Programming coprocessor Intel Xeon Phi: from simple to complex

Efim Sergeev  - Senior Software Engineer at Singularis Lab LLC
Contents

• Specification
• Connection
• Compilation & Run
  – Simple app
  – MKL
  – Pure MPI
  – Hybrid
• Slurm
## RSC PETASTREAM (Standard configuration)

<table>
<thead>
<tr>
<th>Type</th>
<th>Massively Parallel Supercomputer <a href="#">RSC PetaStream</a></th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>Intel Many Integrated Core (MIC)</td>
</tr>
<tr>
<td>Performance</td>
<td>1,2 PFlops per rack (peak)</td>
</tr>
<tr>
<td>Compute resources</td>
<td>1024x Intel® Xeon Phi™ 7120D based nodes (total resources: 61.5K cores, 246K threads)</td>
</tr>
<tr>
<td>Memory</td>
<td>16 TB GDDR5</td>
</tr>
<tr>
<td>Interconnect</td>
<td>Infiniband FDR 56Gbps / Intel Truescale Infiniband / 10 GigE</td>
</tr>
<tr>
<td>Local Data Storage</td>
<td>Up to 640x Intel® DC S3500/S3700 SSDs with total capacity up to 0.5PB</td>
</tr>
<tr>
<td>Operating System</td>
<td>Linux (RedHat, SUSE, Debian, CentOS, Scientific Linux), MS Windows</td>
</tr>
<tr>
<td>Job management</td>
<td>SLURM, PBS Pro, Moab, Platform LSF</td>
</tr>
<tr>
<td>Libraries, Compilers and Tools</td>
<td>Intel® Cluster Studio XE 2015</td>
</tr>
<tr>
<td>Form factor</td>
<td>Dual side access rack</td>
</tr>
<tr>
<td>Cooling</td>
<td>RSC direct liquid cooling system: up to 400kW per Cabinet;</td>
</tr>
</tbody>
</table>
Introducing Intel® Xeon Phi Coprocessors

- Up to 61 IA cores/1.1 GHz/ 244 Threads
- Up to 8GB memory with up to 352 GB/s bandwidth
- 512-bit SIMD instructions
- Linux operating system, IP addressable
- Standard programming languages and tools
- Over 1 TeraFlop/s double precision peak performance
Architecture Overview

- 60(61) cores
- 4 HW threads/core
- 8 memory controllers 16 Channel GDDR5
- High-speed bi-directional ring interconnect Fully Coherent L2 Cache
Petastream Overview
• ssh user@hpc.spbstu.ru
Type your login and password.

:-)
2. Run putty.exe
3. Enter the ip address `hpc.spbstu.ru`, port 22 and connection name.
4. Click “Save” button
5. Click “Open” button and run ssh session
6. Type your login and password in terminal
WinSCP - edit and transfer files

- Download WinSCP from home page [http://winscp.net/eng/download.php](http://winscp.net/eng/download.php)
- Add new connection: Connection -> New connection...:
  - Use host name `hpc.spbstu.ru`
  - Use protocol SFTP
  - Use port 22
  - Use your own user name and password
- Connect to your new connection, and input your password if required.
WinSCP - edit and transfer files

- After a successful connection to one of the panels appear user's home directory on a cluster. You can copy files between any directory on your machine, and home directory on a cluster.

- Any file in the home directory can be edited by F4 command, be sure that favorite editor selected.
Resource allocation

• Allocate Xeon Phi
  # salloc -N 1 --partition=mic
  salloc: Granted job allocation 83350

• Check allocation
  # squeue
  JOBID  PARTITION  NAME   USER  ST   TIME  NODES  NODELIST(REASON)
  5094   mic      bash  user2  R  0:03    1 host-mic0

• Run MPI program from host
  #mpiexec.hydra -p mic -n 64 ./myapp.MIC

• Connect to mic
  # ssh 'host’-mic0
  # ‘run some app’
  # exit

• Cancel resources allocation
  #scancel 83350
Compilation & Run: simple application

• Build app on Host
  #cd ./Workshop2016/src/
  #icc -mmic -openmp omp_native.cpp

• Connect to allocated mic
  #ssh 'hostname-mic0'

• Select directory with binaries
  #cd ./Workshop2016/src

• Run app
  #./a.out 2048 64
  #export OMP_NUM_THREADS=30
  #./a.out 2048 64
  #unset OMP_NUM_THREADS
  #exit
Intel® Math Kernel Library (Intel® MKL) accelerates math processing routines that increase application performance and reduce development time. Intel® MKL includes highly vectorized and threaded Linear Algebra, Fast Fourier Transforms (FFT), Vector Math and Statistics functions.

MKL Mathematical Functionality

– Basic Linear Algebra BLAS, Sparse BLAS, PBLAS
– Linear solvers LAPACK, ScaLAPACK1, Sparse Solver routines
– Vector Mathematical & Statistical functions VML, VSL
– Fourier Transform functions
Compiliation & Run: MKL

- **Build app on Host**
  
  ```bash
  #icc -mmic -openmp -mkl mCarlo.cpp
  ```

- **Connect to allocated mic**
  
  ```bash
  #ssh 'hostname-mic0'
  ```

- **Run app**
  
  ```bash
  #cd ./Workshop2016/src
  #./a.out
  #...
  #exit
  ```
Intel® MPI Library is a message passing library that implements the Message Passing Interface, version 3.0 (MPI-3.0) specification. Use the library to develop applications that can run on multiple cluster interconnects.

The Intel® MPI Library has the following features:

- Scalability up to 262k processes
- Low overhead, enables analyzing large amounts of data
- MPI tuning utility for accelerating your applications
- Interconnect independence and flexible runtime fabric selection
Compilation & Run: MPI

- **Build app on Host**
  
  ```
  # mpiicc -mmic -o test.MIC test.c
  ```

- **Set environment variable** `I_MPI_MIC`
  
  ```
  # export I_MPI_MIC=1
  ```

- **Load modules for mpiexec.hydra and mic:**
  
  ```
  # module load launcher/mpiexec mic
  ```

- **Run app**
  
  ```
  # mpiexec.hydra -n 2 ./test.MIC
  # mpiexec.hydra ./test.MIC
  # mpiexec.hydra -n 60 ./test.MIC
  ```
Run: MPI mapping/pinning

- Set environment variable `I_MPI_DEBUG=4` for explore mapping information
  ```
  # export I_MPI_DEBUG=4
  ```

- The environment variable `I_MPI_PIN_DOMAIN` controls the mapping/pinning. It splits the (logical) processors into non-overlapping domains for which this rule applies: "1 MPI process for 1 domain".
  ```
  # mpiexec.hydra ... -env I_MPI_PIN_DOMAIN auto
  # mpiexec.hydra ... -env I_MPI_PIN_DOMAIN omp
  # mpiexec.hydra ... -env I_MPI_PIN_DOMAIN <N>
  ```

- Run app
  ```
  #mpirun.hydra -env I_MPI_PIN_DOMAIN auto -n 2 ./test.MIC
  #mpirun.hydra -env I_MPI_PIN_DOMAIN omp -n 2 ./test.MIC
  #mpirun.hydra -env I_MPI_PIN_DOMAIN 4 -n 4 ./test.MIC
  #mpirun.hydra -env I_MPI_PIN_DOMAIN 8 -n 4 ./test.MIC
  ```
Compile & Run: Hybrid MPI/OpenMP (1)

- Output code added to test code
  
  # diff test.c test-openmp.c

- Compile
  
  # mpiicc -openmp -mmic -o test-openmp.MIC test-openmp.c

- Reduce output
  
  # unset I_MPI_DEBUG # to reduce the output for now

- Set path to shared libraries of OpenMP
  
  # export MIC_LD_LIBRARY_PATH=/opt/software/intel/composerxe/lib/mic

- Run
  
  # mpiexec.hydra -n 2 ./test-openmp.MIC
**Compile & Run: Hybrid MPI/OpenMP (domains)**

- Split logical processors on two domains, size is defined by the formula 
  \[
  \text{size} = \frac{\# \text{cpu}}{\# \text{proc}}
  \]
  
  \[
  \# \text{mpiexec.hydra} -\text{prepend-rank} -\text{env KMP_AFFINITY verbose} -\text{env OMP_NUM_THREADS 8} -\text{env I_MPI_PIN_DOMAIN auto} -n 2 \ ./\text{test-openmp.MIC} 2>&1 | \text{sort}
  \]

- Domain size is equal to the OMP_NUM_THREADS environment variable value.
  
  \[
  \# \text{mpiexec.hydra} -\text{prepend-rank} -\text{env KMP_AFFINITY verbose} -\text{env OMP_NUM_THREADS 8} -\text{env I_MPI_PIN_DOMAIN omp} -n 2 \ ./\text{test-openmp.MIC} 2>&1 | \text{sort}
  \]

- The domain size is defined by a positive decimal number <n>
  
  \[
  \#\text{mpiexec.hydra} -\text{prepend-rank} -\text{env KMP_AFFINITY verbose} -\text{env OMP_NUM_THREADS 8} -\text{env I_MPI_PIN_DOMAIN 8} -n 2 \ ./\text{test-openmp.MIC} 2>&1 | \text{sort}
  \]
• Short description of the **KMP_AFFINITY** and **MIC_KMP_AFFINITY**
  – Set this environment variable to influence thread affinity generally
  – OpenMP programs are affected on CPU and MIC

```
#export KMP_AFFINITY=<type>    (for CPU)
#export MIC_KMP_AFFINITY=<type> (for MIC)
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>compact</td>
<td>Pack threads close to each other.</td>
</tr>
<tr>
<td>explicit</td>
<td>Use the proclist modifier to pin threads</td>
</tr>
<tr>
<td>none</td>
<td>Does not pin threads.</td>
</tr>
<tr>
<td>scatter</td>
<td>Round-robin threads to cores.</td>
</tr>
<tr>
<td>balanced (Phi only)</td>
<td>Use scatter, but keep OMP thread ids consecutive.</td>
</tr>
</tbody>
</table>
Compile & Run: Hybrid MPI/OpenMP
(affinity #2)

#export MIC_KMP_AFFINITY=<type>

- Imagine a system with 4 cores and 4 hardware threads/core
- Placement of 8 threads is illustrated for the 3 types
- Compact type does not fully utilize all cores; not recommended
Compile & Run: Hybrid MPI/OpenMP (affinity #2)

- Use "scatter", "compact", or "balanced" (Intel® Xeon Phi™ coprocessor specific) to modify the default OpenMP affinity

```bash
mpiexec.hydra -prepend-rank -env KMP_AFFINITY verbose,granularity=thread,scatter -env OMP_NUM_THREADS 8 -env I_MPI_PIN_DOMAIN auto -n 2 ./test-openmp.MIC 2>&1 | sort

$mpiexec.hydra -prepend-rank -env KMP_AFFINITY verbose,granularity=thread,compact -env OMP_NUM_THREADS 8 -env I_MPI_PIN_DOMAIN auto -n 2 ./test-openmp.MIC 2>&1 | sort
```
software development, computer vision and robotics