Primer on Computing with Intel Xeon Phi Coprocessors

SLAC Geant4 Tutorial 2014

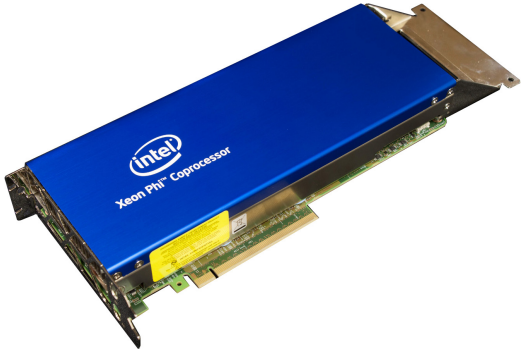
Andrey Vladimirov
Colfax International

March 6, 2014

MIC Architecture from the Programmer's Perspective

Intel Xeon Phi Coprocessors and the MIC Architecture

- PCIe end-point device
- High Power efficiency
- ~ 1 TFLOP/s in DP
- Heterogeneous clustering

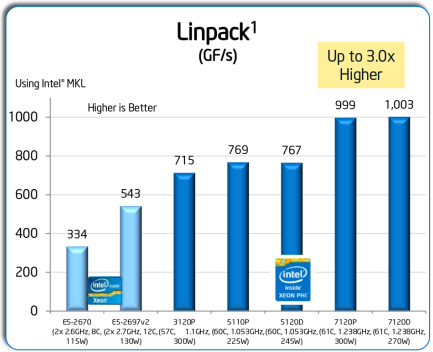


For highly parallel applications which reach the scaling limits on Intel Xeon processors

Xeon Family Product Performance

Many-core Coprocessors (Xeon Phi) vs Multi-core Processors (Xeon) —

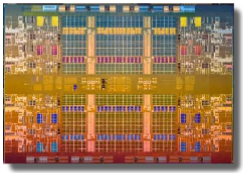
- Better performance per system & performance per watt for parallel applications
- Same programming methods, same development tools.



1. Xeon ran MP Linpack, Xeon Phi ran SMP Linpack. Expected performance difference between the two is estimated in the 3-5% range

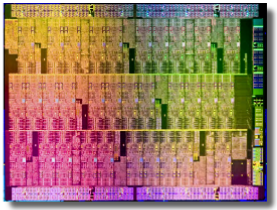
Source: "Intel Xeon Product Family: Performance Brief"

Intel Xeon Phi Coprocessors and the MIC Architecture



Multi-core Intel Xeon processor

- C/C++/Fortran; OpenMP/MPI
- Standard Linux OS
- Up to 768 GB of DDR3 RAM
- ≤12 cores/socket ≈3 GHz
- 2-way hyper-threading
- 256-bit AVX vectors



Many-core Intel Xeon Phi coprocessor

- C/C++/Fortran; OpenMP/MPI
- Special Linux μOS distribution
- 6–16 GB cached GDDR5 RAM
- 57 to 61 cores at ≈1 GHz
- 4-way hyper-threading
- 512-bit IMCI vectors

Examples of Solutions with the Intel MIC Architecture

Colfax's CXP7450 workstation with two Intel Xeon Phi coprocessors

Colfax's CXP9000 server with eight Intel Xeon Phi coprocessors

Linux μOS on Intel Xeon Phi coprocessors (part of MPSS)

```
user@host% lspci | grep -i "co-processor"
06:00.0 Co-processor: Intel Corporation Device 2250 (rev 11)
82:00.0 Co-processor: Intel Corporation Device 2250 (rev 11)
user@host% sudo service mpss status
mpss is running
user@host% cat /etc/hosts | grep mic
172.31.1.1  host-mic0 mic0
172.31.2.1  host-mic1 mic1
user@host% ssh mic0
user@mic0% cat /proc/cpuinfo | grep proc | tail -n 3
processor : 237
processor : 238
processor : 239
user@mic0% ls /
amplxe   dev    home   lib64    oldroot  proc   sbin     sys    usr
bin      etc    lib    linuxrc  opt      root   sep3.10  tmp    var
```

Programming Models and Application Porting

Native Execution

"Hello World" application:

```
1 #include <stdio.h>
2 #include <unistd.h>
3 int main(){
4   printf("Hello world! I have %ld logical cores.\n",
5   sysconf(_SC_NPROCESSORS_ONLN ));
6 }
```

Compile and run on host:

```
user@host% icc hello.c
user@host% ./a.out
Hello world! I have 32 logical cores.
user@host%
```

Native Execution

Compile and run the same code on the coprocessor in the native mode:

```
user@host% icc hello.c -mmic
user@host% scp a.out mic0:~/
a.out 100% 10KB 10.4KB/s 00:00
user@host% ssh mic0
user@mic0% pwd
/home/user
user@mic0% ls
a.out
user@mic0% ./a.out
Hello world! I have 240 logical cores.
user@mic0%
```

- Use `-mmic` to produce executable for MIC architecture
- Must transfer executable to coprocessor (or NFS-share) and run from shell
- Native MPI applications work the same way (need Intel MPI library)

Porting User Applications for Native Execution

Simple CPU applications can be compiled for native execution on Xeon Phi coprocessors by supplying the flag "-mmic" to the Intel compiler:

```
user@host% icpc -c myobject1.cc -mmic
user@host% icpc -c myobject2.cc -mmic
user@host% icpc -o myapplication myobject1.o myobject2.o -mmic
```

Same for coprocessor-only MPI applications:

```
user@host% mpiicpc -c myobject1.cc -mmic
user@host% mpiicpc -c myobject2.cc -mmic
user@host% mpiicpc -o myapplication myobject1.o myobject2.o -mmic
```

Native Applications with Autotools

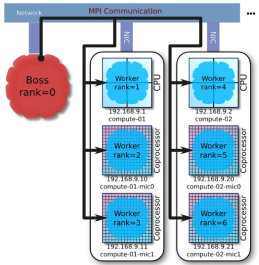
- Use the Intel compiler with flag `-mmic`
- Eliminate assembly and unncecessary dependencies
- Use `--host=x86_64` to avoid “program does not run” errors

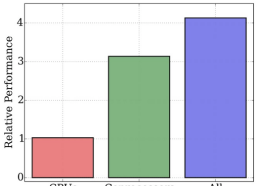
Example, the GNU Multiple Precision Arithmetic Library (GMP):

```
user@host% wget https://ftp.gnu.org/gnu/gmp/gmp-5.1.3.tar.bz2
user@host% tar -xf gmp-5.1.3.tar.bz2
user@host% cd gmp-5.1.3
user@host% ./configure CC=icc CFLAGS="-mmic"  --disable-assembly --host=x86_64
...
user@host% make
...
```

Heterogeneous Clustering with Homogeneous Code: Asian Option Pricing

- Monte Carlo method
- MPI + OpenMP + automatic vectorization
- The same C code for clusters of
  a) CPUs
  b) Coprocessors
  c) CPUs+Coprocessors (heterogeneous)
- **No code modification** to run on the Intel MIC architecture
- No platform-specific tunning
- Bridged network configuration





More information in white paper on research.colfaxinternational.com, including a video.

Explicit Offload Programming Model

"Hello World" in the explicit offload model:

```
1 #include <stdio.h>
2 int main(int argc, char * argv[] ) {
3   printf("Hello World from host!\n");
4 #pragma offload target(mic)
5     {
6       printf("Hello World from coprocessor!\n"); fflush(0);
7     }
8   printf("Bye\n");
9 }
```

Application launches and runs on the host, but some parts of code and data are moved ("offloaded") the coprocessor.

Compiling and Running an Offload Application

```
user@host% icpc hello_offload.cpp -o hello_offload
user@host% ./hello_offload
Hello World from host!
Bye
Hello World from coprocessor!
```

- Code inside of `#pragma offload` is offloaded automatically
- Console output on Intel Xeon Phi coprocessor is buffered and mirrored to the host console
- If coprocessor is not installed, code inside `#pragma offload` falls back to the host system

Offloading Functions and Data

```
1  int* __attribute__((target(mic))) data;
2
3  __attribute__((target(mic))) void MyFunction(int* foo) {
4    // ... implement function as usual
5  }
6
7  int main(int argc, char * argv[] ) {
8      // ...
9  #pragma offload target(mic) inout(data : length(N))
10     {
11       MyFunction(data);
12     }
13 }
```

- Functions and data used on coprocessor must be marked with the specifier `__attribute__((target(mic)))`

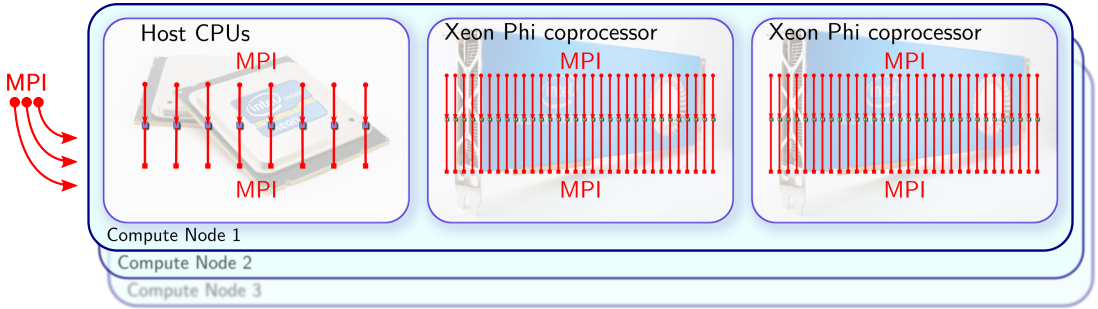
Virtual-shared Memory Model

```
1  _Cilk_shared int arr[N]; // This is a virtual-shared array
2
3  _Cilk_shared void Compute() { // This function may be offloaded
4    // ... function uses array arr[]
5  }
6
7  int main() {
8    // arr[] can be initialized on the host
9    _Cilk_offload Compute(); // and used on coprocessor
10   // and the values are returned to the host
11 }
```

- Alternative to Explicit Offload
- Data synced from host to coprocessor before the start of offload
- Data synced from coprocessor to host at the end of offload

Heterogeneous Computing with the MIC Architecture

Heterogeneous Distributed Computing with Xeon Phi

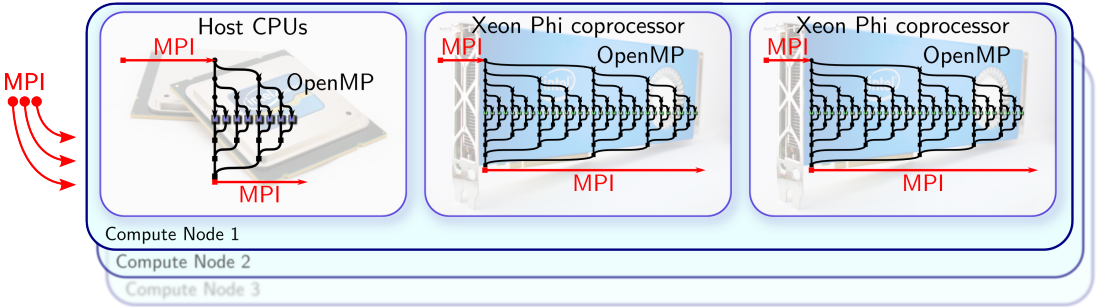


Option 1: Symmetric Pure MPI.

- MPI processes are single-threaded.
- Native MPI processes on the coprocessor.

E.g., 32 MPI processes on each CPU, 240 on each coprocessor.

Heterogeneous Distributed Computing with Xeon Phi

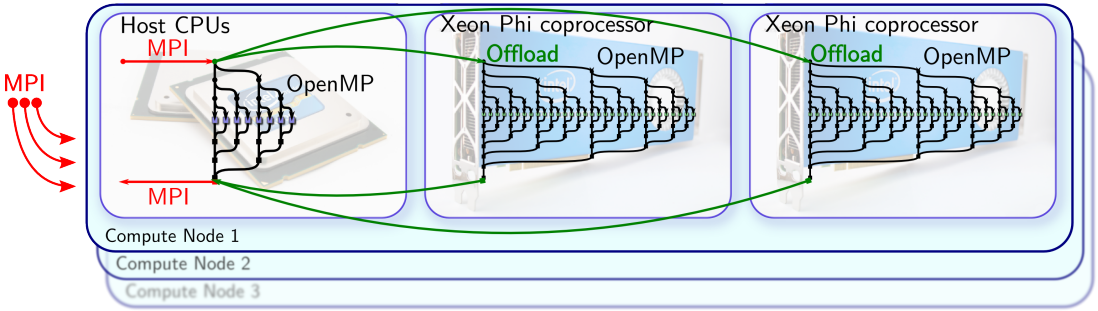


Option 2: Symmetric Hybrid MPI+OpenMP.

- MPI processes are multi-threaded with OpenMP.
- Native MPI processes on the coprocessor.

E.g., one 32-thr MPI proc on each CPU, 240-thr on each coprocessor.

Heterogeneous Distributed Computing with Xeon Phi



Option 3: Hybrid MPI+OpenMP with Offload.

- MPI processes are multi-threaded with OpenMP.
- MPI processes run only on CPUs.
- One or more OpenMP threads perform offload to coprocessor(s).

Optimization for Intel Xeon Family Products

One Size Does Not Fit All

An application must reach scalability limits on the CPU in order to benefit from the MIC architecture.

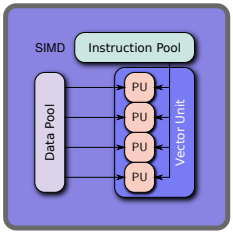
Use Xeon Phi if:

- Scales up to 100 threads
- Compute bound & vectorized, or bandwidth-bound

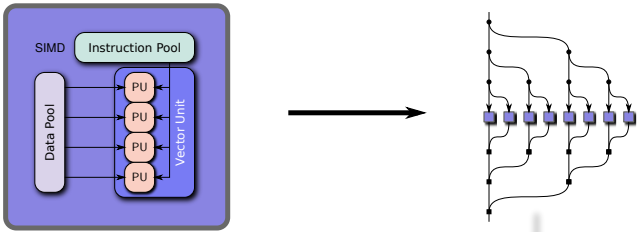
Use Xeon if:

- Serial or scales to $\lesssim$10 threads
- Unvectorized or latency-bound

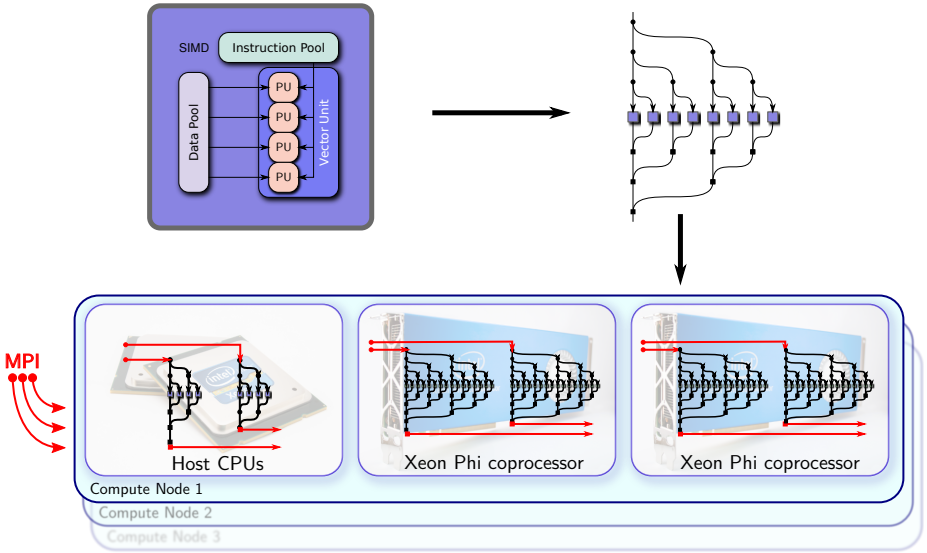
Three Layers of Parallelism



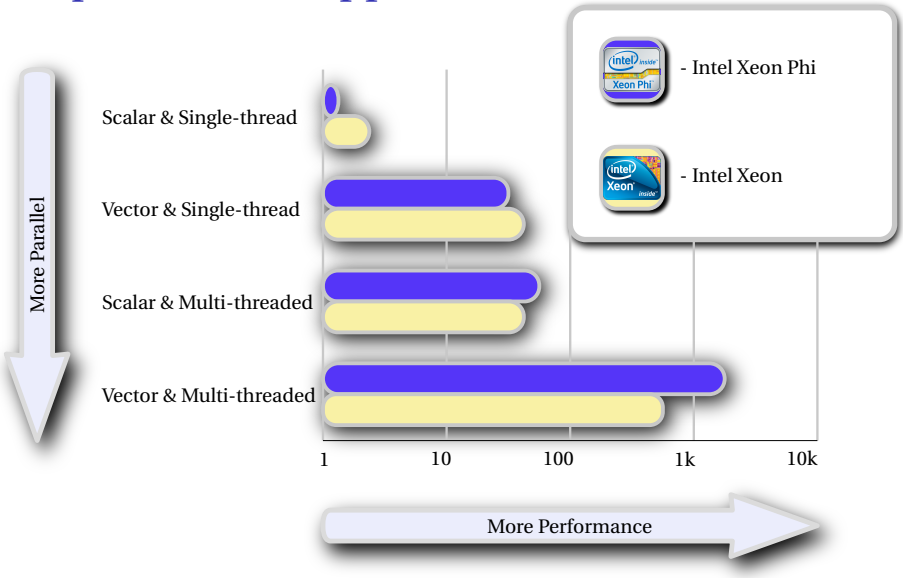
Three Layers of Parallelism



Three Layers of Parallelism

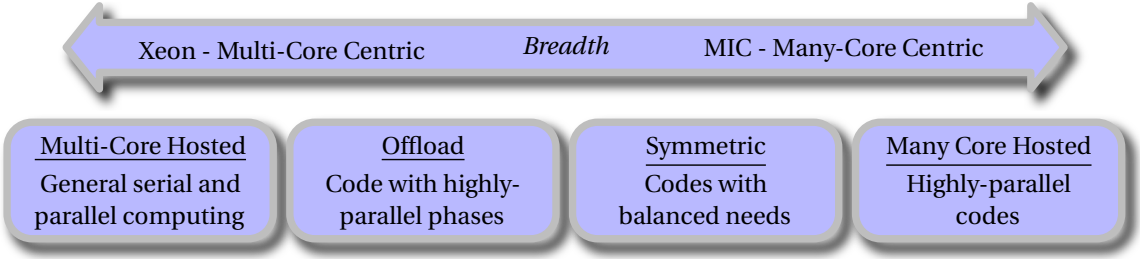


Compute-Bound Application Performance



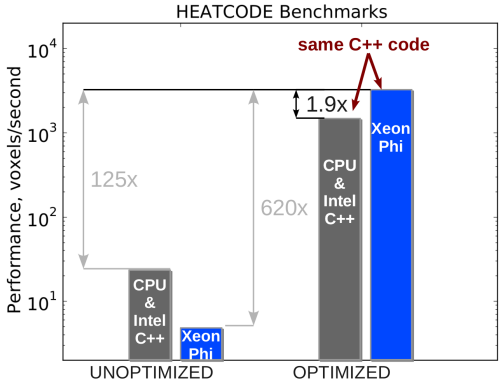
Xeon + Xeon Phi Coprocessors = Xeon Family

Programming models allow a range of CPU+MIC coupling modes



Performance Expectations: “Two Birds with One Stone”

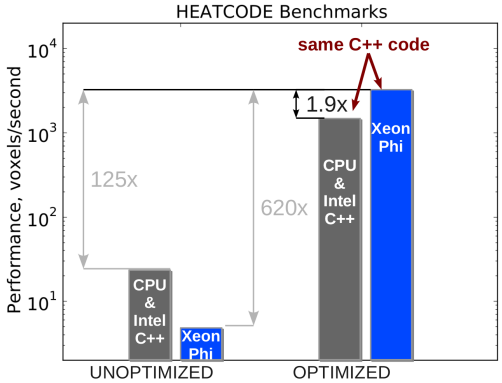
- Performance will be disappointing if code is not optimized for multi-core CPUs
- Optimized code runs better on the MIC platform *and* on the multi-core CPU
- Single code for two platforms + Ease of porting = Incremental optimization



More information in case study on research.colfaxinternational.com

Caution on Comparative Benchmarks

- In most of our benchmarks, "Xeon Phi" = 5110P SKU (60 cores, TDP 225 W, $2.7k), "CPU" = dual Xeon E5-2680 (16 cores, TDP 260 W, $3.4k + system cost)
- Why dual CPU vs single coprocessor? Approximately the same Thermal Design Power (TDP) and cost.



More information in case study on research.colfaxinternational.com

Optimization Checklist

1. Scalar optimization
2. Vectorization
3. Scale above 100 threads
4. Arithmetically intensive or bandwidth-limited
5. Efficient cooperation between the host and the coprocessor(s)

Optimization Example: In-Place Square Matrix Transposition

```
1 #pragma omp parallel for
2   for (int i = 0; i < n; i++) { // Distribute across threads
3     for (int j = 0; j < i; j++) { // Employ vector load/stores
4       const double c = A[i*n + j]; // Swap elements
5       A[i*n + j] = A[j*n + i];
6       A[j*n + i] = c;
7     }
8   }
```

Unoptimized code:

- Large-stride memory accesses
- Inefficient cache use
- Does not reach memory bandwidth limit

Tiling a Parallel For-Loop (Matrix Transposition)

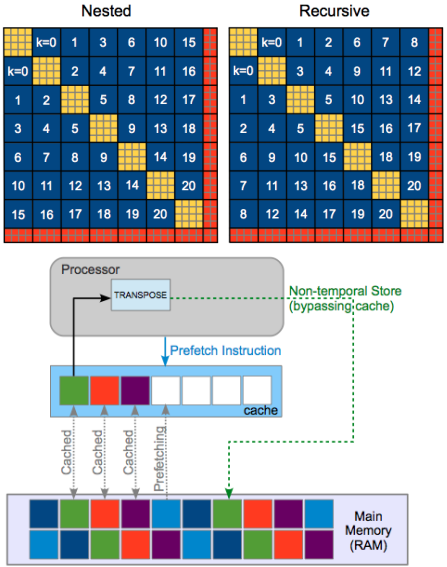
```
1  #pragma omp parallel for
2    for (int ii = 0; ii < n; ii += TILE) { // Distribute across threads
3      const int iMax = (n < ii+TILE ? n : ii+TILE); // Adapt to matrix shape
4      for (int jj = 0; jj <= ii; jj += TILE)  { // Tile the work
5        for (int i = ii; i < iMax; i++) { // Universal microkernel
6          const int jMax = (i < jj+TILE ? i : jj+TILE); // for whole matrix
7  #pragma loop count avg(TILE) // Vectorization tuning
8  #pragma simd // Vectorization hint
9          for (int j = jj; j<jMax; j++) { // Variable loop count (bad)
10           const double c = A[i*n + j]; // Swap elements
11           A[i*n + j] = A[j*n + i];
12           A[j*n + i] = c;
13     } } } }
```

Better (but not optimal) solution:

- Loop tiling to improve locality of data access
- Not enough outer loop iterations to keep 240 threads busy

Further Optimization of Matrix Transposition

- Multi-versioned inner loop for diagonal, edges and body
- Tuning pragma to enforce non-temporal stores
- Expand parallel iteration space occupy all threads
- Control data alignment
- OpenMP thread affinity for bandwidth optimization

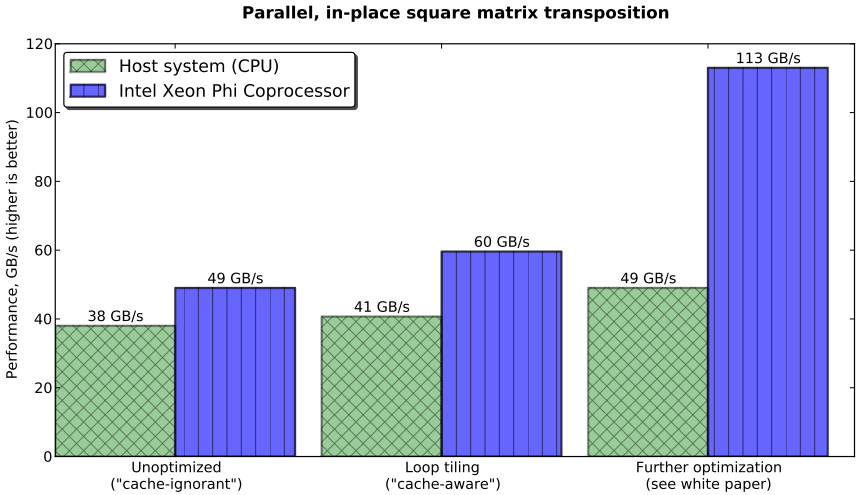


Further Optimization: Code Snippet

```
#pragma omp parallel
  {
#pragma omp for schedule(guided)
    for (int k = 0; k < nTilesParallel; k++) { // Bulk of calculations here
      const int ii = plan[HEADER_OFFSET + 2*k + 0]*TILE; // Planned order
      const int jj = plan[HEADER_OFFSET + 2*k + 1]*TILE; // of operations
      for (int j = jj; j < jj+TILE; j++) { // Simplified main microkernel
#pragma simd // Vectorization hint
#pragma vector nontemporal // Cache traffic hint
        for (int i = ii; i < ii+TILE; i++) { // Constant loop count (good)
          const double c = A[i*n + j]; // Swap elements
          A[i*n + j] = A[j*n + i];
          A[j*n + i] = c;
        } } }
    // Transposing the tiles along the main diagonal and edges...
    // ...
```

- Longer code but still in the C language; works for CPU and MIC

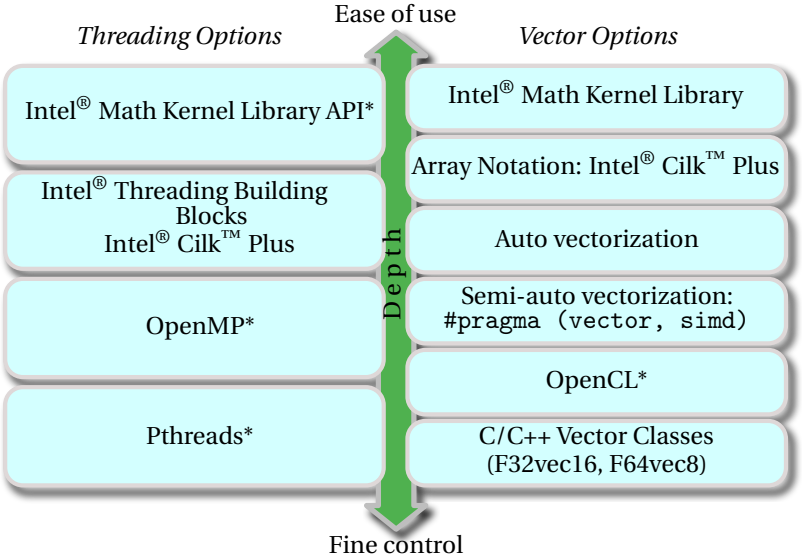
In-Place Square Matrix Transposition: White Paper



More information in the white paper on research.colfaxinternational.com

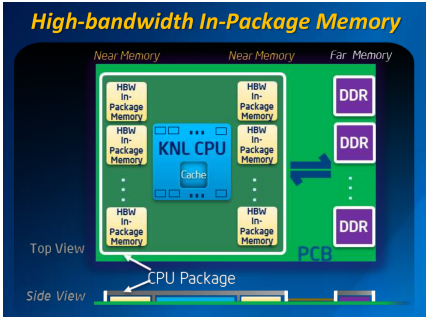
Future-Proofing: Reliance on Compiler and Libraries

Future-Proofing: Reliance on Compiler and Libraries



Next Generation MIC: Knights Landing (KNL)

- 2nd generation MIC product: code name Knights Landing (KNL)
- Intel's 14 nm manufacturing process
- A processor (running the OS) or a coprocessor (PCIe device)
- On-package high-bandwidth memory w/flexible memory models: flat, cache, & hybrid
- Intel Advanced Vector Extensions AVX-512 (public)



Source: Intel Newsroom

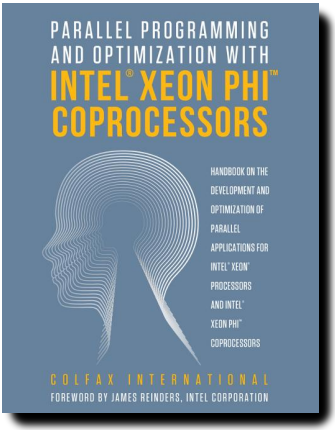
Getting Ready for the Future

- Porting HPC applications to today's MIC architecture makes them ready for future architectures, such as KNL
- Xeon, KNC and KNL are not binary compatible, therefore assembly-level tuning will not scale forward.
- Reliance on compiler optimization and using optimized libraries (such as Intel MKL) ensures future-readiness.

Source: Intel Newsroom

Colfax Developer Training

Colfax runs a one-day and four-day trainings for organizations on parallel programming with Intel Xeon Phi co-processors.



http://www.colfax-intl.com/nd/xeonphi/training.aspx

Thank you for tuning in,
and
have a wonderful journey
to the Parallel World!

research.colfaxinternational.com

P.S.: We are hiring! http://www.colfax-intl.com/nd/Jobs.aspx

Resources/Backup Slides

Reference Guides

- Intel C++ Compiler 14.0 User and Reference Guide
- Intel VTune Amplifier XE User's Guide
- Intel Trace Analyzer and Collector Reference Gude
- Intel MPI Library for Linux* OS Reference Manual
- Intel Math Kernel Library Reference Manual
- Intel Software Documentation Library
- MPI Routines on the ANL Web Site
- OpenMP Specifications

Intel's Top 10 List

1. Download programming books: "Intel Xeon Phi Coprocessor High Performance Programming" by Jeffers & Reinders, and "Parallel Programming and Optimization with Intel Xeon Phi Coprocessors" by Colfax.
2. Watch the parallel programming webinar
3. Bookmark and browse the mic-developer website
4. Bookmark and browse the two developer support forums: "Intel MIC Architecture" and "Threading on Intel Parallel Architectures".
5. Consult the "Quick Start" guide to prepare your system for first use, learn about tools, and get C/C++ and Fortran-based programs up and running

Link to TOP10 List for Starter Kit Developers

Intel's Top 10 List (continued)

6. Try your hand at the beginning lab exercises
7. Try your hand at the beginner/intermediate real world app exercises
8. Browse the case studies webpage to view examples from many segments
9. Begin optimizing your application(s); consult your programming books, the ISA reference manual, and the support forums for assistance.
10. Hone your skills by watching more advanced video workshops

Link to TOP10 List for Starter Kit Developers