Which OS? Considerations for Performance-asymmetric, Multi-core Platforms

By Ken Strandberg

Abstract

The long-standing assumption for developers writing for a parallel environment has been that all processors in a system deliver equal computational performance for their algorithms. This will change as more cores are packed onto a chip. And, it turns out that this performance asymmetry has a significant impact on application scalability and predictability under today's operating systems. This article looks at future performance-asymmetric processors and operating system challenges for performance-asymmetric, many-core processors.
Persistent evolution of processor architectures is critical to meet the continuous demands of increasing performance and performance per watt in computing. Ongoing research into performance-asymmetric, multi-core processor topologies reveals that mixing complex, fast cores and simpler, slower cores on the same die "enables fast parallel performance, fast serial performance, and lower cost than to produce all high-performance cores" on a die. [4]

Performance asymmetry, as opposed to functional asymmetry, is a topology where cores are based on the same instruction set architecture (ISA) but have different performance characteristics, such as clock speed, issue width, or in-order vs. out-of-order. Functional asymmetry describes a mixture of cores based on different ISAs, such as the IBM* Cell processors and Nvidia GPGPUs. In this topology, each processor requires its own explicit code. This article focuses on performance asymmetry.

For application and operating system development, programmers typically assume that a system contains computational elements that deliver equal performance. This assumption breaks down in performance-asymmetric systems. It turns out that this has a significant impact on application performance and stability.

Experiments of several workloads on a performance-asymmetric system revealed "performance asymmetry in systems adversely affects the predictability of a number of shared-memory workloads, and makes them less scalable. This effect increases with increasing concurrency." [1] In the future, the developer will no longer be able to depend on performance equivalence if code is destined for performance-asymmetric platforms.

The problem centers in scheduling threads and processes on appropriate cores and placing them where they can take optimal advantage of the system topology and architecture. Most general-purpose operating system (OS) schedulers favor placing threads on cores that are less loaded to facilitate load balancing, or to place them on cores that are cache-warm to exploit cache affinity. [4] They give no consideration to the performance of the core: one core is good as another. This exhibits in the following behaviors:

- **Repeatability** – without considering performance asymmetry, a scheduler dispatches a thread to a higher performance core in one run and to a lower performance core in another run, causing non-repeatable performance results [4].

- **Application vs. OS scheduling** – some applications, like DB2*, schedule tasks independent of the OS scheduler, or binds queries to particular processors, resulting in application instability [1].

- **Disregarding under or un-utilized fast cores** – without asymmetry awareness, a scheduler can place a task on a slower core when a faster one is available, degrading overall application performance.

Performance asymmetry negatively affects scheduling, whether the OS or application does it. Thus, awareness needs to be built into the OS, and developers should consider performance asymmetry when designing their applications. But how should an asymmetric-aware OS behave?
Asymmetry-aware Schedulers

According to Scott Hahn, a Principal Engineer in the Intel Corporate Technology Group, any OS supporting performance-asymmetric platforms has to be aware of the performance capabilities of all cores and the loads on them, and schedule each tasks in accordance with what each core can do. Research in [1] showed that by changing the way an asymmetric-aware scheduler assigned tasks it helped eliminate unpredictability in some applications. Applications that precluded the operating system's scheduler's actions required additional changes to account for performance asymmetry.

Scott was part of a research team that developed AMPS, an asymmetry-aware, multi-processor scheduler based on the Linux 2.6.16 kernel [4]. The team defined the requirements for AMPS to provide the following:

- Asymmetry-aware load balancing
- Faster-core-first scheduling
- NUMA-aware migration (to accommodate evolving processor architectures and larger system computing models)

Asymmetry-aware load balancing. Most modern multi-processor operating systems maintain one run queue per core, and they periodically balance the load on each core. AMPS extends this load balance approach by using a computing power value assigned to each core and maintaining a calculated ‘scaled’ load on the core proportional to its computing power. Thus, loads are shared fairly across the platform.

Faster-core-first scheduling. AMPs implements “faster-core-first” scheduling by calculating each core’s scaled load, assuming a new task is assigned to it, and choosing the core with the least load for that task. It chooses faster cores whenever they are underutilized, and when calculated loads result in a tie. Furthermore, threads can migrate to cores that have a lower scaled load even if their original cores can become idle. Conventional operating systems perform this only in special cases. AMPS does not allow a thread to migrate if its performance is not likely to improve. This approach helps maintain higher performance for all the workloads.

AMPS’ asymmetry awareness proved to be an efficient solution to achieve speedup and stability on a performance-asymmetric SMP test system [4]. But, as platforms expand and become more complex, developers will likely see NUMA-style memory and cache architectures implemented with larger core counts. Thus, AMPS was also designed to be NUMA-aware by dynamically predicting thread migration overheads in NUMA-style architectures and responding accordingly.

Impact of Memory and Cache Architectures

As platforms scale to a large number of cores, they become far more complex. Some resources are duplicated; some are shared. As an example, consider the Intel® Core™ i7 processor with up to four cores. It introduces both non-uniform cache architecture (NUCA) and non-uniform memory architecture (NUMA) into the platform. Each core of an Intel Core i7 processor can process two threads; An L1 and L2 cache are associated with each core, while an L3 cache is associated with each socket of cores. Intel Core i7 processor also integrates an on-die memory controller, with local memory attached to each socket, but distributed across sockets. The Intel Core i7 processor with four cores processes eight threads, with four L1 and L2 caches, one L3 cache, and a single memory controller.
If performance asymmetry is added to the topology, the complexity takes a significant leap. And the operating system has to become more aware of not only core states, cache sharing, and memory organization, but performance asymmetry. It needs to know where contention will be and how to address it as it assigns tasks to cores with different performance characteristics.

In these types of performance-asymmetric systems, data locality will potentially become a significant issue with certain workloads, especially with rising core counts. In NUMA-style architectures, data sharing across nodes is costly due to remote memory accesses. This is also a concern when migrating threads, especially among different nodes, where cache misses require a remote memory access to another socket’s memory.

Where threads get placed can have a big impact on performance. That means an operating system has to consider the performance risk resulting from task assignment. It needs to be able to predict this impact and have policies for particular conditions, such as when tasks share data and when they don’t, in order to optimize placement.

The more aware the OS is of different performance characteristics of a processor and other performance-impacting architectural features, the better it can optimize scheduling and thread placement.

**Power and Performance**

Performance per watt drives multi-core and many-core design. Power is not just a laptop issue; it significantly affects the server space and data centers.

Performance per watt becomes more critical as core counts increase and the platform becomes more complex. The operating system in a complex, performance-asymmetric platform must be smart enough to ‘do something’ to maintain performance and limit power. Whether that ‘something’ is slow cores down, turn them off, migrate threads to a smaller set, or some other activity, it essentially means schedulers have to be more aware of the topology in the system for the current usage in order to optimize performance and power demand.

Scott Hahn believes we will eventually see mixed cores in processors. But, we’ll probably be looking at how to slow them down first. Today, Intel processors have the capabilities to turn off completely or turn off parts to conserve power. Functional controls available to the operating system can even change the clock’s duty cycle to adjust performance while running at full power. That’s how Scott’s team was able to create a performance-asymmetric platform to test AMPS. But the hard part is slowing down cores to use less power.

Future complex platforms with intelligent power allocation will dynamically allocate performance per watt to meet requirements while optimizing power usage. That means while there are lots of cores, maybe only some of them are at full power and others are slowed down or turned off, so that in fact they’re using less power when things are idle. That all adds even more complexity that the operating system and the scheduler must be aware of, and, in some instances, control.

**Conclusion**

Research regarding future multi-core and many-core processor topologies recommends performance asymmetry to achieve optimal serial and parallel performance, while delivering high performance per watt. Performance-asymmetric topologies might also integrate NUMA- and NUCA-style architectures, plus be able to dynamically control the speed of each core to meet workload performance demands while conserving power.
Awareness of performance asymmetry is not designed into operating system schedulers and applications today, which adversely affects application performance and stability. Operating systems for future performance-asymmetric platforms must become aware of platform topologies in order to make effective decisions about where and when to place threads onto cores. An asymmetric-aware scheduler, like AMPS, should provide asymmetric-aware load balancing, faster-core-first scheduling, and thread migration for SMP and NUMA- and NUCA-style architectures. Plus, it needs to take into account other aspects that create performance asymmetry, such as dynamic power allocation across a platform.

Intel researchers continue to evaluate processor topologies and to help define the right asymmetric-aware operating system for tomorrow's performance-asymmetric platforms.

References/Suggested Reading


About the Author

Ken Strandberg writes technical articles, white papers, seminars, web-based training, and technical marketing and interactive collateral for emerging technology companies, Fortune 100 enterprises, and multi-national corporations. He writes about Software, Industrial Technologies, Design Automation, Networking, Medical Technologies, Semiconductor, and Telecom. Mr. Strandberg can be reached at ken@kenstrandberg.com.