White Paper

Preparing for the Second Stage of Multi-Core Hardware: Asymmetric (Heterogeneous) Cores

By Matt Gillespie

Abstract

The current generations of multi-core processors have successfully improved upon their single-core predecessors, in terms of performance, power consumption, and thermal properties. Hardware parallelism (in the form of the number of cores per processor) has taken the place of clock speed as the specification of first interest when describing a processor.

Asymmetric multi-core computing (where various cores have different performance, architectural, and functional characteristics) is a likely candidate to be the next set of changes on this order of magnitude. This transition will extend the capability of multi-core processors to build performance and efficiency, giving rise to a significant set of software challenges and opportunities.

Overview

Even as increasing numbers of execution cores per processor have become the most visible means of increasing hardware performance[1], opportunities are beginning to become apparent for a variation on that theme. Whereas current mainstream designs are based on multiple, identical cores per processor, it seems clear that upcoming designs will incorporate different types of cores in a single processor package.

That approach is commonly referred to as *asymmetric* or *heterogeneous* multi-core architecture. While those two terms are commonly used synonymously, *asymmetric* tends to be favored more in discussions of software (since it implies the manner in which the hardware architecture is exposed to software), whereas *heterogeneous* tends to be favored more in discussions of hardware (since it more directly describes the physical architecture itself). For the sake of simplicity, this paper uses the term "asymmetric" throughout.

This paper lays out some of the background for why this trend is likely to emerge, likely variations on the hardware architecture, and the distinct challenges and opportunities this change presents from a software-development perspective.

Benefits of Asymmetric Multi-Core Architecture

One of the most significant challenges in the recent history of the microprocessor industry was the rise of inherent limitations to the ability to increase performance by driving up clock speed. Faster processor frequencies began to deliver relatively modest performance increases, due to memory-access limitations and other issues, while at the same time, power consumption and heat dissipation rose dramatically.

Together with other microarchitectural advances in power-efficient performance, multi-core processing addressed those issues. Multiple processing cores now run in parallel at relatively low clock speeds to provide high performance at low power, with favorable thermal characteristics. Looking ahead to the future of symmetric multi-core processors, however, one can see the basis for a new trend of diminishing returns [4], in the sense that increases in performance cannot ultimately keep pace with the increases in the speed and number of transistors per processor core.

Consider that, as process technology scales downward (from 65nm to 45nm, for example, with the Nehalem family of Intel® Xeon® processors), transistors become faster, and more of them can fit per unit of surface area. On the other hand, the performance of some microarchitectural structures such as cache increases at a less-than-linear rate as the components become smaller [4]. Therefore, on-die crowding will increasingly become a limiting factor to processor performance.

Building arrangements of non-uniform cores with different functional specialties and capability levels can potentially allow more performance to be packed into a given amount of space. To see

---

[1] At the same time processors continue to add more cores per processor, other architectural innovations also continue to add performance as well, such as improved pipelining, larger caches, support for new instructions, the integration of formerly software-based functionality onto the chip, etc.

why this is so, consider the case where a few large cores with high serial performance are mixed in the same processor package with many smaller, lower-performance cores.

The larger cores in this example would be well-suited to large serial tasks, while the large number of smaller cores would enable high performance for highly parallel tasks [1]. The combination of sizes would allow a high overall average core count per unit area of silicon (relative to a homogenous population, and the ability to tailor a specific combination of cores for a particular workload would use that spatial efficiency to generate performance benefits.

In scheduling work to hardware resources, moving away from a 'one size fits all' requirement is advantageous, since some workloads will get more benefit from higher-performance cores than others. For example, a compute-intensive, execution-bound workload might be best suited to the most robust processing core available, whereas an I/O-bound or memory-bound workload might not suffer dramatic slowdown if it is run on a lower-performance core.

In such cases, performance as a function of the proportion of overall processor resources consumed would be higher when the less-demanding workload is assigned to the lower-performance core. Thus, the processor as a whole can deliver more execution throughput. Moreover, since execution resources are better tailored to the requirements of the work they are performing, overall energy efficiency will also increase.

Software-Facing Characteristics of Asymmetric Hardware

For the sake of clarity, the discussion above simplifies the potential differences between the various cores within the processor package, characterizing them as large and powerful, versus small and modest. That sort of performance asymmetry might be underpinned by a number of hardware characteristics, ranging from clock speed and cache size to microarchitectural characteristics that enable more instructions to be performed per clock cycle.

For the purposes of this discussion, those differences can be characterized by the fact that they are manifested to software strictly in terms of performance differences. In other words, the software itself is concerned with them only in terms of their ability to complete a given amount of work more quickly or less so. That software does not need to consider any added complexity, in terms of whether or not a specific core can handle a certain type of functionality.

Functional asymmetry among processor cores is also possible, where cores differ in terms of instruction set architecture (ISA). Different types of general-purpose cores may be present with varying support for specific instruction sets; for example, it may be advantageous in terms of hardware complexity and size to have some cores that support SSE4 instructions alongside simpler cores that do not incorporate that support.

Other types of functional asymmetry could encompass any number of architectural differences, such as architectural registers, data types, addressing modes, memory architecture, exception and interrupt handling, and external I/O [7]. In contrast to the differences characterized here as performance asymmetry, software must handle these functional differences among cores appropriately in order to avoid runtime failures. Notably, these functional characteristics may overlap with one another, as in the case where a set of cores shares data types and memory architecture but only a subset of those cores supports a given instruction set [6].

In many respects, programming for functionally asymmetric multi-core processors builds on present work implementing hardware such as graphics processing units, field-programmable gate arrays (FPGAs), and application specific integrated circuits (ASICs) as co-processors. General-purpose programming for GPUs, in particular, has received a lot of attention as a means to potentially raise performance dramatically on floating-point-intensive applications.

One can easily imagine that a comprehensive programming approach might allow integrated use of a large variety of specialized processing cores, similarly to how an approach like NVIDIA's Compute Unified Device Architecture (CUDA) could be used to offload certain operations to specialized hardware. In such a case, the best hardware for a specific operation might even change according to conditions such as workload priority or power-management settings [3].

Scheduling Work in Asymmetric Systems

Because conventional operating systems do not account for asymmetric hardware resources, scheduling work onto such resources introduces unpredictability that can limit scalability [5]. Conversely, creating the means to mitigate that unpredictability is a potentially significant opportunity to software providers.

A number of issues contribute to the requirements of optimal work scheduling on asymmetric cores. For example, threads must be assigned to cores that can support the instructions they include and other computational characteristics. An efficient means of fault handling must be provided in the event that such support is not present. Asymmetric hardware may also be expected to compound the challenges that multi-threaded software encounters in terms of thread imbalance, since various types of execution resources can be assigned to a given thread, depending upon its requirements and priority.

In order to address these and related challenges, work at the Intel® Systems Technology Lab has developed AMPS, an asymmetric multiprocessor scheduler that improves performance on workloads executing on performance-asymmetric multi-core hardware [5]. In addition to performance improvements, AMPS also demonstrates consistent scheduling of threads onto appropriate resources with regard to their priority, so that results are predictable and repeatable. Notably, AMPS does not require changes to applications and only simple changes to the Linux* operating system, providing proof of concept toward lightweight support for asymmetric hardware.

AMPS balances loads among cores according to their individual performance characteristics, assigning work to processing resources in a manner that provides helps provide for high overall CPU utilization. The scheduler also employs a 'faster-core-first' prioritization schema that helps to ensure the utilization of the most capable available processor cores. For non-uniform memory access (NUMA) architectures, AMPS accommodates the added complexity associated, for example, with thread migration among disparate cores connected to different memory controllers. As asymmetric multi-core processors evolve, work on schedulers such as AMPS helps provide the basis for software to support them.

Some other research approaches to software support for asymmetric hardware focus at the application level. While such an approach necessarily places additional burden on the application programmer, its independence from the operating system may have value in closed-source environments, for example. Multiple Instruction Stream Processing (MISP) architecture exposes

processor cores programmatically as abstractions that can be managed by application software [2]. EXOCHI, a software architecture and programming model, extends MISP to include cores that are not based on x86 architecture, as well as providing a C/C++ programming environment designed to take advantage of it [9].

Software Experimentation with Asymmetric Multi-Core

It is possible using off-the-shelf hardware to emulate an asymmetric multi-core execution environment [6], potentially enabling software developers to create an experimental test bed for advance work in this area. For example, in a model where multiple asymmetric cores are present, feature discovery such as that using the CPUID instruction for Intel® architecture can identify the relevant capabilities of individual cores as a key component of the basis for assigning work to appropriate resources.

By various means, the hardware itself can be made to emulate an asymmetric set of cores. Disabling the floating-point unit in a subset of cores, for example, can cause floating-point instructions to generate device-not-present faults on those cores. Similarly, other capabilities can be blocked on certain cores so that they can be made to fault over to other cores.

Another approach is to use a multi-processor system with disparate processors in each socket, which can cause, for example, resources that have different instruction-set capabilities, core frequencies, cache sizes, etc. Modification of the operating system kernel can implement customized fault handling that migrates threads away from cores that lack (or have disabled) the features needed to execute them [6].

Experimentation with asymmetric architectures could also take advantage of the presence or absence of simultaneous multi-threading capabilities within processing cores. With the recent reintroduction of this capability into Intel Xeon and Itanium® processors, another aspect of scheduling among jobs arises. Threads that are scheduled together on a core with simultaneous multi-threading enabled interact and share processor resources very differently from threads scheduled on separate cores [4]. One important consideration is that some combinations of threads coexist more efficiently than others under simultaneous multi-threading [8], perhaps providing the basis for a performance opportunity not only in intelligently directing certain threads to multi-threaded cores but also in pairing them optimally.

Conclusion

Asymmetric processor hardware promises another revolution in performance, and with it, another set of challenges and opportunities for the software industry. While it may be some time before mainstream processors become available with heterogeneous execution resources, research in this area is becoming more advanced, and mainstream software developers can begin to prepare themselves with research and experimentation of their own.

Acknowledgements

The author wishes to thank Tong Li of Intel Labs for his input and help, without which this paper would not have been possible.

References

[1] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai. The impact of performance asymmetry in emerging multicore architectures. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture,* pages 506–517, June 2005.

[2] R. Hankins, G. Chinya, J. Collins, P. Wang, R. Rakvic, H. Wang, and J. Shen. Multiple Instruction Stream Processor. In *Proceedings of the 33rd International Symposium on Computer Architecture,* June 2006.

[3] 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 *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture,* pages 81–92, Dec. 2003.

[4] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In *Proceedings of the 31st Annual International Symposium on Computer Architecture,* pages 64–75, June 2004.

[5] Tong Li, Dan Baumberger, David A. Koufaty, and Scott Hahn. Efficient Operating System Scheduling for Performance-Asymmetric Multi-Core Architectures. In *Proceedings of Supercomputing 07,* November 2007.

[6] Tong Li, Paul Brett, Barbara Hohlt, Rob Knauerhase, Sean D. McElderry, and Scott Hahn. Operating System Support for Shared-ISA Asymmetric Multi-core Architectures. In *Proceedings of the Fourth Annual Workshop on the Interaction between Operating Systems and Computer Architecture (WIOSCA '08),* pages 19-26, June 2008.

[7] R. Ramanathan, R. Curry, S. Chennupaty, R. L. Cross, S. Kuo, and M. J. Buxton. Extending the world's most popular processor architecture. White Paper, Intel Corporation, 2007.

[8] A. Snavely and D. Tullsen. Symbiotic jobscheduling for a simultaneous multithreading architecture. In *Eighth International Conference on Architectural Support for Programming Languages and Operating Systems,* Nov. 2000.

[9] P. H. Wang, J. D. Collins, G. N. Chinya, H. Jiang, X. Tian, M. Girkar, N. Y. Yang, G.-Y. Lueh, and H. Wang. EXOCHI: Architecture and programming environment for a heterogeneous multi-core multithreaded system. In *Proceedings of the ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation,* pages 156–166, June 2007.

Additional Resources

The following materials provide a point of departure for further research on this topic:

- Intel® Multi-Core Technology and Research Portal provides access to a variety of resources about current multi-core technology at Intel, as well as ongoing innovation and research.
- Intel® Software Network Multi-Core Developer Community provides technical information, tools, conversation, and support from industry experts.
- *Multi-Core Programming: Increasing Performance through Software Multithreading*, a book from Intel Press, helps software developers write high-performance multi-threaded code.

About the Author

Matt Gillespie is an independent technical author and editor working out of the Chicago area and specializing in emerging hardware and software technologies. Before going into business for himself, Matt developed training for software developers at Intel Corporation and worked in Internet Technical Services at California Federal Bank. He spent his early years as a writer and editor in the fields of financial publishing and neuroscience.