LightSpeed: A Many-core Scheduling Algorithm

Karl Naden
kbn@cs.cmu.edu

Wolfgang Richter
wolf@cs.cmu.edu

Ekaterina Taralova
etaralova@cs.cmu.edu

September 27, 2010

1 Introduction

The world is heading towards many-core architectures due to many well-known and important present-day research issues: power consumption, clock speed limits, critical path lengths, etc. While existing many-core machines have traditionally been handled in the same way as SMPs, this magnitude of parallelism introduces several fundamental challenges at the architectural level which translates to novel challenges in the design of the software stack for these platforms.

Many-core architectures cause new problems: shared caches, shared memory controllers, shared communication paths—communication between computation now—and other shared resource contentions.

Current many-core scheduling research literature and operating system development have not provided a satisfactory solution to the scheduling problem. Partly because the hardware is not yet widespread—no need to implement or solve a non-existant problem—and partly because many of the issues are hard to solve—especially in the general case.

2 Problem Statement

Many-core scheduling presents a non-trivial challenge to modern schedulers within operating systems which none have solved satisfactorily. Given a set of threads the goal of a scheduler is to minimize the *makespan*—the time it takes to complete all threads. Figure 1 shows current experimental results from cutting edge research literature using several simple scheduling algorithms. Embarrasingly parallel workloads imply linear speedups—linear reduction in *makespan*—when increasing the number of computational units. However, the research literature shows that achieving linear speedup with real-world architectures is hard, and achieving optimal linear speedup is NP-Complete [1].

For example the only algorithm discussed in [2] that gives a linear speedup is the `nosteal` algorithm—cores have separate queues and are not allowed to steal work from other cores. `nosteal` is the simplest algorithm in the scenario of a single queue per core, yet it has a clear Achille's heel—the slowest core defines the potential speedup. The other two algorithms, `3steal`—allow stealing from the 3 closest cores' queues—and `steal`—allow stealing from any core's queue—show a sublinear dropoff in speedup.

We intend to rigorously survey the research literature on many-core scheduling, explore alternative scheduling algorithms and heuristics, and improve upon previous work focusing on embarrasingly parallel workloads on homogeneous many-core systems. Figure 1 shows a clear gap where other unstudied algorithms could exist—where LightSpeed should exist. In fact, the authors of [2] have already attempted to improve upon their existing schedulers in [3].

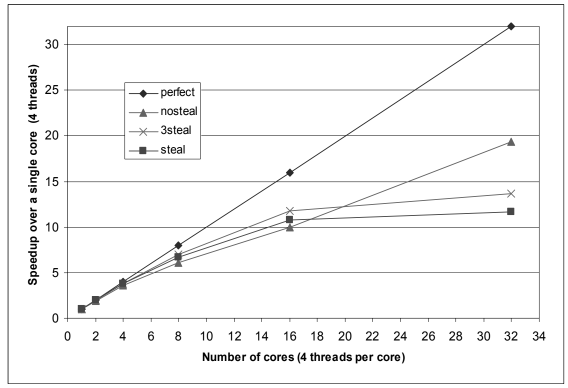

Figure 1: XviD speedup varying number of cores. From: [2]

3 Previous Work

Algorithms for scheduling threads in a many-core setting have been proposed ranging from the simplistic, though in some cases effective, algorithms defined in Saha et al's [2] Many Core Run Time (McRT), to the arbitrarily complex search algorithms surveyed in [1] or the Heterogeneity-Aware Signature-Supported (HASS) algorithm [4]. We are aware of one PhD dissertation on the topic of scheduling on chip multithreaded processors [5], and this seminar paper [6] provides a very gentle introduction to the problem and the research literature.

Algorithm design also hinges on whether or not the assumed underlying architecture is homogeneous or heterogeneous—further complicating optimal schedule assignment. Several recent attempts have been made to create scheduling algorithms in the heterogeneous case such as HASS [4], or the Asymmetric Multiprocessor Scheduler (AMPS) [7]. Other proposed algorithms target generality or fairness such as the fair Distributed Weighted Round-Robin (DWRR) [8], and thread criticality for performance, power and resource management [9]. There have even been attempts at adapting MapReduce [10]—a potential workload for LightSpeed—to the many-core setting [11, 12]. These algorithms provide insight into the techniques and heuristics upon which LightSpeed must extend.

Other domains of computer science research, such as networking, benefit from many-core scheduling research as is evidenced by the attempt of Mallik et al. [13] to create a scheduling algorithm using statistical analysis in network processors. Their use of statistical techniques encourages a line of research that could lead to using machine learning within scheduling algorithms for many-core architectures. Our work may explore this potential synergy between machine learning and thread schedulers in the many-core setting.

4 Algorithm Exploration, Experimental Methods and Plan

We will implement thread scheduling algorithms explored in prior works on a simulator for many-core systems in a MapReduce-like application for highly parallelizable jobs. We will evaluate weaknesses and strengths of these algorithms and propose modifications to increase performance when the number of cores in the system increases.

We will explore scheduling algorithms like the ones used in the McRT system [2], which is a software prototype of an integrated language runtime that was designed to explore configurations of the software stack for enabling performance and scalability on large scale many-core platforms.

We will use a simulator for the Intel Single-chip Cloud Computer, (SCC), or similar. The SCC is a research microprocessor containing the most Intel Architecture cores ever integrated on a silicon CPU chip: 48 cores. It incorporates technologies intended to scale multi-core processors to 100 cores and beyond, such as an on-chip network, advanced power management technologies and support for message-passing.

The potential workloads and benchmarks, in addition to the standard ones used in prior work [10] and [14], will include using existing implementations of MapReduce for various applications that lend themselves to high parallelization.

By the Milestone 1 deadline, October 13th, we will:

- Explore simulation tools. Ideally, we will try to obtain a simulator for the Intel SCC 48-core research architecture [15], or similar (for example the simulator of [16] for multicore simulation). We will have a simulator chosen and running.
- Research existing implementations of libraries for MapReduce-like [10] functionality. For instance, Phoenix [11, 12], developed at Stanford, is a potentially suitable implementation of MapReduce for shared-memory systems.
- Review literature on scheduling jobs in many-core systems - Saha et al. [2], Jin et al. [1], Mallik et al. [13], Rajagopalan et al. [3], and other research that we come across.

By the Milestone 2 deadline, November 1st, we will:

- Reproduce baseline cases or prior results
- Evaluate performance of existing scheduling algorithms
- Determine weaknesses in existing algorithms, propose and implement improvements
- Provide results for some of the improved algorithms and continue evaluation of results

References

[1] S. Jin, G. Schiavone, and D. Turgut, "A performance study of multiprocessor task scheduling algorithms," *The Journal of Supercomputing*, vol. 43, pp. 77–97, 2008, 10.1007/s11227-007-0139-z. [Online]. Available: http://dx.doi.org/10.1007/s11227-007-0139-z

[2] B. Saha, A.-R. Adl-Tabatabai, A. Ghuloum, M. Rajagopalan, R. L. Hudson, L. Petersen, V. Menon, B. Murphy, T. Shpeisman, E. Sprangle, A. Rohillah, D. Carmean, and J. Fang, "Enabling scalability and performance in a large scale cmp environment," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 73–86, 2007. [Online]. Available: http://doi.acm.org/10.1145/1272998.1273006

[3] M. Rajagopalan, B. T. Lewis, and T. A. Anderson, "Thread scheduling for multi-core platforms," in *HOTOS'07: Proceedings of the 11th USENIX workshop on Hot topics in operating systems*. Berkeley, CA, USA: USENIX Association, 2007, pp. 1–6.

[4] D. Shelepov, S. Alcaide, J. Carlos, S. Jeffery, A. Fedorova, N. Perez, Z. F. Huang, S. Blagodurov, and V. Kumar, "Hass: a scheduler for heterogeneous multicore systems," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 2, pp. 66–75, 2009. [Online]. Available: http://doi.acm.org/10.1145/1531793.1531804

[5] A. Fedorova, "Operating system scheduling for chip multithreaded processors," Ph.D. dissertation, Cambridge, MA, USA, 2006, adviser-Margo I. Seltzer.

[6] T. Fahringer, "Optimisation: Operating system scheduling on multi-core architectures," Website, August 2008, http://www.scribd.com/doc/4838281/Operating-System-Scheduling-on-multicore-architectures.

[7] T. Li, D. Baumberger, D. A. Koufaty, and S. Hahn, "Efficient operating system scheduling for performance-asymmetric multi-core architectures," in *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*. New York, NY, USA: ACM, 2007, pp. 1–11. [Online]. Available: http://doi.acm.org/10.1145/1362622.1362694

[8] T. Li, D. Baumberger, and S. Hahn, "Efficient and scalable multiprocessor fair scheduling using distributed weighted round-robin," *SIGPLAN Not.*, vol. 44, no. 4, pp. 65–74, 2009. [Online]. Available: http://doi.acm.org/10.1145/1594835.1504188

[9] A. Bhattacharjee and M. Martonosi, "Thread criticality predictors for dynamic performance, power, and resource management in chip multiprocessors," in *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*. New York, NY, USA: ACM, 2009, pp. 290–301. [Online]. Available: http://doi.acm.org/10.1145/1555754.1555792

[10] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[11] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, "Evaluating mapreduce for multi-core and multiprocessor systems," in *HPCA '07: Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 13–24. [Online]. Available: http://dx.doi.org/10.1109/HPCA.2007.346181

[12] R. Yoo, A. Romano, and C. Kozyrakis, "Phoenix rebirth: Scalable mapreduce on a large-scale shared-memory system," October 2009, pp. 198 –207.

[13] A. Mallik, Y. Zhang, and G. Memik, "Automated task distribution in multicore network processors using statistical analysis," in *ANCS '07: Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*. New York, NY, USA: ACM, 2007, pp. 67–76. [Online]. Available: http://doi.acm.org/10.1145/1323548.1323563

[14] L. Huston, R. Sukthankar, R. Wickremesinghe, M. Satyanarayanan, G. R. Ganger, E. Riedel, and A. Ailamaki, "Diamond: A storage architecture for early discard in interactive search," in *FAST '04: Proceedings of the 3rd USENIX Conference on File and Storage Technologies*. Berkeley, CA, USA: USENIX Association, 2004, pp. 73–86.

[15] Intel Corporation, "The scc platform overview," Website, September 2010, http://communities.intel.com/servlet/JiveServlet/downloadBody/5512-102-1-8627/SCC_Platform_Overview_90110.pdf.

[16] H. Zeng, M. Yourst, K. Ghose, and D. Ponomarev, "Mptlsim: a cycle-accurate, full-system simulator for x86-64 multicore architectures with coherent caches," *SIGARCH Comput. Archit. News*, vol. 37, no. 2, pp. 2–9, 2009. [Online]. Available: http://doi.acm.org/10.1145/1577129.1577132