Development cycle 2/2: Run

C. Berthelot
Christophe.Berthelot@atos.net

Copyright © Bull S.A.S. 2016
Agenda

- Introduction
- MPI
- Thread Affinity
- Slurm
- Demo
- Questions
Introduction

Processor

- **Sockets** The socket refers to a physical connector on a computer motherboard that accepts a single physical chip. On compute node there are 2 sockets

- **Cores** Essentially a core comprise a logical execution unit containing an L1 cache and functional units. Cores are able to independently execute programs or threads. On a SNB sockets there are 8 cores

- **Thread** A thread is essentially a process that does not have a full stack of memory associated for it. On a SNB core there are 2 threads

- **Hyperthreading** Hyperthreading is a technology that preceded multi-core systems in which a single core would logically appear as multiple cores on the same chip.

- **Processor** Processor could describe either a single execution core or a single physical multi-core chip.
Introduction

Data placement policy

First touch
- First processor to touch a page of memory causes it to be allocated from its local memory
- Works well for fully parallelized programs, but serial initialization cause non-local accesses and bottlenecks on memory bandwidth.
Introduction

Memory placement : Numactl

Numactl runs processes with a specific NUMA scheduling or memory placement policy. The policy is set for command and inherited by all of its children. Policy settings are:

- Memory will be allocated using round robin on nodes.
- Allocates only memory from nodes.
- Does always local allocation on the current node.
Introduction

Commands

- `--interleave=nodes, -i nodes`: Set a memory interleave policy.
- `--membind=nodes, -m nodes`: Only allocate memory from nodes.
- `--localalloc, -l`: Do always local allocation on the current node
- `--cpubind=nodes, -c nodes`: Only execute process on the CPUs of nodes
Introduction

Example: CG from NPB 4 CPUs

All on first socket

```
numactl --membind=0 --cpunodebind=0 ./cg.B
```

- Time: 27.50s

CPUs on socket 0, memory on socket 1

```
numactl --membind=0 --cpunodebind=1 ./cg.B
```

- Time: 37.11s
MPI

- Introduction
- MPI
  - Intel
  - Bull
- Thread Affinity
- Slurm
- Demo
- Questions
Introduction

- Intel MPI: based on mpich2
- To compile with Intel compilers: mpiicc; mpiifort, mpiicpc
- To compile with Gnu compilers: mpif90/f77 mpicc/CCC
Tuning areas

- Interconnect fabrics
- Process placement and pinning
- Point-to-Point communication
- Collective algorithms

Tuning: step by step

- Ensure the cluster is sane and you built the application with high optimization level
- Try to use automatically tune Intel MPI
- Learn your application communication with stats
- Use MPI Runtime options and environment variables for application tuning
Use best available communication fabric

Use default fabrice selection if you can and check with $I\_MPI\_DEBUG$ set to 2.

<table>
<thead>
<tr>
<th>$I_MPI_DEVICE$</th>
<th>$I_MPI_FABRICS$</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sock</td>
<td>tcp</td>
<td>TCP/IP-capable network fabric, such as Ethernet and InfiniBand (IPOIB)</td>
</tr>
<tr>
<td>shm</td>
<td>shm</td>
<td>Shared Memory + TCP/IP</td>
</tr>
<tr>
<td>ssm</td>
<td>shm:tcp</td>
<td>Shared-memory+TCP/IP</td>
</tr>
<tr>
<td>rdma</td>
<td>dapl</td>
<td>DAPL-capable network fabrics through DAPL</td>
</tr>
<tr>
<td>rdssm</td>
<td>shm:dapl</td>
<td>Shared-memory + DAPL+sockets</td>
</tr>
<tr>
<td></td>
<td>ofa</td>
<td>OFA-capable network fabrics including InfiniBand through OFED verbs</td>
</tr>
<tr>
<td></td>
<td>tmi</td>
<td>TMI network fabrics including Qlogic</td>
</tr>
</tbody>
</table>

- Select network interface for socket communication IP over IB
  - $I\_MPI\_TCP\_NETMASK=ib0$ for IP over IB
  - $I\_MPI\_NETMASK=...$ for particular subnet

$ mpiexec $env $I\_MPI\_DEBUG 2 $env $I\_MPI\_FABRICS shm:dapl $env $n <# processes> ./a.out
MPI

Use connectionless communication

- Connectionless feature works for DAPL fabrics only
- Works with OFED 1.4.2 and 2.0.24 or higher
- Provides better scalability
- Reduces memory requirements

```bash
$export I_MPI_FABRICS=dapl (or shm:dapl)
$export I_MPI_DAPL_UD=enable (DAPL User Datagrams)
```
Use proper process pinning

- Use `I_MPI_PIN_PROCESSOR_LIST` to define custom map of MPI processes to CPU cores pinning. Use `cpuinfo` utility supplied with Intel MPI library to see the processor topology.

---

**Intel(R) processor family information utility, Version 5.1.1 Build 20150715 (build id: 12269)**

==== Processor composition =====

Processor name : Intel(R) Xeon(R) E5-2680 v3
Packages(sockets) : 2
Cores : 24
Processors(CPUs) : 48
Cores per package : 12
Threads per core : 2

==== Processor identification =====

<table>
<thead>
<tr>
<th>Processor</th>
<th>Thread Id.</th>
<th>Core Id.</th>
<th>Package Id.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>
**MPI**

**Use proper process pinning**

- To pin the processes to the CPU0 and CPU3, use sequential identifier starting to zero
  
  ```
  mpirun -genv I_MPI_PIN_PROCESSOR_LIST=0,3 -n <procs> ./a.out
  ```

- To place consecutive MPI processes to cores sharing L2 cache and occupy different physical package for consecutive pairs of processes, use `grain=cache2,shift=sock`

  ```
  mpirun -genv I_MPI_PIN_PROCS='grain=cache2,shift=sock' -n <procs> ./a.out
  ```
Use proper process pinning with OpenMP

- I_MPI_PIN_DOMAIN. Options can be

<table>
<thead>
<tr>
<th>Options</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>core</td>
<td>Each domain consists of the logical processors that share a particular core</td>
</tr>
<tr>
<td>socket</td>
<td>Each domain consists of the logical processors that share a particular socket</td>
</tr>
<tr>
<td>node</td>
<td>All logical processors on a node are arranged into a single domain</td>
</tr>
<tr>
<td>cache1</td>
<td>Logical processors that share a particular level 1 cache are arranged into a single domain</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>omp</td>
<td>The domain is equal to the OMP_NUM_THREADS value</td>
</tr>
<tr>
<td>auto</td>
<td>The domain is equal #logical-cpu-on-node/#mpi-task</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Choose the best collective algorithms

- Use one of the `I_MPI_ADJUST_<opname>` to change the algorithm. See section 3.5 Intel MPI (section 3.5 Collective Operation Control of Intel MPI Reference Manual). For example `I_MPI_ADJUST_ALLREDUCE` will controls MPI_Allreduce algorithm:
  - 1 - Recursive doubling algorithm
  - 2 - Rabenseinfner’s algorithm
  - 3 - Reduce & Bcast
  - 4 - Topology aware Reduce
  - 5 - Binominal gather & scatter algorithm
  - 6 - Topology aware binominal gather & scatter
  - 7 - Ring algorithm

- Recommendations:
  - Focus on the most critical collective operations (see stats)
  - Run Intel MPI Benchmarks selecting various algorithms to find out the right protocol.
Introduction

- Bull MPI: based on openmpi
- module load bullxmpi

Basic

- To compile with intel compilers mpif90/f77 mpicc/CC
MPI

- As usual for MPI stacks, bindings for several languages: C, F77, F90, C++
- Wrappers are provided with the library mpicc, mpif77, mpif90, mpic++
- Used as the usual compilers
- All the flags / only compiler flags / only linker flags can be viewed –showme | -showme:compile | -showme:link
- Basic Example : mpicc hello.c -o hello

Overriding BullxMPI wrapper compilers flags

<table>
<thead>
<tr>
<th></th>
<th>Preprocessor flags</th>
<th>Compiler Flags</th>
<th>Linker Flags</th>
<th>Linker lib</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>OMPI_CPPFLAGS</td>
<td>OMPI_CFLAGS</td>
<td>OMPI_LDFLAGS</td>
<td>OMPI_LIBS</td>
</tr>
<tr>
<td>C++</td>
<td>OMPI_CPPFLAGS</td>
<td>OMPI_CXXFLAGS</td>
<td>OMPI_LDFLAGS</td>
<td>OMPI_LIBS</td>
</tr>
<tr>
<td>F77</td>
<td>OMPI_CPPFLAGS</td>
<td>OMPI_FFLAGS</td>
<td>OMPI_LDFLAGS</td>
<td>OMPI_LIBS</td>
</tr>
<tr>
<td>F90</td>
<td>OMPI_CPPFLAGS</td>
<td>OMPI_FCFLAGS</td>
<td>OMPI_LDFLAGS</td>
<td>OMPI_LIBS</td>
</tr>
</tbody>
</table>
MPI

Define process mapping and binding

Mapping: How ranks are distributed on nodes
- Use successive cores (default): `mpirun --bycore`
- Cycle on successive sockets: `mpirun --bysocket`
- Cycle on successive nodes: `mpirun --bynode`

Binding: How ranks are affected to core, processor ...
- Bind to a single core (default): `mpirun --bind-to-core`
- Bind to a entire socket: `mpirun --bind-to-socket`
- Do not bind, bind to a node: `mpirun --bind-to-none`
- Custom bindings: `mpirun --cpus-per-rank`
- View the binding: `mpirun --report-bindings`
MPI

BullxMPI tuning: mca parameters are the key

Introduction

- Most of BullxMPI behaviours are customizable through mca parameters
- List of mca parameters:
  - Simple: ompi_info
  - Detailed: ompi_info -a
- Set an mca parameter:
  - Mpirun way: mpirun -mca mycma mymavalue
  - Environment (srun) way: export OMPI_MCA_mymca=myvalue
- For convenience examples will be presented the mpirun way
mca example: Choose communication device

List of common btl's
- self: communicate with myself (mandatory)
- sm: use shared memory (intra node communication)
- openib: communicate through infiniband device (inter node)
- tcp: communicate through tcp device (inter node)

How to
- Exclude a device: mpirun --mca btl ^btl_to_exclude
- Manually set device list: mpirun --mca btl self,btl1,btl2
- Change device priority: mpirun--mca btl_btlx_priority 10 (if priority is set to 0 device is excluded)
MPI

BullxMPI tuning

- **btl_openib_use_eager_rdma**: use RDMA for eager messages. *lower latency* BUT *higher memory footprint*
- **btl_openib_eager_limit**: size of eager eager messages
- **mpi_leave_pinned**:
  - User buffers are left registered (decreases de/(re)-registration costs)
  - BUT application should re-use the exactly the same send buffers
  - With IMB pingpong it allows to reach the maximum bandwidth
  - This variable will used by default in the next bullxmpi version (1.2.1.1)
To Improve collective performance

- `coll_tuned_use_dynamic_rules` Switch used to decide if we use static (compiled/if statements) or dynamic (built at runtime) decision function rules.

- `coll_tuned_alltoall_algorithm` parameter select which alltoall algorithm is used. Can be locked down to choice of:
  - 0: ignore,
  - 1: basic linear,
  - 2: pairwise,
  - 3: modified bruck,
  - 4: linear with sync,
  - 5: two proc only.

- `coll_tuned_allreduce_algorithm` Which allreduce algorithm is used. Can be locked down to any of:
  - 0: ignore,
  - 1: basic linear,
  - 2: non overlapping (tuned reduce + tuned bcast),
  - 3: recursive doubling,
  - 4: ring,
  - 5: segmented ring
MPI

- Introduction
- MPI
  - Intel
  - Bull
- Thread Affinity
- Slurm
- Demo
- Questions
Thread Affinity

Get Info on the System Topology

Before all, you should have a basic understanding of the system topology:

▶ Intel MPI’s cpuinfo tool
  • Delivers information about the number of sockets or packages and the mapping of processor ids used by the operating system to cpu cores

▶ OpenMPI’s hwloc-ls or lstopo tool
  • Display a graphical representation of the system topology, separated into NUMA nodes

Binding Strategy

▶ Putting threads far apart (i.e. on different sockets)
  • May improve the aggregated memory bandwidth available to your application
  • May improve the combined cache size available to your application
  • May decrease performance of synchronization constructs

▶ Putting threads close together (i.e. on two adjacent cores which possibly shared some caches)
  • May improve performance of synchronization constructs
  • May decrease the available memory bandwidth and cache size
Thread Affinity

Intel Compiler

Use environment variable **KMP_AFFINITY**
- KMP_AFFINITY=scatter: Put threads far apart
- KMP_AFFINITY=compact: Put threads close together
- KMP_AFFINITY=<core_list>: Bind threads in the order in which they are started to the cores given in the list, one thread per core
- Add verbose to print out binding information to stdout

Gnu Compiler

Use environment variable **GOMP_CPU_AFFINITY**
- GOMP_CPU_AFFINITY=<core_list>: Bind threads in the order in which they are started to the cores given in the list, one thread per core.
Thread Affinity

- Better support of affinity
- Can be used to get better locality, less false sharing and more memory bandwidth
- Define OpenMP Place: Variable `OMP_PLACES` to specify platform-specific data
  - set OpenMP threads running on one or more processors
  - can be defined by the user
  - pre-defined place:
    - `threads`: one place per hyper-thread
    - `cores`: one place exists per physical core
    - `sockets`: one place per processor package
- Define set of OpenMP thread affinity policies
  - New clause for omp parallel: `proc_bind`. Possibles values: `false, true, master, close, spread`
  - SPREAD: spread OpenMP threads evenly among places
  - CLOSE: pack OpenMP threads near master thread
  - MASTER: collocate OpenMP thread with master thread
  - `OMP_PROC_BIND`
- Goals: locality between OpenMP Threads/less false sharing/memory bandwidth
Slurm

- Introduction
- MPI
- Thread Affinity

**Slurm**
- Basic Usage
- Advanced Usage
- New features SLURM

- Demo
- Questions
Slurm

Basic Usage

User & Admin Commands

- `sinfo` display characteristics of partitions
- `squeue` display jobs and their state
- `scancel` cancel a job or set of jobs.
- `scontrol` is the administrative tool used to view and/or modify SLURM state. But user can also run some commands like
  - `scontrol show partition`
- `sstat` show status of running jobs.
Slurm

Basic Usage

Admin Commands

- sacctmgr setup accounts, specify limitations on users and groups.
- sreport display information from accounting database on jobs, users, clusters.
- sview graphical view of cluster. Display and change characteristics of jobs, nodes, partitions.
## Slurm

### Basic Usage

#### Useful slurm environment variables

<table>
<thead>
<tr>
<th>Environment variables</th>
<th>Correspondence</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLURM_JOB_ID</td>
<td>Job id</td>
</tr>
<tr>
<td>SLURM_NTASKS</td>
<td>#SBATCH -n</td>
</tr>
<tr>
<td>SLURM_JOB_NUM_NODES</td>
<td>#SBATCH -N</td>
</tr>
<tr>
<td>SLURM_CPUS_PER_TASK</td>
<td>#SBATCH -c</td>
</tr>
<tr>
<td>SLURM_ACCOUNT</td>
<td>#SBATCH -A</td>
</tr>
<tr>
<td>SLURM_JOB_NODLIST</td>
<td>Nodelist which is allocated</td>
</tr>
</tbody>
</table>

```bash
#!/bin/bash
#SBATCH -A <account name>
#SBATCH -J test
#SBATCH -N 2
#SBATCH -n 32
#SBATCH -t 00:10:00
#SBATCH -o test.%j.out
#SBATCH -e test.%j.err
NP=$SLURM_NTASKS
echo "jobid: $SLURM_JOB_ID"
echo "numnodes: $SLURM_JOB_NUM_NODES"
echo "nodelist: $SLURM_JOB_NODLIST"
mpirun -np $NP ./a.out
```
SLURM Resource Management

- Fine Resource Management through submission command parameters
  - Adapted for MultiCore/MultiThread Architectures
  - Able to treat all kind of consumable resources (nodes, CPU hierarchies, memory, GPUs, etc)

- Based upon plpa/hwloc or Cpusets for CPUs confinement and optional tasks binding upon particular CPUs
Background

- HPC cluster resource managers originally managed CPU resources in units of whole nodes.
- With the growth of multi-socket, multi-core, multi-thread hardware architectures, it became necessary to manage individual CPU resources within nodes to improve resource utilization efficiency, job performance and system throughput.
- To that end, two new resource management plugins were developed for SLURM:
  - The **select/cons_res plugin** allows sockets, cores and threads to be allocated individually as consumable resources within nodes. This can enhance throughput by allowing jobs to share nodes.
  - The **task/affinity plugin** allows tasks to be bound to specific sets of CPUs within a node. This can enhance performance by increasing the utilization of CPU cache and local memory.
Basic Concepts

- Physical CPU Resource Types (Nodes, Sockets, Cores, Threads)
- Physical CPU vs. Logical CPU (Node definitions, FastSchedule option in slurm.conf)
- Allocation Units (Whole Nodes, Consumable Resources)
- Exclusive Allocation vs. Shared Allocation (CPU Sharing Between Jobs, CPU Sharing Between Tasks of the Same Job)
- Allocation and Distribution Methods (Block, Cyclic)
- Optional Task-to-CPU Binding (Task/affinity plugin, Task/cgroup plugin)
SLURM Nodes/CPUs allocation procedure

SLURM uses four basic steps to manage CPU resources for a job/step:

- Step 1: Selection of Nodes
- Step 2: Allocation of CPUs from the selected Nodes
- Step 3: Distribution of Tasks to the selected Nodes
- Step 4: Optional Distribution and Binding of Tasks to CPUs within a Node

SLURM provides a rich set of configuration and command line options to control each step

- Many options influence more than one step
- Interactions between options can be complex and difficult to predict
- Users may be constrained by Administrator’s configuration choices
Options for Step 1: Selection Nodes

Configuration options in `slurm.conf`

- **Nodename**: Defines a node and its characteristics. This includes the layout of sockets, cores, threads and the number of logical CPUs on the node.

- **FastSchedule**: Allows administrators to define "virtual" nodes with different layout of sockets, cores and threads and logical CPUs than the physical nodes in the cluster.

- **PartitionName**: Defines a partition and its characteristics. This includes the set of nodes in the partition.

Command line options on `srun/salloc/sbatch` commands

- `--partition, --nodelist`: Specifies the set of nodes from which the selection is made

- `-N, --nodes`: Specifies the minimum/maximum number of nodes to be selected

- `-B, --sockets-per-node, --cores-per-socket, --threads-per-core`: Limits node selection to nodes with the specified characteristics
Options for Step 2: Allocation of CPUs

Configuration options in slurm.conf:

- **SelectType:**
  - SelectType=select/linear: Restricts allocation to whole nodes
  - SelectType=select/cons_res: Allows allocation of individual sockets, i cores or threads as consumable resources

- **SelectTypeParameters:** For select/cons_res, specifies the consumable resource type and default allocation method within nodes

Command line options on srun/salloc/sbatch:

- **-n, --ntasks:** Specifies the number of tasks. This may affect the number of CPUs allocated to the job/step
- **-c, --cpus-per-task:** Specifies the number of CPUs per task. This may affect the number of CPUs allocated to the job/step
Notable Options for Step 3: Distribution of Tasks to Nodes

Configuration options in slurm.conf:
- MaxTasksPerNode: Specifies maximum number of tasks per node

Command Line options on srun/salloc/sbatch:
- \(-m, \--distribution\): Controls the order in which tasks are distributed to nodes.
Notable Options for Step 4: Optional Distribution & Binding

Configuration options in slurm.conf:

- **TaskPlugin:**
  - TaskPlugin=task/none: Disables this step.
  - TaskPlugin=task/affinity: Enables task binding using the task affinity plugin.
  - TaskPlugin=task/cgroup: Enables task binding using the new task cgroup plugin.

- **TaskPluginParam:** For task/affinity, specifies the binding unit (sockets, cores or threads) and binding method (sched_setaffinity or cpusets)

Command Line options on srun/salloc/sbatch:

- **--cpu_bind:** Controls many aspects of task affinity
- **-m, --distribution:** Controls the order in which tasks are distributed to allocated CPUs on a node for binding
Slurm

Advanced Usage

Allocation & Distribution Methods

SLURM uses two default methods for allocating and distributing individual CPUs from a set of resources

▶ block method: Consume all eligible CPUs consecutively from a single resource before using the next resource in the set

▶ cyclic method: Consume eligible CPUs from each resource in the set consecutively in a round-robin fashion
Allocation of CPUs from a set of Nodes

The default allocation method for this case is **block** Example: A partition contains 3 nodes. Each node has 8 available CPUs.

```
srun --nodes=3 --ntasks=15
```

<table>
<thead>
<tr>
<th>Node</th>
<th>n0</th>
<th>n1</th>
<th>n2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocate CPUS:</td>
<td>8</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

Users can override this default using the appropriate command line options, e.g. **--nodes**, **--ntasks-per-node**
Allocation of CPUs from a set of Sockets within a Node

The default allocation method for this case is **cyclic** Example: a node contains 2 sockets, each socket has 4 available CPUs.

```
srun --ntasks=6
```

<table>
<thead>
<tr>
<th>Socket</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocate CPUs:</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

- Administrators can change the default allocation method for this case to block with `SelectTypeParameters=CR_CORE_DEFAULT_DIST_BLOCK`
- Users can override the default using the command line option `--distribution`
Distribution of Tasks to Nodes

The default distribution method for this case is **block**. Example: a partition has 3 nodes, each node has 8 available cpus.

```
srun --nodes=3 --ntasks=8 --cpus-per-task=2
```

<table>
<thead>
<tr>
<th>Node</th>
<th>n0</th>
<th>n1</th>
<th>n2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocate CPUS:</td>
<td>8</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Distribution</td>
<td>0-3</td>
<td>4-6</td>
<td>7</td>
</tr>
</tbody>
</table>

- Users can override this default using the command line option `--distribution`.
- The option supports three alternates methods for distributing tasks to nodes: cyclic, plane and arbitrary.
Distribution of Tasks to Allocated CPUs within a Node for Task-to-CPU binding

The default distribution method for this case is **cyclic**. Example: a node has 2 sockets, each socket has 4 available CPUs (cores).

```bash
srun --exclusiv --ntasks=8 --cpu_bind=cores
```

<table>
<thead>
<tr>
<th>Node</th>
<th>n0</th>
<th>Socket#</th>
<th>CPI#</th>
<th>Bounds Task#</th>
</tr>
</thead>
<tbody>
<tr>
<td>n0</td>
<td>0</td>
<td>1 2 3</td>
<td>4 5 6 7</td>
<td>0 2 4 6 1 3 5 7</td>
</tr>
</tbody>
</table>

Users can override this default and specify block distribution of tasks to CPUs using the command line option `--distribution`.
Slurm

Advanced Usage

CPU Resource Sharing between Jobs

Configuration options in slurm.conf:

- Shared keyword on partition definition in slurm.conf

Command line options

- --share, --exclusive options on srun/salloc/sbatch command line

CPU Resource Sharing between Tasks of the same Job

On command line: --overcommit option on srun/salloc/sbatch command line

Note It is important to understand that the Linux scheduler is not aware of CPU allocations by SLURM. Unless Task-to-CPU binding is used, Linux may run any task on any CPU on the node to which the task was distributed. In this way, CPUs may be shared between tasks and jobs even in the absence of shared CPU allocation by SLURM.
Information about Node Selection and CPU allocation

```
srun --nodes=2 --ntasks=3 --cpus-per-task=2 sleep 60
```

```
squeue
JOBID PARTITION  NAME USER ST TIME NODES NODLIST(REASON)
309 allnodes    sleep slurm R 0:02  2 n[6-7]
```

```
scontrol --details show job 309
```

```
NumNodes=2  NumCPUs=6  CPUs/Task=2  ReqS:C:T=**:**:**
Nodes=n6   CPU_IDs=4-7 Mem=0
Nodes=n7   CPU_IDs=6-7 Mem=0
```

**Note** The CPU_IDs reported by scontrol are SLURM abstract CPU numbers, not physical CPU numbers known to Linux.
Slurm

Advanced Usage

Information about Distribution of Tasks to Nodes

```
srun --nodes=3 --ntasks=5 sleep 60 &
```

```
squeue

<table>
<thead>
<tr>
<th>JOBID</th>
<th>PARTITION</th>
<th>NAME</th>
<th>USER</th>
<th>ST</th>
<th>TIME</th>
<th>NODES</th>
<th>NODLIST(REASON)</th>
<th>PIDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>311</td>
<td>allnodes</td>
<td>sleep</td>
<td>slurm</td>
<td>R</td>
<td>0:03</td>
<td>3</td>
<td>n[6-7,13]</td>
<td></td>
</tr>
</tbody>
</table>
```

```
sstat --allsteps --jobs 311 --pidformat

<table>
<thead>
<tr>
<th>JobID</th>
<th>Nodelist</th>
<th>Pids</th>
</tr>
</thead>
<tbody>
<tr>
<td>311.0</td>
<td>n13</td>
<td>7994,7995</td>
</tr>
<tr>
<td>311.0</td>
<td>n6</td>
<td>3838,3839</td>
</tr>
<tr>
<td>311.0</td>
<td>n7</td>
<td>7199</td>
</tr>
</tbody>
</table>
```
Information about Task-to-CPU binding

```bash
srun --nodes=2 --ntasks=4 --cpu_bind=cores,verbose --label cat /proc/self/status
   | grep Cpus_allowed_list
```

```
0: cpu_bind=MASK - scotty, task 0 0 [4070]: mask 0x8 set
3: cpu_bind=MASK - bones, task 3 0 [23808]: mask 0x80 set
1: cpu_bind=MASK - scotty, task 1 1 [4071]: mask 0x20 set
2: cpu_bind=MASK - scotty, task 2 2 [4072]: mask 0x80 set

3: Cpus_allowed_list: 7
0: Cpus_allowed_list: 3
1: Cpus_allowed_list: 5
2: Cpus_allowed_list: 7
```
Slurm

Placement with slurm

<table>
<thead>
<tr>
<th>Bitmask</th>
<th>Examples</th>
<th>=0x0F</th>
<th>=0xF0</th>
<th>=0xAA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

```bash
export OMP_NUM_THREADS=4
srun --cpu_bind=map_cpu=0,4 -N1 -n 2 ./a.out
```
Slurm

Example

export OMP_NUM_THREADS=4
srun --cpu_bind=mask_cpu=0x0F,0xF0 -N1 -n 2 ./a.out

or
srun --exclusiv -N1 -n2 --ntasks-per-node=2 -m block:cyclic --cpu_bind=socket,verbose

Linux scheduler will place inside the socket (no binding)
Slurm

Example 2

```bash
export OMP_NUM_THREADS=4
export OMP_PROC_BIND=true or export KMP_AFFINITY=compact
srun --cpu_bind=mask_cpu=0x0F,0xF0 -N1 -n 2 ./a.out
```

or

```bash
srun --exclusiv -N1 -n2 --ntasks-per-node=2 -m block:cyclic --cpu_bind=socket,verbose
```

Processes will not move at all
New Power Management features

- Record node and jobs energy consumption
- Modular and flexible framework for energy accounting; 2 existing plugins (acct_gather_energy/rapl, acct_gather_energy/ipmi):
  - From the RAPL Interfaces of new Intel processors through MSR registers
  - From the BMC of the node through the IPMI protocol
- Control over the jobs energy usage by setting the cpu frequency on the nodes that the step/job will be executed upon: Introduction --cpu-freq parameter in srun

```
sacct -j 22 -format=jobid,elapsed,aveCPUFreq,consumedenergy

<table>
<thead>
<tr>
<th>JobID</th>
<th>Elapsed</th>
<th>AveCPUFreq</th>
<th>ConsumedEnergy</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>00:04:46</td>
<td>2.60G</td>
<td>20.87K</td>
</tr>
<tr>
<td>22.batch</td>
<td>00:04:46</td>
<td>2.60G</td>
<td>20.87K</td>
</tr>
</tbody>
</table>
```
Slurm

Profiling Jobs

What is Profiling with HDF5?

- More detail than can reasonably be stored in an accounting database
- Data from all tasks on all nodes consolidated in one (HDF5) dataset
- Controls to limit data collection to only a few jobs to minimize overhead on the entire system
Data Flow

- While a job executes, the data collection plugins are periodically called on each node (by Slurmstepd).
- They call the framework plugin to add_sample.
- The Profile plugin stores the data in a node-step HDF5 file on a shared file system.
- When the job ends, sh5util merges all the node-step files into one job HDF5 file (This isn't automatic but is often done as an additional sbatch in an sbatch script).
- sh5util can also extract subsets of data as a text file to be imported into other analysis tools such as spreadsheets.
Slurm

Profiling Jobs

Data Collections

Data collection happens while a step is running. Various plugins periodically sample counters or sensors and call the framework to add the sampled data into a HDF5 file for the step. The plugins run inside the slurmd daemon so there is one step file on each node. (As noted before, these files are on a shared file system.)

- Option --profile option on `salloc`, `sbatch`, `srun` controls whether data is collected and what type of data is collected. The following options are currently available.
  
  - `none` No data types are collected. **This is the default.**
  - `all` All data types are collected
  - `energy` Energy data is collected.
  - `lustre` Lustre I/O data is collected.
  - `network` Network (InfiniBand) data is collected.
  - `task` Task (I/O, Memory, ...) data is collected.

- `--acct-freq` option may be used to override the JobAcctGatherFrequency parameter in `slurm.conf`. Different sample rates can be specified for each data type.

```
srun -N2 -n3 --profile=task,energy --acctg-freq=energy=3,task=60 ./myjob
```

or

```
srun -N2 -n3 --profile=ALL --acctg-freq=energy=3,Network=3,Filesystem=3 ./myjob
```
Slurm

Profiling Jobs

Data Consolidation

The node-step files are merged into one HDF5 file for the job using the sh5util program. They are then deleted.

```
sbatch -n1 -d$SLURM_JOB_ID --wrap="sh5util -j $SLURM_JOB_ID"
```

Data Extraction

The sh5util program can also extract all samples for a specific data item from a time series and write a comma separated value (csv) file for importation into other analysis tools such as spreadsheets.

```
sh5util -j $SLURM_JOB_ID --item-extract --series=Energy --data=power
```

Data Extraction

```
sh5util -j $SLURM_JOB_ID -E -l Node:Timeseries -s Lustre -o extract_$SLURM_JOB_ID_Lustre.csv
```
Demo
COPYRIGHT NOTICE

▶ (c) Copyright Bull. All rights reserved

☑ Users Restricted Rights - Use, duplication or disclosure restricted.
☑ Any copy of these documents should keep all copyright, logos and other proprietary notices contained herein.
☑ This publication may include technical inaccuracies or typographical errors.
☑ This publication is provided "AS IS" without any warranty either expressed or implied including but not limited to the implied warranties of merchantabilities or fitness of the described product.
☑ Course Material Licensing Terms : No sublicensing rights.
☑ For other licensing needs, please contact Bull