The Other 90%

Steve Oberlin, NVIDIA

Make no mistake, Moore's Law is fading. It's not dead yet, and computing has had to change technology horses multiple times over the last 40+ years to maintain the incredible exponential rate of technology advancement, but we are already experiencing the effects of CMOS's demise and there is (as yet) no plausible fresh horse ahead. We first felt the chill as a frequency wall right after the turn of the century, when single-core microprocessors appeared to top out between 3-4 GHz, but one could have seen it coming earlier by plotting the fraction of power devoted to leakage vs. active switching over the prior decade and seen them converging in that same period of time.

The failure isn't really in the terms first stated by Gordon Moore, that the dimensions of silicon features used to fabricate electronics would halve regularly, delivering 4x the number of transistors and other interesting devices, rather it is in a corollary he didn't mention, that CMOS manufacturers would exploit the shrinking dimensions to get an additional efficiency benefit by cutting the voltage supply each generation. We are still on track to regularly shrink dimensions through 2020, so are still reaping a bounty of transistors with each new process node, but the rise of leakage current as threshold voltage is lowered has slowed the power efficiency benefit.

There are mitigations. If we don't fully exploit the reduction in dimensions, if we lengthen the transistor gate, we can significantly reduce the leakage penalty. Longer gate length means slower switching, however, so parallelism must increase in order to compensate for an inability to increase clock speed. This drove the switch from Moore's Law-driven performance manifesting in clock speed and single-core performance to more-or-less constant-clock processors with ever-increasing core counts.

The architecture features driven by the full force of Moore's Law, as seen in the last decades of the prior century, are not optimal in a world of constant (or reducing) clock, relatively-increasing latencies, and a need to rely on replication (parallelism) as the dimension supporting performance scaling. CPUs are latency-optimized, and many of the extreme features (e.g., speculative execution, wide multi-way instruction issue, branch prediction, etc.) designed to reduce the time it takes to execute a single thread of instructions start to cost more than the benefit they deliver when multi-thread execution is more important. Multi-core microprocessor CPUs cores have simplified architecturally and the most energy-efficient are today more simple than they were a decade ago.

The architecture of Graphics Processing Unit (GPU) accelerators evolved under different selection pressures than CPUs. Graphics algorithms are extremely parallel, so the dimension of optimization for GPUs has always been in the throughput dimension. As a result, in a world of constant flattening Moore's Law, their single-instruction, multiple thread (SIMT) architecture (designed to support hundreds to thousands of simple cores streaming pixels to computer screens) were optimized to hide latency, not reduce it. This has proved to be a valuable attribute, when GPUs were generalized with floating point functional units, for acceleration of HPC kernels, and led to the rise of accelerated nodes providing the highest efficiency supercomputing available today. The top 15 of the Green500 most energy-efficient supercomputers on the Top500 list are all GPU-accelerated systems.

CPU core architecture is also evolving in the parallel dimension, with the addition of multiple threads and SIMD ("vector") instructions, but there are still orders of magnitude differences in inherent latency hiding capability between contemporary GPUs and CPUs. There is no free lunch, however, and the extreme latency-hiding ability of GPUs comes at the expense of latency reduction, making single-thread performance of GPU cores terrible when compared to even modestly-capable CPUs. Amdahl's Law, unlike Moore's Law, is still in full effect, so serial (single-thread) performance – even in weak scaling scenarios where work is scaled and spread over many distributed memory nodes in a modern HPC cluster – will ultimately limit the performance of any less-than-100%-parallel application.

The optimal architecture, then, for a general-purpose scientific workload, ideally includes both energy-efficient throughput-optimized capacity for the parallel work and fast latency-optimized capability to execute the serial work as quickly as possible. The current trend in heterogeneous node HPC architecture is to couple the most powerful CPUs with the most powerful GPUs, with the ratio of GPUs to CPUs increasing over time as applications are rewritten to expose more and more parallelism. The recently-announced DoE CORAL systems take this approach, using IBM POWER 9 CPUs and NVIDIA Volta GPUs. The success of the competing CPU-only many-core node architecture will be dependent on how quickly CPU parallelism and latency-hiding can be increased in the core architecture without sacrificing single-thread performance. DoE's Trinity and Cori systems try to straddle this line by spitting the machine into two partitions, one with a many-weak-core Xeon Phi processor per node and the other with high-core-performance Xeon CPUs in each node.

If future NSF compute infrastructure resembles the path being followed by DoE, NSF science applications are going to face a significant challenge. The common attribute of both approaches is that ~90% of the performance potential is embodied in the parallel accelerators. Few science applications running on NSF compute platforms today appear well-positioned to exploit that potential.

The implications for scientific applications programmers are clear: Writing your code to exploit distributed memory, using MPI for example, is no longer enough. To

scale performance and sustain a meaningful fraction of the peak performance potential of the underlying hardware, one must also parallelize the code that executes on each node by threading, vectorization, or some combination of the two. Unfortunately, this almost always requires significant work and performance portable high-level language support is lagging the availability of hardware. OpenACC and OpenMP, closely-related but distinct directives-based programming extensions, probably offer the best hope, but their convergence and performance portability future is somewhat obscure. The good news is that the majority of the work is in the restructuring of code to expose maximum parallelism, not in the placement of directives, so switching *between* the two in the future is likely much less onerous than the burden of switching *to* either today.

The bottom line for the scientific HPC user community is that significant investment in applications re-optimization is once again required. Likely it will be less burdensome and perplexing than the switch from shared memory SMPs to distributed memory 15 years ago, but the code base is also many times larger and more complex. In some cases, programmers may be able to accomplish the required optimization with minimal domain expertise, but for best results, scientists may gain huge benefits from examining their algorithms and numerical methods and perhaps approaching problems from different dimensions. The gain in exposed fine-grain parallelism could be worth order-of-magnitude-scale performance, efficiency, and scaling improvements. There is a bolus of work that needs to be processed and additional encouragement for and enablement of computational scientists to perform computer science work will pay off in greater scientific productivity going forward into a post-Moore's Law, even-more-parallel HPC world.