Dr.Dobb's
THE WORLD OF SOFTWARE DEVELOPMENT

Intel's 50+ core MIC architecture: HPC on a Card or Massive Co-Processor?

Will Intel’s Knights Corner chips function as co-processors like GPUs, or will they be stand-alone many-core Linux systems? The two approaches present very different performance profiles.

April 10, 2012
URL:http://www.drdobbs.com/parallel/intels-50-core-mic-architecture-hpc-on-a/232800139

In September 2011, the Texas Advanced Computer Center (TACC) announced Stampede, a new 10-petaflops-capable (10^{16} or 10,000 trillion floating-point operations per sec.) supercomputer based on the Intel MIC (Many-Integrated Core) architecture. The Stampede announcement demonstrates a substantial and long-term commitment by Intel to deliver massively parallel many-core hardware to the high-performance computing (HPC) market by January 2013. The heart of the Stampede system will be the 50+ core Knights Corner (KNC) processor chips packaged in a PCIe form factor (the same form factor used by GPU computing co-processors). More than 8 of the 10 petaflop/sec. of peak floating-point performance will be provided by the Knights Corner PCIe co-processors.

The entrance of an x86-based many-core design into the HPC leadership class marketplace raises a key question: Will the Intel Knights Corner chips compete as co-processors that accelerate application performance like GPUs do, or will they provide a "compile and run" alternative where the MIC device behaves like a stand-alone many-core Linux system?

The fact that Intel has now made substantial commitment to teraflops-capable, massively-parallel hardware devices comes as no surprise. Many in the computer industry, including me, have observed that CPUs and GPUs are following convergent evolutionary paths. As I note in my *Scientific Computing* article, "HPC's Future", the failure of Dennard's scaling laws forced chip manufacturers to switch to parallelism to increase processor performance. Due to power and heat issues, many-core processors have become a necessity as it is no longer possible to significantly increase the performance of a single processing core.

This new era of multi- and many-core computing has been disruptive to the software industry as it requires that existing applications be redesigned to exploit parallelism (rather than clock speed) to achieve high application performance on this new parallel hardware. During this transition to massively parallel programming, the owners of legacy code bases are faced with some difficult choices because there are no generic "recompile and run" solutions. As I noted in my *Scientific*

Computing article, "Redefining What is Possible":

Legacy applications and research efforts that do not invest in multi-threaded software will not benefit from modern multi-core processors, because single-threaded and poorly scaling software will not be able to utilize extra processor cores. As a result, computational performance will plateau at or near current levels, placing the projects that depend on these legacy applications at risk of both stagnation and loss of competitiveness.

Vendors of parallel-processing hardware are making significant investments to ease the cost of transitioning legacy software to massively parallel computing. The challenge for legacy software owners lies in understanding how well these vendor efforts translate to production application performance.

While still at an early pre-hardware release stage, it is possible to draw some preliminary conclusions based on an analysis of the MIC and GPU architectures and the currently available information about the NVIDIA Kepler and Intel Knights Corner chips. In this article, I consider these two processors based on established high-level comparative measures such as memory capacity, balance ratios, and Amdahl's Law in the context of four programming paradigms:

- MPI (Message Passing Interface)
- Directive-based programming like OpenMP and OpenACC
- Common libraries providing FFT and BLAS functionality
- Language platforms based on a strong-scaling execution model (CUDA and OpenCL™)

Convergent Evolution in HPC: Intel MIC and NVIDIA GPU

Five years ago, NVIDIA disrupted the high performance computing industry with the release of CUDA in February 2007. In combination with low cost of teraflop/sec (single-precision) GPU hardware, NVIDIA brought supercomputing to the masses along with co-processor acceleration of both C and Fortran applications. With the MIC announcement, Intel has followed suit along this convergent evolutionary path.

While similarly packaged as a PCIe device, Intel has taken a different architectural approach to massively-parallel computing hardware. The KNC generation of MIC products appears to be HPC-oriented, which means high-end customers can now choose from two types of teraflop/sec capable PCIe-based co-processors.

Comparing NVIDIA GPUs and Intel MIC

GPU designs utilize many streaming multiprocessors (SM) where each SM can run up to 32 concurrent SIMD (Single Instruction Multiple Data) threads of execution. The current generation of Fermi GPUs supports 512 concurrent SIMD threads of execution that can be sub-divided into 16 separate SIMT (Single Instruction Multiple Thread) tasks. The upcoming NVIDIA Kepler GPUs will support even greater parallelism. For example, the GTX 680 will support 1,536 concurrent SIMD threads.

Teraflop/sec. performance is achieved through a per-SM hardware scheduler that can quickly identify those SIMD instructions that are ready-to-run (meaning they have no unresolved dependencies). Ready-to-run instructions are then dispatched to keep multiple integer, floating-point, and special function units busy. A per-GPU hardware scheduler similarly allocates work (via CUDA thread blocks or OpenCL™ work-groups) to ensure high utilization across all the SM on a GPU.

High flops/watt efficiency is realized through the use of a SIMD execution model inside each SM that requires less supporting logic than non-SIMD architectures. GPU hardware architects have been able to capitalize on this savings by devoting more power and space to 64-bit addressing, additional ALUs, floating-point, and Special Function Units for transcendental functions. Some reviewers report that NVIDIA expects Kepler to deliver "about 3x improvement in [double precision] performance per watt …" over Fermi.
Other notable characteristics include:

- Data-parallel operations are spread across the SMs of one or more GPU devices.
- Task-parallelism is accomplished by running concurrent kernels on different SMs and/or multiple devices plus the host processor.
- MPI jobs are accelerated by using one or more GPUs per process and capabilities like GPUdirect, which optimizes data transfer into device memory.

Intel MIC

The Intel MIC architecture in the KNC chip utilizes x86 Pentium-based processing cores that support four threads per core. According to The Register, the next generation Knights Corner has "*64 cores on the die, and depending on yields and the clock speeds that Intel can push on the chip, it will activate somewhere between 50 and 64 of those cores and run them at 1.2GHz to 1.6GHz*". The preceding implies that each KNC chip will provide between 200 and 256 concurrent threads of execution.

Teraflop/sec floating-point performance can be achieved when enough of the SMP threads issue special SSE-like instructions to fully utilize an enhanced vector/SIMD unit that resides on each core. (Note: this requires the use of a special "**-mmic**" compiler switch to tell the Intel compilers to look for cases when these MIC-specific vector instructions can be utilized, or via hand-coding with intrinsic operations.)

High flops/watt efficiency is realized by leveraging the simplicity of the original in-order short execution pipeline Pentium design and the power savings of chips created with their 22 nm manufacturing process. MIC also derives high flops/watt from using wide vector units. The logic for the Pentium core is small relative to modern processor cores, which left room for additional logic to support 64-bit addressing, four concurrent threads per core, and a large 512-bit wide vector unit. Per the TACC Stampede announcement, the initial revision of KNC per-core vector unit will deliver 50% higher floating-point performance in 2013.
Other notable characteristics include:

- Data-parallel tasks appear to be mainly accelerated by the per-core vector units.
- Task-parallelism is accelerated by running a task per thread and separate tasks on the device(s) and host processor.
- MPI jobs are accelerated by using one or more MIC devices per process or capabilities like MIC-as-a-compute-node discussed later in this article.

	NVIDIA GPU	**Intel MIC**
Degree of Parallelism	Fermi supports 512 concurrent SIMT threads of execution. Kepler will triple this number to 1,536 threads.	Knights Corner expected to support between 200 and 256 concurrent threads.

Achieving High Performance	A per-SM hardware scheduler keeps multiple computational units busy by identifying and dispatching any ready-to-run SIMD instructions.	A compiler or programmer utilizes special SSE-like instructions to keep each per-core vector unit busy.
Achieving Power Efficiency	The per-SM SIMD execution model requires less supporting logic, leading to high power efficiency and floating-point performance. Expect a 3x increase in Kepler double-precision efficiency.	Leverages the simplicity of the original Pentium design and the floating-point capability of a 512-bit vector unit along with the power savings of manufactured with a 22 nm process.
Data-parallel **acceleration**	Data-parallel operations are spread across the SMs of one or more GPU devices.	Data-parallel operations accelerated by the per-core vector units and are spread across the cores of one or more devices.
Task-parallel **acceleration**	Concurrent kernel execution allows multiple kernels to run on one or more SM.	Concurrent threads can run multiple tasks on the device.
MPI acceleration	MPI jobs are accelerated by using one or more GPUs per MPI process and optimized data transfer capabilities like GPUdirect.	MPI jobs are accelerated by using one or more MIC devices per MPI process, or one MPI process per MIC core.

Table 1: GPGPU and MIC architectural approaches to massive parallelism

For more detailed information about the MIC architecture, I recommend reading, "Larrabee" A Many-Core x86 Architecture for Visual Computing" as MIC is based on the Larrabee computing architecture with the visualization capability removed. The NVIDIA documentation such as the Fermi whitepaper, my tutorial series in Dr. Dobb's, and my book, "CUDA Application Design and Development" are good sources for more detailed information about NVIDIA GPUs.

The Code Migration Conundrum

While teraflop/sec. performance is compelling, there is no guarantee that any of these devices will deliver high performance (or even a performance benefit) for any given application. This uncertainty coupled with the risk and costs of a porting effort has kept many customers with legacy code from investing in this new technology.

Chip manufacturers, and the industry as a whole, have invested heavily in several programming models to make porting efforts as fast and risk free as possible. As mentioned, NIVIDA's investment in CUDA has been very successful. Not surprisingly, well established legacy programming models have also attracted much attention. For comparison purposes, this article will focus on four types of programming models that are supported by the industry for massively-parallel co-processors:

- MPI (Message Passing Interface)

- Directive-based programming like OpenMP and OpenACC
- Common libraries providing FFT and BLAS functionality
- Language platforms based on a strong-scaling execution model (CUDA and OpenCL)

The current packaging of GPU and MIC massively-parallel chips as external PCIe devices complicates each of these programming models. For example, the overhead incurred by host/device data transfers breaks an assumption made by the SMP execution model that any thread can access any data in a shared memory system without paying a significant performance penalty. Efforts like OpenACC (and potentially OpenMP 4.0) are attracting attention because they provide a standard method to specify data locality. The hope is that minimal code changes will be required to modify legacy code to run on co-processors.

Limited on-board memory also requires partitioning computational problems into pieces that can fit into device memory. At this time, a human programmer is required to partition larger computational problems into smaller pieces that can run on a co-processor and achieve high performance by efficiently overlapping computation and communication. Hybrid memory cubes hold hope for large memory co-processors in the future, but it is unclear whether the next generation NVIDIA Kepler or Intel MIC cards will use this technology.

Succinctly, achieving performance with co-processors generally requires that the programmer to:

- Transfer the data across the PCIe bus onto the device and keep it there;
- Give the device enough work to do;
- Focus on data reuse within the co-processor(s) to avoid memory bandwidth bottlenecks.

Bottom line: Semantic limitations coupled with the costs and complexity of utilizing data located in multiple memory spaces plus limitations in on-board memory capacity currently prevents the automatic translation of legacy code to both GPUs and MIC co-processors. Some porting effort is required.

MIC as a Linux compute node

The MIC architecture is based on modified Pentium processing cores coupled to a per-core vector unit. The performance implications of this design decision will most certainly be hotly debated by the GPU and MIC communities for years to come. The Intel Server Room Blog notes, "The MIC architecture products are first and foremost compute nodes. They run an open source [L]inux OS, they are networked and can run applications." So, let's analyze MIC as if it were a separate many-core Linux computer connected to the host system by the PCIe bus, or MIC-as-a-compute-node.

From a source compatibility point of view, this model is attractive to organizations with millions of lines of legacy code: take your existing legacy source code, recompile for MIC using a "**–mmic**" compiler flag, and run. Most build systems make it easy to specify both the compiler and any special flags like "**-mmic**", which supports the claim by Jeff Nichols, a Director at Oak Ridge National Laboratory, that they were able to port "millions of lines of code... literally in days" to MIC. MIC-as-a-compute-node is of interest to owners of existing OpenMP, MPI + vector, and hybrid MPI + OpenMP applications as these codes have the potential to recompile and run.

But how well will it run? Even without hardware at hand, it is possible to get a sense of how well recompiled x86-based legacy code will run on MIC-as-a-compute-node by considering three factors: Amdahl's Law, wide SSE-like vector characteristics, and on-board memory capacity.

Amdahl's Law gives an approximation that models the ideal speedup that can happen when single-threaded programs are modified to run on parallel hardware. The speedup of a program using multiple processors in parallel computing is limited by the time required by the sequential fraction of the program. In the best case, those sections of code that can be parallelized can have their runtime reduced by a factor of *N*, where *N* is the number of parallel processing elements. Obviously, the time taken to complete the serial sections of code (e.g. those sections that cannot be parallelized) will not change, which means they can dominate the runtime when the number of parallel processing elements, *N*, is large.

Co-processors have the advantage of being able to exploit the performance capabilities of the latest and highest clock rate processors in the host system. In contrast, using MIC-as-a-compute-node means that serial sections of code will run on a single 1.2 GHz to 1.6 GHz Pentium core. This difference in processor performance relative to a state-of-the-art processing core can increase the fraction of time spent in sequential code and cause applications to run more slowly on MIC-as-a-compute-node compared to MIC-as-a-coprocessor.

The key to MIC floating-point performance is the efficient use of the per core vector unit. To access the vector unit, the compiler must be able to recognize SSE compatible constructs so it can generate the MIC SSE-like assembly language instructions. The test with your current hardware and compiler is simple: tell your compiler to utilize the SSE instructions on your x86 processor through the "**–msse**" or other compiler switch. Applications that run faster will probably benefit from the MIC vector unit. (Conversely, check if the application slows down by disabling the use of SSE instructions.) Those applications that don't benefit from the SSE instruction set will most likely be limited to the performance of the individual Pentium based cores (or that of 50 to 64 1.2GHz - 1.6GHz Pentium processors). For additional analysis and discussion of "**-msse**" compatibility on MIC, see Greg Pfister's "MIC and the Knights" by Greg Pfister.

Balance ratios are conventional, established measures used in HPC to evaluate potential system performance. My 2010 GTC presentation lists the four important balance ratios for the current PNNL (Pacific Northwest National Laboratory) Chinook supercomputer: memory capacity, memory bandwidth, aggregate link bandwidth, and interconnect latency.

Table 2 below compares the MIC balance ratios for legacy workloads against the PNNL Chinook supercomputer. (*Note: This table makes several assumptions about MIC capabilities and so the values should be considered with caution.)* The ratios can be easily updated as Intel publishes more performance data on KNC:

- Intel has not yet released information about the amount of memory that will be available on each MIC card. This table arbitrarily assumes that each MIC card will contain 8 GB of RAM.
- Intel has yet not released information about the how fast MIC can communicate across the PCIe bus. The table arbitrarily assumes effective utilization of the available PCIe bandwidth (e.g. 16 GB/s on a PCIe gen-2 bus and 32 GB/s should the Knights Corner cards utilize a PCIe gen-3 interface).
- Intel has not yet released information about communication latency through the internal ring interconnect or across the PCIe bus. The following table assumes that communications latency will be software limited and comparable to existing Infiniband software stacks.
- This table lists 8-core Chinook and 50-core KNC balance ratios based on the assumption that peak floating-point performance will be achieved by all the per-core vector units.

Balance Category	MIC Knights Corner (50-core)	Chinook (8-core)

Memory Amount (Bytes/flop)	0.008	0.46
Memory Bandwidth (B/s/flop/s)	?	0.21
Aggregate Link BW (B/s/flop/s)	0.016 (PCIe gen-2) 0.032 (PCIe gen-3)	0.17
Interconnect Latency (ms)	< 2	1.1

Table 2 : Potential balance ratios for MIC-as-a-compute-node

With the exception of latency, larger values are preferred for generic legacy workloads such as NWChem – a porting effort referenced in the MIC literature – for which the Chinook supercomputer was designed. The small ratio of MIC memory capacity to flops indicates that memory capacity will be a significant problem and likely obstacle for many legacy applications. From a communications bandwidth and latency stand-point, MIC-as-a-compute-node is very interesting as most applications will run at a fraction of the peak floating-point rate. In particular, a PCIe gen-3 bus has the potential to act as a 256 Gb/s data link, which exceeds current, commodity InfiniBand capabilities.

When running on MIC-as-a-compute-node, legacy "compile and run" customers should consider the following factors as listed in Table 3.

Feature Categories	**Projected Application Profile to Run Well on MIC-as-a-Compute-Node**
Memory Usage	• MPI and OpenMP application code + data must have a small per-core memory footprint. • Must be cache friendly to efficiently use the on-core cache memory and avoid memory bandwidth bottlenecks. • Need to avoid serialization side-effects from semaphores and atomic operations like C++ smart pointers and reference counted objects.
Balance of Scalar and Vector	• High flop rates will be achieved through SSE-like vector operations. • Need to avoid Amdahl's Law sequential bottlenecks due to low Pentium performance compared to modern high clock-rate CPU cores

Table 3: Project application characteristics to run well on MIC-as-a-compute-node

In summary, MIC does not eliminate the need to rewrite legacy applications except for those applications that can run in the memory footprint of the PCIe device. Of that subset of applications, only those that currently do not rely on serial calculations but depend mostly on SSE acceleration will be able to benefit from the per-core vector unit to achieve high floating-point performance. Further, high performance will probably require cache friendly applications. Those applications that can recompile and run will probably still require modification to make full use of the MIC capabilities. In particular, memory limitations will most likely require re-architecting the application to use the current generation of MIC devices as a co-processor.

Programming Model Considerations for GPUs and MIC co-processors

When evaluating co-processors for a legacy application, it is necessary to consider the characteristics of the programming model. Whenever possible, use benchmarks to evaluate the transfer and computational efficiency of co-processor applications that are similar to the intended application.

MPI has strong support as it is the *de facto* standard distributed scientific computing framework. As mentioned above, developers must consider memory footprint and Amdahl's Law sequential code limitations when porting MPI code to MIC. When porting to both MIC and GPU co-processors, some considerations are:

- Most implementations use one co-processor per MPI process.
- Network bandwidth limitations tend to be a key bottleneck making most MPI applications network rather than compute bound. Use whatever features are available (like GPUdirect) to optimize data transfers.
- Use all the co-processors in a single MPI process when communications latency is an issue. This can be particularly useful for applications that perform many latency bound operations such as reductions.

While directives-based programming such as OpenMP has strong support in the developer community, directives for co-processors is a "Work-in-progress." For legacy code based on OpenMP, code modifications will certainly be required as legacy applications do not have a concept of data location (Tthey assume an SMP model). Something like "*#pragma target (device)*" is required. Standardization is moving quickly to prevent a "tower of Babel" proliferation of incompatible pragma specifications.

Note that many OpenMP code bases were developed when two, four, or eight cores were considered "many." High core count processors may expose scaling issues. In this regard, you should:

- Be aware that atomic and common synchronization operations (such as semaphores, reference counting, etc.) might expose unexpected serialization bottlenecks.
- Pay particular attention to scaling behavior on cache-coherent architectures like MIC and the impact of conditional operations on SIMD-based GPU architectures.
- Many legacy OpenMP apps have directives employed at the innermost loop level, which limits the achievable parallelism. Code modification may be required to expose more parallelism.

Common libraries providing FFT and BLAS functionality should perform well on co-processors because they are optimized for a particular architecture and set of hardware capabilities. (This assumes data transfers do not limit performance.)

Language platforms based on a strong-scaling execution model, such as CUDA and OpenCL, will likely perform well on both architectures because they provide linear scaling according to number of processing elements and provide the best Amdahl's Law reduction in parallel code runtime. Scaling behavior of the computational kernels should not be an issue unless global atomic operations are utilized.

In all programming approaches, high performance can be achieved when the compute intensive portions of the application conform to the three rules of high-performance co-processer programming mentioned previously. If not, expect floating-point performance to be either PCIe or device memory limited.

In summary, expect to use MIC and GPUs as co-processors and that software will rapidly evolve to hide differences between co-processor hardware. Growing support for OpenACC by vendors like CAPS and PGI can make co-processors an attractive, highly portable option for legacy OpenMP codes because the source code intrusion is relatively small.   Vendor libraries provided by NVIDIA and Intel already provide an optimized framework for some applications. Generally, applications written in OpenCL and CUDA will deliver the greatest performance and longevity due to their use of a strong scaling execution model that can achieve a linear parallel code speedup regardless of number of processing elements. In addition, these languages provide asynchronous queues that can choreograph tasks and data movement among numerous devices.

Approach	**Programming Considerations for Legacy Codes**
MPI (Message Passing Interface)	• Co-processor accelerated MPI processes can potentially make better use of on-board resources. • Assuming 50 cores and 8 GB per device, each MPI process on a MIC card will have roughly 160 MB for program and data storage. Note: more cores implies less data per core. • MPI processes on MIC cores must be particularly frugal in memory usage because each MPI process requires a separate copy of all data. • MIC-as-a-compute-node may exhibit Amdahl's Law sequential bottlenecks due to low clock rate Pentium performance compared to modern CPU cores.
Directive-based programming (OpenMP and OpenACC)	• OpenACC has the potential to transparently run OpenMP applications on co-processors from any vendor with minimal modification. • Legacy code can potentially "compile and run" on MIC-as-a-compute-node assuming both program and data fits in memory. • High core counts may expose surprising serialization bottlenecks. For example, reference counted objects such as smart pointers in C++ may cause serialization bottlenecks on atomic operations. • Applications should be "cache friendly" to avoid memory bandwidth bottlenecks. The effectiveness and messaging overhead of the MIC cache coherency model at high core counts is currently unknown. • MIC-as-a-compute-node may exhibit Amdahl's Law sequential bottlenecks due to low clock rate Pentium performance compared to modern CPU cores. • Beware the PCIe bottleneck.
Common libraries providing FFT and BLAS functionality	• Optimized libraries should run well and reflect the co-processor hardware performance capabilities.

	• Subject to memory capacity and bandwidth limitations of the PCIe bus.
Accelerator execution model like OpenCL	• Strong-scaling execution model applications like those written OpenCL (and potentially CUDA) have the potential to run well.

Table 4: Summary table of various programming model comments

The rapidity in which the industry is moving to support legacy programming is reflected in NVIDIA's directives based developer effort at SC11 that delivered 5x to 20x speedups for several legacy applications in two days or less. Intel's HPC General Manager Rajeeb Hazra expresses a similar view about MIC compiler technology: "It eliminates code porting to a certain extent," redefining the effort so that, "It just makes it an optimization job." As always, *caveat emptor* still applies regardless of the technology used.

Hands-on comments by TACC about programming MIC as of March 1, 2012 can be found in the "Oil and Gas High Performance Computing Workshop" video.

Summary

Technical innovation is rapidly evolving massively parallel devices into ever more capable computational tools. While both NVIDIA GPUs and Intel MIC devices support multiple programming models, the current PCIe-based packaging imposes memory capacity, data locality and bus bandwidth limitations that strongly favors the use of these devices as external co-processors. At this time, we see a convergence of evolutionary characteristics where the prime selection criterion is performance rather than a head-on collision of technical approaches.

As in any market, price versus expected benefit will dominate procurement decisions and market success.

From a performance point of view, the KNC chip looks to be competitive with GPUs as a teraflops-capable co-processor. Pricing information is not available for NVIDIA Kepler or Intel KNC products, so it is not possible at this time to make a price vs. performance comparison. The TACC announcement shows that Intel is definitely looking at high performance computing. Meanwhile, NVIDIA has established a strong market presence and massive base of CUDA developers with products starting around the $150 - $180 price range and extending to HPC products priced in the thousands of dollars.

Clearly, benchmark results will be a hot topic once Kepler and KNC chips become available. Benchmarks will certainly be devised to exploit architectural differences between both products to accelerate some applications more than others. This will be a good thing as feedback from the forthcoming benchmark battles will doubtless spur technical innovations that will improve the performance of future GPU and MIC generations of products. As this article notes, it does not really matter if a software effort is charged as "software porting work" or "application performance tuning" as both time and money will be required to effectively use MIC and GPU devices.

The intention of all programming models is to abstract the hardware interface to preserve performance and reduce or eliminate porting costs. Eventually, software will mature to the point that performance decisions will focus more on the hardware than software. As a programmer, it is always best to look at what works best for you now and in the future. For this reason, programming languages like CUDA and OpenCL are attractive due to their use of a strong scaling execution model in combination with asynchronous queues. In other words, applications written now will be able to choreograph numerous tasks across one or more devices to scale to whatever number of concurrent threads of execution the hardware vendors can provide us.

Rob Farber is an analyst who writes frequently on High-Performance Computing hardware topics.