Introduction to Xeon Phi programming Part II

Shaohao Chen

High performance computing @ Louisiana State University

Outline of Xeon Phi Programming

Part I (last week)

- Intel Xeon Phi and its computing features
- Usage of Xeon Phi in HPC
- Xeon Phi programming: native mode, offloading

Part II (today)

- Xeon Phi programming: symmetric processing
- Optimization, Debugging and profiling
- Xeon-Phi enabled applications: LAMMPS, NAMD

Review of Native mode

Computer codes

execute natively

MIC

- ❑ Add flag -mmic to create MIC binary files.
- ❑ Log in (ssh) to MIC and execute MIC binary natively.
- ❑ Vectorization is critical.
- ❑ Monitor MIC performance with micsmc.

Review of Offloading

- explicit offload

```
// Some CPU codes ……

 #pragma offload target(mic:0)
  {  // begin offload block
   #pragma omp parallel for
   for ( i=0; i<N; i++ )  {
          // do some works …..
   }
  }  // end offload block

// Some CPU codes ……
```

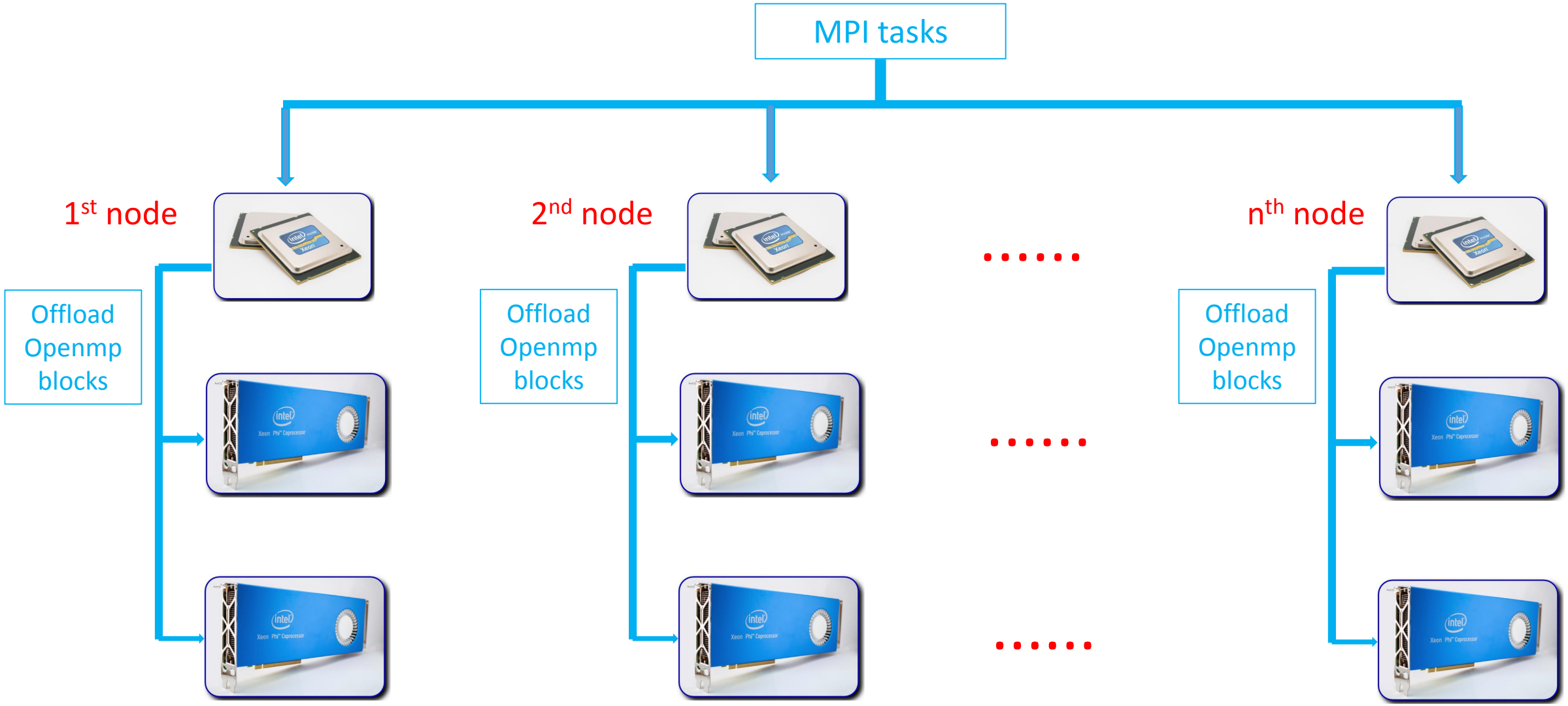
CPU

offload

MIC

CPU

Review: MPI + Offload



Summary for Offloading

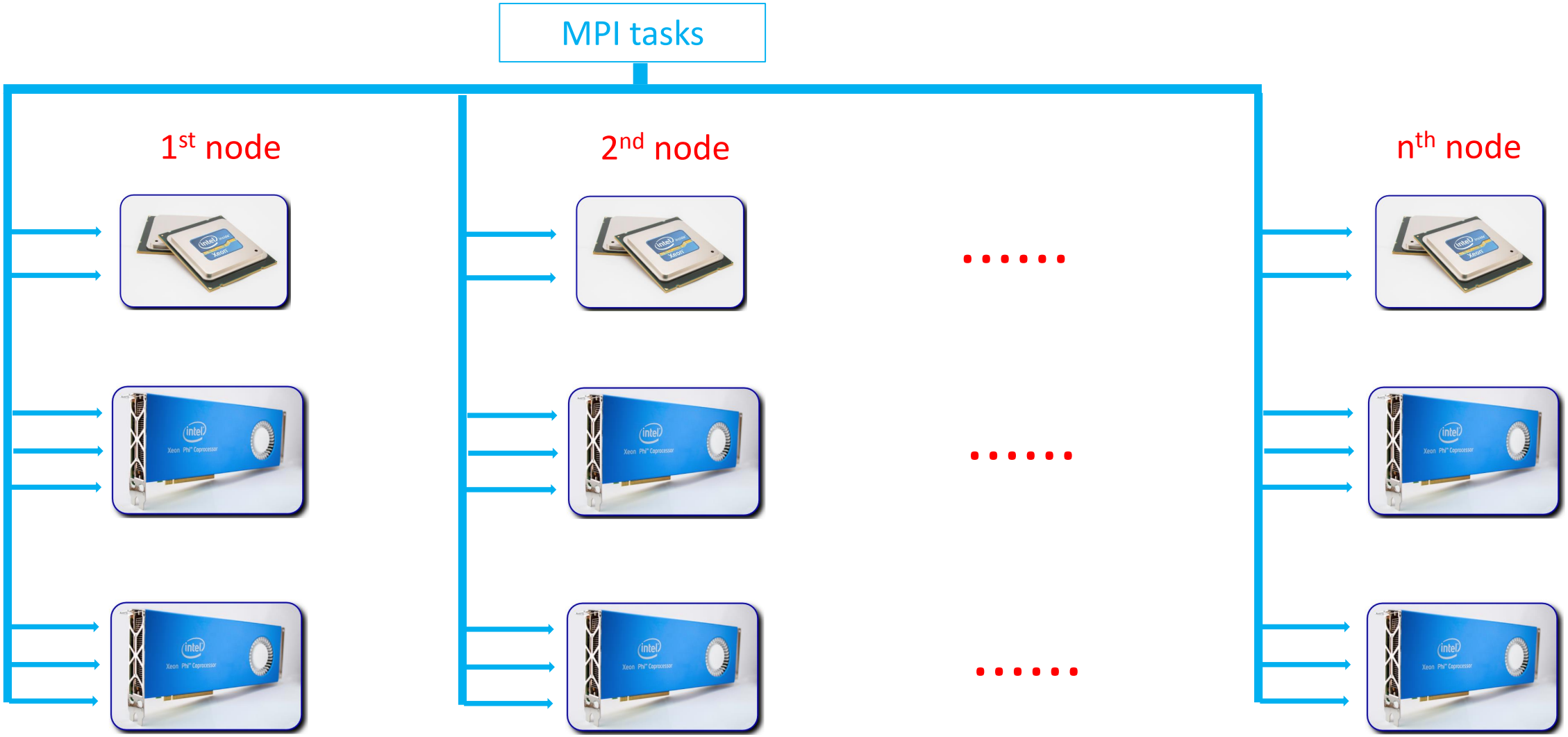
- Explicitly offload blocks by adding lines started with #pragma offload or !dir$ offload in C or Forthran source codes respectively.
- Control data transfer with in, out and inout.
- Place valuables on MIC with attribute or declspec decorations.
- Use wait and offload_wait for asynchronous offload.
- Use offload_transfer for data-only offload.
- Auto offload MKL functions by setting MKL_MIC_ENABLE=1.
- Offload OpenMP blocks in MPI-OpenMP hybrid codes.

Part II

- Xeon Phi programming: symmetric processing
- Optimization, Debugging and profiling
- Xeon-Phi enabled applications: LAMMPS, NAMD

Symmetric processing

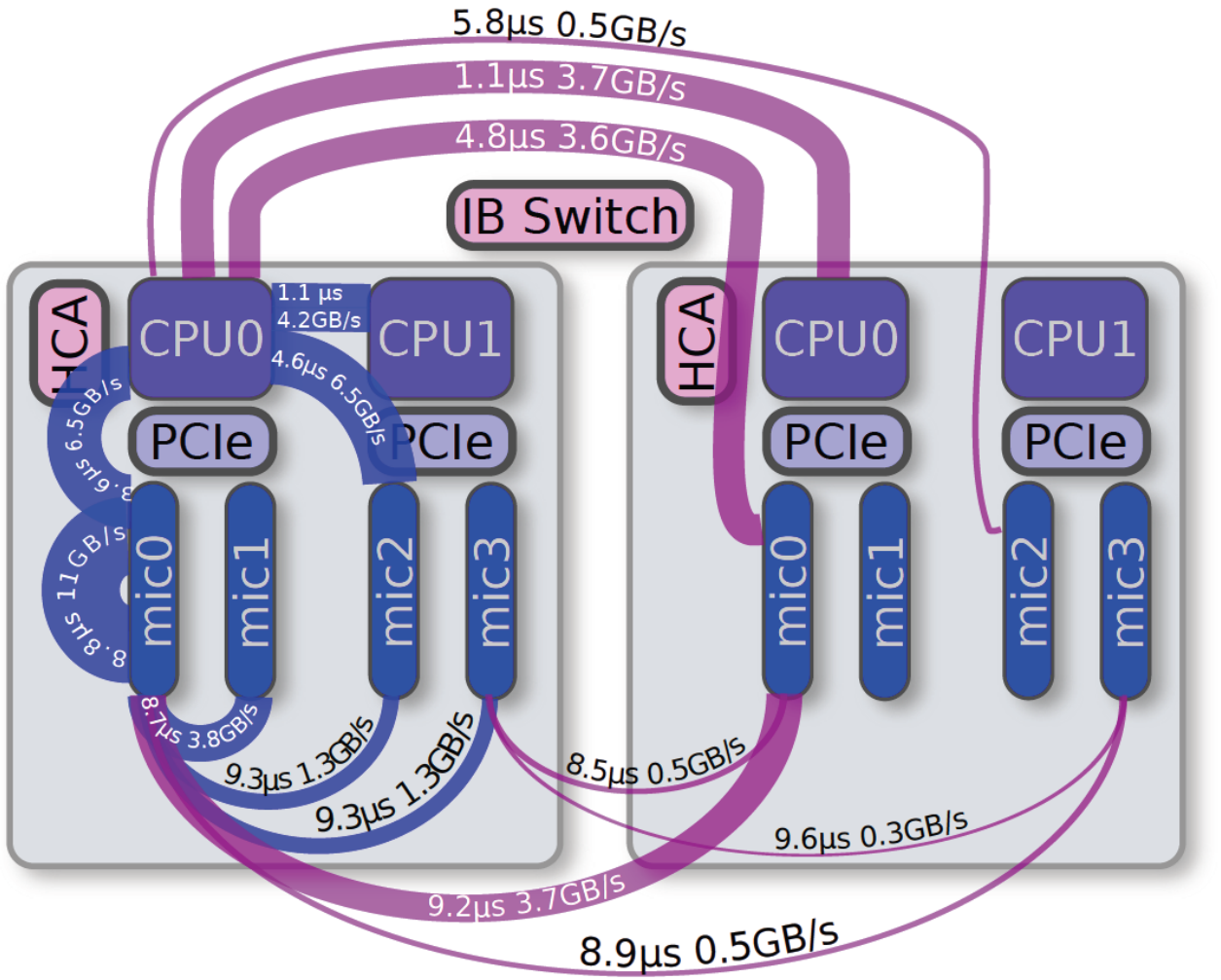
- Distribute MPI tasks “symmetrically” on both CPUs and MICs.



MPI communication

- Latency of short MPI messages and bandwidth of long MPI messages with DAPL InfiniBand

http://research.colfaxinternational.com/



- Benchmark by Colfax

Compilation

❑ Use Intel MPI implementation

```
$ module switch mvapich2/2.0/INTEL-14.0.2 impi/4.1.3.048/intel64
$ module load impi/4.1.3.048/intel64
```

❑ Create CPU and MIC binaries separately

```
$ mpiicc name.c -o name.cpu              # CPU binary, C code
$ mpiicc -mmic name.c -o name.mic        # MIC binary, C code
$ mpiifort name.f90 -o name.cpu          # CPU binary, Fortran code
$ mpiifort -mmic name.f90 -o name.mic    # MIC binary, Fortran code
```

- Add the flag -openmp for an MPI-OpenMP hybrid code.
- There is no change from normal CPU source codes!

Run MPI jobs with mpiexec.hydra

- a command provided by the Intel MPI

```
$ which mpiexec.hydra
/usr/local/compilers/Intel/cluster_studio_xe_2013.1.046/impi/4.1.3.048/intel64/bin/mpiexec.hydra
```

- Launch an MPI job to both host and MICs (from the host)

```
$ mpiexec.hydra -n 20 -host smic017 ./name.cpu : \
  -n 30 -host smic017p-mic0 -env LD_LIBRARY_PATH  $MIC_LD_LIBRARY_PATH ./name.mic : \
  -n 30 -host smic017p-mic1 -env LD_LIBRARY_PATH  $MIC_LD_LIBRARY_PATH ./name.mic
```

- Launch an MPI job only to one MIC (from the host)

```
$ mpiexec.hydra -n 30 -host smic017p-mic0 -env LD_LIBRARY_PATH  $MIC_LD_LIBRARY_PATH ./name.mic
```

A bash script using mpiexec.hydra

```
#!/bin/bash
module load impi/4.1.3.048/intel64      # load Intel  MPI
export TASKS_PER_HOST=2     # number of MPI tasks per host
export THREADS_HOST=10      # number of OpenMP threads spawned by each task on the host
export TASKS_PER_MIC=3      # number of MPI tasks per MIC
export THREADS_MIC=80       # number of OpenMP threads spawned by each task on the MIC
export CPU_ENV="-env OMP_NUM_THREADS $THREADS_HOST"    # CPU run-time environments
export MIC_ENV="-env OMP_NUM_THREADS $THREADS_MIC -env LD_LIBRARY_PATH   $MIC_LD_LIBRARY_PATH"
# MIC run-time environments
mpiexec.hydra \
-n $TASKS_PER_HOST -host smic017 $CPU_ENV ./name.cpu : \              # run on CPU
-n $TASKS_PER_MIC -host smic017p-mic0 $MIC_ENV ./name.mic : \         # run on mic0
-n $TASKS_PER_MIC -host smic017p-mic1 $MIC_ENV ./name.mic             # run on mic1
```

❑Exercise 1: run jobs with mpiexec.hydra

i) Compile pi_hybrid.c or pi_hybrid.f90, then run it with mpiexec.hydra on one compute node. Observe the usage of MICs on the micsmc monitor.

ii) Vary the numbers of MPI tasks and OpenMP threads. Find out the best combination of them so that the computational time is the shortest.

iii) Compare the computational time of the following cases:

1) use only CPU;

2) use only one MIC;

3) use CPU and one MIC;

4) use CPU and two MICs.

- Number of MPI tasks on MIC
  - The theoretical maximum is 61, which is equal to the number of cores on MIC.
  - The practical number should be much less than 61 due to the MIC-memory (16 GB) bottleneck!

- A problem of using mpiexec.hydra

  The command lines become very messy if many nodes are utilized.

Run jobs with micrun.sym

- micrun.sym is a bash script for running symmetric jobs on SuperMIC.

$ which micrun.sym

/usr/local/compilers/Intel/cluster_studio_xe_2013.1.046/impi/4.1.3.048/intel64/bin/micrun.sym

- Automatically obtains the target names, sets up the environments and constructs the complicated command lines.
- Easy for running heavy jobs with many nodes.

- Usage of micrun.sym:

$ mirun.sym -c /path/to/name.cpu -m /path/to/name.mic

$ mirun.sym -c /path/to/name.cpu -m /path/to/name.mic -inp "par1 par2 par3 ..."

A PBS batch script using micrun.sym

```
#!/bin/bash
#PBS -q workq
#PBS -A your_allocation
#PBS -l walltime=01:30:00
#PBS -l nodes=4:ppn=20
……
module load impi/4.1.3.048/intel64      # load Intel  MPI
export TASKS_PER_HOST=20    # number of MPI tasks per host
export THREADS_HOST=1       # number of OpenMP threads spawned by each task on the host
export TASKS_PER_MIC=30     # number of MPI tasks per MIC
export THREADS_MIC=1        # number of OpenMP threads spawned by each task on the MIC
micrun.sym -c /path/to/name.cpu -m /path/to/name.mic      # run with micrun.sym
```

❑Exercise 2: run jobs with micrun.sym

i) Compile pi_hybrid.c or pi_hybrid.f90, then run it with micrun.sym on four compute nodes. Observe the usage of MICs on the micsmc monitor.

ii) Vary the numbers of MPI tasks and OpenMP threads. Find out the best combination of them so that the computational time is the shortest.

iii) Using both CPU and two MICs of each node, compare the computational time of the following cases:

1) use only one node;

2) use four nodes;

3) 16 nodes.

Summary for symmetric processing

- Use Intel MPI (impi) implementation.
- Create CPU and MIC binaries with and without -mmic respectively.
- Run symmetric jobs on few nodes with mpiexec.hydra .
- Run symmetric jobs on many nodes with micrun.sym .
- Balance works on CPU and MICs to obtain the best performance.

Remarks for Xeon Phi programming

❑MKL

❖ If your use MKL, congratulations! MKL functions are automatically offloaded to Xeon Phi and are optimized.

❑Non-MKL: If your code is ……

❖ parallel with OpenMP, explicitly offload the OpenMP blocks to Xeon Phi.

❖ parallel with pure MPI, run it symmetrically on both CPUs and Xeon Phis.

❖ parallel with hybrid MPI and OpenMP, either explicitly offload the OpenMP blocks to Xeon Phi or run it symmetrically on both CPUs and Xeon Phis.

❖ serial, most likely it becomes slower, because the frequency of one Xeon Phi core is much lower than that of one CPU core.

Optimization

- In general, a computer program may be optimized so that it executes more rapidly, or is capable of operating with less memory storage or other resources, or draw less power.

- Optimized codes can be accelerated for both CPU and Xeon Phi.

Optimization Level

- icc -O3 source.c -o mycode

```
#pragma intel optimization_level 3
void my_function() {
 //... Some codes ...
 }
```

- The default optimization level -O2
  - optimization for speed
  - automatic vectorization
  - inlining
  - constant propagation
  - dead-code elimination
  - loop unrolling

- optimization level -O3
  - Enables more aggressive optimization
  - loop fusion
  - block-unroll-and-jam
  - if-statement collapse

Using the *const* Qualifier

```
const int N=1<<28;
double w = 0.5;
double T = (double)N;
double s = 0.0;
for (int i = 0; i < N; i++)
    s += w*(double)i/T;
printf("%e\n", s);
```

slower

```
const int N=1<<28;
const double w = 0.5;
const double T = (double)N;
const double s = 0.0;
for (int i = 0; i < N; i++)
    s += w*(double)i/T;
printf("%e\n", s);
```

faster

Common Subexpression Elimination

```
for (int i = 0; i < n; i++)
{
  for (int j = 0; j < m; j++) {
    const double r =
        sin(A[i])*cos(B[j]);
        // ...
  }
}
```

slower

```
for (int i = 0; i < n; i++){
  const double sin_A = sin(A[i]);
  for (int j = 0; j < m; j++) {
    const double cos_B = cos(B[j]);
    const double r = sin_A*cos_B;
    // ...
  }
}
```

faster

Lower precision is faster

```
const double twoPi = 6.283185307179586;      // double precision
const float phase = 0.3f;                    // single precision
```

```
double sin(double x);        // double precision
float sinf(float x);         // single precision

double exp(double x);        // Double precision
float expf(float x);         // single precision
```

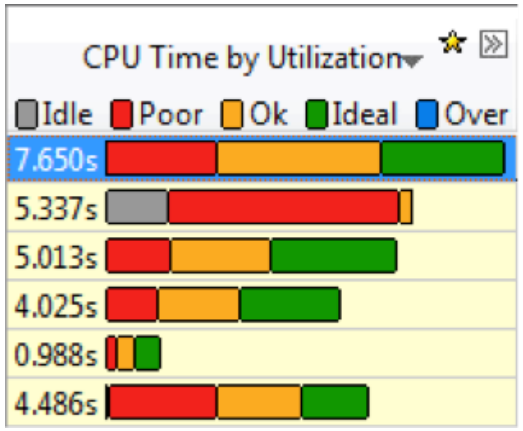
Float is faster than double.

Debugging and Profiling

- Intel VTune Amplifier
- Intel Trace Analyzer and Collector
- GDB: GNU Debug
- TotalView
- Paraview

Intel® VTune™ Amplifier

- Intuitive CPU & coprocessor performance tuning, multi-core scalability, bandwidth and more
- Quick performance insight with advanced data visualization
- Automate regression tests and collect data remotely



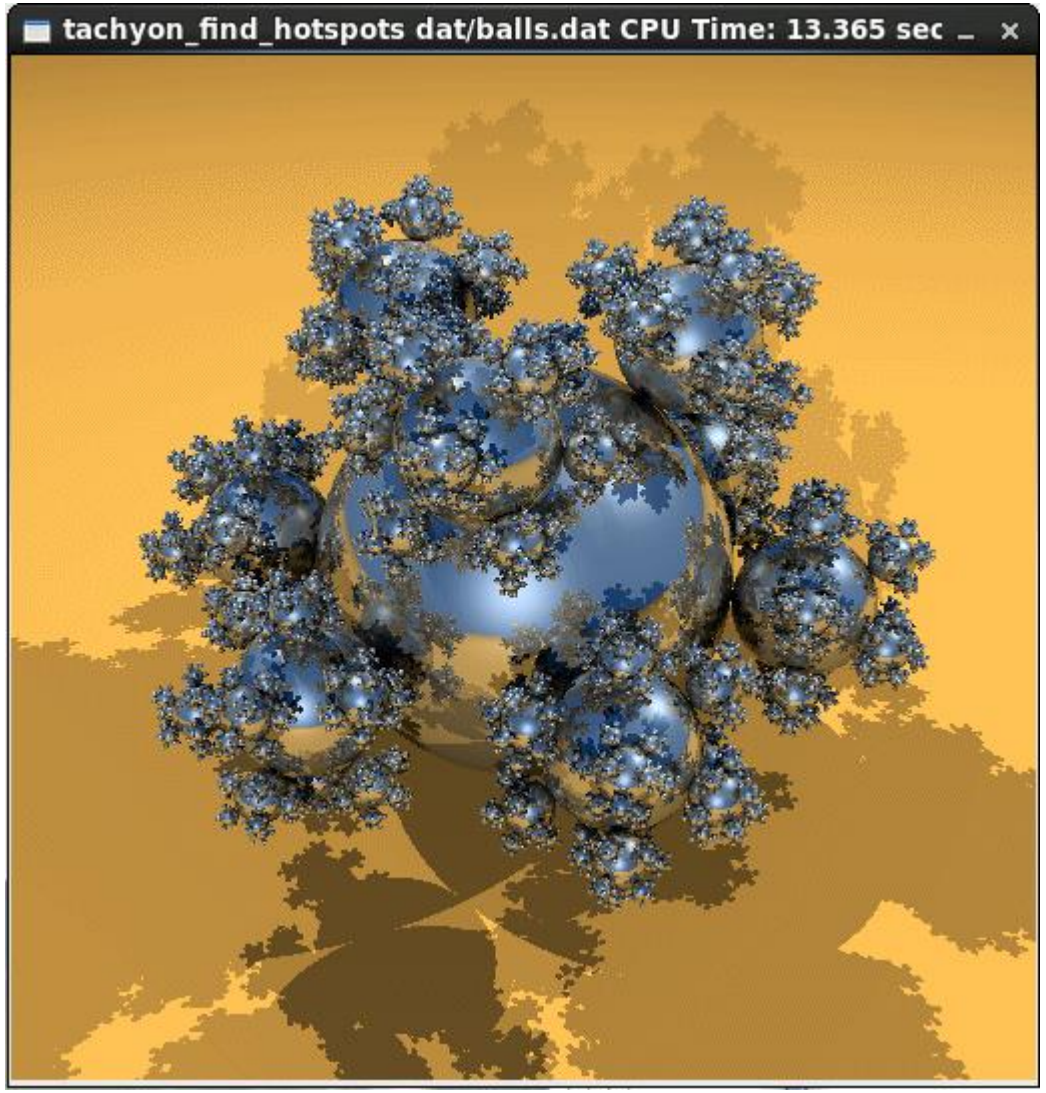
Start VTune Amplifier on SuperMIC

- Currently VTune only works for two compute nodes on SuperMIC: smic099 and smic100.

```
$ ssh –X smic100
$ source /usr/local/compilers/Intel/parallel_studio_xe_2015/vtune_amplifier_xe_2015.1.0.367959/amplxe-vars.sh      # set up environments
$ amplxe-gui &      # graphic interface
$ amplxe-cl         # command line interface
```

A example for using VTune: tachyon

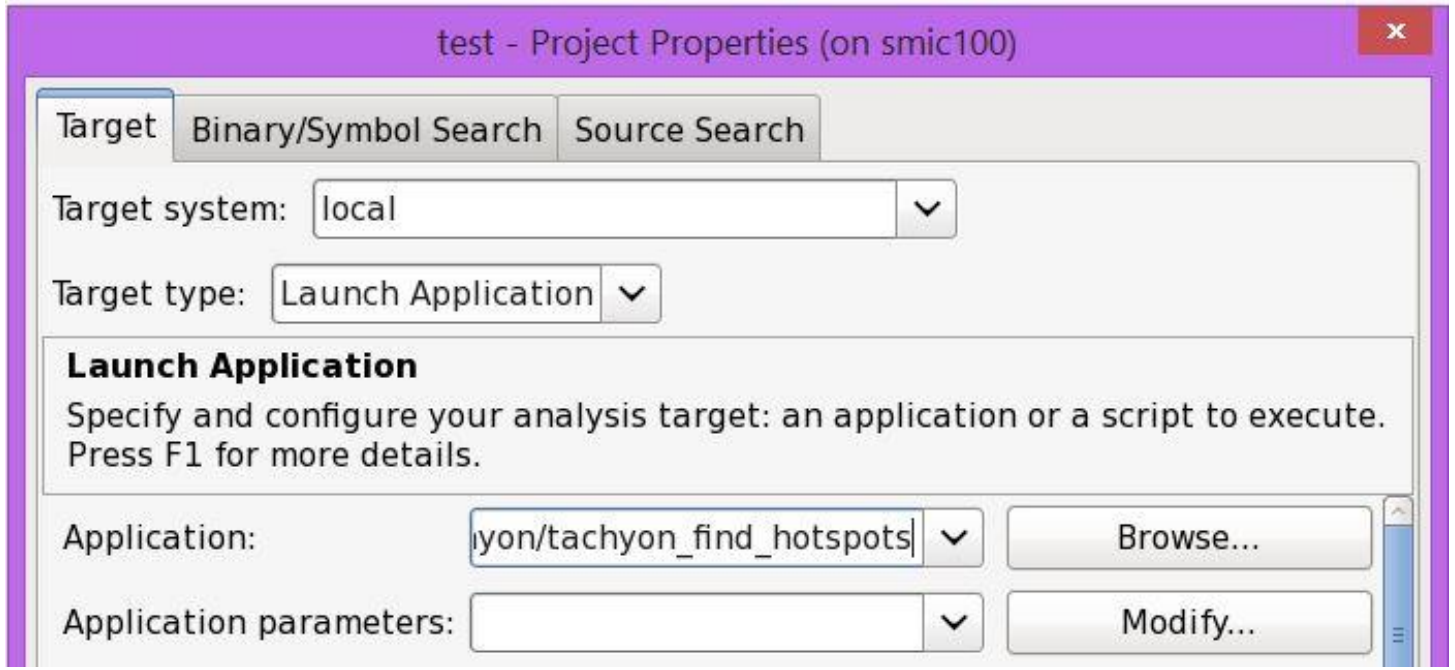
- Fast, high quality parallel ray tracer.
- Renders an image, calculating reflections.



A VTune project for CPU

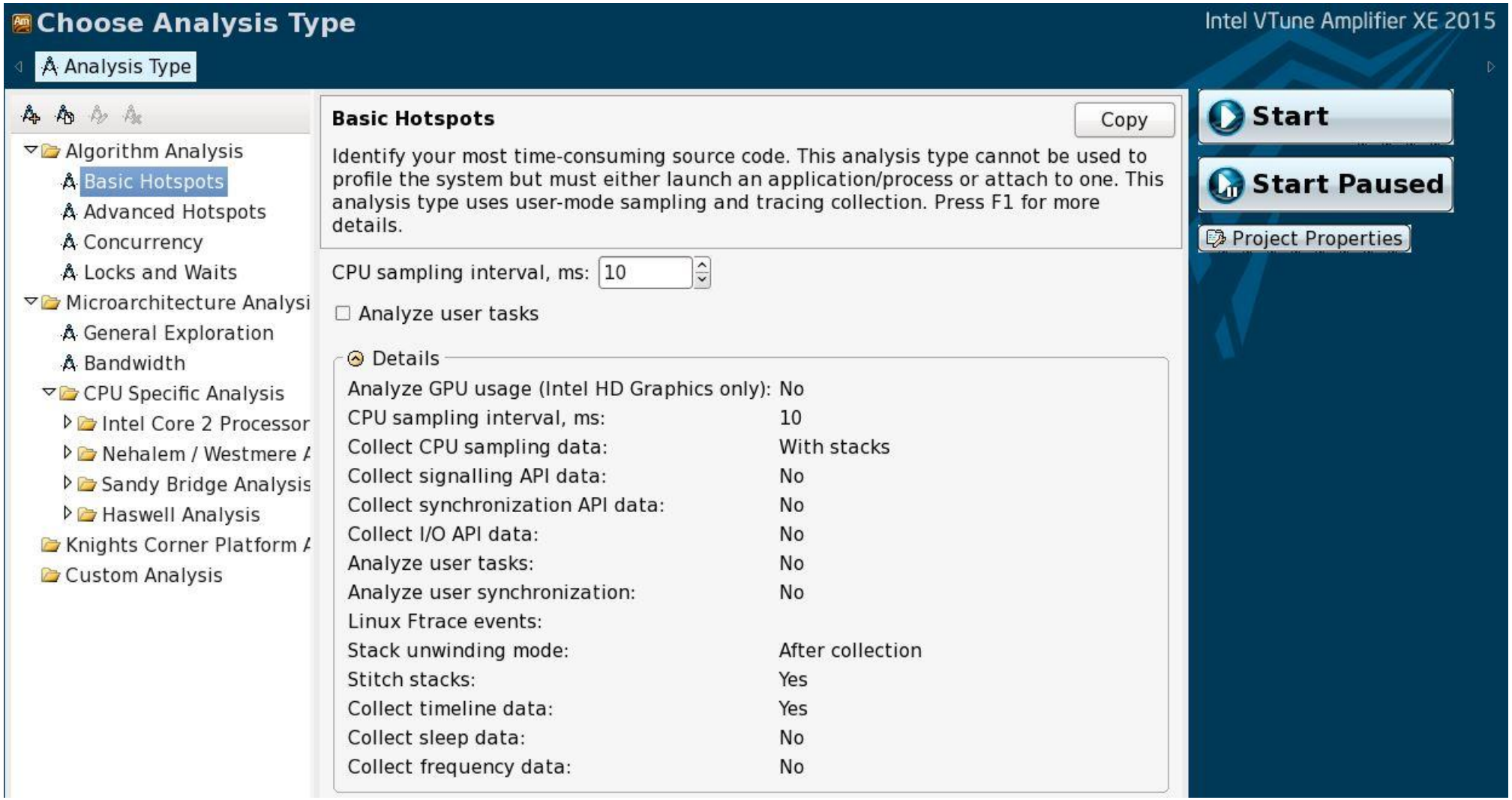
❑ Set up a VTune project

- Create a VTune project: ≡ ->New->Project
- Name the project, click on “Create Project”
- In the “Target System” pull-down menu, select “Local”
- Specify the application /path/to/tachyon_find_hotspots , click on “OK”



❑ Start a new analysis

- Click on “New Analysis”
- Select “Basic Hotspots”, click on “Start”.



VTune analysis: summary

Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree

Elapsed Time: 21.008s

Total Thread Count:	1
Overhead Time:	0s
Spin Time:	0s
CPU Time:	6.850s
Paused Time:	0s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

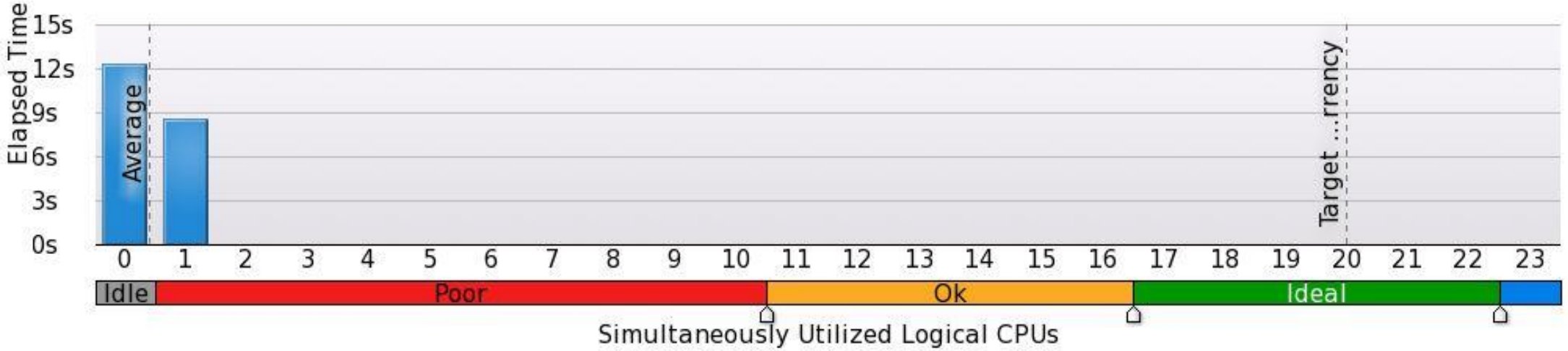
Function	CPU Time
grid_intersect	2.427s
sphere_intersect	2.019s
initialize_2D_buffer	1.753s
grid_bounds_intersect	0.310s
shader	0.070s
[Others]	0.271s

- ❑ CPU time is the sum of the time every thread consumes. (single-threaded in this case).
- ❑ Elapsed time > CPU time. Idle time is large.
- ❑ grid_intersect shows up at the top of the list as the hottest function.

VTune analysis: summary

CPU Usage Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU usage value.



- Ran mostly on one logical CPU, which is classified by the VTune Amplifier as a Poor utilization for a multicore system.

VTune analysis: Bottom-up

Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree

Grouping: Function / Call Stack

Function / Call Stack	CPU Time: Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	Spi. Tim.	O. Ti.	Module	Function
grid_intersect	2.427s	0s	0s	tachyon_find_hotspots	grid_intersect(grid*,
sphere_intersect	2.019s	0s	0s	tachyon_find_hotspots	sphere_intersect(sphe
initialize_2D_buffer	1.753s	0s	0s	tachyon_find_hotspots	initialize_2D_buffer
grid_bounds_intersect	0.310s	0s	0s	tachyon_find_hotspots	grid_bounds_intersec
shader	0.070s	0s	0s	tachyon_find_hotspots	shader(ray*)
pos2grid	0.060s	0s	0s	tachyon_find_hotspots	pos2grid(grid*, vecto
tri_intersect	0.050s	0s	0s	tachyon_find_hotspots	tri_intersect(tri*, ray*
intersect_objects	0.030s	0s	0s	tachyon_find_hotspots	intersect_objects(ray*
XPutImage	0.020s	0s	0s	libX11.so.6.3.0	XPutImage
add_intersection	0.020s	0s	0s	tachyon_find_hotspots	add_intersection(dou
light_intersect	0.020s	0s	0s	tachyon_find_hotspots	light_intersect(point_
VScale	0.011s	0s	0s	tachyon_find_hotspots	VScale(vector*, doubl
ColorScale	0.010s	0s	0s	tachyon_find_hotspots	ColorScale(color*, dou
fscanf	0.010s	0s	0s	libc-2.12.so	fscanf
XPending	0.010s	0s	0s	libX11.so.6.3.0	XPending
Selected 1 row(s):	2.427s	0s	0s		

- All functions are marked with red bars, which means that the processor cores were underutilized.

VTune analysis: source code

- Double click the hottest function.

Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Tasks and Frames | grid.cpp

Source | Assembly | Assembly grouping: Address

S. L.	Source	CPU Time: Total – Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	CPU Time: Total – Spin Time	CPU Time: Total – Ov.. Time	CPU Time: Self – Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)
569	if (ry->maxdist < tmax.x || curvox.x == out.x)	0.0%	0.0%	0.0%	
570	break;	0.0%	0.0%	0.0%	
571	voxindex += step.x;	0.0%	0.0%	0.0%	
572	tmax.x += tdelta.x;	0.0%	0.0%	0.0%	
573	curpos = nXp;	0.1%	0.0%	0.0%	0.010s
574	nXp.x += pdeltaX.x;	0.0%	0.0%	0.0%	
575	nXp.y += pdeltaX.y;	0.0%	0.0%	0.0%	
576	nXp.z += pdeltaX.z;	0.0%	0.0%	0.0%	
577	}	0.0%	0.0%	0.0%	
578	else if (tmax.z < tmax.y) {	0.0%	0.0%	0.0%	
579	cur = g->cells[voxindex];	0.3%	0.0%	0.0%	0.020s
580	while (cur != NULL) {	2.0%	0.0%	0.0%	0.139s
581	if (ry->mbox[cur->obj->id] != ry->serial) {	15.9%	0.0%	0.0%	1.088s
582	ry->mbox[cur->obj->id] = ry->serial;	6.7%	0.0%	0.0%	0.461s
583	cur->obj->methods->intersect(cur->obj, ry);	3.1%	0.0%	0.0%	0.120s
584	}	0.0%	0.0%	0.0%	
585	cur = cur->next;	2.9%	0.0%	0.0%	0.200s
586	}	0.0%	0.0%	0.0%	

A VTune project for offloading to Xeon Phi

- An example (pi_hybrid_off.c):

  calculate the value of pi with integration method.

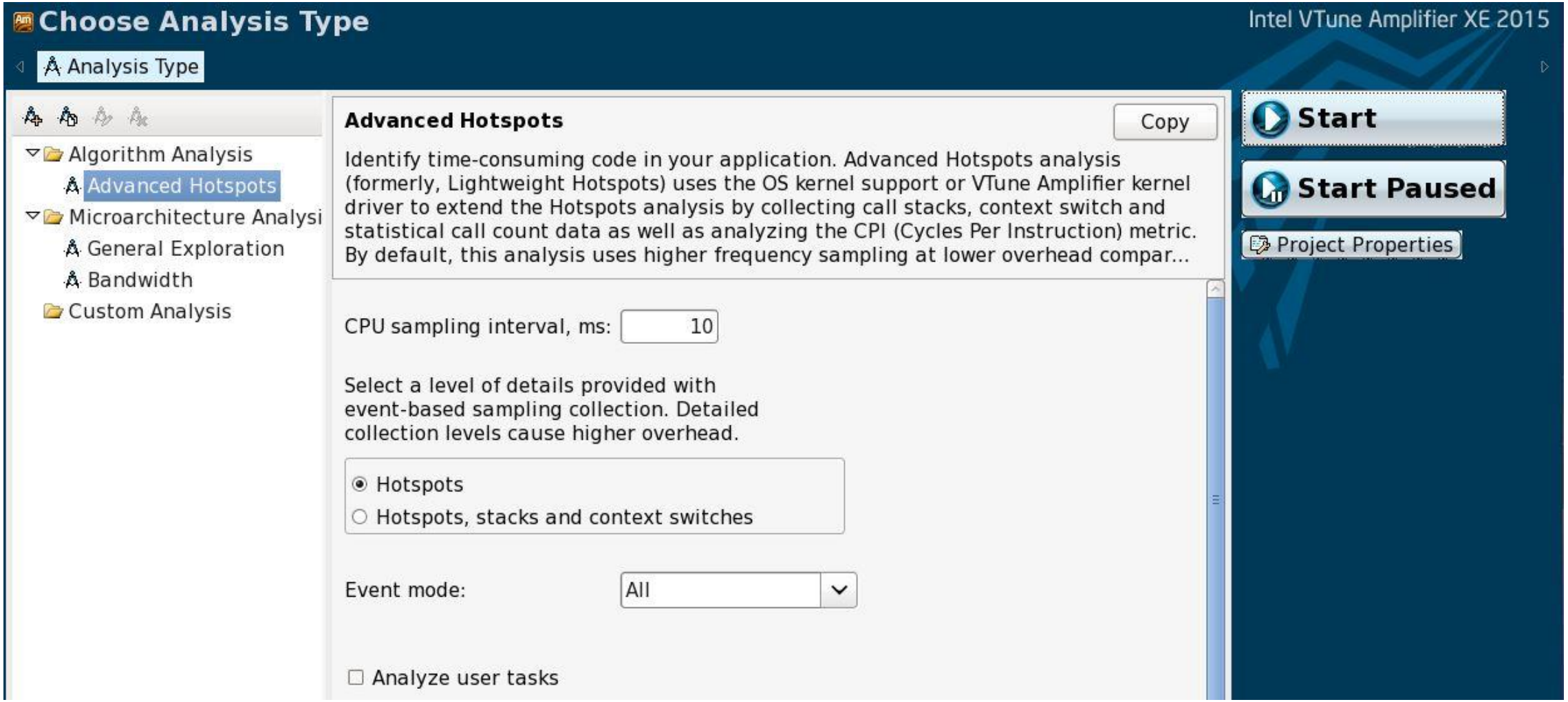
$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

- Set up a VTune project

- Create a VTune project: ≡ ->New->Project
- Name the project, click on "Create Project"
- In the "Target System" pull-down menu, select "Intel Xeon Phi coprocessor (host launch)"
- Specify the application /path/to/pi_hybrid.off , click on "OK"

❑ Start a new analysis

- Click on “New Analysis”
- Select “Advanced Hotspots”, click on “Start”.



VTune analysis: summary

Elapsed Time: 6.673s

CPU Time:	981.536s
Instructions Retired:	296,688,000,000
CPI Rate:	4.096

The CPI may be too high. This could be caused by issues such as memory stalls, instruction starvation, branch misprediction or long latency instructions. Explore the other hardware-related metrics to identify what is causing high...

CPU Frequency Ratio:	1.000
Paused Time:	0s
Overhead Time:	0.026s
Spin Time:	79.587s

OpenMP Analysis. Collection Time: 6.673

Serial Time (outside any parallel region): 2.837s (42.5%)

Serial Time of your application is high. It directly impacts application Elapsed Time and scalability. Explore options for parallelization, algorithm or microarchitecture tuning of the serial part of the application.

Parallel Region Time: 3.836s (57.5%)

Estimated Ideal Time:	3.718s (55.7%)
Potential Gain:	0.118s (1.8%)

Vtune result analysis: summary

Top OpenMP Regions by Potential Gain

This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed time that could be saved if the region was optimized to have no load imbalance assuming no runtime overhead.

OpenMP Region	Potential Gain	(%)	Elapsed Time
do_some_integratinompparallel@unknown:23:43	0.118s	1.8%	3.836s

Top Hotspots

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

Function	CPU Time
do_some_integratin	886.166s
__kmp_wait_sleep	69.454s
[vmlinux]	12.599s
__kmp_static_yield	5.149s
__kmp_yield	2.793s
[Others]	5.375s

VTune result analysis: Bottom-up

/home/shaohao/intel/amplxe/projects/pi_offload_host_launched - Intel VTune Amplifier (on smic100)

Project Navigator
/home/shaohao/intel/...
matmul_native
pi_offload
pi_offload_host_la
r000ah

Welcome | r000ah

Advanced Hotspots Hotspots viewpoint (change) Intel VTune Amplifier XE 2015

Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Tasks and Frames

Grouping: Function / Call Stack

Function / Call Stack	CPU Time: Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	CPU Time: Spin Time	CPU Time: Ove... Time	Instructions Retired	CPI Rate	CP. Fr. Ra.	Module	
[Outside any known module]	0.015s	0s	0s	11,200,000	1.607	1.000		[Outside any kr
[coi_daemon]	0.003s	0s	0s	0	0.000	1.000	coi_daemon	[coi_daemon]
do_some_integratin	886.166s	0s	0s	276,539,200,000	3.967	1.000	pi_hybrid.off	do_some_integr
[libcfs]	0.002s	0s	0s	0	0.000	1.000	libcfs	[libcfs]
[libstdc++.so.6.0.16]	0.003s	0s	0s	0	0.000	1.000	libstdc++.so.6.0.16	[libstdc++.so.6
[libpthread-2.14.90.so]	0s	0.005s	0s	0	0.000	1.000	libpthread-2.14.90.so	[libpthread-2.1
__kmp_stg_find	0s	0s	0.002s	0	0.000	1.000	libiomp5.so	__kmp_stg_find
[ptlrpc]	0.003s	0s	0s	0	0.000	1.000	ptlrpc	[ptlrpc]
[libcoi_device.so.0]	0.010s	0s	0s	5,600,000	2.143	1.000	libcoi_device.so.0	[libcoi_device.s
[dma_module]	0.005s	0s	0s	0	0.000	1.000	dma_module	[dma_module]
[ringbuffer]	0.002s	0s	0s	0	0.000	1.000	ringbuffer	[ringbuffer]
Selected 1 row(s):	886.166s	0s	0s	276,539,200,000	3.967	1.000		

0.5s 1s 1.5s 2s 2.5s 3s 3.5s 4s 4.5s 5s 5.5s 6s 6.5s

Thread: OMP Worke, OMP Worke, OMP Worke, OMP Worke, OMP Worke, OMP Worke, OMP Worke, OMP Worke

CPU Time

Ruler Area
Region ...
Thread
Running
CPU Time
Spin an...
Hardwar...
CPU Time
CPU Time
Spin an...

No filter... applied. | Any Process | Any Thread | Any Module | Utilization: Any Utilization

Call Stack Mode: User functions + 1 | Inline Mode: on | Loop Mode: Functions only

VTune analysis: Top-down tree

/home/shaohao/intel/amplxe/projects/pi_offload_host_launched - Intel VTune Amplifier (on smic100)

Project Navigator
/home/shaohao/intel/...
matmul_native
pi_offload
pi_offload_host_lau
r000ah

Welcome | r000ah

Advanced Hotspots Hotspots viewpoint (change) — Intel VTune Amplifier XE 2015

Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Tasks and Frames

Grouping: Call Stack

Function Stack	CPU Time: Total — Effective Time by Utilization (Idle / Poor / Ok / Ideal / Over)	Spin Time	Ov.. Time	CPU Time: Self — Effective Time by Utilization (Idle / Poor / Ok / Ideal / Over)	Spi. Tim.	Ov. Tim.	Instr... Retir... Total	Ins.. Re.. Self	CPI Rat... Total
Total	91.9%	8.1%	0.0%	0s	0s	0s	100.0%	0	4.096
do_some_integratin	90.3%	0.0%	0.0%	886.166s	0s	0s	93.2%	27..	3.967
__kmp_wait_sleep	0.0%	7.8%	0.0%	0s	69...	0s	5.8%	15,..	5.533
[vmlinux]	1.3%	0.0%	0.0%	12.599s	0s	0s	0.8%	2,3..	6.523
__kmp_yield	0.0%	0.3%	0.0%	0s	2.7..	0s	0.1%	40..	8.636
[libittnotify_collector.so]	0.3%	0.0%	0.0%	2.582s	0s	0s	0.0%	64,..	49....
[libc-2.14.90.so]	0.0%	0.0%	0.0%	0.255s	0s	0s	0.0%	64,..	4.907
[sep3_15]	0.0%	0.0%	0.0%	0.179s	0s	0s	0.0%	0	0.000
[micscif]	0.0%	0.0%	0.0%	0.047s	0s	0s	0.0%	16,..	3.452
[ld-2.14.90.so]	0.0%	0.0%	0.0%	0.019s	0s	0s	0.0%	8,4..	2.857
[Outside any known module	0.0%	0.0%	0.0%	0.015s	0s	0s	0.0%	11,.	1.607
Highlighted 1 row(s):	90.3%	0.0%	0.0%	886.166s	0s	0s	93.2%	27..	3.967

0.5s 1s 1.5s 2s 2.5s 3s 3.5s 4s 4.5s 5s 5.5s 6s 6.5s

Thread: OMP Worke, OMP Worke, OMP Worke, OMP Worke, OMP Worke, OMP Worke, OMP Worke, OMP Worke

CPU Time

Ruler Area
Region Insta...
Thread
Running
CPU Time
Spin and Ove...
Hardware Eve...
CPU Time
CPU Time
Spin and Ove...

No filter... applied. | Any Process | Any Thread | Any Module | Utilization: Any Utilization

Call Stack Mode: User functions + 1 | Inline Mode: on | Loop Mode: Functions only

Xeon-Phi enabled applications

- Physics
  - Chroma QCD
  - QphiX-QCD
- Computational chemistry
  - NWChem
- Molecular dynamics
  - LAMMPS
  - NAMD
  - GROMACS
  - AMBER
- COMPUTATIONAL FLUID DYNAMICS
  - OpenLB
  - LBS3D
- Material science
  - Quantum ESPRESSO
- Finance
  - BlackScholes SP and DP
  - Monte Carlo SP and DP
- Development tools
  - DDT
  - Matlab
  - R
  - TAU
- Libraries
  - Boost
  - MAGMA
  - MVPICH2

Build Libraries for Xeon-Phi

- Static Libraries with Offload

```
$ icc -c myobject1.c myobject2.c
$ ifort -c myobject1.f90 myobject2.f90
$ xiar -qoffload-build libname.a myobject1.o myobject2.o
$ icc name.c -L/path/to/lib -llibname -o name
```

Molecular dynamics simulation I: LAMMPS

- Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) is a classical molecular dynamics code distributed by Sandia National Laboratory.
- Has potentials for solid-state materials (metals, semiconductors) and soft matter (biomolecules, polymers) and coarse-grained or mesoscopic systems. It can be used to model atoms or, more generically, as a parallel particle simulator at the atomic, meso, or continuum scale.
- Simulate the time evolution of the input system of atoms or other particles, as specified in the input script, writing data, including atom positions, thermodynamic quantities, and other statistics computations.
- Runs on single processors or in parallel using message-passing techniques with a spatial-decomposition of the simulation domain. The code is designed to be easy to modify or extend with new functionality.

LAMMPS for Xeon Phi

- A LAMMPS load balancer offloads part of neighbor-list and non-bond force calculations to the Intel® Xeon Phi™ coprocessor for concurrent calculations with the CPU. This is achieved by using offload directives to run calculations well suited for many-core chips on both the CPU and the coprocessor. In this model, the same C++ routine is run twice, once with an offload flag, to support concurrent calculations.
- The dynamic load balancing allows for concurrent 1) data transfer between host and coprocessor, 2) calculations of neighbor-list, non-bond, bond, and long-range terms, and 3) some MPI communications. It continuously updates the fraction of offloaded work to minimize idle times. A standard LAMMPS “fix” object manages concurrency and synchronization.
- The Intel® package adds support for single, mixed, and double-precision calculations on both CPU and coprocessor, and vectorization (AVX on CPU / 512-bit vectorization on Phi™). This can provide significant speedups for the routines on the CPU, too.

Run LAMMPS

- Load LAMMPS (MIC version) with module

```
module load impi/4.1.3.048/intel64                    # Intel MPI
module load lammps/21Jan15/INTEL-14.0.2-impi-4.1.3.048-mic    # MIC-enabled LAMMPS
```

- Prepare an input file:

Number of MICs

Precision:
single
double
mixed

Fraction of offloading works:
0.0 → no offload
0.5 → offload half works to MIC
-1 → automatically balance

```
package intel $m mode mixed balance $b
package omp 0
suffix $s
……
```

Suffix:
omp → use only CPU with OpenMP
intel → use both CPU and MIC

Run LAMMPS

module load impi/4.1.3.048/intel64

- Run with only CPU

$ export OMP_NUM_THREADS=2

$ mpirun -np 10 lmp_intel_phi -in in.lc -v s omp -v m 0 -v b 0

Use the input file in.lc .

Run 10 MPI tasks on CPU. Each MPI task uses 2 threads.

- Run with CPU and MIC

$ export OMP_NUM_THREADS=2

$ mpirun -np 10 lmp_intel_phi -in in.lc -v s intel -v m 1 -v b -1

Run 10 MPI tasks on CPU. Each MPI task uses 2 threads.

And Run 10 MPI tasks on 1 MIC. Each MPI task uses 24 threads.

❑Exercise 3: Use LAMMPS to calculate liquid crystal structure

Run LAMMPS with the sample input file in.lc for the following cases, then compare their computational time.

1) Use only CPU, with 20 MPI task and 1 Openmp thread.

2) Use only CPU, with 10 MPI tasks and 2 Openmp threads.

3) Use CPU and one MIC, with 10 MPI tasks and 2 Openmp threads.

4) Use CPU and two MICs, with 10 MPI tasks and 2 Openmp threads.

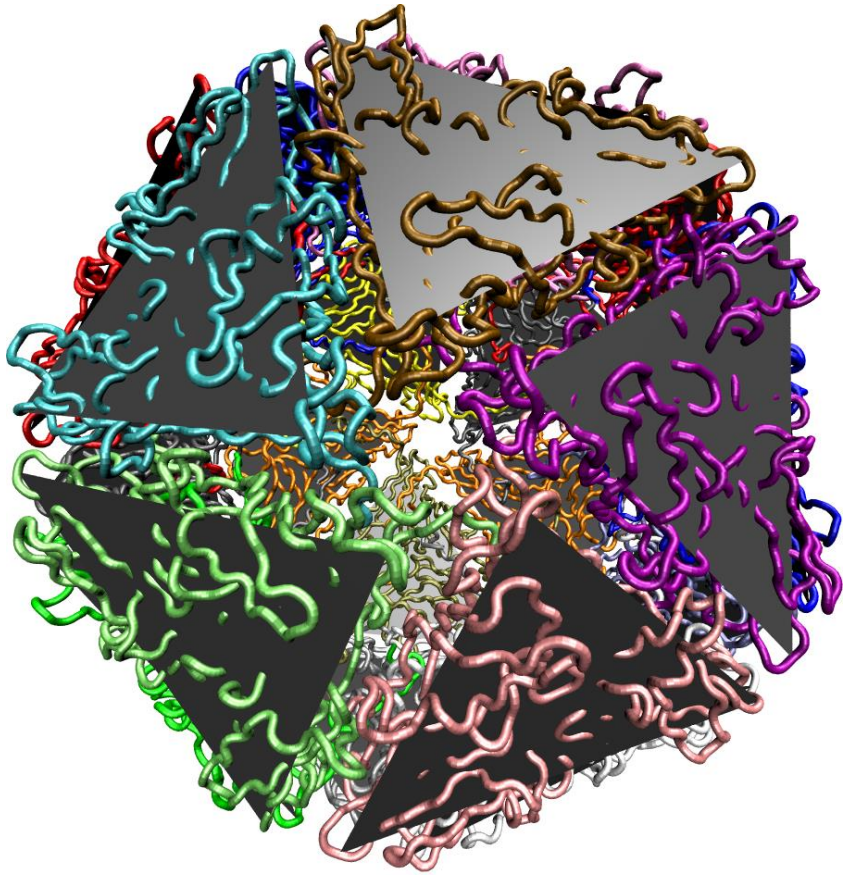
5) Run case 4) using four nodes.

Molecular dynamics simulation II: NAMD

- NAMD (NAnoscale Molecular Dynamics program) is a freeware molecular dynamics simulation package
- Written using the Charm++ parallel programming model, noted for its parallel efficiency and often used to simulate large systems (millions of atoms).
- Simulates the life of bio-molecules
- Forces on each atom calculated every step
- Positions and velocities updated and atoms migrated to their new positions

NAMD: STMV (virus) benchmark

- Satellite Tobacco Mosaic Virus (STMV) is a small, icosahedral plant virus which worsens the symptoms of infection by Tobacco Mosaic Virus.
- The entire STMV particle consists of 60 identical copies of a single protein that make up the viral capsid (coating), and a 1063 nucleotide single stranded RNA genome which codes for the capsid and one other protein of unknown function.
- STMV is useful for demonstrating scaling to thousands of processors.



Theoretical and Computational Biophysics Group
Beckman Institute
University of Illinois at Urbana-Champaign

Run NAMD

- Load NAMD

```
module load namd/2.10/INTEL-14.0.2-ibverbs          # use CPU version
module load namd/2.10/INTEL-14.0.2-ibverbs-mic      # use MIC version
```

- Run NAMD in a PBS script

```
cd $PBS_O_WORKDIR
for node in `cat $PBS_NODEFILE | uniq`; do echo host $node; done > nodelist
export NPROCS=`wc -l $PBS_NODEFILE |gawk '//{print $1}'`
`which charmrun` ++p $NPROCS ++nodelist nodelist ++remote-shell ssh `which namd2`
stmv.namd > output
```

- Note: Auto detect the number of MICs on a node. Use all available MICs and CPU.

❑Exercise 4: Use NAMD to calculate STMV benchmark

Run NAMD with the sample input files stmv.namd, stmv.pdb and stmv.psf for the following cases, then observe the scaling of computational time.

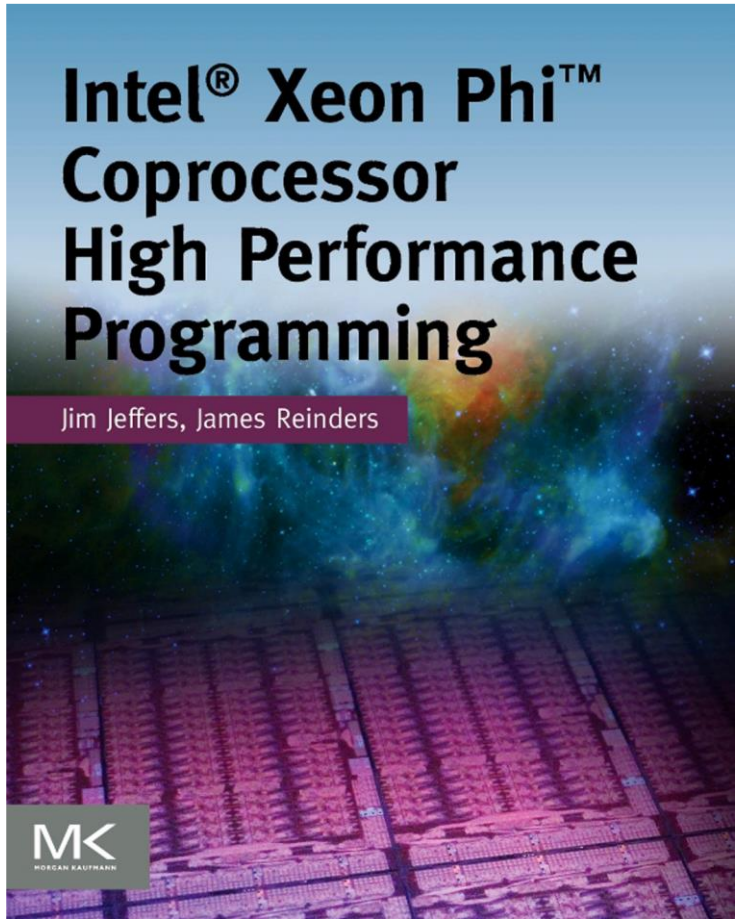
1) Run the CPU version, using 2 nodes.

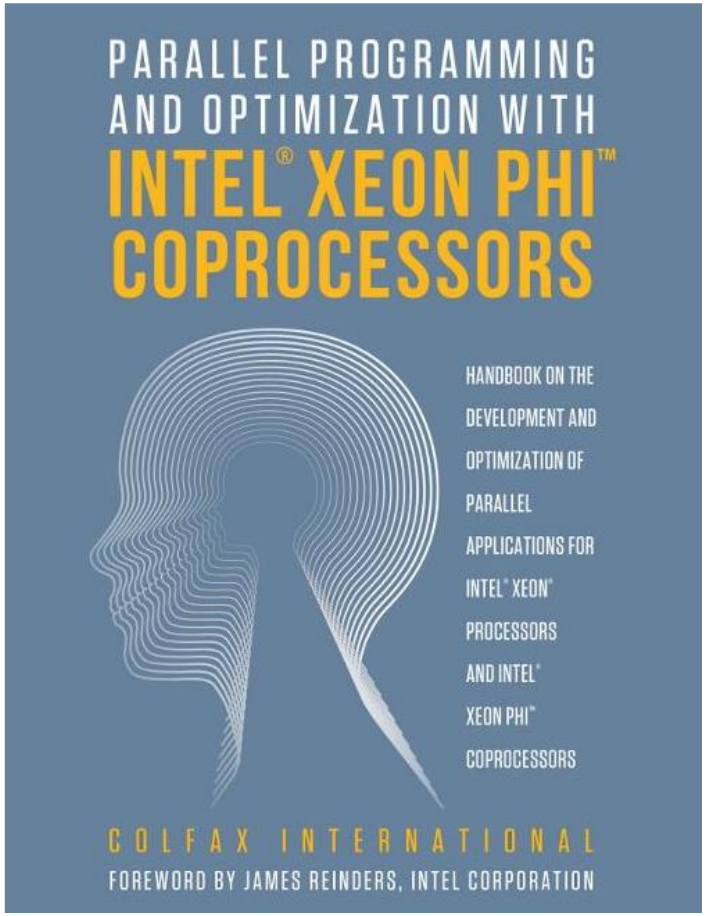
2) Run the CPU version, using 8 nodes.

3) Run the MIC version, using 2 nodes.

4) Run the MIC version, using 8 nodes.

References





- User guide of SuperMIC: http://www.hpc.lsu.edu/docs/guides.php?system=SuperMIC