Multi-level Load Balancing Strategies for Massively Parallel Smoothed Particle Hydrodynamics Simulation

Yi Zhang*
University of Science and Technology of China
Hefei, China
yi_zhang@mail.ustc.edu.cn

Ziyu Zhang*
University of Science and Technology of China
Hefei, China
zzymm@mail.ustc.edu.cn

Yang Zhao
University of Science and Technology of China
Hefei, China
yang_zhao@mail.ustc.edu.cn

Junshi Chen†
University of Science and Technology of China
Hefei, China
Laoshan Laboratory
Qingdao, China
cjuns@ustc.edu.cn

Hong An
University of Science and Technology of China
Hefei, China
Laoshan Laboratory
Qingdao, China
han@ustc.edu.cn

Zhanming Wang
University of Science and Technology of China
Hefei, China
wangzmrr@mail.ustc.edu.cn

Longkui Chen
University of Science and Technology of China
Hefei, China
chenlk@mail.ustc.edu.cn

ABSTRACT

In the field of computational fluid dynamics, Smoothed Particle Hydrodynamics (SPH) serves as a powerful tool for investigating complex fluid interactions and instabilities. For the practical SPH simulation of large-scale fluid phenomena such as tsunamis, volcanic eruptions, and planetary collisions, it typically requires billions of particles, as the numerical resolution increases proportionally with the number of particles. To efficiently conduct large-scale SPH simulations on modern supercomputers with massive many-core processors, we propose a novel SPH implementation leveraging multi-level parallelism and a corresponding three-level load balancing strategy. Our load balancing approach comprises: (1) a process-level domain decomposition algorithm based on an improved 1D partitioning exact algorithm; (2) an adaptive recursive cell subdivision method; (3) a fine-grained dynamic thread-level task scheduling strategy. Our experiment uses 1 billion particles to simulate converging Richtmyer–Meshkov instability and verifies the effect of load balancing on new Sunway supercomputer. As the shockwave converges on the central interface area, our load balancing strategy breaks the bottleneck constraints on the slowest node, increases the balance of computational loads between nodes from 30.01% to 91.48%, and achieves a 2.8× improvement in computational performance. Finally, our implementation enables each CPU to handle 10 million particles and scale from 1 CPU to 100,000 CPUs (in total 39 million cores with 1 trillion particles) with a performance of 80.4% parallel efficiency.

CCS CONCEPTS

• **Applied computing** → *Physical sciences and engineering*; • **Computing methodologies** → **Parallel computing methodologies**; • **Computer systems organization** → **Heterogeneous (hybrid) systems**.

KEYWORDS

Load balance, Smoothed particle hydrodynamics, Many-core computing

ACM Reference Format:
Yi Zhang, Ziyu Zhang, Yang Zhao, Junshi Chen, Hong An, Zhanming Wang, and Longkui Chen. 2024. Multi-level Load Balancing Strategies for Massively Parallel Smoothed Particle Hydrodynamics Simulation. In *The 53rd International Conference on Parallel Processing (ICPP '24), August 12–15, 2024, Gotland, Sweden.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3673038.3673090

*These authors contributed equally to this work.
†Corresponding author

ICPP '24, August 12–15, 2024, Gotland, Sweden

ACM ISBN 979-8-4007-1793-2/24/08
https://doi.org/10.1145/3673038.3673090

1 INTRODUCTION

In the field of computational fluid dynamics, the study of strongly compressible fluid problems has always been a thorny topic. As a particle-based simulation method, Smoothed Particle Hydrodynamics (SPH) method, with its unique advantages, has become a powerful tool for studying such problems [30]. Usually, the SPH simulations of large-scale fluid phenomena typically require billions of particles to provide high resolution [15]. Modern supercomputers

offer powerful computational capabilities to support the extensive computational demands of SPH simulations. However, the significant migration and aggregation of particles during the simulation process result in vast differences in particle density across various regions of the computational domain, ranging from tens to hundreds of times. This extreme load imbalance poses challenges for large-scale parallel SPH simulations.

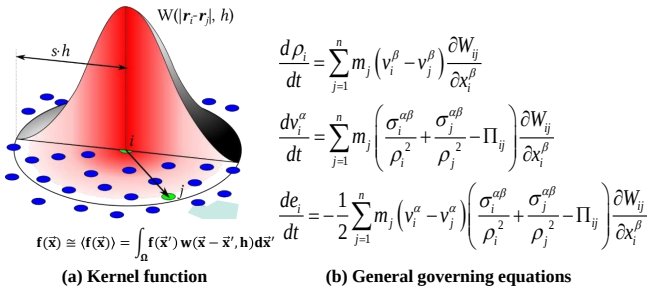
On the other hand, as the marginal effects of Moore's Law diminish, modern supercomputers are increasingly transitioning towards heterogeneous many-core architectures. China's Sunway supercomputer is equipped with over 600,000 SW26010pro processor cores, each featuring a parallel hierarchy composed of 6 management processing elements (MPE) and 384 computing processing elements (CPE). It presents a significant challenge to evenly distribute tasks among a vast number of cores and fully utilize the parallel hierarchy of the Sunway system. Moreover, given the dynamic nature of massive particle migration and aggregation occurring at the macroscopic scale, such as fluid instabilities under simulated converging shock wave [34], there will be serious load imbalance between node during the simulation [13]. Load imbalances among nodes can have critical impacts on the computational speed, scalability, and hardware resource utilization of applications.

Many dynamic load balancing methods [1, 2, 21–23, 32] have been proposed. However, in the context of strongly compressible SPH simulations, these approaches encounter the following issues: (1) The vast differences in particle density result in some tasks being overly burdensome, *significantly slowing down the overall computational time and adversely affecting the performance of load balancing algorithms*; (2) Many methods *coarsely partition tasks or compromise load balancing effectiveness* for efficiency, while some are too *time-consuming* to be suitable for our problem scale.

In this paper, we propose a multi-level load balancing strategy suitable for large-scale parallel SPH simulations. We introduce an adaptive recursive cell subdivision algorithm based on octrees [14] to solve the first issue. The subdivided child cells are used as scheduling units, reducing the granularity of task scheduling and enhancing the partitioning results of the partitioning algorithm. To solve the second issue, we reduce the load balancing problem in n-dimensional space to the 1D partitioning problem by employing the Hilbert space-filling curve mapping, facilitating rapid resolution and producing balance partitioning results in large-scale scenarios. We then solve the problem using our refined exact algorithm to achieve process-level task division and load balancing. This approach allows us to distribute computational tasks evenly across each process without constraints from spatial shapes, while preserving spatial locality to a greater extent [3, 26]. Our third-level strategy constructs task lists based on our neighbor search strategy designed for sub-cells at different depths and dynamically allocates tasks to each thread during runtime, allowing the thread-level task scheduling strategy to be adjusted based on the actual execution time of tasks.

The main contributions of this paper are as follows:

- We propose *a three-level load balancing strategy* that fully utilizes the parallel hierarchy of the Sunway system and finely divides tasks among a large number of nodes in a balanced manner.



(a) Kernel function **(b) General governing equations**

Figure 1: The SPH method employs discrete particles to describe the macroscopically continuous distribution of fluids and (a) utilizes kernel functions to approximate the physical problems. By discretizing the integration domain into a finite number of particles, (b) the governing and conservation equations in physics can be discretized into numerical equations, enabling numerical solutions.

- We propose *an improved exact algorithm*, *EXACT-BISECT+*, which effectively handles the numerous voids generated during SPH simulations and produces more balanced partitioning results than the original greedy implementation.
- To effectively addresses the issue of excessive task load discrepancies under extreme load imbalances and further produces finer-grained 1D partitioning results, we introduce *an adaptive recursive cell subdivision algorithm* based on octrees.
- We design *a dynamic task scheduling strategy* for thread-level load balancing and task parallelism.

The remainder of the paper is organized as follows: Sec. 2. introduces the numerical method of SPH, succinctly summarize the fundamental background of this study and discuss the challenges of massively parallel implementation of SPH on Sunway supercomputer with many-core architecture. Then the refined algorithm and implementation of our three-level load balancing strategy are discussed in Sec.3 and Sec.4. Subsequently, the experimental evaluation is presented in Sec.5. Eventually, Sec.6 shows the visualization of the simulation result and Sec.7 summarizes the main ideas of this work.

2 BACKGROUND

2.1 Smooth Particle Hydrodynamic Method

Smoothed Particle Hydrodynamics (SPH) is a popular mesh-free method and has potential to be the next generation of more effective computational methods for more complicated problems[20][25]. In SPH simulation, the computational domain is filled with a great number of "SPH particles". These particles all possess physical properties we concerned like pressure, density, energy. During the simulation, an SPH particle moves and refreshes its physical properties in each time step following the conservation law of mass, momentum and energy in Lagrangian form and equation of state of the particle's material, based on a set of nearby particles which forms its support domain as shown in Figure 1(b). This support domain is determined by each particle's location and volume, thus it always needs to be refreshed for each particle in each time step[19].

As for refreshing each particle's physical properties, the SPH method can be seen as replacing the Dirac delta function with a smoothed kernel W as shown in Figure 1(a). Thus an SPH approximate scalar function A can be defined as:

$$A(x) = \int A(x')W(x - x', h)\, dx'.$$

And the discretized form can be written as

$$\langle A \rangle_i = \sum_j^n A_j W_{ij} V_j,$$

where W_{ij} is $W(x_i - x_j, h_{ij})$, h is the smoothing length of the particle pairs, V is the volume, and the subscriptions i,j represents particle i and j.

Unlike the Eulerian method, the grid-free nature, adaptivity, and particle-based representation make SPH a versatile tool for handling complex fluid flows with large deformations and free surfaces. This flexibility allows SPH to simulate a wide range of phenomena, from astrophysical events to impact explosions and water dynamics. With the continued development of computing technologies, SPH is poised to play an increasingly important role in fluid dynamics research and applications.

2.2 Space-Filling Curves and the 1D Partitioning Problem

2.2.1 Domain Decomposition Method. The most time-consuming and frequent part of the particle-based methods is the pair-wise interaction. Due to the memory wall, the computing of the interactions on many-core processor are usually memory bounded. To improve the search efficiency, some implementations [10, 12, 31] use link-list-based methods to reduce ineffective searches by maintaining lists of nearby particles. However, this may lead to issues of indirect memory access. When CPEs calculates the interactions among a group of particles, it may require a large number of random memory accesses. This discrete access can seriously reduce the bandwidth utilization of DMA[6, 11]. Moreover, the significant displacement of numerous particles in SPH on the macro scale necessitates frequent reconstruction and maintenance of this type of neighbor lists, leading to considerable performance overhead. Therefore, we adopt the method based on cell lists to solve these problems. The entire computational domain is divided into contiguous cells, and particles are assigned to corresponding cells based on their spatial locations. Since particles within each cell are placed contiguously in memory, this method significantly improves memory locality during computations and the utilization of DMA bandwidth[11, 29]. Given that the sizes and positions of the cells are fixed, the neighbor relationships between cells do not change with particle migration, avoiding the overhead of rebuilding neighbor relationships.

Next, we map cells from 3D space to 1D space using the Hilbert space-filling curve (Hilbert SFC), ensuring that cells adjacent in the original computational domain remain close in the 1D space. This enhances memory locality and reduces the communication overhead for exchanging ghost regions between processes[3, 26]. The Hilbert SFC mapping transforms the problem of the domain decomposition and load balancing between processes into the 1D partitioning problem. The 1D partitioning method aims to divide a series of tasks among multiple partitions in order to balance the computational load across multiple processors. Each partition consists of a set of consecutive tasks, and our goal is to find a partitioning scheme that minimizes the load of the heaviest partition (corresponding to the processor with the longest execution time), thereby minimizing the overall execution time of the system.

2.2.2 Heuristic/Exact Solutions for 1D Partitioning Problem. There are numerous well-established algorithms for solving the 1D partitioning problem. Different algorithms exhibit variations in terms of execution time and the effectiveness of load balancing. Heuristic algorithms perform well, but the balance of the solutions obtained is susceptible to the influence of the input data. Oden et al. [22] proposed a heuristic based on *recursive bisection*, which divides the task array into two sub-partitions with equal loads and recursively repeats the process until the desired number of partitions is obtained. Miguet and Pierson [21] proposed two fully parallel heuristic methods, as the computations for sub-optimal partitions are independent of each other. In contrast, exact algorithms always obtain the optimal solution B_{opt}, but they exhibit lower performance compared to heuristic algorithms. Pınar et al. [24] provided a comprehensive overview of existing exact algorithms. Their experiments demonstrated that parametric-search algorithms exhibit the best performance. The idea behind such algorithms is to use a *PROBE* function to iteratively verify whether there exists a partition such that the load of the heaviest partition does not exceed the given bottleneck value B and derived more precise upper and lower bounds of B for next iteration, regardless of the success. Exact algorithms repeatedly adjust and verify based on the results of *PROBE* to obtain the optimal solution. The fastest exact algorithm proposed by Pınar et al. [24], denoted as *EXACT-BISECT*, enhances the performance of exact algorithms in three key aspects: (1) The probe function *RPROBE* searches for partition boundary in a narrowed range to achieve higher performance. (2) The search bounds for the optimal solution are adjusted to realizable bottleneck values after each iteration. (3) By providing a smaller initial search interval, the number of searching steps for the optimal solution is reduced. However, *EXACT-BISECT* has two issues that prevent its direct application to solving the load balancing problem in SPH simulations: (1) *EXACT-BISECT* may produce incorrect results when there are tasks with zero workload, which can easily occur during SPH simulations with extreme load imbalances. (2) *EXACT-BISECT* uses a naive *PROBE* algorithm to generate corresponding partitions based on the optimal solution, but the greedy nature of *PROBE* might lead to results that are not sufficiently balanced. To address these issues, we propose an improvement, called *EXACT-BISECT+*. Some works are dedicated to combining heuristic and exact algorithms to leverage their advantages. Lieber and Nagel [18] proposed a hierarchical algorithm, *Hier*, aimed at providing scalability for exact algorithms. *Hier* starts by running parallel heuristic algorithms to quickly partition tasks into $G(G \leq P)$ groups. Subsequently, within each group, we run an exact algorithm independently to partition each group into P/G partitions. However, load balancing occupies only a tiny portion of the total simulation time in SPH simulations with massive particle migration, and the effectiveness of load balancing is more crucial for performance enhancement.

2.2.3 Fine-Grained Task Scheduling on SW26010pro Many-Core Processor. Although *EXACT-BISECT+* can always find the optimal

solution, the optimal solution itself is influenced by the distribution of the workload. When massive particles gather in a small area as shown in Figure 11 (b.2), the density will be much greater than the sparse area, resulting in an extremely large computational load. In this case, even *EXACT-BISECT+* is helpless. Traditional cell-based methods cannot handle the vast differences in particle density within the computational domain. Hence, we propose an adaptive recursive cell subdivision method based on octree, capable of dynamically subdividing overly burdened cells into sub-cells and recursively refining them level by level until their load falls below the subdivision threshold. Subsequently, running the exact algorithm on all of the sub-cells produces finer-grained partitioning results, enhancing inter-process load balancing.

On the other hand, the threshold of cell subdivision can also take into account the relevant parameters of the computing core, which can not only generate sufficient parallelism for a large number of computing cores, but also divide the task to a proper size to fit in the local memory, thus improving the locality and utilization of data. To this end, we conducted experiments on the Sunway platform. Sunway supercomputer adopts a high-performance heterogeneous many-core processor SW26010pro. And the SW26010pro processor is composed of 6 core-groups (CGs), each of which includes one management processing element (MPE), and one 64 computing processing elements (CPEs) cluster arranged in an 8 by 8 grid, a total of 390 cores. In each CG, the MPE is in charge of spawning threads for the 64 CPEs and handle management and communication tasks. The CPE cluster is designed to provide high aggregated computing capability and each CPE has 256 KB scratch pad memory (SPM). The SPM can be configured as either user-controlled local data memory (LDM) or hardware cache for automatic data buffering. Data transfer between LDM and main memory can be realized by direct memory access (DMA). As the only local data memory on CPE, effective utilization of SPM is key to achieving high levels of parallelism.

In conclusion, each strive for performance portability, flexibility, and scalability across architecture is by providing optimized data structure, data layout, and data movement. While some previous implementations of SPH [4, 5, 7, 9, 16, 27] have achieved significant success, there is limited research addressing the deep memory hierarchy specific to heterogeneous many-core architectures. In order to improve the resource utilization and scalability of many-core systems, fine-grained task partitioning and multi-level parallel strategies are essential. In addition, due to the phenomenon of massive migration and aggregation of particles unique to the strongly compressible model, the design of a load balancing strategy is crucial. It not only needs to satisfy the load balance among CPUs but also needs to address the load balance among the 384 cores within each CPU. Therefore, multi-level load balancing strategies should complement multi-level parallel strategies.

3 THE 1D PARTITIONING PROBLEM

3.1 Problem Definition

In the 1D partitioning problem, we need to divide a weight array $(w_0, w_1, \ldots, w_{N-1})$ representing the computational workload of N tasks into P partitions. The algorithm to solve this problem should produce a partition $\mathbf{P} = (s_0, s_1, \ldots, s_P)$, where s_p, for $p = 0, 1, \ldots, P-1$, denotes the index of the first task assigned to partition p. Specifically, we define $s_P = N$. Each partition must contain consecutive tasks, meaning partition p consists of tasks $s_p, s_p + 1, \ldots, s_{p+1} - 1$. We define the load L_p of partition p as the sum of the weights of all tasks within it, i.e., $L_p = \sum_{i=s_p}^{s_{p+1}-1} w_i$, thereby L_p characterizes the execution time of the processor owning partition p. We can also compute the prefix sum array W of the weight array w, that is, $W_i = \sum_{j=0}^{i} w_j$, which allows us to efficiently calculate the load of partition p by $L_p = W_{s_{p+1}-1} - W_{s_p-1}$ in $O(1)$ time. The maximum load of all partitions, $B(\mathbf{P}) = \max_{0 \le p < P} L_p$, is defined as the bottleneck of partition $\mathbf{P}$, since the execution time of the entire system depends on the execution time of the partition with the maximum load.

The goal of the 1D partitioning problem is to find the optimal partition $\mathbf{P}_{\text{opt}}$ with the minimum bottleneck value $B_{\text{opt}} = B(\mathbf{P}_{\text{opt}})$. When the loads of all partitions are equal, the bottleneck value of that partition is the ideal bottleneck B^*. Clearly, the ideal bottleneck B^* is a lower bound for the optimal bottleneck B_{opt}.

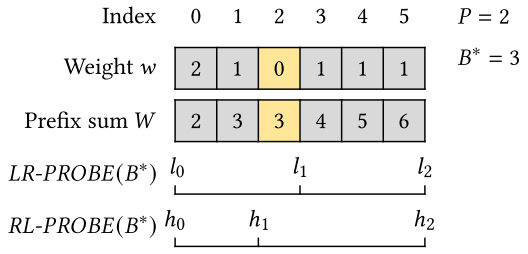


Figure 2: An example of incorrect results generated by *EXACT-BISECT* in the presence of zero load elements. The existence of empty tasks leads to $h_1 < l_1$, causing *RPROBE* to search for the boundary s_1 in an invalid interval.

3.2 Issues in *EXACT-BISECT*

In the *EXACT-BISECT* exact algorithm by Pınar et al. [24], we begin by running two functions, *LR-PROBE* and *RL-PROBE*, to construct the search range $[SL_p, SH_p]$ for each partition boundary s_p, where *LR-PROBE* is the straightforward implementation of the *PROBE* function mentioned above, and *RL-PROBE* is the reverse implementation of *LR-PROBE*, conducting the search for partition boundaries from right to left. Subsequently, *EXACT-BISECT* executes a heuristic to obtain a bottleneck B_H and performs a binary search for B_{opt} using *RPROBE* within the interval $[B^*, B_H]$.

As mentioned in Section 2.2, *EXACT-BISECT* faces two issues that prevent its direct application in SPH simulations:

(1) *EXACT-BISECT may produce incorrect results when zero-load tasks are present* because *LR-PROBE* and *RL-PROBE* could construct incorrect search ranges $[SL_p, SH_p]$ for some partitions. The strongly compressible SPH method often creates transient local vacuum areas, preventing us from using *EXACT-BISECT* to handle cells that contain no particles. Note that we cannot simply remove these cells, as this might result in the loss of some particles in subsequent simulations.

Algorithm 1: Algorithm *LR-PROBE* and *RL-PROBE* along with their modified versions *LR-PROBE-NG* and *RL-PROBE-NG*

Function `LR-PROBE`(B, W, N, P):
- $s_0 \leftarrow 0$; $s_P \leftarrow N$
- **for** $p \leftarrow 1, P-1$ **do**
  - $B_{\text{tgt}} \leftarrow W_{s_{p-1}-1} + B$
  - $s_p \leftarrow$ `UPPER-BOUND`$(W, s_{p-1}, N, B_{\text{tgt}})$

Function `LR-PROBE-NG`(B, W, N, P):
- $s_0 \leftarrow 0$; $s_P \leftarrow N$
- **for** $p \leftarrow 1, P-1$ **do**
  - $B_{\text{tgt}} \leftarrow W_{s_{p-1}-1} + B$
  - $s_p \leftarrow$ `LOWER-BOUND`$(W, s_{p-1}, N, B_{\text{tgt}})$
  - **if** $s_p \neq N$ **and** $W_{s_p} = B_{\text{tgt}}$ **then**
    - $s_p \leftarrow s_p + 1$

Function `RL-PROBE`(B, W, N, P):
- $s_P \leftarrow N$; $B_{\text{tgt}} \leftarrow W_{N-1} - B$
- **for** $p \leftarrow P-1, 0$ **do**
  - $s_p \leftarrow$ `LOWER-BOUND`$(W, 0, s_{p+1}, B_{\text{tgt}}) + 1$
  - $B_{\text{tgt}} \leftarrow W_{s_p-1} - B$

Function `RL-PROBE-NG`(B, W, N, P):
- $s_P \leftarrow N$; $B_{\text{tgt}} \leftarrow W_{N-1} - B$
- **for** $p \leftarrow P-1, 0$ **do**
  - $s_p \leftarrow$ `UPPER-BOUND`$(W, 0, s_{p+1}, B_{\text{tgt}})$
  - **if** $s_p \geq 0$ **and** $W_{s_p-1} = B_{\text{tgt}}$ **then**
    - $s_p \leftarrow s_p - 1$
  - $B_{\text{tgt}} \leftarrow W_{s_p} - B$; $s_p \leftarrow s_p + 1$

(2) *EXACT-BISECT* obtains the final partitioning result by simply invoking $PROBE(B_{\text{opt}})$, which *may lead to imbalanced outcomes* due to the greedy nature of *PROBE*.

Figure 2 provides an example to illustrate the first issue. In this instance, we partition 6 tasks ($N = 6$) into 2 partitions ($P = 2$), with an ideal bottleneck of $B^* = 3$. Running $LR\text{-}PROBE(B^*)$ yields a partition $\mathbf{P}_1 = (0, 3, 6)$, and running $RL\text{-}PROBE(B^*)$ yields $\mathbf{P}_2 = (0, 2, 6)$. So we have $SL_1 = 3$ and $SH_1 = 2$, which implies that *RPROBE* will attempt to search for the boundary s_1 of partition 1 in the interval $[3, 2]$, which is clearly incorrect. Regarding the second issue, consider we need to divide 101 tasks ($N = 101$), each with a load of 1, among 100 processors ($P = 100$). *EXACT-BISECT* identifies the optimal load as $B_{\text{opt}} = 2$. According to the algorithm, we then run $PROBE(2)$ to generate the partition result. However, since *PROBE* is greedy, it initially allocates 2 tasks to each of the first 50 processors, leaving 49 processors idle after assigning the last task to the 51st processor. Although the overall system's execution time depends on the slowest processor, task load isn't the sole determinant of processor execution time; other factors also play a role. Therefore, having the vast majority of tasks executed by the first 50 processors may not be an optimal choice. A preferable strategy would be for one processor to handle 2 tasks, while the remaining 99 processors manage one task each, promoting a more balanced workload distribution.

Algorithm 2: Algorithm *BAL-PART*

Function `BAL-PART`$(B_{\text{opt}}, B^*, W, N, P)$:
- $s^0 \leftarrow$ `LR-PROBE`$(B_{\text{opt}}, W, N, P)$
- $s^1 \leftarrow$ `RL-PROBE`(B^*, W, N, P)
- **for** $p \leftarrow 0, P$ **do**
  - $s_p \leftarrow \min\{s_p^0, s_p^1\}$

3.3 *EXACT-BISECT*+: An Improvement to the *EXACT-BISECT* Algorithm

To address these two issues, we propose an improved algorithm for *EXACT-BISECT*, denoted as *EXACT-BISECT+*. We introduce two new probe algorithms called *LR-PROBE-NG* and *RL-PROBE-NG* to solve the first problem, which are modifications of *LR-PROBE* and *RL-PROBE*, named for their Non-Greedy behavior of leaving all zero-load tasks at the partition boundary to the next partition. *EXACT-BISECT+* utilizes these four probing functions to construct the search boundaries of each partition for *RPROBE*. Specifically, we execute $LR\text{-}PROBE\text{-}NG(B^*)$ and $RL\text{-}PROBE\text{-}NG(B^*)$ to obtain two partitions $(s_0^0, s_1^0, \ldots, s_P^0)$ and $(s_0^1, s_1^1, \ldots, s_P^1)$, respectively. We also run $LR\text{-}PROBE(B_H)$ and $RL\text{-}PROBE(B_H)$ to obtain two partitions $(s_0^2, s_1^2, \ldots, s_P^2)$ and $(s_0^3, s_1^3, \ldots, s_P^3)$, respectively. Subsequently, the lower bound and upper bound for boundary s_p of partition p are set as $SL_p = \max\{s_p^0, s_p^3\}$ and $SH_p = \min\{s_p^1, s_p^2\}$, respectively.

Algorithm 1 presents the probing functions as described. Within the algorithm, $LOWER\text{-}BOUND(W, b, e, tgt)$ performs a binary search within the index range $[b, e)$ in the non-decreasing array W to find the smallest index i such that $W_i \geq tgt$. Conversely, $UPPER\text{-}BOUND(W, b, e, tgt)$ binary searches for the smallest index i where $W_i > tgt$. Both of them set $i = e$ if an element that meets the requirement does not exist.

The idea of the algorithm is to efficiently locate the start and end positions of zero elements at the partition boundary using a combination of *LOWER-BOUND* and *UPPER-BOUND*. This is because if such zero elements exist, there will be multiple consecutive identical elements B_{tgt} in the prefix sum array W, and these elements will fall within the range $[l, h)$, where $l = LOWER\text{-}BOUND(W, b, e, B_{\text{tgt}})$ and $h = UPPER\text{-}BOUND(W, b, e, B_{\text{tgt}})$. It's worth noting that our improvements to the algorithm do not increase its complexity or introduce additional search efforts. All *PROBE* functions in Algorithm 1 maintain a complexity of $O(P \log N)$, as explained in the work by Pınar et al. [24]. Our improvements to the algorithm enable us to quickly differentiate special elements at the partition boundaries without needing to traverse all elements with a value of 0 or introducing additional search efforts.

To address the second issue mentioned earlier, caused by the greedy nature of *PROBE*, we propose the algorithm *BAL-PART*, as demonstrated in Algorithm 2. The idea of *BAL-PART* is that calling *PROBE* with the ideal bottleneck B^* as a parameter results in the most balanced partitioning. Therefore, combining the outcomes of $PROBE(B_{\text{opt}})$ and $PROBE(B^*)$ can produce a more balanced partition. Once the optimal bottleneck B_{opt} is identified,

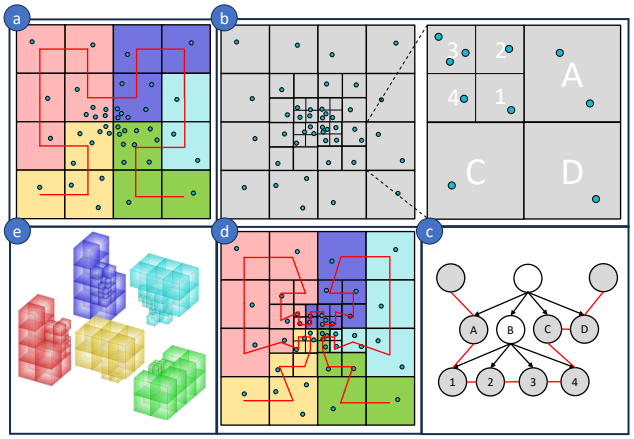


Figure 3: Utilizing the Hilbert space-filling curve for domain decomposition: (a) partitioning without executing the subdivision algorithm; (b) refining the bottleneck regions using the adaptive recursive cell subdivision algorithm; (c) traversing all leaves in the octree forest according to the Hilbert order; (d) partitioning using the *EXACT-BISECT+* algorithm and then distributing; (e) an exploded view of the parallel domain after executing recursive subdivision, with different colors representing different processes.

EXACT-BISECT+ invokes *BAL-PART*(B_{opt}, B^*) to obtain the final partitioning result, instead of calling *PROBE*(B_{opt}) as in the algorithm *EXACT-BISECT*.

4 IMPLEMENTATION AND OPTIMIZATION

4.1 The Adaptive Recursive Cell Subdivision Algorithm

In scenarios of extreme load imbalance, the number of particles contained in different cells can also vary significantly. There are three reasons to further reduce the load of each cell: (1) The maximum value in the load array, $w_{\max}$, is one of the lower bounds for the optimal solution B_{opt}, because this task must be completely assigned to a specific partition. (2) Cells with excessive loads can lead to significant disparities in the load between different partitions, especially when they occur near partition boundaries as shown in Figure 3(a). (3) When scheduling cells to CPE threads for processing, coarse-grained cell partitioning methods result in an imbalance of load among threads.

Our adaptive recursive cell subdivision strategy is capable of identifying cells with excessive loads and subdividing them into sub-cells. Specifically, cells with a load exceeding a given threshold are subdivided into $2 \times 2 \times 2$ sub-cells, and this process is recursively repeated for all sub-cells until their loads do not exceed the threshold, or a certain recursion depth is reached. The top-level cell and all its subdivided sub-cells form an octree, where each cell's 8 sub-cells correspond to its 8 child nodes. All top-level cells form an octree forest. We are interested in the sub-cells on the leaf nodes of all octrees, whose volumes adaptively vary according to the particle density in their respective spaces as shown in Figure 3(b) and Figure 3(c).

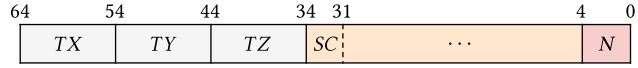


Figure 4: The structure of the cell ID, where TX, TY, TZ represent the x, y, z-dimension indices of the top-level cell, respectively, SC is the index of the sub-cell at each level, and N is the length of the SC segment.

After updating the positions of particles, we need to determine their residing sub-cells based on their spatial locations. In the naive implementation, it is necessary to locate the top-level cell the particle belongs to and then calculate its way down the corresponding octree from the top-level cell through intermediate sub-cells to the leaf sub-cell. This process requires a considerable amount of floating-point coordinate calculations. We propose a strategy that benefits from our carefully designed cell ID system, allowing the determination of the leaf sub-cell a particle resides in with just a single level of floating-point coordinate calculations and a few bit operations. We represent a cell's unique ID with a 64-bit integer, as shown in Figure 4. The high 30 bits of the cell ID denote the three-dimensional index of the top-level cell to which this cell belongs, with each 10 bits storing an index for one dimension, thereby capable of representing 2^{30} top-level cells. The SC segment in the figure uses 30 bits to store the within-level index of the cell and all its ancestor cells in the octree. Since we subdivide a cell into 8 sub-cells at each level, we use 3 bits to represent the index of each level of sub-cell, allowing for the representation of all cells generated by up to 10 levels of recursive subdivision. The remaining 4 bits are used to indicate the used length of the SC segment, i.e., the depth of the cell within the octree.

To calculate the ID of the cell to which the particle belongs based on its spatial coordinates, it is necessary to gather the maximum depth D of the recursive subdivision performed by all processes and the side length (w_x, w_y, w_z) of the sub-cells at that depth in advance. We need to calculate the three-dimensional index (c_x, c_y, c_z) of the cell the particle is in, as if there were an imaginary computational domain division in which all cells have the same size (w_x, w_y, w_z) and the particle belongs to an imaginary cell with coordinates (c_x, c_y, c_z). Then we directly construct the cell ID id_i of this imaginary cell using Algorithm 3 based on (c_x, c_y, c_z), where all arithmetic operations can be efficiently performed using corresponding bitwise operations. We only need to binary search for the largest cell ID id_r satisfying $id_r \leq id_i$ in the ordered sequence of cell IDs composed of all real cells' IDs, and id_r will be the ID of the cell containing the particle. The principle behind this algorithm is that the actual cell id_r where the particle resides must be the same as the imaginary cell id_i or one of its ancestors. The design of our cell ID ensures that if one cell is the ancestor of another, the higher bits of their IDs are identical, and the closer two cells are, the more identical bits they have. This means that id_r and id_i share the maximum number of identical high bits, making id_r the closest ID to id_i.

Algorithm 3: Algorithm for directly constructing the cell ID from its 3D coordinates

$TX \leftarrow \lfloor c_x/2^D \rfloor$; $TY \leftarrow \lfloor c_y/2^D \rfloor$; $TZ \leftarrow \lfloor c_z/2^D \rfloor$; $SC \leftarrow 0$
for $d \leftarrow 0, D-1$ **do**
  Extract the d-th bit x_d, y_d, z_d of c_x, c_y, c_z, respectively
  $SC \leftarrow SC + x_d \times 2^{3d+2} + y_d \times 2^{3d+1} + z_d \times 2^{3d}$
$SC \leftarrow SC \times 2^{30-3D}$; $N \leftarrow D$
Construct the cell ID from (TX, TY, TZ, SC, N)

4.2 Dynamic Thread-Level Task Scheduling Strategy Based on Task List

We propose a dynamic task scheduling strategy based on a task list to further balance the computational tasks assigned to each process among all CPE threads. We treat the interaction computation between a cell and all its neighbors as a single task, aiming to enable different tasks to be executed in parallel without data races as shown in Figure 5(2). Since we cannot predict the computational cost of each task in advance due to the support domain varying with particle density, leading to some particle pairs being ignored in actual calculations, a static scheduling approach that pre-assigns tasks cannot balance the load across each CPE evenly. In our strategy, we maintain a task queue shared by all CPEs. After completing its current task, each CPE autonomously retrieves a new task from the queue, thus dynamically scheduling tasks based on their actual execution times. We use a global counter to manage tasks that have been executed in the task queue and efficiently update this counter using atomic instruction provided by Sunway, to prevent conflicts when multiple CPE threads update the counter simultaneously.

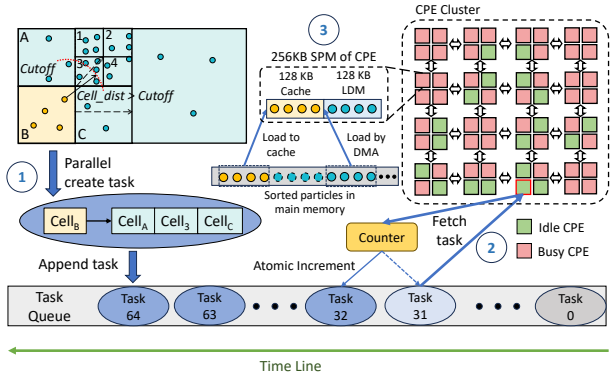


Figure 5: Dynamic thread-level task scheduling for load balancing and task parallelism: 1. excluding cells outside the cut-off radius and create tasks; 2. 64 CPEs act as workers, using atomic increment instructions to acquire ready tasks from the task queue and perform computations; 3. utilizing scratch-pad memory to enhance the spatial and temporal locality of data.

The presence of leaf sub-cells at different subdivision depths increases the difficulty of locating their neighboring cells, due to their varying volumes and irregular positions. To address this, we implement a strategy based on shared neighbor lists, where each cell is associated with a list composed of all its neighboring cells. We consider that if two sub-cells are within neighboring top-level cells, there exists an interaction relationship between these two sub-cells, enabling us to construct the neighbor list based on the top-level cells. The advantage of this method is that all sub-cells of the same top-level cell within the same process can share the same neighbor list, which reduces the frequency of constructing neighbor lists and the overhead of storing them. We dynamically construct the complete task list based on the neighbor list that has been built. Since we may inadvertently include some cells beyond the support domain in the neighbor list due to the smaller support domain radii in high-density particle areas, we compare the distances between cells and remove from the task list those neighbor cells that fall outside the support domain as shown in Figure 5(1). Benefiting from the recursive subdivision strategy, our method can dynamically adjust the number of cell pairs involved in interactions with fine granularity based on particle density. Especially in areas where particles are heavily aggregated, our method can significantly reduce the overhead of searching for neighboring particles.

5 EXPERIMENTAL EVALUATION

In this section, we implement our load balancing strategy on the new Sunway supercomputer, based on the work of Zhang et al. [33]. We evaluate the effects of our strategy on load balancing between processes and threads, as well as the scalability of our implementation across different parallel scales.

5.1 Process-Level Load Balancing Performance Evaluation

In this experiment, we compare the process-level load balancing effects of *EXACT-BISECT+*, *EXACT-BISECT+* combined with the recursive cell subdivision strategy, and the heuristic used in the work of Zhang et al. [33]. Our experiments simulate the converging Richtmyer-Meshkov instability using 1 billion particles. As the shockwave converges towards the central interface area, particle aggregation causes the particle density in the central region to be tens to hundreds of times higher than in other areas. Figure 6 shows our experimental results on small-scale tasks involving 27,000 top-level cells and up to 2000 processes.

Running time and bottleneck. Figure 6 demonstrates a high consistency between the bottleneck values and the overall computational time of the system, indicating that the bottleneck values accurately reflect the computational costs of the slowest process. Moreover, under scenarios of extreme load imbalance, the heuristic algorithm significantly underperforms compared to the exact algorithm, suggesting that heuristic algorithms are susceptible to data distribution and are less suitable for SPH simulations. Using only the exact algorithm can achieve approximately a 1.3× performance improvement.

On the other hand, as the number of target partitions increases, the effectiveness of the exact algorithm gradually declines, as indicated by the optimal bottleneck values B_{opt} no longer decreasing when the number of partitions is high (1200 to 2000 partitions). This "bottleneck limit" results in a persistently high load on the slowest

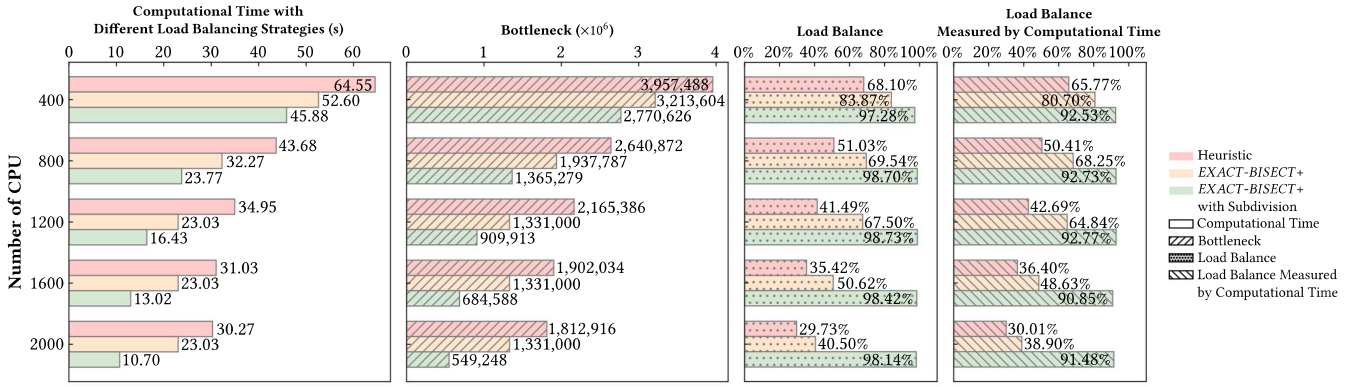


Figure 6: Load balancing effects and computational time of SPH simulations using different strategies for dividing 27,000 top-level cells containing a total of 1 billion particles across 400 to 2000 processes. The computational time is measured based on the slowest process. The bottleneck and load balance derive from the output of the 1D partitioning algorithm. We use the ratio of the average computational time of all processes to the runtime of the slowest process as another metric to assess the load balancing effectiveness.

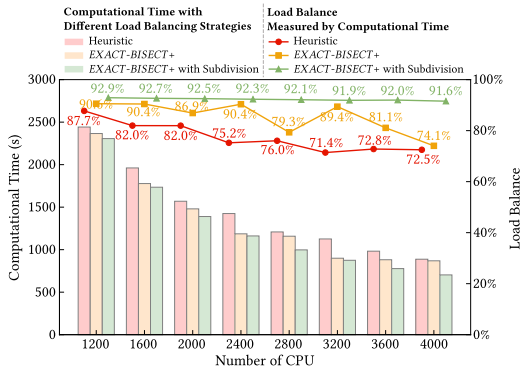


Figure 7: Load balancing effects and computational time of SPH simulations using different strategies for dividing 1 million top-level cells across 1200 to 4000 processes. The computational time is measured based on the slowest process and the load balance is calculated as the ratio of the average computational time of all processes to the computational time of the slowest process.

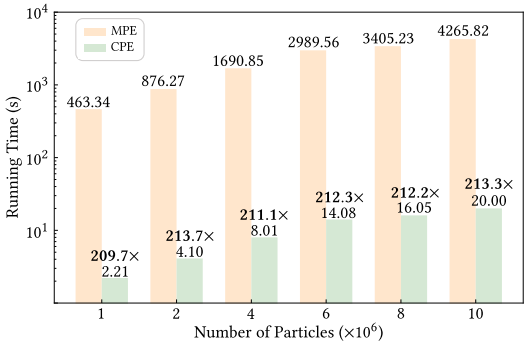


Figure 8: Performance speedup of the MPE-only and CG-accelerated version on a single node.

partition, preventing further reduction in computation time by increasing the parallel scale when the number of processes exceeds 1200. The recursive cell subdivision strategy can break this limit and restore good scalability to the system. Our experiments show that a four-level cell subdivision can yield approximately a 2.8× performance improvement.

Load balance. We record the computational times of each process and calculate the balance of execution times across different processes. The experiments demonstrate that the load balance calculated by the 1D partitioning algorithm almost accurately reflects the load balance measured by actual execution times. It is noted that using either heuristic or exact algorithms alone results in a trend of decreasing load balance as the number of target partitions increases, because the rate of reduction in bottleneck values is slower than the rate of increase in the number of partitions. In contrast, the recursive cell subdivision strategy enables *EXACT-BISECT+* to partition tasks with finer granularity, thus maintaining a nearly constant level of load balance regardless of the number of target partitions.

Figure 7 shows the load balancing performance of our algorithm on large-scale tasks involving 1 million top-level cells and up to 4,000 processes. Due to the significant increase in top-level cells, which makes each cell's load more even—as if several levels of cell subdivision had been pre-executed—the algorithm's effectiveness is not as pronounced as in smaller-scale settings. The experiments demonstrate that our strategy can deliver about a 1.25× performance improvement and produce a load balancing effect that remains nearly constant regardless of the number of partitions, making our process-level load balancing strategy particularly suitable for scenarios with a large number of partitions.

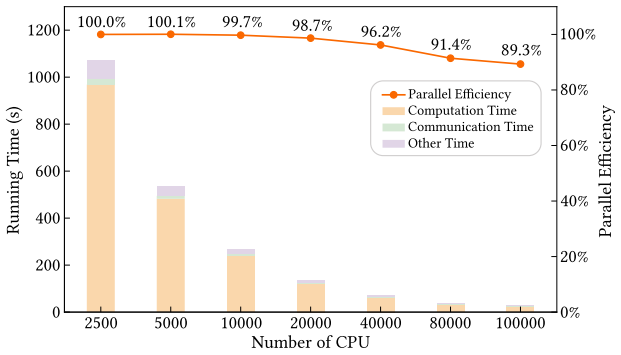


Figure 9: Strong scaling results with 100 billion particles on 2,500 to 100,000 CPUs

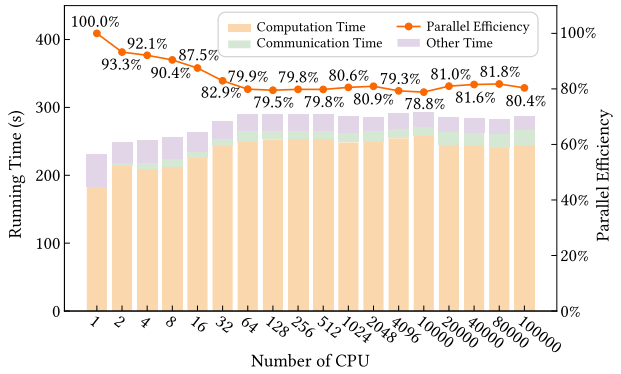


Figure 10: Weak scaling results with 10 million particles per process, scaling form 1 to 100,000 CPUs

5.2 Single Node Evaluation

Figure 8 exhibits the performance speedup of our implementation on a single node. This experiment evaluated the execution time of simulations involving particle counts ranging from 1 million to 10 million on a single CPU. By carefully adjusting the grid subdivision threshold, the particle data within a task was optimized to fit perfectly within a 128KB cache, maximizing computational performance. The experiment recorded the total execution time for both MPE-only computations and those accelerated by 384 CPEs. The results indicate that, in comparison to the MPE-only version, the CPE-accelerated version delivered a remarkable performance speedup of 210 times and 213 times for simulations with 1 million and 10 million particles, respectively.

5.3 Strong and Weak Scaling

Figure 9 demonstrates the strong scalability of our SPH implementation with our load balancing strategy on the Sunway supercomputer. We evaluate our implementation using a test case with 100 billion particles on 2,500 to 100,000 CPUs, assessing computational, communication, and other operational overheads. Experimental results indicate that our implementation can achieve nearly 90% parallel efficiency on up to 100,000 nodes.

Figure 10 illustrates the weak scalability demonstrated by our implementation. This experiment involves each CPU handling 10 million particles, scaling from 1 CPU to 100,000 CPUs, with a total ranging from 10 million to 1 trillion particles. The results indicate that our implementation can maintain a parallel efficiency of 80.4% at a parallel scale of up to 100,000 nodes.

6 VISUALIZATION

The case is characterized by an inward-moving spherical air shock impacting a $40 \times 40 \times 40$ cubic SF_6 gas located at the the center of a spherical domain whose radius r_2 equals 500mm. Similiar to the work of Huang [17], the initial spherical shock is generated by isodensity air with an initial pressure discontinuity at $r_1 = 300mm$. The initial shock takes a Mach number 1.045, which corresponds to an initial pressure ratio of 1.229 between the high-pressure and low-pressure zones. The initial particle spacing is chosen as $l_0 = 0.8mm$ and the total particle number is around 1 billion.

While the initial physical properties of the low pressure air and SF_6 are listed in the Table 1. ρ is mass density, c is sound speed, and γ is specific heats of ideal gas.

Table 1: Parameters corresponding to gas properties

Gas	γ	$\rho(kg/m^3)$	$c(m/s)$
Air	1.4	1.19	345
SF_6	1.094	5.79	135

In order to decrease the kelvin-helmholtz instability effect by initial particle distribution, the density and pressure are smoothed between different zones. The initial specific heat of gas of Air-SF_6 is also smoothed within a limited area using the following formula:

$$\gamma_0(r) = 1.4 - 0.306\left(1 + e^{1.5\frac{r-r_0}{l_0}}\right)^{-1} \quad (1)$$
$$r = min(|x|, |y|, |z|)$$

Here r_0 is the distance from the origin to the geometric center of the square SF_6 surface. Figure 11 plots the results of the simulation at dimensionless time $\tau = t/t_c = 0.94, 1.40, 2.18, 10$, where $t_c = L/W_{i0}$. L is the distance from the origin to the apex of SF_6-Air interface, and W_{i0} is the shock speed when the shock arrives at the apex. As Figure 11 shows, the inward spherical shock first arrives at the geometric center of the square surface, thus pushes the affected area move inward and creates spikes at the apex and the edge of the cubic, as is seen in (a.1) and (b.1). The post-shock fluids move inward, and thus the interior SF_6 is continuously compressed, which indicates a considerable compressibility for the converging RMI, therefore the unbalance of load. As the shock approaches the origin, the surface gradually develops into bubbles, as is shown in (a.2), and in the section in (b.2) the interface looks like a shining star. Meanwhile the density in the origin area has exceeded 8 times the bottleneck value, thus triggering recursive subdivision. After arrival of shock at the origin, the reflected transmitted shock interacts with the evolving interface again, and the bubbles move outward and then reverse into spikes (a.3). The interface in sections (b.1), (b.2), (b.3) is analogous to the 2d experimental results [28]. At dimensionless time $\tau = 10$, the spike head develops into a mushroom structure

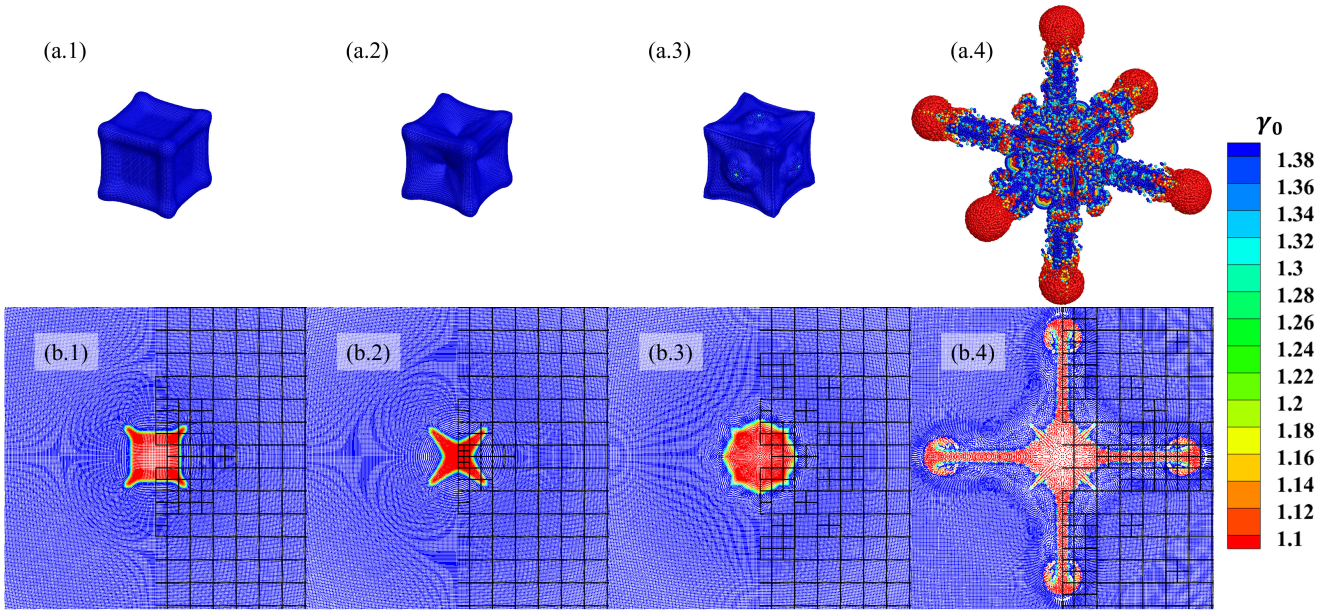


Figure 11: The plots for SF_6-Air interface evolution at dimensionless time $\tau = 0.94, 1.40, 2.18, 10$. (a) is 3d snapshots of SF_6 gas. (b) is section of 3d SF_6 at $x \in [0, l_0]$

(a.4) similar to that of the single-mode RMI[8]. As shown in the Figure 11(b), our adaptive refinement strategy dynamically adjusts the size of the cell as the shock wave evolves, thereby achieving dynamic load balancing.

7 CONCLUSION

In this paper, we propose a three-level load balancing strategy to address the issue of extreme load imbalances in large-scale SPH simulations. The improved 1D exact partitioning algorithm efficiently distributes cells to be processed among processes. The adaptive recursive cell subdivision strategy breaks up heavily loaded cells, enhancing the effectiveness of the 1D partitioning algorithm and reducing the granularity of task division. The thread-level task scheduling strategy dynamically assigns tasks to each CPE based on the actual execution time of tasks, regardless of the subdivision depth of the leaf cells.

Our implementation on the Sunway system demonstrates excellent load balancing capabilities for both small-scale and large-scale tasks. Notably, our strategy's ability to break the bottleneck limit and maintain stable load balancing makes it particularly suitable for large-scale load balancing tasks. The thread-level load balancing strategy efficiently utilizes parallel resources and results in significant performance improvements. Our experiments indicate that the strategy we propose exhibits good scalability under large-scale parallelism across one hundred thousand nodes.

ACKNOWLEDGMENTS

This work was supported by the Strategic Priority Research Program of Chinese Academy of Sciences (XDB0500102) and National Natural Science Foundation of China (Grant No. 62102389). Computing resources are financially supported by Laoshan Laboratory (LSKJ202300305). The authors would like to express their deepest gratitude to the anonymous reviewers for their constructive feedback and suggestions, which significantly contributed to the improvement of this paper. Finally, Junshi Chen is acknowledged as the corresponding author of this study.

REFERENCES

[1] Wilfried Yves Hamilton Adoni, Tarik Nahhal, Moez Krichen, Abdeltif El Byed, and Ismail Assayad. 2020. DHPV: a distributed algorithm for large-scale graph partitioning. *Journal of big Data* 7 (2020), 1–25.

[2] Stephen A Attaway, Edward J Barragy, Kevin H Brown, David R Gardner, Bruce A Hendrickson, Steven J Plimpton, and Courtenay T Vaughan. 1997. Transient solid dynamics simulations on the Sandia/Intel Teraflop computer. In *SC'97: Proceedings of the 1997 ACM/IEEE Conference on Supercomputing*. IEEE, 58–58.

[3] Michael Bader. 2012. *Space-filling curves: an introduction with applications in scientific computing*. Vol. 9. Springer Science & Business Media.

[4] Josh Borrow, Richard G. Bower, Peter W. Draper, Pedro Gonnet, and Matthieu Schaller. 2018. SWIFT: Maintaining weak-scalability with a dynamic range of 10^4 in time-step size to harness extreme adaptivity. *CoRR* abs/1807.01341 (2018). arXiv:1807.01341 http://arxiv.org/abs/1807.01341

[5] Aurélien Cavelan, Rubén M. Cabezón, Michal Grabarczyk, and Florina M. Ciorba. 2020. A Smoothed Particle Hydrodynamics Mini-App for Exascale. In *PASC '20: Platform for Advanced Scientific Computing Conference, Geneva, Switzerland, June 29 - July 1, 2020*. ACM, 11:1–11:11. https://doi.org/10.1145/3394277.3401855

[6] Junshi Chen, Hong An, Wenting Han, Zeng Lin, and Xin Liu. 2021. Towards Efficient Short-Range Pair Interaction on Sunway Many-Core Architecture. *J. Comput. Sci. Technol.* 36, 1 (2021), 123–139. https://doi.org/10.1007/S11390-020-9826-Z

[7] Alejandro J. C. Crespo, José M. Domínguez, Benedict D. Rogers, Moncho Gómez-Gesteira, Stephen M. Longshaw, Ricardo B. Canelas, Renato Vacondio, Anxo Barreiro, and Orlando García-Feal. 2015. DualSPHysics: Open-source parallel CFD solver based on Smoothed Particle Hydrodynamics (SPH). *Comput. Phys. Commun.* 187 (2015), 204–216. https://doi.org/10.1016/j.cpc.2014.10.004

[8] Juchun Ding, Ting Si, Jiming Yang, Xiyun Lu, Zhigang Zhai, and Xisheng Luo. 2017. Measurement of a Richtmyer-Meshkov instability at an air-SF 6 interface in a semiannular shock tube. *Physical review letters* 119, 1 (2017), 014501.

[9] José M. Domínguez, Alejandro J. C. Crespo, Daniel Valdez-Balderas, Benedict D. Rogers, and Moncho Gómez-Gesteira. 2013. New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters. *Comput. Phys. Commun.* 184, 8 (2013), 1848–1860. https://doi.org/10.1016/j.cpc.2013.03.008

[10] Wenqian Dong, Letian Kang, Zhe Quan, Kenli Li, Keqin Li, Ziyu Hao, and Xiang-Hui Xie. 2016. Implementing molecular dynamics simulation on Sunway TaihuLight system. In *2016 IEEE 18th international conference on high performance computing and communications; IEEE 14th international conference on smart city; IEEE 2nd international conference on data science and systems (HPCC/SmartCity/DSS)*. IEEE, 443–450.

[11] Xiaohui Duan, Ping Gao, Meng Zhang, Tingjian Zhang, Hongsong Meng, Yuxuan Li, Bertil Schmidt, Haohuan Fu, Lin Gan, Wei Xue, et al. 2020. Cell-list based molecular dynamics on many-core processors: A case study on sunway TaihuLight supercomputer. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–12.

[12] Xiaohui Duan, Ping Gao, Tingjian Zhang, Meng Zhang, Weiguo Liu, Wusheng Zhang, Wei Xue, Haohuan Fu, Lin Gan, Dexun Chen, et al. 2018. Redesigning LAMMPS for peta-scale and hundred-billion-atom simulation on Sunway TaihuLight. In *SC18: International conference for high performance computing, networking, storage and analysis*. IEEE, 148–159.

[13] Maria Serg Egorova, Sergey A Dyachkov, Anatoly N Parshikov, and VV Zhakhovsky. 2019. Parallel SPH modeling using dynamic domain decomposition and load balancing displacement of Voronoi subdomains. *Computer Physics Communications* 234 (2019), 112–125.

[14] José Antonio Fernández-Fernández, Lukas Westhofen, Fabian Löschner, Stefan Rhys Jeske, Andreas Longva, and Jan Bender. 2022. Fast octree neighborhood search for SPH simulations. *ACM Transactions on Graphics (TOG)* 41, 6 (2022), 1–13.

[15] Mikito Furuichi and Daisuke Nishiura. 2017. Iterative load-balancing method with multigrid level relaxation for particle simulation with short-range interactions. *Computer Physics Communications* 219 (2017), 135–148.

[16] Moncho Gómez-Gesteira, Alejandro J. C. Crespo, Benedict D. Rogers, Robert A. Dalrymple, José M. Domínguez, and Anxo Barreiro. 2012. SPHysics - development of a free-surface fluid solver - Part 2: Efficiency and test cases. *Comput. Geosci.* 48 (2012), 300–307. https://doi.org/10.1016/j.cageo.2012.02.028

[17] Shenghong Huang, Jianyu Xu, Yifan Luo, Pengyue Sun, Xisheng Luo, and Juchun Ding. 2020. Smoothed particle hydrodynamics simulation of converging Richtmyer–Meshkov instability. *Physics of Fluids* 32, 8 (2020).

[18] Matthias Lieber and Wolfgang E Nagel. 2018. Highly scalable SFC-based dynamic load balancing and its application to atmospheric modeling. *Future Generation Computer Systems* 82 (2018), 575–590.

[19] G. R. Liu and M. B. Liu. 2003. Smoothed Particle Hydrodynamics: A Meshfree Particle Method. *World Scientific* (2003).

[20] L. B. Lucy. 1977. A numerical approach to the testing of the fission hypothesis. *The Astrophysical Journal* 8, 12 (1977), 1013–1024.

[21] Serge Miguet and Jean-Marc Pierson. 1997. Heuristics for 1d rectilinear partitioning as a low cost and high quality answer to dynamic load balancing. In *High-Performance Computing and Networking: International Conference and Exhibition Vienna, Austria, April 28–30, 1997 Proceedings 5*. Springer, 550–564.

[22] John Tinsley Oden, Abani Patra, and Yusheng Feng. 1994. Domain decomposition for adaptive hp finite element methods. *Proc. Domain Decomposition* 7 (1994), 295–301.

[23] François Pellegrini and Jean Roman. 1996. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *High-Performance Computing and Networking: International Conference and Exhibition HPCN EUROPE 1996 Brussels, Belgium, April 15–19, 1996 Proceedings 4*. Springer, 493–498.

[24] Ali Pınar and Cevdet Aykanat. 2004. Fast optimal load balancing algorithms for 1D partitioning. *J. Parallel and Distrib. Comput.* 64, 8 (2004), 974–996.

[25] R., A., Gingold, J., J., and Monaghan. 1977. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *MNRAS* (1977).

[26] Hans Sagan. 2012. *Space-filling curves*. Springer Science & Business Media.

[27] Matthieu Schaller, Pedro Gonnet, Aidan B. G. Chalk, and Peter W. Draper. 2016. SWIFT: Using Task-Based Parallelism, Fully Asynchronous Communication, and Graph Partition-Based Domain Decomposition for Strong Scaling on more than 100,000 Cores. *ACM* (2016).

[28] Ting Si, Tong Long, Zhigang Zhai, and Xisheng Luo. 2015. Experimental investigation of cylindrical converging shock waves interacting with a polygonal heavy gas cylinder. *Journal of Fluid Mechanics* 784 (2015), 225–251.

[29] Kevin Verma, Kamil Szewc, and Robert Wille. 2017. Advanced load balancing for SPH simulations on multi-GPU architectures. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–7.

[30] Ting Ye, Dingyi Pan, Can Huang, and Moubin Liu. 2019. Smoothed particle hydrodynamics (SPH) for complex fluid flows: Recent developments in methodology and applications. *Physics of Fluids* 31, 1 (2019).

[31] Yang Yu, Hong An, Junshi Chen, Weihao Liang, Qingqing Xu, and Yong Chen. 2017. Pipelining computation and optimization strategies for scaling gromacs on the sunway many-core processor. In *Algorithms and Architectures for Parallel Processing: 17th International Conference, ICA3PP 2017, Helsinki, Finland, August 21-23, 2017, Proceedings 17*. Springer, 18–32.

[32] Dongliang Zhang, Changjun Jiang, and Shu Li. 2009. A fast adaptive load balancing method for parallel particle-based simulations. *Simulation Modelling Practice and Theory* 17, 6 (2009), 1032–1042.

[33] Ziyu Zhang, Junshi Chen, Zhanming Wang, Yifan Luo, Jineng Yao, Shenghong Huang, and Hong An. 2023. SWSPH: A Massively Parallel SPH Implementation for Hundred-Billion-Particle Simulation on New Sunway Supercomputer. In *European Conference on Parallel Processing*. Springer, 564–577.

[34] Guixun Zhu, Jason Hughes, Siming Zheng, and Deborah Greaves. 2023. A novel MPI-based parallel smoothed particle hydrodynamics framework with dynamic load balancing for free surface flow. *Computer Physics Communications* 284 (2023), 108608.