Introduction to Cori

Steve Leak
NERSC User Engagement Group
July 2017
NERSC Cori

- 34 double-width cabinets
- 9,688 KNL + 2,388 Haswell nodes on Aries High-Speed Network
- 658,784 KNL cores + 76,416 Haswell cores
- Top500 #6 (June 2017)
Agenda

- Cori overview, logging in
- Run a simple job
- Building and running applications on Cori
  - Serial
  - Parallel (MPI)
  - Multithreaded (OpenMP)
- What affects performance?
  - Bottlenecks
  - Task placement and affinity
- Preparing for performance analysis
Cori overview, logging in
Before we start

• The training materials are available on github: https://git.io/v7OuW (and https://git.io/v7LeY)
  — And also on Cori, via:
    cd $SCRATCH
    module load training/csgf-2017
    git clone $TRAINING $SCRATCH/csgf-2017

• The slides are available at (https://www.nersc.gov/users/training/events/csgf-2017-hpc-workshop)

• This is a very hands-on oriented session
  — (so have laptops ready!)

• We will pause for Q&A at the end of each topic
Cori Overview

- Login and compute nodes are distinct
- Large, fast, parallel $\textit{SCRATCH}$ filesystem for running jobs
- Smaller $\textit{HOME}$, configured for building code
- Burst buffer filesystem integrated, on high-speed network
What’s so special about it?

- Your quad-core desktop CPU looks something like this:

- Compared to a Cori Haswell node:

  16 cores x 2 sockets, 128GB RAM

Images courtesy of Google Image Search, Intel, EnterpriseTech.com, NextPlatform.com
What’s so special about it?

- And then there are the KNL nodes (68 cores, MCDRAM, ...)

![Diagram of KNL node layout with highlighting and annotations]
What’s so special about it?

• But high-end CPUs don’t make a supercomputer
  – High speed interconnects between them
  – Lightweight compute node OS
  – Very large (28,000 TB) fast parallel filesystem

• ...and a different usage model
  – Subset of nodes dedicated to a single task, run via batch system (no interactive GUI / desktop)
What’s special about KNL?

• Different choice of compromise between die space allocated to different parts of the CPU

Instruction fetch-and-decode, hyperthreading, branching (important for e.g. compiling, GUI applications)

Arithmetic, vector and floating point units – the actual FLOPS

Memory access – keeps execution engine busy
What’s special about KNL?

- **Xeon (eg Haswell)**
  - Large and fast
  - Power hungry, hot

- **Xeon Phi (KNL)**
  - Small and slow, cool, efficient
  - Big wide vectors
  - Large L2 cache, no L3 (but MCDRAM)
  - Smaller space, shorter vectors
  - Large L3 cache
Why?

- **Exascale challenges – power and heat**
  - CPU frequency plateaued ~15 years ago
  - Transistor density, feature size reaching fundamental limits
  - Power consumption and heat dissipation are now the key constraints for supercomputing
  - ... we can’t get there from here!

- **KNL emphasizes vectorization and parallelism at lower power**
  - Targets scientific computing
• Login and compute nodes are distinct

• Large, fast, parallel $\text{SCRATCH}$ filesystem for running jobs

• Smaller $\text{HOME}$, configured for building code

• Burst buffer filesystem integrated, on high-speed network
Connecting to Cori

• Best: point your browser at https://nxcloud01.nersc.gov and start an NX session
  – Or set up an NX player on your workstation by following the instructions at http://www.nersc.gov/users/connecting-to-nersc/using-nx/

• If you have a UNIX-like computer (or an NX session), you can directly contact NERSC with your built-in SSH client
  1. Open a new terminal
  2. % ssh -Y -l <training_acct_username> cori.nersc.gov

• Many SSH clients exist for Windows
  – A very popular one is putty
    • http://www.putty.org/
  – Advanced users might prefer to use SSH directly within mintty (from Cygwin distribution)
X-forwarding

- Allows you to access GUI programs remotely
- We will need it this afternoon!

Example:

```
localhost% ssh -Y -l elvis cori.nersc.gov
```

```
... e/elvis> module load matlab
e/elvis> matlab
<MATLAB starts up>
```
Example Session

localhost:~elvis$ ssh -Y -l <training_account_name> cori.nersc.gov

*****************************************************************
*                                                               *
*                      NOTICE TO USERS                          *
*                      ---------------                          *
*                                                               *
*  Lawrence Berkeley National Laboratory operates this          *
*  computer system under contract to the U.S. Department of     *
*  Energy. This computer system is the property of the United  *
*  States Government and is for authorized use only. *Users     *
*  (authorized or unauthorized) have no explicit or implicit    *
*  expectation of privacy.*                                     *
*                                                               *
*  Any or all uses of this system and all files on this system  *
*  may be intercepted, monitored, recorded, copied, audited,    *
*  inspected, and disclosed to site, Department of Energy, and   *
*  law enforcement personnel, as well as authorized officials   *
*  of other agencies, both domestic and foreign. *By using      *
*  this system, the user consents to such interception,         *
*  monitoring, recording, copying, auditing, inspection, and    *
*  disclosure at the discretion of authorized site or          *
*  Department of Energy personnel.*                             *
*                                                               *
*  *Unauthorized or improper use of this system may result in   *
*  administrative disciplinary action and civil and criminal    *
*  penalties. _By continuing to use this system you indicate    *
*  your awareness of and consent to these terms and conditions  *
*  of use. LOG OFF IMMEDIATELY if you do not agree to the        *
*  conditions stated in this warning._*                         *
*                                                               *
*****************************************************************

Password: <enter your training account password here>

Prompt on local system
Notification of acceptable use.
Password prompt
After logging in...

- On a login node
  - cori01, cori02, ...
  - Shared by many users
  - Not necessarily the same one each time!
  - But same access to filesystems

- No direct access to compute nodes
  - Only via batch system (salloc, sbatch)

- Haswell (Xeon) architecture
Hands-on exercise

• First: Q&A?

• Exercise: (check README.md at [https://git.io/v7LeY](https://git.io/v7LeY))

1. Log in to Cori
2. Navigate to $SCRATCH
3. Load the module “training/csgf-2017”, this will set $TRAINING to the location of the training materials.
   • Copy (or git-clone) the training materials to your $SCRATCH, and browse the files (especially `ex1-getting_started/README.md`)
4. Is X working? Try to start an xterm
   cori$ xterm &
Agenda

• Cori overview, logging in
• **Run a simple job**
• Building and running applications on Cori
  – Serial
  – Parallel (MPI)
  – Multithreaded (OpenMP)
• **What affects performance?**
  – Bottlenecks
  – Task placement and affinity
• Preparing for performance analysis
Run a simple job
Running jobs – key points

- HPC work is via batch system
  - Dedicated subset of compute resources
  - Login nodes are shared resource for building code, editing scripts, etc. Use batch jobs for real work
- Key commands:
  - sbatch / salloc - submit a job
  - srun - start an (optionally MPI) application within a job
  - sqs - check the queue for my job status
- For today, we have a reservation
  - #SBATCH –reservation=csgftrain

www.nersc.gov/users/computational-systems/cori/running-jobs/
Running jobs – key points

• HPC work is via batch system
  – Dedicated subset of compute resources
  – Login nodes are shared resource for building code, editing scripts, etc. Use batch jobs for real work

• Key commands:
  – sbatch / salloc - submit a job
  – srun - start an (optionally MPI) application within a job
  – sqs - check the queue for my job status

• For today, we have a reservation
  – #SBATCH –reservation=csgftrain

All of this is on the web!

www.nersc.gov/users/computational-systems/cori/running-jobs/
How jobs work

**Desktop / login node**

- Timeslicing
  - core shared by multiple tasks
  - Works when the computer is mostly waiting for you

**HPC**

- You are waiting for the computer
- Subset of pooled resources dedicated to one job
How jobs work

- Start on login node
  - shared by many users, not for computational work
- Access compute nodes with sbatch or salloc
- Batch script
  - Copied to queue
  - Has directives for SLURM, and shell commands to perform on first compute node
- Access your other allocated nodes with srun
- stdout, stderr saved to file
  - (when running in batch mode)
Nodes, cores, CPUs, threads, tasks - some definitions

- **Node** is the basic unit of allocation at NERSC
  - Think “one host” or “one server”
  - Single memory space, multiple CPU cores (24 or 32 or 68 ...)
  - And a core might support hyperthreading
Hyperthreading

- Fast timeslicing
  - Good when arithmetic units frequently wait on memory
- Core holds state of 2 (4 on KNL) processes, they share arithmetic units
- **SLURM views each hyperthread as a CPU**
- But most HPC jobs perform best when not sharing a core!
- Usually best to reserve 2 (or 4) CPUs / core
Slurm tasks

- First, a block diagram of how hyperthreads, cores, cache, and sockets relate within a (haswell) node
• A **SLURM task** is a reservation of CPUs and memory, up to one full node
  – A **job** has many tasks
  – 1 task typically corresponds to 1 MPI rank

srunc -n <ntasks> ..

• Eg: 3 possible tasks on 2 nodes
So what must I request for my job?

What the batch system needs to know:
• How many nodes (or CPUs or tasks) does this job need?
• For how long does it need them?
  – Wallclock time limit

NERSC-specific extras:
• What type of CPU? (-C ...)
  – KNL or Xeon (haswell/ivybridge)?
• Which filesystems will this job use? (-L ...)
  – Usually SCRATCH
#SBATCH -N 64        # request 64 nodes
srun -N 32 ./my_app  # start ./my_app on 32 of them
# (default: 1 per node)
srun -n 128 ./my_app # start 128 instances of ./my_app,
# across my 64 nodes (default is
# to evenly distribute them in
# block fashion)

One MPI rank generally corresponds to one SLURM Task
Requesting time

#SBATCH -t 30       # 30 minutes
#SBATCH -t 30:00    # 30 minutes
#SBATCH -t 1:00:00  # 1 hour
#SBATCH -t 1-0      # 1 day
#SBATCH -t 1-12     # 1.5 days

• Wallclock time, ie real elapsed time
• After this much time, SLURM can kill this job
Hands-on exercise: My first job

A SLURM job script has two sections:

1. Directives telling SLURM what you would like it to do with this job
2. The script itself - shell commands to run on the first compute node

For how long?
How many nodes?

```
#!/bin/bash -l

#SBATCH -t 00:30:00
#SBATCH -N 2
#SBATCH --license=SCRATCH

export RUNDIR=$SCRATCH/run-$SLURM_JOBID
mkdir -p $RUNDIR
cd $RUNDIR

srun -n 4 bash -c 'echo "Hello, world, from node $(hostname)"

elvis@nersc:~> sbatch -C $CRAY_CPU_TARGET myscript.q
Submitted batch job 2774102
```

Note: cannot use env vars in directives, but directives have equivalent command-line options
Hands-on exercise: My first job

A SLURM job script has two sections:

1. Directives telling SLURM what you would like it to do with this job
2. The script itself - shell commands to run on the first compute node

Make starting environment like my login environment

Run from $SCRATCH

Start 4 tasks across my nodes

```
#!/bin/bash -l

#SBATCH --time=00:30:00
#SBATCH --nodes 2
#SBATCH --license=SCRATCH

export RUNDIR=$SCRATCH/run-$SLURM_JOBID
mkdir -p $RUNDIR
cd $RUNDIR

srun -n 4 bash -c 'echo "Hello, world, from node $(hostname)"'
```

```
elvis@nersc:~> vi myscript.q

elvis@nersc:~> sbatch -C CRAY_CPU_TARGET myscript.q
Submitted batch job 2774102
```

“sbatch” submits a job script
Running jobs – key points

- HPC work is via batch system
  - Dedicated subset of compute resources
  - Login nodes are shared resource for building code, editing scripts, etc. Use batch jobs for real work

- Key commands:
  - `sbatch / salloc` - submit a job
  - `srun` - start an (optionally MPI) application within a job
  - `sqs` - check the queue for my job status

- Don’t forget we have a reservation
  - `#SBATCH --reservation=csgftrain`

www.nersc.gov/users/computational-systems/cori/running-jobs/
### Where is my job?

```
where@nersc:~> sqs

<table>
<thead>
<tr>
<th>JOBID</th>
<th>ST</th>
<th>REASON</th>
<th>USER</th>
<th>NAME</th>
<th>NODES USED</th>
<th>REQUESTED ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>2774102</td>
<td>R</td>
<td>Prolog</td>
<td>elvis</td>
<td>myscript.q</td>
<td>2</td>
<td>0:00 30:00</td>
</tr>
</tbody>
</table>

... 

```

```
where@nersc:~> ls -lt

```

```
total 11280
-rw-r----- 1 elvis elvis 132 Nov 18 11:24 slurm-2774102.out
-rw-r----- 1 elvis elvis 208 Nov 18 11:24 myscript.q
```

---

This diagram illustrates the lifecycle of a job on a supercomputer, with the following key stages:

- **Pending (PD)**: Job submitted, waiting in queue
- **Configuring (CF)**: Nodes are being prepared to run the job
- **Running (R)**: Processes are being killed, nodes being returned to pristine state
- **Completing (CG)**: Job is completed and placed in the completed state
- **Completed (CD)**: Job is marked as completed
- **Cancelled (CA)**: Job is cancelled
- **Timeout (TO)**: Job times out
- **Failed (F)**: Job fails
- **Node Fail (NF)**: Node fails

```
sqs
squeue
scontrol

sstat

sacct
```
Hands-on exercise

• First: Q&A ?

• Exercise: running a simple job
  – (check README.md in ex2-running_jobs/)
Agenda

• Cori overview, logging in
• Run a simple job
• Building and running applications on Cori
  – Serial
  – Parallel (MPI)
  – Multithreaded (OpenMP)
• What affects performance?
  – Bottlenecks
  – Task placement and affinity
• Preparing for performance analysis
Building and running applications on Cori
• Building code optimally for Cori requires a complex set of compiler option and libraries
  – eg, static linking by default (important for performance at scale)

• Compiler wrappers ftn, cc and CC manage this complexity for you
  – Using environment variables set by the modules you have loaded

• Also provide MPI (so eg mpicc is not required)
Wait .. “environment modules” ?

- Software on Cori (and most HPC systems) is managed with “environment modules”

- Why?
  - Cori is a shared resource
  - Different people need different combinations of software, at different versions, with different dependencies (and for different jobs)

- Loading and unloading a module updates environment variables (eg $PATH, $LD_LIBRARY_PATH) to make a package available
Module Commands

module load <modulename>
  – Add the module from your environment

module unload <modulename>
  – Remove the module from your environment

module swap <module1> <module2>
  – Unload one module and replace it with another
    % module swap intel intel/16.0.3.210
    (replace current default to a specific version)

module list
  – See what modules you have loaded right now

module show <modulename>
  – See what the module actually does

module help <modulename>
  – Get more information about the software
Key modules for compiling

- **PrgEnv-intel / PrgEnv-cray / PrgEnv-gnu**
  - Which underlying compiler the wrappers should invoke
- **craype-haswell / craype-mic-knl**
  - Remember, login nodes are haswell, but we are building for KNL!

```
module swap craype-haswell craype-mic-knl
```

- Wrappers manage cross-compiling

**Important for today!**
What do compiler wrappers link by default?

- Depending on the modules loaded, MPI, LAPACK/BLAS/ScalAPACK libraries, and more
Compiling code

• Very similar to regular Linux, but using CC / cc / ftn

• Do this bit once:
  module swap craype-haswell craype-mic-knl

• Then:
  ftn -c hack-a-kernel.f90
  ftn -o hack-a-kernel.exe hack-a-kernel.o

• Note that the module looks after CPU target!
Compiling parallel code

- Compiler wrappers give you MPI “for free”
  
  `CC -c hello-mpi.c++`
  
  `CC -o hello-mpi.ex hello-mpi.o`  

- (Cray MPICH – optimized for Aries HSN)

- OpenMP: with PrgEnv-intel (NERSC default):
  
  `cc -qopenmp -c hello-omp.c`
  
  `cc -qopenmp -o hello-omp.ex hello-omp.o`
MPI vs OpenMP

• MPI provides explicit communication between separate processes
  – Optionally on separate nodes – ie packets over a network
  – Most parallel development in last 2 decades has used this approach

• OpenMP provides work-sharing and synchronization between threads in a single process
  – Threads share the same memory image
  – To make the most of a KNL node, most applications will need to use OpenMP
MPI vs OpenMP

• An MPI application can have processes on more than one node

• An OpenMP application exists entirely within 1 node
Why do we suddenly need OpenMP?

• Multi-level parallelism
  – At very large scale, the overheads of MPI (or any parallel approach) become excessively costly
  – Combining (nesting) parallel approaches allows us to operate each at lower scale
    • Sweet spot for best overall efficiency
  – MPI -> OpenMP -> Vectorization
Why do we suddenly need OpenMP?

- **Memory-per-core is trending downwards**
  - Cori Haswell: 128GB for 32 cores
  - Cori KNL: 96GB for 68 cores (16GB MCDRAM for 68 cores)
  - Parallelism within same memory footprint is necessary
Hands-on exercise

• First: Q&A?

• Exercise: building and running a simple serial, OpenMP and MPI application
  – (check README.md in ex3-building_apps)
Agenda

• Cori overview, logging in
• Run a simple job
• Building and running applications on Cori
  – Serial
  – Parallel (MPI)
  – Multithreaded (OpenMP)

• What affects performance?
  – Bottlenecks
  – Task placement and affinity

• Preparing for performance analysis
What affects performance?
Performance bottlenecks

• Any point at which some component of the system or application is stalled, waiting on some other component, is a bottleneck

• Eg waiting for MPI call to complete
  – Load imbalance or communication overhead is a bottleneck

• BUT for today we are interested in KNL specifics
  – Bottlenecks within the node
• Firstly: Am I running it right?
  – Cori KNL has 68 cores per node, arranged on a mesh of 34 tiles
  – Each tile has 2 cores and a shared L2 cache
  – Each core has 4 hyperthreads sharing 2 VPUs and an L1 cache
Bottlenecks within the node – Affinity issues

- Where are my threads?
  - Is each using a different core at least?

- Linux does not always choose best placement
  - Use srun options to ensure optimal thread/process placement
Solution: use `--cpu_bind`:
```
srun -n  64 -c 4 --cpu_bind=verbose,cores   ./my_exec
srun -n 128 -c 2 --cpu_bind=verbose,threads ./my_exec
```

- Controls what a task (MPI rank) is bound to
  - If no more than 1 MPI rank per core: `--cpu_bind=cores`
    - ![Diagram showing task allocation when `--cpu_bind=cores`]
  - If more than 1 MPI rank per core: `--cpu_bind=threads`
    - ![Diagram showing task allocation when `--cpu_bind=threads`]

Thread affinity (OpenMP)

export OMP_NUM_THREADS=2
export OMP_PROC_BIND=spread     # or close
export OMP_PLACES=cores         # or threads, or sockets
srun -n 32 -c 8 --cpu_bind=verbose,cores   ./my_exec

...If using hyperthreads, use OMP_PLACES=threads
Memory affinity

Linux default behavior is to allocate to closest NUMA-node, if possible

Not always optimal:

- KNL nodes: DDR is “closer” than MCDRAM

```bash
#SBATCH -C knl,quad,flat
export OMP_NUM_THREADS=4
srun -n16 -c16 --cpu_bind=cores --mem_bind=map_mem:1 ./a.out
```

- NUMA node 1 is MCDRAM in quad, flat mode
- “Mandatory” mapping: if using >16GB, malloc will fail

NOTE: today’s reservation is for “cache-mode” nodes, so MCDRAM is invisible
**Bottlenecks within the node – Affinity issues**

`OMP_PROC_BIND=true`

`OMP_PLACES=cores (threads)`

`srun -c 4 --cpu_bind=cores ...`

Diagram:
- `srun -c 4 --cpu_bind=cores` (reserve 4 cpus - i.e. 1 core - per SLURM task / MPI rank)
- `srun -c 2 --cpu_bind=threads` (reserve 2 cpus - i.e. half of one core - per SLURM task / MPI rank)
- `OMP_PLACES=threads` (each OMP thread is assigned to one hyperthread)
Bottlenecks within the node

- Performance tends to be dominated by:
  - Doing arithmetic operations (# FLOPS/cycle)
  - Moving data between arithmetic units and memory

- Bottlenecks are usually due to one of these parts waiting on the other
Bottlenecks within the node

- **Bandwidth!**
  - MCDRAM is very high bandwidth (~450 GB/s)
  - But aggregate bandwidth is not the same as bandwidth to each core!
• **First:** Q&A?

• **Exercise:** Getting the affinity settings right
  
  – (check README.md in ex4-affinity)
  
  – We have two jobs in this exercise, the first is fast and sufficient to demonstrate the effect of affinity settings. The second will take longer, so we’ll continue with the presentation without waiting for it to complete.
Agenda

- Cori overview, logging in
- Run a simple job
- Building and running applications on Cori
  - Serial
  - Parallel (MPI)
  - Multithreaded (OpenMP)
- What affects performance?
  - Bottlenecks
  - Task placement and affinity
- Preparing for performance analysis
Preparing for performance analysis
A small, short, but representative test case for your application

- Profiling tends to be costly
  - Runtime overhead
  - Size of data collected

- **BUT**: must cover same paths through code as “real” example (or at least, the differences must be understood)

- What could go wrong with this test case?
...But my application just is big!

• Start with a low-overhead profiling method
  – Eg sampling-based (gprof, TAU, CrayPat, ...)
  – Identify hotspots

• Only profile part of a run
  – Some tools (eg Vtune) allow you to start the run “paused” and “resume collection” via an API call

• Only profile 1 MPI rank
  – Via srun options, eg run Vtune on one rank, not others (beyond today’s scope)

• Luckily, our hack-a-kernel is already small! 😊
Today’s exercise

• Hackathon this afternoon: Using Vtune to analyze hack-a-kernel performance on KNL

• Vtune: powerful tool, but very information-dense
  – Best suited to node-level performance analysis (OpenMP, vectorization, memory bandwidth .. Not really MPI)
  – Many knobs to turn
  – Command-line interface: used at “collect”
    • Optionally can use to “report” too
  – GUI interface: explore performance reports
Performance Analysis Gotchas

• **Overhead!**
  – Your job will (usually) take longer while profiling

• **Performance tools often don’t play nicely together!**
  – Trying to use same resources

• **Especially (at NERSC):**
  – Darshan is loaded by default, collects I/O performance data
  – Must unload it before using Vtune (and most other performance tools)
Performance Analysis Gotchas

• Many tools require dynamically-linked executable
  – Including Vtune
  – NERSC/Cray: applications are statically linked by default, must use “-dynamic” at compile time

• “Uncore” (eg memory related) performance counters usually require special permissions (eg kernel module)
  – NERSC/Slurm supports this with #SBATCH –perf=vtune directive
Hands-on exercise

- This afternoon: we will optimize hack-a-kernel for KNL
- Exercise: First, we’ll build it for Vtune and run an experiment, so we have results ready to look at this afternoon
  - check README.md in ex5-vtune)
Summary and Recap

• **Now you can:**
  – Log in
  – Build an application
  – Run an application, and be aware of how and where it is running
  – Prepare an application for performance analysis with Vtune

• **Q & A?**