CCF Transactions on High Performance Computing
https://doi.org/10.1007/s42514-025-00246-x

REGULAR PAPER

Para-FDS: a scalable multilevel parallel scheme for fire dynamic simulator on multicore architectures

Dazheng Liu[1] · Sheng Xiao[1,2] · Xiaoli Ren[3] · Wenjuan Liu[4] · Dajiang Yi[1] · Zean Tian[1] · Jianping Wu[3] · Yongan Wu[1] · Zuodong Niu[1] · Keqin Li[5] · Shaoliang Peng[1]

Received: 25 May 2025 / Accepted: 24 August 2025

Abstract

The Fire Dynamics Simulator (FDS) is widely used for fire simulation but faces scalability challenges due to its limited grid partitioning capabilities. To address this issue, we propose Para-FDS, a scalable multilevel parallel scheme for accelerating FDS targeted at multicore architectures. Para-FDS integrates three key optimizations: (1) an adaptive grid partitioning algorithm to enhance scalability, (2) a communication localization optimization approach to reduce overhead between computing nodes, and (3) a NUMA-aware process mapping strategy to improve core utilization within Non-Uniform Memory Access (NUMA) architecture. Implemented on the Tianhe next-generation supercomputer, Para-FDS achieves a speedup of up to 214× on a practical example. It further reduces communication overhead by up to 38% and execution time by up to 22%, significantly improving FDS scalability and efficiency.

Keywords Communication optimization · Fire dynamics simulator · Adaptive grid partitioning · HPC · Multicore architecture · Parallel algorithm

1 Introduction

Indoor fires pose significant risks to life and property, making accurate and rapid fire simulations crucial for building design and disaster analysis. FDS (Verda et al. 2021; Yakovchuk et al. 2020) is a widely used open-source tool for fire modeling. However, its block-structured input grids cannot be subdivided, limiting scalability and parallelization, especially for large and complex simulations. Furthermore, as each input grid can only be assigned to a

✉ Sheng Xiao
xiaosheng@hnu.edu.cn

Dazheng Liu
liudz@hnu.edu.cn

Xiaoli Ren
renxiaoli18@nudt.edu.cn

Wenjuan Liu
liuwenjuan89@hnu.edu.cn

Dajiang Yi
yidajiang@hnu.edu.cn

Zean Tian
tianzean@hnu.edu.cn

Jianping Wu
wjp@nudt.edu.cn

Yongan Wu
wyoa@hnu.edu.cn

Zuodong Niu
niuzuodong@hnu.edu.cn

Keqin Li
lik@newpaltz.edu

Shaoliang Peng
slpeng@hnu.edu.cn

1 College of Computer Science and Electronic Engineering, Hunan University, Lushan South Road, Changsha 410082, Hunan, China

2 Xiangjiang Laboratory, Jianshan Road, Changsha 410013, Hunan, China

3 College of Meteorology and Oceanography, National University of Defense Technology, Deya Road, Changsha 410073, Hunan, China

4 School of Computer, Hunan First Normal University, Fenglin Third Road, Changsha 410205, Hunan, China

5 Department of Computer Science, State University of New York, 1 Hawk Drive, New York 14260, State of New York, USA

Published online: 05 November 2025

single MPI (Message Passing Interface) process, significant disparities in grid dimensions result in severe load imbalance.

Large-scale numerical simulations in FDS are critically dependent on high-performance computing (HPC) systems. Nowadays, multicore architectures playing a dominant role due to their exceptional computational capabilities and rapid evolutionary progress (Diaz et al. 2012). However, the parallelism of FDS significantly lags behind hardware ability, resulting in inefficient hardware resource utilization and high energy consumption of modern multi-core architectures.

Achieving full acceleration of FDS on modern multicore HPC platforms presents three critical challenges: (1) developing scalable parallelization strategies with balanced workload distribution across input blocks, (2) minimizing frequent inter-node communication overhead, and (3) ensuring effective utilization of NUMA-aware multi-core architectures.

To address the challenges in enhancing the scalability of FDS on multicore architectures, we propose Para-FDS, a multilevel parallel scheme that integrates several optimization strategies. At the process level, we introduce an adaptive grid partitioning algorithm that dynamically subdivides the input blocks into smaller subblocks, effectively improving workload balance. At the inter-node level, we implement a communication localization strategy that minimizes communication overhead by grouping processes based on data locality, thereby optimizing inter-process communication. At the intra-node level, we leverage a NUMA-aware process mapping scheme that efficiently utilizes the Non-Uniform Memory Access (NUMA) nodes, ensuring better memory access patterns and reducing contention. Together, these components significantly enhance the overall performance and scalability of FDS on multicore systems.

The most important contributions of this paper are summarized as follows.

- We propose an adaptive grid partitioning algorithm to automatically decompose and scale FDS input blocks.
- We scale practical FDS input blocks from several to 1,024 processes with 214× speedup on the Tianhe next-generation (Tianhe NG) supercomputer.
- We develop a communication localization strategy and a NUMA-aware process mapping scheme for FDS on multicore architectures, reducing communication time by 22%–38% and execution time by 18%–22%.

This paper is structured as follows: In Sect. 2, we introduce related work. Section 3 provides background information on FDS. In Sect. 4, we present the details of Para-FDS. Section 5 illustrates the experiments and analyzes the optimization results. Section 6 concludes this paper with a discussion of our future work.

2 Related work

As the complexity and scale of FDS simulations increase, efficient parallelization becomes critical to ensure scalability and performance. Although established algorithms for grid partitioning, process mapping, and load balancing have been extensively researched and applied in various domains, their integration and adaptation to FDS software pose unique challenges. This section reviews these key techniques and discusses their relevance to FDS, which motivated our approach.

Grid partitioning methods Traditional domain decomposition methods, which divide the original domain into multiple sub-domains evenly, are widely used to meet the requirements for parallel computation (Qian and Zhang 2012; Rantakokko 2000; Schamberger and Wierum 2003; Allen et al. 2011). For instance, Qian and Zhang (2012) developed an octree-based method to generate basic block-structured blocks, while (Wang et al. 2013) introduced a grid partitioning tool, TH-MeshSplit, to divide a given single- or multi-block structured grid into many subblocks for parallel computing. TH-MeshSplit aims to minimize the ratio of surface area to the volume of the subblock cells, i.e., to partition the grid into cubes in three directions as much as possible. This approach reduces the communication area and balances the computational load between subblocks.

However, when there is a significant disparity in the lengths of the three dimensions of input grids, partitioning them into near-cubes via 3D partitioning becomes challenging, as one or two dimensions may not meet the requirements.

Communication optimizing In computational fluid dynamics (CFD) applications, inter-process communication constitutes a critical performance bottleneck, particularly when scaling to massive parallel systems with numerous dynamically generated subblocks (Rabenseifner and Wellein 2003; Maliszewski et al. 2019). The communication overhead grows exponentially with system scale due to increasing complexity in cross-node data exchanges. Recent advances in topology-aware process placement strategies (Valgren et al. 2007; Hoefler et al. 2014; Georgiou et al. 2017) have demonstrated effective solutions by optimizing data affinity through intelligent mapping of communicating processes to adjacent compute nodes or shared memory domains.

A representative work by Jeannot (2022), TopoMatch, addresses this challenge through dynamic communication pattern adaptation, specifically designed for modern hierarchical architectures featuring NUMA domains and multi-level interconnects. This approach achieves superior communication efficiency by preserving data locality while accommodating complex topology constraints.

NUMA-aware process mapping Load balance is another critical factor in ensuring the efficient use of multicore architectures,

particularly in NUMA-based systems (Muddukrishna et al. 2016; Drebes et al. 2016). Psaroudakis et al. (2016) designed an adaptive NUMA-aware data placement for analytical workload in main-memory column stores, which could balance the utilization of resources across NUMA nodes. By applying NUMA-aware process mapping strategy for FDS, we optimized intra-node workload distribution, ensuring that each NUMA node was fully utilized while minimizing memory access cross NUMA node. This led to a more efficient use of computing resources and reduced simulation runtimes for large-scale fire models.

3 Background

FDS adopts the low-mach, large-eddy simulation (LES) method to simulate thermally-driven flows within buildings (McGrattan et al. 2005). The governing equations are approximated using second-order accurate finite differences on uniformly spaced 3D grids. Blocks are constructed by multiple grid cells, and each block can be processed in parallel using MPI (Harlow and Welch 1965; Morinishi et al. 1998) processes.

One FDS input block is a structured hexahedron composed of fundamental units called grid cells. The solution process for FDS occurs within each grid cell. For example, an input block may contain 8, 12, and 10 grid cells in the X, Y, and Z axes, respectively, totaling $8 \times 10 \times 12$ grid cells. Each input block is assigned to a separate MPI process. The computational complexity of an input block is determined by its number of grid cells; consequently, a higher number of grid cells in an input block leads to increased computing time.

FDS process comprises three phases: initialization, prediction, and correction. The initialization phase begins with setting up the MPI, reading FDS input files, and initializing data. During the main cycle, variables are initially estimated with coarse precision and subsequently refined in the correction phase. After computations are completed within a time step, variable data must be communicated across the boundary grid cells of the input block.

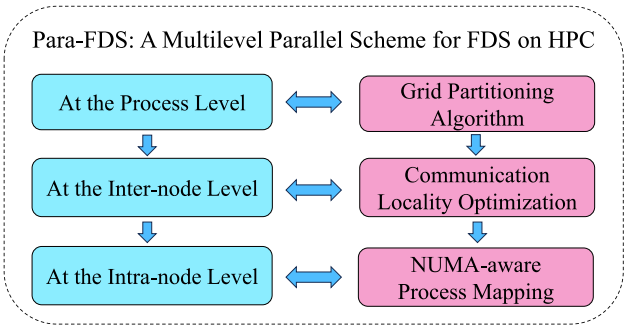


Fig. 1 The multilevel parallel framework of Para-FDS on multicore HPC platforms

Table 1 Notation introduction in the adaptive grid partitioning algorithm

Abbreviation	Full name
m	The number of input blocks
M_i	The i-th input block
V_i	The number of grid cells in the i-th input block
n_i	The number of subblock of the i-th input block
n	The number of all divided subblocks
V_{all}	The sum of grid cells over all input blocks
V_{ave}	The average number of grid cells over all input blocks
n_z	The quotient of V_i divide by V_{ave}
S	The sum of all n_i
n_{over}	The difference between Sum_n_i and n
n_t	The initial value of n_i
U_j	The number of grid cells in the j-th subblock
L	The number of grid cells of X axis
W	The number of grid cells of Y axis
H	The number of grid cells of Z axis
K	Maximum value of L, W, H
N_j	The j-th divided subblock
n_o	The difference between n_t and n_i

4 Methods

As shown in Fig. 1, Para-FDS mainly includes three parts: adaptive grid partitioning algorithm at the process level, communication localization optimization at the inter-node level and NUMA-aware process mapping within a node.

4.1 Adaptive grid partitioning

In FDS simulations, the distribution of grid cells across computational blocks plays a critical role in determining computational efficiency. Significant disparities in grid cell counts between adjacent blocks often lead to severe workload imbalance. Moreover, the lack of automatic input block partitioning in FDS fundamentally restricts its computational scalability. To mitigate these limitations, we propose an adaptive grid partitioning algorithm that optimizes workload distribution by minimizing the ratio of maximum to minimum grid cells among subblocks. The key variables of this algorithm are systematically presented in Table 1.

The problem is formally defined with m input blocks, where the i-th block denoted as M_i contains V_i grid cells and is partitioned into n_i subblocks. The total number of generated subblocks is n, where:

$$n = \sum_{i=1}^{m} n_i. \tag{1}$$

The proposed algorithm consists of two phases: (1) computing the optimal number of subblocks n_i for each input block

M_i, and (2) adaptively decomposing the block M_i into its corresponding n_i subblocks.

Algorithm 1 Optimal Subblock Computing Algorithm

Input: m **input blocks, named** M_i **with** V_i **grid cells**
Output: n_i **subblocks for** M_i**,** n **subblocks in all**
1: $V_{\text{all}} \leftarrow \sum_1^m V_i, V_{\text{ave}} \leftarrow V_{\text{all}}/n$
2: $n_z \leftarrow V_i/V_{\text{ave}}, n_i \leftarrow [n_z]$ // Round n_z to n_i
3: **if** $n_z - n_i > 0.5$ **then**
4: $n_i \leftarrow n_i + 1$
5: **end if**
6: $S \leftarrow \sum_1^m n_i, n_{over} \leftarrow S - n$
7: **if** $n_{over} > 0$ **then**
8: $\max\{n_i\} \leftarrow \max\{n_i\} - n_{over}$
9: **end if**

4.1.1 Phase 1: Calculation of the subblocks for input blocks

The partitioning process, as outlined in Algorithm 1, operates on each input block M_i containing V_i grid cells and determines the optimal number of subblocks n_i. The algorithmic procedure proceeds as follows:

- For all input blocks, derive the total count and average value of their constituent grid cells. (L1).
- Determine the number of subblocks n_i to be partitioned for the i-th input block M_i (L2 – L5).
- Adjust the maximum value of n_i to obtain the number of subblock of the i-th input block (L6 – L9).

Algorithm 2 Adaptive Factorization Grid Partitioning Algorithm

Input: n_i **subblocks for** M_i **from 1 to** m
Output: n **subblocks with grid cells** U_j
1: **while** i from 1 to m **do**
2: $n_t \leftarrow n_i$, $L \times W \times H \leftarrow V_i$
3: **if** $V_i \bmod n_i = 0$ **then**
4: divide V_i into $U_1, U_2, \cdots, U_{n_i}$
5: **else if** $n_i \bmod V_i \neq 0 \cap n_i \notin$ Prime **then**
6: factor n_i into $a \times b \times c$ in three dimensions
7: divide L into a parts
8: divide W into b parts
9: divide H into c parts
10: $U_1 \leftarrow L_1 \times W_1 \times H_1, \cdots, U_{n_i} \leftarrow L_a \times W_b \times H_c$
11: // divide V_i in three dimensions
12: **else if** $n_i \bmod V_i \neq 0 \cap n_i \in$ Prime $\cap \max\{L, W, H\} > n_i$ **then**
13: $K \leftarrow \max\{L, W, H\}$
14: divide K into n_i parts
15: divide V_i into n_i parts along the K dimension
16: **else**
17: $n_i \leftarrow n_i - 1$
18: return to line 3
19: **end if**
20: **if** $n_t - n_i > 0$ **then**
21: $n_o \leftarrow n_t - n_i$
22: **while** j from 1 to n_o **do**
23: divide U_j into two parts equally
24: **end while**
25: **end if**
26: **end while**

4.1.2 Phase 2: Decomposition of input blocks

Building upon the subblock count n_i determined in Algorithm 1, this phase performs 3D partitioning of input blocks. The algorithm decomposes n_i into a three-factor product $a \times b \times c$, representing the partitioning dimensions. Subsequently, the grid cells are uniformly distributed along the three spatial dimensions, with proportional scaling preserved. The optimization objective aims to minimize the directional subblock cost, defined as $L/a + W/b + H/c$. For prime-valued n_i, the algorithm implements uniform partitioning along the longest dimension. The complete implementation details are specified in Algorithm 2:

- Exact Division: When V_i is divisible by n_i, uniformly partition V_i along the optimal spatial axis (1D/2D/3D) into n_i sub-units (L3 - L4).
- Composite Divisor: For non-divisible n_i where n_i is composite:
  - Factorize n_i into three integers $a \times b \times c$.
  - Partition V_i proportionally along three dimensions to minimize the spatial cost function $(L/a + W/b + H/c)$ (L5–L10).
- Prime Divisor Handling:
  - Adequate Grid Capacity: If maximum grid dimension exceeds prime n_i, partition along that dimension (L12–L15).
  - Insufficient Capacity: reduce n_i by 1 and recursively re-evaluate partitioning (L16–L18).
- Recursive Refinement: For residual partitions from recursion ($n_o > 0$), bisect each subblock uniformly (L20–L25).

4.2 Communication localization optimization

In distributed high-performance computing systems such as supercomputers, data communication between nodes relies on network-based transmission, whereas processes within a single node can access shared memory and exchange data via direct copying, free from network interference (Lu et al. 2022). As a result, intra-node communication is inherently faster than inter-node communication. For example, the default mapping scheme automatically assigns 16 processes to four nodes in a fixed order, as shown in the right panel of Fig. 2. Here, the physical spatial distribution of processes within each node is relatively scattered, leading to suboptimal data locality.

To address this issue, we aim to minimize inter-node communication by designing a communication locality-based mapping scheme. By strategically co-locating processes that frequently communicate within the same node, inter-node communication overhead can be significantly reduced. As illustrated in the left panel of Fig. 2, the original scheme incurs 12 instances of inter-node communication, whereas the communication-optimized scheme reduces this number to 8, achieving a one-third reduction in inter-node communication through communication locality optimization.

4.3 NUMA-aware process mapping

In modern HPC architectures, NUMA is a prevalent design where each computing node consists of multiple NUMA nodes, each containing several cores (Hager and Wellein 2008). Efficient NUMA utilization requires distributing MPI processes across nodes to balance computational load. However, conventional block-based scheduling sequentially assigns processes, leading to overloading of certain nodes while others remain underutilized.

Performance in such systems is often constrained by the most heavily loaded NUMA node due to memory contention. To alleviate this problem, we propose a NUMA-aware

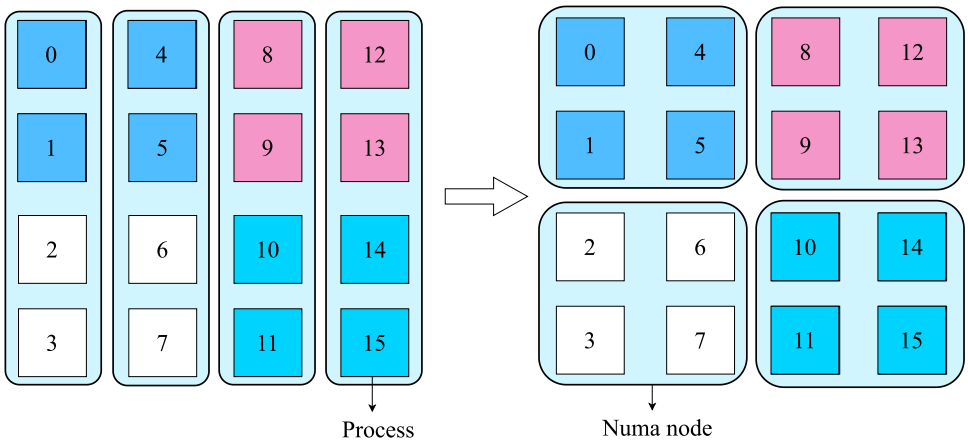


Fig. 2 Communication localization optimization strategy on four nodes. Blocks of the same color indicate blocks with communication locality. Each number represents a process that a block is assigned

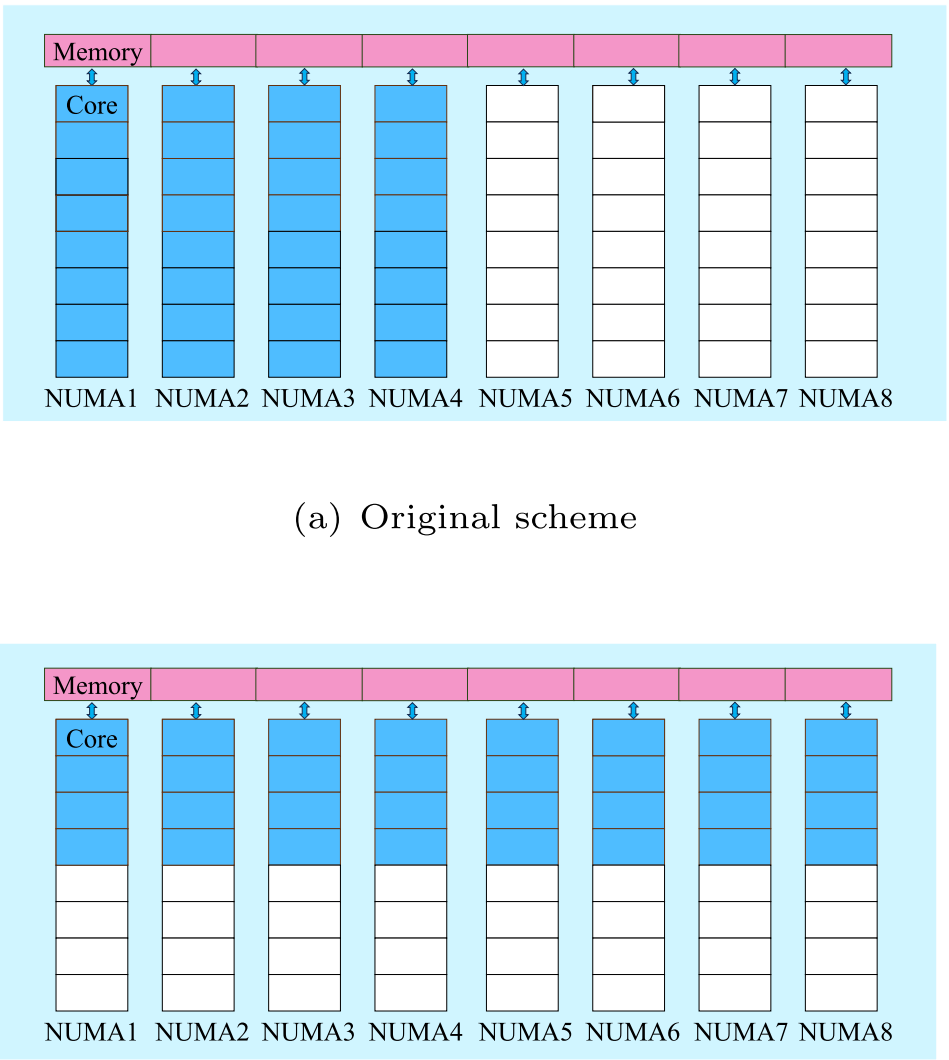


Fig. 3 The NUMA-aware process mapping when 32 processes are assigned to a processor with 8 NUMA nodes, with 8 cores each. The blue block represents the core of the assigned task, and the white block represents no assigned task

process mapping scheme that evenly distributes tasks across NUMA nodes evenly, ensuring workload balance within a node. As illustrated in Fig. 3, with 32 MPI processes allocated on a node containing eight NUMA nodes (each with eight cores), the original scheduling method only utilizes four nodes, leaving the remaining nodes idle. In contrast, NUMA-aware process mapping distributes 32 processes across all eight NUMA nodes, maximizing resource utilization and improving system performance.

5 Evaluation and experiments

We conduct experiments on the Tianhe NG supercomputer (Lu et al. 2022; Wang et al. 2020) to test the performance of Para-FDS. Each computing node contains an FT2000+ processor, which consists of eight NUMA nodes, each with eight cores, totaling 64 cores, all running at a clock speed of 2.2 GHz. Each core is supported by a 32 KB L1 cache, and four cores share a 32 MB L2 cache. Together, the 64 cores can achieve a peak performance of 563.2 Gflop/s of double-precision using the ARM 64 instruction set.

Table 2 Comparison of block-wise grid cell distributions and R_b values between the adaptive grid partitioning scheme and the initial scheme for the *Subway* case

Scheme	Subblocks	Maximum-grid-cell	Minimum-grid-cell	R_b
Initial	7	333,600	13,680	24.39
	16	48,000	13,680	3.51
	32	20,850	13,680	1.52
	64	9828	6,840	1.44
Adaptive grid partitioning	128	4914	4140	1.19
	256	2400	1800	1.33
	512	1365	1140	1.20
	1024	650	468	1.39

The system is equipped with 124 GB of DDR4 memory. The software system in use is Linux Ubuntu 20.04.5 with Infiniband interconnection. The MPI is OpenMPI version 4.1.4, and FDS is version 6.7.4.

First, we propose a metric to evaluate the workload distribution across subblocks by comparing the grid partitioning method with the original undivided one. Next, we assess the efficiency of the adaptive grid partitioning algorithm through practical scenarios. Subsequently, we analyze the effectiveness of the communication localization optimization approach. Finally, we evaluate the NUMA-aware process mapping method.

5.1 Workload balancing estimation

We introduce an indicator R_b to measure workload balancing. R_b is defined as the ratio of the grid cell number of the maximal subblock to the minimum one:

$$R_b = \frac{\max(U_i)}{\min(U_i)}. \tag{2}$$

The balance metric R_b quantitatively reflects workload distribution, where $R_b = 1$ indicates perfect balance and higher values denote increasing imbalance. We evaluate our algorithm using the *Subway* engineering case comprising seven input blocks. As shown in Table 2, the initial configuration yields $R_b = 24.39$ when treating each input block as a single unit, revealing severe workload imbalance. However, our adaptive grid partitioning algorithm achieves remarkable balance when dividing the system into ≥ 32 subblocks, maintaining R_b below 1.5 and approaching the ideal value of 1. This demonstrates the algorithm's effectiveness in creating balanced workload distributions across partitioned subblocks.

5.2 Scalability

The adaptive grid partitioning algorithm effectively scales input blocks and achieves substantial speedup across various scenarios. Performance evaluation using the *Subway* case reveals key insights. As shown in Fig. 4a, execution time and speedup are analyzed over a 10-s simulation with varying subblock counts. Initially, both wall clock and iteration times decrease significantly, reaching a minimum at 128 subblocks before gradually increasing. Here, speedups of 169.92 (based on wall-clock time) and 214.29 (based on iteration time) are achieved, with a workload balancing factor R_b of 1.19, indicating an effectively balanced workload distribution. Here, at most 214.29× speedup is obtained when dividing into 128 subblocks from seven initial blocks with the same configuration. These results confirm our algorithm's scalability and effectiveness in enhancing input block simulations.

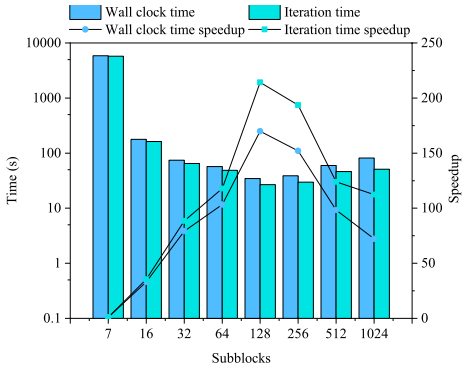
To evaluate the adaptability of the adaptive grid partitioning algorithm, we tested it on four FDS scenarios: *Strong- scale*, *Box- burn*, *Pressure*, and *Enthalpy*. Figure 4b–-e present the results of execution time and speedup over a 10-s simulation with varying subblock partitions. In *Strong- scale* (Fig. 4b), execution time decreases up to 256 subblocks, achieving peak speedups of 38.86 (wall clock) and 71.46 (iteration time), before increasing due to communication overhead. *Box- burn* (Fig. 4c) follows a similar trend, with optimal speedups of 27.77 and 31.52 at 64 subblocks. *Pressure* (Fig. 4d) reaches its lowest execution time at 256 subblocks, yielding speedups of 46.16 and 47.66. *Enthalpy* (Fig. 4e) performs best at 128 subblocks, with speedups of 42.96 and 50.79.

These results confirm the adaptive grid partitioning algorithm's effectiveness across different scenarios, demonstrating scalability and significant speedup. However, increasing subblocks beyond an optimal point introduces more communication overhead, limiting scalability. The results in Fig. 4 show the optimal subblock count varies (128, 256, 64, 256, and 128), influenced by simulation scale and computational complexity. Thus, determining the optimal grid partitioning requires empirical tuning based on specific physical and chemical conditions.

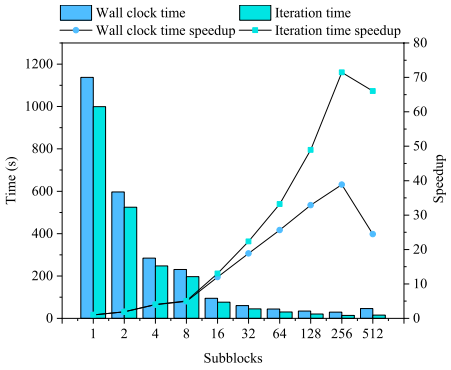
5.3 Communication localization optimization

In this section, we evaluated the communication localization optimization strategy using the *Subway* case with 128 subblocks on 16 nodes. As shown in Fig. 5, the communication localization scheme reduces the communication time by 22.92%–38.31% and the execution time by 18.36%–22.33%.

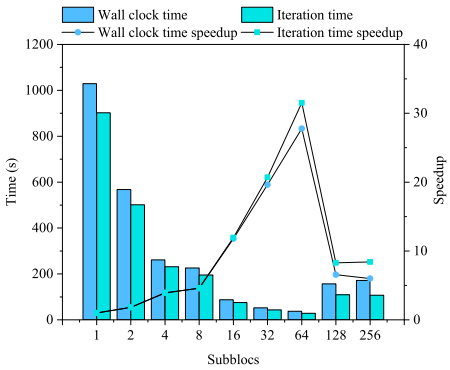
The results above illustrate that the communication localization optimization strategy, which puts as many communicating processes as possible into a single computing node,



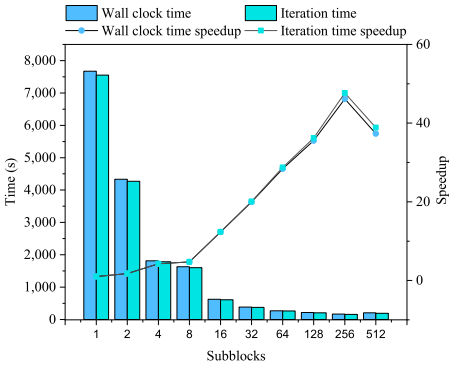
(c) Case *Subway*



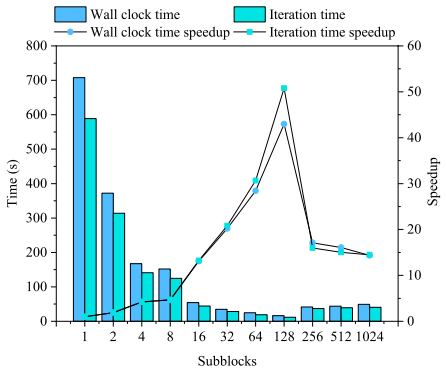
(d) Case *Strong-scale*



(e) Case *Box-burn*



(f) Case *Pressure*



(g) Case *Enthalpy*

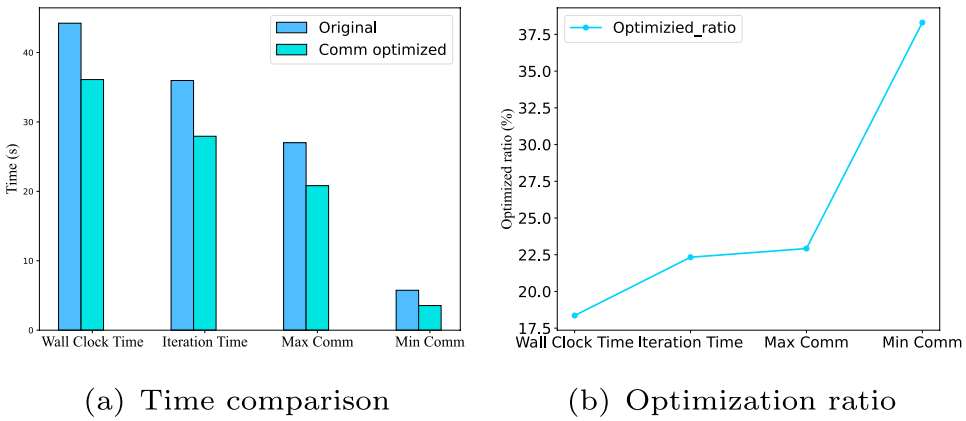
Fig. 4 Execution time and speedup for FDS cases when dividing into subblocks for five practical cases

reduces the long communication distances between nodes, and thus significantly reduces the communication time, as well as reduces the execution time.

5.4 NUMA-aware process mapping

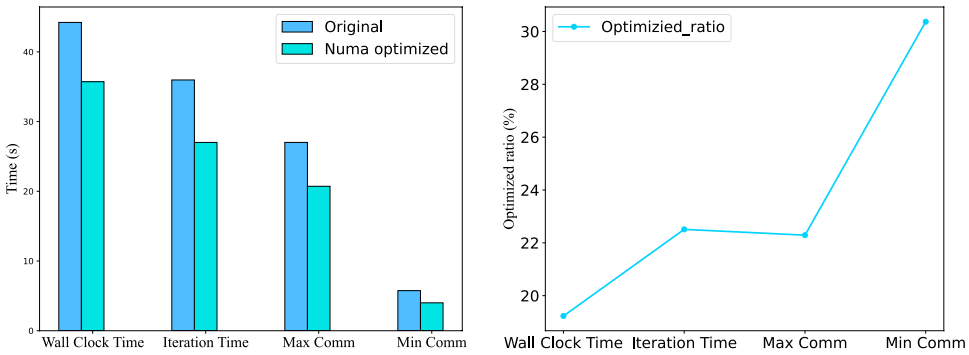
To assess the efficacy of the proposed NUMA-aware process mapping scheme, we performed experiments on the *Subway* benchmark, utilizing 128 subblocks distributed on 16 computing nodes. As depicted in Fig. 6, the experimental results reveal substantial performance enhancements. Notably, the proposed scheme reduced wall-clock time by 19.23%, iteration time by 22.51%, communication time by 23.29%–30.37%.

In this configuration, 128 processes were mapped to 128 computational blocks distributed across 16 nodes, with each node hosting eight processes. Under the initial scheme, all eight processes per node were constrained to a single



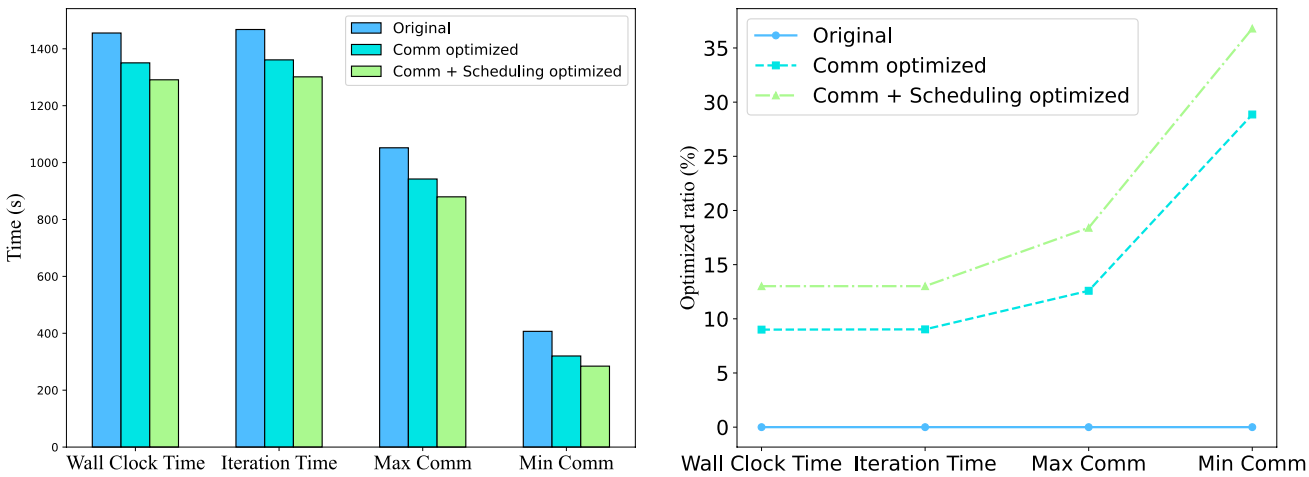
(a) Time comparison (b) Optimization ratio

Fig. 5 Comparison of execution time and communication time between the communication localization optimization and the original scheme on 16 nodes with 128 processes in case *Subway*



(a) Time comparison (b) Optimization ratio

Fig. 6 Comparison of execution time and communication time between the NUMA-aware process mapping scheme and the original scheme on 16 nodes with 128 processes in case *Subway*



(a) Time comparison (b) Optimization ratio

Fig. 7 Execution time for the communication localization optimization and NUMA-aware process mapping scheme with 128 processes on eight nodes in case *Subway*

NUMA node, leading to resource contention and degraded performance. In contrast, the optimized scheme strategically distributes these processes across eight NUMA nodes, ensuring a balanced workload allocation within each node. This approach mitigates resource contention and yields significant performance improvements.

Further testing combined this scheme with the communication localization optimization with case *Subway* on eight nodes over a 100-s simulation. Figure 7 shows that only a single communication optimization reduces the communication time by 12.59%–28.86% and the execution time by 9%, while two optimization strategies reduce the communication time by 18.40%–36.79% and the execution time by 13%. This result indicates that the two schemes can be superimposed to have simultaneous optimization effects both at the inter-node level of the process and at the intra-node level.

6 Conclusion

This paper presents the deployment and optimization of block-structured FDS on multicore HPC platforms via a scalable multilevel parallel framework, Para-FDS. To address the challenges of load imbalance and limited scalability in conventional FDS deployments, we propose a hierarchical optimization approach. At the process level, an adaptive grid partitioning algorithm dynamically balances computational workloads. At the inter-node level, a communication locality optimization minimizes inter-node communication overhead, while at the intra-node level, a NUMA-aware process mapping strategy ensures efficient workload distribution across NUMA domains.

Experimental results on the Tianhe NG supercomputer demonstrate the effectiveness of Para-FDS, achieving a 214× speedup by partitioning seven blocks into 1024 subblocks. Communication time and overall execution time are reduced by up to 38% and 22%, respectively. These improvements directly alleviate performance bottlenecks caused by load imbalance and poor scalability, significantly enhancing the parallel efficiency and scalability of FDS simulations.

In future work, we plan to extend Para-FDS to heterogeneous computing architectures and scale it to larger HPC clusters, aiming to support more complex, data-intensive, and dynamic fire simulation scenarios.

Acknowledgements This study was supported by the Major Program of Xiangjiang Laboratory (No.22XJ01004); NSFC-FDCT (62361166662); National Key R&D Program of China (2023YFC3503400, 2022YFC3400400); The Innovative Research Group Project of Hunan Province (2024JJ1002); Top 10 Technical Key Project in Hunan Province (2023GK1010); Key R&D Program of Hunan Province (2023GK2004, 2023SK2059, 2023SK2060); Key Technologies R&D Program of Guangdong Province (2023B1111030004); The National Natural Science Foundation of China (41875121, 42305170), and the Natural Science Foundation of Hunan Province (2023JJ40678); The Funds of State Key Laboratory of Chemo/Biosensing and Chemometrics; the National Supercomputing Center in Changsha (http://nscc.hnu.edu.cn/), and Peng Cheng Lab.

Declarations

Conflict of interest The authors have no conflict of interest to declare that are relevant to the content of this article.

References

Allen, S.D., Burke, E.K., Kendall, G.: A hybrid placement strategy for the three-dimensional strip packing problem. Eur. J. Oper. Res. **209**(3), 219–227 (2011)

Diaz, J., Munoz-Caro, C., Nino, A.: A survey of parallel programming models and tools in the multi and many-core era. IEEE Trans. Parallel Distrib. Syst. **23**(8), 1369–1386 (2012)

Drebes, A., Pop, A., Heydemann, K., Drach, N., Cohen, A.: Numa-aware scheduling and memory allocation for data-flow task-parallel applications. In: Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 1–2 (2016)

Georgiou, Y., Jeannot, E., Mercier, G., Villiermet, A.: Topology-aware resource management for hpc applications. In: Proceedings of the 18th International Conference on Distributed Computing and Networking, pp. 1–10 (2017)

Hager, G., Wellein, G.: Architecture and performance characteristics of modern high performance computers. In: Computational Many-Particle Physics, pp. 681–730. Springer (2008)

Harlow, F.H., Welch, J.E.: Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. Phys. Fluids **8**(12), 2182–2189 (1965)

Hoefler, T., Jeannot, E., Mercier, G.: An overview of topology mapping algorithms and techniques in high-performance computing. High-performance computing on complex environments **95**, 75 (2014)

Jeannot, E.: Process mapping on any topology with topomatch. J. Parallel Distrib. Comput. **170**, 39–52 (2022)

Lu, P.-J., Lai, M.-C., Chang, J.-S.: A survey of high-performance interconnection networks in high-performance computer systems. Electronics **11**(9), 1369 (2022)

Lu, K., Wang, Y., Guo, Y., Huang, C., Liu, S., Wang, R., Fang, J., Tang, T., Chen, Z., Liu, B.: Mt-3000: a heterogeneous multi-zone processor for hpc. CCF Trans. High Performance Comput. **4**(2), 150–164 (2022)

Maliszewski, A.M., Vogel, A., Griebler, D., Roloff, E., Fernandes, L.G., Oa, N.P.: Minimizing communication overheads in container-based clouds for hpc applications. In: 2019 IEEE Symposium on Computers and Communications (ISCC), pp. 1–6 (2019). IEEE

McGrattan, K.B., Forney, G.P., Floyd, J., Hostikka, S., Prasad, K.: Fire Dynamics Simulator (version 4)—User's Guide. US Department of Commerce, Technology Administration, National Institute of... (2005)

Morinishi, Y., Lund, T.S., Vasilyev, O.V., Moin, P.: Fully conservative higher order finite difference schemes for incompressible flow. J. Comput. Phys. **143**(1), 90–124 (1998)

Muddukrishna, A., Jonsson, P.A., Brorsson, M.: Locality-aware task scheduling and data distribution for openmp programs on numa systems and manycore processors. Sci. Program. **2015**, 5–5 (2016)

Psaroudakis, I., Scheuer, T., May, N., Sellami, A., Ailamaki, A.: Adaptive numa-aware data placement and task scheduling for analytical workloads in main-memory column-stores. Proc. VLDB Endow. **10**(2), 37–48 (2016)

Qian, J., Zhang, Y.: Automatic unstructured all-hexahedral mesh generation from b-reps for non-manifold cad assemblies. Eng. Comput. **28**, 345–359 (2012)

Rabenseifner, R., Wellein, G.: Communication and optimization aspects of parallel programming models on hybrid architectures. Int. J. High Performance Comput. Appl. **17**(1), 49–62 (2003)
Rantakokko, J.: Partitioning strategies for structured multiblock grids. Parallel Comput. **26**(12), 1661–1680 (2000)
Schamberger, S., Wierum, J.-M.: Graph partitioning in scientific simulations: Multilevel schemes versus space-filling curves. In: International Conference on Parallel Computing Technologies, pp. 165–179 (2003). Springer
Valgren, C., Duckett, T., Lilienthal, A.: Incremental spectral clustering and its application to topological mapping. In: Proceedings 2007 IEEE International Conference on Robotics and Automation, pp. 4283–4288 (2007). IEEE
Verda, V., Borchiellini, R., Cosentino, S., Guelpa, E., Tuni, J.M.: Expanding the fds simulation capabilities to fire tunnel scenarios through a novel multi-scale model. Fire Technol. **57**(5), 2491–2514 (2021)
Wang, Y.-X., Zhang, L.-L., Liu, W., Che, Y.-G., Xu, C.-F., Wang, Z.-H., Zhuang, Y.: Efficient parallel implementation of large scale 3d structured grid cfd applications on the tianhe-1a supercomputer. Comput. Fluids **80**, 244–250 (2013)
Wang, R., Lu, K., Chen, J., Zhang, W., Li, J., Yuan, Y., Lu, P., Huang, L., Li, S., Fan, X.: Brief introduction of tianhe exascale prototype system. Tsinghua Sci. Technol. **26**(3), 361–369 (2020)
Yakovchuk, R., Kuzyk, A., Skorobagatko, T., Yemelyanenko, S., Borys, O., Dobrostan, O.: Computer simulation of fire test parameters façade heat insulating system for fire spread in fire dynamics simulator (fds) (2020)

Dazheng Liu is a Ph.D. student in College of Computer Science and Electronic Engineering at Hunan University, Changsha, Hunan, China. His research interests mainly include parallel computing, AI for Science.

Sheng Xiao is currently an associate professor in College of Computer Science and Electronic Engineering, Hunan University, China. He obtained B.S.(02) from Tsinghua University, China, M.S.(03) from National University of Singapore, and Ph.D.(13) from University of Massachusetts, Amherst, U.S.A.

Xiaoli Ren obtained the Ph.D. degree with the College of Computer Science and Technology, National University of Defense Technology, Changsha, China, in 2021. She is currently an assistant researcher with the College of Meteorology and Oceanography, National University of Defense Technology. Her research focuses on high performance computing, parallel optimization, and meteorological and oceanographic big data.

Wenjuan Liu received her Ph.D. from Tongji University and was a postdoctoral fellow at the National Supercomputing Center in Changsha, Hunan University. Her main research areas include high-performance computing, system expansion, and large-scale data processing.

Dajiang Yi who graduated from National University of Defense Technology with a doctoral degree, currently serves as the Vice President of the Innovation Research Institute of National Supercomputing Center in Changsha. His main research areas include satellite navigation, high-performance computing, and large model technology.

Zean Tian got his Ph.D degree from Hunan University in 2009, and was engaged in postdoctoral research at the School of Materials Science and Engineering, University of New South Wales, Australia from 2010 to 2014. He is currently working at National Supercomputing Center in Changsha and College of Computer Science and Electronic Engineering in Hunan University. Dr. Tian's research interests include: high-performance computing, dynamic analysis of complex systems, molecular dynamics simulation, structure analysis of disordered system, visualization in scientific computing, and artificial intelligence applications, etc.

Jianping Wu born in 1974, PhD, researcher. His main interests include large-scale science and engineering computing, and numerical weather prediction, and ocean modeling.

Yongan Wu received his Ph.D. from the National University of Defense Technology and serves as the Deputy Director of the High-Performance Computing Department at the National Supercomputing Center in Changsha, Hunan University. His main research areas cover high-performance computing, artificial intelligence, and big data processing.

Zuodong Niu received the M.S. Degree in control engineering from the College of Electrical Engineering, Guizhou University, Guiyang, China, in 2020. He is currently working toward the Ph.D. degree with the College of Computer Science and Electronic Engineering, Hunan University, Changsha, China. His main research interests include computer vision, deep learning, and object detection.

Keqin Li received a B.S. degree in computer science from Tsinghua University in 1985 and a Ph.D. degree in computer science from the University of Houston in 1990. He is a SUNY Distinguished Professor at the State University of New York and a National Distinguished Professor at Hunan University (China).

Shaoliang Peng is the executive director/professor of College of Computer Science and Electronic Engineering. His main research areas include high-performance computing, system expansion, and large-scale data processing, AI for medical big data.