Impact of CMP Design on High-Performance Embedded Computing

Patrick Crowley
Mark A. Franklin
Jeremy Buhler
Roger D. Chamberlain

Patrick Crowley, Mark A. Franklin, Jeremy Buhler, and Roger D. Chamberlain, "Impact of CMP Design on High-Performance Embedded Computing," in *Proc. of 10th High Performance Embedded Computing Workshop*, September 2006, pp. 33-34.

Dept. of Computer Science and Engineering
Washington University
Campus Box 1045
One Brookings Dr.
St. Louis, MO 63130-4899

Impact of CMP Design on High-Performance Embedded Computing

Patrick Crowley, Mark A. Franklin, Jeremy Buhler, and Roger D. Chamberlain
Department of Computer Science and Engineering
Washington University in St. Louis
pcrowley@wustl.edu, jbf@wustl.edu, jbuhler@wustl.edu, roger@wustl.edu

Introduction

Chip multiprocessors (CMPs) will dominate commercial processor designs for at least the next decade, during which we will likely see an annual doubling of the number of processor cores integrated onto a single chip. This trend is expected in both desktop and server processors, as well as within high-performance embedded systems where, in fact, the phenomenon first appeared. Both Intel's IXP network processors (16 cores) and the MPOC project (8 cores) from Hewlett-Packard were early embedded CMP designs for use in routers and printers, respectively. The force behind this trend is technology-based: it is possible to efficiently scale a CMP design by increasing the number of cores while maintaining or reducing overall power consumption by reducing clock frequency. As long as the increase in cores offsets the reduction in clock frequency, peak system performance will improve; or, conversely, a given level of performance can be maintained with less power.

Of course, achieving peak system performance depends on how well a target application can be mapped onto system resources. Notably, CMP architectures provide thread-level parallelism, thus suggesting that performance-sensitive applications will need to be thread-parallel. In the embedded systems context, one can consider tailoring the application, the processor, or both in order to meet system constraints. While this enlarged design landscape provides great flexibility, it also creates design challenges.

Thus, the use of CMPs in embedded systems has strong implications for high-performance embedded computing. In this abstract, we highlight some of these implications with several examples drawn from our current research program exploring the design and use of CMPs. In particular, we illustrate the following topics.

1. **CMP Organization**. Specifically, how the use of multiple processor core types within a CMP impacts system efficiency and complexity.
2. **Achieving Performance/Area Efficiency**. We focus on a novel CMP instruction delivery hierarchy built with μ -caches.
3. **Promising Thread-Parallel Applications**. We illustrate how an embedded CMP excels at implementing a Hidden Markov Model-based bioinformatics application.

This work supported by NSF grants CCF-0430012, CCF-0427794, and DBI–0237902 and NIH/NGHRI grant 1 R42 HG003225–01.

CMP Organization: Evaluating Heterogeneity

While replicating cores is an efficient strategy, architects are confronted with a basic question: what type of core should be replicated? A given die area can accommodate: many small, simple cores; fewer cores of a larger more sophisticated variety; or some combination of the two. Thus, in CMPs it is common to see either many simple processors or a moderate quantity of high performance cores. The first solution meets the needs of computing environments characterized by higher thread parallelism, while the second better accommodates scenarios with lower thread parallelism and higher individual thread complexity.

Heterogeneous CMP systems, consisting of a combination of processor cores of varying type on the same chip, represent a compromise between the above alternatives. The motivation comes from the observation that a multi-programmed computing environment may present threads of execution with different hardware resource requirements, and that such needs may vary over time. In embedded systems, such as Intel's IXP, heterogeneous cores are the dominant model. With a heterogeneous CMP, an appropriate mapping of different program threads to heterogeneous processor cores can maximize resource utilization and, at the same time, achieve a high degree of inter-thread parallelism.

In order to take advantage of a heterogeneous architecture, an appropriate policy to map running tasks to processor cores must be determined. The overall goal of such a strategy must be to maximize the performance of the whole system by accurately exploiting its resources. The control mechanism must take into account the heterogeneity of the system and of the workload, and the varying behavior of the threads over time. Moreover, it must be easily implementable and introduce as little overhead as possible.

In a recent study [1], we show that a heterogeneous system adopting a dynamic assignment policy is able to accommodate a variety of degrees of thread parallelism more efficiently than both a homogeneous and a heterogeneous system adopting a static assignment policy. Our results, based on a collection of 11 sample programs, show that heterogeneity can provide a 20%-80% improvement in system performance compared to a comparable homogeneous CMP, and that dynamic thread migration can achieve a further 20%-40% gain.

Design Goals: Performance/Area Efficiency

CMP architectures place a relatively novel emphasis on the area efficiency of the replicated processor cores; in particular on the performance/area efficiency of the cache

hierarchy. In single-processor designs, cache hierarchies have traditionally focused on computational performance, with relatively little emphasis placed on area efficiency.

In recent work [2], we explore the use of very small instruction caches, called μ-caches, which range in size from 64 to 256 bytes. In a CMP system with L1 instruction μ-caches, processors are arranged in clusters that share an on-chip L2 instruction cache. This shared cache is configured much like a traditional L1 instruction cache. Provided that the use of μ-caches does not reduce performance greatly, a substantial area savings can be achieved due to the reduced number of L1 I-caches. If, for example, the traditional I-cache accounts for a third of total core area—the other components being the processor core proper and the data cache (D-cache)—then, in the best case, effectively removing the I-cache from each core will allow a cluster of 4 cores with μ-caches to occupy roughly the same area as a traditional 3-core cluster.

To evaluate the effectiveness of μ-caches, we performed a simulation-based experimental study that compares the performance/area efficiency of μ-caches to that of traditional instruction cache hierarchies. Our study has two crucial characteristics. First, rather than trying to demonstrate that μ-caches are effective in the general case, we restrict our study to applications that are commonly or easily deployed on clustered CMP processors and consist of pipelines of processing kernels, namely networking/communications and bioinformatics. These kernels are used to construct workloads that correspond to the three natural approaches to mapping programs to multiple cores. Second, we obtain performance and area estimates by the use of a commercial design environment.

For these workloads, a cluster with μ-caches can maintain the same performance with 25% of the area, or provide 25% greater performance in the same amount of area.

Thread-Parallel Application in Bioinformatics

Certain classes of compute-intensive, thread-parallel applications are well-suited to CMP-based systems. To explore how new levels of performance may be achieved, we recently implemented and described [4] a CMP-based version of a scientific workload drawn from bioinformatics: the HMMer program [3] for protein motif finding. HMMer compares protein sequences to a database of motifs – sequences known to occur, with some variation, in a large family of other proteins. These motifs are represented as hidden Markov models (HMMs), which allows HMMer to search for them in a protein using well-developed mathematical machinery for parsing discrete sequences with an HMM. Because HMMer works on a large database of motifs, each of which can be compared separately to a target protein, its computation can benefit greatly from systems with substantial coarse-grained parallelism. In our work, we show that this computation is therefore a natural fit to CMPs, specifically Intel's IXP processor. While HMMer is not itself an embedded application, its mathematical structure is similar HMM-based embedded applications, such as speech processing and pattern matching, thus it is amenable to an embedded implementation.

The primary artifact of this work is JackHMMer, a version of HMMer that runs on an Intel IXP 2850 network processor. The IXP implements the Viterbi algorithm for HMMs, which is the core component of HMMer's search algorithm. JackHMMer achieves a 2x-5x performance gain relative to a hand-optimized HMMer version on a hyper-threaded Pentium 4.

New Tools and Design Approaches

In addition to influencing the design and use of embedded systems, the trend toward CMP-based design also influences a designer's tool-suite. In the three projects described here, three different development and simulation environments were employed. The heterogeneity study relies on the M5 multiprocessor system simulator from Michigan. The μ-cache simulation environment was built using Tensilica's Xtensa design tools. The Xtensa environment allows us to construct cycle-accurate system simulations with multiple processor cores, as well as sophisticated on- and off-chip memory components and interconnects, while obtaining die area and clock frequency estimates. Notably, the Xtensa environment was used to design Cisco's Metro NP. Finally, JackHMMer was developed with Intel's IXP development and simulation environment, prior to deployment in a real system.

Moreover, each study involved the creation of new, parallel workloads. We expect to see aggressive increases in the numbers of 1) thread-parallel workloads available, and 2) the design tools needed to support their development. A major challenge lies ahead as programmer tools and productivity confront thread-parallel development.

Conclusions

In summary, a synthesis of our recent work in CMP design and applications illustrates how the multi-core trend impacts high-performance embedded computing. The central observation is that the presence of multiple cores within a single embedded processor drives new directions and opportunities in system and processor design, design objectives, and target applications.

References

[1] M. Becchi and P. Crowley. "Dynamic Thread Assignment on Heterogeneous Multiprocessor Architectures." In *Proceedings of the 3rd ACM Int'l Conference on Computing Frontiers*, Ischia, Italy. May, 2006.

[2] M. Becchi, M. A. Franklin, and P. Crowley. "Performance/area efficiency in CMP processors with micro-caches." Under review. May, 2006.

[3] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*, Cambridge University Press, 1998.

[4] B. Wun, J. Buhler, and P. Crowley. "Exploiting Coarse-Grained Parallelism to Accelerate Protein Motif Finding with a Network Processor." In *Proceedings of the 2005 International Conference on Parallel Architectures and Compilation Techniques (PACT)*. Saint Louis, MO. September, 2005.