ISSN: 0067-2904

An Adaptive Parallel Pattern Based Design for Molecular Dynamic Simulation

Nilesh N. Maltare[1]*, Viral N. Kamat[2]
[1]*Information Technology Department, Government Engineering College, Modasa, Gujarat, India*
[2]*Centre for Apparent Energy Research, Anand, Gujarat, India*

Received: 29/9/2021 Accepted: 12/3/2023 Published: 29/2/2024

Abstract

In Parallel programming, a programmer needs to understand hardware environment, programming paradigm and primitives available in the programming language. Most of the time, parallel programmes are written for a specific architecture and cannot typically adapt to other architectures Particularly, programs written for shared memory architectures are unsuitable for distributed or hybrid architectures. This paper proposes Adaptive Design Pattern for Parallel Programming to improve adaptability, flexibility with achieving performance on different architectures.

Molecular Dynamics (MD) simulation is required to scale to various architectures from simple machine to cluster of workstations. In this study, MD Simulation experimented using both pure benchmark code and code based on adaptive design patterns. Redesigned MD Simulation with Adaptive Design Pattern claims parallel efficiency from 56% to 90% for different number of processing elements used. The solution demonstrates adaptability to different architectures and scalability to use with large number of atoms and long duration simulation.

Keywords: Parallel Programming, Parallel Design Patterns, Hybrid OpenMP-MPI Programming, MD Simulation.

1. Introduction

Molecular dynamics (MD) is a technique for simulating the atom-by-atom behaviour of molecules and deriving macroscopic properties from these atomistic motions. It has application in materials science and nanotechnology. MD simulation starts with considering interaction of 100 atoms by Haile [1]. The advent of High-Performance Computing (HPC) facilitates a wide range of applications [2], [3], [4], [5], [6] of MD simulation including genome sequencing, protein structure prediction, molecular docking and drug design. The algorithm matured continuously with more capabilities [7], [8], [9]. The historical development in field of MD Simulation is captured in [10]. MD Simulation is important, and it has wide applications [11], [12], [13], [14].

The classical MD Simulation [15] calculates primitive parameters like energy, forces, velocity, acceleration, spatial coordinate of atoms. In this paper, MD Simulation is experimented with the standard benchmark code i.e., Large scale Atomic Molecular Massive

*Email: nilesh.maltare@gecmodasa.org

Parallel Simulator (LAMMPS) [16]. The pure MD Simulation code is compared with Adaptive Design Pattern based code. The objective is to demonstrate benefits of adaptive parallel design pattern-based design on different parallel environment (MPI, OpenMP and hybrid OpenMP-MPI).

MD simulation calculates different parameters like distance, force, energy, etc. on the window range. Earlier MD Simulation considers 5000 to 100000 atoms, but due to HPC millions and trillions of atoms can be simulated today. MD Simulation is computationally intensive application. It belongs to an embarrassing parallel problem class in which one can take benefit of parallelism most. Due to this property; MD simulations are continuously improving by taking benefit of advancement in High Performance Computing (HPC).
The important steps in MD Simulation can be summarized in following flowchart (Figure 1):

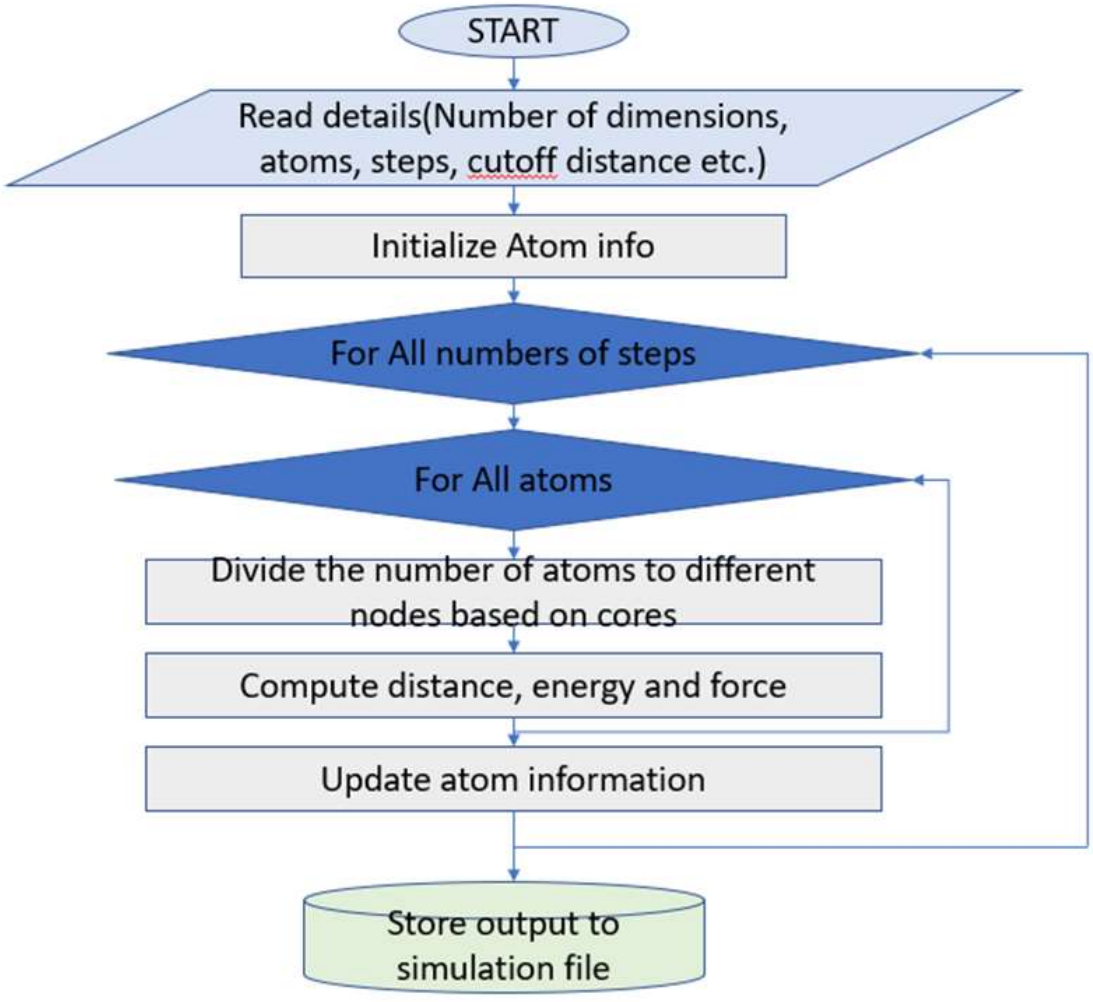

Figure 1: MD Simulation Flowchart

The paper discusses the need of Adaptive design pattern in the context of parallel programming. This paper proposed Adaptive Design Pattern for Parallel Programming to make application more adaptive for any kind of parallel architecture. The same approach is compared with conventional pure code written in OpenMP, MPI and Hybrid Programming.

2. Parallel Programming Tools and Methodologies

2.1 Shared Memory Paradigm

OpenMP (Open Multiprocessing) is based on the shared memory paradigm. Open MP provides programmers variety of compiler directives and library routines for parallelizing their codes. OpenMP [17] supports various programming languages like C, C++, Fortran, etc., operating systems like Windows, Linux, Mac, HP-UX, etc. Programmers can use OpenMP to handle most aspects of the parallelization by providing enough information for it to do so instead of having to manually create and manage threads.

2.2 Distributed Memory Paradigm

Message Passing Interface (MPI) is the most popular programming interface for distributed computing. MPI [18], [19] is a collection of API functions for facilitating communication between processes running in a computing cluster. MPI is based on a distributed memory paradigm in which processes communicate with one another by passing messages. Applications do not have to deal with the specifics of the interconnect network while using API communication functions. The processes can communicate with one another by using logical numbers. Data and tasks are divided among processes in a typical MPI application. All the processes in MPI communicate through the exchange of messages. Sometimes, the processes when working together on a collaborative work, processes use synchronization constructs and produce collaborative output. The MPI collective API functions are used for this.

2.3 Hybrid Programming

Hybrid programming facilitates efficient programming of clusters of shared memory (SMP) nodes [20]. In Hybrid programming MPI is used at the nodes and Shared memory programming inside of each SMP node. We can take benefit of both (Scalability of MPI and efficient sharing between peer threads of OpenMP). Hybrid Programming can use following combinations:

a. MPI Process – OpenMP Threads

b. MPI Process- Lightweight MPI Process (MPI-3.0 shared memory programming [21])

2.4 Parallel Design Patterns

A design pattern is the common solution to a recurring problem in particular context. Parallel programming patterns classified by Matson et al [22], [23]. Our Pattern Language (OPL) [24] is also categorize patterns for high-performance computing applications. Design patterns which describe the overall parallel architecture and software structure are known as Program Structure Patterns. Popular patterns for program structure are Single Program Multiple Data (SPMD), Master/Worker, Loop parallelism, Fork/Join are available.

2.4.1 Master Worker

In Master Worker pattern Master process works as coordinator. A master process or thread set up a pool of worker processes of threads and a bag of tasks. The workers execute concurrently, with each worker perform a task or set of tasks from the bag. Master collects results from all workers. Master Worker pattern is useful for embarrassingly parallel problem, in which parallelism is easy to exploit. Figure 2 shows typical working of Master Worker Pattern

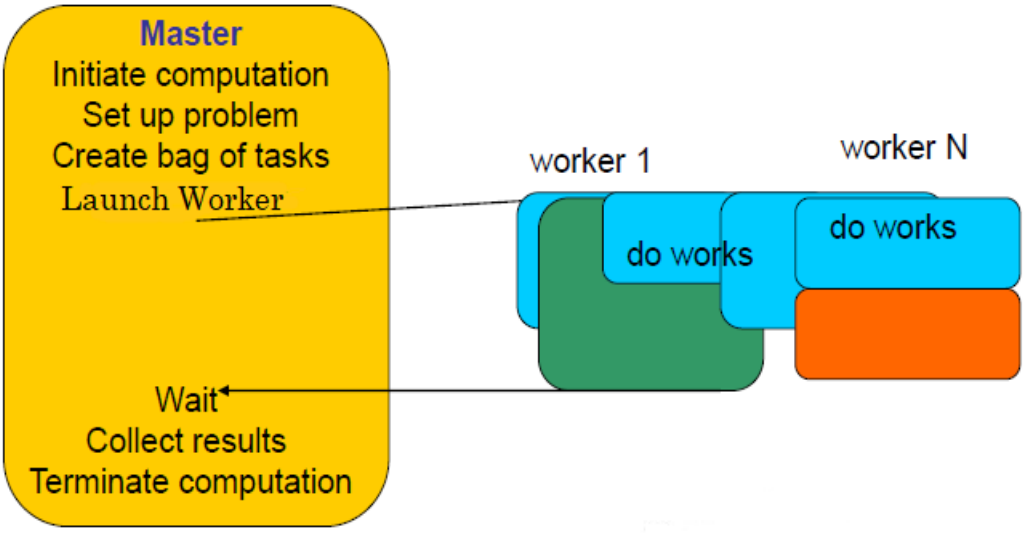

Figure 2: Master Worker Pattern

2.4.2 Single Program Multiple Data (SPMD)

SPMD is suitable for Symmetric Multiprocessing (SMP) Architectures. In SPMD, all Cores execute the same program in parallel, but each has its own set of data. The typical SPMD program structure is shown in Figure 3.

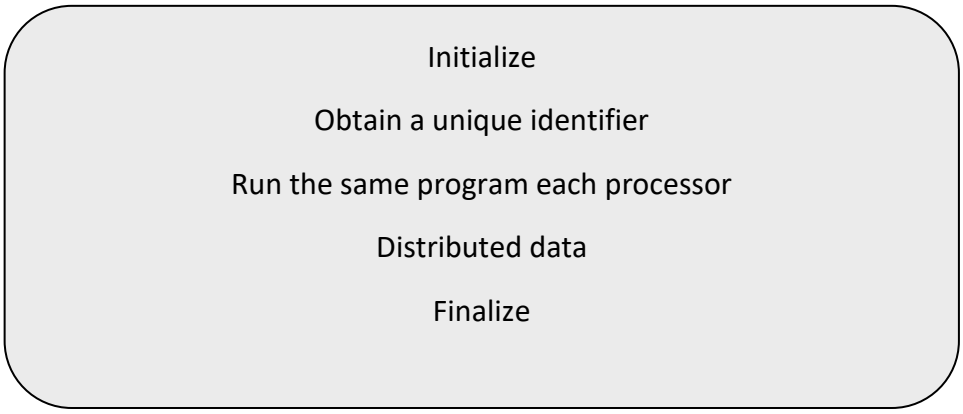

Figure 3: SPMD Pattern

3. The Need of Adaptive Design for Parallel Software Development

The software community provided notion of patterns to capture and communicate best practices of software development. The use of pattern improves communication and reduces ambiguity. Design patterns applied in parallel programming problem can be solved through generalized reusable solution. The design pattern for parallel programming is not new and many papers [25], [26], [27] talked about it. Parallel Design patterns identify structural parallelism for application in given context and suggest reusable solution. The Parallel Pattern Language (PPL) [22] is a catalogue of parallel patterns useful for parallel software development. PPL helps parallel programmers in every phase of parallel software design and development.

In Architecture with multicores communication latency is reduced but for scalability we may have to go for cluster of multicores. That is the reason we need to design application to exploit parallelism by use of structural and communication patterns. The pattern-based design enhances flexibility, scalability, and adaptability. Novice programmers can develop quality application by following best practices captured with design patterns.

The design of parallel software is often biased with the architecture for which it is expected to deploy. The interoperability of design pattern in parallel programming is not effortless. Programmer can use best practices and start with parallel program structures without studying details of computer architecture available. In [28] the need of patterns which can behave according to architecture is emphasized. The separation of concerns can also be achieved using patterns. Model View Controller (MVC) [29] is example in context of information system development. In similar way it is possible to apply best practices of advanced software engineering in the parallel program development.

4. Adaptive Parallel Design Pattern

This paper proposed Adaptive design pattern (Figure 4) which provides adaptability for incorporating different patterns in different situations. The pattern is generic and not only defined for MD Simulation, but It can also be applied to any parallel programming problem. The MD Simulation is used to demonstrate pattern benefit due to its computationally intensive property and the parallelism available in it. Adaptive Design Pattern gives flexibility to the selection of pattern dynamically. It will select Open MP or Hybrid MPI – OpenMP Implementation. Single Program Multiple Data (SPMD) based design and OpenMP implementation triggered for shared memory architecture. Adaptive design and Hybrid OpenMP, MPI implementation will be triggered in case of cluster and Master – Coordinator-Worker design may be used.

Adaptive pattern-based design gives us the flexibility to select a particular design based on suitability of design pattern to environment, programming language constructs. The selection of pattern is based on best practices of parallel program design. The best practices captured by [30], [31] and discussed in [28]. We summarize it for different scenario in Table 1.

Table 1: Suitability of pattern and languages

Pattern	Open MP	MPI
Pipeline	Suitable	Suitable
Recursive splitting	Suitable	
Geometric Decomposition	Suitable	Suitable
Discrete Event		Suitable
Actors		Suitable
Master/worker		Suitable
BSP		Suitable
SPMD	Suitable	Suitable
Loop Parallelism	Suitable	
Fork/Join	Suitable	

Table 2: Mapping of hardware, implementation pattern and programming languages

Hardware Environment	Suitable Implementation Pattern	Programming Language	Capabilities added dynamically
Shared Memory	SPMD	OpenMP	Load balancing
Distributed Memory Cluster	Master Worker	Hybrid OpenMP/MPI, MPI	Load balancing, Code migration, Fault tolerance
	Two level Master Worker	Hybrid OpenMP/MPI, MPI	Load balancing, code migration, Fault tolerance

Table 2 is based on experiments on prime number generation [32] and MD Simulation [33]. It shows selection of pattern with considering hardware environment and programming language. The experiment implemented with Single Program Multiple Data (SPMD), Master worker and Multilevel Master worker but design will be able to accommodate any pattern. It demonstrated selection of pattern-based hardware environment and programming language used. The adaptive design implemented with suitable implementation pattern and programming language. The implementation also flexibly switches over to different pattern-based design based on situations shown in Table 2.

Social insects like ant have been extraordinarily successful in task allocation. The study [34] shows parallel nature of ant colony suits it for HPC. Social insect's exhibit most advanced form of sociality. In Adaptive design pattern, we have tried to inherit features from Honeybee task allocation [35]. Honeybees exhibits two patterns of organization of work, In the spring and summer, division of labour is used to maximize growth rate and resource accumulation. In winter, honeybee focuses on survival. They rely on honey and become generalists. Honeybee task allocation also focus on context for division of work.

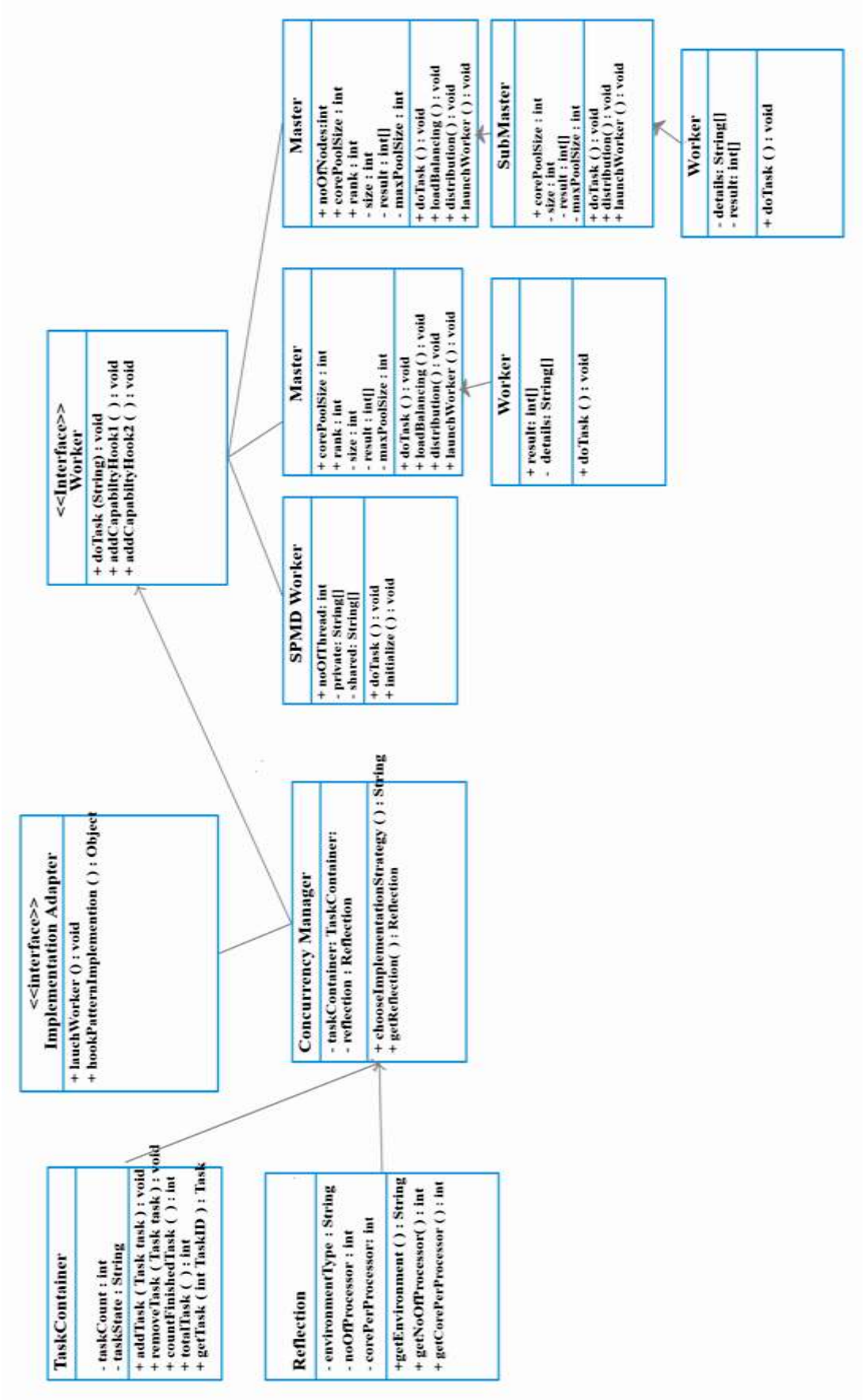

Figure 4: Proposed Adaptive Design Pattern for Parallel Programming

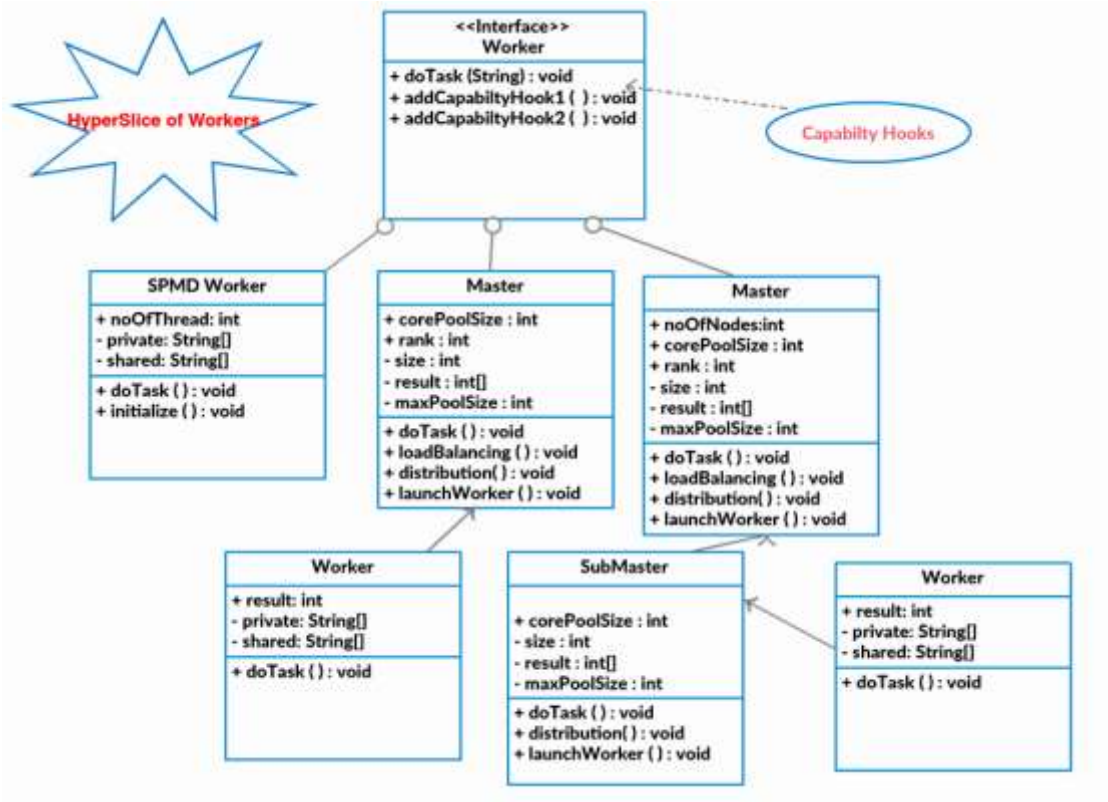

Figure -5 Hierarchy of worker in Adaptive design pattern

In Adaptive design pattern, we are using honeybees' flexible role transition model. The Adaptive design pattern uses capabilities which can be added dynamically to workers. The worker hierarchy (Figure 5) is implemented using the concept of hyperslice [36], [37], each worker can be replaced by other workers. The worker can increase its capabilities dynamically by hooks [38]. Each worker is capable to acquire capabilities dynamically.

It accommodates adaptability in benefit of such design is flexibility of adding capabilities to workers. This design choose pattern based on suitability to architecture, programming environment and optimization available. The Flowchart shown in Figure 6 explains the algorithm to choose different patterns and capabilities. The framework can be planned based on this algorithm to relieve parallel developers from jargon of patterns and their mapping to hardware.

5. Adaptive Parallel Design Pattern Based Solution

MD Simulation requires high computational resources, and its complexity grows with number of atoms, range of forces considered, step size and other parameters. There has been lots of effort to parallelize MD Simulation [39], [40], [41], [42], [43], [44]. The objective of this study is not to formulate better parallel algorithm, but with the same algorithm achieving better mapping to different parallel architectures.

The MD Simulation uses data structures such as positions and velocities of particles and the computing procedures such as calculations of forces and updates of particle positions are partitioned and allotted to each processor.

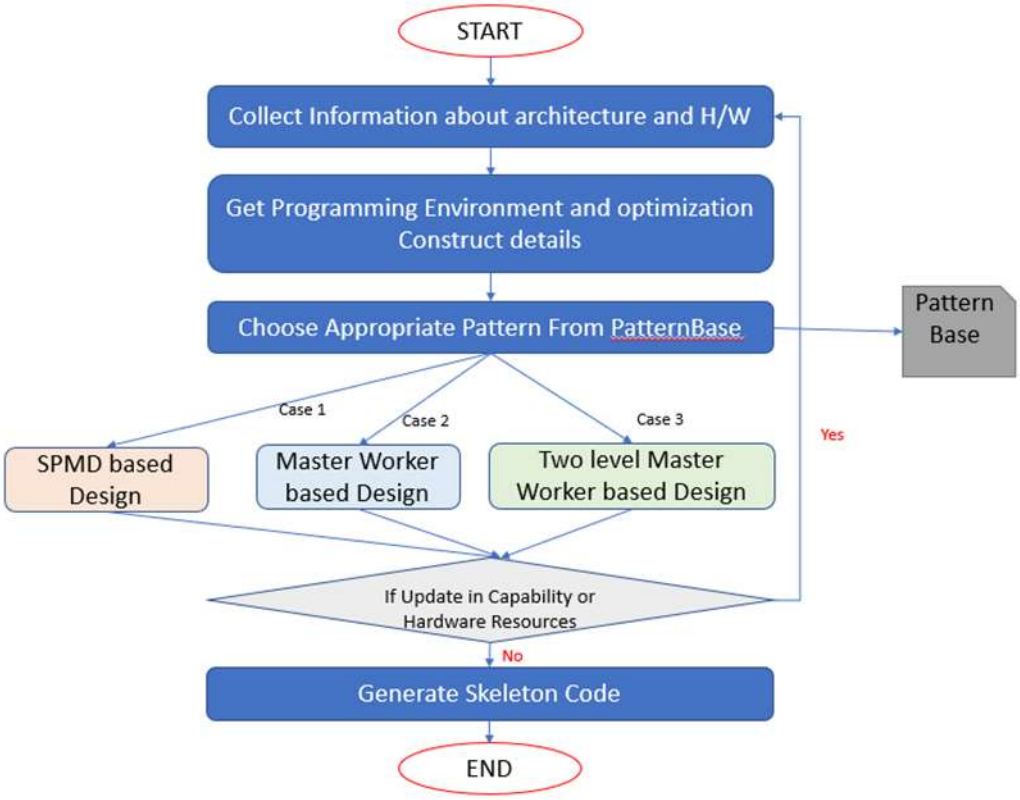

Figure 6: Selection of patterns in Adaptive design pattern

Important metrics will be communication time, efficiency, and core utilization. The other factors for comparing different solutions from software engineering perspectives are:
1.Ease of Programming
2.Fault tolerance
3.Adaptability of solution to different architectures
4.Flexibility

MD Simulation uses different methods of partitioning for calculating parameters in parallel. These methods use particle or spatial decomposition. In this paper, we are considering adaptability and better mapping to hardware (shared, distributed, hybrid) of MD Simulation design. If MD Simulation solution adopts variety of architecture, it will be useful in different situation. The other computation intensive problems can also be designed to adopt different architectures with proposed pattern. We have also used hybrid programming in which we can benefit of both OpenMP and MPI. MPI approach provides scalability and fault tolerance but suffers from communication latency. Open MP is faster in creation of threads and share resources efficiently. Hybrid approach takes advantages of both OpenMP and MPI.
The pattern-based solution can achieve separation of concern, improved readability, debugging, adaptability and flexibility.

6. Results

This experiment considered 3D physical space, 1000 atoms and 16-time step. The following information is used to compute distance, forces, and energy:
•Velocity
•Position
•Charge
•Acceleration

All the variables are initialized with random inputs. The simulation space in problem is divided into rectangular cells. Each region is calculated in parallel. The experiment only considering short range forces and if atom goes outside range it will move in respective region. With cell partitioning method, the pair list for an atom in a particular cell is constructed, which contains atom and neighbouring atom within cell.

The cell partitioning and short-range calculation minimize the communication required with other cells. Table 3 gives Latency and communication/computation ratio in Pure MPI code of MD Simulation. The ratio of communication to computation grows with an increase in MPI processes. Parallel efficiency will be impacted by the increase in latency. Interleaved execution is preferred to cut down communication time. Every process in an MPI implementation interacts with other processes through explicit MPI communication (MPI send/recv). Speedup is limited by the explicit MPI communication between nodes as more and more time is lost in communication. The OpenMP threads are running at the SMP (Symmetric Multiprocessor Node) in this implementation that is the reason we are considering negligible latency in OpenMP threads. Figure 7 and Figure 8 shows comparison of execution time in second with different approaches.

Table 3: Latency and Communication /Computation Ratio

No. Of Process	Pure MPI Latency (in seconds)	Communication /Computation Ratio
1	0	0
2	0.000462	0
4	0.000468	0
8	0.000466	0
16	0.000599	0
32	1.553018	13.71
64	1.941602	20.91
128	2.149467	48.98

Table 4 summarize our experiment in terms of execution time. The execution time is given in seconds. We have compared four different cases:
1.Pure Open MP code
2.Pure MPI code
3.Best case of Hybrid code (OpenMP-MPI)
4.Adaptive Design Pattern based code

Table 4: Execution time in seconds for MD Simulation

No. of Process/Threads	Pure OpenMP (in seconds)	Pure MPI (in seconds)	Hybrid MPI-Open MP (in seconds)	Adaptive Design Pattern (in seconds)
1	179.3	218.7	219.53	192.3
2	91.9	116.2	115.5	101.7
4	51.3	62.8	61.6	52.6
8	31.5	32.9	32.5	31.7
16	16.3	19.83	17.2	16.6
32	16.7	11.33	9.1	10
64	16.8	5.96	5.1	5.3
128	17.1	4.48	3.4	3.6

The study [45] uses SIMD vectorization for MD Simulation. The simulation is done using thread base parallelization on many core processors (Sandy Bridge and Haswell processors). The parallel efficiency obtained in [45] is around 78% while 88% in Adaptive Pattern based approach for four nodes. For more than 8 nodes parallel efficiency drops below 70% in both cases due to overhead of communication. In [46] short rage forces are considered for MD Simulation and for more than 32 cores parallel efficiency is more than 63% is maintained. The adaptive pattern-based solution provides 66% parallel efficiency in case of 32 nodes. The study [47] uses hybrid programming and shows 68% parallel efficiency. The experiment uses combination of MPI and Open MP simulating 5000,000-10,000,000 atoms. The recent study [48] on uses hybrid CPU–GPU architectures uses MPI-CUDA claims better parallel efficiency close to 80% than all previous approaches.

The proposed approach can deliver performance like SPMD in shared memory in case of a smaller number of cores. Adaptive pattern-based MD Simulation performs like Master Worker in case of large number of cores. The Pattern based code also provides separation of concerns and flexibility to switchover the code to various architectures. The capability of adding more features like fault tolerance, reducing halo can be achieved by accommodating relevant patterns. The proposed approach provides better parallel efficiency than [45], [46] while for large number of nodes [46] and [48] obtained better speedup and efficiency. The performance of Adaptive pattern-based approach still requires test on hybrid CPU–GPU architectures. The pattern-based approach provides facility to accommodate different patterns which are not exhibited in [45], [46], [47], [48].

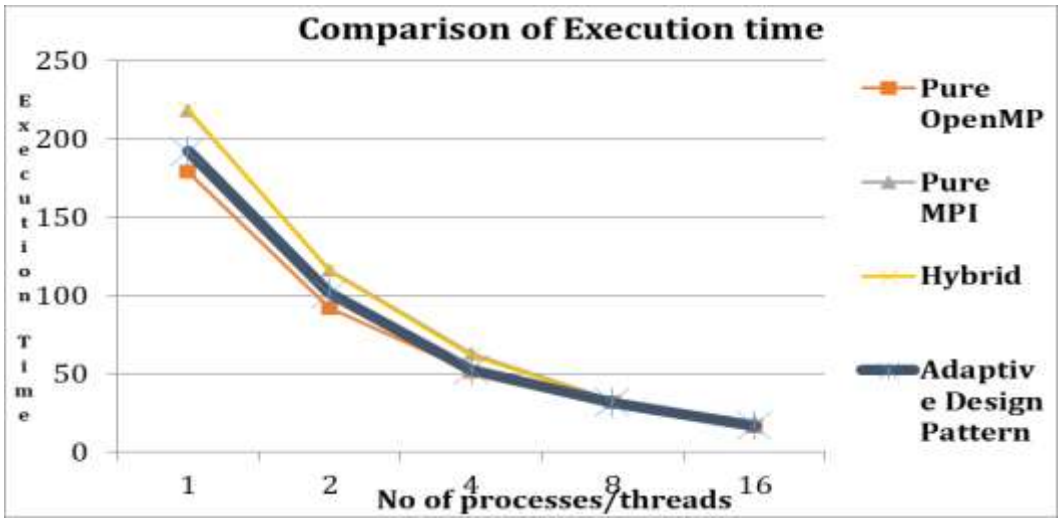

Figure 6: comparison of execution time in second with different approaches

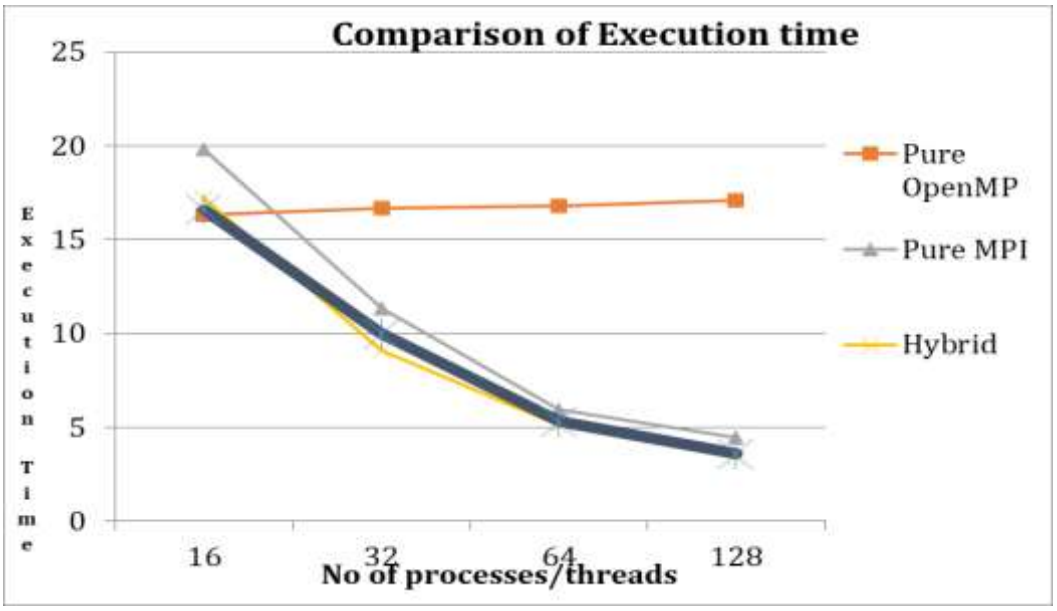

Figure 7: comparison of execution time in second with different approaches

7. Conclusion

MD simulation is useful for a wide range of applications. It requires high end computing facility. The MD simulation solution will be more flexible if it adapts different architectures. This study demonstrates Pure MPI code performs better in large number of processors and when problem size is large. Pure Open MP code is better if we have number of threads matches with the number of cores in processor. The handmade parallelized program serves as benchmark (LAMMPS) for comparison. The pattern-based design compared with the handmade parallelisation of OpenMP, MPI and Hybrid approach (MPI-Open MP) on various parameters execution time, latency, and flexibility to switchover code. Pattern based program is slightly slower than pure hybrid code due to increase Line of Code (LOC). The parallel efficiency obtained in this study is better in case of small number of processing elements (less than 32 cores) while competitive in case of large number of processing elements (more than 32 cores).

In this study, the efficiency of algorithm has not considered, as the objective is to show that design is adaptive to various architectures, and it can be flexible to add capabilities required in certain situations. The adaptive pattern-based approach shows better parallel efficiency in small number of processing elements. Hybrid programming with multilevel master worker architecture is applied cluster of nodes. The multilevel Master-Submaster-Worker pattern get inherent benefit of fault tolerance with dynamic load balancing facility.

The Adaptive Pattern proposed is suitable to problems which can map with Task Parallelism. The task specification is handled by programmer and incorrect task specification degrades performance. The experiment has only used Param Yuva –II architecture and still it need to apply on different architectures like Hybrid CPU -GPU and large number of parallel patterns. The work done motivates for development of framework which completely frees programmer from understanding the details of hardware, programming language constructs for parallelization.

8. Limitation and Future Work

The study has attempted to formulate pattern which is able to adapt environment and accommodate other patterns in design. The pattern is defined for considering problems mapped

with task parallelism. It may be extended or modified to support all kind of problems. The CUDA and GPU architectures need to experiment in context of parallel design patterns.
The pattern specified will be reusable if we transform them into components. The componentizable patterns are defined for common design patterns. Such components help in rapid application development and better quality.

The pattern in parallel programming still needs to experiment in different situation to provide pattern base from which best pattern for situation. If such concrete pattern base will be available in future. It will be utilized to develop a framework provide adaptive design for parallel software. The entirely new parallel programming framework is possible that makes use of parallel programming best practises. The formulation of such framework will be very much useful for parallel software designers. It will be also useful for parallel programmer for porting their application to various architectures.

9. Disclosure and conflict of interest

All authors declare that they have no conflicts of interest.

References

[1] J. M. Haile, I. Johnston, A. J. Mallinckrodt and S. Mckay, "Molecular Dynamics Simulation: Elementary Methods," *Computers in Physics*, vol. 7, no. 6, p. 625, 1993.

[2] Y. Shibuta, M. Ohno and T. Takaki, "Advent of Cross-Scale Modeling: High-Performance Computing of Solidification and Grain Growth," *Advanced Theory and Simulations*, vol. 1, no. 9, p. 180065, 2018.

[3] N. B. Gonzalo, N. Marcos, A. O. Miguel and S. Alberto, "MDScale: Scalable multi-GPU bonded and short-range molecular dynamics," *Journal of Parallel and Distributed Computing*, vol. 157, pp. 243-255, 2021.

[4] T. A. Abed Alhussien and H. Y. Fadhil, "Analysis of Mutations in Conserved and Susceptible Regions Across the Whole Genome Sequencing Analysis for SARS-CoV-2 in Iraqi Patients," *Iraqi Journal of Science*, vol. 64, no. 1, pp. 56-64, 2023.

[5] S. M. Hussein , F. A. Abdul Jabbar and H. M. Khalaf, "Detection of Genetic Polymorphism of HER2 Gene in HER2 Positive Breast Cancer Women in Iraq," *Iraqi Journal of Science*, vol. 62, no. 10, p. 3507–3520, 2021.

[6] J. N. Gaaib, "Prediction of Deleterious Non-Synonymous Single Nucleotide Polymorphisms (Nssnps) of Human TLR7 Gene," *Iraqi Journal of Science*, vol. 63, no. 6, pp. 2444-2452, 2022.

[7] K. Raymond and C. Giovanni , "Molecular dynamics: an account of its evolution," in *Theory and Applications of Computational Chemistry: The First Fory Years*, Elsevier , 2005, pp. 425-441.

[8] M. Dobson, I. Fox and A. Saracino, "Cell list algorithms for nonequilibrium molecular dynamics," *Journal of Computational Physics*, vol. 315, pp. 211-220, 2016.

[9] T. R. Law, J. Hancox, S. A. Wright and S. A. Jarvis, "An algorithm for computing short-range forces in molecular dynamics simulations with non-uniform particle densities," *Journal of Parallel and Distributed Computing*, vol. 130, pp. 1-11, 2019.

[10] W. G. Hoover, "Molecular Dynamics," in *Lecture Notes in Physics(LNP)*, Springer, 1986, pp. 1-41.

[11] A. Hospital, J. R. Goni, M. J. Orozco and J. L. Gelpi, "Molecular dynamics simulations: Advances and applications," *Advances and Applications in Bioinformatics and Chemistry*, vol. 15, pp. 37-47, 2022.

[12] X. Hu, Z. Zeng, J. Zhang, D. Wu, H. Li and F. Gen, "Molecular dynamics simulation of the interaction of food proteins with small molecules," *Food Chemistry*, vol. 405, p. 134824, 2023.

[13] N. Thangavel and M. Albratty, "Benchmarked molecular docking integrated molecular dynamics stability analysis for prediction of SARS-CoV-2 papain-like protease inhibition by olive secoiridoids," *Journal of King Saud University - Science*, vol. 35, no. 1, p. 102402, 2023.

[14] A. F. Pina, S. F. Sousa, L. Azevedo and J. Carnei, "Non-B DNA conformations analysis through molecular dynamics simulations," *Biochimica et Biophysica Acta (BBA) - General Subjects,* vol. 1866, no. 12, p. 130252, 2022.

[15] A. K. Padhi, M. Janežič and K. Y. Zhang, "Chapter 26 - Molecular dynamics simulations: Principles, methods, and applications in protein conformational dynamics," in *Advances in Protein Molecular and Structural Biology Methods*, Eds. Academic Press, 2022, pp. 439-454.

[16] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier and A. Kohlymeyer, "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Computer Physics Communications,* vol. 271, p. 108171, 2022.

[17] P. S. Pacheco and M. Malensek, "Chapter 5 - Shared-memory programming with OpenMP," in *Introduction to Parallel Programming (Second Edition)*, Morgan Kaufmann, 2022, pp. 221-289.

[18] M. Brinskiy, M. Lubin and J. Dinan, "Chapter 16 - MPI-3 Shared Memory Programming Introduction," in *High performance parallelism pearls volume Two - multicore and many-core PR*, Elsevier Science & Technology, 2015, pp. 305-319.

[19] R. E. Grant and S. L. Olivier, "Chapter 6 - Networks and MPI for cluster computing," in *Topics in Parallel and Distributed Computing*, Morgan Kaufmann, 2015, pp. 117-153.

[20] I. Gelado and J. Cabezas , "Chapter 18 - Programming a heterogeneous computing cluster," in *Programming massively parallel processors: A hands-on approach, Third. Ed.*, Cambridge, MA, United States, Morgan Kaufmann, 2017, p. 387–441.

[21] M. Brinskiy, M. Lubin and J. Dinan, "Chapter 16 - MPI-3 Shared Memory Programming Introduction," in *High performance parallelism pearls volume Two - multicore and many-core PR*, Elsevier Science & Technology, 2015, p. 305–319.

[22] T. G. Mattson, B. A. Sanders and B. L. Massingill, Patterns for Parallel Programming, Addison-Wesley Professional, 2005.

[23] K. Keutzer, , B. L. Massingill, T. G. Mattson and B. A. Sanders, "A Design Pattern Language for Engineering (Parallel) Software: Merging the PLPP and OPL Projects," in *Workshop on Parallel Programming Patterns,*, Carefree, Arizona, US, 2010.

[24] "A Pattern Language for Parallel Programming ver2. 0, ParLab Patterns Wikipage .," berkeley.edu, 2012. [Online]. Available: http://parlab. eecs. berkeley. edu/wiki. [Accessed 02 January 2023].

[25] S. Siu and A. Singh, "Design patterns for parallel computing using a network of processors," *Proceedings. The Sixth IEEE International Symposium on High Performance Distributed Computing* , p. 293–304, 1997.

[26] J. Y. Wan, Y. Q. Sun and J. Y. Xue, "Expanding design pattern to support parallel programming," in *36th International Conference on Technology of Object-Oriented Languages and Systems*, TOOLS-Asia, 2000.

[27] D. Buono, M. Danelutto, S. Lametti and M. Torquati, "Parallel Patterns for General Purpose Many-Core," in *21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, 2013.

[28] B. Catanzaro and K. Keutzer, "Parallel Computing with Patterns and Frameworks," *XRDS ACM*, pp. 22-27, September 2010.

[29] A. Holzinger, K. H. Struggl and M. Debevc, "Applying Model-View-Controller (MVC) in Design and Development of Information Systems: An example of smart assistive script breakdown in an e-Business Application," in *International Conference on E-Business (ICE-B 2010)*, 2020.

[30] M. Voss, R. Asenjo and J. Reinders, 'Mapping Parallel Patterns to TBB', in Pro TBB: C++ Parallel Programming with Threading Building Blocks, Berkeley, CA: Apress, 2019.

[31] S. Amrasinghe, *Multicore Programming Primer,* MIT OpenCourseware, 2007.

[32] N. Maltare and C. Chudasama,, "Experimenting Large Prime Numbers Generation in MPI Cluster," in *International Congress on Information and Communication Technology*, 2016.

[33] N. Maltare and V. Kamat, "Applying Parallel Design Patterns on Molecular Dynamics Simulation," *International Journal of Computer Applications,* vol. 181, no. 50, pp. 21-24, 2019.

[34] P. González, R. R. Osorio, X. C. Pard, J. R. Banga and R. Doallo, "An efficient ant colony optimization framework for HPC environments," *Applied Soft Computing,* vol. 114, p. 108058, 2022.

[35] L. Ng, J. E. Garcia and A. G. Dyer, "Mission impossible: honeybees adjust time allocation when facing an unsolvable task," *Animal Behaviour,* vol. 182, pp. 59-66, 2021.

[36] E. Y. Nakagawa, F. C. Ferrari, M. M. F. Sasaki and J. C. Maldonado, "An aspect-oriented reference architecture for Software Engineering Environments," *Journal of Systems and Software,* vol. 84, no. 10, pp. 1670-1684, 2011.

[37] R. Chitchyan and I. Sommerville, "AOP and Reflection for Dynamic Hyperslices," in *RAM-SE'04-ECOOP'04, Workshop on Reflection, AOP, and Meta-Data for Software Evolution, Proceedings*, Oslo, 2004.

[38] L. Jicheng, "The implementation of template method pattern by aspect based on configuration file," in *10th International Conference on Computer Science & Education (ICCSE)*, 2015.

[39] D. M. Beazley, P. S. Lomdahl, N. Grønbech-Jensen, R. Giles and P. Tamayo, "PARALLEL ALGORITHMS FOR SHORT-RANGE MOLECULAR DYNAMICS," World Scientific, 1996.

[40] S. Plimpton, "Fast Parallel Algorithms for Short-Range Molecular Dynamics," *Journal of Computational Physics,* vol. 117, no. 1, pp. 1-19, 1995.

[41] K. Tarmyshov and F. Müller-Plathe, "Parallelizing a Molecular Dynamics Algorithm on a Multiprocessor Workstation Using OpenMP," *Journal of chemical information and modeling,* vol. 45, no. 11, pp. 1943-1952, 2005.

[42] M. Kunaseth, R. K. Kalia, A. Nakano, K. I. Nomura and P. Vashishta, "A Scalable Parallel Algorithm for Dynamic Range-Limited n-Tuple Computation in Many-Body Molecular Dynamics Simulation," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Denver, Colorado, 2023.

[43] P. Malczyk, J. Frączek, F. González and J. Cuadrado, "Index-3 divide-and-conquer algorithm for efficient multibody system dynamics simulations: theory and parallel implementation" , 727–747, 2019.," *Nonlinear Dynamics,* vol. 95, pp. 727-747, 2019.

[44] J. Christopher, R. D. Falgout, J. B. Schroder, S. M. Guzik and S. Gao, "A space-time parallel algorithm with adaptive mesh refinement for computational fluid dynamics," *Computing and Visualization in Science,* vol. 23, no. 13, 2020.

[45] C. M. Mangiardi and R. Meyer, "A hybrid algorithm for parallel molecular dynamics simulations," *Computer Physics Communications,* vol. 219, p. 196–208, 2017.

[46] T. D. Nguyen, "GPU-accelerated Tersoff potentials for massively parallel Molecular Dynamics simulations," *Computer Physics Communications ,* vol. 212, pp. 113-122, 2017.

[47] L. Kenli, L. Dapu, L. Jie, Y. Yu, L. Yingqiang, L. Rangsu and M. Yunfei, "Performance analysis of parallel algorithms in physics simulation for molecular dynamics simulation liquid metals solidification processes," *Computers & Fluids,* vol. 110, pp. 19-26, 2015.

[48] J. Castagna, X. Guo, M. Seaton and A. O. Cais, "Towards extreme scale dissipative particle dynamics simulations using multiple GPGPUs," *Computer Physics Communications,* vol. 251, p. 107159, 2020.