Tuning application performance with dynamically scalable parallelism

Paper **#XX**

NN pages

Abstract

In recent years, economic and physical realities have pushed both chip designers and application developers to rely increasingly on parallelism to satisfy performance demands (3). However, there is no free lunch. Even after an application developer has wrestled with several target architectures, she still does not know what resources the application will get at runtime. Often, it is easier to over-provision units of execution and hope for the best. Yet even mild contention can drastically reduce overall system throughput and wall-clock execution times of individual applications. On the other hand, under-provisioning units of execution means leaving resources unused and falling short of performance potential. In this paper, we show that by selectively and dynamically changing application parallelism, we can avoid both pitfalls [in some way].

Stronger guarantees from scheduler to application (monitor actual resource utilization). Insights from application to scheduler (e.g., some information about app?).

1. Questions

- How do developers typically provision number of threads? How do they determine the maximum?

2. To do

- DS Paper outline
- MS AC Understand difference in performance for different levels of parallelism
- DS YW Understand Intel TBB
  - http://cas.ee.ic.ac.uk/people/dt10/teaching/2012/hpce/hpce-lec6-tbb-intro-handout-4up.pdf
  - https://drive.google.com/file/d/0B$_1$0$gtxnPV-_UFUyRGQ4MTQ4Y sharing$
- Design interface between TBB and coordinator
- Implement and integrate coordinator with TBB
- Integrate TBB into applications
  - http://wiki.cs.princeton.edu/index.php/PARSEC
- Evaluations

3. Introduction

What's the problem

- Parallel hardware + software hard to reason about
- Too many combinations to hand-tune
- Existing approach either over or under provision parallelism

Contributions

- User-level coordinator to determine appropriate level of parallelism for registered applications
- Thread library that can dynamically scale application parallelism

Not sufficient to set parallelism just at application start

4. Motivation for a dynamic parallelism

- Many cores are here to stay. Need more automated methods to take advantage of. (1)
- OS schedulers already quite complex. Must serve many masters; hard to make changes without harming certain workloads. Don't want to change the kernel if we can improve performance in user space (2)
- Applications are being increasingly run in containers / VMs on a single host
- Centralized schedulers in distributed systems struggle to scale (4)

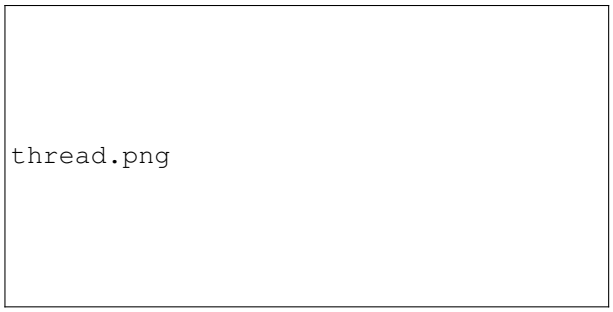

Figure 1. How groups of applications (same or mixed) do under contention at varying # of threads

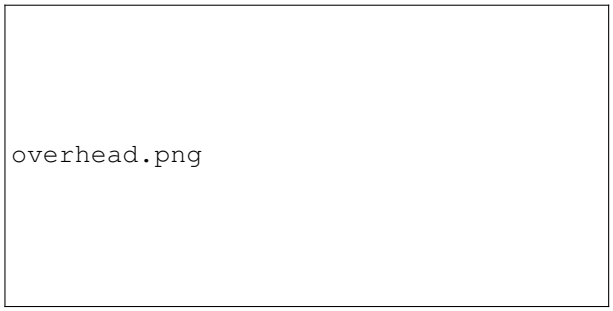

Figure 2. What is performance overhead of thread library over pthreads?

5. Performance of parallel programs under CPU contention

- speed-up curve
- slow-down curve

5.1 Common strategies for dealing with unknown resources

- Overprovision number of threads and hope for the best. But this can cause problems (e.g., lock or resource contention)

6. Design

6.1 Thread library

Similar to Intel Thread Building Blocks, but dynamically scale number of workers across applications, rather than just across workers

- Types of blocks: iterative / non-iterative parallelized operations
- DAG
- Unstructured

6.2 Driver application

Track CPU utilization by application (any other metrics?) Change target parallelism (how to decide, when to decide, how much to change by, which to change) Change CPU share or quota

6.3 Client applications

7. PARSEC and Splash2 Benchmarks

8. Imeplementation

9. Evaluation

9.1 Workloads

Different classes of applications. Mixture of applications.

9.2 Comparison with other thread libraries

OS (pthreads), OpenMP, Intel TBB

9.3 Varying guarantees (CPU share / quota)

9.4 Various machines

10. Related work

- Intel TBB, OpenMP, Cilk (http://parsec.cs.princeton.edu/publications/c
- Scheduler activations
- Multikernel
- Exokernel
- Hierarchical schedulers
- Cache-aware / contention-aware / numa-aware schedulers
- Arachne
- Capriccio
- Wasted cores

11. Experience and future work

Acknowledgements

References

[1] BAUMANN, A., BARHAM, P., DAGAND, P.-E., HARRIS, T., ISAACS, R., PETER, S., ROSCOE, T., SCHÜPBACH, A., AND SINGHANIA, A. The multikernel: a new os architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (2009), ACM, pp. 29–44.

[2] LOZI, J.-P., LEPERS, B., FUNSTON, J., GAUD, F., QUÉMA, V., AND FEDOROVA, A. The linux scheduler: a decade of wasted cores. In *Proceedings of the Eleventh European Conference on Computer Systems* (2016), ACM, p. 1.

[3] MACK, C. A. Fifty years of moore's law. *IEEE Transactions on semiconductor manufacturing 24*, 2 (2011), 202–207.

[4] OUSTERHOUT, K., WENDELL, P., ZAHARIA, M., AND STOICA, I. Sparrow: distributed, low latency scheduling. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (2013), ACM, pp. 69–84.