Brief Announcement: Parallel Depth First vs. Work Stealing Schedulers on CMP Architectures

Vasileios Liaskovitis*, Shimin Chen†, Phillip B. Gibbons†,
Anastassia Ailamaki*, Guy E. Blelloch*, Babak Falsafi*, Limor Fix†,
Nikos Hardavellas*, Michael Kozuch†, Todd C. Mowry*,†, Chris Wilkerson‡
*Carnegie Mellon University †Intel Research Pittsburgh ‡Intel Microprocessor Research Lab

Categories and Subject Descriptors: D.4.1 Operating Systems: Process Management—*threads, scheduling.*
General Terms: Algorithms, Measurement, Performance.
Keywords: Chip Multiprocessors, Scheduling, Caches.

1. ABSTRACT

In chip multiprocessors (CMPs), limiting the number of off-chip cache misses is crucial for good performance. Many multithreaded programs provide opportunities for *constructive cache sharing*, in which concurrently scheduled threads share a largely overlapping working set. In this brief announcement, we highlight our ongoing study [4] comparing the performance of two schedulers designed for fine-grained multithreaded programs: Parallel Depth First (PDF) [2], which is designed for constructive sharing, and Work Stealing (WS) [3], which takes a more traditional approach.

Overview of schedulers. In PDF, processing cores are allocated ready-to-execute program tasks such that higher scheduling priority is given to those tasks the sequential program would have executed earlier. As a result, PDF tends to co-schedule threads in a way that tracks the sequential execution. Hence, the aggregate working set is (provably) not much larger than the single thread working set [1]. In WS, each processing core maintains a local work queue of ready-to-execute threads. Whenever its local queue is empty, the core steals a thread from the bottom of the first non-empty queue it finds. WS is an attractive scheduling policy because when there is plenty of parallelism, stealing is quite rare. However, WS is not designed for constructive cache sharing, because the cores tend to have disjoint working sets.

CMP configurations studied. We evaluated the performance of PDF and WS across a range of simulated CMP configurations. We focused on designs that have fixed-size private L1 caches and a shared L2 cache on chip. For a fixed die size (240 mm^2), we varied the number of cores from 1 to 32. For a given number of cores, we used a (default) configuration based on current CMPs and realistic projections of future CMPs, as process technologies decrease from 90nm to 32nm.

Summary of findings. We studied a variety of benchmark programs to show the following findings.

For several application classes, PDF enables significant constructive sharing between threads, leading to better utilization of the on-chip caches and reducing off-chip traffic compared to WS. In particular, bandwidth-limited irregular programs and parallel divide-and-conquer programs present a relative speedup of 1.3–1.6X over WS, observing a 13–41% reduction in off-chip traffic. An example is shown in Figure 1, for parallel merge sort. For each schedule, the number of L2 misses (i.e., the off-chip traffic) is shown on the left and the speed-up over running on one core is shown on the right, for 1 to 32 cores. Note that reducing the off-chip traffic has the additional benefit of reducing the power consumption. Moreover, PDF's smaller working sets provide opportunities to power down segments of the cache without increasing the running time. Furthermore, when multiple programs are active concurrently, the PDF version is also less of a cache hog and its smaller working set is more likely to remain in the cache across context switches.

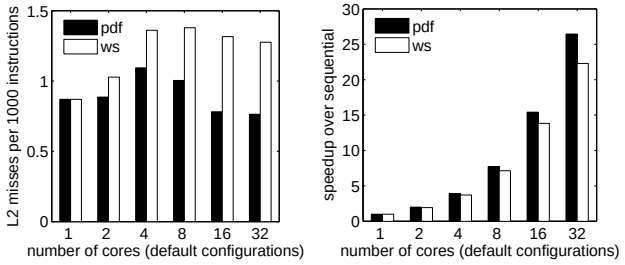

Figure 1: PDF vs. WS for parallel merge sort

For several other applications classes, PDF and WS have roughly the same execution times, either because there is only limited data reuse that can be exploited or because the programs are not limited by off-chip bandwidth. In the latter case, the constructive sharing PDF enables does provide the power and multiprogramming benefits discussed above.

Finally, most parallel benchmarks to date, written for SMPs, use such a coarse-grained threading that they cannot exploit the constructive cache behavior inherent in PDF. We find that mechanisms to finely grain multithreaded applications are crucial to achieving good performance on CMPs.

2. REFERENCES

[1] G. E. Blelloch and P. B. Gibbons. Effectively sharing a cache among threads. In *Proc. ACM SPAA*, 2004.
[2] G. E. Blelloch, P. B. Gibbons, and Y. Matias. Provably efficient scheduling for languages with fine-grained parallelism. *JACM*, 46(2), 1999.
[3] R. D. Blumofe and C. E. Leiserson. Scheduling multithreaded computations by work stealing. *JACM*, 46(5), 1999.
[4] V. Liaskovitis, S. Chen, P. B. Gibbons, A. Ailamaki, G. E. Blelloch, B. Falsafi, L. Fix, N. Hardavellas, M. Kozuch, T. C. Mowry, and C. Wilkerson. Scheduling threads for constructive cache sharing on CMPs. Intel Research Pittsburgh tech. rep., June 2006.

SPAA'06, July 30–August 2, 2006, Cambridge, Massachusetts, USA
ACM 1-59593-262-3/06/0007.