Proposal of a Transparent Dynamic Core Binding Method for MPI/OpenMP Hybrid Parallelization

1st Yusuke Ishihara
Tohoku University
Sendai, Japan
ishihara.yusuke.s4@dc.tohoku.ac.jp

2nd Masatoshi Kawai
Tohoku University
Sendai, Japan
m.kawai@tohoku.ac.jp

3rd Sichen Tao
Tohoku University
Sendai, Japan
sichen.tao@tohoku.ac.jp

4th Hiroyuki Takizawa
Tohoku University
Sendai, Japan
takizawa@tohoku.ac.jp

I. Introduction

In MPI/OpenMP applications on modern HPC systems, mitigating load imbalances is critical. Unbalanced workloads cause CPU wait times, underutilizing resources and wasting energy. Dynamic Core Binding (DCB) mitigates this by dynamically binding CPU cores based on computational load (see Fig. 1). However, traditional DCB requires manual code modifications, increasing developer burden. This paper proposes a fully transparent DCB method utilizing OMPT and PMPI interfaces to eliminate user-level modifications, and quantitatively evaluates its performance and energy efficiency on a modern supercomputer.

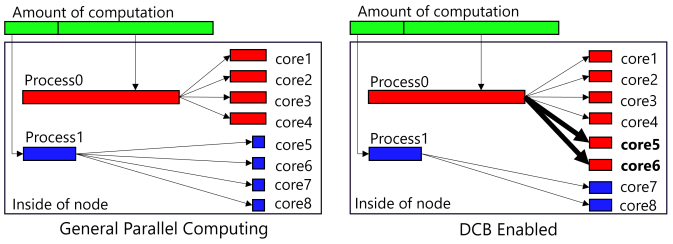


Fig. 1. Conceptual overview of Dynamic Core Binding (DCB) addressing process-level load imbalance.

II. Transparent Dynamic Core Binding

A. OMPT and PMPI Integration

The proposed method leverages OpenMP Tools (OMPT) and Profiling MPI (PMPI) standard interfaces via a dynamically linked shared library (Fig. 2). OMPT callbacks automatically intercept computation phases without altering the binary. Concurrently, PMPI intercepts MPI calls to insert initialization and cleanup routines. This dual-interception ensures dynamic resource optimization while maintaining original application behavior.

B. Dynamic Parameter Calculation

A critical component of DCB is determining the appropriate load parameters for core binding. Traditional DCB methods rely on static, user-defined metrics, which are difficult to estimate accurately for complex, non-linear computational tasks. The combination of OMPT and PMPI enables the automatic calculation of the load parameters based on actual computation time. By subtracting the MPI wait time captured by PMPI from the execution time of OpenMP parallel regions measured by OMPT, the proposed method obtains a highly accurate estimate of the computational load of each process.

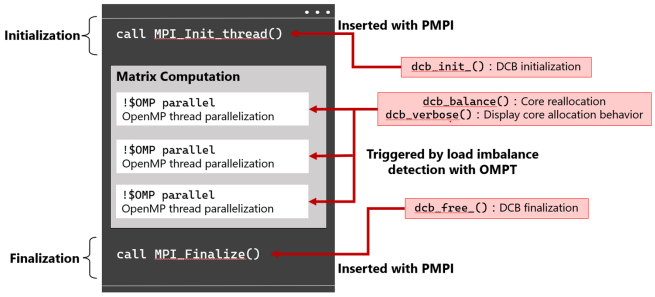


Fig. 2. Architecture of the proposed fully transparent DCB method using OMPT and PMPI

III. Evaluation Setup

A. Evaluation Environment

Evaluations were conducted on the Miyabi-C supercomputer. Each node is equipped with an Intel Xeon CPU Max 9480 processor (112 cores per node) and 128 GB of HBM2e memory. The baseline setup utilizes a static mapping, assigning 4 MPI processes per node, with 28 physical cores and OpenMP threads distributed evenly to each process.

B. Application and Datasets

We utilized a Lattice H-matrix application. H-matrices hierarchically partition dense matrices and apply low-rank approximations to off-diagonal blocks, significantly reducing computational and memory costs. However, variations in approximation ranks introduce extreme computational load imbalances among processes, making it an ideal test case for dynamic load balancing. Evaluations were conducted using multiple datasets, focusing primarily on the large-scale `input_1188kp25` dataset ($nd = 1,188,000$).

IV. Evaluation Results and Discussion

A. Single-Node Performance

Initial testing on a single node demonstrated the performance benefit of the proposed method on large-scale datasets

compared to both the static baseline and the conventional DCB (mem8) approach. For large datasets, the proposed transparent DCB reduced execution time, achieving a maximum speedup of 1.54x as shown in Fig. 3. Furthermore, an in-depth analysis of the processing phases (see Fig. 4) revealed that while the conventional DCB achieved a 1.35x speedup over the static baseline (without DCB) in the total execution time, the proposed method further improved performance. Specifically, the single-iteration calculation time was reduced to approximately 0.6x to 0.7x of the baseline. This improvement is directly attributed to the mitigation of synchronization wait times in lightly loaded processes, as surplus cores were effectively redistributed to heavily loaded processes. Unlike the conventional DCB that relies on static computation volumes, the proposed method utilizes actual execution times as load parameters, enabling it to accurately reflect effective performance differences caused by dynamic cache hit rates in memory-bound applications like the Lattice H-matrix.

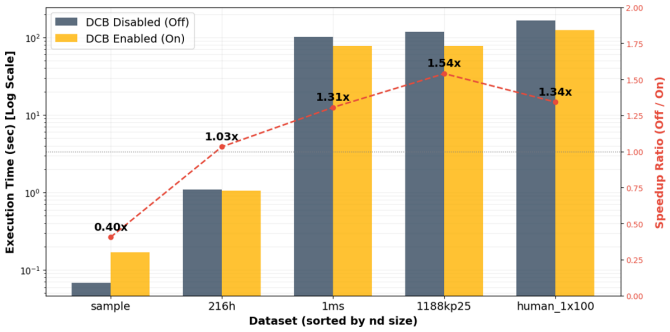


Fig. 3. Execution time and speedup ratio when applying DCB on a single node.

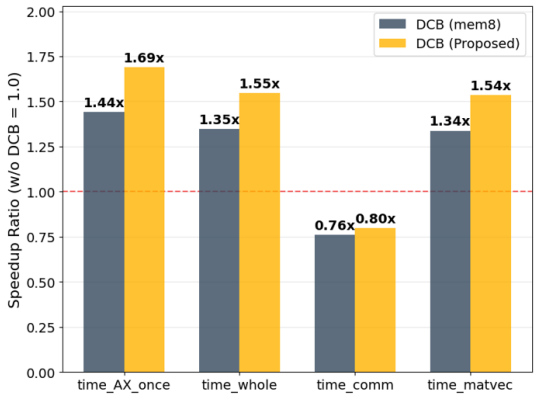


Fig. 4. Performance comparison between the conventional DCB (mem8) and the proposed DCB method on a single node.

Furthermore, despite a slight increase in active power consumption (Watts) due to denser core utilization during peak computation phases, the reduced execution time lowered total energy consumption (Joules) by up to 32.0

B. Multi-Node Performance

To confirm scalability, the methodology was further evaluated in a 16-node distributed memory environment (totaling 1,792 cores and 64 MPI processes). A specific hybrid policy was applied to balance both computational speed and power reduction across nodes. The proposed method reduced the overall execution time by 6.1% and total energy consumption by 10.6%. The average power consumption also decreased by 4.8%, proving that the automatic transition of surplus cores into a deep sleep state effectively suppresses idle power waste without degrading the performance of the critical path.

V. Conclusion

This study proposed a transparent dynamic load balancing method for MPI/OpenMP hybrid applications. By leveraging OMPT and PMPI, this method accurately monitors execution states and reallocates cores automatically via dynamic linking, successfully eliminating the need for manual source code modifications. Experimental evaluations on the Miyabi-C system confirmed that the method accurately captures dynamic computational loads, yielding up to a 1.54x speedup and a 32.0% reduction in energy consumption on a single node, with verified scalability in large-scale multi-node environments.

Acknowledgment

The authors utilized ChatGPT (OpenAI) and Gemini (Google) for proofreading and language refinement; however, the authors bear full responsibility for the final content.