ISSN: 2584-0495 Vol. 2, Issue 9, pp. 1162-1169

International Journal of Microsystems and IoT

ISSN: (Online) Journal homepage: https://www.ijmit.org

# MPI for SPH Methods and Parallel Computing on CPUs and GPUs

**Premkumar S J Nayaka, Sunil Kumar, Aniket Singh, Mohammad Sohail**

**Cite as:** Nayaka, P. S. J., Kumar, S., Singh, A., & Sohail, M. (2024). MPI for SPH Methods and Parallel Computing on CPUs and GPUs. International Journal of Microsystems and IoT, 2(9), 1162-1169. https://doi.org/10.5281/zenodo.14066770



Published online: 23 Sept 2024



**DOI:** https://doi.org/10.5281/zenodo.14066770

International Journal of Microsystems and IoT
Vol. 2, Issue 9, pp.1162-1169; DOI: https://doi.org/10.5281/zenodo.14066770

# MPI for SPH Methods and Parallel Computing on CPUs and GPUs

Premkumar S J Nayaka, Sunil Kumar, Aniket Singh, Mohammad Sohail

Maharshi Patanjali CPS Lab, BRICS Lab, Department of Computer Science and Engineering, National Institute of Technology, Karnataka, Mangalore, India

**ABSTRACT**
Message Passing Interface (MPI) is a standard designed for parallel programming on distributed memory systems, enabling multiple processors to work together by dividing and distributing tasks. This paper presents a comprehensive approach to addressing computational challenges in smoothed particle hydrodynamics (SPH) simulations through a novel MPI-based parallel SPH code. The research emphasizes code optimization for both CPU and GPU architectures, incorporating CUDA parallel programming to enhance GPU performance. Detailed insights into the code design, implementation, algorithm flowchart, and multi-GPU usage with MPI are provided. Experimental results demonstrate the model's efficiency and scalability across various scenarios, laying a solid foundation for advancing research in fluid dynamics and parallel computing.

**KEYWORDS**
Message Passing Interface (MPI); Smoothed Particle Hydrodynamics (SPH); GPU; Parallel Computing; CUDA

## 1. INTRODUCTION

Parallel computing has revolutionized computational simulations, empowering researchers to address increasingly complex and large-scale problems [1-3]. The message passing interface is a standardized protocol crucial for this advancement, as it distributes computational tasks across multiple processors, enabling the simultaneous execution of intricate simulations. This capability is vital for scientific and engineering disciplines, where simulations often involve massive datasets and require significant processing power. By dividing tasks among processors, MPI alleviates computational bottlenecks, thus enhancing the efficiency of high-fidelity simulations.

Parallel to the evolution of MPI, smoothed particle hydrodynamics has emerged as a powerful method for modeling complex fluid flows and interactions. Applied initially to astrophysical simulations, SPH has gained widespread adoption in various fields, including computational fluid dynamics, biomechanics, and materials science [4]. The method's strength lies in its ability to simulate complex physical processes, especially those involving fluid dynamics and astrophysical phenomena. However, as SPH simulations increase in complexity and scale, the computational demands also grow significantly.

To address these challenges, this paper explores the integration of MPI to enhance the computational efficiency of SPH simulations. By parallelizing SPH simulations, MPI can help overcome the computational hurdles associated with large-scale simulations, allowing for more detailed exploration and analysis of fluid dynamics on an intricate side.



The subsequent sections cover the fundamentals of SPH, the challenges posed by growing simulation complexity, and the rationale for parallelizing SPH with MPI [5]. Details of the implementation process, performance improvements, and practical implications of integrating MPI into SPH simulations are provided. The goal is to showcase MPI's potential as a powerful tool for optimizing SPH computational efficiency, paving the way for more accurate and scalable simulations in fluid dynamics and related fields.

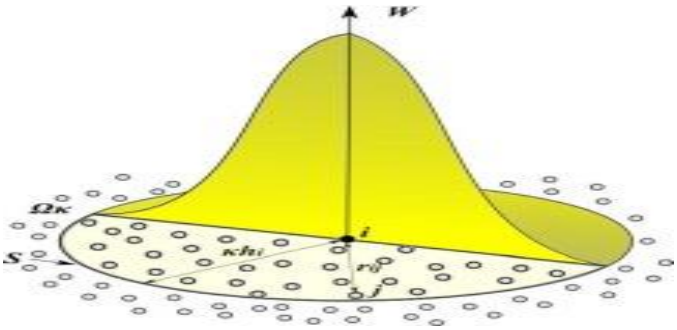

**Fig. 1** Particle approximations using a circular kernel.

Despite its advantages, SPH can be computationally expensive, especially for large-scale problems involving millions or billions of particles [6,7]. In SPH, particles represent fluids, with each particle's fluid properties determined through interactions with neighboring particles. This mesh-free approach is well-suited for complex geometries and large deformations. Still, it underscores the need for efficient computational strategies, such as those provided by MPI, to manage the intensive computational demands [8-10].

The smoothed particle hydrodynamics addresses the governing partial differential equations using two essential processes: kernel approximation and particle approximation. In kernel approximation, the function values from nearby particles are summed to estimate integrals. Particle approximation utilizes a collection of particles, each containing field variable data, to model the computational domain, as depicted in Figure 1.

The rest of the paper is structured as follows: Section 2 provides related work. Section 3 details the proposed methodology. Section 4 presents experimental results. Finally, Section 5 concludes the paper.

## 2. RELATED WORKS

Research in the field of Smoothed Particle Hydrodynamics and its computational challenges has been extensive. Numerous studies have focused on improving SPH simulations' accuracy, stability, and efficiency. The integration of parallel computing techniques, particularly the use of MPI, has been a major area of interest to address the computational demands of large-scale simulations.

Barcarolo et al. [11] introduced adaptive particle refinement (APR) and dynamic de-refinement techniques to enhance the accuracy and efficiency of SPH simulations. APR involves refining particle resolution in critical regions, while de-refinement reduces resolution in less essential areas, aiming for better computational efficiency and accuracy. However, the de-refinement technique is still under development, facing challenges such as the need for effective flow stabilization and managing varying rates of refinement and de-refinement. These issues can affect overall simulation stability and accuracy, making it necessary to refine these techniques further.

Domínguez et al. [12] developed a multi-GPU implementation for SPH on heterogeneous clusters, leveraging the computational power of both CPUs and GPUs to achieve high performance. The approach utilized parallel programming languages such as MPI, CUDA, and OpenMP, incorporating dynamic load balancing and overlapping data communication to optimize performance. The implementation demonstrated significant improvements in efficiency and scalability, handling various particle numbers and GPU configurations. However, its heavy reliance on modern high-performance computing architectures might limit its applicability in less specialized or conventional systems. Efforts to adapt this implementation for a broader range of hardware configurations could enhance its usability.

Holmes et al. [13] proposed a comprehensive framework for parallel computational physics algorithms on multi-core architectures. Their method addresses challenges in achieving efficient parallel computing within shared-memory environments, focusing on mitigating race conditions and optimizing memory usage. The framework includes a spatial sub-division algorithm that ensures thread safety by operating on isolated memory blocks. Microsoft's concurrency and coordination runtime (CCR) facilitates multi-core programming, allowing for the effective management of parallel tasks. Despite its innovative design, the framework's complexity and focus on shared-memory systems might limit its application in distributed memory environments where alternative strategies might be necessary.

Oger et al. [14] presented an MPI-based parallelization scheme specifically for high-performance computing (HPC) environments using the DualSPHysics code. They focused on enhancing the scalability and efficiency of SPH simulations across various test problems. The implementation achieved notable scalability improvements, facilitating efficient simulations on large HPC clusters. However, the scheme's dependence on MPI as the primary communication protocol could restrict its flexibility, particularly when integrating with other parallelization methods or adapting to less sophisticated computing infrastructures. Future work may explore hybrid approaches to address these limitations.

Liu et al. [15] introduced an MPI-based parallel framework designed for SPH simulations of extreme mechanic problems, such as high-speed impacts and large deformations. The framework emphasizes optimized memory management, including cache-friendly data storage and spatial/temporal locality improvements. A two-step load-balancing method enhances parallel efficiency, achieving a maximum efficiency of 97% on 10,020 CPU cores. While this framework effectively addresses extreme conditions and achieves high performance, its specialization and substantial computational resource requirements may limit its applicability to less intensive simulations. Broader applicability might require adjustments to handle a wider range of scenarios.

Zhu et al. [16] proposed a novel MPI-based parallel framework to address computational challenges in SPH for modeling free surface flows. This framework incorporates domain decomposition, a regular background grid, and an index ordering method to enhance scalability on high-performance computing systems. A dynamic load-balancing strategy is introduced to optimize computational load distribution based on particle numbers and running times. Although this framework significantly reduces computational costs for large-scale simulations, its complexity and dependence on HPC systems could limit its use in smaller-scale or non-HPC environments. Adapting this framework to a broader range of computing environments could expand its utility.

Antonelli et al. [17] developed a CUDA-based implementation of an enhanced SPH method on GPUs to improve accuracy and computational efficiency for fluid dynamics simulations. Their approach uses a Taylor series expansion to refine the SPH method, achieving substantial speedups and enhanced accuracy through parallelization on NVIDIA GPUs. While the method demonstrates impressive performance improvements, its reliance on CUDA and specific NVIDIA GPU architectures limits its general applicability. Systems without NVIDIA hardware might not benefit from these advancements, suggesting a need for alternative implementations compatible with different hardware platforms.

Li et al. [18] designed "petaPar," a scalable petascale framework for meshfree and particle simulations, including SPH and the material point method (MPM). Implemented within a unified object-oriented structure, petaPar uses MPI and Pthreads for

dynamic load balancing and fully overlapping communication and computation. Evaluated on various HPC platforms, including the Titan supercomputer, petaPar demonstrated excellent scalability. However, its current design is optimized for CPU-based platforms, with plans to extend support to GPU-based systems. The framework's immediate applicability to GPU environments is limited, though its expansion could enhance its utility across diverse computing platforms.

**Table. 1** MPI-Based Parallel Computing

| Author | Approach | Metrics | Limitation |
|---|---|---|---|
| Barcarolo et al. [11] | Adaptive Particle Refinement and Coarsening method in SPH | Numerical Validation | Flow stabilization requirement, Derefinement technique under develop |
| Domínguez et al. [12] | Multi-GPU Implementation for SPH on Heterogeneous Clusters using MPI, CUDA, OpenMP | Efficiency and Scalability on Different Clusters | Only available for heterogeneous clusters |
| Holmes et al. [13] | Spatial Sub-Division Algorithm, Load-Balanced Spatial Distribution for Parallel Computing in MultiCore Shared-Memory Environment | Scalability, Memory Efficiency, Load Balancing, Robustness | Bottleneck in Distributed Computing, Thread Safety Challenges, Complexity in Implementation |
| Oger et al. [14] | Distributed Memory MPI-Based Parallelization Scheme for SPH in DualSPHysics | Scalability, Efficiency on Test Problems | Only implemented in DualSPHysics code |
| Liu et al. [15] | MPI-based Parallelization, Memory Management, Load Balancing for SPH | Numerical Validation, Scalability Tests | Hardware Dependence, Complexity, Specific Use Cases, Computational Resources |
| Zhu et al. [16] | Parallel MPI Framework Incorporating Regular Background Grids and Dynamic Load Balancing for SPH Simulations. | Benchmarks, Numerical Experiments, Parallel Performance | Specificity to SPH, Computational Costs, Hardware Dependence, Complexity |
| Antonelli et al. [17] | CUDA Implementation of Improved SPH Method | Runtime Evaluation, Water Entry, Shock Wave Interaction | GPU-specific implementation |
| Li et al. [18] | PI and P-threads for Overlapping Communication and Computation, Dynamic Load Balancing in Particle Simulation | Scalability Evaluation on HPC Platforms | CPU-based platform, potential GPU extension |

These studies underscore the importance of integrating advanced computational strategies to enhance SPH simulation performance. This paper builds on these foundational works by introducing a novel MPI-based parallel SPH code optimized for CPU and GPU architectures, providing detailed insights into the implementation and performance benefits mentioned in Table 1.

## 3. PROPOSED METHODOLOGY

This section focuses on developing a new MPI-based parallel SPH code designed to run on both CPUs and GPUs. The goal is to address the computational challenges faced by existing MPI-based parallel SPH codes, which are primarily designed for CPU-based computers. GPUs can accelerate SPH simulations significantly, but their programming complexity poses a hurdle. The proposed work aims to bridge this gap and achieve an efficient distribution of the computational load between CPUs and GPUs.

### 3.1 Code Design and Implementation

A new MPI-based parallel SPH code has been designed and optimized for CPU and GPU architectures. This involves exploring techniques for efficient load balancing, data communication, and task distribution to harness the parallel processing capabilities of both types of processors. The implementation leverages the CUDA parallel programming architecture to utilize GPU capabilities efficiently. The three core processes of SPH—neighbor search, force calculation, and system updates—are parallelized on the GPU through the use of execution threads. After transferring particle data from the CPU to the GPU, all particle information is retained on the GPU for the entire duration of the simulation [4] [19]. Occasional data transfers from GPU to CPU are performed when saving simulation information, minimizing computationally expensive data transfers. The GPU implementation utilizes a neighbor list analogous to those employed in serial CPU approaches, such as the cell-linked list. CUDA's radix-sort algorithm enhances the parallelization of operations involved in constructing the cell-linked list [20]. Particle properties are updated using separate execution threads on the GPU, and CUDA's reduction algorithm further optimizes parallelization for certain tasks. The most computationally expensive part of particle interaction uses one thread per particle to calculate the forces arising from interactions with neighboring particles.

### 3.2 Algorithm Flowchart

The flowchart depicted in Figure 2 visually represents the sequential order of these steps and their links to one another.

#### 3.2.1 Initialize MPI Processes

Enable parallel computation, set up communication channels among processes, and determine the total number of processes in the MPI communicator [20].

#### 3.2.2 Initialize SPH Parameters

Set fundamental parameters for SPH simulation, including particle properties, simulation domain specifications, and the time step definition.

#### 3.2.3 Domain Subdivision

Divide the physical domain into subdomains, assigning each to a specific MPI process. Ensure adjacent MPI processes handle neighbouring subdomains for efficient communication during force calculations [21].

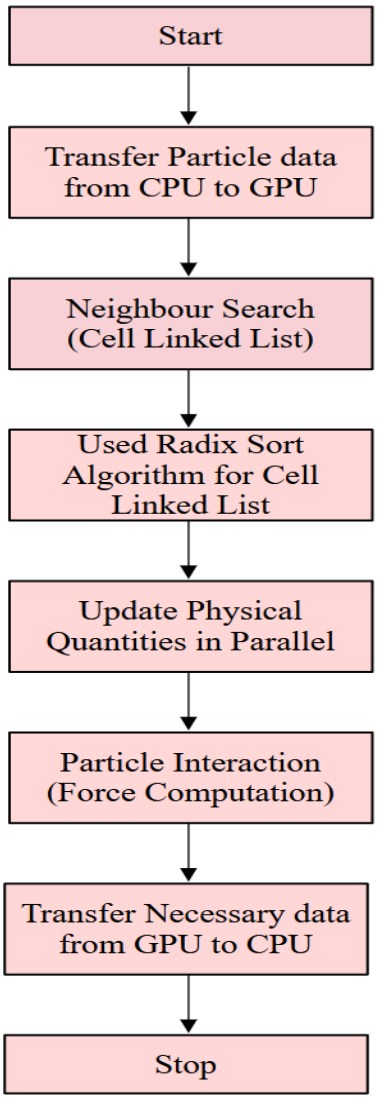


**Fig. 2** Abstract view of Proposed Framework

#### 3.2.4 Overlapping Communication and Computation

Implement a strategy for overlapping communication and computation to minimize idle time. Optimize force calculations to occur concurrently with data exchanges between MPI processes, enhancing overall simulation efficiency.

#### 3.2.5 Load Balancing

Incorporate a load balancing strategy to distribute the computational load evenly. This may involve dynamically adjusting the time step based on the workload of each MPI process, minimizing computational time differences.

#### 3.2.6 Compute Density and Kernel Approximation

Calculate particle density using the SPH density equation and kernel approximation for particles within the assigned subdomain. Sum contributions from neighboring particles within the kernel support an accurate density estimate.

#### 3.2.7 Compute Momentum

Utilize the momentum equation to compute particle acceleration, considering pressure, density, gravitational effects, and viscous terms. This comprehensive evaluation accounts for particle motion.

#### 3.2.8 Update Density

Update particle density based on the continuity equation. Sum contributions from neighboring particles within the interaction distance to reflect changes in density due to particle motion and interaction.

#### 3.2.9 Compute Pressure

Tait's equation of state is applied to calculate the pressure of particles. Evaluate the equation to determine pressure based on the updated density [22].

### 3.3 Multi-GPU Implementation with MPI

A secondary level of parallelization is achieved through the use of MPI, which enables inter-device communication and the integration of resources across multiple machines. The simulation's spatial domain is segmented into subdomains, with each MPI process responsible for managing and processing a portion of the particle data within its respective subdomain. This setup allows for effective computation across heterogeneous clusters, optimizing the utilization of all available processing resources.

#### 3.3.1 Domain Subdivision

The physical domain is subdivided into subdomains, with each subdomain assigned to separate MPI processes. This subdivision scales the simulation to expand proportionally with the number of machines employed. The main factors contributing to inefficiency, such as data exchange and synchronization, are addressed through careful domain subdivision and load-balancing strategies.

#### 3.3.2 Subdivision Strategy

The domain is partitioned into particle blocks, with each block assigned to a different MPI process. Each subdomain interacts with two neighboring subdomains, except those located at the edges of the simulation box. Data exchange between processes is necessary to calculate forces, and each process needs to obtain data on neighboring particles within the interaction distance (halo). Figure 3, as mentioned in [23], illustrates the division of a domain into three subdomains (0, 1, and 2) and highlights the halo concept, essential for force calculations.

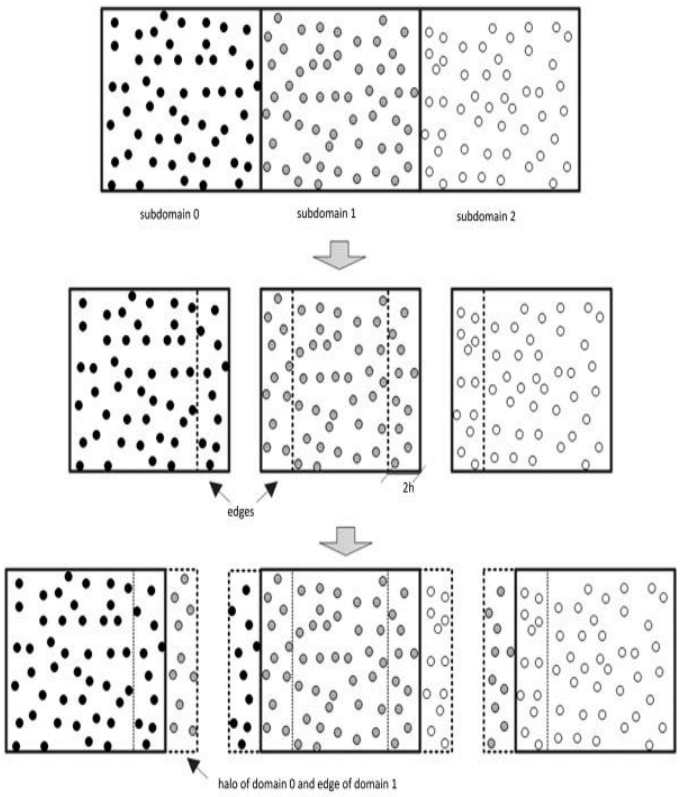


**Fig. 3** Domain subdivision into three subdomains with halo regions

#### 3.3.3 Overlapping Communication and Computation

To reduce communication time, forces on each subdomain are calculated so that communications and computations overlap. This optimization aims to enhance efficiency, especially with an increasing number of MPI processes.

#### 3.3.4 Load Balancing

Load balancing is crucial for efficiency, and a variable time step computed following a specified method is employed. This variable time step minimizes the difference in computation time required for the fastest and slowest processes, mitigating efficiency loss during synchronizations.

### 3.4 Algorithm Description

The proposed MPI-based parallel SPH algorithm utilizes the following equations to simulate fluid dynamics efficiently. It briefly overviews the fundamental equations, and their application in the algorithm is discussed below.

#### 3.4.1 Density and Kernel Approximation

The density function $\rho(x)$ in equation 1 is discretized using the SPH method. The discrete form based on particles is expressed as follows, where the approximation is interpolated at a particle i, and a summation is executed over all particles located within the kernel's region of compact support.

$$F(r_a) \approx \sum_b F\,(r_b) W(r_a - r_b, h) \Delta v_b \qquad (1)$$

Here, $m_j$ is the mass of particle $j$, and $W$ is the kernel function. If $W(x_i - x_j, h)$ is chosen such that $W = 1$ when $x_i - x_j$, with $m_j - p_j$, equation 2 becomes:

$$F(r_a) \approx \sum_b F\,(r_b) \frac{m_b}{\rho_b} W(r_a - r_b) \qquad (2)$$

#### 3.4.2 Momentum Equation

The momentum equation in equation 3, as proposed by [4], is employed to determine the acceleration $(a_i)$ of a particle $(i)$ resulting from interactions with its neighbors $(j)$:

$$\frac{dv_a}{dt} = -\sum_b m_b \left( \frac{p_b}{\rho_b^2} - \frac{p_a}{\rho_a^2} + \Pi_{ab} \right) \nabla_a W_{ab} + g \qquad (3)$$

Here $v_i$ is the velocity, $p_i$ is the pressure, $\rho_i$ is density, $m_i$ denotes mass, $g$ stands for gravitational acceleration, $\Pi_{ij}$ is the viscous term based on the artificial viscosity model as proposed in [4].

#### 3.4.3 Continuity Equation

The continuity equation or mass conservation principle, as expressed in equation 4, in SPH form [4], is used to compute change in fluid density:

$$\frac{d\rho_a}{dt} = -\sum_b m_b\, v_{ab} * \nabla_a W_{ab} \qquad (4)$$

#### 3.4.4 Pressure Calculation

Pressure $(p_a)$ is computed from density based on Tait's equation [17] of state, as specified in equation 5.

$$P_a = B\left( \left( \frac{\rho_a}{\rho_o} \right)^{\gamma} - 1 \right) \qquad (5)$$

Where $B$ is the constant, $\gamma$ is the adiabatic index, $\rho_0$ is the reference density, and the speed of sound $(c_o)$is determine by equation 6:

$$c_0 = (c\rho_0) = \sqrt{\left( \frac{\partial P}{\partial \rho} \right)_{\rho o}} \qquad (6)$$

#### 3.4.5 Temporal Integration Method

A variable time-step Verlet algorithm [24-25] is utilized for time integration, with the step size determined by the Courant–Friedrich–Levy (CFL) condition, the applied forces, and viscous diffusion, as outlined in [26].

### 3.5 MPI-Based Parallelization

The algorithm is parallelized using MPI, where the physical domain is categorized into subdomains assigned to separate MPI processes. Communication between devices is established, allowing the combination of resources from multiple machines. The load-balancing strategy is employed to minimize computational time differences among processes.

#### 3.5.1 Domain Subdivision

The domain is subdivided into particle blocks, each allocated to separate MPI processes. All subdomains are surrounded by two neighboring subdomains, except for those positioned at the boundaries of the simulation box. Data exchange between processes is necessary for force calculations, and each process needs to obtain data on neighboring particles within the interaction distance (halo).

#### 3.5.2 Overlapping Communication and Computation

To reduce communication time, force calculation for each subdomain is structured in a way that allows communication and computation to occur simultaneously. This optimization aims to enhance efficiency, especially with an increasing number of MPI processes.

#### 3.5.3 Load Balancing

It is crucial for efficiency, and a variable time step is used to minimize the difference in computation time between the fastest and slowest processes, mitigating efficiency loss during synchronizations.

## 4. EXPERIMENT RESULTS

This section presents the experimental outcomes from the smoothed particle hydrodynamics simulations. The simulations were performed on an 11th Gen Intel(R) Core(TM) i5-11500 @ 2.70GHz processor with six cores and 12 threads, using the mpic++ compiler on a linux ubuntu system. The focus was on evaluating the computational time required for executing SPH simulations across multiple cores. The simulation parameters were set to $T = 5.0, \Delta t = 0.0001, \wedge\, h = 0.01$, with no compiler optimizations applied.

Table 2 details the computational times, measured in milliseconds, for both serial and parallel implementations of the SPH simulations. These also include the number of processes used in the parallel simulations. These results provide a thorough assessment of the SPH model's efficiency and scalability, demonstrating its performance under the specified computational conditions.

### 4.1 Simulation Output Analysis

The SPH simulation was executed using the following command:$mpirun - np6\,./SPH_{main}\,.\,outicdroplet - -dt0.01 - -T5 - -h0.01$. This command utilizes six processes for parallel execution, simulates the droplet scenario $(- - ic - droplet)$, uses a time step $(- - dt)$ of 0.01, runs the simulation for a period $(- - T)$ of 5 seconds, and sets the particle radius of influence $(- - h)$ to 0.01 shown in Figure 4.

```
Rank = 2 | World Size = 4
Rank = 3 | World Size = 4
Rank = 1 | World Size = 4
Rank = 0 | World Size = 4
-- SPH Solver and Managers instantiated. --
-- Define simulation parameters.
Simulation Mode: ic-droplet.
Time Step, dt = 0.0001.
Simulation Period, t_end = 0.02.
Particle Radius of Influence, h = 0.01.

Definition completed. --
-- Setup initial condition of simulation.
No. of Global Particles, n_particle_G = 340
Rank = 0 | No. of Local Particle = 85
Rank = 0 | Global Index = 0
Rank = 1 | No. of Local Particle = 85
Rank = 1 | Global Index = 85
Rank = 2 | No. of Local Particle = 85
Rank = 2 | Global Index = 170
Rank = 3 | No. of Local Particle = 85
Rank = 3 | Global Index = 255
Initial condtion setup completed. --

-- Run SPH Simulation. --
-- Start Time Stepping.
Mass, m = 0.0785255
Time Stepping Completed. --

Time taken: 241 milli seconds
```

**Fig. 4** Performance of the SPH model under different settings

### 4.2 Performance Metrics

The simulations employed varying numbers of processes to evaluate the model's performance. Table 2 presents millisecond running times for the Dam-break and Droplet scenarios, measured under different parallel processing conditions. These simulations are executed in an HPC environment to assess the scalability and efficiency of the implemented model. The number of particles for each simulation is also indicated in Table 2.

The experimental results demonstrate significant improvements in computational efficiency with the MPI-based parallel SPH implementation. The adopted parallelization strategy effectively reduces computational times, as evidenced by the droplet simulation time decreasing from 2468 ms with a single process to 859 ms with six processes, an improvement factor of approximately 2.87. These results align with and extend the findings of others [5,8] who utilized multi-GPU and MPI techniques for SPH simulations. This implementation explicitly addresses overlapping communication and computation challenges and incorporates a load-balancing strategy to minimize computation time differences among processes. These enhancements position this work as a major contribution to the

field of SPH simulations, highlighting the effectiveness of MPI in enhancing computational performance in fluid dynamics and related fields.

**Table. 2** Simulation Performance Metrics (Times in (ms)

| No of Process | Dam Break (ms) | Droplet (ms) |
|---|---|---|
| 1 | 3292 | 2468 |
| 2 | 1907 | 1465 |
| 4 | 1273 | 1120 |
| 6 | 1118 | 859 |
| No of Particle | 400 | 340 |

## 5. CONCLUSION

This paper presents a novel MPI-based parallel SPH code designed to leverage CPU and GPU architectures, addressing the computational challenges of large-scale SPH simulations. Results show substantial performance improvements, with parallel implementations reducing computational times by up to 66% compared to serial approaches. The scalability of the approach is evident, with efficiency gains observed as the number of processes increased. For instance, the parallelization using six processes reduced the running time for the Dam-break scenario to 1118 milliseconds and the Droplet scenario to 859 milliseconds. These results highlight the potential of MPI-based parallelization to optimize SPH simulations, facilitating more detailed and accurate fluid dynamics analysis in complex scenarios. The implementation's capability to handle large particle counts efficiently opens avenues for further exploration and refinement, particularly in adapting the methodology to a broader range of computational environments.

## REFERENCES


1. Jin, H., Jespersen, D., Mehrotra, P., Biswas, R., Huang, L., & Chapman, B. (2011). High performance computing using MPI and OpenMP on multi-core parallel systems. *Parallel Computing*, *37*(9), 562-575. https://doi.org/10.1016/j.parco.2011.02.002
2. Kumar, S., & Bhowmik, B. (2024, January). Emergence, Evolution, and Applications of Cyber-Physical Systems in Smart Society. In *2024 Fourth International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT)* (pp. 1-8). IEEE. https://doi.org/10.1109/ICAECT60202.2024.10468864
3. Saxena, D., & Bhowmik, B. (2024, May). Analysis of Selected Load Balancing Algorithms in Containerized Cloud Environment for Microservices. In *2024 IEEE 4th International Conference on VLSI Systems, Architecture, Technology and Applications (VLSI SATA)* (pp. 1-6). IEEE. https://doi.org/10.1109/VLSISATA61709.2024.10560139
4. Monaghan, J. J. (2005). Smoothed particle hydrodynamics. *Reports on progress in physics*, *68*(8), 1703. https://doi.org/10.1088/0034-4885/68/8/R01
5. Jagtap, P., Nasre, R., Sanapala, V. S., & Patnaik, B. S. V. (2021). Efficient parallelization of SPH algorithm on modern multi-core CPUs and massively parallel GPUs. *International Journal of Modeling, Simulation, and Scientific Computing*, *12*(06), 2150054. https://doi.org/10.1142/S1793962321500549
6. Girish, K. K., Kumar, S., & Bhowmik, B. R. (2024). Industry 4.0: Design Principles, Challenges, and Applications. *Topics in Artificial Intelligence Applied to Industry 4.0*, 39-68. https://doi.org/10.1002/9781394216147.ch3
7. Yang, E., Bui, H. H., De Sterck, H., Nguyen, G. D., & Bouazza, A. (2020). A scalable parallel computing SPH framework for predictions of geophysical granular flows. *Computers and Geotechnics*, *121*, 103474. https://doi.org/10.1016/j.compgeo.2020.103474
8. Violeau, D., & Rogers, B. D. (2016). Smoothed particle hydrodynamics (SPH) for free-surface flows: past, present and future. *Journal of Hydraulic Research*, *54*(1), 1-26. *https://doi.org/10.1080/00221686.2015.1119209*
9. Hegde, A., & Bhowmik, B. (2024, April). Big Data Insights: Pioneering Changes in FinTech. In *2024 IEEE 9th International Conference for Convergence in Technology (I2CT)* (pp. 1-6). IEEE. https://doi.org/10.1109/I2CT61223.2024.10543820
10. Saxena, D., & Bhowmik, B. (2023, November). Paradigm shift from monolithic to microservices. In *2023 IEEE International Conference on Recent Advances in Systems Science and Engineering (RASSE)* (pp. 1-7). IEEE. https://doi.org/10.1109/I2CT61223.2024.10543820
11. Barcarolo, D. A., Le Touzé, D., Oger, G., & De Vuyst, F. (2014). Adaptive particle refinement and derefinement applied to the smoothed particle hydrodynamics method. *Journal of Computational Physics*, *273*, 640-657. https://doi.org/10.1016/j.jcp.2014.05.040
12. Domínguez, J. M., Crespo, A. J., Valdez-Balderas, D., Rogers, B. D., & Gómez-Gesteira, M. (2013). New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters. *Computer Physics Communications*, *184*(8), 1848-1860. https://doi.org/10.1016/j.cpc.2013.03.008
13. Holmes, D. W., Williams, J. R., & Tilke, P. (2011). A framework for parallel computational physics algorithms on multi-core: SPH in parallel. *Advances in Engineering Software*, *42*(11), 999-1008. https://doi.org/10.1016/j.advengsoft.2011.05.017
14. Oger, G., Le Touzé, D., Guibert, D., De Leffe, M., Biddiscombe, J., Soumagne, J., & Piccinali, J. G. (2016). On distributed memory MPI-based parallelization of SPH codes in massive HPC context. *Computer Physics Communications*, *200*, 1-14. https://doi.org/10.1016/j.cpc.2015.08.021
15. Liu, J., Yang, X., Zhang, Z., & Liu, M. (2024). A massive MPI parallel framework of smoothed particle hydrodynamics with optimized memory management for extreme mechanics problems. *Computer Physics Communications*, *295*,108970. https://doi.org/10.1016/j.cpc.2023.108970
16. Zhu, G., Hughes, J., Zheng, S., & Greaves, D. (2023). A novel MPI-based parallel smoothed particle hydrodynamics framework with dynamic load balancing for free surface flow. *Computer Physics Communications*, *284*,108608. https://doi.org/10.1016/j.cpc.2022.108608
17. Antonelli, L., Francomano, E., & Gregoretti, F. (2021). A CUDA-based implementation of an improved SPH method on GPU. *Applied Mathematics and Computation*, *409*, 125482. https://doi.org/10.1016/j.amc.2020.125482
18. Li, L., Wang, Y., Ma, Z., & Tian, R. (2014, August). petaPar: a scalable Petascale framework for meshfree/particle simulation. In *2014 IEEE International Symposium on Parallel and Distributed Processing with Applications* (pp. 50-57). IEEE. https://doi.org/10.1109/ISPA.2014.16
19. Khayyer, A., Gotoh, H., Shimizu, Y., & Gotoh, T. (2024). An improved Riemann SPH-Hamiltonian SPH coupled solver for hydroelastic fluid-structure interactions. *Engineering Analysis with Boundary Elements*, *158*, 332-355. https://doi.org/10.1016/j.enganabound.2023.10.018
20. Gao, J., Fan, H., Chen, G., Wang, W., & Zhang, H. (2024). Verification of 3D DDA-SPH coupling method and its application in the analysis of geological disasters. *Engineering Analysis with Boundary Elements*, *158*, 68-84. https://doi.org/10.1016/j.enganabound.2023.10.013
21. Pourlak, M., Akbari, H., & Jabbari, E. (2023). Importance of initial particle distribution in modeling dam break analysis with SPH. *KSCE Journal of Civil Engineering*, *27*(1), 218-232. https://doi.org//10.1007/s12205-022-0304-1
22. Chen, Y. K., Meringolo, D. D., & Liu, Y. (2024). SPH numerical model of wave interaction with elastic thin structures and its application to elastic horizontal plate breakwater. *Marine Structures*, *93*, 103531. https://doi.org/10.1016/j.marstruc.2023.103531
23. Domínguez, J. M., Crespo, A. J., Valdez-Balderas, D., Rogers, B. D., & Gómez-Gesteira, M. (2013). New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters. *Computer*

*Physics Communications*, *184*(8), 1848-1860. https://doi.org/10.1016/j.cpc.2013.03.008
24. Batchelor, G. K. (2000). *An introduction to fluid dynamics*. Cambridge university press.
25. Verlet, L. (1967). Computer" experiments" on classical fluids. I. Thermodynamical properties of Lennard-Jones molecules. *Physical review*, *159*(1), 98. https://doi.org/10.1103/PhysRev.159.98
26. Monaghan, J. J., & Kos, A. (1999). Solitary waves on a Cretan beach. *Journal of waterway, port, coastal, and ocean engineering*, *125*(3), 145-155. https://doi.org/10.1061/(ASCE)0733-950X(1999)125:3(145)

## AUTHORS:

**Premkumar S J Nayak** received his BTech degree from V.T.U, Karnataka, India in 2023. He is currently pursuing MTech at the Department of Computer Science and Engineering, National Institute of Technology Karnataka, Mangalore, India. His areas of interest are big data, deep learning and embedded systems.

E-mail: sjpremkumarnayaka.232cs030 @nitk.edu.in

**Sunil Kumar** received his BTech degree from A.K.T.U, Uttar Pradesh, India in 2016 and MTech degree in Computer Science and Engineering from National Institute of Technology Jalandhar, Punjab, India in 2019. He is currently pursuing PhD at the Department of Computer Science and Engineering, National Institute of Technology Karnataka, Mangalore, India. His research interest includes medical cyper-physical systems, formal verification, deep learning, medical imaging, embedded systems, artificial intelligence, computational intelligence and internet of things.

E-mail: sunilk.217cs010@nitk.edu.in

**Aniket Singh** received his BTech degree from Sagar Institute of Research and Technology, Madhya Pradesh, India in 2023. He is currently pursuing MTech at the Department of Computer Science and Engineering, National Institute of Technology Karnataka, Mangalore, India. His areas of interest are computer networking, big data, and computational intelligence.

E-mail: aniketsingh.232cs001@nitk.edu.in

**Mohammad Sohail** received his BTech degree from Madhav Institute of Technology and Science, Madhya Pradesh, India in 2020. He is currently pursuing MTech at the Department of Computer Science and Engineering, National Institute of Technology Karnataka, Mangalore, India. His areas of interest are big data, deep learning, and embedded systems.

E-mail: mohammadsohail.232cs018@nitk.edu.in