A Case for Co-scheduling for Hybrid Memory Based Systems

[Extended Abstract]

Eishi Arima
The University of Tokyo
Tokyo, Japan
arima@cc.u-tokyo.ac.jp

Carsten Trinitis
Technical University of Munich
Munich, Germany
Carsten.Trinitis@in.tum.de

ABSTRACT

Manycore architectures and hybrid memory designs using emerging memory technologies (e.g., 3D stacked DRAM and Non-Volatile RAM) have been adopted in recent HPC systems, and the total system throughput/bandwidth have been significantly improved by them. However, at the same time, these technology trends incur more significant resource wastes within a node as CPU/memory intensive applications do not effectively utilize memories/CPUs. To mitigate the resource wastes, prior researches have shown that co-scheduling, i.e., launching multiple applications on one node, is a promising approach, but they mostly did not consider the combination of co-scheduling and hybrid memory systems. In this work, we explicitly target the combination and make a case for it. Our preliminary evaluation clarifies the following key insight: the problem size of each application plays an important role in optimizing co-scheduling strategies for hybrid memory based systems.

1 INTRODUCTION

HPC systems will continue to face the severe memory bandwidth/-capacity limitations, and **hybrid memory systems**, which are composed of multiple different memory technologies, are promising to combat the problem. On the one hand, 3D stacking memories, such as HBM [4], are effective to increase the memory bandwidth, but they suffer from serious capacity limitations [5]. On the other hand, Non-Volatile RAMs, such as 3D Xpoint memories [2, 3], are promising to considerably scale memory capacity, however, their performance is much lower than that of conventional DRAMs. Due to the trade-off in bandwidth and capacity, hybrid memory architectures are promising design choices [5, 8, 9].

Meanwhile, to improve the processor performance, the industry has gradually increased the core count in the last decade. As a consequence, many-core processors have appeared on the market [5] and are used in various HPC systems. However, at the same time, it has become more and more difficult to efficiently convert high core counts into performance due to the thread-level scalability limitations caused by sequential code portions, shared resource contentions, and intensive memory accesses.

Co-scheduling, i.e., launching multiple applications on one single node, is a well-known approach to mitigate such resource wastes [1, 6]. On the one hand, memory intensive applications need less processor cores as performance is restricted by the memory system (e.g., bandwidth). On the other hand, CPU intensive applications need more cores but consume less memory bandwidth. Therefore, the resources can be more efficiently utilized by simultaneously executing different kinds of applications on one single node.

Although prior studies have shown the effectiveness of co-scheduling, ***the combination of co-scheduling and hybrid memory systems has not been well-studied yet***. In this work, we explicitly target the combination and make a case for it by deploying a preliminary evaluation. In particular, we demonstrate the following key insight: ***the problem size of each application plays a key role in optimizing co-scheduling for hybrid memory based systems***.

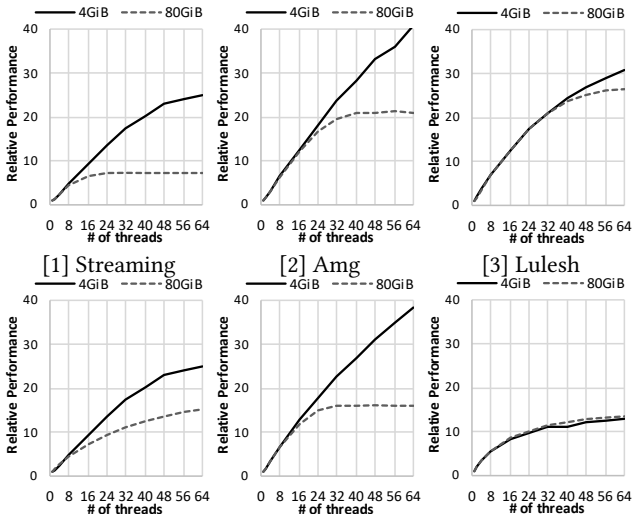

Figure 1: Thread-level scalability v.s. problem size

2 EVALUATION ENVIRONMENT

We target systems with emerging hybrid memories, which are heterogeneous in terms of bandwidth and capacity (e.g., MCDRAM + DDR4 in Knights Landing [5] or systems using Optane DIMM/SSD [2, 3] as DRAM extensions). In this work, we use a Knights Landing based system as an example whose settings are listed in Table 1. Note that our observations shown later are caused by the bandwidth/capacity heterogeneity in the memories and thus will happen on any of our target systems. In the evaluation, we utilize the hardware-based approach (cache mode) for the data management on the different memories. As for the workloads, we chose Streaming as well as several mini applications chosen from CORAL benchmark suite [7].

3 KEY OBSERVATION: THREAD-LEVEL SCALABILITY V.S. PROBLEM SIZE

Figure 1 demonstrates how thread-level scalability depends on the problem (or data) size. As shown in the figures, applications generally scale worse when the problem size is larger on our system.

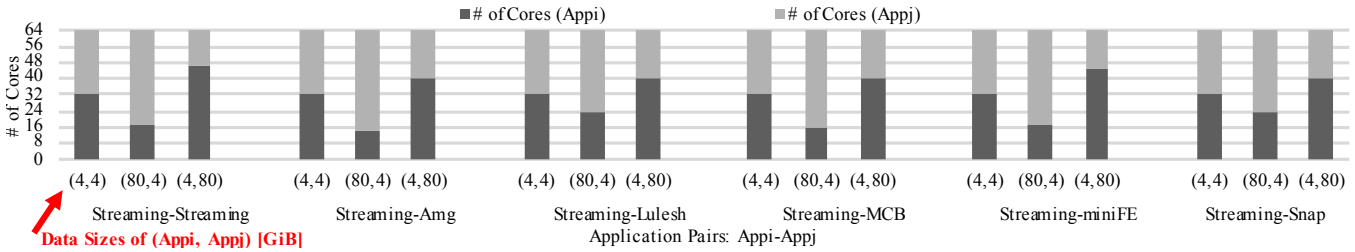

Figure 2: Optimal core resource allocations

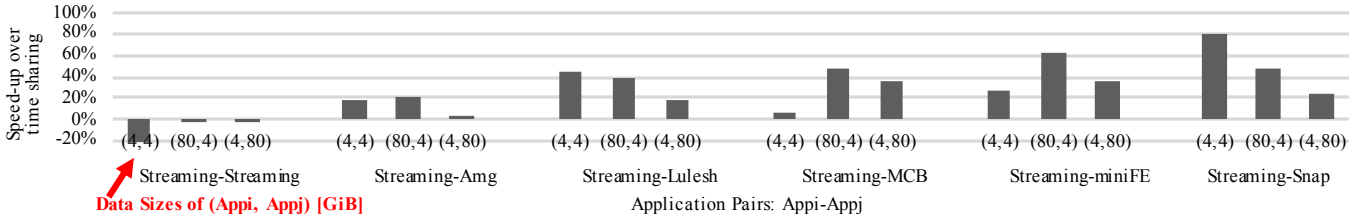

Figure 3: Performance improvement brought by co-scheduling

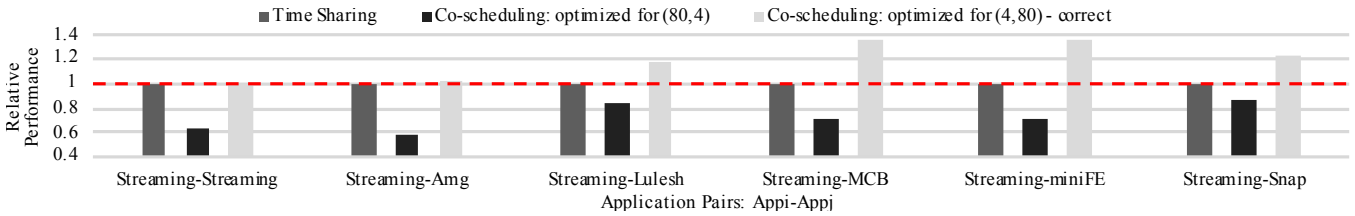

Figure 4: Performance comparison at (4,80) (= data sizes of (Appi, Appj)[GiB])

Name	Remarks
CPU	XeonPhi 7210, 64cores, 1.3GHz
Memory	MCDRAM (Fast Mem.): 16GiB, 450GiB/s DDR4 (Large Mem.): 96GiB, 90GiB/s
OS	CentOS 7
Compiler	ICC 19.0.1.144

Table 1: System configuration

This is because when the data size is increased, ***the large but slow memory is more frequently accessed, which shifts the performance bottleneck from the CPU or the fast memory to it.*** As for SNAP, the scalability is limited by the other reasons (e.g., interference among threads).

4 IMPACT ON CO-SCHEDULING

Figure 2 shows the optimal[1] core allocations to two applications when we execute them at the same time. As shown in the figure, this highly depends on the problem sizes of applications as the core resource requirements are strongly related to their thread-level scalability (see Figure 1). Then next, Figure 3 shows the speed-up over *Time Sharing*[2] when the core allocations are optimized. This figure suggests that ***the optimal co-scheduling pair choices highly depend on the problem sizes.*** Finally, Figure 4 describes the performance comparison, which suggests that ***a non-problem-size-aware core allocation approach can considerably degrade performance that can be even worse than Time Sharing.***

[1]Optimal: minimizing total execution time
[2]Time Sharing: conventional exclusive solo-runs

5 CONCLUSION AND FUTURE CHALLENGES

In this work, we made a case for co-scheduling on a hybrid memory based system and demonstrated that the optimal co-scheduling pair selections and resource allocations highly depend on the problem sizes of applications. In our future work, we are going to develop a problem size aware co-scheduling algorithm and extend a conventional co-scheduling framework to support it.

ACKNOWLEDGEMENTS

We thank Prof. Martin Schulz and the members of his research group at TU Munich. This work is partly supported by JSPS Grant-in-Aid for Early-Career Scientists (JP18K18021), and Research on Processor Architecture, Power Management, System Software and Numerical Libraries for the Post K Computer System of RIKEN.

REFERENCES

[1] Bhadauria, M., et al. An Approach to Resource-aware Co-scheduling for CMPs. In *ICS* (2010), pp. 189–199.
[2] Intel. Intel® Memory Drive Technology, Set Up and Configuration Guide, 2019.
[3] Izraelevitz, J., et al. Basic Performance Measurements of the Intel Optane DC Persistent Memory Module. *arXiv preprint arXiv:1903.05714* (2019).
[4] JEDEC Standard. High Bandwidth Memory (HBM) DRAM. *JESD235* (2013).
[5] Jeffers, J., et al. *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition.* Morgan Kaufmann Publishers Inc., 2016.
[6] Jiang, Y., et al. Analysis and Approximation of Optimal Co-Scheduling on Chip Multiprocessors. In *PACT* (2008), pp. 220–229.
[7] ORNL, ANL, and LLNL. CORAL Benchmark Codes. https://asc.llnl.gov/CORAL-benchmarks/, 2013.
[8] Vetter, J. S., et al. Opportunities for Nonvolatile Memory Systems in Extreme-Scale High-Performance Computing. *Computing in Science Engineering 17*, 2 (2015), 73–82.
[9] Vijayaraghavan, T., et al. Design and Analysis of an APU for Exascale Computing. In *HPCA* (2017), pp. 85–96.