CHECKLIST FOR PROGRAMMING

Intel® Xeon Phi™ Coprocessors

by James Reinders,
Director, Software Evangelist, Intel
The Intel® Xeon Phi™ coprocessor extends the reach of the Intel® Xeon® family of computing products into higher realms of parallelism. This article offers the key tips for programming such a high degree of parallelism, while using familiar programming methods and the latest Intel® Parallel Studio XE 2013 and Intel® Cluster Studio XE 2013—which both support the Intel Xeon Phi coprocessor.

It is worth explaining this checklist in more depth, and that is the purpose of this article. You can see that preparing for Intel Xeon Phi coprocessors is primarily about preparing for a 50+ core x86 SMP system with 512-bit SIMD capabilities. That work can happen on most any large, general purpose system, especially one based on Intel Xeon processors. Intel Parallel Studio XE 2013 and Intel Cluster Studio XE 2013 will support your work on an Intel Xeon processor-based system with or without Intel Xeon Phi coprocessors. All the tools you need are in one suite.

Checklist

A checklist for programming for an Intel Xeon Phi coprocessor looks like this:

- Make sure your application scales beyond 100 threads (usually 150 or more).
- Make sure your application either:
  - Does most computations as efficient vector instructions (requires vectorization)
  - Uses a lot of memory bandwidth with decent locality of reference
- The application is written using your favorite programming languages and parallel models to achieve the above.
- Use your favorite tools, the same ones you use for programming for Intel® Xeon® processors. Get the latest versions that include support for Intel® Xeon Phi™ coprocessor support (such as Intel® Cluster Studio XE 2013, Intel® Parallel Studio XE 2013, Rogue Wave TotalView* debugger and IML* Library, NAG Library*, or Allinea DDT* debugger)

It is worth explaining this checklist in more depth, and that is the purpose of this article. You can see that preparing for Intel Xeon Phi coprocessors is primarily about preparing for a 50+ core x86 SMP system with 512-bit SIMD capabilities. That work can happen on most any large, general purpose system, especially one based on Intel Xeon processors. Intel Parallel Studio XE 2013 and Intel Cluster Studio XE 2013 will support your work on an Intel Xeon processor-based system with or without Intel Xeon Phi coprocessors. All the tools you need are in one suite.
Introduction

Intel Xeon Phi coprocessors are designed to extend the reach of applications that have demonstrated the ability to reach the scaling limits of Intel Xeon processor-based systems, and have also maximized usage of available vector capabilities or memory bandwidth. For such applications, the Intel Xeon Phi coprocessors offer additional power-efficient scaling, vector support, and local memory bandwidth, while maintaining the programmability and support associated with Intel Xeon processors.

Advice for successful programming can be summarized as: “Program with lots of threads that use vectors with your preferred programming languages and parallelism models.” Since most applications have not yet been structured to take advantage of the full magnitude of parallelism available in an Intel Xeon Phi coprocessor, understanding how to restructure to expose more parallelism is critically important to enable the best performance. This restructuring itself will generally yield benefits on most general purpose computing systems—a bonus due to the emphasis on common programming languages, models, and tools across the Intel Xeon family of products. You may refer to this bonus as the dual-transforming-tuning advantage.

A system that includes Intel Xeon Phi coprocessors will consist of one or more nodes (a single node computer is “just a regular computer”). A typical node consists of one or two Intel Xeon processors, plus one to eight Intel Xeon Phi coprocessors. Nodes cannot consist of only coprocessors.

The First Intel Xeon Phi Coprocessor, Codename Knights Corner

While programming does not require deep knowledge of the implementation of the device, it is definitely useful to know some attributes of the coprocessor. From a programming standpoint, treating it as an x86-based SMP-on-a-chip with over 50 cores, over 200 hardware threads, and 512-bit SIMD instructions is the key.

The cores are in-order, dual-issue x86 processor cores (which trace some history to the original Intel® Pentium® design). But with the addition of 64-bit support, four hardware threads per core, power management, ring interconnect support, 512-bit SIMD capabilities, and other enhancements, these are hardly the Intel Pentium cores of 20 years ago. The x86-specific logic (excluding L2 caches) makes up less than 2 percent of the die for an Intel Xeon Phi coprocessor.

Here are key facts about the first Intel Xeon Phi coprocessor product:

- It is a coprocessor (requires at least one processor in the system); in production in 2012
- Boots and runs Linux® (source code available at http://intel.com/software/mic)
- It is supported by standard tools including Intel Parallel Studio XE 2013. Listings of additional tools available can be found online (http://intel.com/software/mic).
- It has many cores:
  - More than 50 cores (This will vary within a generation of products, and between generations. It is good advice to not hard code applications to a particular number.)
  - In-order cores support 64-bit x86 instructions with uniquely wide SIMD capabilities.
  - Four hardware threads on each core (resulting in more than 200 hardware threads on a single device) are primarily used to hide latencies implicit in an in-order microarchitecture. As such, these hardware threads are much more important for HPC applications to utilize than hyperthreads on an Intel Xeon processor.
  - Cache coherent across the entire coprocessor.
  - Each core has a 512K L2 cache locally with high-speed access to all other L2 caches (making the collective L2 cache size over 25M).
- Special instructions in addition to 64-bit x86:
  - Uniquely wide SIMD capability via 512-bit wide vectors instead of MMX, SSE or AVX.
  - High performance support for reciprocal, square root, power, and exponent operations
  - Scatter/gather and streaming store capabilities for better effective memory bandwidth
  - Performance monitoring capabilities for tools like Intel® VTune™ Amplifier XE 2013

Maximizing Parallel Program Performance

The choice whether to run an application solely on Intel Xeon processors, or to extend an application run to utilize Intel Xeon Phi coprocessors, will always start with two fundamentals:

1. Scaling: Is the scaling of an application ready to utilize the highly parallel capabilities of an Intel Xeon Phi coprocessor? The strongest evidence of this is generally demonstrated scaling on Intel Xeon processors.

2. Vectorization and Memory Locality: Is the application either:
   - Making strong use of vector units?
   - Able to utilize more local memory bandwidth than available with Intel Xeon processors?

If both of these fundamentals are true for an application, then the highly parallel and power-efficient Intel Xeon Phi coprocessor is most likely worth evaluating.
Ways to Measure Readiness for Highly Parallel Execution

To know if your application is maximized on an Intel Xeon processor-based system, you should examine how your application scales, as well as how it uses vectors and memory. Assuming you have a working application, you can get some impression of where you are with regards to scaling and vectorization by doing a few simple tests.

To check scaling, create a simple graph of performance as you run with various numbers of threads (from one up to the number of cores, with attention to thread affinity) on an Intel Xeon processor-based system. This can be done with settings for OpenMP*, Intel® Threading Building Blocks (Intel® TBB) or Intel® Cilk™ Plus (e.g., OMP_NUM_THREADS for OpenMP). If the performance graph indicates any significant trailing off of performance, you have tuning work you can do to improve your application before trying an Intel Xeon Phi coprocessor.

To check vectorization, compile your application with and without vectorization. If you are using Intel compilers: disable vectorization via compiler switch: -no-vec, use at least -O2 xhost for vectorization. Compare the performance you see. If the performance difference is insufficient, you should examine opportunities to increase vectorization. Look again at the dramatic benefits vectorization may offer as illustrated in Figure 7. If you are using libraries, such as the Intel® Math Kernel Library (Intel® MKL), you should consider that time in Intel MKL routines offer vectorization invariant to the compiler switches. Unless your application is bandwidth limited, effective use of Intel Xeon Phi coprocessors should be done with most cycles executing having computations utilizing the vector instructions. While some may tell you that “most cycles” needs to be over 90 percent, we have found this number to vary widely based on the application and whether the Intel Xeon Phi coprocessor needs to be the top performance source in a node or just to contribute to performance.

The Intel® VTune™ Amplifier XE 2013 can help measure computations on Intel Xeon processors and Intel Xeon Phi coprocessor to assist in your evaluations.

Aside from vectorization, being limited by memory bandwidth on Intel Xeon processors can indicate an opportunity to improve performance with an Intel Xeon Phi coprocessor. In order for this to be most efficient, an application needs to exhibit good locality of reference and utilize caches well in its core computations.

The Intel VTune Amplifier XE product can be utilized to measure various aspect of a program, and among the most critical is “L1 Compute Density.” This is greatly expanded upon in a paper titled Using Hardware Events for Tuning on Intel® Xeon Phi™ Coprocessor (codename: Knights Corner).

When using MPI, it is desirable to see a communication vs. computation ratio that is not excessively high in terms of communication. Because programs vary so much, this has not been well characterized other than to say that, like other machines, Intel Xeon Phi coprocessors favor programs with more computation vs. communication. Programs are most effective using a strategy of overlapping communication and I/O by computation. Intel® Trace Analyzer and Collector, part of Intel Cluster Studio XE 2013, is very useful for profiling. It can be used to profile MPI communications to help visualize bottlenecks and understand the effectiveness of overlapping with computation to characterize your program.

Compiler and Programming Models

No popular programming language was designed for parallelism. In many ways, Fortran has done the best job adding new features, such as DO CONCURRENT, to address parallel programming needs, as well as benefiting from OpenMP. C users have OpenMP as well as Intel Cilk Plus. C++ users have embraced Intel Threading Building Blocks and, more recently, have Intel Cilk Plus to utilize as well. C++ users can use OpenMP as well.
Intel Xeon Phi coprocessors offer the full capability to use the same tools, programming languages, and programming models as an Intel Xeon processor. However, with this coprocessor designed for high degrees of parallelism, some models are more interesting than others.

In essence, it is quite simple: an application needs to deal with having lots of tasks (call them “workers” or “threads” if you prefer), and deal with vector data efficiently (a.k.a., vectorization).

There are some recommendations we can make based on what has been working well for developers. For Fortran programmers, use OpenMP, DO CONCURRENT, and MPI. For C++ programmers, use Intel TBB, Intel Cilk Plus, and OpenMP. For C programmers, use OpenMP and Intel Cilk Plus. Intel TBB is a C++ template library that offers excellent support for task-oriented load balancing. While Intel TBB does not offer vectorization solutions, it does not interfere with any choice of solution for vectorization. Intel TBB is open source and available on a wide variety of platforms supporting most operating systems and processors. Intel Cilk Plus is a bit more complex in that it offers both tasking and vectorization solutions. Fortunately, Intel Cilk Plus fully interoperates with Intel TBB. Intel Cilk Plus offers a simpler set of tasking capabilities than Intel TBB, but uses keywords in the language to enable full compiler support for optimizing.

Intel Cilk Plus also offers elemental functions, array syntax, and “#pragma SIMD” to help with vectorization. The best use of array syntax is implemented along with blocking for caches, which unfortunately means naive use of constructs such as $A[i] = B[i] + C[i]$; for large arrays may yield poor performance. The best use of array syntax ensures that the vector length of single statements is short (some small multiple of the native vector length, perhaps only 1X). Finally, and perhaps most important to programmers today, Intel Cilk Plus offers mandatory vectorization pragmas for the compiler called “#pragma SIMD.” The intent of “#pragma SIMD” is to do for vectorization what OpenMP has done for parallelization. Intel Cilk Plus requires compiler support. It is currently available from Intel for Windows*, Linux*, and Apple OS* X. It is also available in a branch of gcc.

If you are happy with OpenMP and MPI, you are in great shape to use Intel Xeon Phi coprocessors. Additional options may be interesting to you over time, but OpenMP and MPI are enough to get great results. Your key challenge will remain vectorization. Auto-vectorization may be enough for you, especially if you code in Fortran, with the possible additional considerations for efficient vectorization, such as alignment and unit-stride accesses. The “#pragma SIMD” capability of Intel Cilk Plus (available in Fortran, too) is worth a look. In time, you may find it has become part of OpenMP.

Dealing with tasks means specification of tasks, and load balancing amongst them. MPI has provided this capability for decades with full flexibility and control given to the programmer. Shared memory programmers have Intel TBB and Intel Cilk Plus to assist them. Intel TBB has widespread usage in the C++ community. Intel Cilk Plus extends Intel TBB to offer C programmers a solution, as well as help with vectorization in C and C++ programs.

**Coprocessor Major Usage Model: MPI vs. Offload**

Given that we know how to program the Intel Xeon processors in the host system, the question arises of how to involve the Intel Xeon Phi coprocessors in an application. There are two major approaches: (1) “offload” selective portions of an application to the Intel Xeon Phi coprocessors, and (2) run an MPI program where MPI ranks can exist on Intel Xeon processors cores, as well as on Intel Xeon Phi coprocessor cores with connections made by MPI communications. The first is call “offload mode” and the second “native mode.” The second does not require MPI to be used, because any SMP programming model can be employed, including just running on a single core. There is no machine “mode” in either case, only a programming style that can be intermingled in a single application if desired. Offload is generally used for finer-grained parallelism and, as such, generally involves localized changes to a program. MPI is more often done in a coarse-grained manner, often requiring more scattered changes in a program. RDMA support for MPI is available.

The choice is certain to be one of considerable debate for years to come. Applications that already utilize MPI can actually use either method by either limiting MPI ranks to Intel Xeon processors and use offload to the coprocessors, or distributing MPI ranks across the coprocessors. It is possible that the only real MPI ranks be established on the coprocessor cores, but if this leaves the Intel Xeon processors unutilized then this approach is likely to give up too much performance in the system.

Being separate and on a PCIe bus creates two additional issues: (1) the limited memory on the coprocessor card, and (2) the benefits of minimizing communication to and from the card. It is worth noting as well, that the number of MPI ranks used on an Intel Xeon Phi coprocessor should be substantially less than the number of cores—in no small part because of limited memory on the coprocessor. Consistent with parallel programs in general, the advantages of overlapping communication (e.g., MPI messages or offload data movement) with computation are important to consider, as well as techniques to load balance work across all available cores. Of course, involving Intel Xeon processor cores and Intel Xeon Phi coprocessor cores adds the dimension of “big cores” and “little cores” to the balancing work, even though they share x86 instructions and programming models. While MPI programs often already tackle the overlap of communication and computation, the placement of ranks on coprocessor cores still requires dealing with the highly parallel programming needs and limited memory. This is why an offload model can be attractive, even within an MPI program.

The offload model for Intel Xeon Phi coprocessors is quite rich. The syntax and semantics of the Intel® Language Extensions for Offload are generally a superset of other offload models including OpenACC. This provides for greater interoperability with OpenMP; ability to manage multiple coprocessors (cards); and the ability to offload complex program components that an Intel Xeon Phi coprocessor can process, but that a GPU could not (and hence, OpenACC does not allow). We expect that a future version of OpenMP will include offload directives that provide support for these needs, and Intel plans to support such a standard for Intel Xeon Phi coprocessors.
as part of our commitment to providing OpenMP capabilities, Intel Language Extensions for Offload also provides for an implicit sharing model that is beyond what OpenMP will support. It rests on a shared memory model supported by Intel Xeon Phi coprocessors that allow a shared memory programming model (Intel calls this “MYO”) between Intel Xeon processors and Intel Xeon Phi coprocessors. This is most similar to partitioned global address space (PGAS) programming models; not an extension provided by OpenMP. The Intel “MYO” capability offers a global address space within the node, allowing sharing of virtual addresses for select data between processors and coprocessor on the same node. It is offered in C and C++, but not Fortran, since future support of coarray will be a standard solution to the same basic problem. Offloading is available as Fortran offloading via pragmas, C/C++ offloading with pragmas, and optionally shared (MYO) data. Use of MPI can also distribute applications across the system.

Summary: Transforming-and-Tuning
Double Advantage

Programming should not be called easy, and neither should parallel programming. However, we can work to keep the fundamentals the same: maximizing parallel computations and minimizing data movement. Parallel computations are enabled through scaling (more cores and threads) and vector processing (more data processed at once). Minimal data movement is an algorithmic endeavor, but can be eased through the higher bandwidth between memory and cores that is available with the Intel® Many Integrated Core (Intel® MIC) architecture used by Intel Xeon Phi coprocessors. This leads to parallel programming using the same programming languages and models across the Intel Xeon family of products, which are generally also shared across all general purpose processors in the industry. Languages such Fortran, C, and C++ are fully supported. Popular programming methods such as OpenMP, MPI, and Intel TBB are fully supported. Newer models with widespread support such as Coarray Fortran, Intel Cilk Plus, and OpenCL* can apply as well.

Tuning on Intel Xeon Phi coprocessors for scaling, and vector and memory usage, also benefits the application when run on Intel Xeon processors. Maintaining a value across the Intel Xeon family is critical, as it helps preserve past and future investments. Applications that initially fail to get maximum performance on Intel Xeon Phi coprocessors generally trace problems back to scaling, vector usage, or memory usage. When these issues are addressed, the improvements to the application usually have a related positive effect when run on Intel Xeon processors. This is the double advantage of “transforming-and-tuning,” and developers have found it to be among the most compelling features of the Intel Xeon Phi coprocessors.

Learn More

Additional material regarding programming for Intel Xeon Phi coprocessors can be found at http://intel.com/software/mic.

ispc: Intel® Xeon® and Intel® Xeon Phi™ support now
JAMES REINDERS, (Intel)
Director of Parallel Programming Evangelism

Vectorization is an industry-wide challenge—and if you are interested in seeing some one of the industry-leading exploration projects (and trying it on your code), then you may want to look at ispc.

ispc is an R&D compiler for a C-based language that is targeted to exploring the performance available from doing SPMD (single program, multiple data) computation on SIMD units found on CPUs and on Intel® Xeon Phi™ coprocessors (using the Intel® Many Integrated Core [MIC] architecture). It has delivered performance competitive with hand-coded SSE and AVX for a variety of graphics and throughput kernels, and typically delivers a 3x to 4x speedup vs. scalar C and C++ code on SSE and a 5 to 7x speedup on AVX (for computations that are amenable to SPMD implementation), while still providing the ease of use of a C-like language.

The paper “ispc: A SPMD Compiler for High-Performance CPU Programming” by Matt Pharr and Bill Marks won Best Paper Award at InPar 2012. It is an excellent paper that articulates the challenges of vectorization and explains the important context very well. It also advances a solid demonstration of what is possible when you think about SPMD on SIMD models clearly...

Learn More

Additional material regarding programming for Intel Xeon Phi coprocessors can be found at http://intel.com/software/mic.

ispc: Intel® Xeon® and Intel® Xeon Phi™ support now
JAMES REINDERS, (Intel)
Director of Parallel Programming Evangelism

Vectorization is an industry-wide challenge—and if you are interested in seeing some one of the industry-leading exploration projects (and trying it on your code), then you may want to look at ispc.

ispc is an R&D compiler for a C-based language that is targeted to exploring the performance available from doing SPMD (single program, multiple data) computation on SIMD units found on CPUs and on Intel® Xeon Phi™ coprocessors (using the Intel® Many Integrated Core [MIC] architecture). It has delivered performance competitive with hand-coded SSE and AVX for a variety of graphics and throughput kernels, and typically delivers a 3x to 4x speedup vs. scalar C and C++ code on SSE and a 5 to 7x speedup on AVX (for computations that are amenable to SPMD implementation), while still providing the ease of use of a C-like language.

The paper “ispc: A SPMD Compiler for High-Performance CPU Programming” by Matt Pharr and Bill Marks won Best Paper Award at InPar 2012. It is an excellent paper that articulates the challenges of vectorization and explains the important context very well. It also advances a solid demonstration of what is possible when you think about SPMD on SIMD models clearly...

Learn More

Additional material regarding programming for Intel Xeon Phi coprocessors can be found at http://intel.com/software/mic.

ispc: Intel® Xeon® and Intel® Xeon Phi™ support now
JAMES REINDERS, (Intel)
Director of Parallel Programming Evangelism

Vectorization is an industry-wide challenge—and if you are interested in seeing some one of the industry-leading exploration projects (and trying it on your code), then you may want to look at ispc.

ispc is an R&D compiler for a C-based language that is targeted to exploring the performance available from doing SPMD (single program, multiple data) computation on SIMD units found on CPUs and on Intel® Xeon Phi™ coprocessors (using the Intel® Many Integrated Core [MIC] architecture). It has delivered performance competitive with hand-coded SSE and AVX for a variety of graphics and throughput kernels, and typically delivers a 3x to 4x speedup vs. scalar C and C++ code on SSE and a 5 to 7x speedup on AVX (for computations that are amenable to SPMD implementation), while still providing the ease of use of a C-like language.

The paper “ispc: A SPMD Compiler for High-Performance CPU Programming” by Matt Pharr and Bill Marks won Best Paper Award at InPar 2012. It is an excellent paper that articulates the challenges of vectorization and explains the important context very well. It also advances a solid demonstration of what is possible when you think about SPMD on SIMD models clearly...

Learn More

Additional material regarding programming for Intel Xeon Phi coprocessors can be found at http://intel.com/software/mic.

ispc: Intel® Xeon® and Intel® Xeon Phi™ support now
JAMES REINDERS, (Intel)
Director of Parallel Programming Evangelism

Vectorization is an industry-wide challenge—and if you are interested in seeing some one of the industry-leading exploration projects (and trying it on your code), then you may want to look at ispc.

ispc is an R&D compiler for a C-based language that is targeted to exploring the performance available from doing SPMD (single program, multiple data) computation on SIMD units found on CPUs and on Intel® Xeon Phi™ coprocessors (using the Intel® Many Integrated Core [MIC] architecture). It has delivered performance competitive with hand-coded SSE and AVX for a variety of graphics and throughput kernels, and typically delivers a 3x to 4x speedup vs. scalar C and C++ code on SSE and a 5 to 7x speedup on AVX (for computations that are amenable to SPMD implementation), while still providing the ease of use of a C-like language.

The paper “ispc: A SPMD Compiler for High-Performance CPU Programming” by Matt Pharr and Bill Marks won Best Paper Award at InPar 2012. It is an excellent paper that articulates the challenges of vectorization and explains the important context very well. It also advances a solid demonstration of what is possible when you think about SPMD on SIMD models clearly...

Learn More

Additional material regarding programming for Intel Xeon Phi coprocessors can be found at http://intel.com/software/mic.

ispc: Intel® Xeon® and Intel® Xeon Phi™ support now
JAMES REINDERS, (Intel)
Director of Parallel Programming Evangelism

Vectorization is an industry-wide challenge—and if you are interested in seeing some one of the industry-leading exploration projects (and trying it on your code), then you may want to look at ispc.

ispc is an R&D compiler for a C-based language that is targeted to exploring the performance available from doing SPMD (single program, multiple data) computation on SIMD units found on CPUs and on Intel® Xeon Phi™ coprocessors (using the Intel® Many Integrated Core [MIC] architecture). It has delivered performance competitive with hand-coded SSE and AVX for a variety of graphics and throughput kernels, and typically delivers a 3x to 4x speedup vs. scalar C and C++ code on SSE and a 5 to 7x speedup on AVX (for computations that are amenable to SPMD implementation), while still providing the ease of use of a C-like language.

The paper “ispc: A SPMD Compiler for High-Performance CPU Programming” by Matt Pharr and Bill Marks won Best Paper Award at InPar 2012. It is an excellent paper that articulates the challenges of vectorization and explains the important context very well. It also advances a solid demonstration of what is possible when you think about SPMD on SIMD models clearly...

Learn More

Additional material regarding programming for Intel Xeon Phi coprocessors can be found at http://intel.com/software/mic.