

The hardware resource requirements of the optimized design show minimal increase when scaling to a larger number of cells, even when scaled up to approximately 500 cells. In contrast to baseline 1 and baseline 2, where the complexity of wiring and resource consumption is largely determined by the number of cells, the memory block transpose method enables easy scalability of the cell count. When a new cell is added to the simulation space, the particles within it are evenly distributed to the caches. With each cache having a depth of 512, this data structure can handle up to 512 cells effectively.

If the number of cells exceeds 512, the design layout remains intact. The caches can be extended by incorporating additional block RAMs as the depth of a block RAM is normally 512, and minor adjustments can be made to the cell selection logic to accommodate the higher cell count. This scalability feature allows the optimized design to flexibly adapt to simulations with a larger number of cells without significantly increasing the complexity of the hardware resources or wiring.

Summary

To conclude, the Optimized Design 1 exhibits multiple advantages over the Baseline Design 1, including simplified and less entangled data path, more localized neighbor particle distribution, effective particle migration handling that automatically balances the workload, and the capability and flexibility in scaling of the simulation space.

4.3.2 Optimized Design 2: On-chip Ring Network

This architecture represents an advancement over the 3rd mapping scheme, where each processing element (PE) is assigned to a different cell. In Baseline Design 3, as the number of PEs increases, the wiring complexity escalates significantly. Furthermore, the neighbor particle data exhibits poor locality, with neighbor positions not being reused and neighbor forces not being aggregated before integration into force caches.

To address the routing challenge, we introduce a ring-based structure to replace the original direct connections. However, the utilization of a ring introduces longer latency and data life within the network, which can lead to congestion. In this section, we present a solution that minimizes data transfers within the network by localizing the neighbor particle

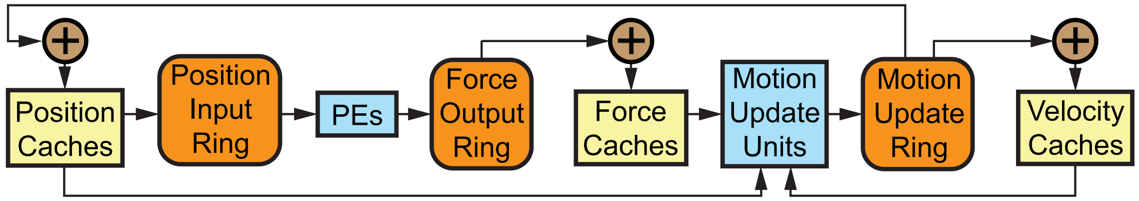


Figure 4·4: RL dataflow in brief with rings. Yellow: Memory blocks; orange: Routing rings; blue: computing units.

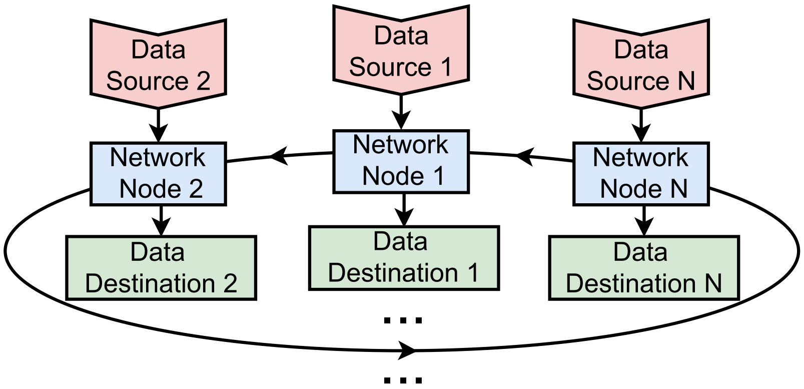


Figure 4·5: The ring routing path. Position input ring: from position caches to PEs. Force output ring: from PEs to force caches. Motion update ring: from motion update units to position caches and velocity caches.

data. This approach effectively reduces congestion, minimizes bubbles in pipelines, and decreases the number of invalid particle pairs.

Additionally, we discuss a method that reduces design redundancy. The double buffers previously employed in the motion update process are replaced with linked lists, resulting in a 50% reduction in memory usage for the position and velocity caches. This optimization enhances the efficiency and resource utilization of the architecture.

Memory-PE Interconnect

To address concerns regarding frequency degradation, we introduce a daisy-chain-based uni-directional ring interconnect system, as shown in Figure 4·4, to replace the direct interconnects depicted in Figure 3·5. The choice of a 1-dimensional (1-D) topology is based on its cost-effectiveness and the ability to hide the long latency associated with this configuration (as discussed later in this section). The routing rings are utilized for three types of data transfer: position caches to processing elements (PEs), PEs to force caches, and motion update units to position and velocity caches.

Ring Behavior: Each of the three rings exhibits different behavior. A position packet is broadcast to multiple destinations and remains within the ring until it reaches its final destination. In contrast, a force or motion update packet has a single destination. It is important to note that particles may migrate to other cells after each iteration, necessitating a ring for motion update packet routing. The motion update ring experiences much less congestion compared to the other rings due to the relative rarity of particle migration, thus requiring fewer details.

Figure 4·5 illustrates the interconnect of the ring nodes. Each ring node has two data sources and two destinations. New data from a source can only be injected into the network when the node is not occupied. A packet within the network is discarded once it reaches its final destination; otherwise, it is forwarded to the next node.

Ring Configuration: The mapping from 3-D cells to the 1-D ring is yet to be described. A straightforward mapping method is as follows:

$$I_r = N^y_{cell} N^z_{cell} (x-1) + N^z_{cell} (y-1) + z \tag{4.1}$$

Here, I_r represents the index shown in Figure 4·5, ranging from 1 to N, N^x_{cell} denotes the number of cells along the x-axis, and x represents the coordinate of cells in the x direction, ranging from 1 to N^x_{cell}. While finding the optimal mapping from 3-D cells to 1-D rings can be a complex task, we utilize the aforementioned mapping method as it significantly reduces the average packet lifetime.

Another parameter to consider is the number of rings. Although routing within the rings overlaps with force evaluation, the latency may not be entirely hidden when there are too many cells or too few particles per cell, particularly for the force output ring. To maximize latency hiding, the concurrency of the rings is adjusted according to the dataset, allowing multiple data slots to be deployed on a single network node.

Data Caching

By removing the direct connections, we merge the specific datapaths for cell-PE pairs into the narrow rings. As a consequence, the desired throughput cannot be satisfied. To tackle this problem, we develop a two-fold data caching method in order to reuse the input position data and to effectively aggregate the force fragments.

Neighbor Particle Caching: The algorithmic explanation of this method is outlined in Algorithm 3 and 5, while the hardware description can be found in Figure 4·7(a). Instead of directly retrieving neighbor particle data from neighbor caches as previously done, the neighbor data is now cached in registers located on top of the filters. This means that instead of traversing the neighbor particles for a single home cell particle (caching only one particle), we now traverse the home cell particles for multiple neighbor cell particles (caching multiple particles). This approach significantly reduces the data transfer pressure from position caches to force pipelines. For instance, if two neighbor particles are evaluated together with the home cell, it requires 50% fewer transfers compared to the previous method.

An additional advantage of the neighbor particle caching method is a more balanced workload distribution among the force computation units. Figures 4·6(a) and (b) depict two different scenarios related to the filtering rate. When traversing neighbor particles with respect to a single reference particle, situations (a) and (b) may arise, resulting in highly imbalanced filtering. However, in the current scheme, multiple neighbor particles act as reference particles, significantly reducing the chances of filtering imbalance.

To complete the design, the neighbor force data are cached in registers (at the bottom of Figure 4·7(a)). The locality of neighbor particles is taken advantage of and the neighbor forces are accumulated before being sent to the force output ring. As a result, the payload

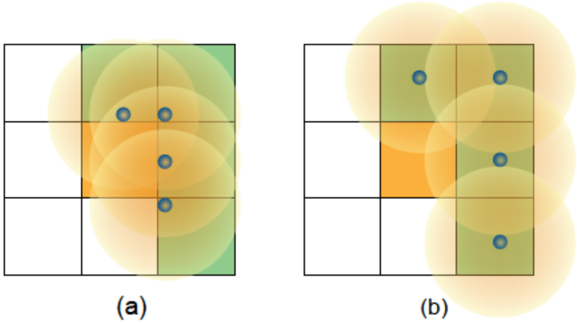


Figure 4·6: Example cases of (a) high filter pass rate, (b) low filter pass rate in 2D illustration.

on the force output ring is reduced considerably.

Home Particle Caching. Instead of storing the entire position data in the filter buffers below the filters, only the particle IDs are used (see Figure 4·7(a)) to save BRAM resources. To compensate the home cell particles are localized in the temporary home position cache so that the home particles can be accessed with particle IDs. A home particle is collected in the temporary home position cache upon arrival, and can be overwritten after the evaluation of the current home cell is finished. As a result, both home and neighbor particles are localized for arbitration and are ready to be fed into force computation units.

Merits in System Design. With data caching methods corresponding to the loop reordering in Algorithm 3 and 5, the designs in Figure 3·5 can be reduced in concurrency demand except Figure 3·5(a) for its overly fine-grained data access mode. The home cells now do not need to broadcast multiple particles simultaneously as the cached data is sufficient to keep the force pipelines busy.

Neighbor Particle Dispatching

In order to accommodate the data caching methods, several adaptive components have been implemented. A dispatch mechanism has been developed to effectively cache neighbor particles, and Figure 4·7(b) provides detailed insights.

Position data from the position input ring is injected directly into the position input

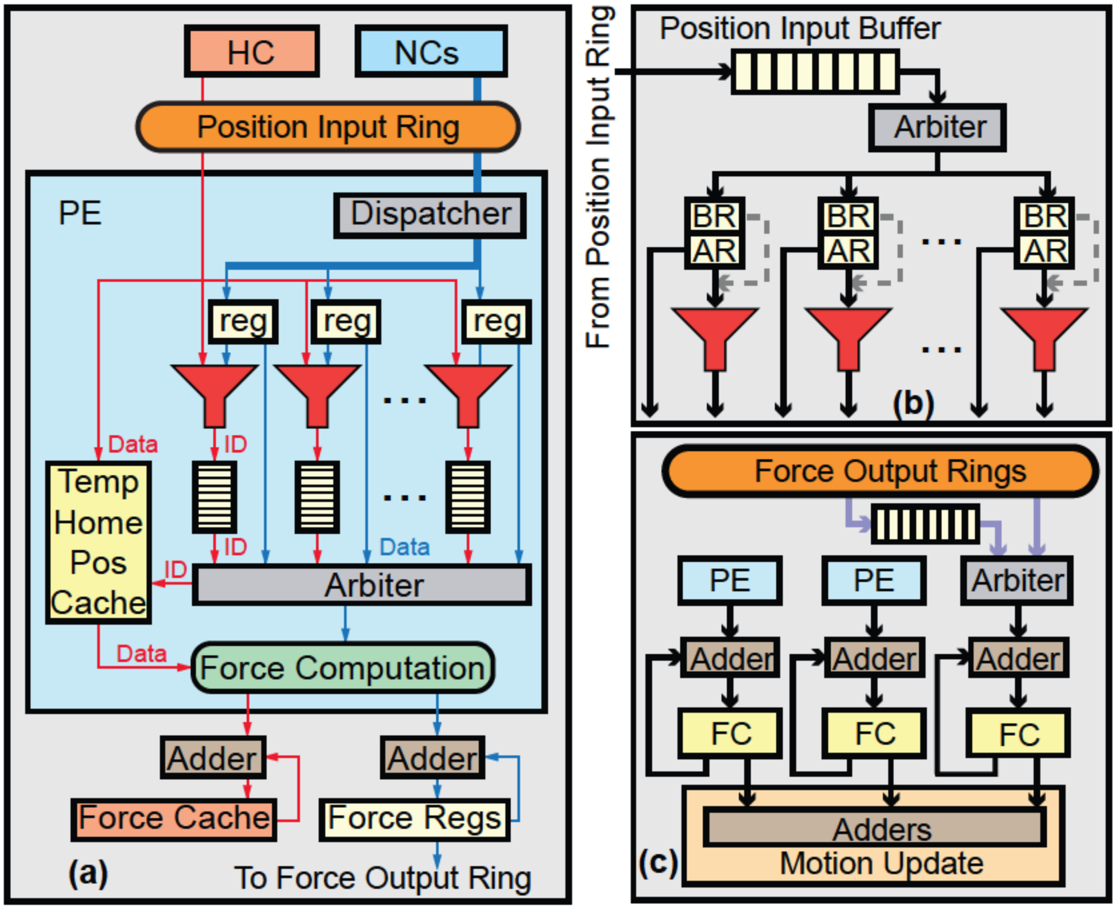


Figure 4·7: The optimized design for the third mapping scheme. (a): The data path of a single force computation pipeline. (b): Particle dispatching layout with duo registers. AR: Active Register; BR: Backup Register. (c): The partial force caches and force aggregation, where the two PEs work on a same home cell, and the purple paths are to resolve the conflict when two force output rings are used.

buffer and is ready to be dispatched. An arbiter continuously monitors the status of the registers below and fills new data into the registers once any register is completely evaluated. A neighbor particle present in an active register (AR) can be utilized for both filtering and force computation, while a particle in a backup register (BR) can only be used for filtering when the AR is being consumed in the force computation. The A-B register design aims to optimize filter utilization and ensure sufficient workload for the force computing units. Adding more filter pipelines is also feasible. Despite having less control logic, much higher memory cost is demanded in the form of additional filter buffers.

To determine if a neighbor particle has completed filtering, the home particle ID is recorded upon the arrival of the neighbor particle. The filtering process is considered complete when the same ID is observed again as the home particles are traversed repeatedly. A filter buffer can store up to two home particle IDs, or only one if only the AR is being used.

Partial Force Caches

Due to the difficulty of caching home particle forces in this design, these forces are directly accumulated into force caches without passing through a force output ring (refer to Figure 4·7(a) bottom and Figure 4·7(c)). When multiple force pipelines are operating on a single cell (as shown in system designs (b) and (d) in Figure 3·5), more than one force cache is required to match the throughput of the force pipelines.

The forces received from the rings are accumulated in a dedicated force cache, as depicted in Figure 4·7(c). If multiple force output rings are deployed to achieve higher concurrency, the forces are buffered and arbitrated in a round-robin manner before being accumulated into the same force cache. This approach works effectively because the arrival rate of forces through the rings is much lower than that of the forces from the processing elements (PEs).

The force caches, which hold partial forces, are properly aligned in memory based on particle IDs. It is not necessary to aggregate the forces until the motion update phase begins. Consequently, only a small number of adders are required to perform the aggregation of the partial forces. Since motion update is not computationally intensive, and the number of motion update units is relatively small compared to the number of force pipelines, the resource demand for the aggregation process remains manageable.

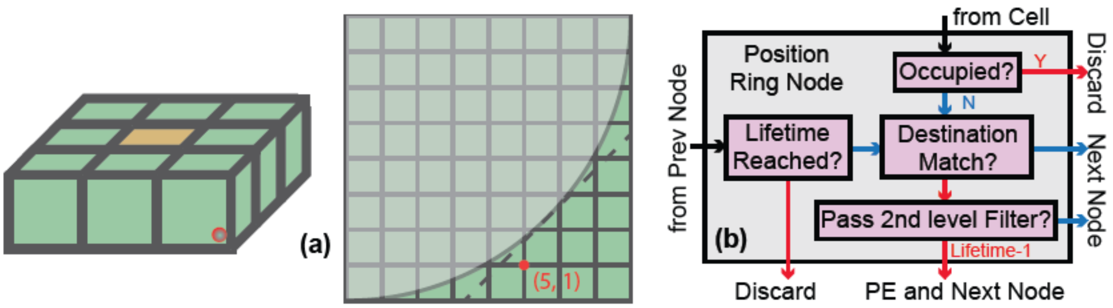


Figure 4·8: (a): The 1st level filtering in a position input ring node. Dashed line: planar filter criterion. (b): The flowchart of a position input ring node.

Position Input

The position input units upstream are also upgraded to support the force processors downstream. To be specific, the position routing and position broadcast are upgraded to enhance hardware utilization.

Hierarchical Filters. In order to avoid unnecessary processing in the processing elements (PEs), certain particles can be discarded before reaching the PEs. For instance, as depicted in Figure 4·8(a), a neighbor particle may not find any suitable particle to pair with in the home cell. In such cases, sending these particles to the PEs would result in wasteful computation cycles. To optimize the system, it is advantageous to implement a higher-level filter that eliminates these “bad” particles during data transfer.

Given that a particle can have multiple destinations in a position input ring, it is logical to incorporate second-level filters into the position input ring nodes in order to perform filtering along the data movement. A second-level filter operates similarly to a first-level filter, but it compares the particle position with the coordinates of the corner. By employing hierarchical filtering, Monte Carlo experiments have demonstrated an increase in the filtering rate of the first-level filter from 17% (using a planar filter) to 25% when dealing with uniformly distributed homogeneous particles.

Figure 4·8(b) illustrates the integration of second-level filters into the position ring nodes. Each node has the capability to receive data either from the previous node or from

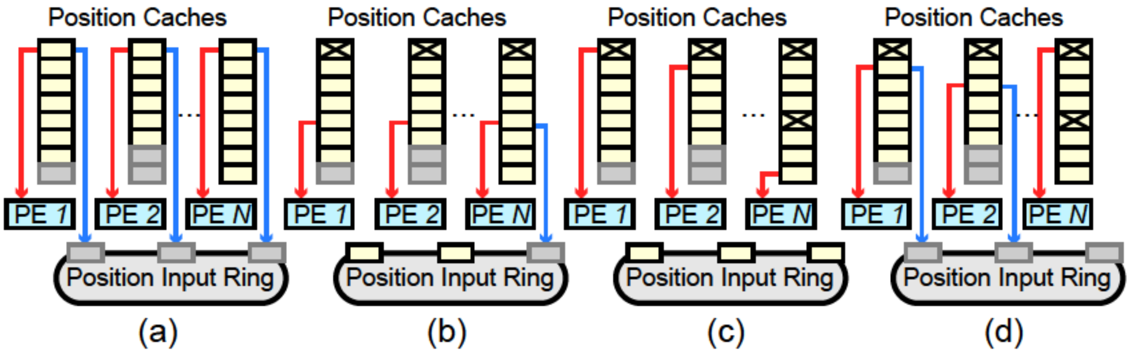
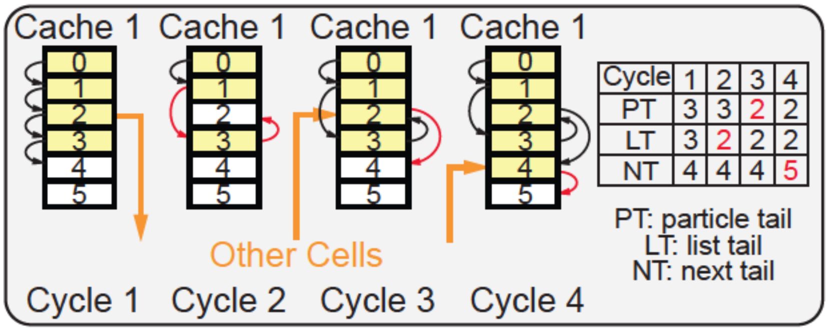


Figure 4·9: The out-of-order broadcast method. Gray boxes: Empty slots. Boxes with "x": Marked as used. (a) The 1st cycle. (b) The 5th cycle. (c) The 8th cycle. (d) the 9th cycle. Red arrows: home particle position; blue arrows: neighbor particle position.

the local position cache, with data from the previous node being given higher priority.

Out-of-Order Position Broadcast. Prioritizing the previous node is justified by the fact that data directly from the position caches can be retransmitted later, as the position caches are iterated repeatedly for particle filtering with the assistance of cached neighbor particles. However, a challenge arises because the position caches not only provide the position of the home particles but also the position of the neighbor particles. The question is how to efficiently handle both aspects simultaneously.

Figure 4·9 presents an efficient out-of-order broadcast method in four representative cycles. This method enables seamless injection of neighbor positions during the traversal of home positions, allowing the data to be transmitted downstream whenever the ring nodes are available. During the broadcast process, each position cache has two paths: one directly connected to a PE for home particles and another connected to a position input ring for neighbor particles. In the first cycle (a), the initial entries are sent to both the PEs and the ring since the ring nodes are unoccupied. In the fifth cycle (b), the data previously sent to the ring is marked as "dirty" to prevent redundant transmission. Position caches 1 and 2 cannot send data to the position input ring due to occupied slots, but position cache *N* can still transmit data as the slot is empty. In the eighth cycle (c), the home particles (indicated by red arrows) are no longer synchronized, and empty gray boxes are skipped to



Cycle	1	2	3	4
PT	3	3	2	2
LT	3	2	2	2
NT	4	4	4	5

Figure 4·10: Particle migration on a linked list with a table showing the pointer values. PT points at the last particle, LT points at the last slot of the list, and NT points at the next unused slot to be allocated.

avoid unnecessary delays. In the ninth cycle (d), the neighbor particles sent to the ring also become asynchronous. Eventually, the neighbor particles are evaluated out-of-order.

Memory Organization in Particle Migration

In addition to the challenges related to the routing rings, there is another issue concerning memory fragmentation caused by particle migration. Even though the memory is efficiently organized with double buffering, it requires twice the amount of memory due to the separation of memory contents and empty slots. Even if the two buffers are integrated in one memory block that is partitioned into two halves, twice the entries are used in the memory block. To address this memory efficiency problem, a scalable linked list implementation has been developed (refer to Figure 4·10).

Initially, the linked list is initialized as 0-1-2-3, where each number represents a slot in the memory. The particle tail (PT) points to the last particle in the list, which is particle 3. The list tail (LT) points to the last slot in the list, also 3 in this case. The next tail (NT) points to the next available slot, which is slot 4.

During the motion update phase, let us say particle 2 migrates out at the beginning, leaving empty slot 2 attached to the list tail (LT). As a result, the list becomes 0-1-3-2, and LT is updated to 2 accordingly. Subsequently, a new particle arrives, which is stored in the first empty slot in the list (obtained from PT), namely, slot 2. At the same time, PT is modified to 2. To prepare for the arrival of the next particle, slot 2 points to slot 4, which is the next slot to be allocated, as indicated by NT. When the next particle migrates in, slot 4 is filled and points to the subsequent slot to be allocated.

The method of particle read during force evaluation is not significantly affected by this linked list implementation. The particles can be traversed either by following the linked list or by using slot indices. If the hardware supports data streaming such that a throughput of one particle per cycle can be achieved with the linked list, the performance remains uncompromised. Otherwise, when traversing by slot indices, only a few empty slots are encountered since the variation in the number of particles per cell is typically small.

Supporting Larger Simulations

The discussions presented so far have been based on the assumption that the number of processing elements (PEs) is equal to or greater than the number of cells in the molecular dynamics (MD) simulation. However, what if the simulation space is large and consists of, let's say, a thousand cells? In such cases, it becomes necessary for PEs to work on multiple cells simultaneously. However, several challenges arise in this scenario.

Firstly, with more cells involved, storing the additional data on-chip becomes a challenge due to the limited resources of BRAM. Secondly, the latency in the routing rings becomes overwhelming when more ring nodes are added to support a larger number of cells. And thirdly, for each simulation step, the number of cells accessed remains greater than the number of PEs due to the absence of periodic boundary conditions (see Figure 4·11(a), the orange cells are to be processed in the next step but still need to be accessed).

As a BRAM typically has 512 entries in depth and a cell has <80 particles per cell, most BRAM entries are wasted if a BRAM only contains the particle data from a single cell. To address the issue of data storage, a solution is to partition the BRAM into multiple interleaved cell regions. Each region contains the particle data from separate cells that are to be processed in different steps (as shown in Figure 4·11(b)). This approach allows for

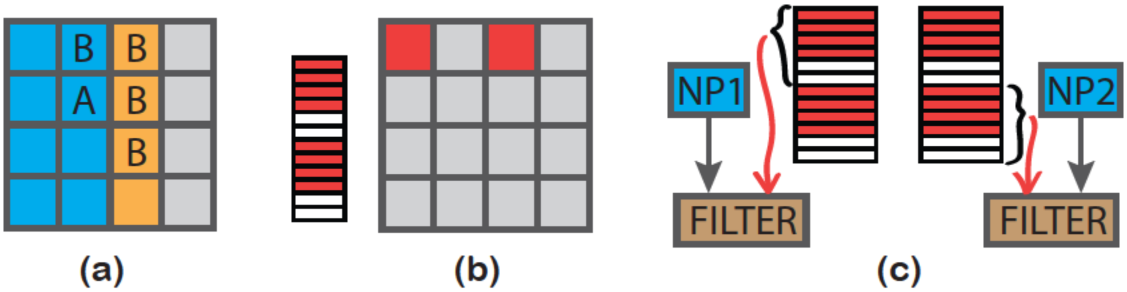


Figure 4·11: The solution to the problems emerge in large scale simulations. We intuitively use 4×4 2-D cells with eight PEs for example. (a): blue cells: the cells currently being processed; orange cells: the extra cells whose data are required to process the blue cells. In practice, particles from cell A are broadcast to all B cells through a position input ring for particle pairing. (b): the memory structure used to achieve higher memory utilization. The two red cells are sharing the same memory block at different address domains. (c): the double buffers are restored to avoid unnecessary memory traversal. Two neighbor particles (NP1 and NP2) require data from different cells.

more efficient utilization of the BRAM, as most entries would otherwise be wasted if a single cell's data occupied the entire BRAM. Additionally, this partitioning strategy helps reduce the length of the routing rings, as the interleaved cells can share the same ring node for routing.

While the storage and ring latency problems are mitigated through the interleaved cell regions, the issue of concurrency still remains. In Figure 4·11(c), two neighbor particles, NP1 and NP2, are being processed by the same PE, but with data from different cells. One approach to tackle this is to traverse all the particles in a memory block, but this results in wasted time waiting for unnecessary data to pass through. To address this, double buffering is necessary for high efficiency. Initially, during the force evaluation phase, only one buffer is used while the other remains unused and is only utilized during the motion update phase. With the increased concurrency demand as Figure 4·11(a) illustrates, the unused buffer can be duplicated from the other buffer, allowing for 2× concurrency. This enables NP1 and NP2 to be evaluated simultaneously, utilizing only the data from the desired cells.

4.3.3 General Summary of the Optimized Designs

The two designs, optimized design 1 and optimized design 2, exhibit distinct strengths and weaknesses in the context of their application.

Optimized design 1, derived from high-level mapping scheme 1, offers a straightforward approach to the molecular dynamics (MD) process with convenient scaling in space and automatic workload balancing mechanism. However, it faces two significant challenges. Firstly, the design struggles to be extended to multiple FPGA chips. In an intuitive sense, each FPGA chip is responsible for evaluating a specific region within the simulation space, and data exchange is primarily required among a few boundary cells. Nevertheless, due to the scattering of particles from a single cell among all caches, all caches on a single chip are uniformly involved in inter-FPGA communications. Consequently, this intensifies the complexity of communication and poses difficulties for multi-chip extension. Secondly, in this design, the neighbor particles need to be reloaded after the evaluation of each reference cell, leading to an overhead of approximately 20% in the overall processing time.

On the other hand, optimized design 2 offers advantages in terms of multi-chip extension. It maintains the storage of particles by cell, effectively localizing particles in space. This characteristic facilitates the distribution of workload across multiple FPGA chips. However, it requires additional resources for constructing the routing rings, which serve as the communication infrastructure. Moreover, the scalability of this design on a single chip is relatively weak compared to optimized design 1. This is primarily because optimized design 2 relies on a more sophisticated mechanism to support larger-scale simulations, which may strain the available resources on a single chip.

4.4 Evaluation

4.4.1 Experiment Setup

The implementations of the designs are conducted on a Xilinx Alveo U280 acceleration card, which offers ample resources for the computations. The card provides 1065K CLB (Configurable Logic Block) LUTs, 2134K CLB registers, 1490 block RAMs, 960 ultra RAMs, and 8490 DSPs within the dynamic region.

In the simulations, it is assumed that each cell initially contains 64 particles, which aligns with physical considerations. Additionally, all designs are equipped with eight motion update units, ensuring that the time spent on motion update is negligible compared to the force evaluation phase.

The baseline designs 1 (B1) and 2 (B2), as well as the optimized design 1 (O1), have a fixed number of processing elements (PEs). B1 and B2 employ 128 PEs, while O1 utilizes 160 PEs, with 20% of the PEs reserved to handle particle migration. On the other hand, baseline design 3 (B3) and optimized design 2 (O2) have a number of PEs equal to the number of cells in the simulation space, providing a more flexible and scalable approach.

For the purpose of performance comparison, the hierarchical filters in O2 are temporarily disabled. The impact of various configurations and settings is discussed in greater detail in Section section 4.4.3.

4.4.2 Performance and Comparison

To comprehensively analyze all five designs, we conduct evaluations both vertically and horizontally. The vertical comparison focuses on assessing the performance of each design with a variety of simulation size spaces, while the horizontal comparison evaluates the PE efficiencies of the designs within the same simulation space. The vertical comparison aims to show the scalability of the designs as the simulation space scales up. However, directly comparing the performance of the designs can be misleading due to the different scaling patterns of the PEs in each design. Therefore, to ensure a fair and universal comparison, we use PE utilization as the criterion in our horizontal comparison, which allows us to evaluate the efficiency of PEs across different designs within the same simulation space.

It is important to note that the results for design O2 in the 6×6×6 cell space are obtained through cycle-accurate simulation, as the available on-chip resources of the Alveo U280 card are not sufficient to map all 216 PEs. Results for the 6×6×6 space with 108 PEs will be presented in a subsequent section to provide a comprehensive analysis of the design's performance.

Vertical Comparison

Scaling results are given in Figure 4·12. The performances of the three designs on the left with fixed number of PEs scale linearly with the increasing space size. This shows there is no extra overhead involved as the simulation space scales up. From another perspective, O1 is suitable for evaluating a limited number of cells on a single chip, i.e., strong scaling. The performance is not down-graded with a larger number of PEs working on a small number of particles/cells.

Ideally, for the designs on the right, the number of PEs scales linearly with the number of cells, and the bars in the figure should have the same height. However, the results are clearly influenced by the increasing number of cells. In B3, as discussed above, traversing all neighbor particles for a single reference particle leads to an imbalanced filtering rate. This situation deteriorates as the number of PEs increases as the highest and the lowest filtering rates are farther apart. In O2, the poor scaling is due to the extended rings (all bars should be with same height for perfect scalability). Long rings result in long latency and congestion in the rings. The slots in the rings are occupied by the packets still in transit, preventing the new packets from entering the rings. This suggests O2 also works well with a limited number of cells. In contrast with O1, however, O2 has more potential to be extended to multi-chips (see discussion in section 4.3.3).

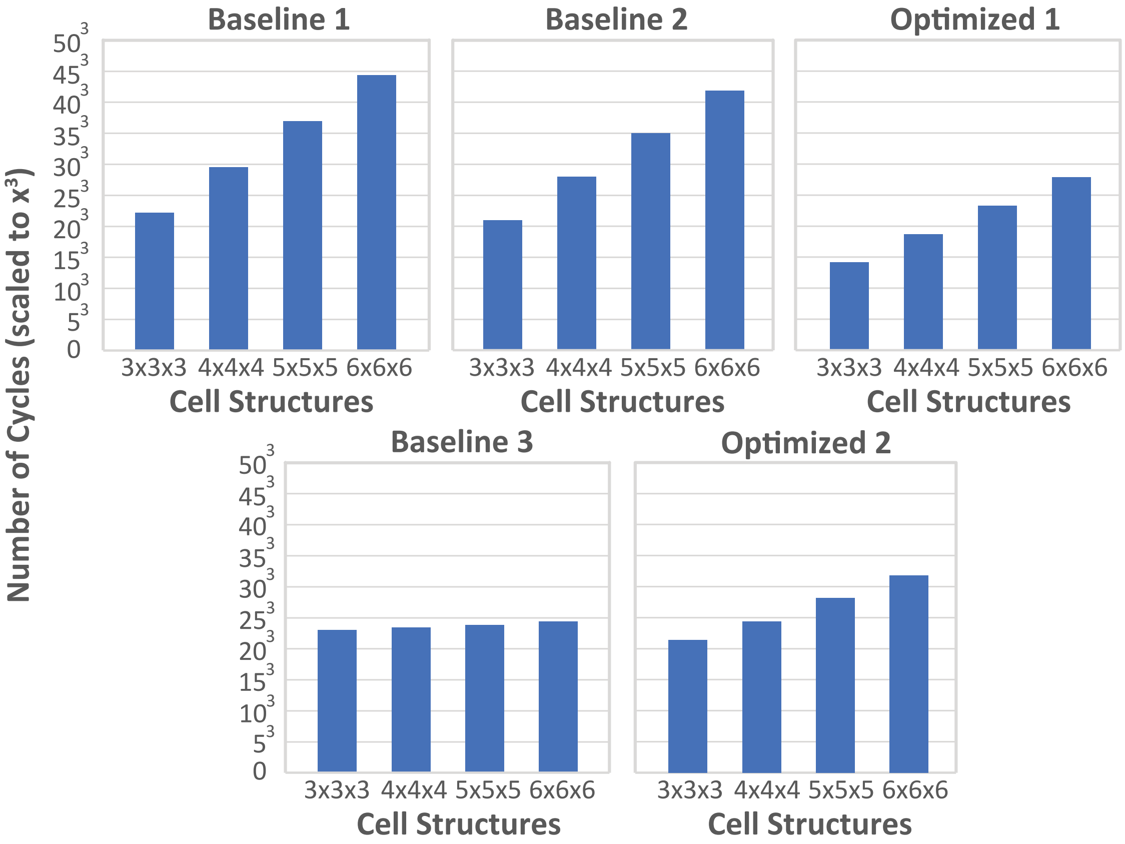


Figure 4·12: The performance in cycles per iteration of three baseline and two optimized designs. We show results for four different cubic cell structures in the simulation space to show scalability. The *y*-axis is scaled to x^3 for a better illustration of linearity. The dashed line is to split the designs with substantial scalability differences.

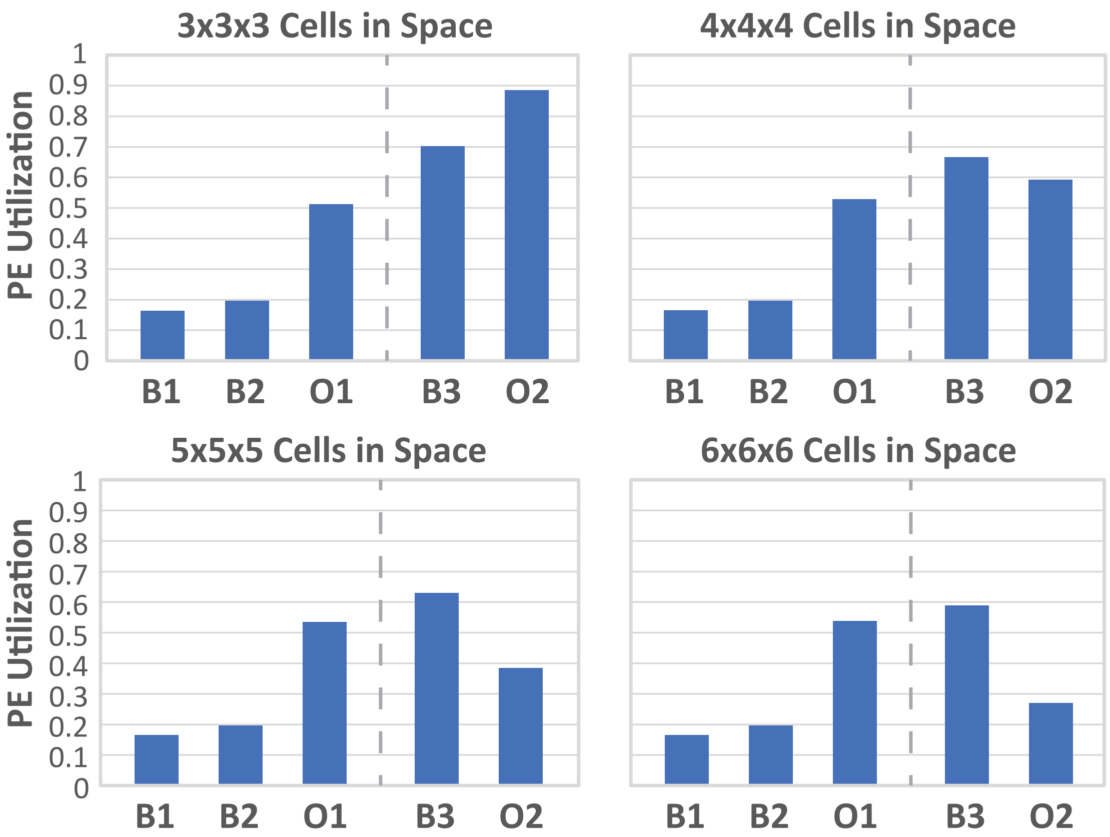


Figure 4·13: The PE utilizations of three baseline and two optimized designs. The results for four different cubic cell structures in the simulation space are provided. B1∼B3: baseline designs; O1 and O2: optimized designs. The dashed lines are used to split the designs with substantial scalability differences.

Horizontal Comparison

We define PE utilization as the average number of valid force pairs evaluated per PE divided by the overall working cycles in an MD iteration. This metric provides insights into the hardware efficiency of the designs since the majority of hardware resources are dedicated to force computation.

The results, illustrated in Figure 4·13, reveal several observations. Firstly, B1 and B2 exhibit significantly lower PE utilization, ranging from 15% to 20%. This can be attributed to the global synchronization mechanism implemented to prevent forces from different reference particles from entering the adder trees. Additionally, B1 suffers from the overhead introduced by the neighbor particle pre-loading process.

In contrast, O1 demonstrates a considerable improvement in utilization, surpassing the baselines by over 30%. By employing buffers above the adder tree, O1 only allows forces from the same reference particle to enter the adder tree, eliminating the need for global synchronization. However, the presence of reserved PEs and the overhead of neighbor particle pre-loading still impose limitations on PE utilization.

For the $3 \times 3 \times 3$ cell space, both B3 and O2 achieve higher PE utilizations. These designs benefit from fully streaming particles without the need for pre-loading. However, B3 experiences a lower PE utilization compared to O2 due to the overhead caused by global synchronization in a $3 \times 3 \times 3$ cell space. On the other hand, O2 encounters a drop in utilization for larger simulation spaces due to the elongated rings, particularly the force output ring, which experiences heavy congestion.

It is worth noting that all baseline designs face challenges with complex data routing paths, as depicted and discussed in Figure 4·1(d). This results in unacceptable frequency degradation. In contrast, the optimized designs generally achieve frequencies exceeding 200 MHz, while the baselines can only operate at 70 MHz or lower for simulation spaces larger than $4 \times 4 \times 4$ cells. Thus, both optimized designs represent significant improvements over the baselines from a frequency perspective as well.

4.4.3 Evaluating Design Options

In this Section, several further optimizations of Design O2 are evaluated, First, the hierarchical filtering mechanism is evaluated. The hierarchical filtering enhances the filtering rate of planar filters from 17% to 25% without introducing redundant particle pairs. Second, the number of rings can be configured to compensate for the limited throughput of rings.

Hierarchical Filtering

The hierarchical filtering mechanism not only improves the pass rate of filters in a PE but also eliminates the need for PEs to compute zero forces. In the absence of hierarchical filtering, every force register in Figure 4·7(a) must undergo zero checking before injecting forces into a force output ring. Without this checking, the load on the force output rings would increase due to unnecessary transfers. Although only eight exponent bits per dimension per register (24 bits in 3-D) need to be checked, the hardware cost can be significant considering the potentially massive number of force registers in Figure 4·7(a). In our evaluation, we compare the original O2 design without zero force checking to the modified O2 design with hierarchical filters.

Figure 4·14 illustrates the performance of the two O2 configurations. In the case of a 3×3×3 cell space, the hierarchical version reaches equilibrium at four filters, whereas the original version reaches equilibrium at six filters. For larger cell spaces, they both reach equilibrium at a similar number of filters but with varying performance. This is because the performance bottleneck lies in the force output and/or position input. The rings reach their maximum capacity, and adding more filters does not improve their performance. As discussed earlier, the hierarchical version reduces force output traffic without the need for zero checking, resulting in better performance, especially in heavily congested force output rings.

Number of Rings

Given that the rings become the bottleneck with larger cell spaces, higher ring bandwidths are expected. We intuitively add more rings to the system to tackle the problem. We inspect the number of force rings from one to four with up to two position input rings. The results are given in Figure 4·15.

The number of rings are denoted as [number of position input rings, number of force

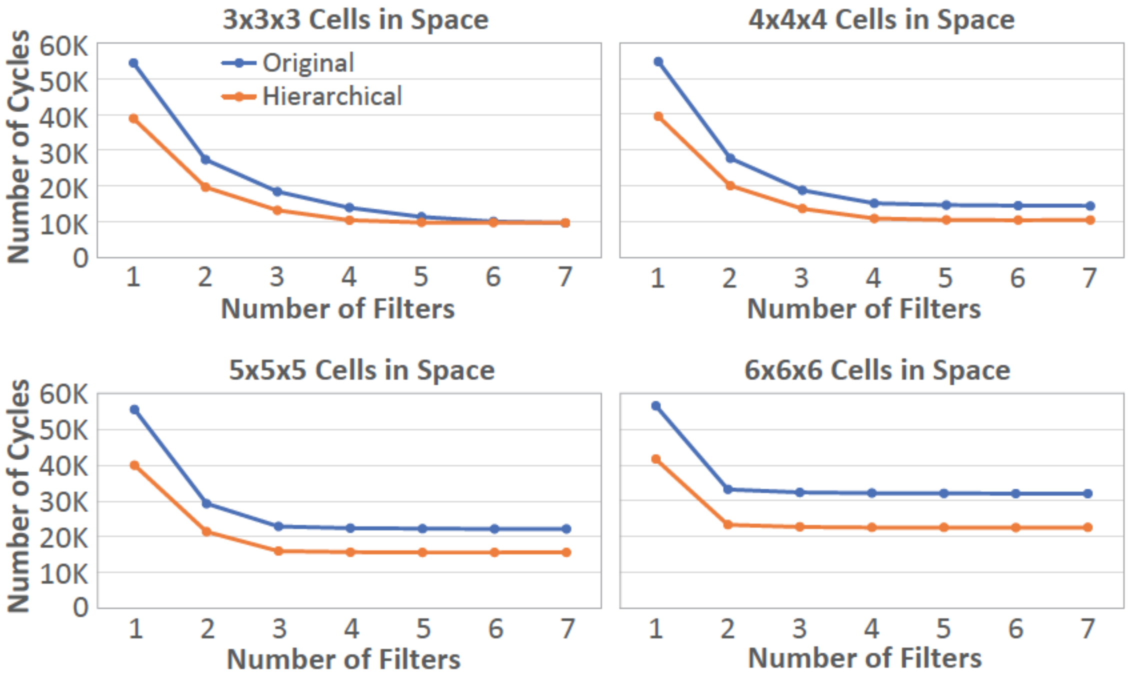


Figure 4·14: The performance comparison of the original O2 and O2 with hierarchical filters for four representative cell spaces.

output rings]. We observe in the figure that the number of cycles saturate at [1, 1], [1, 2], [2, 3] and [2, 4] rings for the four example cell spaces in the case of original O2, and [1, 1], [1, 2], [2, 2], and [2, 3] rings in terms of O2 with the collaboration of hierarchical filters. This indicates that with hierarchical filters, the number of rings required is efficiently reduced. With both configurations applied, the utilization of PEs is greater than 75%.

4.4.4 Benchmarking

In this study, we utilize a state-of-the-art OpenMM liquid argon benchmark as the reference system since we do not model the bonded force or long-range force. The reference systems consist of an Intel Xeon Gold 6226R CPU with 32 threads and a Quadro RTX 8000 GPU. The results are presented in two sections, each focusing on different levels of simulation space sizes, with the FPGA-O2 design serving as the reference. The conventional performance unit used is "ns/day" (for a certain dataset, how many nanoseconds can be simulated

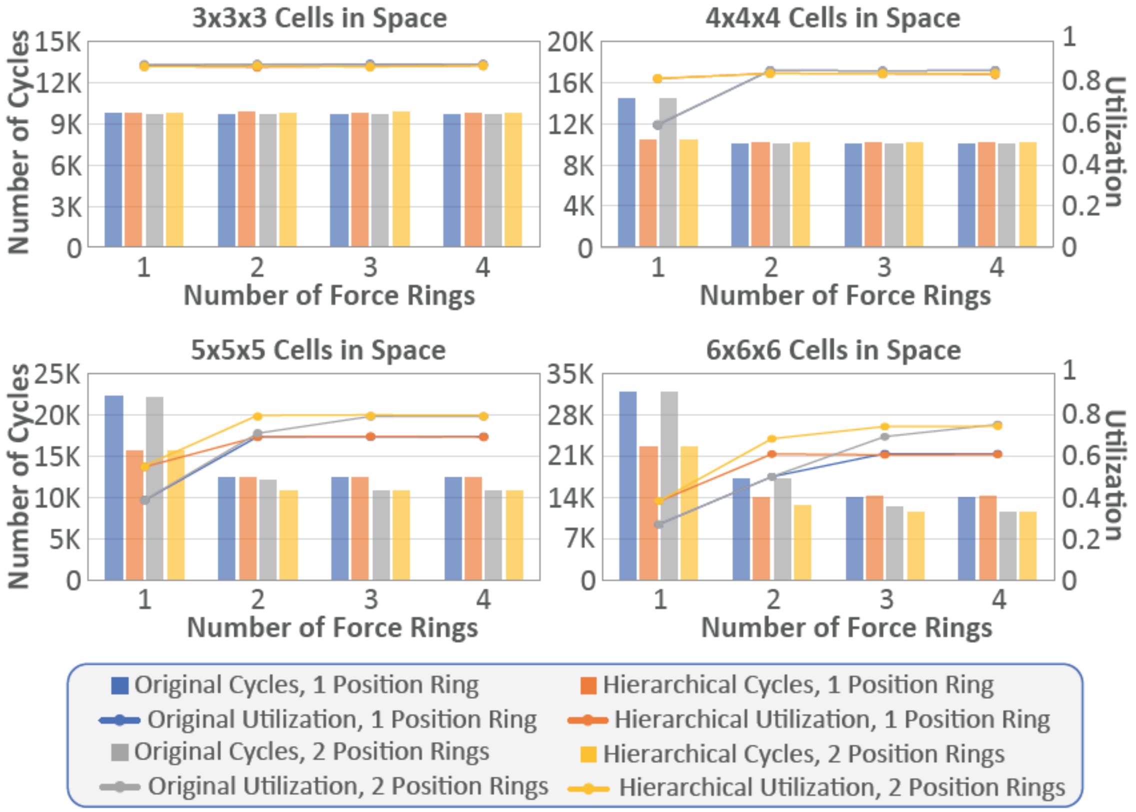


Figure 4·15: The performance of the original and hierarchical O2 for four representative cell spaces with different numbers of rings equipped.

Table 4.1: FPGA-O2 performance compared to FPGA-O1, GPU, and CPU with up to 32 threads.

Cell #	3×3×3		4×4×4		5×5×5	
Particle #	1728		4096		8000	
Performance	ns/day	speedup	ns/day	speedup	ns/day	speedup
FPGA-O1	12222	0.29	5303	0.64	2750	1.16
FPGA-O2	3540	1.00	3411	1.00	3195	1.00
GPU	2921	1.21	2126	1.60	1841	1.74
CPU-1×	80.1	44.2	35.3	96.8	19.4	165
CPU-2×	136	26.1	60.6	56.3	32.8	97.4
CPU-4×	205	17.3	92.8	36.8	60.1	53.2
CPU-8×	236	15.0	111	30.8	88.7	36.0
CPU-16×	183	19.3	95.3	35.8	78.0	40.9
CPU-32×	153	23.2	102	33.6	77.5	41.2

Table 4.2: FPGA-O2 performance of large simulation spaces compared to FPGA-O1, GPU, and 32-thread CPU.

Cell #	6×6×6		8×8×8		12×8×8	
Particle #	13824		32768		49152	
Performance	ns/day	speedup	ns/day	speedup	ns/day	speedup
FPGA-O1	1602	1.87/1.02[1]	680.4	1.13	455	1.12
FPGA-O2	2997/1637	1.00	765.9	1.00	511	1.00
GPU	1542	1.94/1.06	801.8	0.96	619	0.83
CPU-1×	11.1	271/148	5.26	146	3.49	146
CPU-2×	18.7	160/87.5	8.82	86.8	5.78	88.3
CPU-4×	31.4	95.4/52.1	16.0	48.0	10.6	48.4
CPU-8×	49.2	60.9/33.3	20.9	28.4	17.6	29.0
CPU-16×	53.1	56.4/30.8	28.5	26.8	21.5	23.7
CPU-32×	52.7	56.8/31.0	33.2	23.1	27.1	18.9

[1] 216 PEs (estimated) / 108 PEs, where the former assumes sufficient resources are provided on an FPGA chip.

within a day), and all FPGA designs are run at a frequency of 200 MHz. FPGA-O2 is designed with a sufficient number of rings to ensure that the bottleneck lies elsewhere, and the number of rings is provided in the subsequent section.

Table 4.1 displays the results for the first set of simulation spaces. For small cell spaces, such as 3×3×3 and 4×4×4, FPGA-O1 exhibits significant speedup compared to the other designs due to its straightforward scaling capability, utilizing all available PEs in a small cell space. It achieves a speedup of 4.18× compared to the GPU on a 3×3×3 cell space. On the other hand, FPGA-O2 demonstrates consistent performance scaling with increasing cell space size and delivers substantial speedup compared to the GPU, with improvements of up to 74%.

The second set of results is presented in Table 4.2. It includes the 216-PE version and the 108-PE version for a 6×6×6 cell space. The former demonstrates the hardware's scalability, while the latter showcases the capability of handling larger simulation spaces. The on-chip resources enable us to simulate approximately 50,000 particles in total, with performances comparable to those of the GPU.

4.4.5 Hardware Utilization

In order to investigate the suitability of different configurations of O2, Table 7.1 presents the utilization of hardware resources for the performance evaluation of O2 in the representative cell spaces evaluated above. The number of position rings and force rings are carefully optimized to ensure that the performance is not compromised by congestion in the rings. To accommodate the limited number of BRAMs, some of the buffers are implemented using logic as LUTRAMs.

Note that the resource utilizations for the 8×8×8 and 8×8×12 cell spaces are nearly identical. This is because the cache is not fully utilized in terms of depth, allowing it to store data from multiple cells without incurring additional memory cost. Moreover, since the number of PEs remains unchanged, only a negligible amount of additional resources is required to adapt to the larger space size.

Table 4.3: Hardware Utilization of FPGA-O2 with spatial configurations.

Cell Space	3×3×3	4×4×4	5×5×5	6×6×6	8×8×8	8×8×12
Particle #	1728	4096	8000	13824	32768	49152
PE #	27	64	125	108	128	128
Position Ring #	1	1	2	1	2	2
Force Ring #	1	2	2	2	3	3
LUT	16%	38%	73%	65%	85%	85%
FF	9%	22%	41%	38%	51%	51%
BRAM	13%	33%	63%	55%	74%	74%
URAM	17%	40%	78%	68%	80%	80%
DSP	16%	39%	74%	64%	84%	84%

4.5 Related Work

Accelerating MD with FPGAs has been studied for many years [Azizi et al., 2004, Gu et al., 2005b, Hamada and Nakasato, 2005, Scrofano and Prasanna, 2006, Gu et al., 2006a, Kindratenko and Pointer, 2006, Alam et al., 2007, Schaffner and Benini, 2018, Jones et al., 2022]. As resources per chip have multiplied, so has the number of compute units to the point where thousands of elements need to be sourced and sinked every cycle. MD is fundamentally a data movement problem: even with standard optimizations (cell lists and Newton's third law) data are necessarily used and updated simultaneously by many compute units. With the previous routing solutions no longer viable, maintaining high efficiency has required several design innovations. On ASIC implementations, although both Anton [Young et al., 2009, Shaw et al., 2007] and MDGRAPE-4 [Ohmura et al., 2014] adopted the ring-based mechanism, we had no a priori reason to believe that this mechanism would be applicable to FPGAs.

4.6 Conclusion

In this study of FPGA-based range-limited MD work, we present multiple 3D-to-2D workload mapping schemes: three baseline designs based on distinct, and, at a high level, exhaustive mappings, and two optimized designs. The two optimized designs have scalability in two directions. The first design scales easily with the size of a simulation space with a constant number of PEs, while the second allows the number of PEs scale with the size of a simulation space to obtain sustained performances. We also analyze several configurations of the 2nd optimized design, and the effectiveness of the configurations is highlighted.

The overall motivation for using FPGAs for MD computation is that FPGA-based clusters fill a vital niche: scalable COTS-based systems that provide long timescales on problem sizes crucial to both drug discovery and basic science. The necessary condition is for the per-device performance to be comparable to that of the accelerator alternatives. That has been demonstrated here in multiple applicable scenarios. For example, optimized design 2 achieves a 1.74× speed-up for 8000 particles, and optimized design 1 achieves 4.18× for 1728 particles. While these simulations are a small fraction of tens of thousands typical in production runs, they are exactly the workload per device when these runs are partitioned over 8 or 32 devices, respectively.

Chapter 5

Bandwidth Analysis of FPGAs on a 3-D Torus

5.1 Introduction

The scalability challenge in Molecular Dynamics (MD) simulations is well-known, particularly concerning the global communication involved during the long-range force computation (LR). This global communication poses a strong scaling problem in classical MD, where increasing the number of processors to reduce iteration time ultimately leads to increased communication time dominating the overall performance. ASIC- and FPGA-centric clusters have proven effective in addressing this issue by employing dedicated chip-to-chip communication that can transfer data quickly (in less than 100 nanoseconds) and at high bandwidth, directly at the application-layer level with minimal additional overhead [Young et al., 2009, George et al., 2016, Sheng et al., 2015, Sheng et al., 2017b].

While the mapping of LR onto FPGA-based clusters has been extensively studied, the mapping of RL, which accounts for 90% of the FLOPs in MD and involves a larger amount of data transfer [Gu et al., 2006b, Gu et al., 2008, Khan and Herbordt, 2012, Herbordt, 2013], has received less attention. The local nature of RL communication may have initially made the problem seem straightforward, particularly for small clusters [Grossman et al.,]. However, our research has revealed that the reality is quite the opposite.

The transition from a single-chip RL design to a multi-chip design introduces several scalability challenges. Despite the considerable transceiver bandwidth available on commercial FPGA boards, we have found it to be insufficient due to poor data locality in typical MD layouts. If we were to directly inherit the data transmission design from the single-chip version, achieving significant scalability would require transceiver bandwidth on the order of terabits per second (Tbps), or else most of the scalability potential would be lost. To overcome this, we propose a neighbor data caching method that leverages data locality for improved data reuse, resulting in negligible hardware overhead.

Another challenge arises from the irregularity of the inter-board communication pattern, which leads to imbalanced loads on different transceiver ports. To optimize bandwidth utilization, we exploit the underlying MD-RL particle interaction topology. Specifically, particles only interact with other particles within a halo, and the half-shell method is employed to leverage Newton's 3rd law [Shaw, 2005, Chiu and Herbordt, 2009, Chiu and Herbordt, 2010a, Chiu and Herbordt, 2010b, Chiu et al., 2011]. Considering this specific topology, we investigate various workload distributions (slab, pillar, and block) and their corresponding data transfer patterns. To address the communication imbalance, we propose a routing configuration technique that improves inter-chip bandwidth utilization for all three workload distributions.

The major contributions of this work are as follows. We propose

- a communication-efficient multi-chip MD-RL design;
- a neighbor data caching technique that improves neighbor particle data reuse for reduced data movements for inter-node communication; and
- a routing configuration technique for improved inter-chip bandwidth utilization.
- Also, experimental results show the performance of the proposed design provides at least 10× speedup comparing with the baseline design.

In the following sections, we first briefly recap on the architecture of a prior art single-chip design [Yang et al., 2019b] based on the uniform-spread mapping scheme discussed in Section 3.2.3. Second, we propose a multi-chip direct extension of this design which we refer to as the *baseline*; in Section 5.3, we compare the baseline with our optimized multi-chip designs. Finally, we present the communication bottleneck of the baseline design and discuss the causes and solutions for it.

5.2 Multi-chip baseline design

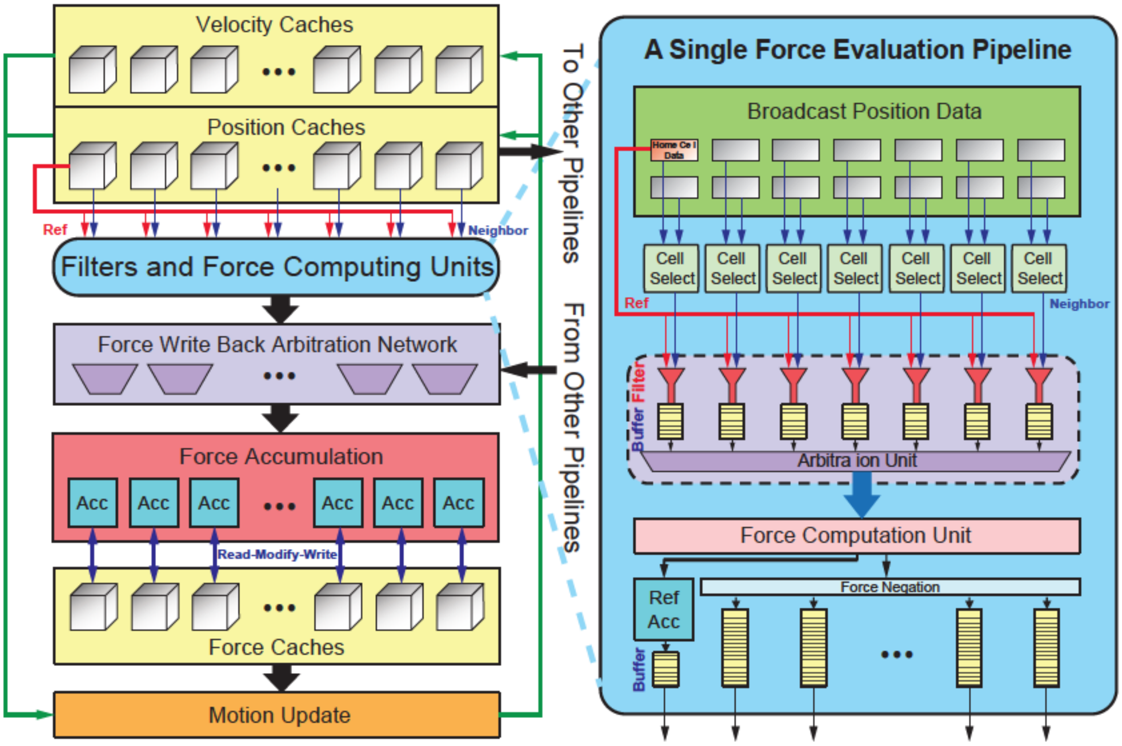
5.2.1 Prior-art Single-chip Design

The prior art single-chip MD architecture is illustrated in Figure 5·1. The particles in a simulation space are partitioned into *N* cells. The force, position, and velocity information of the particles in each cell are stored in 3 caches (force, position, and velocity), respectively. Force Computation Units (FCU) are in charge of force calculation. Each FCU evaluates the force applied on the particles in a unique cell. To evaluate forces, the filters above FCUs first read position data from position caches and then identify valid particle pairs. Subsequently, the partial forces of particle pairs are computed and sent to accumulation units through an arbitration network and so complete the force computation. Newly accumulated force results update the force caches until all the complete forces are calculated.

5.2.2 Baseline Multi-chip Design

We enhance the single-chip design mentioned earlier to support multi-chip configurations with necessary modifications for inter-chip data exchange. In the baseline design, both position data forwarding and force write back involve inter-board communication. In the case of position data forwarding, the broadcast of neighbor particle position data can potentially occur across FPGA boards. Similarly, the destination cells for force write back can be located on other FPGAs within the system.

Inter-board transactions for force data can be frequent, resulting in a high data production rate compared to the low data consumption rate. In the baseline design, it is common to send data less frequently when the available bandwidth is insufficient, leading to increased latency and performance degradation. To mitigate the impact of long communication la-



(a) MD-RL Overview

(b) Force Pipeline Architecture

Figure 5·1: Architecture of the single-chip design. (a): The general architecture of the iterative force evaluation and motion update. (b): Architecture of force evaluation pipelines

tency, a scheduling mechanism is employed in the baseline design. The evaluated partial forces, computed from individual particle pairs, are stored in an additional local buffer to facilitate the scheduling of inter-board transmission (as depicted by the scheduling buffer in Figure 5·2).

Furthermore, in the single-chip design, the partial forces are accumulated only after being selected by the arbitration network, which introduces high latency due to the select signal from the remote arbitration network. To address this issue, modifications are made in the multi-chip design, as illustrated in Figure 5·2. Instead of storing the returning neighbor forces in local output buffers, the buffers are located in the destination nodes. This modification eliminates the need to transmit the force write back arbitration result between nodes, resulting in a savings of approximately 100 ns level latency per transaction.

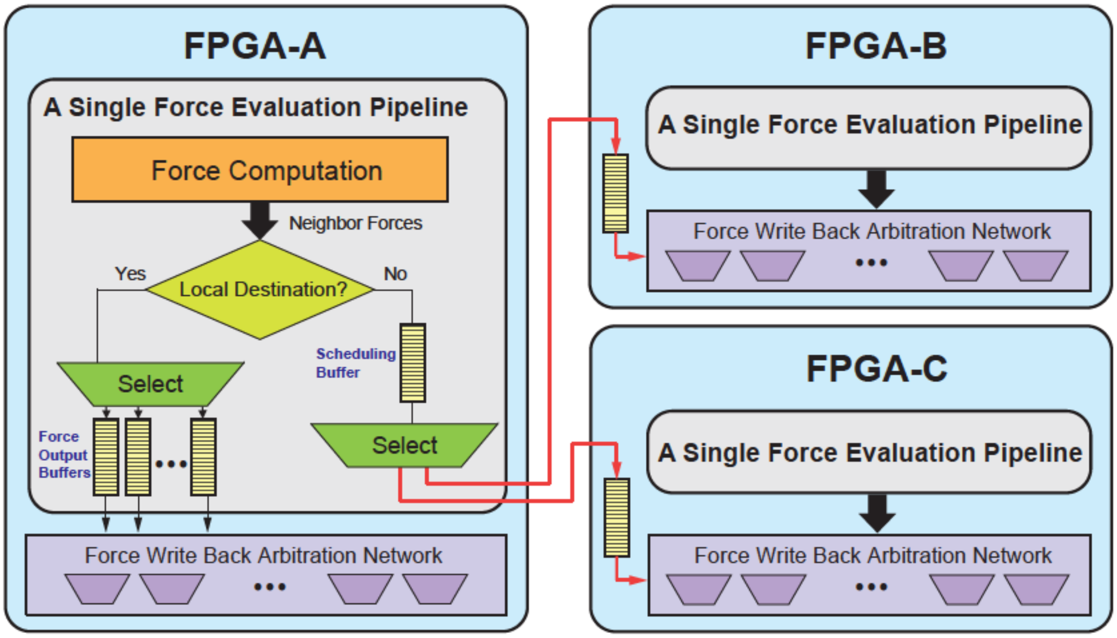

FPGA-A
A Single Force Evaluation Pipeline
Force Computation
Neighbor Forces
Local Destination?
Yes
No
Scheduling Buffer
Select
Select
Force Output Buffers
Force Write Back Arbitration Network
FPGA-B
A Single Force Evaluation Pipeline
Force Write Back Arbitration Network
FPGA-C
A Single Force Evaluation Pipeline
Force Write Back Arbitration Network

Figure 5·2: Force output buffers are re-located to prevent high latency caused by data dependency. The scheduling buffer is added in case the bandwidth bottleneck is met.

5.2.3 Communication Bottlenecks of Baseline

We observe that in the existing system, neighbor particle position data is discarded after being used by a single reference particle. As a result, these data need to be repeatedly transmitted from neighboring cells to their respective home cells. This low data reuse pattern creates a significant demand for bandwidth. Even at an operating frequency of only 100 MHz, the transmission of data between two cells consumes approximately 10 Gbps of bandwidth. Considering that there are hundreds cell-to-cell communications between nodes per MD iteration, the overall bandwidth requirement sums up to Tbps level, which exceeds the current hundred-Gbps level data transfer rate with FPGA transceivers.

Furthermore, it is worth noting that not all ports on FPGA boards are fully utilized, leading to imbalanced port utilization. This imbalance can result in significant performance degradation. For example, a node with 6 ports is responsible for managing 8×6×4 cells, where each port is connected to a neighboring node along the x, y, and z directions. Assuming that the port with the highest bandwidth demand is fully utilized (represented as U_z = 100%), it is common to observe a scenario where U_x = 78% and U_y = 59%. This leads to an overall port utilization of 80%, indicating that 20% of the data transmission opportunity is lost due to imperfect utilization. It is important to address this issue to achieve a more balanced workload distribution among different directions, especially considering that the situation may worsen with a different number of ports on the board.

5.3 Optimized Multi-chip Design

In this section, we present an optimized multi-chip design that effectively addresses the communication bottleneck observed in the baseline architecture. Our approach focuses on alleviating two key challenges identified in Section 5.2.3: the data reuse problem and the load imbalance problem.

To tackle the data reuse issue, we propose the utilization of remote neighbor data caching. This technique enables the caching of neighbor particle position data at the receiving end, allowing for efficient reuse of the data without the need for repeated transmission from neighboring cells. By leveraging this caching mechanism, we significantly reduce the demand for bandwidth and mitigate the strain on inter-chip communication.

Furthermore, we address the load imbalance problem through routing configuration. By carefully configuring the routing paths and distributing the workload more evenly among different directions, we ensure a more balanced utilization of ports on the FPGA boards. This approach maximizes the available data transmission opportunities and minimizes the performance loss caused by imbalanced port utilization.

Additionally, we briefly discuss the motion update process, which also involves inter-chip data exchange. By considering the requirements and challenges associated with motion update, we ensure that the optimized multi-chip design effectively handles the necessary data exchanges while maintaining overall system performance and efficiency.

5.3.1 Neighbor Data Caching

In the baseline design, the continuous broadcasting of neighbor particles and frequent force write-back operations impose significant demands on inter-chip data transmission. While a high-end FPGA board can support data rates of up to several hundreds of Gbps, our baseline design requires a much higher rate of Tbps.

To address this challenge, we propose a solution that involves trading the temporal locality of reference particles for the temporal locality of neighbor particles. By prioritizing the efficient transmission of neighbor particle data and optimizing the force write-back process, we aim to reduce the overall bandwidth requirements and alleviate the strain on inter-chip communication. This trade-off allows us to better utilize the available data transmission capacity and meet the demanding data transfer needs of our design.

Neighbor Particle Position Caching

For the data reusability optimization, instead of traversing neighbor particles for reference particles, reference particles are traversed for incoming neighbor particles, as Figure 5·3(b) illustrates, where the neighbor particles are cached in registers on destination nodes. The position reading architectures are compared in Figures 5·3(a) and (b). In the new design, neighbor position data only need to be transmitted once a few cycles due to the enhanced locality.

Neighbor Particle Force Caching

In the baseline design, the accumulation of partial forces applied to a reference particle is performed prior to the read-modify-write process at the force caches. However, in the modified design, where the reference particle changes every cycle, it is no longer feasible to accumulate the reference forces beforehand. As a result, the accumulator on the red path in Figure 5·3(c) is removed in Figure 5·3(d). Instead, the reference force fragments

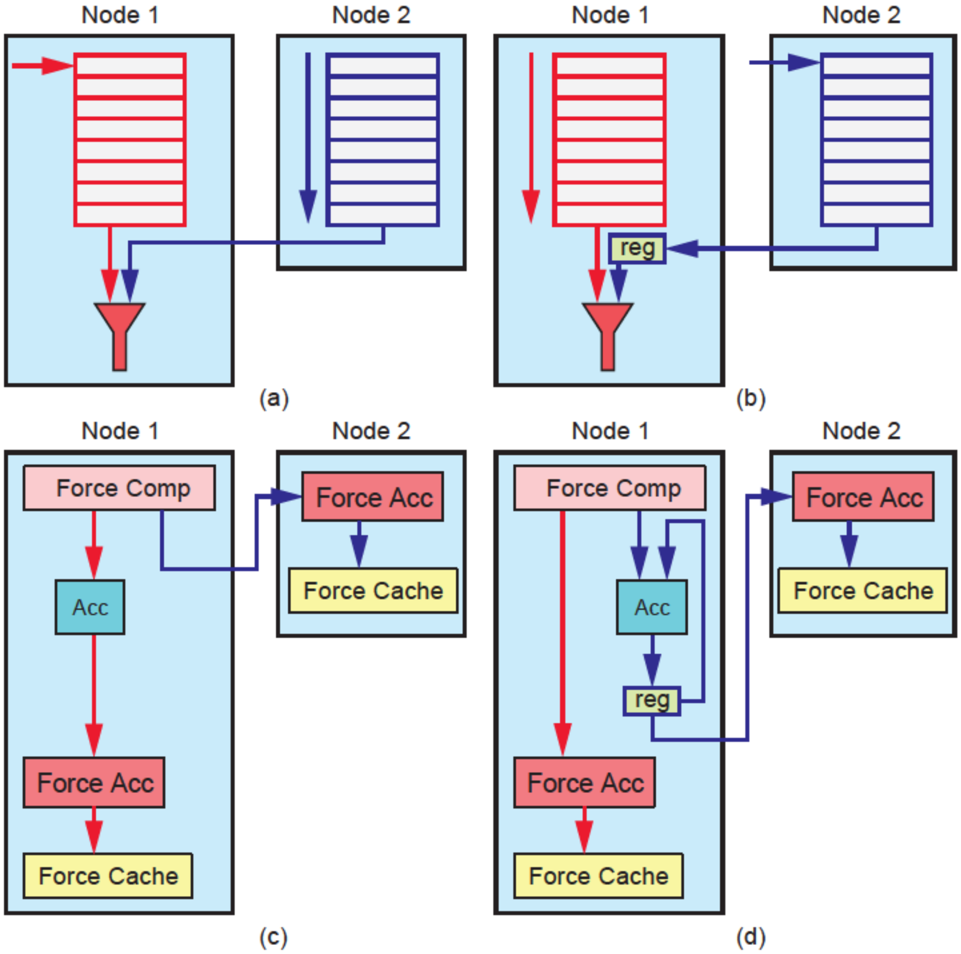


Figure 5·3: Position reading and force writing architecture (only 2 position caches in 2 nodes are shown for demonstration). (a): Baseline position reading. (b): New position reading. (c): Baseline force writing. (d): New force writing. Red arrows: Reference data paths. Blue arrows: Neighbor data paths.

are directly accumulated at the destination force caches without any conflicts, ensuring a consistent throughput.

On the other hand, although the temporal locality of the reference particles is lost, it is gained by the neighbor forces. As illustrated in Figure 5·3(d), the partial forces for neighbor particles are registered and accumulated in place. Since only one neighbor force is generated per cycle in a Force Computation Unit (FCU), all neighbor forces can share the same accumulator. The accumulated result is stored back in registers. Once all the neighbor partial forces are accumulated, they are sent back to their original nodes and accumulated in the corresponding force caches. This approach allows for efficient accumulation of neighbor forces while maximizing data reuse and minimizing conflicts.

5.3.2 Routing Configuration

Different data and workload distributions exhibit distinct data transfer patterns. In this context, we explore three fundamental scenarios: slab, pillar, and block. Each of these distributions has its own characteristic data transfer pattern, and these patterns can be independently adjusted to different extents. However, such modifications can potentially result in workload imbalances.

To overcome this issue, we propose a software-based approach that automatically configures routing paths based on the distribution of cells and the number of available ports. By analyzing the cell distribution and considering the port configuration, this approach aims to mitigate workload imbalances and ensure efficient data transfer across the system.

Workload Distribution

The workload is spatially distributed so that device memory only contains data from a group of cells in Euclidean space. In this section, three typical distribution patterns are studied and the theoretical total bandwidth usage is given.

Figure 5·4 illustrates slab, pillar and block distributions. For small simulation spaces, slab and pillar are the first picks because of their advantage in local data access with fewer neighbors, whereas the block holds the highest generality. If a slab of cells is assigned to a node, the home FPGA only communicates with two FPGAs next to it. For the pillar distributions, cells on the sides require or receive data that arrives after up to two hops. For block distributions, corner cell communications are involved and requires data to be transmitted three hops away.

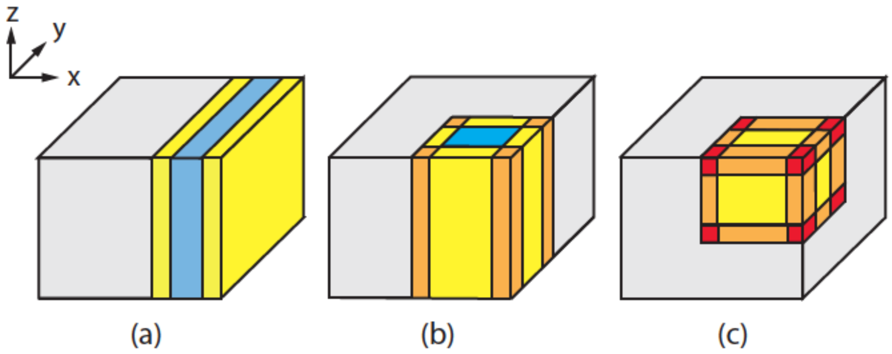


Figure 5.4: Workload distribution patterns in a cuboid simulation space. Grey: The cells evaluated on other nodes. Blue: Cells in which particle pairs are formed without external memory involved. Yellow: Cells in which data from a neighboring FPGA are needed or delivered (1 hop). Orange: 2 hops. Red: 3 hops. (a): Slab distribution. (b): Pillar. (c): Block.

Table 5.1: Bandwidth Demand - Pillar (Unit: $w_d f$)

1-hop		2-hop	
Direction	Bandwidth	Direction	Bandwidth
x_+	$l_y l_z$	$x_+ y_+$	l_z
y_+	$l_x l_z$	$x_+ y_-$	l_z
y_-	$l_x l_z$		

For the three distributions the total bandwidth is

$$B_{slab} = 2 l_y^s l_z^s w_d f \tag{5.1}$$

$$B_{pillar} = 2(2 l_x^p l_z^p + l_y^p l_z^p + 2 l_z^p) w_d f \tag{5.2}$$

$$B_{block} = 2(2 l_x^b l_y^b + 2 l_x^b l_z^b + l_y^b l_z^b + 4 l_x^b + 2 l_y^b + 2 l_z^b + 2) w_d f \tag{5.3}$$

where l_x, l_y, and l_z are the numbers of cells located on a node along x, y and z directions, w_d is the width of a piece of data to be sent from a cell to another, and f is the data transmission frequency of a position cache or a force cache. The factor of 2 signifies that this includes both position send and force receive. Here we assume the amount of position data is the same as force due to their symmetry.

For example, let 256 cells be mapped to a node. Then a slab is configured as 1×16×16, a pillar as 4×16×4, and a block as 4×8×8; the pillar prevails in total bandwidth. It can

Table 5.2: Bandwidth Demand - Block (Unit: $w_d f$)

1-hop		2-hop		3-hop	
Direction	Bandwidth	Direction	Bandwidth	Direction	Bandwidth
x_+	$l_y l_z$	$x_+ y_\pm$	l_z	$x_+ y_+ z_+$	1
$y_\pm$	$l_x l_z$	$x_+ z_\pm$	l_y	$x_+ y_+ z_-$	1
$z_\pm$	$l_x l_y$	$y_+ z_\pm$	l_x	$x_+ y_- z_+$	1
		$y_- z_\pm$	$l_x - 1$	$x_+ y_- z_-$	1

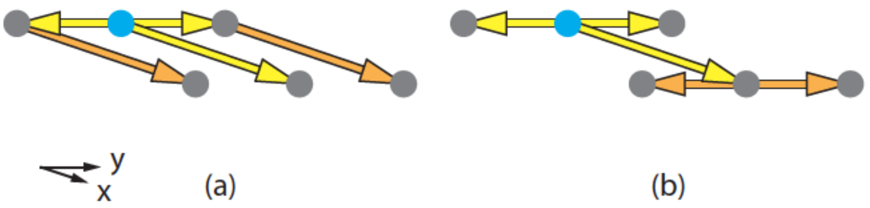


Figure 5·5: Reconfigurable force data forwarding pattern examples for the pillar distribution. Yellow and orange arrows represent the 1st hop and 2nd hop, separately. Blue node: Home FPGA node. Grey node: Neighbor FPGA node.

be problematic, however, for a board with 6 transceiver ports to send data to 4 directions, but it becomes an advantage for boards with 4 transceiver ports. Users can decide which distribution best suits their hardware and problem.

Data Path Analysis

In this section, we explore how data are transmitted across nodes and how the data forwarding paths can be configured. The data path for the slab pattern is trivial, since an FPGA only sends data to the node on x-positive direction and receives data from the node on x-negative direction (see Figure 2·1). This indicates that a node sends data to, or receives data from, 5 neighboring nodes for the pillar layout and 17 neighboring nodes for the block layout.

As an intermediate FPGA node is involved in data forwarding, the I/O bandwidth demand naturally escalates. Figure 5·5 and 5·6 illustrate instances of data transfer patterns associated with pillar and block distributions, respectively. In these patterns, the objective is to acquire 2-hop data from 1-hop data and 3-hop data from 2-hop data, facilitating

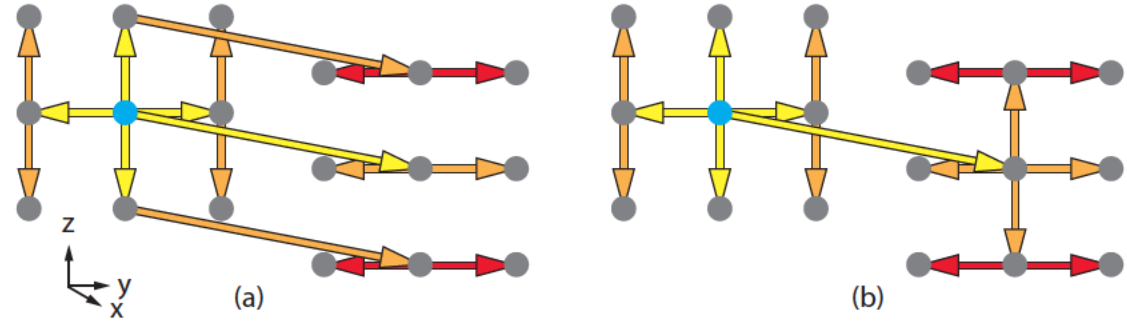


Figure 5·6: Reconfigurable force data forwarding pattern examples for the block distribution. Yellow, orange and red represent the 1st hop, 2nd hop and 3rd hop separately. Blue node: Home FPGA node. Grey node: Neighbor FPGA node.

data reuse. By reconfiguring the bandwidth along various directions, as exemplified in the figures, distinct bandwidth requirements arise.

Bandwidth Balancing

Since the number of cells on a single node is limited, the amount of data being forwarded two hops or three hops away is considerable compared with the data forwarded directly (one hop). As previously shown, the 2-hop and 3-hop data forwarding paths can be adjusted so that the data flow on different paths. This property can be exploited to achieve balanced communication workload in all directions and, ultimately, balanced data flow through all ports.

We now assume that all nodes use the same data forwarding pattern so that the data flow in opposite directions is symmetric, i.e., the bandwidths on direction α_+ and α_- are identical, where α can be x, y, or z direction, so are β and γ in the following discussions. The bandwidth of the block pattern (including the bandwidth demand of the 2-hop and 3-hop data to be forwarded) in a certain direction is then

$$B_\alpha = \sum_{faces} l_\beta l_\gamma w_d f + \sum_{edges\ \beta} l_\gamma w_d f + \sum_{edges\ \gamma} l_\beta w_d f + \sum_{vertices} w_d f \qquad (5.4)$$

In the equation, l_β denotes the number of cells on a node along β direction. The edges β term represents the bandwidth demand to forward the edge cell data towards β_+ and β_-

directions.

For 2-hop data transmissions along the α direction, data on the sides with cell dimension l_β and l_γ can only be sent via neighbor nodes on the γ and β directions separately first, then forwarded along the α direction. This behavior is abstracted into the 2nd and the 3rd term. To achieve balance, the bandwidth bottleneck must be minimized as shown here

$$B_{bottleneck} = max(\{B^1_{port}, ..., B^N_{port}\}) \tag{5.5}$$

where B^i_{port} is the bandwidth demand for the *i*th port, based on routing configurations. The bandwidth is naturally balanced for a slab since data transmission workload can be uniformly distributed. However, slab and pillar distributions have scalability issues: they may be overwhelmed by an oversized simulation space. Based on the distributions in Figure 5·4, position and force bandwidth demands are decomposed by directions in Table 5.1 and 5.2.

1-hop data paths are not configurable since the destinations are just next to the origin. In the 2-hop and 3-hop cases, however, the data transfer paths can be re-configured to alternative directions. For example, in Figure 5·5, the 2-hop data can be forwarded along x direction (Figure 5·5(a), or y direction (Figure 5·5(b), such that the bandwidth demand for the 2-hop forwarding is included in total x_+ bandwidth or y_+ bandwidth. In other words, the bandwidth demand on a direction can be traded for the bandwidth demand on another direction. In the case of blocks, more freedom for configuration is granted since more path combinations are available. In the evaluation section, we demonstrate that up to 30% more data can be transmitted compared to average.

5.3.3 Motion Update Conflict

Motion update is comparably light-weighted than the position or force in terms of data transmission. The number of cycles for motion update is exactly the number of particles N, while for force evaluation, the number is $\frac{\alpha N}{N_p}$, where α is the average number of neighbor particles that can form valid pairs with a reference particle. Based on our assumption that $N_p < N_c$ or $N_p \approx N_c$,

$$\frac{\alpha}{N_p} \gg 1 \tag{5.6}$$

is obtained. Amdahl's law suggests that it is reasonable to deploy only a smaller number of update modules than force evaluation modules in a single node. In our case, for the sake of simplicity in demonstration, we assume only one motion update unit is equipped per FPGA node.

Despite the smaller workload, the data conflict problem caused by particle migration needs to be resolved. Same as the single-chip designs, throughout the motion update process, a particle can migrate to another cell in another node. Rarely but possibly, the remote update can cause a conflict with local motion update when they are being written to a same cache. The solution is to pause the local update and let the remote update finish first. Given the fact that all cells have a similar number of particles, the motion update units are almost synchronized and the conflict between two remote updates is nearly impossible. If more motion update units are added for more parallelism, the conflict becomes a problem and extra input buffers can be deployed to resolve the conflict.

5.4 Evaluation

Within this section, our initial analysis revolves around independently assessing the advantages brought forth by routing configuration and neighbor data caching. Subsequently, we proceed to evaluate the comprehensive speedup achieved by our proposed communication-efficient molecular dynamics (MD) design compared to the baseline. To gauge this improvement, we employ the metric of "execution time per MD iteration".

5.4.1 Evaluation of Routing Configuration

The discussions above show that bandwidth balancing is affected by various variables and can be optimized through different ways. In this section, we evaluate bandwidth balancing by taking into consideration the number of transceiver ports per board, the workload distribution pattern, and the cell distribution pattern.

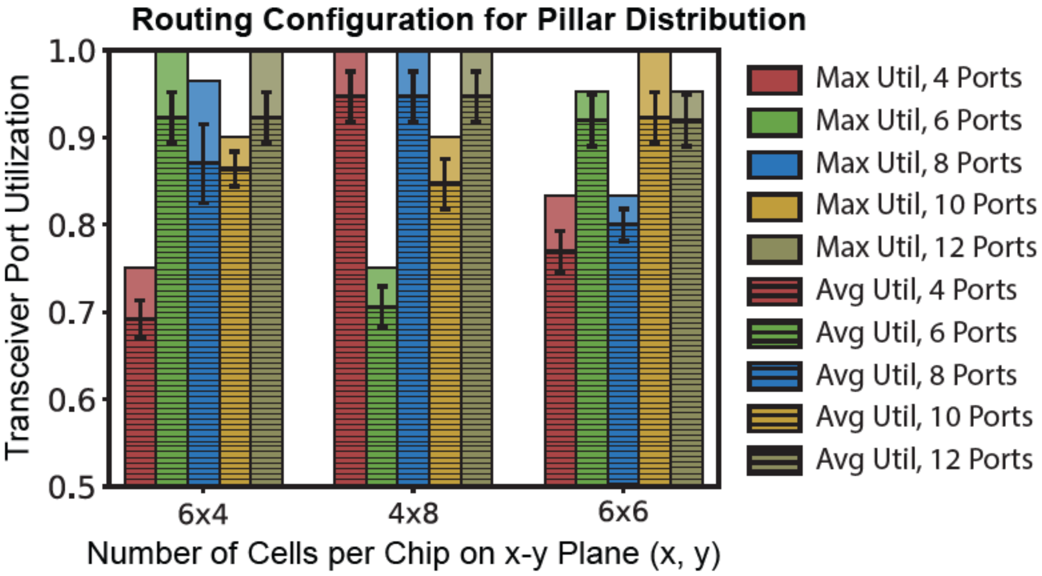


Figure 5·7: The port utilization due to the best routing configuration (Max Util) and the average utilization (Avg Util) of all routing configurations for 3 typical pillar distribution cases. Error bar: Standard deviation

Currently FPGA boards with high-bandwidth transceivers are common. For example, 2×, 4× and even 6× QSFP28 transceiver ports are available on various of boards. With FPGA Mezzanine Cards (FMC) involved, data transfer along more directions can potentially be supported. Furthermore, The bandwidth balancing mechanism can be adopted for any workload partitioning of simulation spaces, but we only demonstrate some representative cases here. To justify the effectiveness of the configurations discussed, transceiver port utilization is defined:

$$U_{ports} = \frac{B_{ideal}}{B_{bottleneck} \times N} \tag{5.7}$$

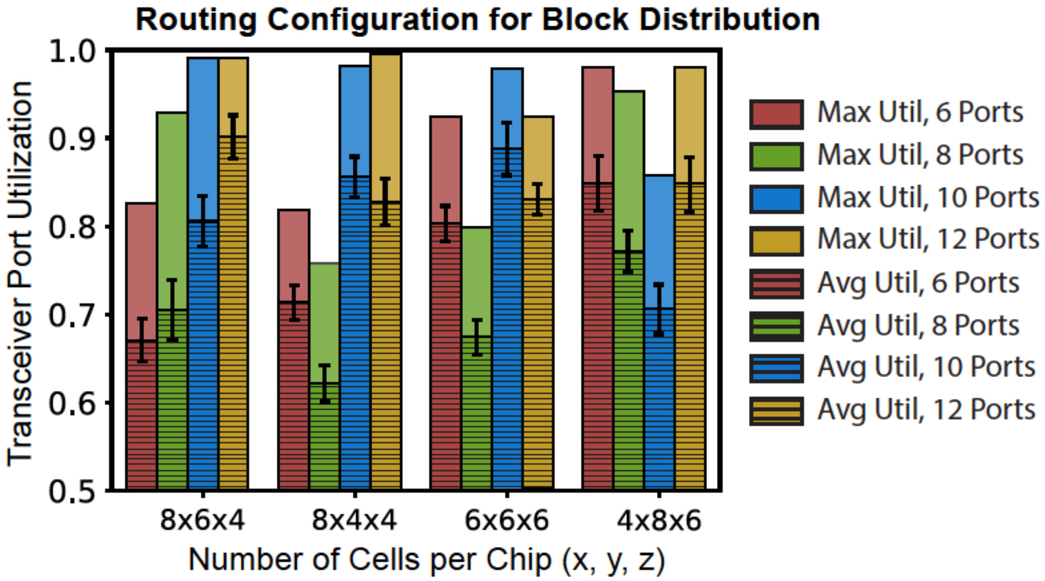


Figure 5·8: The port utilization due to the best routing configuration (Max Util) and the average utilization (Avg Util) of all routing configurations for 4 typical block distribution cases.Error bar: Standard deviation

where B_{ideal} is the overall bandwidth budget (equation (4), (5), and (6)), and N is the number of ports. For convenience, we assume that at least one of the ports is fully utilized, in order to evaluate the highest data transfer intensity that can be supported by the hardware. Because the average amount of data transfer per cycle is relatively stable, “maximum operating frequency supported” is used to evaluate at which FPGA operating frequency the communication becomes the bottleneck.

Figure 5·7 and Figure 5·8 illustrate port utilization cases with conspicuous dependency on the number of ports and the distribution of cells on chip, and reveals the problem that ports may stay idle for a significant amount of time if the routing path is randomly configured. We use as a baseline the average of all routing paths. In Figure 5·7, the highest utilization cases can be achieved when the 6×4×N_z cases are higher than 0.9, except for when there are only 4 ports (N_z is the number of cells along z direction, which is redundant for pillar evaluation). However, the utilization of 4 ports surpasses 6 ports by 25% for the 4×8 distribution, indicating that 4×8 is one of the optimal choices when only 4 ports are

Table 5.3: Throughput of a 2×2×2 FPGA cluster. The design runs at 350 MHz frequency, with 50 particles in each cell. Particle: Liquid Argon. Cutoff radius: 8.5 Å. Number of ports: 6. Bandwidth of each port: 100 Gbps. Data size per packet: 120 bits

Cell Space	8×6×4		8×4×4		6×6×6		4×8×6	
Design	baseline	new	baseline	new	baseline	new	baseline	new
Force Evaluation Time (μs)	3784	76	2621	76	3051	76	2586	76
Overall Time per Iteration (μs)	3854	147	2573	123	3130	156	2656	147
Max Frequency Supported (MHz)	7.0	434	9.2	527	8.7	502	10.3	595
Communication-Computation Ratio	26.2	0.42	20.9	0.41	20.1	0.34	18.1	0.30

available. The last case shows that the 6×6 distribution can be fully exploited for 10-port boards, but not 4-port or 8-port boards. All those boards have significant weaknesses except for 12-port boards, for they lack sufficient configurability. On the other hand, the highest utilization possible exceeds the average by 3% to 9%, meaning that the data transmission efficiency is boosted by these amounts for free.

Similar phenomena occur in block distributions, as Figure 5·8 shows. 6-port, 6 and 8-port, 8-port, and 10-port boards behave badly in those cases. It is worth noting that the amount of data transmission can be increased by 10% to 30%, for multiple possible routing paths can be taken, which gives more importance to the route configuration.

5.4.2 Evaluation of Neighbor Data Caching

The efficiency of neighbor data caching depends on the number of particles per cell. Since the reference particles are traversed as in Figure 5·3(b), neighbor position data are cached for the same number of cycles as the number of reference particles. Therefore, the bandwidth demand reciprocally depends on the number of particles per cell. Assuming that each particle participates in at least one valid pair,

$$B_{new} = \frac{B_{baseline}}{N_{max}} \tag{5.8}$$

where N_{max} is the largest number of particles of a cell among all cells, as all position broadcasts must be synchronized.

The neighbor data caching mechanism also brings the opportunity for routing configurations. The same amount of force data is sent in return for each piece of position data received. Therefore, the symmetry of position reading and force writing is reserved, which makes the routing configuration work for both position and force paths; i.e. the bandwidth requirement is the same for both reading and writing. Otherwise, uncertainty occurs in force packets' return, since destinations of the packets are random.

5.4.3 Overall Performance Evaluation

To avoid too much overhead in synchronization, a $2\times2\times2$ FPGA cluster is chosen to demonstrate the overall speedup due to the optimizations. Each FPGA has six transceiver ports (QSFP28). Table 5.3 illustrates the amount of time used in force evaluation per iteration, and the time needed for an entire iteration for four different cell distributions (block). For further improvements, the maximum supported frequencies are also listed, i.e. the highest frequency at which the communication latency can be entirely hidden by computation. The communication-computation ratio is listed as well for more straightforward comparison. In the baseline design, the communication latency exceeds the computation latency by a factor of 10. Also, with a reasonable operating frequency, the communication latency can be hidden completely.

For simplicity, the number of FCUs per node is set to be the same as the number of cells, and only one motion update unit is used. Therefore, for our optimized design, the force evaluation time remains unchanged. For all four cell distributions, the performance roofline of the baseline design is data transmission, while the roofline of the optimized design is the operating frequency (350MHz), which is below the maximum frequency supported. It is evident that by solving the bandwidth problem alone, the overall performance can be increased by a factor of 10, compared with the baseline.

5.5 Related Work

MD on FPGAs has been studied for many years [Azizi et al., 2004, Gu et al., 2006a, Scrofano and Prasanna, 2006, Kindratenko and Pointer, 2006, Alam et al., 2007, Cong et al., 2016]. The first complete MD systems on FPGAs accelerated the Range-limited (RL) force computations and used a CPU for the rest [Gu et al., 2005b, Kindratenko and Pointer, 2006, Scrofano et al., 2008]. Khan accelerated NAMD with four FPGAs, but they were used as independent accelerators [Khan et al., 2013]; see [Pascoe et al., 2020] for a recent four FPGA design.

When we broaden prior work to ASICs, a variety of spatial decomposition methods have been studied more than a decade ago by D.E. Shaw Research [Shaw, 2005], where zonal methods [Bowers et al., 2007] are discussed in detail. Among the zonal methods, the neutral-territory method and, specifically, tower-plate method, is used in their first Anton system [Larson et al.,] because of its low import volume. In this study, however, since a large amount of resources are available on modern FPGA chips and most data can be consumed locally, we chose half-shell decomposition so the data locality is naturally preserved.

Load balancing was further refined during the development of Anton 2 [Towles et al., 2014] where data forwarding routes can be adjusted at runtime. We have chosen not to use in-network reduction since an FPGA only exchanges data with nearby neighbors. As a result, the force and position packets can be scheduled and the data forwarding route can be determined offline.

5.6 Conclusion

This study showcases the data forwarding and communication patterns observed in the slab, pillar, and block distributions within a RL MD system. The transmitted data is categorized into three types: 1-hop, 2-hop, and 3-hop cases. Leveraging the inherent symmetry in data flow, we demonstrate the possibility of reconfiguring the paths of 2-hop and 3-hop data to achieve a balanced I/O bandwidth across FPGA nodes. Additionally, we establish that by appropriately configuring the routing path and cell distribution, the efficiency of data transmission can surpass the average by up to 30%. Furthermore, through the localization of neighbor particles, the total bandwidth cost is significantly reduced. By effectively combining both approaches, it becomes conceivable to achieve a potential tenfold increase in overall performance compared to the baseline design, with the added benefit of completely hiding communication latency.

Chapter 6

Optimized Mappings for Symmetric Force Calculations on FPGAs

Thus far, we have discussed the single-chip designs and multi-chip communication pattern configurations. In this chapter, we further focus on the communication of MD RL on multiple FPGAs, but from a more fundamental angle compared to Chapter 5. Motivated by force symmetry and prior ASIC work, a modified Manhattan method is proposed to replace the half-shell method under communication-intensive circumstances, in order to obtain drastic reduction in communication intensity.

6.1 Introduction

Exploiting physical symmetries to reduce computation is a fundamental method in Scientific Computing. In Molecular Dynamics (MD), Newton's 3rd law (N3L) results in the force symmetry that reduces by half the computation in short range (or range-limited, RL) force evaluation. The theory is straightforward, but, when coupled with the commonly used hard cut-off (range limit), the optimal computation-processor-memory mapping is dependent on the hardware model. N-body mapping has been the focus of several high-profile studies [Snir, 2004, Bowers et al., 2007, Bowers et al., 2006b], where particle, space, and force partitioning are justified. However, the models are not hardware-specific and, in particular, the mapping model for FPGA-based MD [Azizi et al., 2004, Gu et al., 2005b, Hamada and Nakasato, 2005, Kindratenko and Pointer, 2006, Alam et al., 2007, Scrofano et al., 2008, Chiu and Herbordt, 2010a, Yang et al., 2019a] remains to be established.

The problem reduces to the following: How should particle data be organized with respect to block RAM (BRAM) configuration to minimize data access redundancy?

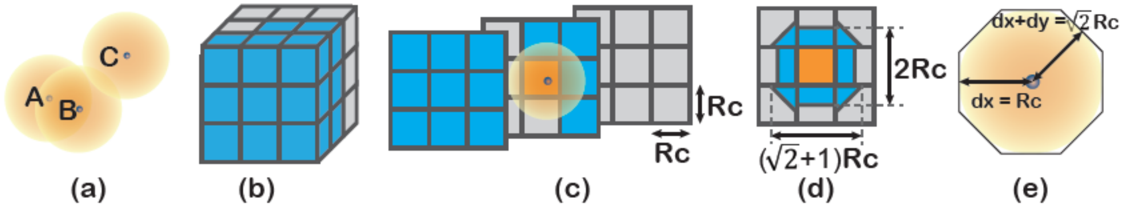


Figure 6·1: Cells, spatial partitioning methods, and filtering. R_c: Cutoff radius. (a): the cutoff radius of particles. The force A-B is valid, and A-C and B-C forces are too small and neglected. (b): the import volume of the half-shell method. (c): the spread layers of (b), where the orange cell at the center interacts with 13 blue cells and itself; the middle slice also shows the 2-D import volume. (d): 2-D import volume of the Manhattan method. (e): planar filtering method. dx and dy are the distance components between two particles; particles outside the octagon are not paired with the particle at the center.

Since current BRAMs typically have a depth of 512, a block of BRAMs (whose number depends on the BRAM configuration) is used to store up to a few hundred particles, which are accessed sequentially rather than concurrently. A commonly used optimization is to group particles geometrically in cells, namely, with cell-lists [Yao et al., 2004, Brown et al., 2011]. In previous FPGA-based MD work (since, e.g., [Gu et al., 2006a, Gu et al., 2008]), the edge length of a cubic cell is fixed at the cutoff radius (a criterion to neglect small forces, see Figure 6·1(a)); with current practice [Obeidat et al., 2018], a cell typically contains fewer than 80 particles in a water environment. The particles from a cell are evaluated in serial with respect to particles from neighboring cells as discussed earlier in single-chip designs.

Larger or smaller cells are less advantageous. Larger cells result in redundancy in computation, which directly degrades performance as a particle only interacts with a smaller portion of particles in neighboring cells, yet all the neighboring particles need to be accessed. For smaller cells, more neighbor cells need to be accessed for a particle, and a cell contains a smaller number of particles, so more slots in a BRAM are wasted. Subboxing [Grossman et al.,] in Anton 2, for example, is an extreme case of using small cells for more fine-grained data access, but not suitable for FPGAs based on the discussion above. Note that the advantages of larger boxes accrued in CPU implementations, where they reduce frequency of neighbor list recalculation [Phillips et al., 2005], are not applicable to ASICs/FPGAs where particle filtering [Chiu and Herbordt, 2009] is used instead.

The cell size now being set, it is straightforward to evaluate a cell, with N3L applied, with respect to the surrounding 14 cells (including itself) using the half shell method [Chiu and Herbordt, 2010a] (illustrated in Figures 6·1(b) and 6·1(c)). The major downside, however, is the *import volume*; i.e., 13 external cells need to be imported. An alternative is the Manhattan method used by the Anton 3 [Shaw et al., 2021]. Compared to the half-shell method, it demands a much lower import volume, as Figure 6·1(d) suggests in 2-D illustration. In the 3-D case, the import volume is reduced to the equivalent of about 7 cells.

Applying the Manhattan method to FPGAs, however, appears to require that cells be further partitioned into fine-grained subboxes, with the disadvantages just described. To tackle this problem, we

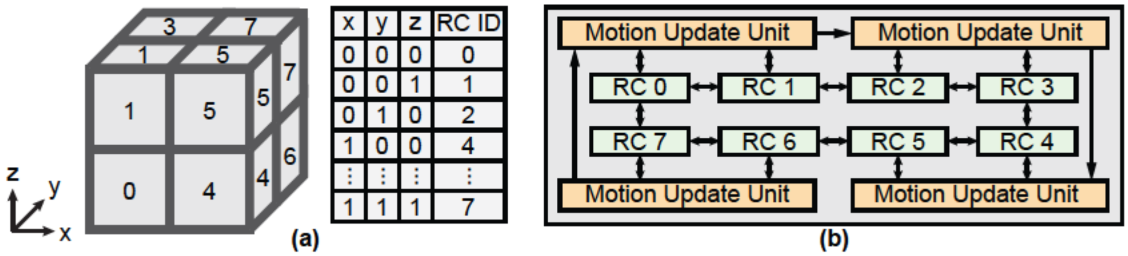
- propose a position cache overlapping method that maps our modified Manhattan method onto FPGA hardware without reducing the size of cells;
- design a complete MD RL architecture with minimal additional resources cost compared to the baseline half-shell design on FPGA;
- demonstrate that the Manhattan method used on multi-FPGA clusters can reduce the data transfer by 40% - 75%, while balancing data transfers along all directions.

The practical consequence is that nearly $2\times \sim 4\times$ the workload can be handled without upgrading the network of FPGA connections. This is a critical finding given the relatively limited bandwidth available in many common accelerator boards and the strong-scaling, communication-bound, applications to which FPGA clusters are being applied.

6.2 Design

As fine-grained concurrent data access is incompatible with common FPGA BRAM configurations, we demonstrate that the use of two sets of position caches (cell caches and corner caches, see Section III-B) with overlapped contents significantly reduces memory accesses, even without fine-grained space partitioning. To coordinate the two sets of position caches, the memory misalignment problem of the particles is also resolved with address and cell indexing methods. The BRAM allocation, filters, and rings are evolved from the baseline design described in [Wu et al., 2021a].

6.2.1 Logical Topology



x	y	z	RC ID
0	0	0	0
0	0	1	1
0	1	0	2
1	0	0	4
⋮	⋮	⋮	⋮
1	1	1	7

Figure 6·2: The overall layout of the design. (a): cell ID numbers are calculated from their x, y, and z coordinates in space. (b): the topology of 8 ring cores (RC) and 4 motion update units (MUs) for example.

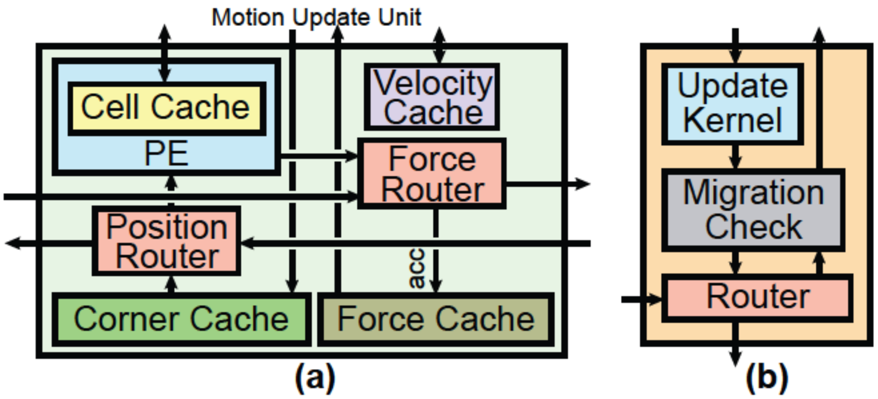


Figure 6·3: The overall layout of the design with more details. (a): the internals of a ring core. (b): the internals of a MU.

The ring-shaped topology is inherited from the baseline design with minor modifications. To demonstrate the ring topology, we label each cell in space with a cell ID, which follows the simple 3-D to 1-D mapping method:

$$CID = xD_yD_z + yD_z + z \tag{6.1}$$

where D_y and D_z are dimensions of the y and z directions. For example, in Figure 6·2(a), D_y and D_z are both 2. Each cell is assigned a ring core (RC) to process the interactions between the local particles in the cell and neighbor particles from other ring cores. The ring cores possess the same IDs as their local cells. In Figure 6·2(b), the RCs are logically connected as a ring corresponding to their IDs. This 3-D cell to 1-D ring mapping is both simple and minimizes the data travel time in the ring. In the case shown in the figure, two adjacent RCs share one motion update unit, and the motion update units are connected in a ring to resolve particle migration (i.e., when a particle travels to another cell).

An RC block diagram is given in Figure 6·3(a). The position, velocity, and force data, with respect to the particles in the local cell, are stored in cell cache, velocity cache, and force cache, respectively. In distinction from the baseline design, a duplicate cell cache is now a corner cache. A corner cache consists of $\sim$1/8 particles from the local cell and particles from 7 other cells. The **corner cache** is a key concept in this work and is discussed further in Section 6.2.2. The position input ring (i.e., the network of position routers from RCs) and the force output ring forward data in opposite directions. Since the computed forces are returned to the RCs where the position data originates, the effect is to reduce the data travel time.

Figure 6·3(b) shows the functions of a motion update unit (MU). A MU gathers particles' position, velocity, and force data and computes the particles' position and velocity after, say, 2 femtoseconds ($2 \times 10^{-15} s$). If a local particle has moved to another cell, the migration-check function passes the updated data to the router so the data is forwarded to the target MU through the motion update ring. The router in the target MU recognizes that the correct data have arrived and forwards it to the corresponding RC for update.

6.2.2 Corner Caches and Overlapping Position Caches

Figure 6·4 shows the principle of cell caches and corner caches. For conciseness in context, we refer them as yellow caches and green caches, respectively. The particles in the caches are named as cell particles (yellow particles) and corner particles (green particles). Both yellow caches and green caches contain particle position information (i.e., both, unless otherwise noted, are position caches, to be distinguished from the caches containing velocity and force data. There are also velocity and force cell caches with the same spatial layout as position cell caches, but there is no velocity or force corner caches.). A green cache is overlapped with a quarter (an eighth in 3-D) of each yellow cache, and that overlapped portion of position data is duplicated. The green caches cover the entire simulation space, making the number of green caches equal to the number of yellow caches. At runtime, execution proceeds with particles from a green cache successively being broadcast to all adjacent cells to be evaluated with respect to all of the yellow particles.

Position cache duplication is also necessary in the half-shell baseline method for home cell yellow particles caching in the force computation; otherwise the data locality is lost and all of the particle position data would need to be stored in the buffers attached to the filters.

6.2.3 The Modified Manhattan Method

In Figure 6·4, the green particle is a corner cell particle stored in the green cache and is sent to all 4 neighboring yellow cells for pairing, while the yellow particles outlined in black are stored in yellow caches. Each processing element (PE, as we will show in the following context) is mapped to a yellow cell, and the pairing is done by local PEs associated with the yellow cells. Now that the cached positions in the yellow caches and green caches overlap, the traditional Manhattan method needs to be modified: because there's no clear boundary between a yellow cache and a green cache, we compare the Manhattan distances to the green cell boundary instead of the boundary of two cells.

Figure 6·4(a) and (b) show the two common particle pair cases. Both cases indicate that the pair should be evaluated when the green particle has greater Manhattan distance ($D2 > D1$, opposite to the traditional), otherwise the pair may not be evaluated (the yellow particles is never sent to the cell holding the top-left particle).

To better illustrate the handling of the irregularity for the modified Manhattan method, we define two concepts: *shadow region* (SR) and *shadow particle* (SP). The SRs are the brown areas in Figure 6·4(c) and the SPs are the particles located in SRs. As the figure illustrates, the two particles are within the cutoff range and form a valid pair. However, the yellow particle is outside the yellow cell area. If the green particle only interacts with the particles in its four surrounding square cells (the closest cell to the yellow particle is the yellow cell), then the yellow particle is never evaluated with the green particle and vice versa. In other words, they are outside each other's import region. Therefore, we extend the size of the yellow cells to include the SRs, causing overlaps in yellow cells. If a particle is updated into a new SR during motion update, it is then updated to more than one destination.

6.2.4 Cell Cache Partitioning and Corner Particle Pre-checking

Figure 6·4(d) shows the total accessed area, including the SRs. Without optimization, 71% more particles are accessed for a green particle in the 3-D scenario, leading to a drastically decreased particle pairing rate. Although the size of SRs looks formidable, this situation can be improved.

Fortunately, in Figure 6·4(d), only the dark brown areas may be needed for the particles in the green cell at the center. Therefore the SPs can use the addressing spaces separated from the normal particles in the yellow cell (e.g. normal particles are located at address

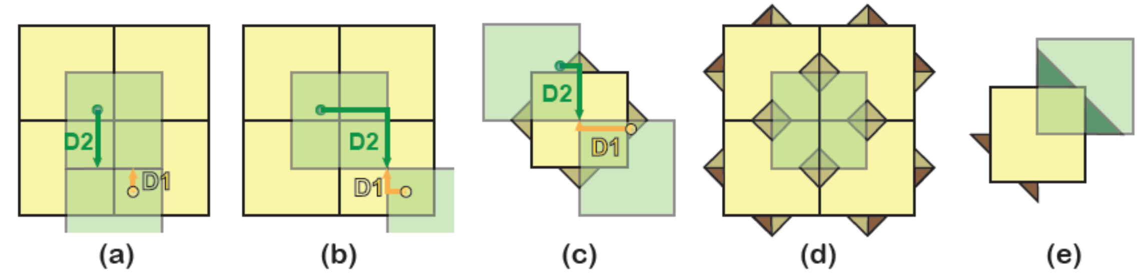


Figure 6·4: The rules of position cache overlapping and the modified Manhattan method in 2-D. Yellow cells: cell caches. Green: corner caches. The yellow/green particles are cell particles/corner particles separately, where the yellow particles are outlined in black. (a) and (b): the pair is evaluated if the neighbor particle has greater Manhattan distance. (c): the brown SRs should be included a yellow cache, otherwise the two particles shown will never be paired. (d): SR overview. Dark brown: the SRs potentially need to be accessed by green particles. (e): Only the green particles in the dark green region can may be paired with the SPs in the yellow cell.

1∼120, the SR requested by its northeastern corner cell is at 121∼140). The SRs are not accessed if not requested. Furthermore, Figure 6·4(e) shows that only a small number of green particles (in dark green regions) may request access to the SR, meaning the access can be further reduced by checking the green particles before pairing. In 3-D, ∼50% of the green particles can be exempted from accessing the SRs by pre-checking their position values.

Although the yellow cells look highly irregular, the actual operations required are only fixed point number additions and comparisons, and only the leading bits of the position data are used for such operations. For the rest of the chapter, the green caches are illustrated as cubic cells for simplicity.

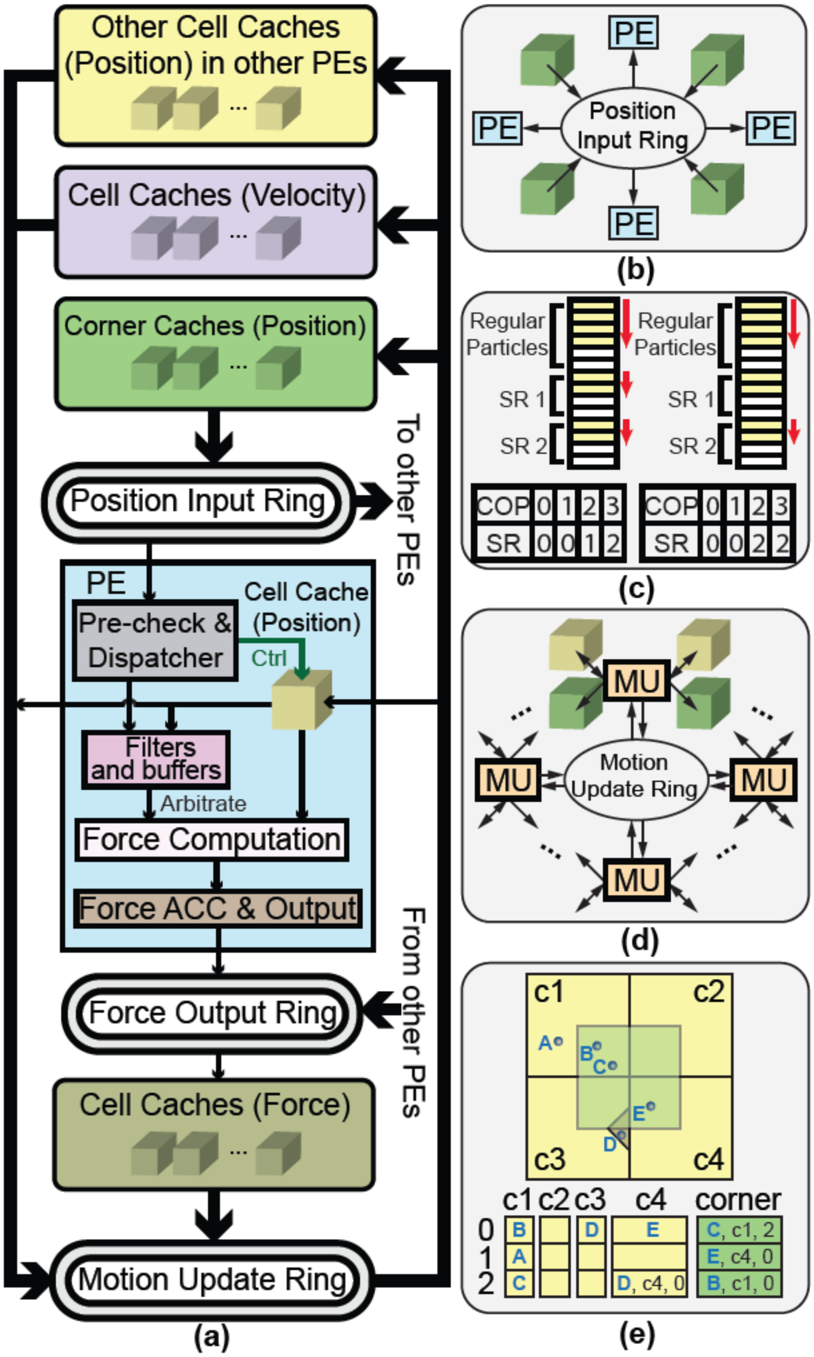
6.2.5 Architecture

The design flow is depicted in Figure 6·5(a) from a different perspective compared with the overall layout in figure 6·2 and 6·3. Here we mainly focus on the behaviors of PEs and abstract the caches and routing components into block diagrams for conciseness. The execution of MD RL starts from the green caches. The particle positions are injected into the position input ring as sketched in Figure 6·5(b), and each position has only 8 (instead of 14) destinations corresponding to 8 cells in 3-D. This also reduces the lifetime of packets in the ring. For each cycle, a position packet stops at a force processing element (PE) to check if a destination has been reached. Once there, the position packet is buffered and dispatched to one of the filters by the dispatcher.

Before dispatching a green particle downstream, the particle is first pre-checked to determine if any SR needs to be accessed based on its position. After pre-checking, the requested SR ID is obtained. The ID is further used in SR access handling as Figure 6·5(c) shows. For the left case, green particles 0 and 1 do not need to access SRs, but 2 and 3 request both SRs be accessed. As a result, all yellow particles, including the SPs, are traversed for all the green particles. For the right case, SR 1 is not requested, only the regular yellow particles and particles in SR 2 are traversed. In practice, there are up to 8 SRs corresponding to 8 corner blocks surrounding a cell block, and the number of green particles processed simultaneously in a single PE is relatively small.

The filters are originally used to evaluate the distance of two particles and check if the pair is valid. At runtime, each filter is mapped to a single green particle and vice versa. The Cell BRAM in the PE is traversed iteratively during force evaluation for pairing. With the Manhattan method involved, the filters also check the Manhattan distance between the particles and the corner block boundary. The good news is, not all the position bits are evaluated in the filters, but only, say, 8 leading bits are used. The number of bits is subject to the precision desired. Compared with the half-shell method, the filter pass rate increases from $\sim$17% to $\sim$30% (disregarding the overhead, the actual pass rate subjects to the number of filters and is further discussed in the evaluation section) for uniformly distributed particles, and the number of filters is therefore effectively reduced. The resources saved can then contribute to building more PEs for higher throughput.

The force fragments of green particles and the accumulated SP forces are then injected



COP	0	1	2	3
SR	0	0	1	2

COP	0	1	2	3
SR	0	0	2	2

	c1	c2	c3	c4	corner
0	B		D	E	C, c1, 2
1	A				E, c4, 0
2	C			D, c4, 0	B, c1, 0

Figure 6·5: The complete design with the Manhattan method. (a): the data flow with respect to a single PE. (b): the green particles enter the position input ring and are rotated to several destination PEs (c): two SR handling cases. Red arrow: data traversal order; left: all SRs are requested; right: SR 1 is not requested and is skipped. COP: corner particle. (d): the motion update ring. Each MU on the ring directly updates several yellow/green caches. (e): corner BRAMs store particle positions, cell IDs and the particle IDs for force write back and memory misalignment handling. The same mechanism is applied on SPs.

to the force output ring, while the forces of the regular yellow particles are directly integrated in the force caches of the cell. Each force packet in the ring only has one destination, such that the force data is not duplicated, and each force cache only covers a cubic volume without SRs. The SPs are inherently far away from the green particles, and their pairing chances are slim. Among all the cell-corner pairs, only $\sim$1% is contributed by shadow-corner pairs. The extra pressure on the force output ring caused by SP forces is therefore negligible.

After all forces are evaluated and integrated in force caches, the MUs start. A MU on the motion update ring inputs all three types of data (position, velocity, force) and obtain the position and velocity of particles for the next time step. The packets are either consumed locally (used to update the directly connected caches) or injected to a motion update ring (Figure 6·5(d), velocity and force caches omitted) to update the position corner caches and both position and velocity cell caches.

The workload of the motion update phase is significantly smaller than force evaluation, such that a much smaller number of MUs are equipped compared to the number of cells. Furthermore, with a MU directly connected to multiple yellow/green caches, the latency of a motion update ring is short compared to other rings. Moreover, because the force evaluation is not active during motion update, the position input ring and the force output ring can be reused to construct the motion update ring with minor cost in hardware resources.

6.2.6 Memory Misalignment

During force evaluation, the particles are easily aligned in force yellow caches and position yellow caches (excluding SRs). That is, a particle has the same address in both caches. However, the alignment cannot be preserved for green particles or SPs. For example, particle B in Figure 6·5(e) is at address 0 in yellow cache C1, the force is also located at address 0 in its corresponding force cache; therefore B can be directly updated with the force at the same address. However, B is at address 2 in the green cache instead of 0, i.e., memory misalignment. Similarly, the SP D is stored in the shadow addressing space in cell C4, with its original location at address 0 in C3. To correctly update the green particles and SPs, address and cell indexing information is added during motion update. In Figure 6·5(e), the green caches and the shadow addressing spaces in yellow caches not only contain the position values of particles, but also the cell IDs and addresses.

When there is no particle migration, a MU first updates a particle in a directly connected cell, then sends a packet to the motion update ring to find the destination green cache. If a particle migrates to a cell handled by another MU, the packet is delivered to the MU and next sent to the destination corner cell with the updated address. Fortunately, the particle migration is rare and the latency introduced is negligible.

6.2.7 Multi-chip Solution

The advantage of the cache overlapping method is particularly remarkable when applied to multiple FPGA nodes. Figure 6·6 compares the half-shell method and the cache overlapping method on 4 FPGA nodes with periodic boundary condition. For demonstration, each node contains 4×4 2-D cells. If the data are structured in plain cells, as Figure 6·6(a) shows, 10 of 16 cells (blue) need to be transferred to other nodes. In the new mapping with the cache overlapping, only the equivalent of 7 cells are transferred as shown in Figure 6·6(b).

The arrows only represent the position data transfer directions. The force data are returned to the source nodes along the arrows in reverse. This feature results in the natural balance between inbound force data and outbound position data with direct transceiver connections among FPGAs. However, there are two-way arrows in (a), potentially leading to heavy and imbalanced data transfer between the affected nodes compared to other transfer paths. Typically, each FPGA node provides ∼100 Gbps level bandwidth (2×100 Gbps QSFP28 for Intel D5005 and Xilinx Alveo U280), making the problem far more significant. The situation is much relieved in (b), where only one-way arrows are observed. This

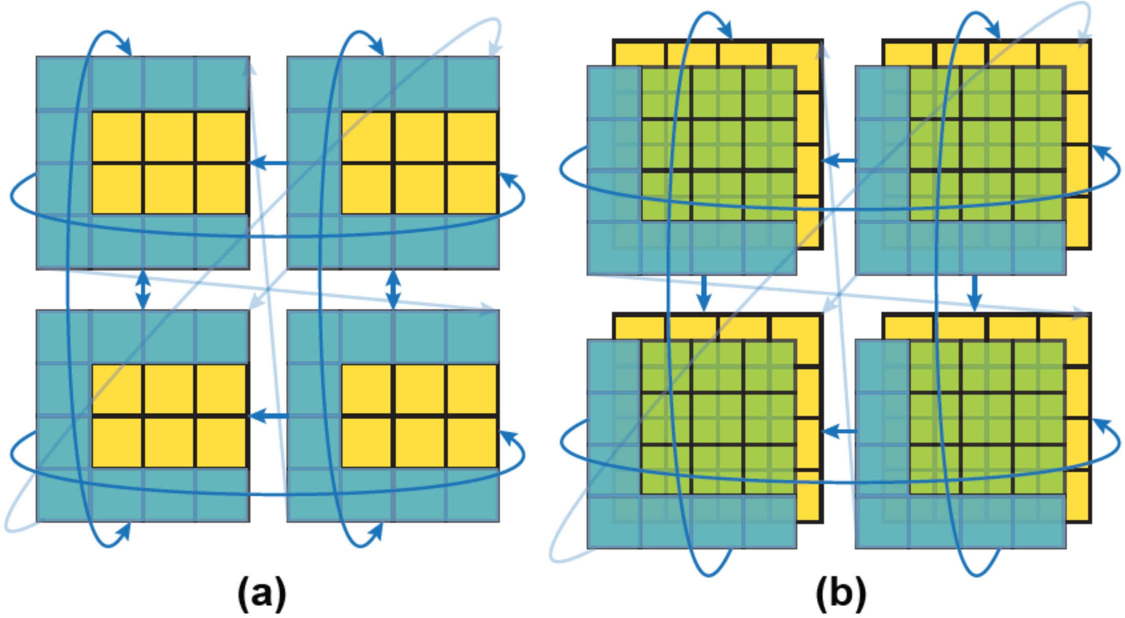


Figure 6·6: 2-D illustration of the data transfer pattern for 4 FPGA nodes. (a): half-shell method; (b): the proposed position cache overlapping Manhattan method. Yellow: cell region (SR omitted); Blue: data to be transferred; Green: corner region. The arrows indicate the transfer directions of the position data. For small data transfers at corners, the arrows are lightened.

implies that data transfer is almost perfectly balanced along any direction (up, down, right, left, front in 3-D, back in 3-D) with negligible small data transfers at corners.

6.3 Evaluation

In this section, we evaluate and compare the efficiency of the half-shell (based on FPGA-O2 architecture without hierarchical filtering mechanism in Chapter 4) and the cache overlapping methods in four aspects: How much more efficient can the filters get with the Manhattan method? How much latency can be reduced in the position input ring with reduced number of packet destinations? Can we hide the latency in the motion update ring with corner cells involved? How much data transfer can be avoided by using the cache overlapping method for different simulation space configurations and node topologies?

Although the design is relatively board-independent, the evaluations are based on the resources available on resource-abundant Intel D5005 boards with Stratix 10 SX FPGA chips, where each chip has 933120 ALMs, 5760 DSPs, and 11721 M20K BRAMs. The designs are implemented with Verilog and SystemVerilog HDL on Quartus 19.2 and validated on the D5005 boards. The resource/frequency results are obtained from reports generated by the Quartus software.

6.3.1 Performance

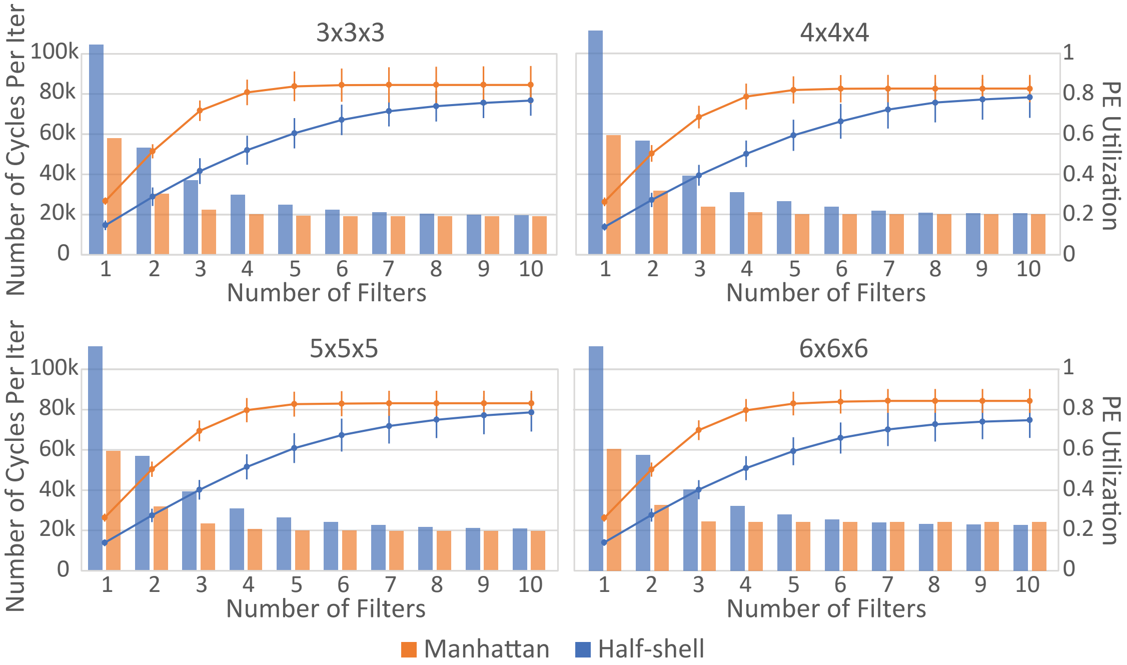


Figure 6·7: The performance and PE utilization versus the number of filters for 4 cell geometries. 3 motion update rings and 3 force rings are used. For consistency, we assume each cell is processed by 1 PE, and all 4 cases share the same y-axis. Bar: cycle. Curve: utilization

Figure 6·7 shows both the performance and PE utilization. For all four geometric cases, four Manhattan filters are almost as efficient as eight half-shell filters. The PE utilization is saturated at $\sim 85\%$, which can be regarded as the utilization limit of both designs. The Manhattan method reaches the limit at $\sim$5 filters, while the half-shell method requires $>$10 filters to achieve the same goal.

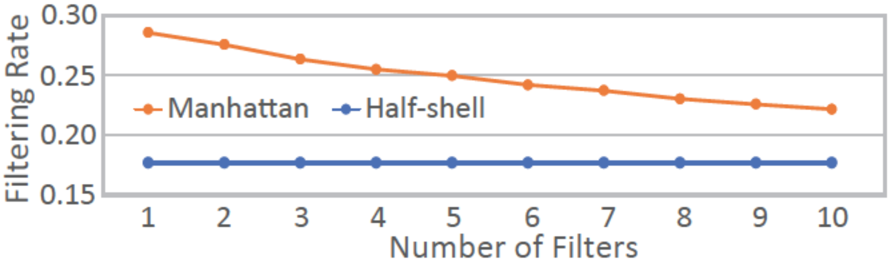


Figure 6·8: Filtering rate and the number of cycles required for filtering for the two methods. For 3^3, 4^3, 5^3 and 6^3 cell geometries, the filtering rates remain the same. We assume each cell is only processed by 1 PE for consistency against the scaling of the simulation space.

We also observe that for 6×6×6 cell geometry, the Manhattan method is bested by the half-shell method in performance. The reason is, the latency in the motion update rings is no longer negligible. During motion update, the Manhattan method updates more particle than half-shell due to the corner caches and the shadow regions. Moreover, with the upscaled number of ring nodes (27 MUs for 6×6×6 cells), the latency situation is worsened.

6.3.2 Filtering Rates

We observe from Figure 6·8 that the half-shell filtering rate is stable at ∼17%, yet the Manhattan filtering rate decreases, starting from ∼28%. This is because the SR handling mechanism in Figure 6·5(c) tends to include more filtering overhead as more green particles are processed simultaneously. As a result, more SRs are involved for pairing, but only for some of the green particles.

6.3.3 Position Input Ring Latency

Figure 6·9 shows the latency of the position input ring from the injection of the first particle to the departure of the last particle. The latency scales almost linearly with the number of cells per dimension for both methods and is robust against the increasing number of cells. The latency also scales linearly with the number of particles per cell. Although a cell usually contains fewer than 80 particles, we still are interested in our design's capability in handling more particles. Therefore, we use 80/100/120 particles per cell to investigate how the results scale with the number of particles.

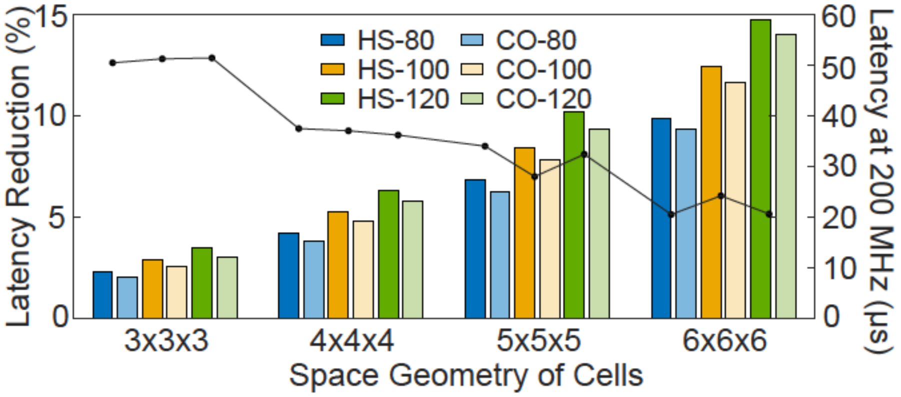


Figure 6·9: The latency of the position input ring of a single MD iteration. HS: half-shell; CO: cache overlapping. 80, 100, 120: number of particles per cell. The line plot indicates the percentage of latency reduced using the cache overlapping method.

The proportion of latency reduction is higher for small numbers of cells. Up to 13% of latency reduction is observed for 3×3×3 cells. The latency reduction is important because it allows more particles be allocated to PEs in a certain amount of time. 13% less latency means 15% more particles can be provided for PEs, potentially increasing the number of PEs working on a single cell, and eventually enhancing the performance for a limited number of cells.

6.3.4 Motion Update Ring Latency

As a trade-off, the latency of a motion update ring is increased. The comparison of the new and ideal latency is shown in Figure 6·10, where each cell contains 100 particles. For balanced resources, throughput, and wiring complexity, each MU is used to update 8 yellow caches and 8 green caches.

The ideal latency is obtained assuming all MUs only update their local cells without communication. The extra latency introduced is sensitive to the geometry. For smaller numbers of cells, only 2 or 3 motion update rings (2× or 3× ring concurrency) are enough

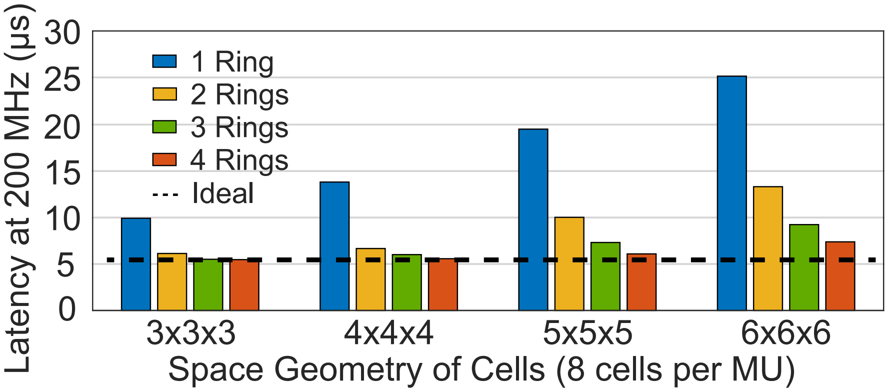


Figure 6·10: MU ring latency compared with ideal. Each MU is in charge of updating 8 yellow caches and 8 green caches. A cell contains 80 particles.

to reduce the latency to a negligible degree. For 6×6×6 cells, 3μs extra latency is introduced even for 4 rings. Fortunately, the latency is small compared to the overall latency of an iteration ($\sim$ 100μs for 6×6×6 cells, 200MHz). Furthermore, because the position and force rings can be reused for the motion update ring, and the number of MUs is significantly smaller than the number of cells, only a small amount of hardware resources are required to construct the extra rings, especially for smaller numbers of cells.

6.3.5 Multi-FPGA Data Transfer

The number of cells to be sent to remote FPGAs is reduced by a considerable amount ($\sim$30%), and scales almost linearly with the space edge length (Figure 6·11(a)). In fact, the actual amount of data transfer is reduced more significantly, as data forwarding is common in FPGA clusters (e.g., [George et al., 2016, Sheng et al., 2017a, Boku et al., 2019, Mondigo et al., 2020, Shahzad et al., 2021] and all-to-all connections [Plessl, 2018] are not always available.

Figure 6·11(b) gives the data transfer behavior on two likely FPGA cluster configurations. We assume each MD RL iteration takes 100 μs, each packet is 120 bits and each cell contains 100 particles. The bandwidth numbers are ideal without taking imbalanced data transfer into account. In (b1), the cluster has a 3-D torus topology. For all listed space ge-

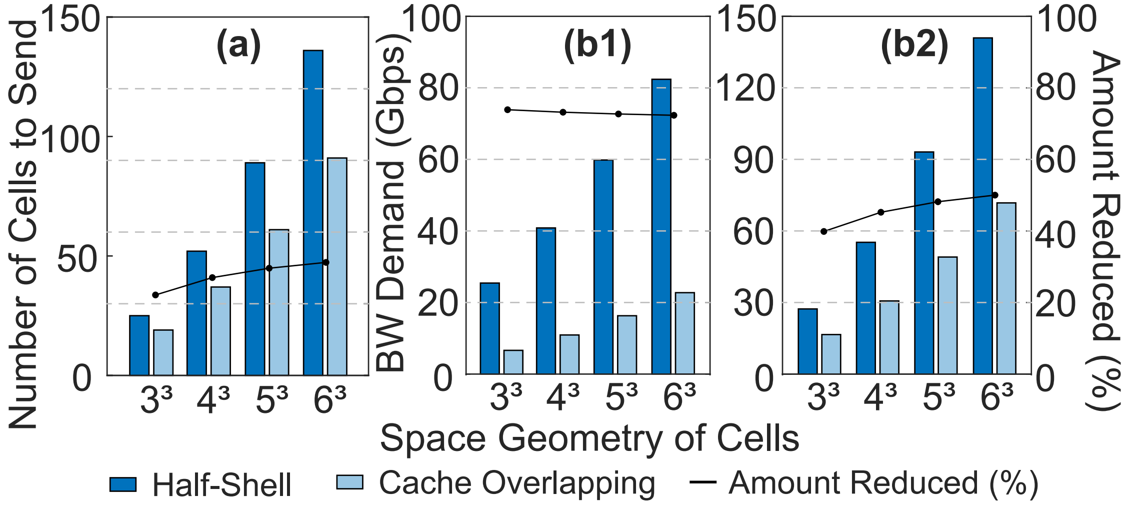


Figure 6·11: Data transfer per FPGA using the two methods. Each iteration (from force evaluation to the next force evaluation phase) takes 100 μs, with 120 bits per packet and 80 particles per cell. (a): the number of cells to be sent to remote FPGAs. (b1): the estimated ideal bandwidth demand per FPGA for a 3-D torus FPGA cluster. (b2): the estimated ideal bandwidth demand per FPGA for 8 FPGAs connected as a ring.

ometries, the data transfer reductions all approach 75%. In the 3-D torus, a packet needs to travel through at most 3 nodes to reach its destination. With the cache overlapping method applied, the proportion of data that require heavy forwarding is further reduced. In (b2), the cluster is 8 FPGAs in a ring, a likely scenario since many FPGA boards only have 2 or 4 transceiver ports. In this case, data may travel with more hops than a 3-D torus, such that the overall bandwidth requirement is raised. Still, 40%∼50% of data transfer can be saved.

The significance of these results is found especially in the most likely FPGA cluster use-cases, which involve strong scaling challenges (e.g., small molecule docking). With, say, 3×3×3 cells per node, FPGA capacity allows multiple PEs (8 in our case, 216 PEs in total) to work on the same cell, reducing the time per iteration to 1/8th the previous. As a result, 8× bandwidth is required, which is 200 Gbps. On the other hand, compared to the 6×6×6 case (with one PE per cell), only ∼80 Gbps are required. This is because the ratio of surface area to volume (SATV) ratio for the 6×6×6 cells is much lower. With strong-scaling (more PEs and/or fewer particles), the SATV ratio increases. At that point

Table 6.1: Hardware Costs

Cell Space	Design	ALM	BRAM	DSP
3×3×3	M[1]	112924 (12.1%)	1296 (11.1%)	621 (10.8%)
	HS[2]	116146 (12.4%)	1215 (10.4%)	621 (10.8%)
4×4×4	M	236220 (25.3%)	3072 (26.2%)	1472 (25.6%)
	HS	243464 (26.1%)	2880 (24.6%)	1472 (25.6%)
5×5×5	M	429812 (46.1%)	6000 (51.2%)	2875 (49.9%)
	HS	443011 (47.5%)	5625 (48.0%)	2875 (49.9%)
6×6×6	M	713675 (76.5%)	10368 (88.5%)	4968 (86.3%)
	HS	735274 (78.8%)	9720 (82.9%)	4968 (86.3%)

[1] Manhattan
[2] half-shell

the computation is completely compute bound and the performance is proportional to the amount of data transferred.

6.3.6 Hardware Resource Usage

The hardware resources demanded for both designs are listed in Table 7.1. Each half-shell PE has 6 filters, while each Manhattan PE has 4 filters. Both designs are equipped with 3 force output rings and 3 motion update rings. The Manhattan filters have higher logic expenses for their higher complexity compared to half-shell filters, but the overall ALM consumption is slightly reduced due to the reduction in the number of filters. The overall BRAM consumption is slightly higher because with ∼10 more bits included in position caches for cell indexing (see Figure 6·5(e)). Originally, the fixed-point position data and particle type (e.g., oxygen) together are ∼75 bits. With the 10 additional bits, we need 3 BRAMs side-by-side to satisfy the concurrent access of all 85 bits, where each BRAM is 40-bit wide, meaning another BRAM is required in each position cache. The new design requires no extra DSPs.

6.4 Related Work

The study of parallel N-body computation with cutoff dates back to two decades ago [Snir, 2004]. This work analyzes the communication requirements with respect to space decomposition and force decomposition algorithms. However, the force symmetry from Newton's 3rd Law is not taken into account. In [Shaw, 2005, Bowers et al., 2006a, Bowers et al., 2006b, Bowers et al., 2007], force symmetry is adopted, and Newton's 3rd Law is interpreted in a variety of ways. In their work, methods such as half-shell, midpoint, and tower-plate are summarized in depth. It is also proven that compared to half-shell, neutral territory methods (midpoint, tower-plate, etc.) generally result in reduced import volumes. For all the Anton series [Shaw et al., 2007, Shaw et al., 2014, Shaw et al., 2021], neutral territory methods are employed. Particularly, Anton 3 adopts the Manhattan Method that leads to the import volume both small and symmetric.

6.5 Conclusion

In this chapter, we compare the baseline half-shell model with the improved cache overlapping model based on the Manhattan method and find almost no difference in resource usage. The findings are as follows.

First, the filters in the Manhattan design are much more efficient compared to the half-shell filters. The PE utilization approaches its limit with only 4 Manhattan filters, whereas 8 filters are needed for the same with half-shell. Although the filtering rate of the Manhattan filters decreases due to SP handling, it is still considerably higher than that of the half-shell filters. Second, the latency of the overall position data input is reduced, especially for smaller numbers of cells ($3\times3\times3$) where 15% more work can be distributed to PEs in the same amount of time. Third, as the trade-off, the motion update latency is increased, but can be negligible thanks to the reusable hardware. Fourth, $\sim$75% of the data transfer can be saved with the new method on a 3-D FPGA torus, and 40% to 50% of the transfer can be saved with 8 FPGAs connected as a ring. A major benefit is that the pressure in data transfer is greatly relieved for FPGA-based MD where commercially available boards have significantly less available bandwidth than custom ASIC-based systems.

Chapter 7

FASDA: An FPGA-Aided, Scalable and Distributed Accelerator for Range-Limited Molecular Dynamics

7.1 Introduction

Molecular dynamics (MD) is a scientific technique that utilizes physical laws to simulate the movements and interactions of atoms. With the recent COVID-19 pandemic, drug discovery has gained significant attention, and MD has proven to be essential in predicting drug-target interactions and optimizing drug properties [Ganesan et al., 2017, Zhao and Caflisch, 2015, Salo-Ahen et al., 2020, Liu et al., 2018]. However, discovering a new drug typically requires a significant investment of hundreds of millions of US dollars and many years of work [Mullard, 2014]. To mitigate these costs, it is imperative to accelerate the drug discovery process through the use of MD simulations, especially for small sets of particles ($\sim$50K) as they are crucial in drug discovery [Mortier et al., 2015, Aminpour et al., 2019, Salo-Ahen et al., 2020].

Despite having a small dataset, the execution time of MD simulations is not necessarily short. To achieve accuracy, MD runs iteratively over discrete, infinitesimal time intervals (approximately femtoseconds, or 10^{-15} seconds) at runtime. Even 100 ns is considered a long timescale because it involves millions of iterations with complicated processes. However, due to the data dependency, the sequential iterations cannot be parallelized, which poses a significant challenge in strong-scaling as we need to accelerate the limited workload in a single iteration. To address this challenge, we need an approach that can preserve high efficiency with a vast amount of powerful computing nodes, where each node works only on a small piece of data.

There are numerous MD software packages available today that are regularly updated [Case et al., 2005, Phillips et al., 2005, Eastman and Pande, 2010, Abraham et al., 2015, Thompson et al., 2022, Bowers et al., 2006a], among which many support both CPUs and GPUs. However, CPUs have lower parallelism capabilities, resulting in generally lower performance compared to GPUs. GPUs are extremely powerful in throughput, but their exceptional capability in parallel computing requires exceedingly regular data to be fully exploited, resulting in somewhat suboptimal scalability for small datasets [Páll et al., 2020, Glaser et al., 2015]. It is also common that one single GPU outperforms more GPUs. Apart from the software approaches, application-specific integrated circuits (ASIC) often provide unrivaled performance due to their flexibility, such that the data paths can be customized and optimized for the data with low parallelism. For instance, known for its excellent scalability, Anton 3 can attain a simulation rate of $\sim$200 μs-per-day for 10K particles with 512 nodes [Shaw et al., 2021]. Meanwhile, its predecessor, Anton 2, can achieve around 100 μs-per-day with the same number of nodes [Shaw et al., 2014]. However, ASICs are expensive in production an maintenance, and are less accessible to the public thus giving limited contribution to the community.

FPGAs, like ASICs, offer hardware flexibility, but unlike ASICs, they do not suffer from the general availability or maintenance issues. Additionally, FPGAs are known for their low-latency communication capabilities, making them ideal for the latency-critical MD application that requires frequent data exchange between multiple FPGA nodes. The combination of flexibility and low-latency communication makes FPGAs an excellent foundation for strong-scaling purposes.

Besides, the computing power of FPGAs is highly competitive. The on-chip resources

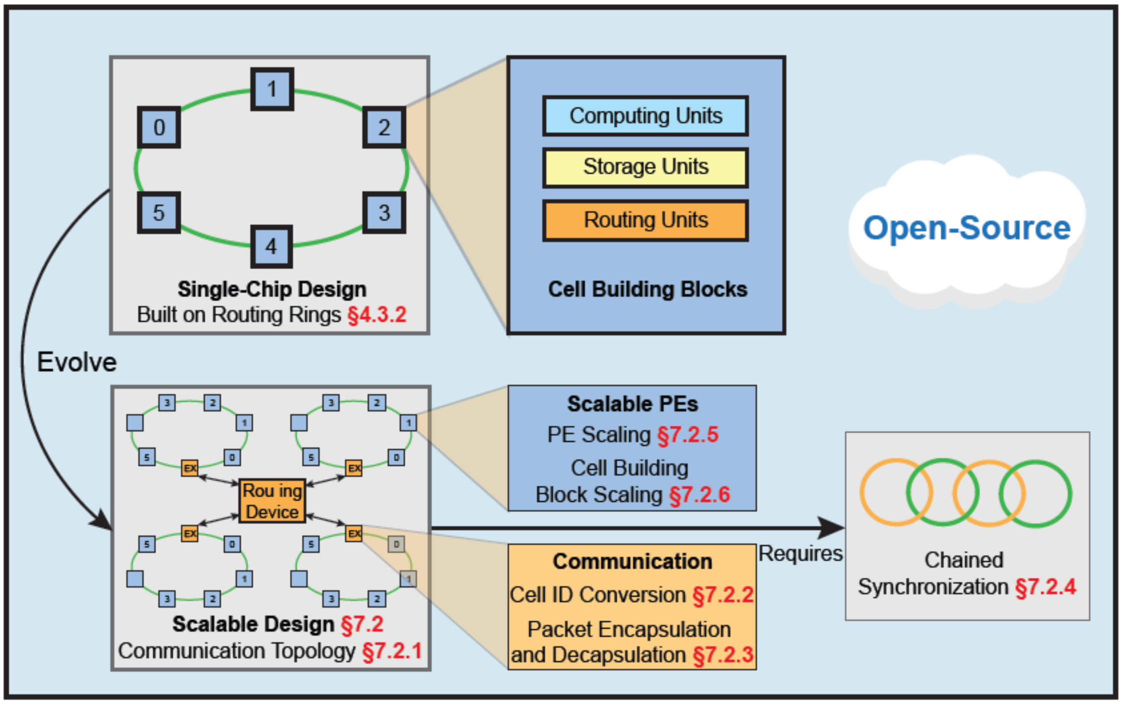

0
1
2
3
4
5
Single-Chip Design
Built on Routing Rings §4.3.2
Computing Units
Storage Units
Routing Units
Cell Building Blocks
Open-Source
Evolve
EX
Rou ing Device
Scalable Design §7.2
Communication Topology §7.2.1
Scalable PEs
PE Scaling §7.2.5
Cell Building Block Scaling §7.2.6
Communication
Cell ID Conversion §7.2.2
Packet Encapsulation and Decapsulation §7.2.3
Requires
Chained Synchronization §7.2.4

Figure 7·1: Overview of FASDA.

of FPGAs have grown from hundreds of logic blocks a decade ago to almost a million currently, enabling researchers and engineers to program FPGAs for complicated applications such as MD in our case with high performance. In recent years, FPGA-based MD works have been reported to outperform GPUs of comparable generations, even on a single node [Yuan et al., 2022, Yang et al., 2019a, Wu et al., 2021b].

MD simulations mainly involve computing non-bonded forces, which are partitioned into N-body range-limited (RL) with a cutoff, and long-range (LR) force evaluation components. RL is computing-intensive and consists of approximately 90% of the computation, while LR is more memory and communication oriented. The two components are relatively independent in terms of data flow and can be treated as two separate tasks. In this work, our focus is on the RL component, which is also the primary focus of the initial studies on FPGA-based MD [Azizi et al., 2004, Hamada and Nakasato, 2005, Gu et al., 2005a, Kindratenko and Pointer, 2006, Scrofano et al., 2008].

With the operating platform and the problem determined, in order to fulfil our anticipation for strong-scaling in MD, a completely distributed, decentralized computing system is particularly advantageous. Without a central authority, inter-node communication is simplified with latency reduced, and adding more nodes to the system requires minimal additional effort. However, similar to ASICs, designing and implementing such a system on FPGAs requires a significant amount of effort and professional knowledge, resulting in a research area that is largely unexplored. Although FPGAs cause widespread discussions in high performance computing, there is no answer clear enough to demonstrate the great potential within FPGAs, especially considering there is not a lot of work available publicly on an FPGA comparing to the general purpose platforms, let alone multi-FPGA implementations. To bridge this research gap and turn it into a new frontier, we present FASDA, to our best knowledge, the first open-source, scalable, fully distributed RL N-body simulation system with a cutoff that takes advantage of the benefits of FPGAs described above.

As highlighted in figure 7·1, FASDA is built with a series of easily plugable components that can be adjusted subjecting to user requirements with minor modification. First, a baseline **Cell Building Block (CBB)** is presented for a single chip, which is composed of a set of computing, storage, and routing units. Second, the system is evolved to the decentralized design with a hyper-ring shaped communication topology, synchronized using the **chained synchronization** technique, together with a **cell ID conversion** method to ensure its decentralized feature. Finally, for strong-scaling purposes, the original CBB is evolved to **scalable cell building block (SCBB)** with **scalable PEs (SPE)** equipped. Our system demonstrates almost linear performance scaling with 8 nodes and achieves over 4x speedup compared to GPUs.

The system could not have been established without the recently emerging FPGAs in the cloud. Various cloud platforms, including Microsoft Azure [Zhang et al., 2017], Amazon EC2 F1 [ama,], FAbRIC [uta,], Chameleon Cloud [Keahey et al., 2020], and Open

Cloud Testbed [Handagala et al., 2022], offer FPGA clusters that can be readily accessed. With the availability of these FPGA clusters and the designs that can be mapped to them, we anticipate a growing trend of FPGA applications emerging in the near future.

We list our overall contributions as follows:

- A strongly scalable, decentralized, and open-sourced MD simulation solution designed for high performance computing system is proposed.
- To maintain a decentralized system, a cell ID conversion technique and a chained synchronization method are employed, which ensure consistent nodes and PEs on each node without the need for global synchronization.
- Two levels of plugable strong-scaling modules are demonstrated, giving users the flexibility to customize the MD system according to their performance demand and available hardware resources.
- Our evaluation demonstrates that the proposed FASDA delivers over 4.67x speedup for drug discovery based on molecular dynamics simulations compared with the state-of-the-art GPU-based solution.

7.2 Scalable Architecture

In this section, the evolution of the scalable, multi-chip version from the single-chip version is outlined, which is capable of both weak-scaling and strong-scaling. Additionally, the methodology used to construct a synchronized and decentralized MD system is demonstrated.

7.2.1 Hyperring-like Communication Topology

Hyperring topology in network is known for its low bisection width and relatively low diameter [Sibai, 1998], granting the minimal cost in adding more PEs or more nodes to the system with relatively small latency overhead. However, this comes at the cost of increased bandwidth demands for nodes communicating over greater distances. Despite having poor bandwidth demand scalability, the hyperring topology exhibits distinct advantages for our MD system. The number of CBBs or nodes may need to be adjusted to compensate for on-chip resources and scale the system to a different size, and the effortless insertion of CBBs or nodes into the system offers flexibility to both the system and users. Additionally, range-limited results in a limited number of neighboring nodes, resulting in relatively constant latency overhead and bandwidth demand as the system scales, avoiding the bandwidth degradation in distant communication.

Figure 7·2 provides a depiction of the hyperring-like topology of the inter-FPGA connection. The on-chip position ring (PR) and force ring (FR) are both equipped an extra "EX" ring node for external data transaction, merely adding 1-cycle latency to the rings. Instead of exchanging data between PEs and caches, the "EX" nodes exchange data between local and remote nodes in a similar manner. This allows remote data to join the sub-network in the local node immediately upon arrival.

The network routing device can be replaced by other FPGA nodes directly connected as a ring to facilitate a hyperring of 2nd order, or a network switch to form a hyperring-like communication topology. As the system scales up, cascading switches can be used to connect nodes, and a hyperring of 3rd order can also be established through direct connections between FPGAs, which can be achieved using FPGA mezzanine cards (FMC).

The right side of figure 7·2 shows a demonstration of eight FPGAs, where each FPGA is assigned to evaluate a specific section of the simulation space. While physically connected on the user-defined network routing device, the FPGAs are logically organized to form a 3-D torus, which indicates the communication pattern.

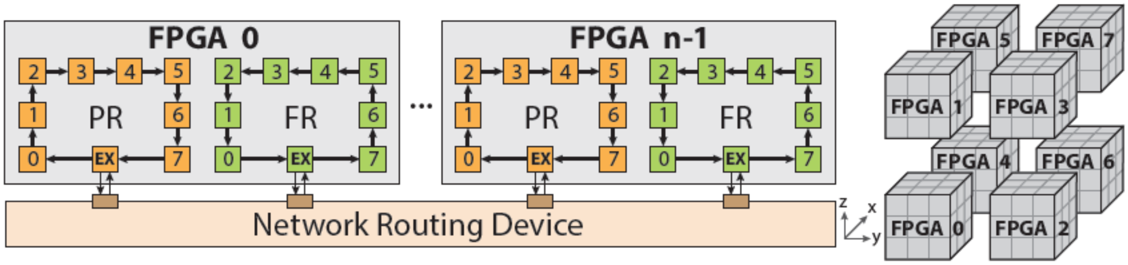


Figure 7·2: The topology of inter-node connection and an example of cell-to-FPGA mapping.

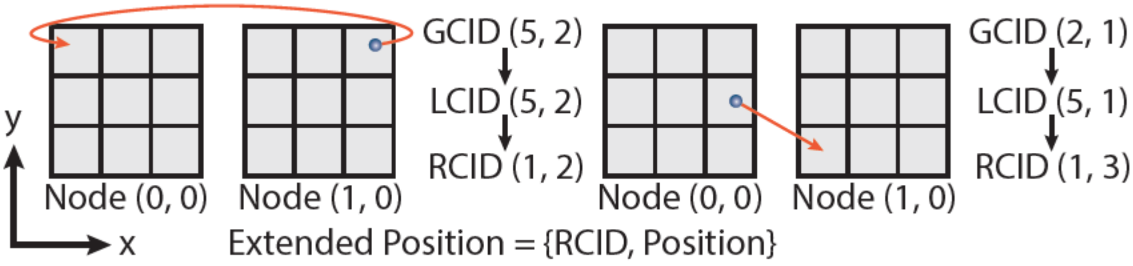


Figure 7·3: Two levels of cell ID conversion with examples.

7.2.2 Cell ID Conversion

To maintain homogeneity among FPGA nodes and CBBs, two levels of ID conversion are set up: one to convert the global cell ID (GCID) to local cell ID (LCID), and another to convert LCID to relative cell ID (RCID).

Each cell in space has a unique GCID for identification, but dynamic cell ID matching is required to check if a particle has arrived at its destination when using GCID directly. To avoid frequent dynamic ID matching, the GCID of a particle from another node is converted to LCID only upon arrival. This assigns a static cell ID to each local cell as if they are all from node (0, 0), ensuring homogeneous cell ID matching on any FPGA node. Figure 7·3 illustrates two 2D ID conversion examples. The left example shows a particle from the cell with GCID (5, 2) in node (1, 0) being sent to node (0, 0) for evaluation. Its LCID stays the same as its GCID since node (0, 0) does not require GCID conversion. The right example shows a particle from the cell with GCID (2, 1) in node (0, 0) being sent to node (1, 0). Its converted LCID becomes (5, 1), indicating the relative position between the source and the

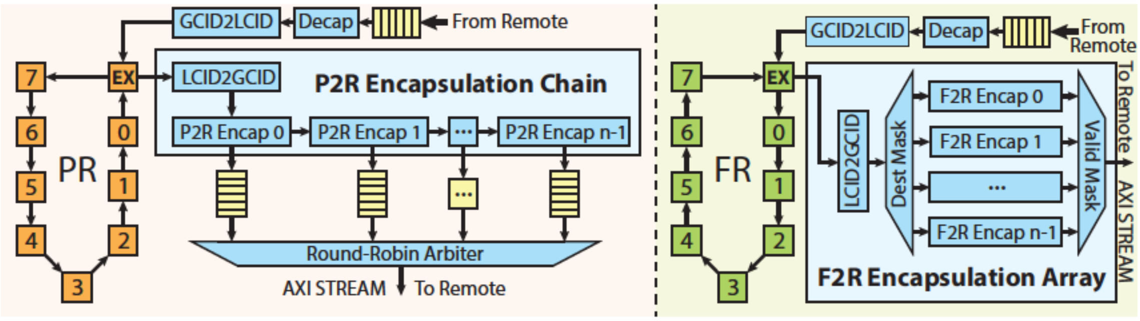


Figure 7·4: Arriving/departing data processing. P2R/F2R: position/force to remote.

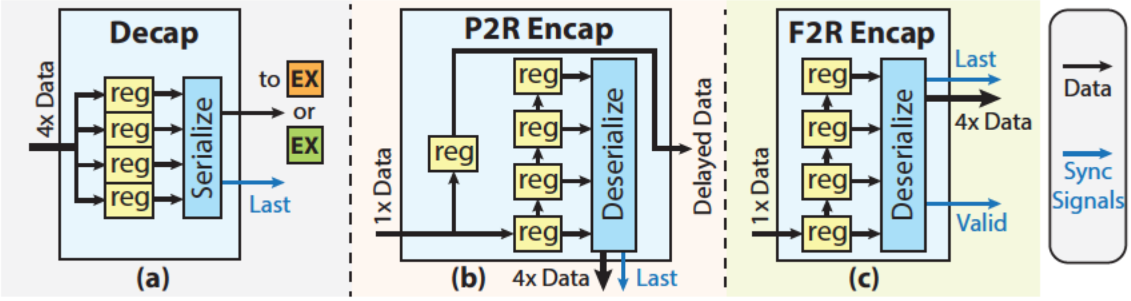


Figure 7·5: Details of data processing sub-modules.

destination cell. The destination cell has GCID (3, 0) but appears as (0, 0) in its local node.

Furthermore, after a particle reaches a destination CBB, its LCID is converted to RCID and then concatenated with the fixed-point position for easy distance calculation by direct subtraction. RCID is a range from 1 to 3 in a direction, with a particle in its local cell assigned an RCID of (2, 2, 2) in 3-D. This is because the leading “1” in the fixed-point needs to be located for fixed-to-float conversion for further force evaluation. Therefore, starting from 1 simplifies the conversion process. The use of fixed-point positions is motivated by the reduction in hardware resource cost for filters, which can number in the hundreds in our design.

7.2.3 Communication Interface

The processing of data upon arrival or departure is illustrated in Figure 7·4. Upon arrival, a 512-bit AXI-Stream position packet or force packet that contains four pieces of data is received and decapsulated into separate data pieces with headers that contain particle identification information (as shown in Figure 7·5(a)). Afterward, the data is serialized to the EX node on either a position ring or a force ring, where it is injected into the ring for further processing.

The encapsulation process differs for positions and forces, as positions may have multiple destinations while forces have a unique destination. To encapsulate a position packet, an encapsulation chain is utilized to reuse the position data and reduce fan-out. A P2R encapsulator can be analogously regarded as a departure gate. The position data is passed through a series of *n* P2R encapsulators to find its departure gates, where *n* is the number of neighboring FPGAs to which a local position may be transmitted. Four positions are wrapped up as a position packet, and buffered once all four registers are filled as shown in figure 7·5(b), and then arbitrated for departure.

Since each force only has one destination, there is at most one force packet departure in a cycle without the need for an arbiter. Thus, a force is directed to its departure gate using a destination mask and is sent out once the valid signal becomes high after all four registers are filled, as shown in Figure 7·5(c).

Both position and force packets contain a "last" signal for synchronization, which is activated after all data associated with a destination node has been processed. The chained synchronization mechanism is explained in the subsequent part of the chapter.

7.2.4 Chained Synchronization

Distributed spatial simulations often utilize the Bulk Synchronous Parallel (BSP) model, which can be problematic due to its centralized nature and susceptibility to the straggler problem [Bin Khunayn et al., 2017], where the delay of a single worker slows down the entire system. To address these issues, instead of relying on traditional bulk synchronization methods, we leverage the advantages of our MD architecture by analyzing its data dependency pattern and implementing a chained synchronization approach. This technique

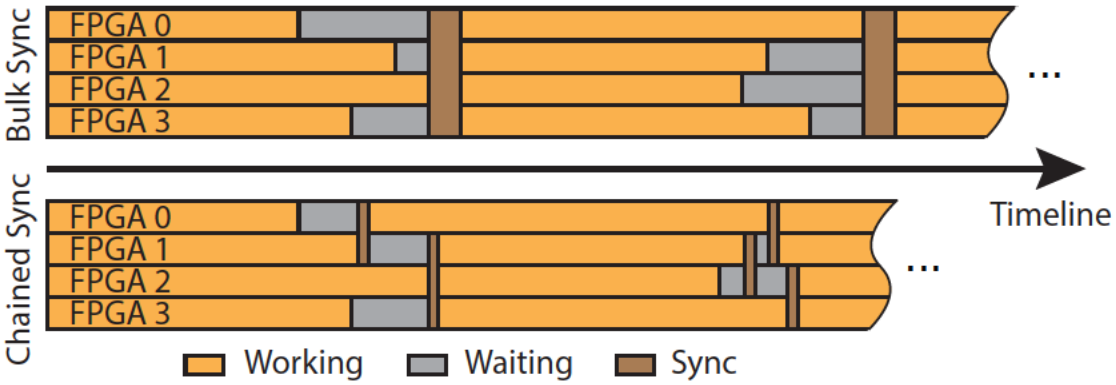


Figure 7·6: Two synchronization methods for 4 FPGAs. In this example, each FPGA only communicates with its neighbors.

maintains a fully-distributed and scalable system while mitigating the impact of stragglers on performance.

Figure 7·6 illustrates a comparison between the conventional bulk global synchronization and our proposed chained synchronization method. Bulk synchronization can be performed using a host or a central FPGA, but using a host can result in latency of milliseconds for a single MD iteration, while using a central FPGA can destroy the decentralized and homogeneous nature of the system, as well as introduce additional latency due to routing. Additionally, the number of valid particle pairs for each FPGA to evaluate can be hard to predict and is affected by the straggler problem. In contrast, our approach is to use localized, fine-grained synchronization among FPGAs as depicted in the figure. Each FPGA only synchronizes with its immediate neighbors in a chain-reaction manner. Although the chained synchronization method still relies on the slowest board to finish, it decouples the distant FPGA nodes from the slow one, providing them with a head start into the next iteration while preserving a decentralized and homogeneous system.

Figure 7·7 demonstrates the operation of the chained synchronization method, with the motion update phase excluded for brevity. At the beginning, node 0 and node 1 send position data to each other. After some time, FPGA 1 transmits a “last position” signal along with the data packet after all positions have been routed by its local PR. Node 0 receives

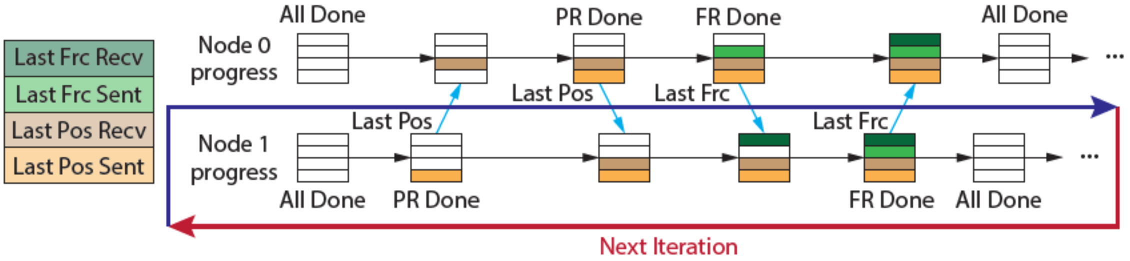


Figure 7·7: The behavior of the chained synchronization between two FPGA nodes as an example. Frc: force. Pos: position.

the signal and recognizes that a“last force” signal can be returned as a feedback to the “last position” signal only after processing all positions from node 1. This process is mutual, such that node 0 also sends “last position” to node 1, and receives “last force” from node 1. When all four criteria are fulfilled, a node can proceed to the motion update phase independently, and the motion update phase follows a similar but simplified synchronization pattern, only to receive/transmit one last signal from/to another node instead of two.

When an FPGA node needs to synchronize with multiple neighboring FPGAs, the synchronization pattern is unchanged, but counters are added to keep track of the number of “last position” and “last force” signals received. A criterion is met when the number of “last” signals sent or received equals the number of neighboring nodes.

7.2.5 PE Scaling

Thus far, we have presented a system that includes both computation and communication components in a weak-scaling manner. In order to further improve the scalability to strong-scaling, we take advantage of the re-configurable platform and trade the number of CBBs for a greater number of PEs with a certain amount of hardware resources on-chip.

By reducing the number of CBBs, an FPGA now is responsible for a smaller portion of a simulation space. The size of the rings is also reduced, which in turn reduces the routing latency on the rings. This results in less congestion on the rings, allowing us to allocate more resources directly to the PEs without worrying too much about the bandwidth on the

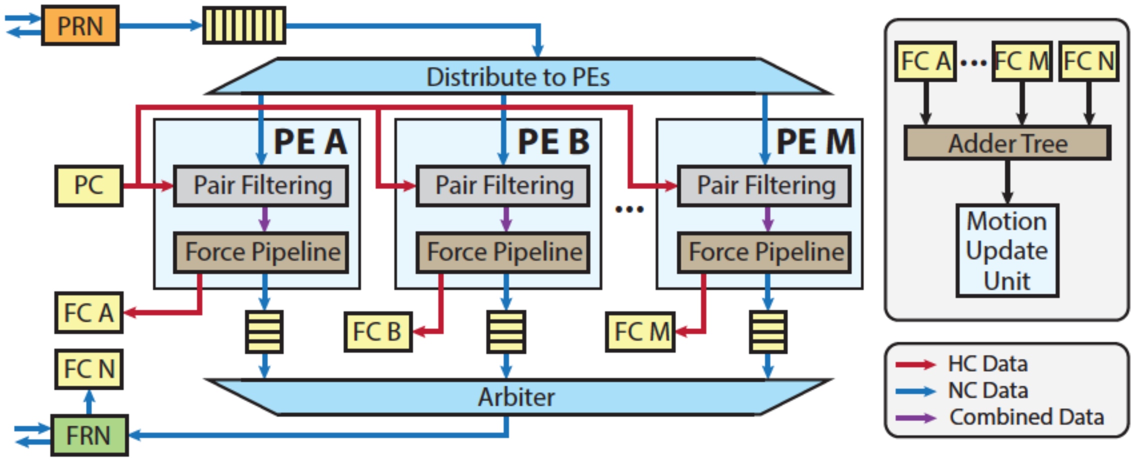


Figure 7·8: The architecture of a CBB with multiple PEs.

data path. To enhance the performance of an FPGA on the reduced dataset in order to achieve strong-scaling, we employ multiple PEs to process a single cell, as opposed to the previous CBB architecture where one PE was used for each cell.

Figure 7·8 displays the updated architecture. The internal structure of a PE remains unchanged, while the data input and output patterns are evolved. We recognize that the PR has lower utilization compared to a PE or FR due to the exceptional data locality of position, as a single position is broadcast to many CCBs for processing. Besides, even though many neighboring positions can be processed in parallel in a PE, each position still requires over 100 cycles of processing before the next one can be processed, granting the position ring ample routing time. Thus, a single PR is sufficient to provide multiple PEs with neighbor position data. Meanwhile, since all PEs in the figure are processing the same cell, the home position can be broadcast to all PEs effortlessly.

Regarding force output, ideally, a force pipeline produces a pair of force fragments per cycle. The home force component in the force fragment pair is accumulated directly into a FC, requiring more FCs to meet the increasing demand for home force accumulation. The neighbor forces are first accumulated in the PE for locality, and then arbitrated and injected into the FR for routing. The local FRN feeds the neighbor forces to FC N, which

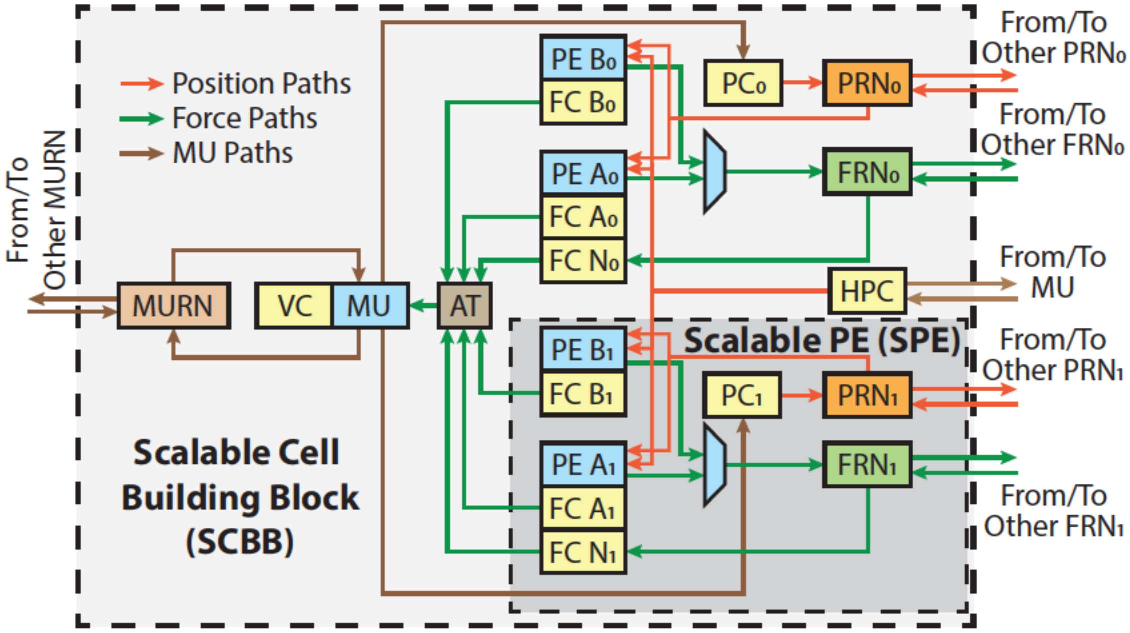


Figure 7·9: The architecture of an SCBB with 2 SPEs, where an SPE consists of 2 PEs for example. AT: Adder Tree. HPC: Home Position Cache.

is designated to store neighbor forces due to the increased throughput of the shortened FR.

Following the multiple FCs, force combination takes place. The forces in FCs are stored and indexed by particle ID to maintain memory alignment, facilitating easy force additions. The forces are not accumulated directly during force evaluation, but are accumulated during motion update to simplify the force data path. Upon requested by a MU, partial forces are combined in an adder tree to maintain a one-per-cycle throughput that matches the MU.

In addition to the benefit of having shorter routing rings, another advantage of scaling the processing elements (PEs) is that it only requires scaling the FCs with the number of PEs, rather than PCs or VCs. This approach provides more memory resources for other purposes.

7.2.6 CBB Scaling

At a certain point, adding more PEs will not enhance performance anymore because the performance is limited by the bandwidth of the routing rings or the particle broadcast from

PCs. Hence, we propose a “stronger”-scaling approach, which aims to scale the CBB for higher throughput.

In Figure 7·9, we present the architecture of a Scalable CBB that consists of two SPEs, where each SPE is grouped by n PEs, $n+1$ FCs, a PC, a PRN, and a FRN in the way described in PE Scaling section. Since the bottlenecks originate from the routing rings and PCs, such grouping method restricts the bottlenecks within an SPE, and allows higher performance being achieved by populating SPEs. Additional SPEs can be attached to the highlighted SPE to increase throughput as shown in the figure. The second SPE uses a separate set of routing paths to ensure equally short rings. Since both SPEs work on the same cell within a single SCBB, the force results can be combined directly in an adder tree. The VC, MU, and MU routing path do not scale with the SCBB as MU takes much less time compared to force evaluation. The number of EX nodes on rings are scaled as well, with their inputs dispatched and outputs arbitrated.

To prevent the transmission of duplicated neighbor positions to PRNs, the positions of all home cells are divided into two disjoint subsets and stored in PC_0 and PC_1. These PCs are now used only for neighbor position broadcast, and a single separate home position cache (HPC) is responsible for home position traversal. During the local MU update, HPC, PC_0, and PC_1 are all updated simultaneously. It’s worth noting that PC_0 only takes positions with even particle ID, while PC_1 only takes odd ones. If more than 2 SPEs are instantiated, they only need to work on particles with interleaved IDs to ensure a balanced workload. Meanwhile, only the HPC is used to update local position, as it contains all the position information for the cell, while the other PCs only hold partial information.

7.3 Evaluation

7.3.1 Experimental Setup

The FASDA system was developed using Verilog/SystemVerilog HDL, implemented and validated on the New England Research Cloud (NERC) using Xilinx Vitis and Vivado 2021.2. Our experiment was carried out on the Open Cloud Testbed [Handagala et al., 2022], where we used up to 8× Xilinx UltraScale+ Alveo U280 FPGA boards running at 200 MHz with both QSFP28 ports connected to a Dell Z9100-ON 100 GbE Switch, with UDP protocol established. Each FPGA contains 1303K CLBs, 2607K flip-flops, 2016× 36-Kb Block RAMs (BRAM), 960× 288-Kb Ultra RAMs (URAM), and 9024× DSP slices. For comparison, we also run OpenMM, one of the state-of-the-art MD software packages on high-end CPUs and GPUs, including an Intel Xeon Gold 6226R processor with up to 32× threads, as well as up to 2× Nvidia A100 GPUs connected via NVLink and up to 4× Nvidia V100 GPUs all-to-all connected with NVLinks. To ensure fairness, we ran OpenMM only with the LJ force field.

In order to ensure the generality of our results, we utilized a custom dataset that involves the initialization of 64 randomly distributed sodium particles in each cell, while ensuring that none of the particles are too close to be excluded. In both FASDA and OpenMM simulations, a cutoff radius of 8.5 Å was used, along with a simulation time step of 2 fs (2^{-15} seconds).

7.3.2 Overall Performance

Figure 7·10 and 7·11 display the assessed simulation rate of different platforms in terms of µs/day, which refers to the number of microseconds of simulation that can be executed in a day. The former shows weak-scaling while the latter shows strong-scaling capabilities of FPGAs, GPUs, and CPUs. The assessment is carried out on five simulation spaces, with the first four spaces scaled up by 3×3×3 cell blocks to demonstrate weak-scalability, while

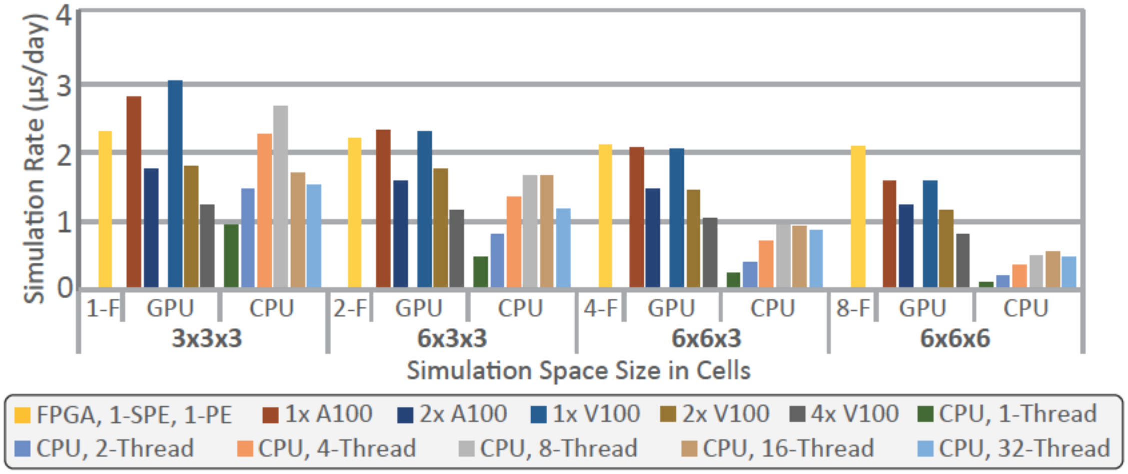


Figure 7·10: The weak scalability comparison. 1-F: 1× FPGA board. 1-SPE: Each SCBB contains 1× SPE. 1-PE: Each SPE contains 1 PE.

the last 4×4×4 simulation space is used to exhibit the strong-scalability characterizations.

In the case of weak scaling demonstrated by the first four configurations, it is evident that both GPUs and CPUs experience performance loss at a larger number of computing nodes. For example, doubling the number of GPUs does not preserve the simulation rate for a doubled workload; instead, it only provides half the simulation rate. As for CPUs, they exhibit competitive performance for smaller space sizes and good weak-scaling up to four threads, but performance degrades for larger numbers of threads. In contrast, the simulation rate of FPGAs remains consistent at around 2 µs/day for all four configurations, indicating their outstanding weak-scalability.

In terms of strong scaling, it is clear that the simulation rate of GPUs decreases as the number of GPUs increases, primarily due to the long communication latency and low efficiency for a small number of particles. CPUs scale well for up to 4 threads, but suffer from significant overhead for more than 8 threads and eventually result in negative scaling for 16 threads and beyond. On the other hand, FPGAs provide strong scalability through the use of SCBBs and SPEs. In the case of a 4×4×4 simulation space, where each FPGA is responsible for 2×2×2 cells, the performance is increased to 5.26× with 3 PEs per SPE

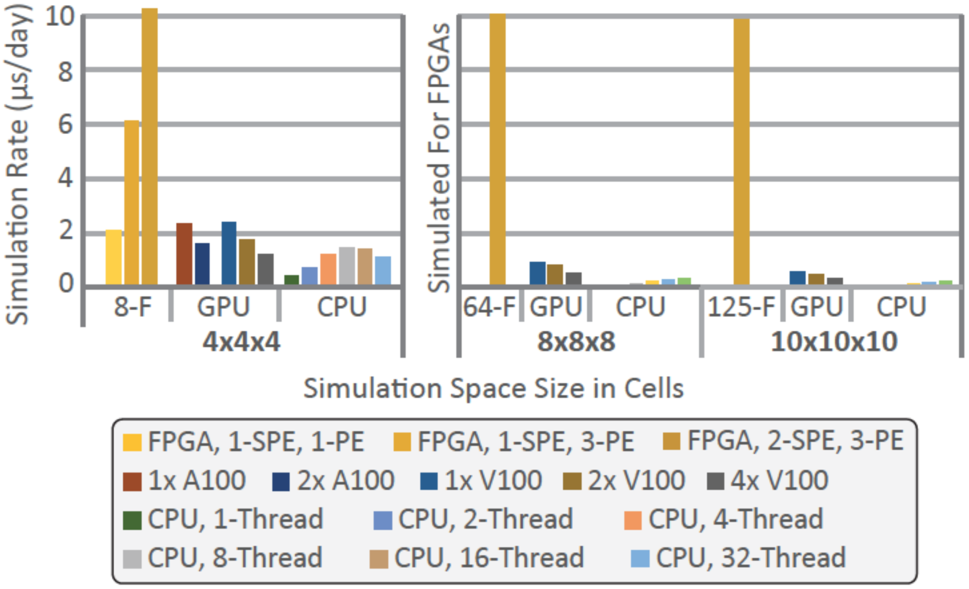


Figure 7·11: The strong scalability comparison. 1-F: 1× FPGA board. 1-SPE: Each SCBB contains 1× SPE. 1-PE: Each SPE contains 1 PE.

and 2 SPEs per SCBB compared to 1 PE per cell. In contrast, 2 GPUs and 4 GPUs result in 26% and 49% performance loss respectively compared to 1 GPU. With the negative strong-scaling of GPUs, it is fair compare the FPGA results with the best GPU result and obtain a 4.67× speedup in performance.

The right section of Figure 7·11 displays the measured results for GPUs and CPUs, as well as the simulated results for FPGAs. The efficiency of a single GPU increases as the workload grows, as its performance only drops by 60% when transitioning from 4×4×4 to 8×8×8 cells. We then observe that the GPU's efficiency peaks at 8×8×8 cells, as its performance is halved for 10×10×10 cells, matching the scaling of the workload. Even for 10×10×10 cells (64K particles), GPUs still demonstrate negative strong-scaling due to their frequent synchronization with long latency and the reduced efficiency when mapped with a smaller portion of the particles. The FPGA simulation assumes communication latency between two FPGAs remains the same as with 8 boards, and is performed for 64 and 125 FPGAs respectively with each working on 2×2×2 cells, providing an intuitive insight into the system.

For the rest of the evaluations, each FPGA only works on 3×3×3 or 2×2×2 cells, and the number of FPGAs is scaled with respect to the simulation space configurations without further notice.

7.3.3 Utilization Breakdown

The utilization of critical components in the design, including PRs, FRs, filters, PEs, and MUs, is illustrated in Figure 7·12. The figure provides a detailed breakdown of hardware and time utilization for all design variations. Hardware utilization refers to the average amount of work performed by a component in comparison to its capacity, while time utilization represents the average proportion of time that a component is active, during which the pipeline may not be full, but is functioning. The former indicates the effectiveness of the components, while the latter provides a understanding of the system's overall operation flow.

It is observed that the PEs remain active for about 80% of the total operating time, with a hardware utilization of approximately 50%∼60% across all the design variations. This suggests that the FPGA design satisfies a critical requirement for strong-scaling, whereby the computing units retain efficiency even when a larger number of computing units are utilized with each working on a smaller number of particles. Furthermore, the upstream filters match the PEs well in terms of hardware utilization, indicating that the number of filters (6 in our experiments) matches the PE throughput that generates one force per cycle.

The utilization of the PR and FR routing components is also presented. It is noteworthy that in weak scaling scenarios (the first four configurations of the simulation space), both the hardware and time utilizations of PR increase. This is because the position data is fragmented and needs to be sent to multiple nodes, weakening the data locality and resulting in more data fragments spinning in rings. The FR hardware utilization also increases with more nodes involved, which extends the routing path for forces. For strong scaling, the utilizations of both PR and FR increase from 4×4×4-A to 4×4×4-B due to the populated PE in an SPE. However, the utilizations remain almost the same from 4×4×4-B to 4×4×4-C. This is because doubling the SPEs involves doubling the routing rings, splitting workload in two halves and almost doubling the performance. In general, the system is relatively well-balanced for all the design varieties, only with the PR underused due to the outstanding locality of position data.

Last but not not least, the MU only occupies the least of the overall utilization which is less than 5%, leaving the majority of the computing power focusing on the most complex tasks, i.e., force evaluation.

7.3.4 Communication Intensity

Figure 7·13 and 7·14 provide the communication status of the multi-FPGA implementations, demonstrating the communication requirement and intensity between distinct FPGA nodes. In our experiment, we utilized separate QSFP28 ports for position and force communication. Figure 7·13 reveals that even with the strong-scaling configuration of 2-SPEs and 3-PEs, the average bandwidth demand for an FPGA is below 25 Gbps for either position or force, which is well below the available 100 Gbps bandwidth. However, peaks in communication intensity could potentially overwhelm the routing device such as a switch, causing packet loss, and therefore we limit the transmission of each board to once per several cycles using cooldown counters, effectively spreading out a peak over a period of time. As communication and computation are executed simultaneously, with computation typically much more intense than communication, the latency loss in communication caused by cooldown is concealed.

Figure 7·14 displays the breakdown of position and force communication intensity in percentage with respect to other FPGA nodes. We assume that the FPGA nodes are logically connected as previously illustrated in figure 7·2. In reality, an FPGA only communicates intensely with the nodes logically close to it, particularly for forces. This is

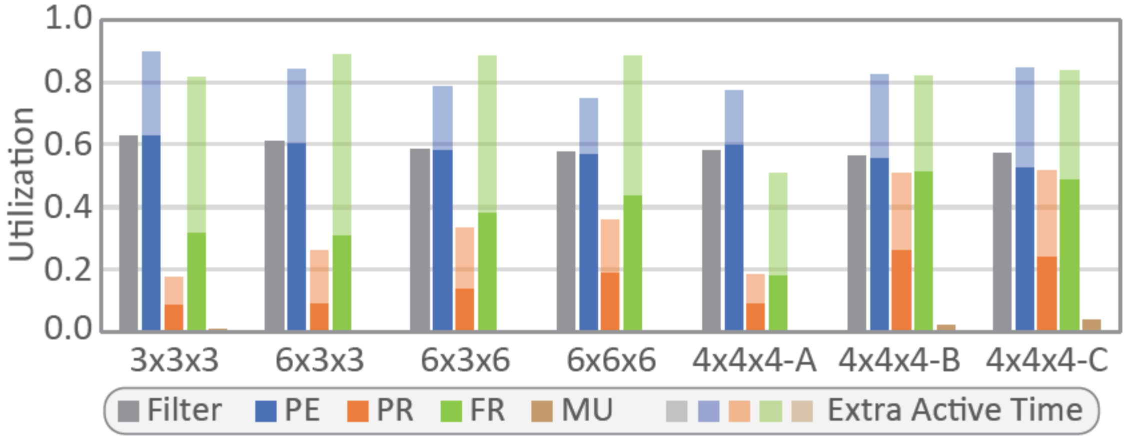


Figure 7·12: The utilization of key components for the design varieties. A: 1-SPE, 1-PE. B: 1-SPE, 3-PE. C: 2-SPE, 3-PE.

because particles sent from node 0 to node 7 sometimes do not pass through any filter at all, since node 7 is located at the corner of node 0. This occurrence significantly reduces the bandwidth demand for force communication, as zero force is simply discarded rather than being returned. This communication characteristic also provides an opportunity for the use of hyper-ring routing topology, as the bandwidth demand for routing data to more distant nodes diminishes, compensating for the shortcomings of hyper-rings due to the lack of communication links.

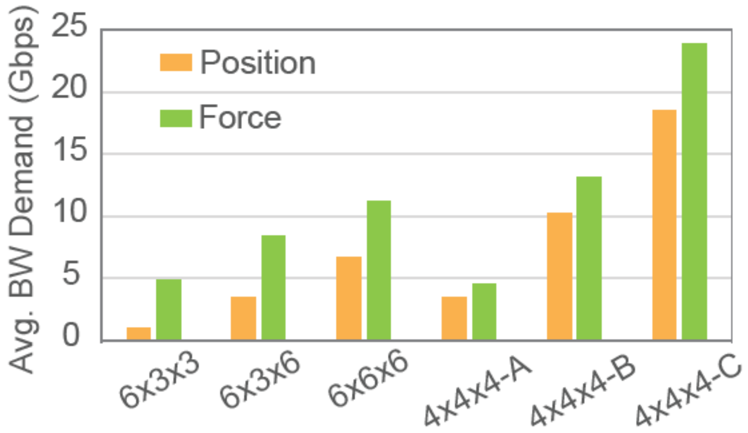


Figure 7·13: The communication bandwidth demand for different design configurations.

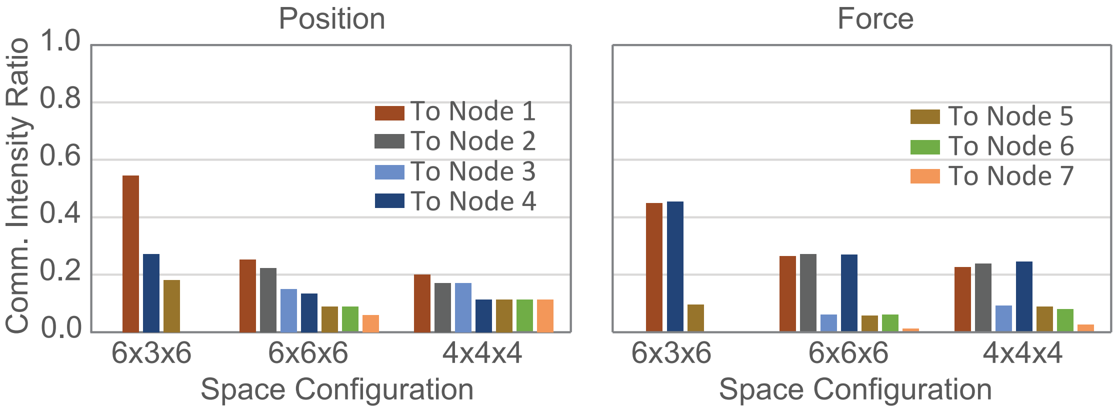


Figure 7·14: The communication bandwidth demand breakdown for different design configurations.

Table 7.1: Hardware Utilization of All Design Variations

Design	# FPGA	LUT	FF	BRAM	URAM	DSP
3×3×3	1	40%	22%	29%	20%	20%
6×3×3	2	44%	24%	38%	31%	20%
6×6×3	4	46%	24%	33%	42%	20%
6×6×6	8	46%	24%	33%	42%	20%
4×4×4-A	8	23%	16%	31%	13%	6%
4×4×4-B	8	35%	20%	51%	18%	14%
4×4×4-C	8	52%	26%	76%	28%	27%

A : 1-SPE, 1-PE. B: 1-SPE, 3-PE. C: 2-SPE, 3-PE.

7.3.5 Resources Consumption

Table 7.1 presents the resource consumption for all the implementation variations discussed earlier as a reference, including three configurations for the 4×4×4 simulation space. In the weak-scaling scenario, the cost of LUTs and FFs remains stable, while the memory cost increases significantly, especially from 3×3×3 to 6×3×3. This is due to the significant change in design required to handle and process data from remote nodes. To achieve a balanced resource consumption, LUT, BRAM, and URAM can be traded among each other to a certain extent.

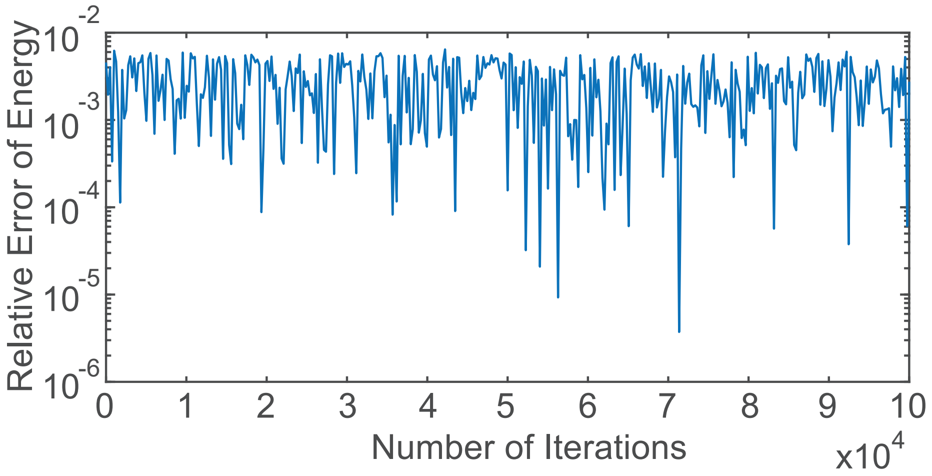


Figure 7·15: Energy Relative error with respect to OpenMM.

7.3.6 Energy Conservation

In order to ensure the stability of the physical simulation system, we evaluated the energy convergence status of FASDA by comparing it with a 64-bit double precision floating-point simulation that was run using OpenMM for 100,000 iterations on the 4×4×4 simulation space. Our observations from figure 7·15 indicate that the relative error generally falls around 10^{-4} and 10^{-3} with the overall energy converged.

7.4 Related Work

Most MD simulations are typically performed using software packages on CPUs and GPUs, which have a large user base. Some popular software that support GPUs are: AMBER [Case et al., 2005], Desmond [Bowers et al., 2006a], GROMACS [Abraham et al., 2015], LAMMPS [Thompson et al., 2022], NAMD [Phillips et al., 2005], and OpenMM [Eastman and Pande, 2010]. These packages offer similar main functionalities and come with various user-customizable options, such as Monte Carlo simulations and implicit water environment.

Apart from software packages for general-purpose machines, dedicated ASIC architectures have been designed by researchers to speed up MD. The Anton series [Shaw et al., 2007, Shaw et al., 2014, Shaw et al., 2021] have been the foundation of ASIC-based MD accelerator, providing an approximately tenfold performance increase with each successive generation. The MDGRAPE series [Fukushige et al., 1996, Susukita et al., 2003, Narumi et al., 2006, Ohmura et al., 2014], which dates back to the 1990s, has evolved from four chips with a total peak performance of 4.2 GFLOPs to today's 512 SoCs with 2.5 TFLOPs per chip. Its most recent upgrade, MDGRAPE-4A [Morimoto et al., 2021] offloads the 3D-FFT based convolution to Intel Arria 10 FPGAs, indicating that FPGAs are valuable and promising devices in MD acceleration.

Sensing the potential in FPGAs, researchers also develop MD accelerators on FPGAs. Targeting N-body simulations in high performance computing, researchers implement a cosmological-like (without cutoff) MD accelerator on multiple Intel Stratix 10 FPGAs with OpenCL to demonstrate the advantages of strong scaling on FPGAs [Menzel et al., 2021]. ARUZ, consisting of ∼26,000 FPGAs including Xilinx Artix and Zynq FPGAs, is a massive FPGA cluster specifically built for MD as described in [Kiełbik et al., 2018]. Unlike classical MD methods used in drug design, dynamic lattice liquid (DLL) method is adopted in ARUZ, which requires global synchronization. A variety of single-FPGA MD solutions are conveyed as well. In [Yang et al., 2019a], the idea of a complete MD system integrated onto a single chip is outlined, presenting several design options to accommodate various scenarios. In [Yuan et al., 2022], an MD accelerator designed for developing semiconductor materials is deployed on a Xilinx Alveo U200 FPGA, outperforming an Nvidia RTX 2080 Ti by 20%.

7.5 Conclusion

The aim of this chapter is to present FASDA, an open-sourced hardware acceleration system for range-limited molecular dynamics based on FPGAs. FASDA is fully distributed and capable of significant strong-scaling with a small number of particles.

To achieve this distributed capability, we utilize two techniques: (1) a two-level cell ID conversion that eliminates the need for dynamic ID computing and creates a homogeneous environment for FPGA nodes and PEs, and (2) a chained synchronization technique that enables global synchronization through local synchronizations in a chain-reaction manner, without requiring central FPGA nodes or host control.

To achieve strong-scaling, we have scaled an original cell building block into a scalable cell building block which includes scalable PEs that contain several original PEs. The structured design has relatively well-balanced key components with high hardware utilization, enabling users to easily parameterize the design to meet their resource or performance requirements.

In comparison with GPUs, FASDA achieves a 4.67× speedup compared to the state-of-the-art GPU solution in MD, demonstrating the outstanding strong-scalability of our FPGA-based solution.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

To reiterate, our overall goals are two-fold: **maximize the potential of a single FPGA**, and **constructing a high-performance multi-FPGA system**. The two goals are achieved in a progressive manner.

Firstly, we establish a single chip design that handles 3-D to 2-D mapping based on the load partitioning schemes, leverages data locality for data reuse, and utilizes an efficient on-chip routing mechanism to avoid high fan-in-fan-out and frequency degradation.

Secondly, based on the single chip designs, we focus on the communication problem on two aspects: data transfer balancing for FPGA clusters of 3-D torus topology, and the reduction in communication intensity brought by an alternative interpretation of force symmetry.

Furthermore, the emerging FPGA-on-the-cloud allows us to actually prototype our MD RL system on an FPGA cluster. In our case, 8× Xilinx U280 FPGAs on an Ethernet switch are employed to fulfill our goal, achieving both strong scaling and higher performance compared to very high-end GPUs. Specifically, 4.67× speedup is achieved against an Nvidia V100 GPU.

From a high-level perspective, we exploit the emerging FPGA cluster platforms to deploy and execute our designs. By conducting thorough experimentation and analysis, we present convincing evidence that FPGA clusters not only hold their own against other computing technologies but also exhibit superior performance and scalability, particularly in the realms of high-performance computing and strong scaling.

8.2 Future Work

The work we have accomplished thus far is not yet an end-to-end MD system that delivers high performance in industry-level drug development. The missing components to integrate into the system are:

- Long range force computing that takes electrostatic forces into account, enabling the dynamics of non-electric-neutral particles.
- Bonded force computing that involves, for instance, chemical bond and hydrogen bond. For micro-second level molecular simulations, the bonds are considered as solid constraints which groups atoms into clusters.

Despite the separate efforts in previous works (Xiong et al., 2017; Sheng et al., 2017; Sanaullah et al., 2016a, 2016b), there remains a lack of comprehensive integration among the three components. To bridge the gap between academic prototypes and industry requirements, our next objective is to develop a well-designed floor plan that effectively balances resource utilization and operational time for all three components. This will enable us to achieve optimized overall performance in the integrated system.

Furthermore, there is a lack of a comprehensive compilation system that assists users in parameterizing their inputs, enhancing the user experience. This feature is of utmost importance since FPGA designs typically involve a higher learning overhead and steeper learning curve compared to GPU implementations. The need for a compiler that enables users to leverage the scalability and high performance offered by customizable hardware is therefore essential and highly anticipated.

In the realm of research, the field of AI-aided scientific computing has experienced rapid growth, facilitated by the advancements in machine learning. This trend presents promising opportunities for the integration of AI techniques into molecular dynamics (MD). One such example is the adoption of Graph Neural Network (GNN) approaches to accelerate solid-state molecular simulations, particularly for phase transitions.

In the context of classical MD, there exist possibilities to leverage the capabilities of generative models such as Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and stable diffusion models. These models can be employed to generate the atomic states based on the movements of a small subset of atoms, leading to improved performance. Additionally, trained AI models can be utilized to achieve higher accuracy by leveraging MD computations with reduced precision, such as simulating molecules using only 8-bit precision.

References

Amazon EC2 F1 instances. https://aws.amazon.com/ec2/instance-types/f1/. Accessed: 2023-03-20.

FAbRIC (FPGA research infrastructure cloud). https://wikis.utexas.edu/display/fabric/Home. Accessed: 2023-03-21.

Abraham, M. J., Murtola, T., Schulz, R., Páll, S., Smith, J. C., Hess, B., and Lindahl, E. (2015). GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, 1-2:19–25.

Alam, S., Agarwal, P., Smith, M., Vetter, J., and Caliga, D. (2007). Using FPGA devices to accelerate biomolecular simulations. *Computer*, 40(3):66–73.

Aminpour, M., Montemagno, C., and Tuszynski, J. A. (2019). An overview of molecular modeling for drug discovery with specific illustrative examples of applications. *Molecules*, 24(9):1693.

Azizi, N., Kuon, I., Egier, A., Darabiha, A., and Chow, P. (2004). Reconfigurable molecular dynamics simulator. In *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines*, pages 197–206.

Bandara, S., Sanaullah, A., Tahir, Z., Drepper, U., and Herbordt, M. (2022). Enabling VirtIO Driver Support on FPGAs. In *8th International Workshop on Heterogeneous High Performance Reconfigurable Computing*. doi: 10.1109/H2RC56700.2022.00006.

Benkrid, K. and Vanderbauwhede, W., editors (2013). *High Performance Computing Using FPGAs*. Springer Verlag. doi: 10.1007/978-1-4614-1791-0_4.

Bin Khunayn, E., Karunasekera, S., Xie, H., and Ramamohanarao, K. (2017). Exploiting data dependency to mitigate stragglers in distributed spatial simulation. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '17, New York, NY, USA. Association for Computing Machinery.

Boku, T., Kobayashi, R., Fujita, N., Amano, H., Sano, K., Hanawa, T., and Yamaguchi, Y. (2019). Cygnus: GPU meets FPGA for HPC. In *International Conference on Supercomputing*. https://www.r-ccs.riken.jp/labs/lpnctrt/assets/img/ lspanc2020jan_boku_light .pdf.

Bolaria, J. and Byrne, J. (2009). *A Guide to FPGAs for Communications*. The Linley Group.

Bowers, K., Dror, R., and Shaw, D. (2007). Zonal methods for the parallel execution of range-limited n-body simulations. *Journal of Computational Physics*, 221(1):303–329.

Bowers, K. J., Chow, D. E., Xu, H., Dror, R. O., Eastwood, M. P., Gregersen, B. A., Klepeis, J. L., Kolossvary, I., Moraes, M. A., Sacerdoti, F. D., Salmon, J. K., Shan, Y., and Shaw, D. E. (2006a). Scalable algorithms for molecular dynamics simulations on commodity clusters. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, pages 43–43.

Bowers, K. J., Dror, R. O., and Shaw, D. E. (2006b). The midpoint method for parallelization of particle simulations. *The Journal of Chemical Physics*, 124(18):184109.

Brown, W., Wang, P., Plimpton, S., and Tharrington, A. (2011). Implementing molecular dynamics on hybrid high performance computers–short range forces. *Computer Physics Communications (CPC)*, 182(4):898–911.

Case, D., Cheatham III, T., Darden, T., Gohlke, H., Luo, R., Merz, Jr., K., Onufriev, A., Simmerling, C., Wang, B., and Woods, R. (2005). The Amber biomolecular simulation programs. *Journal of Computational Chemistry*, 26:1668–1688.

Caulfield, A. M., Chung, E. S., Putnam, A., Angepat, H., Fowers, J., Haselman, M., Heil, S., Humphrey, M., Kaur, P., Kim, J.-Y., Lo, D., Massengill, T., Ovtcharov, K., Papamichael, M., Woods, L., Lanka, S., Chiou, D., and Burger, D. (2016). A cloud-scale acceleration architecture. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13.

Chiu, M. and Herbordt, M. (2009). Efficient filtering for molecular dynamics simulations. In *2009 International Conference on Field Programmable Logic and Applications*. doi: 10.1109/ FPL15426.2009.

Chiu, M. and Herbordt, M. (2010a). Molecular dynamics simulations on high performance reconfigurable computing systems. *ACM Transactions on Reconfigurable Technology and Systems*, 3(4):1–37. doi: 10.1145/1862648.1862653.

Chiu, M. and Herbordt, M. (2010b). Towards production FPGA-accelerated molecular dynamics: Progress and challenges. In *2010 4th High Performance Reconfigurable Technology and Applications*. doi: 10.1109/HPRCTA.2010.5670800.

Chiu, M., Herbordt, M., and Langhammer, M. (2008). Performance potential of molecular dynamics simulations on high performance reconfigurable computing systems. In *2008 Second International Workshop on High-Performance Reconfigurable Computing Technology and Applications*. doi: 10.1109/ HPRCTA.2008.4745685.

Chiu, M., Khan, M., and Herbordt, M. (2011). Efficient calculation of pairwise nonbonded forces. In *2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*. doi: 10.1109/ FCCM.2011.34.

Cong, J., Fang, Z., Kianinejad, H., and Wei, P. (2016). Revisiting FPGA Acceleration of Molecular Dynamics Simulation with Dynamic Data Flow Behavior in High-Level Synthesis. *Computing Research Repository (CoRR) in arXiv*, abs/1611.04474.

Darden, T., York, D., and Pedersen, L. (1993). Particle Mesh Ewald: an $N\log(N)$ method for Ewald sums in large systems. *Journal of Chemical Physics*, 98:10089–10092.

Eastman, P. and Pande, V. (2010). OpenMM: A Hardware-Independent Framework for Molecular Simulations. *Computing in Science and Engineering*, 4:34–39.

Ebisuzaki, T., Makino, J., Fukushige, T., Taiji, M., Sugimoto, D., Ito, T., and Okumura, S. K. (1993). GRAPE project: an overview. *Publications of the Astronomical Society of Japan*, 45:269–278.

Eran, H., Zeno, L., Tork, M., Malka, G., and Silberstein, M. (2019). NICA: An Infrastructure for Inline Acceleration of Network Applications. In *USENIX Annual Technical Conference*.

Fukushige, T., Taiji, M., Makino, J., Ebisuzaki, T., and Sugimoto, D. (1996). A highly parallelized special purpose computer for many-body simulations with and arbitrary central force: MD-GRAPE. *The Astrophysical Journal*, 468:51–61.

Ganesan, A., Coote, M. L., and Barakat, K. (2017). Molecular dynamics-driven drug discovery: leaping forward with confidence. *Drug Discovery Today*, 22(2):249–269.

Geng, T., Li, A., Shi, R., Wu, C., Wang, T., Li, Y., Haghi, P., Tumeo, A., Che, S., Reinhardt, S., and Herbordt, M. (2020a). AWB-GCN: A Graph Convolutional Network Accelerator with Runtime Workload Rebalancing. In *53rd IEEE/ACM International Symposium on Microarchitecture (MICRO).*

Geng, T., Wang, T., Sanaullah, A., Yang, C., Patel, R., and Herbordt, M. (2018a). A framework for acceleration of CNN training on deeply-pipelined FPGA clusters with work and weight load balancing. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL 2018): 394–402.* doi: 10.1109/ FPL.2018. 00074.

Geng, T., Wang, T., Sanaullah, A., Yang, C., Xu, R., Patel, R., and Herbordt, M. (2018b). FPDeep: Acceleration and Load Balancing of CNN Training on FPGA Clusters. In *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, page 81–84. doi: 10.1109/ FCCM.2018. 00021.

Geng, T., Wang, T., Wu, C., Li, Y., Yang, C., Wu, W., Li, A., and Herbordt, M. (2021a). O3BNN-R: An Out-Of-Order Architecture for High-Performance and Regularized BNN Inference. *IEEE Transactions on Parallel and Distributed Systems*, 32(1):199–213. doi: 10.1109/TPDS.2020.3013637.

Geng, T., Wang, T., Wu, C., Yang, C., Li, A., Song, S., and Herbordt, M. (2019a). LP-BNN: Ultra-low-Latency BNN Inference with Layer Parallelism. In *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, volume 2160, pages 9–16. doi: 10.1109/ASAP.2019.00-43.

Geng, T., Wang, T., Wu, C., Yang, C., Wu, W., Li, A., and Herbordt, M. (2019b). O3BNN: An Out-Of-Order Architecture for High-Performance Binarized Neural Network Inference with Fine-Grained Pruning. In *ACM International Conference on Supercomputing*, volume 2160, pages 461–472. doi: 10.1145/ 3330345. 3330386.

Geng, T., Wu, C., Tan, C., Fang, B., Li, A., and Herbordt, M. (2020b). CQNN: a CGRA-based QNN Framework. In *IEEE High Performance Extreme Computing Conference*. doi: 10.1109/HPEC43674.2020.9286194.

Geng, T., Wu, C., Tan, C., Xie, C., Guo, A., Haghi, P., He, S., Li, J., Herbordt, M., and Li, A. (2021b). A Survey: Handling Irregularities in Neural Network Acceleration with FPGAs. In *IEEE High Performance Extreme Computing Conference*. doi:10.1109/HPEC49654.2021.9622877.

Geng, T., Wu, C., Zhang, Y., Tan, C., Xie, C., You, H., Herbordt, M., Lin, Y., and Li, A. (2021c). I-GCN: A Graph Convolutional Network Accelerator with Runtime Locality Enhancement Through Islandization. In *54th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. doi:10.1145/3466752.3480113.

George, A., Herbordt, M., Lam, H., Lawande, A., Sheng, J., and Yang, C. (2016). Novo-G#: A Community Resource for Exploring Large-Scale Reconfigurable Computing Through Direct and Programmable Interconnects. In *2016 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA*, pages 1–7. doi: 10.1109/ HPEC.2016. 7761639.

Glaser, J., Nguyen, T. D., Anderson, J. A., Lui, P., Spiga, F., Millan, J. A., Morse, D. C., and Glotzer, S. C. (2015). Strong scaling of general-purpose molecular dynamics simulations on GPUs. *Computer Physics Communications*, 192:97 – 107.

Gokhale, M. and Graham, P. (2005). *Reconfigurable Computing: Accelerating Computation with Field Programmable Gate Arrays*. Springer.

Grossman, J., Towles, B., Greskamp, B., and Shaw, D. Filtering, reductions and synchronization in the Anton 2 network. In *2015 IEEE International Parallel and Distributed Processing Symposium*, Hyderabad, India, 2015, pp. 860-870, doi: 10.1109/IPDPS.2015.42.

Gu, Y. and Herbordt, M. (2007a). FPGA-based multigrid computations for molecular dynamics simulations. In *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 117–126. doi: 10.1109/ FCCM.2007.42.

Gu, Y. and Herbordt, M. (2007b). High performance molecular dynamics simulations with FPGA coprocessors. In *Reconfigurable Systems Summer Institute*.

Gu, Y., VanCourt, T., DiSabello, D., and Herbordt, M. (2005a). FPGA acceleration of molecular dynamics computations. In *IEEE Symposium on Field Programmable Custom Computing Machines*. DOI: 10.1109/FCCM.2005.54.

Gu, Y., VanCourt, T., and Herbordt, M. (2005b). Accelerating molecular dynamics simulations with configurable circuits. In *IEEE Conference on Field Programmable Logic and Applications*. DOI: 10.1109/FPL.2005.1515767.

Gu, Y., VanCourt, T., and Herbordt, M. (2006a). Accelerating molecular dynamics simulations with configurable circuits. *IEE Proceedings on Computers and Digital Technology*, 153(3):189–195. doi: 10.1049/ip-cdt:20050182.

Gu, Y., VanCourt, T., and Herbordt, M. (2006b). Improved interpolation and system integration for FPGA-based molecular dynamics simulations. In *2006 International Conference on Field Programmable Logic and Applications*, pages 21–28. doi: 10.1109/ FPL.2006.311190.

Gu, Y., VanCourt, T., and Herbordt, M. (2006c). Integrating FPGA acceleration into the ProtoMol molecular dynamics code: Preliminary report. In *2006 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 315–316. doi: 10.1109/ FCCM.2006.52.

Gu, Y., VanCourt, T., and Herbordt, M. (2008). Explicit design of FPGA-based coprocessors for short-range force computation in molecular dynamics simulations. *Parallel Computing*, 34(4-5):261–271. doi: 10.1016/j.parco.2008.01.007.

Guo, A., Geng, T., Zhang, Y., Haghi, P., Wu, C., Tan, C., Lin, Y., Li, A., and Herbordt, M. (2022a). A Framework for Neural Network Inference on FPGA-Centric SmartNICs. In *International Conference on Field-Programmable Logic and Applications*. DOI: 10.1109/FPL57034.2022.00071.

Guo, A., Geng, T., Zhang, Y., Haghi, P., Wu, C., Tan, C., Lin, Y., Li, A., and Herbordt, M. (2022b). FCsN: A FPGA-Centric SmartNIC Framework for Neural Networks. In *30th IEEE International Symposium on Field-Programmable Custom Computing Machines*. DOI: 10.1109/FCCM53951.2022.9786193.

Guo, A., Hao, Y., Wu, C., Haghi, P., Pan, Z., Si, M., Tao, D., Li, A., Herbordt, M., and Geng, T. (2023). Software-hardware co-design of heterogeneous SmartNIC system for recommendation models inference and training. In *ICS 2023: International Conference on Supercomputing*.

Haghi, P., Geng, T., Guo, A., Wang, T., and Herbordt, M. (2020a). FP-AMG: FPGA-Based Acceleration Framework for Algebraic Multigrid Solvers. In *28th IEEE International Symposium on Field-Programmable Custom Computing Machines*. DOI: 10.1109/ FCCM48280.2020.00028.

Haghi, P., Geng, T., Guo, A., Wang, T., and Herbordt, M. (2020b). Reconfigurable Compute-in-the-Network FPGA Assistant for High-Level Collective Support with Distributed Matrix Multiply Case Study. In *IEEE Conference on Field Programmable Technology*.

Haghi, P., Guo, A., Xiong, Q., Patel, R., Yang, C., Geng, T., Broaddus, J., Marshall, R., Skjellum, A., and Herbordt, M. (2020c). FPGAs in the Network and Novel Communicator Support Accelerate MPI Collectives. In *IEEE High Performance Extreme Computing Conference*.

Haghi, P., Guo, A., Xiong, Q., Yang, C., Geng, T., Broaddus, J., Marshall, R., Schafer, D., Skjellum, A., and Herbordt, M. (2022). Reconfigurable switches for high performance and flexible MPI collectives. *Concurrency and Computation: Practice and Experience*, 34(2). doi: 10.1002/cpe.6769.

Haghi, P., Krska, W., Tan, C., Geng, T., Chen, P., Greenwood, C., Guo, A., Hines, T., Wu, C., Li, A., Skjellum, A., and Herbordt, M. (2023). FLASH: FPGA-accelerated smart switches with GCN case study. In *ICS 2023: International Conference on Supercomputing*.

Hamada, T. and Nakasato, N. (2005). Massively parallel processors generator for reconfigurable system. *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines*.

Handagala, S., Leeser, M., Patle, K., and Zink, M. (2022). Network attached FPGAs in the open cloud testbed (OCT). In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6.

Hauck, S. and DeHon, A. (2008). *Reconfigurable Computing: The Theory and Practice of FPGA-Based Computing*. Morgan Kaufmann.

Herbordt, M. (2013). Architecture/algorithm codesign of molecular dynamics processors. In *2013 Asilomar Conference on Signals, Systems, and Computers*, pages 1442–1446. doi: 10.1109/ ACSSC.2013.6810534.

Herbordt, M. (2019). Advancing OpenCL for FPGAs: Boosting performance with Intel FPGA SDK for OpenCL technology. In *Parallel Universe Magazine*, 35:17-32.

Herbordt, M., Gu, Y., VanCourt, T., Model, J., Sukhwani, B., and Chiu, M. (2008a). Computing models for FPGA-based accelerators with case studies in molecular modeling. *Computing in Science and Engineering*, 10(6):35–45. doi: 10.1109/ MCSE.2008.143.

Herbordt, M., Khan, M., and Dean, T. (2009). Parallel discrete event simulation of molecular dynamics through event-based decomposition. In *In 2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors, Boston, MA*, pages 129–136. doi: 10.1109/ ASAP.2009.39.

Herbordt, M., Kosie, F., and Model, J. (2008b). An efficient $O(1)$ priority queue for large FPGA-based discrete event simulations of molecular dynamics. In *In 2008 16th International Symposium on Field-Programmable Custom Computing Machines*, pages 248–257. doi: 10.1109/ FCCM.2008.49.

Herbordt, M., Model, J., Sukhwani, B., Gu, Y., and VanCourt, T. (2006). Single pass, BLAST-like, approximate string matching on FPGAs. In *2006 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 217–226. doi: 10.1109/ FCCM.2006.64.

Herbordt, M., Model, J., Sukhwani, B., Gu, Y., and VanCourt, T. (2007a). Single pass streaming BLAST on FPGAs. *Parallel Computing*, 33(10-11):741–756.

Herbordt, M., VanCourt, T., Gu, Y., Sukhwani, B., Conti, A., Model, J., and DiSabello, D. (2007b). Achieving high performance with FPGA-based computing. *IEEE Computer*, 40(3):42–49.

Hilbert, D. (1952). *Geometry and the Imagination*. Chelsea Publishing Company, New York.

Humphries, B., Zhang, H., Sheng, J., Landaverde, R., and Herbordt, M. (2014). 3D FFT on a Single FPGA. In *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*. doi: 10.1109/ FCCM.2014.28.

Ito, T., Ebisuzaki, T., Makino, J., and Sugimoto, D. (1991). A Special-Purpose Computer for Gravitational Many-Body Systems: GRAPE-2. *Publications of the Astronomical Society of Japan*, 43:547–555.

Jamieson, P., Sanaullah, A., and Herbordt, M. (2018). Benchmarking Heterogeneous HPC Systems Including Reconfigurable Fabrics: Community Aspirations for Ideal Comparisons. In *IEEE High Performance Extreme Computing Conference*.

Jones, D., Allen, J. E., Yang, Y., Drew Bennett, W. F., Gokhale, M., Moshiri, N., and Rosing, T. S. (2022). Accelerators for classical molecular dynamics simulations of biomolecules. *Journal of Chemical Theory and Computation*, 18(7).

Keahey, K., Anderson, J., Zhen, Z., Riteau, P., Ruth, P., Stanzione, D., Cevik, M., Colleran, J., Gunawi, H. S., Hammock, C., Mambretti, J., Barnes, A., Halbach, F., Rocha, A., and Stubbs, J. (2020). Lessons learned from the Chameleon testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association.

Khan, M., Chiu, M., and Herbordt, M. (2013). FPGA-Accelerated Molecular Dynamics. In Benkrid, K. and Vanderbauwhede, W., editors, *High Performance Computing Using FPGAs*, pages 105–135. Springer Verlag. doi: 10.1007/978-1-4614-1791-0_4.

Khan, M. and Herbordt, M. (2011). Parallel discrete event simulation of molecular dynamics with speculation and in-order commitment. *Journal of Computational Physics*, 230(17):6563–6582. doi: 10.1016/j.jcp.2011.05.001.

Khan, M. and Herbordt, M. (2012). Communication requirements for FPGA-centric molecular dynamics. In *Symposium on Application Accelerators for High Performance Computing*. https:// www.bu.edu/ caadlab/saahpc12.pdf.

Kiełbik, R., Hałagan, K., Zatorski, W., Jung, J., Ulański, J., Napieralski, A., Rudnicki, K., Amrozik, P., Jabłoński, G., Stożek, D., Polanowski, P., Mudza, Z., Kupis, J., and Panek, P. (2018). ARUZ — large-scale, massively parallel FPGA-based analyzer of real complex systems. *Computer Physics Communications*, 232:22–34.

Kindratenko, V. and Pointer, D. (2006). A case study in porting a production scientific supercomputing application to a reconfigurable computer. In *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines*, pages 13–22.

Komeiji, Y., Uebayasi, M., Takata, R., Shimizu, A., Itsukashi, K., and Taiji, M. (1997). Fast and accurate molecular dynamics simulation of a protein using a special-purpose computer. *Journal of Computational Chemistry*, 18(12):1546–1563.

Larson, R., Salmon, J., Deneroff, M., Young, C., Grossman, J., Shan, Y., Klepseis, J., and Shaw, D. High-throughput pairwise point interactions in Anton, a specialized machine for molecular dynamics simulation. In *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, Salt Lake City, UT, USA, 2008, pp. 331-342, doi: 10.1109/HPCA.2008.4658650.

Lawande, A., George, A., and Lam, H. (2016). Novo-G#: a multidimensional torus-based reconfigurable cluster for molecular dynamics. *Concurrency and Computation: Practice and Experience*, 28(8).

Liu, X., Shi, D., Zhou, S., Liu, H., Liu, H., and Yao, X. (2018). Molecular dynamics simulations and novel drug discovery. *Expert Opinion on Drug Discovery*, 13(1):23–37. PMID: 29139324.

Makino, J. and Daisaka, H. (2012). GRAPE-8: An accelerator for gravitational n-body simulation with 20.5gflops/w performance. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '12, Washington, DC, USA. IEEE Computer Society Press.

Makino, J., Fukushige, T., Koga, M., and Namura, K. (2003). GRAPE-6: Massively-Parallel Special-Purpose Computer for Astrophysical Particle Simulations. *Publications of the Astronomical Society of Japan*, 55(6):1163–1187.

Makino, J., Taiji, M., Ebisuzaki, T., and Sugimoto, D. (1994). GRAPE 4: a one-tflops special-purpose computer for astrophysical n-body problem. In *Supercomputing '94:Proceedings of the 1994 ACM/IEEE Conference on Supercomputing*, pages 429–438.

Menzel, J., Plessl, C., and Kenter, T. (2021). The strong scaling advantage of FPGAs in HPC for n-body simulations. *ACM Transactions on Reconfigurable Technology and Systems*, 15(1).

Model, J. and Herbordt, M. (2007). Discrete event simulation of molecular dynamics with configurable logic. In *2007 International Conference on Field Programmable Logic and Applications*, pages 151–158. doi: 10.1109/FPL.2007.4380640.

Mondigo, A., Ueno, T., Sano, K., and Takizawa, H. (2020). Comparison of Direct and Indirect Networks for High-Performance FPGA Clusters. In Rincon, F., Barba, J., So, H., Diniz, P., and Caba, J., editors, *ARC 2020. Lecture Notes in Computer Science, vol 12083*. Springer. 10.1007/978-3-030-44534-8_24.

Morimoto, G., Koyama, Y. M., Zhang, H., Komatsu, T. S., Ohno, Y., Nishida, K., Ohmura, I., Koyama, H., and Taiji, M. (2021). Hardware acceleration of tensor-structured multilevel Ewald summation method on MDGRAPE-4A, a special-purpose computer system for molecular dynamics simulations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15.

Mortier, J., Rakers, C., Bermudez, M., Murgueitio, M. S., Riniker, S., and Wolber, G. (2015). The impact of molecular dynamics on drug design: applications for the characterization of ligand–macromolecule complexes. *Drug Discovery Today*, 20(6):686 – 702.

Mullard, A. (2014). New drugs cost us $2.6 billion to develop. *Nature reviews. Drug discovery*, 13(12):877.

Narumi, T., Ohno, Y., Okimoto, N., Suenaga, A., Yanai, R., and Taiji, M. (2006). A high-speed special-purpose computer for molecular dynamics simulations: MDGRAPE-3. In *NIC Workshop*, volume 34, pages 29–36. Citeseer.

Narumi, T., Susukita, R., Koishi, T., Yasuoka, K., Furusawa, H., Kawai, A., and Ebisuzaki, T. (2000). 1.34 TFLOPS molecular dynamics simulation for NaCl with a special-purpose computer: MDM. In *ACM/IEEE Conference on Supercomputing (SC)*, pages 54:1–54:20.

Obeidat, A., Jaradat, A., Hamdan, B., and Abu-Ghazleh, H. (2018). Effect of cutoff radius, long range interaction and temperature controller on thermodynamic properties of fluids: Methanol as an example. *Physica A: Statistical Mechanics and its Applications*, 496.

Ohmura, I., Morimoto, G., Ohno, Y., Hasegawa, A., and Taiji, M. (2014). MDGRAPE-4: a special purpose computer system for molecular dynamics simulations. *Philosophical Transactions of the Royal Society A*, 372(20130387).

Okumura, S., Makino, J., Ebisuzaki, T., Ito, T., Fukushige, T., Sugimoto, D., Hashimoto, E., Tomida, K., and Miyakawa, N. (1992). GRAPE-3: highly parallelized special-purpose computer for gravitational many-body simulations. In *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, volume i, pages 151–160 vol.1.

Páll, S., Zhmurov, A., Bauer, P., Abraham, M., Lundborg, M., Gray, A., Hess, B., and Lindahl, E. (2020). Heterogeneous parallelization and acceleration of molecular dynamics simulations in gromacs. *The Journal of Chemical Physics*, 153(13):134110.

Pascoe, C., Stewart, L., Sherman, B., Sachdeva, V., and Herbordt, M. (2020). Execution of Complete Molecular Dynamics Simulations on Multiple FPGAs. In *IEEE High Performance Extreme Computing Conference*.

Patel, R., Haghi, P., Jain, S., Kot, A., Krishnan, V., Varia, M., and Herbordt, M. (2022a). COPA Use Case: Distributed Secure Joint Computation. In *30th IEEE International Symposium on Field-Programmable Custom Computing Machines*. doi: 10.1109/FCCM53951.2022.9786156.

Patel, R., Haghi, P., Jain, S., Kot, A., Krishnan, V., Varia, M., and Herbordt, M. (2022b). Distributed Hardware Accelerated Secure Joint Computation on the COPA Framework. In *IEEE High Performance Extreme Computing Conference*. doi: 10.1109/HPEC55821.2022.9926388.

Patel, R., Wolfe, P.-F., Munafo, R., Varia, M., and Herbordt, M. Arithmetic and Boolean Secret Sharing MPC on FPGAs in the Data Center. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, Waltham, MA, USA, 2020, pp. 1-8, doi: 10.1109/HPEC43674.2020.9286159.

Phillips, J., Braun, R., Wang, W., Gumbart, J., Tajkhorshid, E., Villa, E., Chipot, C., Skeel, R., Kale, L., and Schulten, K. (2005). Scalable molecular dynamics with NAMD. *Journal of Computational Chemistry*, 26:1781–1802.

Plessl, C. (2018). Bringing FPGAs to HPC Production Systems and Codes. In *H2RC'18 workshop at Supercomputing (SC'18)*. doi: 10.13140/RG.2.2.34327.42407.

Putnam, A. (2014). A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services. In *International Symposium on Computer Architecture*, pages 13–24. doi: 10.1109/ISCA.2014.6853195.

Salo-Ahen, O. M., Alanko, I., Bhadane, R., Bonvin, A. M., Honorato, R. V., Hossain, S., Juffer, A. H., Kabedev, A., Lahtela-Kakkonen, M., Larsen, A. S., et al. (2020). Molecular dynamics simulations in drug discovery and pharmaceutical development. *Processes*, 9(1):71.

Sanaullah, A., C.Yang, Alexeev, Y., Yoshii, K., and Herbordt, M. (2018a). Application Aware Tuning of Reconfigurable Multi-Layer Perceptron Architectures. In *IEEE High Performance Extreme Computing Conference*.

Sanaullah, A. and Herbordt, M. (2018a). An Empirically Guided Optimization Framework for FPGA OpenCL. In *2018 International Conference on Field Programmable Technology (FPT)*, pages 46–53. doi: 10.1109/FPT.2018.00018.

Sanaullah, A. and Herbordt, M. (2018b). FPGA HPC using OpenCL: Case Study in 3D FFT. In *9th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, page 1–6. doi: 10.1145/3241793.3241800.

Sanaullah, A. and Herbordt, M. (2018c). Unlocking Performance-Programmability by Penetrating the Intel FPGA OpenCL Toolflow. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*. doi: 10.1109/HPEC.2018.8547646.

Sanaullah, A., Khoshparvar, A., and Herbordt, M. (2016a). FPGA-Accelerated Particle-Grid Mapping. In *IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines*, pages 192–195. doi: 10.1109/ FCCM .2016.53.

Sanaullah, A., Lewis, K., and Herbordt, M. (2016b). GPU Accelerated Particle-Grid Mapping. In *IEEE High Performance Extreme Computing Conference*. DOI: 10.1109/ HPEC.2016.7761599.

Sanaullah, A., Sachdeva, V., and Herbordt, M. (2018b). SimBSP: Enabling RTL Simulation for Intel FPGA OpenCL Kernels. In *Proc. Heterogeneous High Performance Reconfigurable Computing*. https://www.bu.edu/caadlab/H2RC18.pdf.

Sanaullah, A., Yang, C., Alexeev, Y., Yoshii, K., and Herbordt, M. (2018c). Real-Time Data Analysis for Medical Diagnosis using FPGA Accelerated Neural Networks. *BMC Bioinformatics*, 19 Supplement 18. doi: 10.1186/s12859-018-2505-7.

Sasaki, T., Betsuyaku, K., Higuchi, T., and Nagashima, U. (2005). Reconfigurable 3D-FFT Processor for the Car-Parrinello Method. *Journal of Computer Chemistry, Japan*, 4(4):147–154.

Schaffner, M. and Benini, L. (2018). On the feasibility of FPGA acceleration of molecular dynamics simulations. https://arxiv.org/abs/1808.04201.

Scrofano, R., Gokhale, M., Trouw, F., and Prasanna, V. (2006). A hardware/software approach to molecular dynamics on reconfigurable computers. In *Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines*, pages 23–32.

Scrofano, R., Gokhale, M., Trouw, F., and Prasanna, V. (2008). Accelerating Molecular Dynamics Simulations with Reconfigurable Computers. *IEEE Transactions on Parallel and Distributed Systems*, 19(6):764–778.

Scrofano, R. and Prasanna, V. (2006). Preliminary investigation of advanced electrostatics in molecular dynamics on reconfigurable computers. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*.

Shahzad, H., Sanaullah, A., Arora, S., Munafo, R., Yao, X., Drepper, U., and Herbordt, M. (2022). Reinforcement Learning Strategies for Compiler Optimization in High Level Synthesis. In *The Eighth Workshop on the LLVM Compiler Infrastructure in HPC*. DOI: 10.1109/LLVM-HPC56686.2022.00007.

Shahzad, H., Sanaullah, A., and Herbordt, M. (2021). Survey and Future Trends for FPGA Cloud Architectures. In *Proceedings of the IEEE High Performance Extreme Computing Conference*.

Shaw, D. (2005). A Fast, Scalable Method for the Parallel Evaluation of Distance-Limited Pairwise Particle Interactions. *Journal of Computational Chemistry*, 26(13):1318–1328.

Shaw, D. E., Adams, P. J., Azaria, A., Bank, J. A., Batson, B., Bell, A., Bergdorf, M., Bhatt, J., Butts, J. A., Correia, T., Dirks, R. M., Dror, R. O., Eastwood, M. P., Edwards, B., Even, A., Feldmann, P., Fenn, M., Fenton, C. H., Forte, A., Gagliardo, J., Gill, G., Gorlatova, M., Greskamp, B., Grossman, J., Gullingsrud, J., Harper, A., Hasenplaugh, W., Heily, M., Heshmat, B. C., Hunt, J., Ierardi, D. J., Iserovich, L., Jackson, B. L., Johnson, N. P., Kirk, M. M., Klepeis, J. L., Kuskin, J. S., Mackenzie, K. M., Mader, R. J., McGowen, R., McLaughlin, A., Moraes, M. A., Nasr, M. H., Nociolo, L. J., O'Donnell, L., Parker, A., Peticolas, J. L., Pocina, G., Predescu, C., Quan, T., Salmon, J. K., Schwink, C., Shim, K. S., Siddique, N., Spengler, J., Szalay, T., Tabladillo, R., Tartler, R., Taube, A. G., Theobald, M., Towles, B., Vick, W., Wang, S. C., Wazlowski, M., Weingarten, M. J., Williams, J. M., and Yuh, K. A. (2021). Anton 3: twenty microseconds of molecular dynamics simulation before lunch. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11.

Shaw, D. E., Deneroff, M. M., Dror, R. O., Kuskin, J. S., Larson, R. H., Salmon, J. K., Young, C., Batson, B., Bowers, K. J., Chao, J. C., Eastwood, M. P., Gagliardo, J., Grossman, J. P., Ho, C. R., Ierardi, D. J., Kolossváry, I., Klepeis, J. L., Layman, T., McLeavey, C., Moraes, M. A., Mueller, R., Priest, E. C., Shan, Y., Spengler, J., Theobald, M., Towles, B., and Wang, S. C. (2007). Anton, a special-purpose machine for molecular dynamics simulation. In *Proceedings of the International Symposium on Computer Architecture*, pages 1–12.

Shaw, D. E., Grossman, J., Bank, J. A., Batson, B., Butts, J. A., Chao, J. C., Deneroff, M. M., Dror, R. O., Even, A., Fenton, C. H., Forte, A., Gagliardo, J., Gill, G., Greskamp, B., Ho, C. R., Ierardi, D. J., Iserovich, L., Kuskin, J. S., Larson, R. H., Layman, T., Lee, L.-S., Lerer, A. K., Li, C., Killebrew, D., Mackenzie, K. M., Mok, S. Y.-H., Moraes, M. A., Mueller, R., Nociolo, L. J., Peticolas, J. L., Quan, T., Ramot, D., Salmon, J. K., Scarpazza, D. P., Schafer, U. B., Siddique, N., Snyder, C. W., Spengler, J., Tang, P. T. P., Theobald, M., Toma, H., Towles, B., Vitale, B., Wang, S. C., and Young, C. (2014). Anton 2: raising the bar for performance and programmability in a special-purpose molecular dynamics supercomputer. In *SC '14: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 41–53.

Sheng, J., Humphries, B., Zhang, H., and Herbordt, M. (2014). Design of 3D FFTs with FPGA Clusters. In *IEEE High Performance Extreme Computing Conference*. doi: 10.1109/ HPEC.2014.7040997.

Sheng, J., Xiong, Q., Yang, C., and Herbordt, M. (2017a). Collective Communication on FPGA Clusters with Static Scheduling. *ACM SIGARCH Computer Architecture News*, 44(4). doi: 10.1145/ 3039902.3039904.

Sheng, J., Yang, C., Caulfield, A., Papamichael, M., and Herbordt, M. (2017b). HPC on FPGA Clouds: 3D FFTs and Implications for Molecular Dynamics. In *27th International Conference on Field Programmable Logic and Applications*. doi: 10.23919/ FPL.2017.8056853.

Sheng, J., Yang, C., and Herbordt, M. (2015). Towards Low-Latency Communication on FPGA Clusters with 3D FFT Case Study. In *International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*. https:// pdfs.semanticscholar.org /832d/ c69145f5ba0ed6a951583201b1b20dd 2096e.pdf.

Sheng, J., Yang, C., and Herbordt, M. (2016). Application-Aware Collective Communication on FPGA Clusters. In *IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. doi: 10.1109/ FCCM.2016.55.

Shi, R., Dong, P., Geng, T., Ding, Y., Ma, X., So, H., Herbordt, M., Li, A., and Wang, Y. (2020). CSB-RNN: A Faster-than-Realtime RNN Acceleration Framework with Compressed Structured Blocks. In *International Conference on Supercomputing*.

Sibai, F. N. (1998). The hyper-ring network: a cost-efficient topology for scalable multicomputers. In *ACM Symposium on Applied Computing*.

Snir, M. (2004). A note on N-body computations with cutoffs. *Theory of Computing Systems*, 37:295–318.

Stewart, L., Pascoe, C., Davis, E., Sherman, B., Herbordt, M., and Sachdeva, V. (2021). Particle Mesh Ewald for Molecular Dynamics in OpenCL on an FPGA Cluster. In *IEEE Symposium on Field Programmable Custom Computing Machines*.

Sukhwani, B. and Herbordt, M. GPU acceleration of a production molecular docking code. In *GPGPU-2: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, pp. 19–27. https://dl.acm.org/doi/proceedings/10.1145/1513895.

Sukhwani, B. and Herbordt, M. (2008). Acceleration of a Production Rigid Molecule Docking Code. In *2008 International Conference on Field Programmable Logic and Applications*, pages 341–346. doi: 10.1109/ FPL.2008.4629955.

Sumanth, J., Swanson, D., and Jiang, H. (2003). Performance and cost effectiveness of a cluster of workstations and MD-GRAPE 2 for MD simulations. In *Second International Symposium on Parallel and Distributed Computing*, pages 244–249. doi: 10.1109/ISPDC.2003.1267670.

Susukita, R., Ebisuzaki, T., Elmegreen, B. G., Furusawa, H., Kato, K., Kawai, A., Kobayashi, Y., Koishi, T., McNiven, G. D., Narumi, T., and Yasuoka, K. (2003). Hardware accelerator for molecular dynamics: MDGRAPE-2. *Computer Physics Communications*, 155(2):115–131. doi: 10.1016/S0010-4655(03)00349-7.

Thompson, A. P., Aktulga, H. M., Berger, R., Bolintineanu, D. S., Brown, W. M., Crozier, P. S., in 't Veld, P. J., Kohlmeyer, A., Moore, S. G., Nguyen, T. D., Shan, R., Stevens, M. J., Tranchida, J., Trott, C., and Plimpton, S. J. (2022). LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer Physics Communications*, 271:108171. doi: 10.1016/j.cpc.2021.108171.

Towles, B., Grossman, J., Greskamp, B., and Shaw, D. (2014). Unifying on-chip and inter-node switching within the Anton 2 network. In *Proceedings of the International Symposium on Computer Architecture*, pages 1–12. doi: 10.1109/ISCA.2014.6853238.

VanCourt, T., Gu, Y., and Herbordt, M. (2004). FPGA acceleration of rigid molecule interactions. In *12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 300–301. doi: 10.1109/ FCCM.2004.33.

VanCourt, T. and Herbordt, M. (2004). Families of FPGA-based algorithms for approximate string matching. In *Proceedings. 15th IEEE International Conference on Application-Specific Systems, Architectures and Processors, 2004.*, pages 354–364. doi: 10.1109/ ASAP.2004.1342484.

VanCourt, T. and Herbordt, M. (2005a). LAMP: A tool suite for families of FPGA-based application accelerators. In *International Conference on Field Programmable Logic and Applications, 2005.* doi: 10.1109/ FPL.2005.1515797.

VanCourt, T. and Herbordt, M. (2005b). Three dimensional template correlation: Object recognition in 3D voxel data. In *Seventh International Workshop on Computer Architecture for Machine Perception (CAMP'05)*, pages 153–158. doi: 10.1109/ CAMP.2005.52.

VanCourt, T. and Herbordt, M. (2006a). Application-dependent memory interleaving enables high performance in FPGA-based grid computations. In *IEEE Conference on Field Programmable Logic and Applications*, pages 395–401. doi: 10.1109/ FCCM.2006.25.

VanCourt, T. and Herbordt, M. (2006b). Rigid molecule docking: FPGA reconfiguration for alternative force laws. *Journal on Applied Signal Processing*, v2006:1–10. doi: 10.1155/ ASP/2006/97950.

VanCourt, T. and Herbordt, M. (2006c). Sizing of processing arrays for FPGA-based computation. In *2006 International Conference on Field Programmable Logic and Applications*, pages 755–760. doi: 10.1109/ FPL.2006.311307.

VanCourt, T. and Herbordt, M. (2007). Families of FPGA-based accelerators for approximate string matching. *Microprocessors and Microsystems*, 31(2):135–145. doi: 10.1016/ j.micpro.2006.04.001.

VanCourt, T. and Herbordt, M. (2009). Elements of high performance reconfigurable computing. In Zelkowitz, M., editor, *Advances in Computers*, volume v75, pages 113–157. Elsevier. doi: 10.1016/ S0065-2458(08)00802-4.

Wang, T., Geng, T., Jin, X., and Herbordt, M. (2019a). Accelerating AP3M-Based Computational Astrophysics Simulations with Reconfigurable Clusters. In *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 181–184. doi: 10.1109/ ASAP.2019.000-5.

Wang, T., Geng, T., Jin, X., and Herbordt, M. (2019b). FP-AMR: A Reconfigurable Fabric Framework for Block-Structured Adaptive Mesh Refinement Applications. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 245–253. doi: 10.1109/ FCCM.2019. 00040.

Wang, T., Geng, T., Li, A., Jin, X., and Herbordt, M. (2020). FPDeep: Scalable Acceleration of CNN Training on Deeply-Pipelined FPGA Clusters. *IEEE Transactions on Computers*, C-69(8):1143–1158. doi: 10.1109/TC.2020.3000118.

Wolfe, P.-F., Patel, R., Munafo, R., Varia, M., and Herbordt, M. (2020). Secret Sharing MPC on FPGAs in the Datacenter. In *IEEE Conference on Field Programmable Logic and Applications*.

Wu, C., Bandara, S., Geng, T., Guo, A., Haghi, P., Sherman, W., Sachdeva, V., and Herbordt, M. (2022). Optimized Mappings for Symmetric Range-Limited Molecular Force Calculations on FPGAs. In *International Conference on Field-Programmable Logic and Applications*. DOI: 10.1109/FPL57034.2022.00026.

Wu, C., Bandara, S., Geng, T., Sachdeva, V., Sherman, B., and Herbordt, M. (2021a). System-Level Modeling of GPU/FPGA Clusters for Molecular Dynamics Simulations. In *IEEE High Performance Extreme Computing Conference*. doi: 10.1109/HPEC49654.2021.9622838.

Wu, C., Geng, T., Bandara, S., Yang, C., Sachdeva, V., Sherman, W., and Herbordt, M. (2021b). Upgrade of FPGA Range-Limited Molecular Dynamics to Handle Hundreds of Processors. In *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. doi: 10.1109/FCCM51124.2021.00024.

Wu, C., Geng, T., Guo, A., Bandara, S., Haghi, P., Liu, C., Li, A., and Herbordt, M. (2023). FASDA: An FPGA-Aided, Scalable and Distributed Accelerator for Range-Limited Molecular Dynamics. In *International Conference for High Performance Computing, Networking, Storage and Analysis*.

Wu, C., Geng, T., Sachdeva, V., Sherman, W., and Herbordt, M. (2020). A Communication-Efficient Multi-Chip Design for Range-Limited Molecular Dynamics. In *2020 IEEE High Performance extreme Computing Conference (HPEC)*.

Xiong, Q. and Herbordt, M. (2017). Bonded Force Computations on FPGAs. In *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 72–75. doi: 10.1109/ FCCM.2017.49.

Xiong, Q., Yang, C., Haghi, P., Skjellum, A., and Herbordt, M. (2020). Accelerating MPI Collectives with FPGAs in the Network and Novel Communicator Support. In *IEEE Symposium on Field Programmable Custom Computing Machines*.

Yang, C., Geng, T., Wang, T., Patel, R., Xiong, Q., Sanaullah, A., Wu, C., Sheng, J., Lin, C., Sachdeva, V., Sherman, W., and Herbordt, M. (2019a). Fully integrated FPGA molecular dynamics simulations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, New York, NY, USA. Association for Computing Machinery.

Yang, C., Geng, T., Wang, T., Sheng, J., Lin, C., Sachdeva, V., Sherman, W., and Herbordt, M. (2019b). Molecular Dynamics Range-Limited Force Evaluation Optimized for FPGA. In *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 263–271. doi: 10.1109/ ASAP.2019.00016.

Yang, C., Sheng, J., Patel, R., Sanaullah, A., Sachdeva, V., and Herbordt, M. (2017). OpenCL for HPC with FPGAs: Case Study in Molecular Electrostatics. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–8. doi: 10.1109/ HPEC.2017. 8091078.

Yao, Z., Wang, J.-S., Liu, G.-R., and Cheng, M. (2004). Improved neighbor list algorithm in molecular simulations using cell decomposition and data sorting method. *Computer Physics Communications*, 161(1):27 – 35.

Young, C., Bank, J., Dror, R., Grossman, J., Salmon, J., and Shaw, D. (2009). A 32x32x32, spatially distributed 3D FFT in four microseconds on Anton. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–11.

Yuan, M., Liu, Q., Deng, Q., Xiang, S., Gan, L., Yang, J., Duan, X., Fu, H., and Yang, G. (2022). FPGA-accelerated tersoff multi-body potential for molecular dynamics simulations. In Gan, L., Wang, Y., Xue, W., and Chau, T., editors, *Applied Reconfigurable Computing. Architectures, Tools, and Applications*, pages 17–31, Cham. Springer Nature Switzerland.

Zhang, J., Xiong, Y., Xu, N., Shu, R., Li, B., Cheng, P., Chen, G., and Moscibroda, T. (2017). The feniks FPGA operating system for cloud computing. In *Proceedings of the 8th Asia-Pacific Workshop on Systems*, pages 1–7.

Zhao, H. and Caflisch, A. (2015). Molecular dynamics in drug design. *European journal of medicinal chemistry*, 91:4–14.

CURRICULUM VITAE

[redacted]

[redacted]

[redacted]

[redacted]