Application Optimization for Intel® Processors and Coprocessors

Programming CodeBook
## CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>Intel® Processor and Coprocessor Overview</td>
<td>4</td>
</tr>
<tr>
<td>Compilation</td>
<td>5</td>
</tr>
<tr>
<td>Debugging</td>
<td>7</td>
</tr>
<tr>
<td>Math Processing</td>
<td>9</td>
</tr>
<tr>
<td>Performance Analysis</td>
<td>11</td>
</tr>
<tr>
<td>Clustering</td>
<td>13</td>
</tr>
<tr>
<td>Resources</td>
<td>16</td>
</tr>
</tbody>
</table>
As core counts and vector widths increase, programmers are challenged to take advantage of all the resulting processing power. This CodeBook explores some of the optimization techniques and models that can help developers create code that utilizes these capabilities for better performance. With Intel® software development tools, applications can scale from few cores to many cores with consistent models, languages, tools, and techniques—whether developing for Intel® Core™ or Intel® Xeon® processors, or the Intel® Xeon Phi™ coprocessor.

Understanding how to restructure code to expose more parallelism is critically important to enable the best performance on processors, GPUs, or coprocessors. For James Reinders, Intel’s director of parallel programming evangelism, the path to successful parallel programming is simple and achievable: “program with lots of threads that use vectors, while using preferred programming languages and parallelism models.” This restructuring itself will generally yield benefits. Reinders emphasizes the value of common programming languages, models, and tools that work across processors and coprocessors—so developers do not have to rewrite code in order to scale their applications.

Here, we’ll take a brief look at some of the possibilities when using Intel processors and coprocessors. Coprocessors extend the scaling capabilities of Intel Xeon processor-based systems by exploiting additional processor vector capabilities or memory bandwidth.
INTEL® PROCESSOR AND COPROCESSOR OVERVIEW

Programs that utilize multicore processors and many-core coprocessors have a wide variety of options to meet varying needs. These options fully utilize existing widely adopted solutions, such as C, C++, Fortran, OpenMP®, MPI, and Intel® Threading Building Blocks (Intel® TBB). They are rapidly driving the development of additional emerging standards such as OpenCL®, as well as open entrants such as Intel® Cilk™ Plus.

Each Intel Xeon processor and Intel Xeon Phi coprocessor core is a fully functional, multithread execution unit. Intel Xeon Phi supports a variety of programming models:

- Run as an accelerator for offloaded host computation
- Run as a native or MPI* compute node via IP or OFED
- Run in Symmetric Mode using MPI*
- Run from a Windows* host machine

Extend capabilities with key features

- Scalar unit based on Intel® Pentium® processor family
- 64-bit data path
- 4 hardware threads per core
- 32 512-bit vector registers per context
- Fully pipelined instruction decoder
- Increases resource utilization
- Faster hosting on serial code

Use familiar Intel development environment

- Parallel programming models
- OpenMP® 4.0
- Intel® Composer: C, C++, and Fortran Compilers
- Intel® Threading Building Blocks (Intel® TBB)
- Intel Cilk™ Plus
- Intel® MPI Library support for Intel Xeon Phi
- Intel support for GDB on Intel Xeon Phi
- Intel® Performance Libraries (e.g. Intel® Math Kernel Library)
- Intel® VTune™ Amplifier XE for performance analysis
- Standard runtime libraries, including pthreads

Explore

Introduction to the Intel® Xeon Phi™ Coprocessor (PDF) »
Comprehensive resources for manycore developers »
Intel® Xeon Phi™ Coprocessor Quick Start Guide (PDF) »

Tips

Intel® Xeon® Processors
Most commonly used parallel processor
- Parallel processor supports a wide variety of applications
- Provides accelerated parallel and serial performance
- Offers industry-leading performance per core

Intel® Xeon Phi™ Processors
Optimized for highly parallel applications
- Use for applications that are highly parallel and can benefit from wide vectors
- Based on Intel® Many Integrated Core Architecture (Intel® MIC) (up to 61 cores)
- Utilizes standards-based programming languages and models

For in-depth coverage, view the webinar: Intel® Xeon® Processors and Xeon® Phi™ Coprocessors—Introduction to High Performance Application Development for Multicore and Manycore »
Programming and optimizing for the Intel® Xeon Phi™ Coprocessor is a simple extension of programming and optimizing for the host processor. Intel® C, C++, and Fortran compilers support the same optimization features for the Intel® Xeon Phi™ coprocessor as for the host processor, including:

- OpenMP® 4.0 for both native and offload models
- Intel® Cilk™ Plus array notation
- SIMD optimization
- Vectorization
- Expanded C++11 support
- Expanded Fortran 2003 and 2008 support
- Improved MPI performance and scalability
- Loop optimizations, optimized math libraries, IPO

**Example of Intel® Cilk™ Plus Array Notation**

```c
void addit(double* a, double* b, int m, int n, int x)
{
    for (int i = m; i < m+n; i++)
        a[i] = b[i] + a[i-x];
}
```

Loop was not vectorized: existence of vector dependence.

```c
void addit(double* a, double* b, int m, int n, int x)
{
    // I know x<0
    a[m:n] = b[m:n] + a[m-x:n];
}
```

LOOP WAS VECTORIZED.

**Example of Intel® Cilk™ Plus Pragma**

```c
void addit(double* a, double* b, int m, int n, int x)
{
    for (int i = m; i < m+n; i++)
        a[i] = b[i] + a[i-x];
}
```

Loop was not vectorized: existence of vector dependence.

```c
void addit(double* a, double* b, int m, int n, int x)
{
    #pragma simd    // I know x<0
    for (int i = m; i < m+n; i++)
        a[i] = b[i] + a[i-x];
}
```

SIMD LOOP WAS VECTORIZED.
The Importance of SIMD Parallelism is Increasing

For good performance, it’s not sufficient to use all the cores. Utilizing 512-bit SIMD registers and instructions are also necessary. Because of the greater SIMD width, vectorization is even more important on Intel® Many Integrated Core (Intel® MIC) architecture than on Intel® Xeon® processors. (There is no need to re-optimize vectorizable code for new processors.) The compiler can be used to vectorize, or there are libraries that are already vectorized, such as Intel® Math Kernel Library.

For in-depth coverage, view the webinar: Optimizing and Compilation for Intel® Xeon Phi™ Coprocessor »

Leading Compiler Performance

Industry Leading Performance using the Intel® C/C++ and Fortran Compilers

(Higher is Better)

<table>
<thead>
<tr>
<th>Windows*</th>
<th>Linux*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® C++ Compiler 14.0</td>
<td>Intel® C++ Compiler 14.0</td>
</tr>
<tr>
<td>for Windows</td>
<td>for Linux</td>
</tr>
<tr>
<td>1.66</td>
<td>1.32</td>
</tr>
<tr>
<td>Microsoft</td>
<td>GCC 4.8.1</td>
</tr>
<tr>
<td>Visual C++ 2012</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>Estimates SPECint® base2006 integer benchmark</td>
<td>Estimates SPECint® base2006 floating point benchmark</td>
</tr>
</tbody>
</table>

C++ Floating

<table>
<thead>
<tr>
<th>Windows*</th>
<th>Linux*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® C++ Compiler 14.0</td>
<td>Intel® C++ Compiler 14.0</td>
</tr>
<tr>
<td>for Windows</td>
<td>for Linux</td>
</tr>
<tr>
<td>1.85</td>
<td>2.45</td>
</tr>
<tr>
<td>Microsoft</td>
<td>GCC 4.8.1</td>
</tr>
<tr>
<td>Visual C++ 2012</td>
<td>1.0</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>Estimates SPECint® base2006 integer benchmark</td>
<td>Estimates SPECint® base2006 floating point benchmark</td>
</tr>
</tbody>
</table>

Fortran

<table>
<thead>
<tr>
<th>Windows*</th>
<th>Linux*</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel® C++ Compiler 14.0</td>
<td>Intel® C++ Compiler 14.0</td>
</tr>
<tr>
<td>for Windows</td>
<td>GCC 4.8.1</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
</tr>
<tr>
<td>Estimates SPECint® base2006 integer benchmark</td>
<td>Estimates SPECint® base2006 floating point benchmark</td>
</tr>
</tbody>
</table>

For more information regarding performance and optimization choices in Intel® software products, visit: http://software.intel.com/en-us/articles/optimization-notice
DEBUGGING

GDB, the GNU Project debugger, allows you to see what is going on ‘inside’ another program while it executes—or what another program was doing at the moment it crashed. This Intel® debug solution supports both Intel® processors and coprocessors, including the Intel® Xeon Phi™ coprocessor. The Intel implementation includes:

- Support for multiple debug models
- Native/remote target debugging
- Eclipse® GUI integration
- Aware of extended programming models
- Explicit offload model and SVM model
- C, C++, and Fortran support
- Parallel Debug Extensions (PDBX)

GDB is available with the Intel® Manycore Platform Software Stack (Intel® MPSS) and Intel® Parallel Studio XE.

For in-depth coverage, view the webinar: GNU Debugger Intel® Xeon Phi™ Coprocessor »

Intel® Xeon Phi™ Coprocessor Architecture Features

```
List all new vector and mask registers
(gdb) info registers zmm
R0  0x0  0
Zmm31 {v16_float = {0x0 <repeats 16 times>},
v3_double = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
v64_int8 = {0x0 <repeats 64 times>},
v32_int16 = {0x0 <repeats 32 times>},
v16_int32 = {0x0 <repeats 16 times>},
v8_int64 = {0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0},
v4_int128 = {0x0, 0x0, 0x0, 0x0}}

Disassemble instructions
(gdb) disassemble $pc, +10
Dump of assembler code from 0x11 to 0x24:
0x0000000000000011 <foobar+17>:
vpackstoreps %zmm0,-0x10(%rbp)(&k1)
0x0000000000000018 <foobar+24>:
vbroadcastss -0x10(%rbp),%zmm0
```
Native/Manycore-Hosted Debugging

Run GDB* on the Intel® Xeon Phi™ Coprocessor

```
ssh -t mic0 /usr/bin/gdb
```

To attach to a running application via the process-id

```
(gdb) shell pidof my_application
42
(gdb) attach 42
```

To run an application directly from GDB*

```
(gdb) file /target/path/to/application
(gdb) start
```

Eclipse® IDE integration

Simultaneous and seamless thread debugging

- Seamless debugging of host and coprocessor
- Simultaneous view of host and coprocessor threads
- Supports offload language extensions (auto-attach to offload process)
- Supports multiple coprocessor cards
- Supports C, C++, and Fortran
MATH PROCESSING

Intel® Math Kernel Library (Intel® MKL) is the industry’s leading math library—with support for Intel® Xeon® processors and Intel® Xeon Phi™ coprocessors. Intel Xeon Phi coprocessor support has been extended, including:

- Heterogeneous computing
- Takes advantage of both multicore host and manycore coprocessors.
- Optimized for wider (512-bit) SIMD instructions and threaded for many cores
- Highly optimized functions

Sample Usage Models on Intel® Xeon Phi™ Coprocessors

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic Offload (AO)</td>
<td>• Offloading is automatic and transparent&lt;br&gt;• Takes advantage of multiple coprocessors&lt;br&gt;• By default, Intel® MKL determines when to offload: work division between host and coprocessors&lt;br&gt;• Automatic host and target parallelism&lt;br&gt;• Users can still specify work division between host and target (for BLAS only)</td>
</tr>
<tr>
<td>Compiler Assisted Offload (CAO)</td>
<td>• Offloading is explicitly controlled by compiler pragmas or directives&lt;br&gt;• Exploit the full potential of compiler’s offloading facility&lt;br&gt;• More flexibility in data transfer and remote execution management&lt;br&gt;• Advantage of data persistence: reusing transferred data for multiple operations</td>
</tr>
<tr>
<td>Native Execution</td>
<td>• Use the coprocessor as an independent compute node&lt;br&gt;• Programs can be built to run only on the coprocessor by using the -mmic build option&lt;br&gt;• Intel® MKL function calls inside an offloaded code region execute natively&lt;br&gt;• Get better performance when input data are already available on the coprocessor, and output is not immediately needed on the host side</td>
</tr>
</tbody>
</table>

For in-depth coverage, view the webinar: Get Ready for Intel® Math Kernel Library on Intel® Xeon Phi™ Coprocessor »

Explore

Resources for Manycore Developers »
Faster Math Performance with Intel® Math Kernel Library (video) »
Automatic Offload with Intel® Math Kernel Library (video) »
Intel® Math Kernel Library forum »
Intel® Math Kernel Library Link Options Wizard »
User’s Guide »
Performance charts »
LU factorization using Intel MKL 11.1 can reach ~440 GFlops on an Ivy Bridge-EP system with 24 cores.

When running on the Intel Xeon Phi coprocessor, native execution (without data transfer) can double the performance, if the matrix is big enough (M, N > 25K). This will require developers to specifically build the code for native execution and to have input data on the coprocessor.

Using automatic offload with one Intel Xeon Phi coprocessor—without the need of user code change and link line change—LU can achieve the same performance as native execution. This is because Intel MKL simultaneously uses compute resources on both the host CPU and the coprocessor. Data transfer, synchronization, and execution management are automatically taken care of by the Intel MKL runtime and are transparent to users.

Using automatic offload with two Intel Xeon Phi coprocessors can further boost the LU performance by up to 1.5x compared to using automatic offload with one coprocessor, or up to 3x compared to the performance on a 24-core Ivy Bridge-EP CPU.
Performance Analysis

Vectorization, parallelism, and data locality are critical to good performance for the Intel® Xeon® processor and Intel® Xeon Phi™ coprocessor. Intel® VTune™ Amplifier XE can help with performance analysis and tuning for optimization, for example by collecting essential metrics and providing special analysis types, such as General Exploration and Memory Bandwidth.

**Intel® VTune™ Amplifier XE: Visualize Performance**

New capabilities are available to support the Intel Xeon Phi coprocessor, including:

- Tune applications for scalable multicore performance
- Fast, accurate performance profiles
- Hotspot (statistical call tree)
- Hardware event-based sampling
- Thread profiling: visualize thread interactions on timeline
- Microsoft*, GCC*, Intel compilers
- C, C++, Fortran, Assembly, .NET*
- Balance workloads
- Use a normal production build

For in-depth coverage, view the webinar: Performance analysis on Intel® Xeon Phi™ Coprocessor webinar »
Start with host-based profiling to identify vectorization, parallelism, and offload candidates.

Use Intel® Inspector XE on code with offload disabled (on host) to identify correctness errors (e.g., deadlocks, races).

Project Properties provides the means to invoke data collection by target type.

Launch Application serves many uses, from host/offload to native execution.

Search directories have been reorganized to speed symbol resolution during finalization.

General Exploration runs a set of events to drive top-down analysis.

For event collection the coprocessor is treated as a special HW architecture.
CLUSTERING

Intel is the leading vendor of Message Passing Interface (MPI) implementations and tools. The ease of use of the Intel® MPI Library and related tools, such as Intel® Trace Analyzer and Collector, extends from the Intel® Xeon® architecture to the Intel® Many Integrated Core (Intel® MIC) architecture. With MPI, Intel® Xeon Phi™ Coprocessor acts as a network node, providing:

- Optimized MPI application performance
- Cluster and hybrid computing
- Thread parallelism
- Load balancing
- Automatic and application-specific tuning
- Industry leading low latency
- Interconnect independence and runtime selection

For in-depth coverage, view the webinar: Message Passing Interface (MPI) on Intel® Xeon Phi™ Coprocessor webinar »

Intel® MPI library eliminates the need to develop, maintain, and test applications running on multiple fabrics.

Applications

CFD | Crash | Climate | QCD | BIO | Other...

Develop applications for one fabric

Intel® MPI Library

Select interconnect fabric at runtime

TCP/IP | Myrinet | Infini-band | I/WARP | Shared Memory | ...Other Networks

Fabrics

Cluster

For more information regarding performance and optimization choices in Intel® software products, visit: http://software.intel.com/en-us/articles/optimization-notice
Privacy | Terms of Use | Learn More
Industry Leading Performance with Intel® Library 4.1
Relative (Geomean) MPI Latency Benchmarks on Linux® 64 (Higher is Better)
1024 Processes on 64 nodes (InfiniBand + shared memory)

![Graph showing speedup times for different MPI communication models.]

Configuration Info: -Sv Versions: Intel® C++ version 13.1.1, Intel® MPI Library 4.1.1, MVAPICH2 1.9, OpenMPI 1.7.1, Intel® MPI Benchmarks 3.2.4, Hardware: Intel® Xeon® CPU E5-2670 @ 2.60GHz, RAM 64GB, Interconnect: InfiniBand, ConnectX adapters, ODR: Operating System: SLES 11.1; Notes: 1024 Processes on 64 nodes (InfiniBand + shared memory). All listed MPI libraries were built with the Intel® C++ Compiler 12.1 Update 10 for Linux®.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations, and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. * Other brands and names are the property of their respective owners. Benchmark Source: Intel Corporation.

Optimization Notice: Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations are not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice. Notice revision #20110804

**MPI Programming Models**

<table>
<thead>
<tr>
<th>Coprocessor Only</th>
<th>Symmetric</th>
<th>MPI and Offload</th>
</tr>
</thead>
<tbody>
<tr>
<td>• MPI ranks on Intel® Xeon Phi™ coprocessor (only)</td>
<td>• MPI ranks on Intel® Xeon Phi™ coprocessors and host CPUs</td>
<td>• MPI ranks on Intel® Xeon® processors (only)</td>
</tr>
<tr>
<td>• All messages into/out of the coprocessors</td>
<td>• Messages to/from any core</td>
<td>• All messages into/out of host CPUs</td>
</tr>
<tr>
<td>• Intel® Cilk™ Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads* used directly within MPI processes</td>
<td>• Intel® Cilk™ Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads* used directly within MPI processes</td>
<td>• Offload models used to accelerate MPI ranks</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Intel® Cilk™ Plus, OpenMP*, Intel® Threading Building Blocks, Pthreads* within Intel® Xeon Phi™ coprocessor</td>
</tr>
</tbody>
</table>

**Intel® Trace Analyzer and Collector with Intel® Xeon Phi™**

Run with `-trace` flag (without linkage) to create a trace file

- MPI+Offload
  # mpirun -trace -n 2 ./test
- Coprocessor only and Symmetric
  # export I_MPI_MIC=enable
  # mpirun -trace -f mpi_hosts -n 2 ~/test_hello[.MIC]

Flag `-trace` will implicitly pre-load the libVT.so (which finally calls libmpi.so to execute the MPI call)

---

**Explore**

- Resources for Manycore Developers »
- Intel® MPI Library »
- Intel® Trace Analyzer and Collector »
- Distributed Computing with Intel® MPI Library (video) »
- Analyzing and Balancing MPI applications (video) »
- Intel® Cluster and HPC technology forums »
- Intel® Xeon Phi™ coprocessor developer community »
Optimization Notice

Intel’s compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

© 2013, Intel Corporation. All rights reserved. Intel, the Intel logo, Cilk Plus, Core, VTune, Xeon, and Xeon Phi are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.