HPC Prototypes

Overview

This wiki contains end user documentation for the CSC HPC Prototype '**Hybrid**', intended for assessing Xeon Phi and GPGPUs.

For support requests, send E-Mail to: **proto-support@postit.csc.fi**

PRACE Spring School 2013 Lectures on Xeon Phi

Table of Contents

System Configuration

The system is a cluster consisting of 10 T-Platforms T-Blade V200 blades.

- Dual 6-core Intel Xeon Sandy Bridge E5-2640
- 32GB 1600MHz RAM
- FDR InfiniBand (Mellanox ConnectX3)
- Accelerator card: Intel Xeon Phi (5 nodes) or Nvidia Kepler (5 nodes)

The node coprocessor/accelerator configuration is as follows:

- **master**: Intel Xeon Phi 5110P (Interactive time-shared frontend node)
- **node[02-05]**: Intel Xeon Phi 5110P
- **node[06-09]**: NVidia Kepler K20
- **node[10]**: NVidia Kepler K20X

The system currently has only local disk. There are two main directories:

- /home/users/[username] - Home directory (limited space!)
- /share/[username] - Work directory

In the near future these directories will be merged.

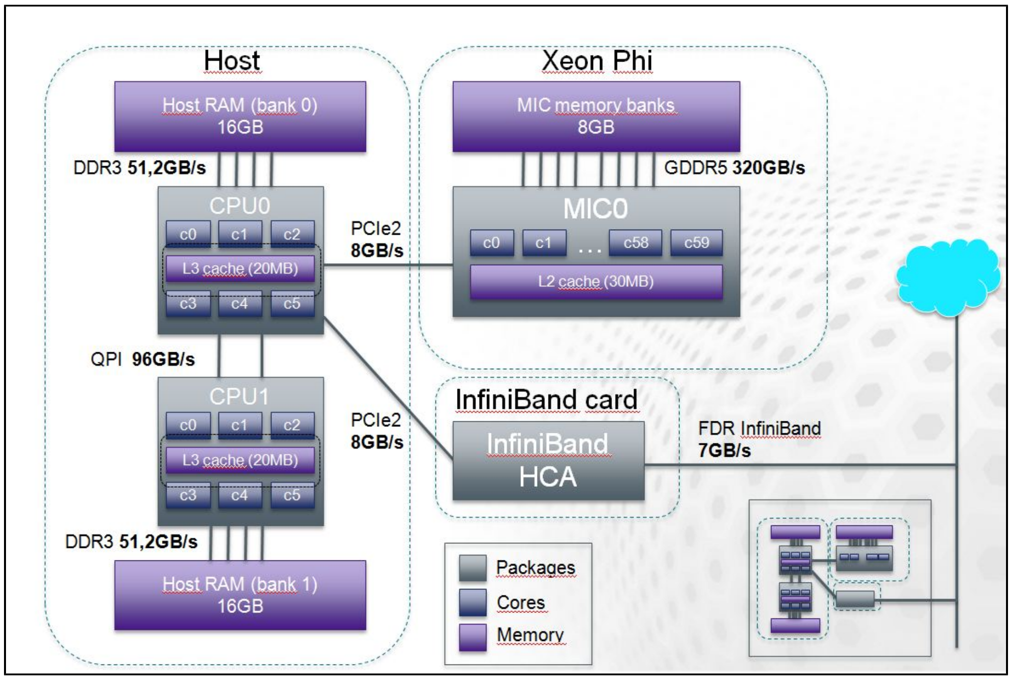
Xeon Phi -specific configuration

Unlike GPGPUs, each Xeon Phi runs a very stripped-down Linux with the busybox shell. The Xeon Phi can be accessed using ssh. Each Phi can be accessed with two hostnames:

- Global hostname: host node name with a **-mic0** suffix (e.g. **node02-mic0**)
- Local hostname: On each node, **mic0** points to the local MIC card of that node

Do not access the MICs of the compute nodes directly to run jobs! Use the batch job queuing system.

Diagram of the node internal topology of a Xeon Phi node



Usage Policy

The system is intended to be used by following groups:

- PRACE 2IP users
- CSC internal users
- CSC advanced pilot users

The system is a prototype, which has the following implications:

- The system may crash or do other weird and possibly interesting things, including skewing results.
- The storage subsystem is not very robust and no backups are performed. Back up your data to avoid grief
- Experiments for the PRACE project may supercede work on a relatively short notice
- Documentation, training and support resources are fairly minimal. We expect users to be skilled in HPC development and be able to troubleshoot things fairly independently.
- The resource is fairly limited so it would be appreciated if job size and duration can be kept as short as possible.
- There are not too many enforced limitations in the queue system right now and it would be nice not to have any. Please behave nicely. :)

Applying for an account

To apply for an account, send the following information to **proto-support@postit.csc.fi**

- User info:
  - If a CSC user: CSC user ID
  - If a PRACE user: Full contact information (name, address, phone, e-mail) and affiliation
- IP address range that will be used by the account-keeper for system access
- Public RSA or DSA key for ssh logins (we do not send plaintext passwords via E-Mail)
- Freeform description of the work to be performed

Logging In

The system can be accessed by logging into **hybrid.csc.fi** using ssh.

When you login for the first time, please create a passphraseless ssh keypair to access the MIC cards:

```
$ ssh-keygen -f $HOME/id_rsa -N ''; cp $HOME/.ssh/id_rsa.pub $HOME/.ssh
/authorized_keys2
```

The OS is CentOS 6.3, which is based on RedHat Enterprise Linux.

To toggle between different compilers, libraries etc. the **modules** command is used. Information on the modules command can be found in the Vuori user guide

Running Jobs

master (frontend node)

The master node (frontend) and it's Xeon Phi coprocessor are time-shared (cannot be reserved by a single user). The master node is intended for code development, compiling and porting. Do not run computationally intensive, memory hungry and/or long running jobs there.

Compute nodes

The batch job queue manager is SLURM. The basic usage is similar to using SLURM on CSC's Taito supercluster or most other commodity clusters.

The following sections (Xeon Phi development and NVidia Tesla develpment) have basic examples on how to run jobs via SLURM and focuses on the special features necessitated by the GPUs and Phis. Please refer to the Taito SLURM documentation for more detailed information about using SLURM (constructing batch job scripts etc.).

In this prototype cluster, SLURM daemons run both on the compute node hosts as well as the Xeon Phi cards. This is an experimental feature still under development and currently unique to the CSC cluster.

There are the following partitions available:

- **cpu**: Contains all nodes
- **michost**: All hosts which have the MIC card - node[02-05] ***This is the default queue***
- **gpu**: All GPUs - node[06-10]
- **k20**: All Kepler K20s - node[06-09]
- **k20x**: The Kepler K20X - node10

On occasion some nodes may be down or drained for experiments. To check node availability, use the **sinfo** command.

Currently all nodes are in the EXCLUSIVE mode. This means only a job reserves a node completely.

Xeon Phi Development

The following tools are available for Xeon Phi:

- Intel Composer XE 2013 (**module load intel**)
- Intel MPI (**module load impi**)
    - to load the Xeon Phi native MPI stack, use **module load impi**
- Intel OpenCL for Phi (**module load intel-opencl**)
- Intel VTune (**module load vtune**)

This document currently describes only the practical usage of the Phi in the prototype system. To learn about developing for Xeon Phi, here are a few starting points:

- PRACE Spring School 2013 Lectures on Xeon Phi
- PRACE Best Practice Guide for Intel Xeon Phi
- Overview of Programming for Xeon Phi (PDF)
- Intel Xeon Phi Developer Pages
- Xeon Coprocessor Performance Programming book (Amazon)

The following types of execution models are supported:

- Offloading (Binary is launched on the CPU host and parts are offloaded to the Phi)
    - Offloading MKL routines

        - Offloading LEO or OpenMP TR routines
- Native (Binary is launched directly on the Xeon Phi)
    - Running a single multithreaded task on Phi
    - Running multiple multithreaded tasks of MPI on multiple Phis
    - Running MPI tasks on both CPU hosts and Phis (Symmetric model)

Executable Auto-Offloading

The Phi nodes have **Executable Auto-Offloading** (EAO) enabled by default. This feature is developed at CSC and is not currently in the standard Xeon Phi distribution.

With this feature, any executable in the K1OM (MIC) binary format that the user tries to run on the host, will transparently be executed on the Xeon Phi coprocessor card instead. The execution is performed using the **/usr/bin/micrun** script.

By default all environment variables with the MIC_ prefix will be passed to the binary, with the prefix stripped away. For example (MIC_LD_LIBRARY_PATH -> LD_LIBRARY_PATH).

EAO can be disabled by setting the environment variable **MICRUN_DISABLE** (i.e. **export MICRUN_DISABLE=1**).

Offload programming model

The Intel compilers support offload compilation automatically. This means either offloading a code section using offload pragmas or calling an offload-enabled library. (e.g. MKL).

In order to run offload jobs, one needs to set the GRES (Generic Resource Scheduling) parameter '**--gres=mic:1**'. For example:

```
$ srun --gres=mic:1 ./hello
```

If this is not set, the user will the following warning:

```
offload warning: OFFLOAD_DEVICES device number -1 does not correspond to a physical device
```

Native programming model

Currently the Intel and GNU compilers support native compilation Phi. However only the Intel compiler can exploit vectorization and should always be used for perfomance-critical code. The GNU compilers can be used to compile non-performance critical support programs and libraries.

We recommend that you use the .mic suffix in binaries to differentiate MIC binaries from x86_64 binaries.

While the instructions here discuss OpenMP, any other pthread-based programming model could potentially be used in a similar way.

Native OpenMP code

To compile OpenMP code natively, you can use the **-mmic**flag.

```
$ module load intel
$ icc -mmic -openmp hello.c -o hello.mic
```

To run, use the **mic** partition, for example:

```
$ srun ./hello.mic
```

Native MPI+OpenMP

To compile:

```
$ module load intel mic-impi
$ mpiicc -mmic -openmp ompmpihello.c -o ompmpihello.mic
```

The execution is performed using a wrapper script using the **michost** queue.

The executable is launched in a SLURM batch job script using the **mpirun-mic -m** *<mic_binary>* command.

The following environment variables and SLURM parameters are used to control the MPI task and OpenMP thread count.

- Environment variables
  - **MIC_PPN** <tasks>   The number of MPI tasks on eeach MIC card
  - **MIC_OMP_NUM_THREADS** <threads>   The number of OpenMP threads per MIC MPI task
- SLURM parameters used to determine the number of nodes
  - **-N** <nodes> The number of nodes (MIC cards) to be used

The following example runs:

- Uses two nodes (*-N 2*)
- On each MIC: **4** MPI processes (*MIC_PPN=4*) with **60** threads each (*MIC_OMP_NUM_THREADS=60*)

```
$ MIC_PPN=4 MIC_OMP_NUM_THREADS=60 srun -N 2 mpirun-mic \
  -m /share/mic/examples/mpiomphello/mpiomphello.mic
```

Symmetric

In the symmetric model, MPI tasks are executed both on the host and on the MIC.

To create a symmetric job, please compile the application for both the host architecture and MIC.

```
$ module load intel impi
$ mpiicc -openmp mpiomphello.c -o mpiomphello
$ mpiicc -mmic -openmp mpiomphello.c -o mpiomphello.mic
```

The execution of symmetric jobs is performed using a wrapper script called **mpirun-mic** using the **michost** queue.

The **mpirun-mic** script takes two parameters:

- **-m** <mic_binary> The MIC executable
- **-c** <host_binary> The CPU host executable

The following environment variables and SLURM parameters are used to control the MPI task and OpenMP thread count.

- Environment variables
  - **MIC_PPN** <tasks>   The number of MPI tasks on the MIC card
  - **MIC_OMP_NUM_THREADS** <threads>   The number of OpenMP threads per MIC MPI task
  - **OMP_NUM_THREADS** <threads>   The number of OpenMP threads per host MPI task
- SLURM parameters are used to determine host tasks and topology in a traditional way
  - **-n** <tasks>   total number host MPI tasks
  - **--tasks-per-node** <tasks>   total number of host MPI tasks per node

The following example runs:

- Uses two nodes (*-n 12 --tasks-per-node 6*)
- On each MIC: **4** MPI processes (*MIC_PPN=4*) with **60** threads each (*MIC_OMP_NUM_THREADS=60*)
- On each host: **6** MPI processes (*-tasks-per-node 6*) with **2** threads (*OMP_NUM_THREADS=2*)

```
$ MIC_PPN=4 MIC_OMP_NUM_THREADS=60 OMP_NUM_THREADS=2 srun -n 12 --tasks-
per-node 6 mpirun-mic
  -m /share/mic/examples/mpiomphello/mpiomphello.mic \
  -c /share/mic/examples/mpiomphello/mpiomphello
```

The tasks will be sorted in consecutive order with CPU tasks first and MIC tasks next for each node. For example, for the above example the ranks would be placed as follows:

Node	Host ranks	MIC ranks
node 1	0-5	6-9
node 2	10-15	16-19

Debugging with Intel IDB

ToDo

Debugging with TotalView

ToDo

NVidia Kepler Development

The following tools are available for NVidia Kepler:

- CUDA SDK (**module load cuda**)
- PGI 13.x compilers with OpenACC (**module load pgi**)

In order to run offload jobs, one needs to set the GRES (Generic Resource Scheduling) parameter '**--gres=gpu:1**'.

```
$ module load cuda
$ srun -p gpu --gres=gpu:1 ./gpuhello
```