On Hiding Multicore Complexity from System Software

Philip M. Wells, Koushik Chakraborty and Gurindar S. Sohi
Computer Sciences Department
University of Wisconsin-Madison
{pwells, kchak, sohi}@cs.wisc.edu

Abstract

Future multicores will be very complex: at the very least, they may contain statically heterogeneous cores*, which are designed with different engineering trade-offs, and* dynamically heterogeneous cores*, which have different, and rapidly changing, execution characteristics. Hardware companies traditionally expose chips to system software at a very low level, effectively saying, "Here is what we built, now do something with it."*

However, there are several advantages to having the chip itself manage these emerging complexities, while exposing a more generic interface to software. We do not have all of the answers for the appropriate role of system software, but we do suggest that system architects should carefully consider the benefits of abstraction when designing future systems.

1 Statically and Dynamically Heterogeneous

Several proposals have exalted the benefits of systems with statically heterogeneous cores — cores which are designed to have different physical characteristics in order to capitalize on different engineering trade-offs (e.g., [4, 6]).

Similarly, multicore chips will contain *dynamically heterogeneous* cores as well — cores which observe different, and rapidly changing, execution characteristics, even though they may be physically homogeneous. These differing characteristics may arise from power and thermal management, intermittent faults, different predictive state, among others sources, and include not only changes in the characteristics of an individual core, but also changes in the *number* of available cores.

2 Opportunities for Innovation

All of this heterogeneity adds complexity to the organization and use of future multicores, but also provides many opportunities for innovative use of these resources. Several innovative uses of dynamic heterogeneity are discussed below, which improve upon a variety of chip design goals.

Instruction Locality Chakraborty, et al., proposed the idea of *Computation Spreading*, and have shown that intelligent, and frequent, thread migration in commercial workloads can improve instruction cache and branch predictor locality without disturbing data caches [2]. Computation spreading not only takes advantage of the dynamic heterogeneous capabilities of different cores, but actively *creates* this heterogeneity through the assignment of computation fragments to different cores, and can lead to a significant reduction in cache and branch predictor miss rates.

Power/Thermal In follow-on work, Chakraborty, et al., have shown that future chips will contain more cores than can actively execute instructions at any given time, due to power and thermal constraints [3]. Yet by leveraging techniques such as Computation Spreading, it is possible to efficiently use all of the cores, at different times, to improve thermal characteristics, energy, and delay.

Reliability Intermittent faults, caused in part by manufacturing process variation or in-progress wear-out, can cause bursts of frequent faults that last from several cycles to several seconds or more. Wells, et al., argue that cost-effective reliability techniques to tolerate these faults will likely require, or be greatly simplified by, the ability to temporarily suspend execution on a core during periods of intermittent faults [9].

Joining Cores Techniques such as *Core Fusion* [5], which dynamic couple multiple cores into one logical processor can allow the same chip to be effective for single or multithreaded applications, as well as mitigate the effects of Amdahl's Law for semi-parallel applications.

3 The Multicore Interface Status Quo

These, and many other innovative uses for statically or dynamically heterogeneous chips, have a variety of features that make them attractive. Yet, these uses raise several issues about whether the status quo — exposing the details of the chip and requiring system software to manage its use — will remain appropriate for future multicores.

Rapidly Changing Conditions The cost of migrating execution state among processing cores may be greatly improved by multicore chips, but the overhead of trapping into the OS to make scheduling decisions is not. If the characteristics and capabilities of the chip changes frequently enough, the overhead of both monitoring and adapting at the system software level may be excessively high. As one concrete example, Wells, et al., show that expecting the OS to adapt to dynamic changes in the number of available cores can greatly impact the performance of the entire system [8, 9]. At the same time, the latency of OS adaptation may actually take longer than the *need* for the adaptation.

Compatibility & Innovation Software (including system software) often has a longer lifetime than hardware. This means that hardware companies must maintain backward compatibility with older software, support system software from multiple vendors, and at the same time, add value (e.g. performance or reliability innovations) to their products to entice users to upgrade.

Meanwhile, system software vendors must support multiple chips from multiple vendors, but are not always willing to implement an efficient resource management policy to support hardware innovations from only one company.

Abstracting Details Conceptually, abstracting the details of a hardware implementation is attractive because it fits well into the layered model of computer systems. But abstraction has practical advantages as well.

First, hardware companies may not wish to disclose all of the hardware details, including the occurrence of faults, or certain microarchitectural innovations, for reasons of competitive advantage. Second, hardware companies may not wish to abdicate responsibility for the proper workings of their chip (e.g., thermal management, reliability, etc.) to a third party system software vendor.

4 The Solution?

Given the preceding opportunities and issues, we argue that future heterogeneous multicores may do well to expose a simpler, traditional, homogeneous multiprocessor interface to the system software, and then manage the use of hardware innovations with hardware/firmware layers beneath the ISA. This abstraction is directly analogous to out-of-order processors exposing a sequential execution model to the software, and then performing a variety of optimizations under the hood.

This model provides many opportunities for the innovative uses of future heterogeneous multicores, while at the same time allows the management of these innovations to quickly adapt to changing resources, remain compatible with existing and future software, and abstract the details from third party vendors, users, and competitors.

We acknowledge that several challenges arise in such a scenario, especially when it is necessary to perform these optimizations with no assistance from system software. Yet we have no reason to believe that the challenges are insurmountable, or even particularly difficult. It has already been shown that simple hardware/firmware techniques can intelligently migrate execution state among cores with low overhead and without involving the OS, and that high performance operation can be maintained even when the chip has both more or fewer physically available cores than are visible to the OS [2, 3, 7–9]. Additional challenges will arise as we continue to explore this model, but in all likelihood, they can be dealt with similarly: by inferring information from the higher layers in the system (i.e., using a gray-box approach [1]).

While we certainly do not have all of the answers about how this should be done, we do hope that system architects will carefully consider the benefits of abstraction when designing future systems.

References

[1] A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau. Information and Control in Gray-Box Systems. In *Proc. of 18th SOSP*, 2001.

[2] K. Chakraborty, P. M. Wells, and G. S. Sohi. Computation spreading: Employing hardware migration to specialize CMP cores on-the-fly. In *Proc. of 12th ASPLOS*, 2006.

[3] K. Chakraborty, P. M. Wells, and G. S. Sohi. A case for an over-provisioned multicore system: Energy efficient processing of multithreaded programs. Technical Report CS-TR-2007-1607, University of Wisconsin-Madison, Aug 2007.

[4] M. Gschwind, P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki. A novel simd architecture for the Cell heterogeneous chip-multiprocessor. In *Hot Chips 17*, 2005.

[5] E. Ípek, M. Kirman, N. Kirman, and J. F. Martínez. Core fusion: accommodating software diversity in chip multiprocessors. In *Proc. of 34th ISCA*, 2007.

[6] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *Proc. of 36th MICRO*, 2003.

[7] P. M. Wells, K. Chakraborty, and G. S. Sohi. Hardware support for spin management in overcommitted virtual machines. In *Proc. of 15th PACT*, 2006.

[8] P. M. Wells, K. Chakraborty, and G. S. Sohi. Adapting to intermittent faults in future multicore systems. In *Proc. of 16th PACT (Poster)*, 2007.

[9] P. M. Wells, K. Chakraborty, and G. S. Sohi. Adapting to intermittent faults in multicore systems. Technical Report CS-TR-2007-1605, University of Wisconsin-Madison, Aug 2007.