Practicing the benefits of co-execution in HPC

Nikolaos Triantafyllis
ntriantafyl@cslab.ece.ntua.gr
National Technical University of Athens

Efstratios Karapanagiotis
skarapan@cslab.ece.ntua.gr
National Technical University of Athens

Alexios Papavasileiou
apapavas@cslab.ece.ntua.gr
National Technical University of Athens

Georgios Goumas
goumas@cslab.ece.ntua.gr
National Technical University of Athens

Nectarios Koziris
nkoziris@cslab.ece.ntua.gr
National Technical University of Athens

1 Introduction

In MPI applications, processes are spawned to execute commonly identical code on distinct data, resulting in a homogenized behavior that often imposes similar demands on the system. In typical HPC infrastructures, dedicated node allocation is a standard strategy, therefore, MPI applications are more likely to obstruct each other's progress as they contend for shared resources, such as LLCs, and memory and network bandwidth. On the other hand, node sharing revolves around the idea that applications can coexist when they do not concurrently exert pressure on the same resources, which could lead to improved performance [2]. According to our work, this approach seems to have a notable impact on system utilization, especially as compute nodes continue to scale up, ultimately resulting in cost and energy savings.

2 Co-executing in operational HPC facilities

We conducted co-execution experiments of various application kernels of the NPB and the SPEChpc 2021 benchmarks in the GRNET/ARIS and the CINECA/Marconi supercomputers. Applications ran repeatedly in compact mode and in pairs, obtaining full and half CPU socket capacity each, respectively. Speedup for each application is calculated as the ratio of the median execution time of the compact runtime to the median execution time of the co-executed scenario runtime. The practical observation is that co-execution contributes to an over 13% increased average speedup. Additionally, co-executed pairs should be chosen wisely, because improper selection can lead to unilateral or bilateral performance degradation, which is a known fact [1]. Our experiments involve applications under diverse computational kernels, different problem sizes, and various numbers of spawn MPI processes, strengthening the requirement for a co-execution policy.

3 Simulating an HPC Scheduler

Due to the inherent challenges of conducting real-world experiments, simulations are required to study the complex problem of co-scheduling. Leveraging collected data, our simulation framework seeks to determine the viability of a co-scheduling method and assess its possible performance benefits. It utilizes the measurements from our experiments to construct virtual workloads and try various job scheduling approaches with the use of heuristic functions for sub-optimal decision-making. It is worth noting that the software provides a Gantt representation of the workload runtime, including a Boxplot analysis of the related applications' speedups.

4 Integrating co-execution mechanisms in a real-world resource manager

OAR3 is a flexible resource and job manager targeted for HPC clusters. Within it, we developed new resource allocation policies oriented toward co-execution, whereas the new enhanced code was deployed on an HPC cluster. Since job pairs need to be selected appropriately, we designed a job's life cycle prototype within OAR3 where a job can be assigned to a resource allocation policy based on its performance and MPI counters by ML-driven decisions. The ML model was built on a small dataset of performance and MPI counters harvested from runs of the previously mentioned benchmarks in compact mode.

5 Future Work

As for our next step, we are considering, running experiments in more HPC facilities with more MPI benchmarks. Concerning the simulation, we are examining supporting dynamic waiting queues, code optimizations, and implementing more targeted co-scheduling approaches. Regarding the OAR3 integration, the development of job queue and system-aware resource allocation policies are to be investigated.

6 Acknowledgement

This work was co-funded by the European High-Performance Computing Joint Undertaking and the General Secretariat of Research and Innovation, Ministry of Development & Investments under the project REGALE (G.A. No 956560).

References

[1] Andreas de Blanche and Thomas Lundqvist. 2016. Terrible twins: A simple scheme to avoid bad co-schedules. In *Proceedings of the 1st COSH Workshop on Co-Scheduling of HPC Applications*. 25.

[2] Alex D Breslow, Leo Porter, Ananta Tiwari, Michael Laurenzano, Laura Carrington, Dean M Tullsen, and Allan E Snavely. 2016. The case for colocation of high performance computing workloads. *Concurrency and Computation: Practice and Experience* 28, 2 (2016), 232–251.