Matthew Francis-Landau
Nathan Pemberton
Sven Schwermer
University of California, Berkeley

Too Many Cooks in the Kitchen:

Gang Scheduling for Predictable Performance

Introduction

In the cloud, ease of use and interactivity are key. Typically, jobs are deployed on multi-core virtual machines. Each core of the VM is scheduled independently, leading to high variability in each thread's completion time. The HPC community requires predictable performance of every thread in order to minimize the time spent in barriers. To achieve this, supercomputers typically use batch-managers which run each job to completion. This maximizes the performance of each job, but may cause small jobs to wait behind larger jobs.

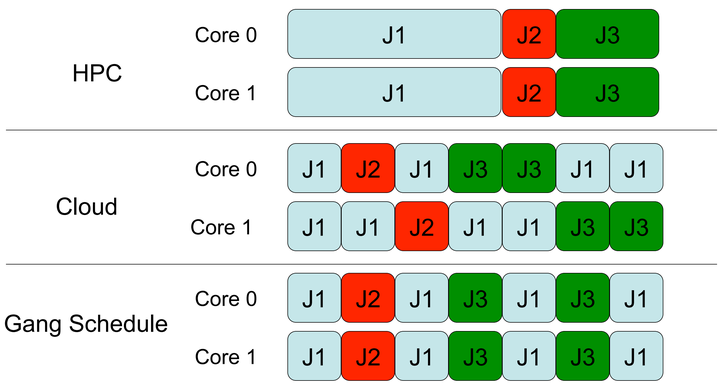

Figure 1: Example schedules under different policies. HPC uses a batch manager (such as Slurm or Torque). Cloud uses the default Xen scheduler which makes no attempt to gang schedule. The Gang Scheduled policy starts all of a job's threads at the same time.

This project seeks a middle ground, where programmers can rely on predictable performance of each thread (leading to efficient barriers) without having to wait for large jobs to finish before starting. We achieve this using gang scheduling, where all of a job's threads are started and ended together.

Experimental Setup

The benchmark ran Linux virtual machines on top of the Xen 4.4 hypervisor[2]. The Linux virtual machines used a stripped down Linux kernel 3.14 For the baseline cases we used Xen's default credit scheduler. For gang-scheduling we implemented a custom scheduler on top of Xen and ran the domains in an isolated CPU pool.

CoEVP

For our experiments we used the CoEVP proxy-application from the Exascale Co-design Center for Materials in Extreme Environments (ExMatEx)[1]. It works by spawning a number of fine-scale models, and interpolating their results into a coarse-grained model.

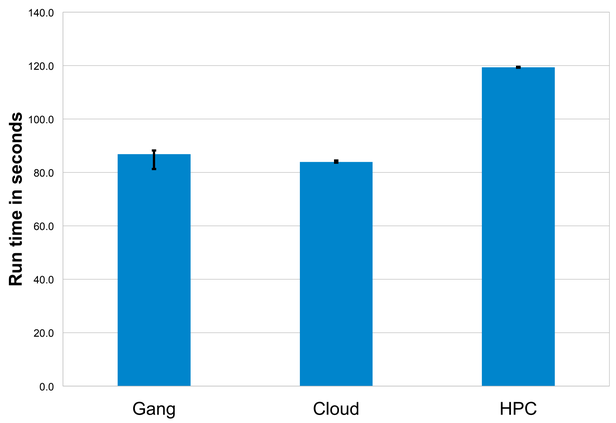

Figure 2: Average run-time of CoEVP under different scheduling policies.

Despite the focus on performance in the HPC community, batch processing produces the worst runtimes of any experiment. We speculate this performance difference is due to blocked threads in CoEVP. While a thread from one job is waiting for communication, other jobs could run.

Micro-Benchmark

To see the effects of gang-scheduling we implemented a micro-benchmark that performs a compute-bound task repeatedly. We then start this benchmark simultaneously on each core. In figure 3 we plot how many times a core completes more than 1 iteration (roughly 100ms) after the others on the two schedulers. As you can see from Figure 3, the credit scheduler has significantly higher variation in execution times with many out-of-sync threads.

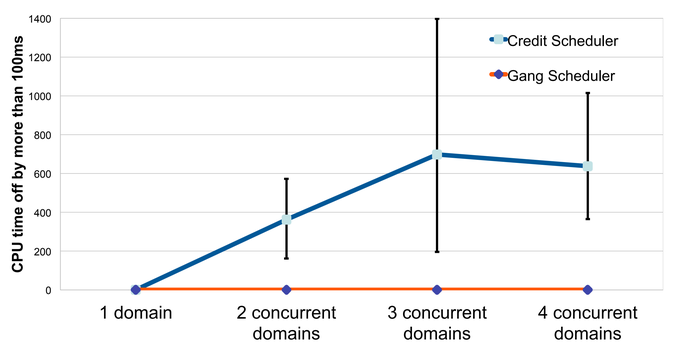

Figure 3: Variability in execution time with over-subscription of domains on Xen.

Conclusions

While gang scheduling does indeed improve the consistency of simple, compute-bound tasks like our micro-benchmark, complex benchmarks like CoEVP seem to benefit less. While this may be a fluke of CoEVP, it is clear that not all tasks will benefit from this style of scheduling. In the future we plan explore more flexible forms of scheduling that minimize barrier delays while still accounting for blocked tasks and the benefits of CPU oversubscription.

References

[1] https://github.com/exmatex/CoEVP
[2] http://www.xenproject.org/