Increasing Throughput of Multiprogram HPC Workloads:
Evaluating an SMT Co-Scheduling Approach

Elias Lundmark
University West
Sweden
elias.lundmark@outlook.com

Chris Persson
University West
Sweden
chris.persson@gmail.com

Andreas de Blanche
Department of Engineering Science
University West, Sweden
andreas.de-blanche@hv.se

Thomas Lundqvist
Department of Engineering Science
University West, Sweden
thomas.lundqvist@hv.se

Poster at Supercomputing 2017, Denver, USA, November 13-16, 2017.

Introduction

A bit surprising, many HPC-systems have SMT turned off, thus, limiting the scheduling slots in the system to one per core. Enabling hyperthreading, Intel's SMT technology, would double the number of available slots [1]. However, more slots do not automatically lead to higher performance. In [2], Tuck and Tullsen observed increases in system throughput ranging from 0.90 to 1.58 when enabling SMT and an average speedup of 1.20, when running the SPEC benchmarks. Thus, some of the benchmarks actually showed a slowdown of up to 10%. **The problem with obtaining high throughput lies in determining which programs should be co-scheduled, i.e. share a physical core**.

Several co-scheduling schemes aimed at achieving high throughput have been suggested. Snavely and Tullsen [3] achieved an average throughput increase of 17% using; Sample, Optimize, Symbiosis which samples hardware counters and uses heuristics to determine which processes are suitable to execute together. Eeckhout et al. [4] introduced a probabilistic model for co-scheduling on SMT processors. A simple approach needing no pre-runtime analysis is suggested in [5, 6] that identify instances of the same program as being more likely to heavily utilize the same resources in a compute node, like caches, memory buses, or disks, than a random set of programs. In [5-7] it was shown that disallowing several instances of the NAS benchmarks to be co-scheduled on the same compute node increases the throughput compared to random co-scheduling.

Our aim is to evaluate job co-scheduling principles:

- ❑ Will disallowing the co-scheduling of several instances of the same program affect the throughput positively?
- ❑ Will enabling SMT, practically doubling the number of available slots, will, increase throughput?
- ❑ Our evaluation assumes an HPC system that is used for many different programs.

Results

To evaluate a full system, we created 30 job cases, each consisting of multiple instances of two random SPEC CPU benchmarks.

- ❑ A baseline was established by executing six instances of each benchmark on one core each with SMT turned off (12 jobs on 12 cores), marked by X in Fig. 3.
- ❑ The job mixes were then executed using six and twelve cores with SMT enabled using two co-scheduling algorithms:
  - ➢ AllPairs: two identical processes are placed on each core
  - ➢ NoPairs: one process from each benchmark is placed on each core.
- ❑ Compared to the twelve-core baseline:
  - ➢ Six SMT cores (12 threads): 3 cases perform better than the baseline, on average 39% (AllPairs) and 38% (NoPairs) lower throughput.
  - ➢ Twelve SMT cores (24 threads): 20 cases perform better than the baseline using AllPairs and 18 performs better using NoPairs, on average 8% (AllPairs) and 10% (NoPairs) higher throughput

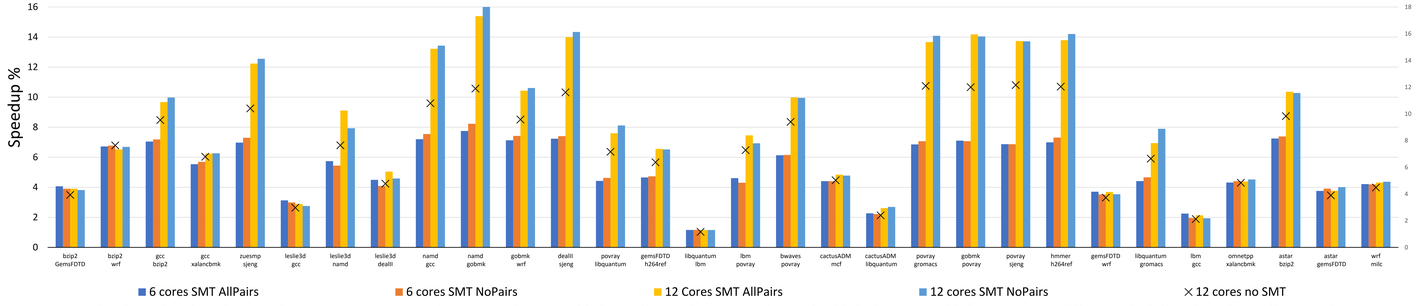
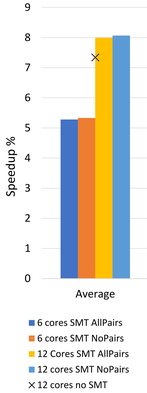


Figure 3. Speedup for 30 benchmark pairs when running on six or twelve cores with SMT enabled as well as twelve cores with SMT disabled. The SMT enabled pairs use two different scheduling algorithms, AllPairs and NoPairs.

Conclusion

We performed an experimental study comparing the throughput of 30 random SPEC CPU job mixed on a Broadwell-based node using two different co-scheduling strategies.

- ❑ A six-core processor with SMT enabled has 38% lower throughput than a twelve-core processor with SMT disabled.
  - ➢ Downgrading the processor is not beneficial.
- ❑ A twelve core processor with SMT enabled has 10% higher throughput than a twelve core processor with SMT disabled.
  - ➢ I.e. Enabling SMT is beneficial, on average.
- ❑ Using the NoPairs co-scheduling policy increased the throughput by, on average, two percentage points.
  - ➢ There has to be multiple programs running on the system for NoPairs to work.
  - ➢ For some combinations of jobs the AllPairs co-scheduling policy was better.



Hardware

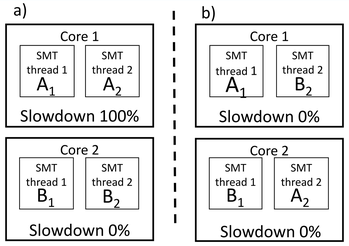


Figure 1. Example showing a bad SMT co-schedule (a) and a good SMT co-schedule (b) of multiple instances of job a, a computationally bound program and job B, a memory bound program.

The experiments were performed on a Xeon E5-2650 v4 Broadwell-based cluster. Each node has 32 GB ram and an E5-2650 processor with 12 physical cores and Intel's Hyperthreading with a SMT level of two. The nodes operating system is Centos 7.3.

All experiments rely on the benchmarks provided in the SPEC CPU 2006 benchmark suite [8]. During the experiments all frequency scaling techniques were turned off, i.e. both speed step and turbo frequency.

Full Factor Evaluation

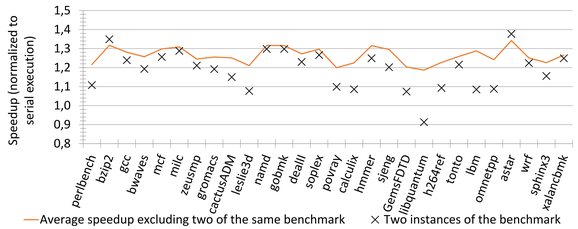


Figure 2. Average speedup for all SMT combinations excluding two instances of the same program. X marks the speedup of two co-scheduled instances of the same program. All numbers are normalized to the serial execution.

We evaluate the effect of disallowing the execution of two instances of the same SPEC benchmark on a SMT enabled core.

This was done by executing all possible combinations of the programs in the SPEC CPU benchmark suite and calculating the speedup each same-program pair gained compared to executing the program serially.

- ❑ Single-core SMT is beneficial in all cases where the X is below the line.
- ❑ In the entire test set, involving 784 combinations, only one has a negative speedup and that is when two instances of libquantum shares a core.
- ❑ There are only two cases, out of 28, where co-scheduling two instances of the same program has a positive effect on the average speedup. Hence, same program co-scheduling has a negative effect in 26 cases.

References

[1] L. Porter, A. Jundt, William A. Ward Jr, R. Campbell, M. A. Laurenzano, A. Tiwari, and L. Carrington. 2015. Making the Most of SMT in HPC. *ACM Transactions on Architecture and Code Optimization* 11.

[2] N. Tuck and D. M. Tullsen. 2003. Initial observations of the simultaneous multithreading pentium 4 processor. IEEE, International Conference on Parallel Architectures and Compilation Techniques.

[3] A. Snavely and D. M. Tullsen. 2000. Symbiotic job scheduling for a simultaneous multithreading processor. *ACM SIGPLAN Notices* 35..

[4] S. Eyerman and L. Eeckhout. 2012. Probabilistic modeling for job symbiosis scheduling on SMT processors. *ACM Transactions on Architecture and Code Optimization* 9.

[5] A. de Blanche and T. Lundqvist. 2015. Addressing characterization methods for memory contention aware co-scheduling. *Springer Journal of Supercomputing* 71.

[6] A. de Blanche and T. Lundqvist. 2016. Terrible twins: A simple scheme to avoid bad co-schedule. In *Workshop on Co-Scheduling of HPC Applications (COSH), HIPEAC.*

[7] A. de Blanche and T. Lundqvist. 2017. Disallowing same-program co-schedules to improve effciency in quad-core servers. In *Workshop on Co-Scheduling of HPC Applications (COSH), HIPEAC.*

[8] SPEC cpu2006. [n. d.]. Standard Performance Evaluation Corporation. ([n. d.]). https://www.spec.org/cpu2006/

[9] R. Hood, D. Barker, P. Mehrotra, S. Saini, H. Jin and R. Biswas. 2011. The impact of hyper-threading on processor resource utilization in production applications. In *International Conference on High Performance Computing.*