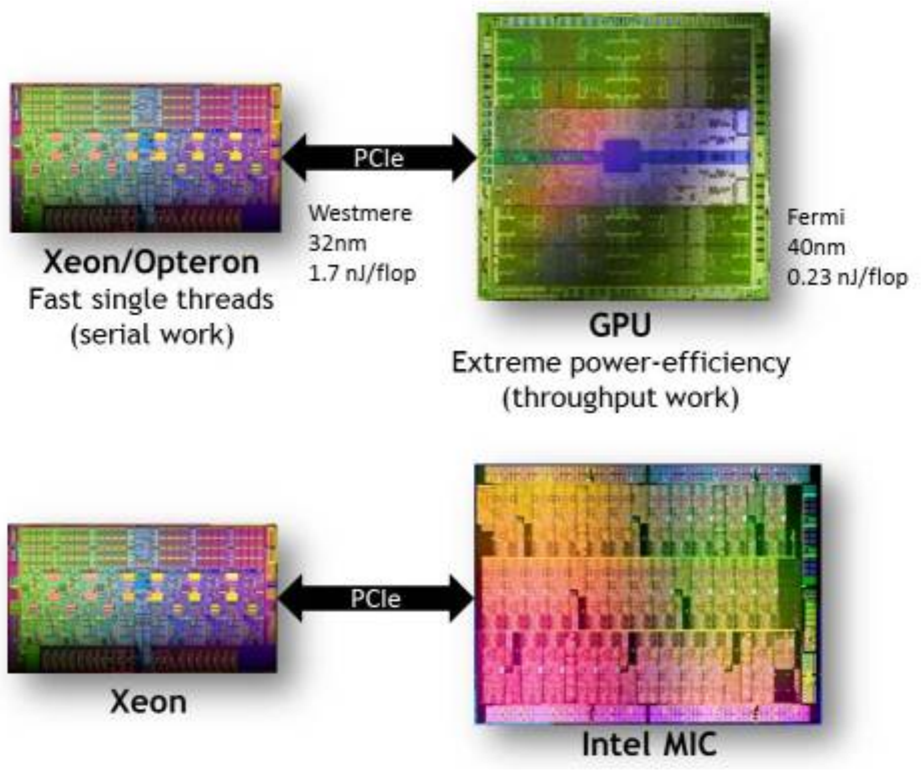
NVIDIA Pokes Holes in Intel's Manycore Story

April 03, 2012 | Michael Feldman

As NVIDIA's upcoming Kepler-grade Tesla GPU prepares to do battle with Intel's Knight Corner, the companies are busy formulating their respective HPC accelerator stories. While NVIDIA has enjoyed the advantage of actually having products in the field to talk about, Intel has managed to capture the attention of some fence-sitters with assurances of high programmability, simple recompiles, and transparent scalability for its Many Integrated Core (MIC) coprocessors. But according to NVIDIA's Steve Scott, such promises ignore certain hard truths about how accelerator-based computing really works.

Over the past couple of years, Intel has been telling would-be MIC users that its upcoming Knights Corner coprocessor will deliver the performance of a GPU without the challenges of a having to adopt a new programming model -- CUDA OpenCL, or whatever. And since the MIC architecture is x86-based (essentially simple Pentium cores glued to extra wide vector units), developing Knights Corner applications will not be that different than programming a multicore Xeon CPU.



Leveraging that commonality, Intel says their compiler will be able generate MIC executables from legacy HPC source code. And it will do so for applications based on both MPI and OpenMP, the two most popular parallel programming frameworks used in high performance computing. Essentially Intel is promising a free port to MIC.

Not so fast, says Scott, the former Cray alum who joined NVIDIA last year its chief technology officer of the Tesla business. According to him, porting applications for MIC, or even developing new ones, won't be any easier than programming GPUs, or for that matter, any accelerator. In a blog posted on Tuesday, he described the problems with Intel's manycore narrative and its claims of superiority over GPU computing.

Scott is not arguing against the MIC as an accelerator, per se. He and most of the community are convinced that HPC needs a hybrid (or heterogeneous) computing to move performance forward without consuming unreasonable amounts of energy. Traditional CPUs, whose cores are optimized for single-threaded performance, are not designed for work requiring lots of throughput. For that type of computing, much better energy efficiency can be delivered using simpler, slower, but more numerous cores. Both GPUs and the MIC adhere to this paradigm; they just come at the problem from different architectural pedigrees.

The problem is that running throughput code on a serial processor sucks up too much energy, which is the situation many users are facing today with conventional CPUs. Conversely, running serial code on a throughput processor is just too slow, and defeats the purpose of having an accelerator in the first place.

Even if low single-threaded performance wasn't an issue, today's accelerators live on PCIe cards with limited amounts of memory (usually just a handful a gigabytes) that exists at the end of a PCIe bus. So if the entire application were to run on the accelerator, all its data and instructions would have to be shuttled in from main memory in chunks. Consider that today, with only a portion of the application living on the GPU, the PCIe bottleneck can still hinder performance. Stuffing the whole program on the accelerator would make it that much worse.

So the main thrust of Scott's critique is that for hybrid computing to work, you have to split the application intelligently between the CPU host and the accelerator. That's true, he says, whether you're talking about an x86-based accelerator like MIC or a graphic-based one like Tesla. "The entire game now is how do we deliver performance as power efficiently as possible," he told HPCwire.

Intel has revealed very little about application performance on the future MIC parts, and has not really addressed how that application split is going to work programmatically, or even that it's necessary. To date, they and some of the early MIC adopters have mostly talked about recompiling existing codes, based on OpenMP and/or MPI, and running the resulting executable natively on MIC.

Running MPI codes on a manycore architecture is particularly problematic. First there's the memory capacity problem mentioned above (each MPI process uses quite a bit of data). And then there's the fact that once the number of MPI processes exceeds the accelerator core count -- 50-plus for Knights Corner -- the application would have to use the server node's network card to communicate with MPI processes running on other nodes. As Scott points out in his blog, that's far too many MPI processes for a typical network interface; all the contention would overwhelm the available bandwidth.

OpenMP has the opposite problem, since most programs using this model don't scale beyond more than 4 to 8 tasks. As a result, there would no way for most OpenMP applications to utilize the 50-plus cores expected on Knights Corner-equipped nodes. And once again, there's the memory capacity problem. Like MPI, OpenMP expects to live in the relatively spacious accommodations of the CPU's main memory.

Scott says if you're just going to use a compiler to transform your existing application to run on the MIC, you're not doing hybrid computing at all. More importantly, running the entire code on the accelerator does not take performance into account. After all, the idea is to speed up the application, not just recompile it so that it functionally works. "We don't think it's legitimate to talk about ease of programming without talking about performance," he says.

Scott argues that for applications to take advantage of these new throughput processors, programmers will have delve into some sort of hybrid programming model that splits off the parallel throughput code from the serial code. For NVIDIA GPUs, the parallelism can be exposed with CUDA or with the emerging set of OpenMP-like directives for accelerators, known as OpenACC. There is already an initial CUDA port for x86 developed by PGI, so that's one option. But the OpenACC framework is likely to reach a larger audience of developers since it offers a higher level of abstraction than CUDA and it looks like it will eventually be folded into the industry-standard OpenMP API.

The idea is that programmers can use OpenACC today to develop GPU-accelerated applications with the anticipation they will be able to use the same code for other accelerator-based hardware platforms, like MIC and AMD's Fusion or discrete GPU processors. Intel and AMD have not jumped on the OpenACC bandwagon as of yet, but were it to be adopted as a standard and demanded by their customers, they would certainly have to support it.

Even OpenACC is not a magic bullet though. The programmer still has to do dive into the source code and tell the compiler where and how to carve out parallel code for the accelerator. And as Scott admits, that can be a significant effort, especially for large legacy HPC applications that were written for homogeneous CPU-only machines.

But, he maintains, if you're interested in taking advantage of the performance offered by throughput processor like GPUs and MIC, the work has to be done. Processor clocks are not likely get any faster than they are today. So the only way to increase performance is via parallelism. As Scott says, "Computers aren't getting faster, they're only getting wider."

Related Articles

The Heterogeneous Programming Jungle

NVIDIA Eyes Post-CUDA Era of GPU Computing

Intel Touts Manycore Coprocessor at Supercomputing Conference

---