SCALABLE HYBRID PROTOTYPE

Scalable Hybrid Prototype

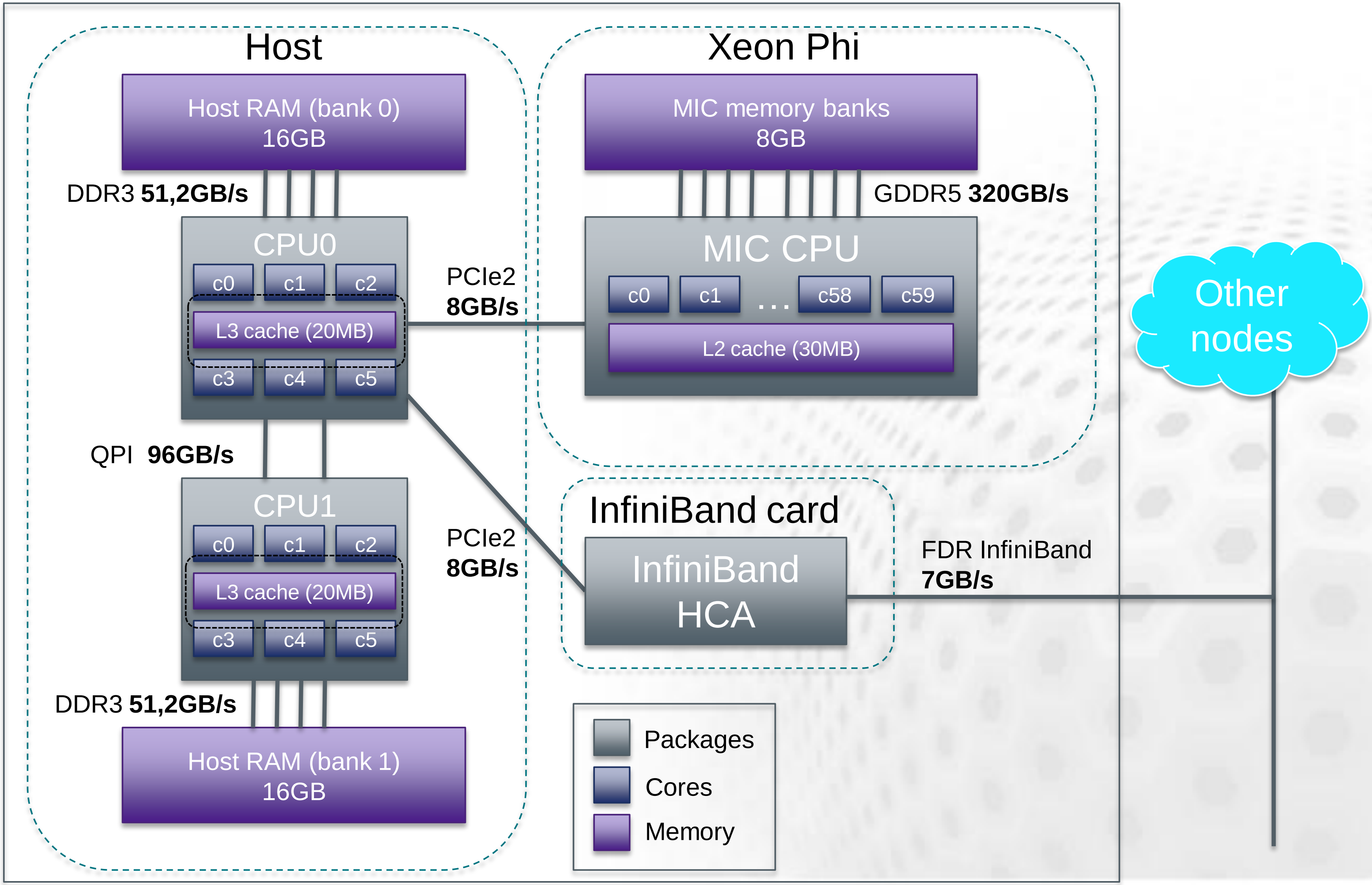
- Part of the PRACE Technology Evaluation
- Objectives
  - Enabling key applications on new architectures
  - Familiarizing users and providing a research platform
  - Whole system benchmarking energy efficiency, productivity and performance
- Located at CSC – IT Center for Science Ltd
  - Espoo, Finland
- Documentation of the system

https://confluence.csc.fi/display/HPCproto/HPC+Prototypes

Current Configuration

- **master –** Head node (frontend)
  - Users login here
  - Program development and test runs
  - Contains a single Xeon Phi for development
  - Freely shared resource: Do not use for heavy computation or performance measurements!
- **node[02-10]** Compute nodes
  - Accessible via the batch job queue system
  - **node[02-05]** Xeon Phi
    - **node[02-05]-mic0** Xeon Phi hostnames
  - **node[06-10]** Nvidia Kepler

Diagram of a Xeon Phi node



First Login

- ssh to **hybrid.csc.fi** with your training account

```
$ ssh -Y hybrid.csc.fi -l trngNN
```

- Create a passwordless host key

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

- Try logging into the MIC card
  – Hostname mic0 or master-mic0

```
$ ssh mic0
```

Using Modules

- Environment-modules package used to manage different programming environment settings
- Examples of usage
  - To load the latest version of Intel compilers, use:

```
$ module load intel
```

  - To see all available modules:

```
$ module avail
```

  - To see what modules are loaded

```
$ module list
```

Custom configuration on Hybrid

- NFS mounts
  - /home, /share, /usr/local
- Additional native support libraries and programs
  - Python, HDF5, gcc etc.
  - Small libraries and utilities (strace etc.)
- SLURM batch job queuing system
- Execution auto-offload on frontend
- Some common paths preset on the Phi
  - i.e. /opt/intel/composerxe/mic/lib64

Execution Auto-offload

- Developed at CSC
  - Implemented in the frontend node
  - Makes e.g. cross-compiling much easier

1. Detects if MIC binary is executed on the host
   - Normally this fails with ”`cannot execute binary file`”
2. Runs the binary on the Xeon Phi using `micrun`
   - Transparent to the end user
   - Environment variables are passed with MIC_ prefix
   - Return values are passed correctly

- Can be disabled by MICRUN_DISABLE=1

SLURM Batch Job Queue System

- Reserves and allocates nodes to jobs
- At CSC we are moving to use SLURM on all systems
  - Designed for HPC from the ground up
  - Open source, extendable, lightweight
  - Becoming increasingly popular in the HPC community
- MIC support in development
  - Offload support in v. 2.5 (Nov 2012)
  - Native/symmetric model via a helper script

SLURM commands

- Checking the queue

```
$ squeue
```

- Checking node status

```
$ sinfo [-r]
```

- Running a job interactively

```
$ srun [command]
```

- Sending a batch job

```
$ sbatch [job script]
```

For simplicity all of the following examples use interactive execution (srun). However for ”real” work you should run batch jobs.

Submitting interactive jobs (srun)

- Interactive shell session

```
$ srun --pty /bin/bash -l
$ hostname
node02
$ exit
$ hostname
master
```

Remember to exit the Interactive session!

- Single thread on MIC

```
$ srun ./omphello.mic
Hello from thread 0 at node02-mic0
```

- Multiple threads on MIC

```
$ export MIC_OMP_NUM_THREADS=2
$ srun ./omphello.mic
Hello from thread 0 at node02-mic0
Hello from thread 1 at node02-mic0
```

All MIC_ prefixed env. variables will be passed to the MIC card

Submitting an Offload Job

- Applicable to LEO, OpenCL, MKL offload …
- Requires the GRES parameter to be used

```
$ srun --gres=mic:1 ./hello_offload
Hello from offload section in node02-mic0
```

  – If you don’t use it, you get a cryptic error

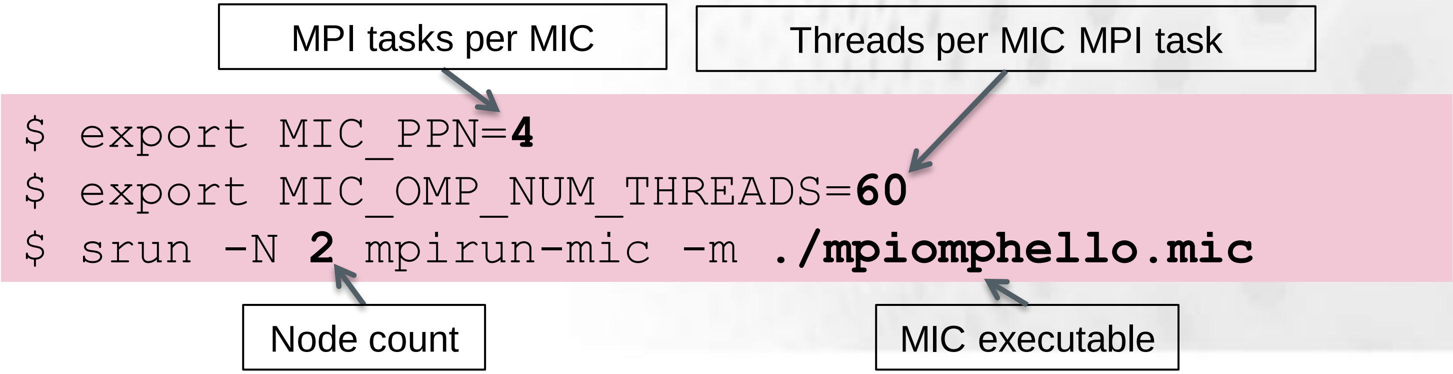
```
$ srun ./hello_offload
offload warning: OFFLOAD_DEVICES device number -1
does not correspond to a physical device
```

- MPI offload job

```
$ srun -n 2 --tasks-per-node 1 ./mpihello_offload
Hello from offload section in node02-mic0
Hello from offload section in node03-mic0
```

Submitting a native MPI job

- MPI tasks only on MIC nodes
- Several parameters must be defined
  - Define # tasks and threads with environment variables
    - `MIC_PPN` and `MIC_OMP_NUM_THREADS`
  - Set number of nodes using `–N` slurm flag
  - Use **mpirun-mic** to launch the executable
    - Use the **–m** flag to specify the MIC executable



MPI tasks per MIC

Threads per MIC MPI task

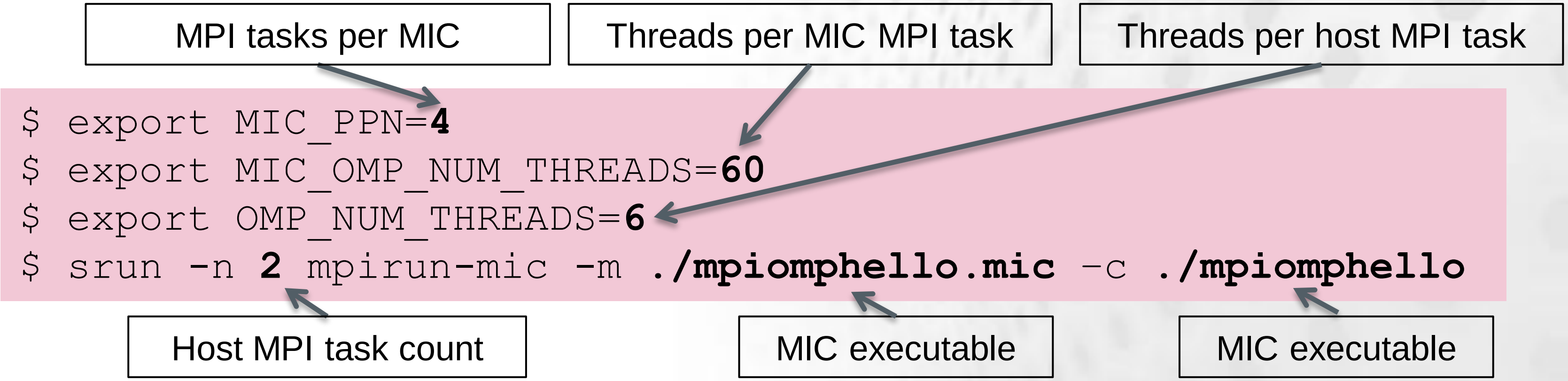
```
$ export MIC_PPN=4
$ export MIC_OMP_NUM_THREADS=60
$ srun -N 2 mpirun-mic -m ./mpiomphello.mic
```

Node count

MIC executable

Submitting a Symmetric Job

- MPI tasks on MIC and host
- Similar to native MPI but some more parameters
  - Define # of host tasks with environment variable
    - `OMP_NUM_THREADS`
  - Use SLURM flags to define # of CPU host tasks
    - For example **`-n`** and **`--tasks-per-node`**
  - Add the executable to the **`mpirun-mic`** command
    - Use the **`-c`** flag to specify the CPU host executable



MPI tasks per MIC

Threads per MIC MPI task

Threads per host MPI task

```
$ export MIC_PPN=4
$ export MIC_OMP_NUM_THREADS=60
$ export OMP_NUM_THREADS=6
$ srun -n 2 mpirun-mic -m ./mpiomphello.mic -c ./mpiomphello
```

Host MPI task count

MIC executable

MIC executable

Further mpirun-mic settings

- The `–v` flag shows the underlying mpiexec command to be run
- The `–h` flag provides help
- You can define additional parameters to the underlying mpiexec –command by setting the following env variables
  - `MPIEXEC_FLAGS_HOST` & `MPIEXEC_FLAGS_MIC`
  - For example:

```
$ export MPIEXEC_FLAGS_HOST=”-prepend-rank \
  -env KMP_AFFINITY verbose”
```

Protip: MIC Environment Variables

- You may want to load a set of environment variables to the MIC card but not on the host
- This might be difficult with a shared home directory
- Put a conditional like this in your `$HOME/.profile` to run MIC-specific environment setup commands

```
if [ `uname -m` == 'k1om' ]; then
    echo I am MIC!
fi
```

Protip: Cross-compiling in Practice

- GNU cross-compiler environment for Phi
  - Located in `/usr/x86_64-k1om-linux`
- Enables building legacy libraries and applications for Xeon Phi
  - In practice it can be difficult
  - Typical build script (usually ./configure) rarely designed with good cross-compiling support
- Requires a varying extent of hand tuning
  - The executable auto offload makes things somewhat easier

Typical Cross-compile on Hybrid

1. Set environment variables to point to cross-compiler and support libraries

```
export LDFLAGS='-L/usr/local/linux-k1om-4.7/x86_64-k1om-linux/lib/ \
-Wl,-rpath=/usr/local/linux-k1om-4.7/x86_64-k1om-linux/lib/’
export CFLAGS="-I/usr/local/linux-k1om-4.7/x86_64-k1om-linux/include”
```

2. Run configure

```
./configure --host=x86_64-k1om-linux [other configure flags]
```

3. Fix linker flags in all **Makefile**s and **libtool**s that are probably incorrect

```
for i in `find -name Makefile` ;do sed -i -e \
's@-m elf_x86_64@-m elf_k1om@' $i;done
for i in `find -name libtool`; do sed -i -e \
's@-m elf_x86_64@-m elf_k1om@' $i;done
```

4. Run make

```
make
```