The Analysis of Next-Generation HPC Architecture Based on CXL through RAMSES-HR5 Performance Optimization

1st Hyun Mi Jung
center for supercomputing technology development
Korea Institute of Science and Technology Information
Daejeon, Korea
hmjung@kisti.re.kr

2nd Hyunjo Lee
dept. General Education
Korea National University of Agriculture and Fisheries
Jeonju, Korea
o2near@gmail.com

3rd Cheol-Joo Chae
dept. General Education
Korea National University of Agriculture and Fisheries
Jeonju, Korea
chae.cheoljoo@gmail.com

***Abstract*—High-performance computing (HPC) is an essential tool for solving complex and massive computational problems in advanced scientific research fields such as application simulations in astronomy, climate science, and particle physics. This is because simulations require maximizing floating-point operations and extensive parallel processing based on large-scale data. As HPC systems evolve into heterogeneous environments combining CPUs and GPUs, new performance bottlenecks beyond simple computational overhead have emerged as major challenges. Specifically, overhead from data memory copying between CPUs and GPUs, system-wide data bottlenecks, and increased computational latency and energy consumption are becoming pervasive. Resolving these inefficiencies to save time and costs while enhancing research efficiency is the primary goal. Therefore, this study analyzes performance optimization strategies for the HPC (High-Performance Computing) application simulation code RAMSES-HR5 to design and analyze a CXL-based next-generation HPC architecture. First, code profiling analysis revealed that the primary bottlenecks in the existing system stem from synchronization and communication delays based on OpenMP/MPI, rather than computational overhead. Excessive wait times were particularly observed in the *_kmp_fork_barrier* and *pmpi_allreduce_* functions. To address this issue, we propose solutions including merging OpenMP parallelization regions, utilizing asynchronous MPI communication, and changing the AMR (Adaptive Mesh Refinement) index structure to a hash-based approach. We also explore leveraging next-generation devices like CXL (Compute Express Link) to resolve memory bottlenecks and maximize data sharing efficiency. These approaches are expected to enhance the performance of RAMSES-HR5 and further contribute to building a simulation environment optimized for next-generation HPC system architectures.**

***Keywords*—HPC, CXL, Next generation HPC system, HPC application**

I. INTRODUCTION

HPC application simulations are essential for solving complex and massive computational problems across diverse fields such as astronomy, climate science, and particle physics. These simulations rely on large-scale data and require parallel processing support and maximized floating-point operations. However, in heterogeneous system environments, issues arise including overhead from memory copying between CPUs and GPUs, data bottlenecks, computational delays, and increased energy consumption[1]. Therefore, optimizing system performance to save time and costs while enhancing research efficiency is crucial. This paper presents a multidimensional solution combining code-level improvements (OpenMP and

MPI) with structural innovations Adaptive Mesh Refinement (AMR) indexing and Compute Express Link (CXL) integration based on next-generation hardware architectures to resolve the severe communication and synchronization bottlenecks in RAMSES-HR5[2]. Specifically, it thoroughly examines the technical feasibility of fundamentally resolving memory bottlenecks within heterogeneous systems and maximizing data sharing efficiency by leveraging CXL technology. Chapter 2 of this paper provides an overview of CXL and explores HPC utilization strategies for different CXL device types. Chapter 3 analyzes the hotspots of the HPC application simulation RAMSES-HR5 to design computational optimization strategies. Chapter 4 designs performance improvement strategies using enhanced OpenMP and MPI communication, along with performance optimization approaches leveraging CXL. Chapter 5 presents the conclusions and future research directions.

II. CASE STUDY

A. Overview of CXL Protocols (CXL.cache, CXL.mem) and Coherence Mechanisms

CXL is an open-standard interconnect designed for high-speed, high-capacity CPU-to-device and CPU-to-memory connections in high-performance data center computing. CXL builds upon the existing PCI Express (PCIe) physical and electrical interfaces but overcomes the limitations of PCIe by introducing a new cache coherence protocol. The core protocols of CXL are as follows:

- *CXL.io:* A PCIe-based block I/O protocol.
- *CXL.mem:* Enables the CPU to access extended memory (CXL memory) connected to CXL devices. This overcomes the capacity and socket packaging limitations of standard DIMM memory.
- *CXL.cache:* Maintains cache coherence between CPUs and accelerators (GPUs, DPUs, etc.), ensuring data consistency across heterogeneous computing domains.

By managing cache coherence at the hardware level, CXL provides a new paradigm that integrates the CPU, various accelerators, and extended memory components into a single, unified, and consistent memory domain[3-5].

This research was supported by Korea Institute of Science and Technology Information (KISTI). (No. K25L1M2C2)

B. HPC Utilization Approaches by CXL Device Type

The CXL specification classifies devices into three types based on their functionality and role, each performing a distinct function within HPC architectures[3-5].

- *Type 1 (Accelerator):* Possesses its own fully coherent cache but lacks globally visible local memory (Device Local Memory). Type 1 devices extend the functionality of the PCIe protocol, particularly by maintaining cache coherence when accessing host memory. They are used to replace traditional DMA-based incoherent accelerators, thereby optimizing data sharing with the CPU.
- *Type 3 (Memory Expander):* Possesses only host-managed device memory (HDM) and is primarily used to expand memory capacity via CXL.mem transactions. This overcomes server memory channel limitations and the constraints of increasing DRAM density, enabling large-scale memory expansion and memory pooling.
- *Type 2 (Heterogeneous Accelerator):* The most complex type, comprising devices like GPUs, ASICs, and FPGAs that attach their own high-bandwidth memory (DDR or HBM). Type 2 devices may optionally have a cache and expose their memory to the CPU via Host-Managed Device Memory (HDM). Simultaneously, like Type 1, they can access CPU memory consistently using CXL.cache. This bidirectional memory sharing support is essential for dramatically improving the performance of heterogeneous HPC workloads.

Table 1 below compares the roles of CXL device types within HPC architectures.

TABLE I. COMPARES THE ROLES OF CXL DEVICE TYPES WITHIN HPC ARCHITECTURES.

CXL Device Type	Primary Protocol	Cache Coherence (Host)	Key HPC Use Cases
Type 1 (Accelerator)	CXL.cache, CXL.io	Fully Coherent	Accelerator with its own cache (Coherent DMA replacement)
Type 2 (Accelerator/ Memory)	All (Cache, Memory, I/O)	Optional Coherence	GPU, FPGA, etc.
Type 3 (Memory Expander)	CXL.mem, CXL.io	Host-Managed Memory (HDM)	Large-scale memory expansion and pooling (Capacity expansion)

C. CXL Memory Pooling: Establishing a global data sharing environment between heterogeneous processors (CPU/GPU)

CXL's Memory Pooling feature is a core technology that resolves data sharing inefficiencies in HPC environments. CXL consolidates memory resources distributed across the system into a single unified pool. Through this global memory pool, CPUs, GPUs, and other accelerators can dynamically allocate and utilize memory resources as needed. This represents a strategic approach to break away from traditional siloed memory structures, maximize data sharing efficiency between heterogeneous components, and resolve memory bottlenecks.

III. RAMSES-HR5 HOTSPOT ANALYSIS

This study performed performance analysis using Intel VTun Profiler on RAMSES-HR5, one of the HPC application simulation codes. RAMSES is a code used for large RAMSES-HR5, using Intel Vtune Profiler. RAMSES is a code used for large-scale astrophysical and cosmological simulations, based on the Adaptive Mesh Refinement AMR technique and MPI + OpenMP hybrid parallelization.

Figure 1 below shows the total profiling time, and Figure 2 displays the TOP5 results from the profiling.

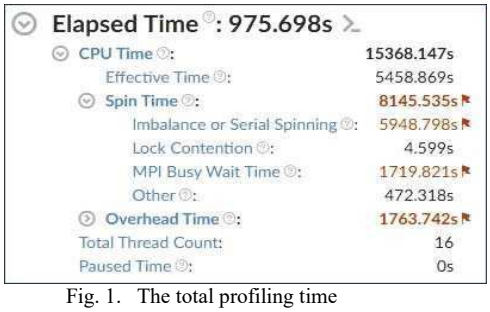

Fig. 1. The total profiling time

Function	Module	CPU Time	% of CPU Time
__kmp_fork_barrier	libiomp5.so	6311.070s	41.1%
pmpi_allreduce_	libmpifort.so.12	1998.537s	13.0%
__kmp_fork_call	libiomp5.so	1706.718s	11.1%
pmpi_waitall_	libmpifort.so.12	1141.301s	7.4%
__intel_avx_rep_memset	ramses3d	417.669s	2.7%
[Others]	N/A*	3792.852s	24.7%

*N/A is applied to non-summable metrics.

Fig. 2. The displays the TOP5 results from the profiling

Analysis revealed that approximately 53% (8145.535 seconds) of the total CPU execution time (15368.147 seconds) was spent in spin time. This indicates that bottlenecks occurring during parallelization and communication processes are more severe than the computational tasks themselves. Profiling results show the top four functions consuming the majority of CPU time are as follows:

- **_kmp_fork_barrier(41.1%):** Excessive calls occurred during OpenMP parallel region creation and thread synchronization.
- **pmpi_allreduce_(13.0%):** Excessive global reductions during Poisson iteration calculations.
- **_kmp_fork_call(11.1%):** Repeatedly called during entry into OpenMP parallel regions.
- **pmpi_waitall_(7.4%):** Concentrated wait time for asynchronous communication completion during ghost cell data exchange.

These results clearly indicate that performance optimization for RAMSES-HR5 should focus on improving the communication-intensive structure rather than computation-intensive performance enhancements.

IV. Study on HPC Application Performance Improvement through CXL Adoption

A. Performance Improvement Direction for HPC Simulation Code (RAMSES-HR5)

HPC application simulation codes utilize complex algorithms and large-scale data structures, making in-depth application-level analysis and code redesign essential for optimization on new hardware architectures like CXL. The RAMSES-HR5 case study demonstrates the importance of this approach. RAMSES-HR5 is based on MPI + OpenMP hybrid parallelization[6,11]. Performance analysis identified communication and synchronization delays as key bottlenecks. To address this, the following specific code improvements were implemented. Figure 3 below is an example of code improvement for OpenMP , and Figure 4 is an example of code improvement for MPI Communication [12,13].

```
!$omp parallel default(shared)
do it = 1, nsteps
  !$omp single
  call post_halo_irecv_isend(level_info, reqs)

  !$omp do schedule(dynamic,4) nowait
  do blk = 1, nblocks_interior
     call hydro_sweep(blk)
  end do

  !$omp do schedule(dynamic,1) nowait
  do lvl = 1, nlevels
     call refine_or_coarsen(lvl)
  end do

  !$omp single
  call halo_waitall_and_apply(reqs)

end do
!$omp end parallel
```

Fig. 3. An example of code improvement for OpenMP

```
integer :: req(2), stat(MPI_STATUS_SIZE)
logical :: done
do k = 1, max_iter
  ! 1) Local Calculation (Preparing Values for Reduction)
  call spmv(A, p, Ap)
  alpha_local = dot_product(p, Ap)
  r2_local    = dot_product(r, r)

  ! 2) Start global reduction asynchronously
  call MPI_Iallreduce(...)
  call MPI_Iallreduce(...)

  ! 3) Execute internal calculations for overlapping

  call prefetch_next_block_data()
  call compute_interior_cells()
  call apply_rhs_smoothing()

  ! 4) Wait for completion only when results are necessary

  call MPI_Wait(req(1), stat, ierr)
  call MPI_Wait(req(2), stat, ierr)
...
end do
```

Fig. 4. An example of code improvement for MPI Communication

- OpenMP Improvements: To reduce the overhead of the existing Fork/Barrier approach, which creates/terminates parallel regions per loop, the entire timestep is consolidated into a single parallel region. No wait and task loop are used to eliminate unnecessary synchronization points. This is expected to reduce Fork/Barrier calls by 15% to 40%.
- MPI Communication Improvements: To reduce excessive MPI All reduce calls, we use asynchronous communication via MPI_Iallreduce to overlap communication with CPU computation. To decrease NIC overhead from small packet communications, we bundle multiple messages into derived types. These improvements aim to reduce latency by 10% to 20%.

B. Optimization of Adaptive Message Refinement (AMR) Structure and CXL Utilization

AMR is a core HPC technology that reduces computational and memory requirements by applying high resolution only to specific complex regions within a simulation. However, the performance of AMR-based applications is often hindered by inefficiencies in the data structure[7,8].

Existing Oct-tree-based AMR index structures suffer from increased cache misses as pointer traversal depth grows, and cause load imbalance issues due to recursive searches. AMR's dynamic data structures, often reaching hundreds of megabytes in size with frequent random accesses, tended to be processed on the CPU rather than the accelerator.

CXL technology presents a structural alternative to resolve these fundamental bottlenecks in AMR.

- Innovation in Index Structure: First, the complex Oct-tree-based index structure is replaced with a hash-based index structure that maps each cell by converting it into a Morton key (Z-order). This transforms significantly improves cache efficiency by converting parent/child/neighbor node lookups into simple bit operations and hash lookups.
- Utilizing CXL Global Memory Pool: The improved hash-based index structure is deployed within a global memory pool leveraging CXL technology.

C. Maximizing Data Sharing Efficiency in Heterogeneous Environments

By placing the hash-based AMR index structure in the CXL global memory pool, multiple CPUs and GPUs can simultaneously access this index structure without data copying. CXL's cache coherence feature moves the complex pointer tracking paths of dynamic data structures into a high-speed shared memory space, maximizing parallelization efficiency. Particularly when Halo (boundary) data exchange occurs, using the CXL pool allows only the data array itself to be transferred, with the actual data referenced directly from the pool. This eliminates the overhead of traditional memory copying and minimizes I/O and communication bottlenecks, significantly enhancing the scalability and performance of HPC simulations. The cases of RAMSES-HR5 and AMR demonstrate that adopting CXL technology requires more than just hardware replacement. Fundamental co-design and redesign of the application's core algorithms and data structures are essential conditions for success.

V. Conclusion and Future Work

A. Authors and Affiliations Expected Benefits from OpenMP/MPI Code Improvements and CXL Integration for Comprehensive Performance Enhancement

The multifaceted optimization strategy proposed for RAMSES-HR5 provides an integrated solution to address the chronic issues that have hindered the performance of the existing simulation code. The code-level improvements for OpenMP and MPI improvements directly target synchronization and communication wait times (Spin Time), which accounted for 53% of total execution time. Through reducing fork/join overhead, mitigating load imbalance, enabling asynchronous communication overlap, and message vectorization, we anticipate performance gains exceeding 20%.

By integrating CXL-based structural innovations on top of these code improvements, performance gains are further maximized. Placing the hash-based AMR index structure within CXL's global memory pool and enabling heterogeneous processors to access it directly while maintaining cache coherence revolutionarily improves RAMSES's most inefficient memory access and data sharing processes. This reflects how HPC application optimization is shifting beyond merely maximizing CPU or GPU computational power, transitioning toward a data-centric architecture focused on data access speed and communication efficiency.

B. Proposed Future HPC Architecture through Role Division: CXL (Data Sharing) - DPU (Communication) - CPU/GPU (Computing)

Ultimately, the approach proposed in this study contributes to building a simulation environment optimized for next-generation HPC system architectures. Future HPC systems will evolve toward maximizing overall system efficiency through role specialization among dedicated devices:

- CPU/GPU (Computational): Dedicated to complex physics models and floating-point operations, maximizing computational efficiency in a low-latency environment.
- CXL (Data Sharing): Integrates memory hierarchies across heterogeneous devices via a high-speed, cache-coherent interconnect, providing a Global Memory Pool function to eliminate memory access bottlenecks.
- DPU (Data Processing Unit) (Communication): Offloads long-distance/inter-rack MPI communication protocol processing and I/O tasks from the CPU, enabling the CPU to focus purely on computational tasks and ultimately alleviating communication bottlenecks.

CXL serves as the core interconnect in this new architecture, minimizing data movement and ensuring memory access consistency. It presents a blueprint for overcoming the scalability limitations of communication-intensive, large-scale simulation codes like RAMSES.

This study proposed specific solutions to address synchronization and communication issues, which are key bottlenecks hindering the performance of HPC application simulations. Alongside code improvements for OpenMP and MPI communication structures, it suggested structural alternatives: transitioning the AMR index structure to a hash-based approach and introducing CXL technology. These approaches are expected to establish a simulation environment optimized for next-generation HPC architectures, where computation is distributed to CPUs/GPUs, data sharing to CXL, and communication to DPUs. Future research should directly implement the proposed solutions into the RAMSES code to validate their actual performance improvement effects.

References

[1] Hyun Mi Jung at al. , “A Study on GPU Parallelization and Performance Optimization of the Athena++ Simulation Code in High-Performance Computing Environments,” Smart Media Journal, vol 14 no6,2025.

[2] Jaehyun Lee and Jihye Shin at al. “The Horizon Run 5 Cosmological Hydrodynamical Simulation: Probing Galaxy Formation from Kilo- to Giga-parsec Scales”, arXiv:2006.01039 , 5 Oct 2023

[3] Daniel S. Berger, Yuhong Zhong, Fiodar Kazhamiaka, Pantea Zardoshti, Shuwei Teng, Mark D. Hill, Rodrigo Fonseca , “Octopus: Scalable Low-Cost CXL Memory Pooling” arXiv, [Online]. Available: https://arxiv.org/html/2501.09020v1

[4] Synopsys Blog, “How CXL and Memory Pooling Reduce HPC Latency” . [Online]. Available: https://www.synopsys.com/blogs/chip-design/cxl-protocol-memory-pooling.html

[5] Danny Moor, Debendra Das Sharma,”Enabling composable systems with expanded fabric capabilities”, [Online]. Available: https://computeexpresslink.org/wp-content/uploads/2023/12/CXL_3.0-Webinar_FINAL.pdf

[6] R. Teyssier, “Cosmological hydrodynamics with adaptive mesh refinement: The RAMSES code,” Astronomy & Astrophysics, vol. 385, no. 1, pp. 337‑364, 2002.

[7] M. J. Berger and J. Oliger, “Adaptive mesh refinement for hyperbolic partial differential equations,” Journal of Computational Physics, vol. 53, no. 3, pp. 484‑512, 1984.

[8] Z. Liu, F. B. Tian, and X. Feng, “An efficient geometry-adaptive mesh refinement framework and its application in the immersed boundary lattice Boltzmann method,” Computer Methods in Applied Mechanics and Engineering, vol. 392, p. 114662, 2022.

[9] Message Passing Interface Forum, “MPI: A Message-Passing Interface Standard, Version 4.0,” 2021.

[10] T. Mattson et al., “A Hands-on Introduction to OpenMP,” OpenMP Tutorial, National Center for Supercomputing Applications (NCSA), 2008.

[11] D. L. Caballero de Gea et al., “Barriers and reductions in OpenMP,” Technical Report, Universitat Politècnica de Catalunya, 2015.

[12] Hyunjo Lee et al., “A Study on Code Modification for Hotspot Optimization in OpenMP Parallel Regions in the RAMSES Simulator”, 2025 Autumn Academic Conference of Smart Media, 2025.

[13] Hyunjo Lee et al., “A Study on Code Modification Strategies to Improve MPI Communication Bottlenecks in the RAMSES Simulator”. 2025 Autumn Academic Conference of Smart Media, 2025.