OAK RIDGE National Laboratory

Explicit Resource Files (ERFs)

Tom Papatheodore

Oak Ridge Leadership Computing Facility

jsrun Tutorial

February 18, 2020

U.S. DEPARTMENT OF ENERGY

Explicit Resource Files (ERFs)

ERFs allow more control over defining resource sets and mapping processes to those resources.

For example,

- Heterogeneous resource sets
- Fine-grained mapping of processes to HW threads
- Multiple executables

Basic Syntax

At a high level, ERFs contain a preamble and a body

Preamble

- Set options that influence an entire job step. E.g.,
  - How processes are distributed within and among resource sets
  - Allowing/disallowing CPU oversubscription
  - Specify executables to be run within resource sets

Body

- Define resource sets, processes that can access those resource sets, and the binding of processes within the resource sets

Preamble

There are many options that can be set in the preamble, but we will only cover the following two in this tutorial:

`cpu_index_using: VALUE` (where **`VALUE`** accepts values of **`logical`** or **`physical`**)

- Sets whether HW threads are identified by their physical or logic values (see next slide)

`app <#>: COMMAND LINE` (where **`COMMAND LINE`** should be e.g., **`./a.out [--args]`**)

- Define applications to be run within specific resource sets

For the full list of options, see the IBM Documentation on ERFs.

A brief aside on physical vs logical CPU indexing

Assuming SMT4…

Physical indexing labels the HW threads as expected for 22 physical cores (88 HW threads) per socket.

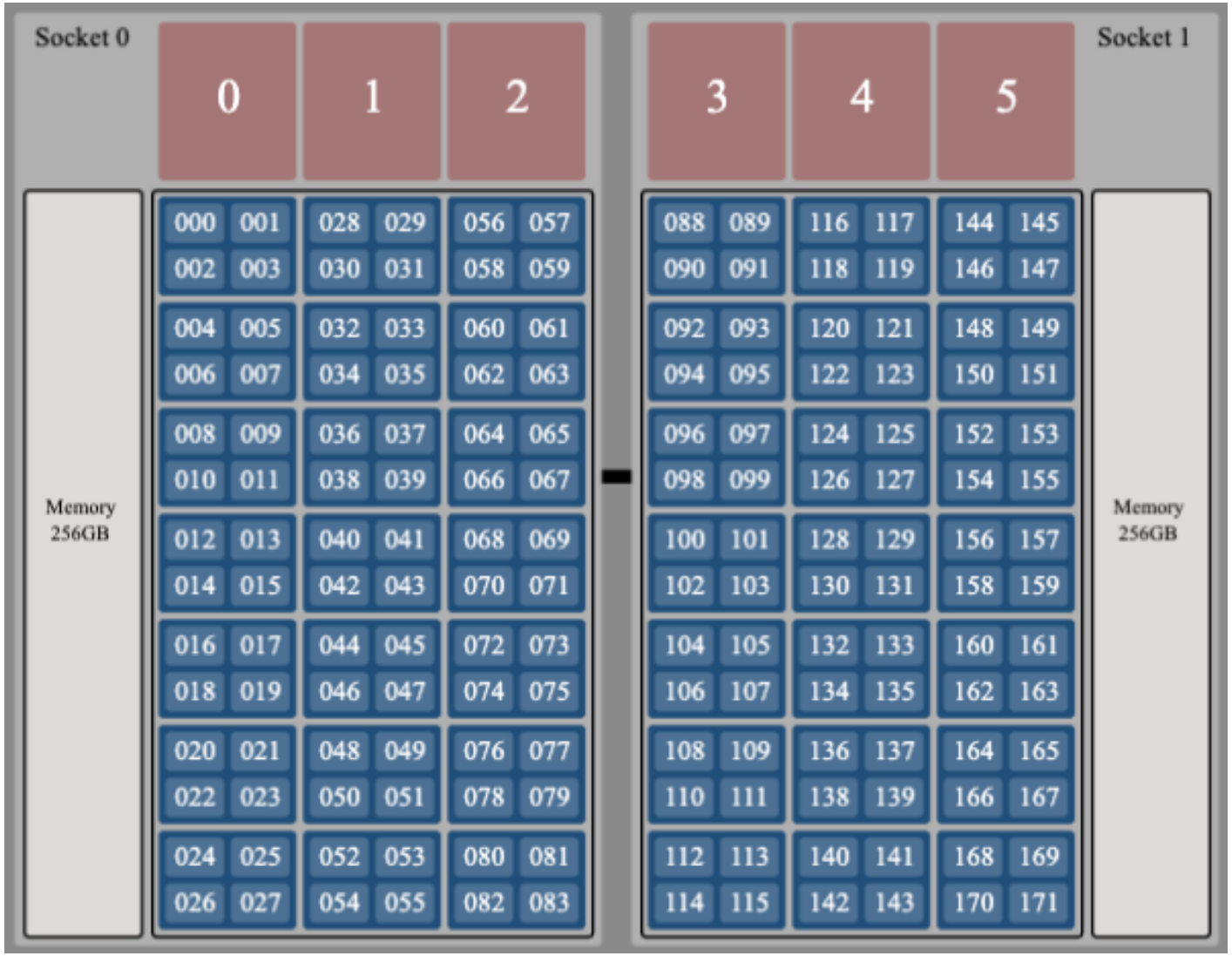
Socket 0: HW threads 000-083
- 084-087 missing due to core isolation

Socket 1: HW threads 088-171
- 172-175 missing due to core isolation

Logical indexing labels the HW threads as if only the available HW were present.

Socket 0: HW threads 000-083

Socket 1: HW threads 084-167



For more information, see https://github.com/olcf-tutorials/ERF-CPU-Indexing

Body

This is where you define resource sets, processes that can access the resource sets, and binding of processes to the resources.

The definitions can be written in different ways, but the general structure is as follows:

```
<process specification> : { <resource set specification> }
```

<process specification> is where you specify processes (e.g., MPI ranks)

<resource set specification> is where you specify hardware into resource sets

The binding of processes to hardware will be discussed later.

Process Specification

```
<process specification> : { <resource set specification> }
```

Process Specification can be accomplished in two different ways:

- Specific Process Specification
- Process Per Resource Set Specification

Specific Process Specification

```
<process specification> : { <resource set specification> }
```

This method allows you to define which specific processes (e.g., MPI ranks) have access to the resources (e.g., hardware; CPUs, GPUs) within the associated resource sets.

```
rank: <PROCESSES> : { <resource set specification> }
```

<PROCESSES> can be defined as a comma-separated list or range (or both) of processes.

For example: `rank: 0,1,3 : { <resource set specification> }`

Process Per Resource Set Specification

```
<process specification> : { <resource set specification> }
```

This method allows you to set the number of processes (e.g., MPI ranks) that have access to the resources (e.g., hardware; CPUs, GPUs) within the associated resource sets.

```
<NUMBER OF PROCESSES> : { <resource set specification> }
```

<NUMBER OF PROCESSES> is the number of processes that can access the resource set.

For example: `6 : { <resource set specification> }`

Resource Set Specification

```
<process specification> : { <resource set specification> }
```

This section allows you to define resource sets and the hosts (compute nodes) that contain those resource sets.

This is accomplished using a semicolon-separated list of key-value pairs:

```
<process specification> : { key1: value1; key2: value2; <etc> }
```

Resource Set Specification – Key-Value Pairs

<process specification> : { key1: value1; key2: value2; <etc> }

The possible keys are:

host – specifies the hosts (nodes) where the resource set will be created

- The values can be a comma-separated list and/or range of host identifiers numbered from 1 to the total number of nodes in your allocation.
- An asterisk (*) can be used to specify "all" hosts.
- On Summit, a value of 0 represents the batch/launch node but it is inaccessible during the job step

cpu – specifies sets of HW thread groups.

- The values (sets of HW thread groups) are defined as comma-separated lists of HW thread groups, where the HW thread groups can be defined as
    - Comma-separated lists and/or ranges of HW threads (e.g., **{0-3,5,7}**)
    - Starting HW thread ID + depth of HW threads (e.g., **{0:4}**). This method can differ based on the SMT-mode
    - These two methods can be mixed within a "set of HW thread groups" (e.g., **{0,2-3},{4:4}**) but not within a "single HW thread group" (e.g., **{0:4,5,7-8}**).
    - An asterisk (*) can be used to specify "all" HW threads on a node.

Resource Set Specification – Key-Value Pairs (cont.)

<process specification> : { key1: value1; key2: value2; <etc> }

The possible keys are (cont.):

gpu – specifies a group of GPUs. All processes with access to the resource set have access to the specified GPUs.

- The values (groups of GPUs) can be defined as a comma-separated list and/or range.
- An asterisk (*) can be used to specify "all" GPUs.

mem – The memory option is not currently available on Summit so it will not be covered here.

Defining the key-value pairs, as well as process binding, will become clear in the examples to follow.

Resource Set Specification – Process Binding

<process specification> : { key1: value1; key2: value2; <etc> }

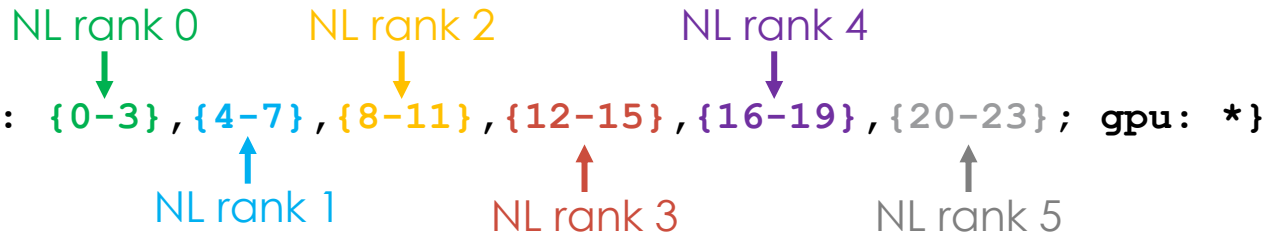
Each process defined in **<process specification>** is bound to a group of HW threads sequentially.

Specific Rank Specification

```
rank: 0,1,2,3,4,5 : { host: 1; cpu: {0-3},{4-7},{8-11},{12-15},{16-19},{20-23}; gpu: *}
```

Process Per Resource Set Specification

NL rank = Node-Local MPI rank



Using ERFs with jsrun

To use an ERF, pass **`--erf_input my_erf`** to **`jsrun`** followed by an executable and its arguments:

```
$ jsrun --erf_input my_erf ./a.out [--args]
```

As we'll see in Example 3, you can also specify the executable(s) to be run within the ERF itself. In that case, you would omit the executable and its arguments from the **`jsrun`** command:

```
$ jsrun --erf_input my_erf
```

It's also possible to create an ERF file by using **`--erf_output my_erf_name`** when running a normal **`jsrun`** command:

```
$ jsrun --erf_output my_erf_name -n6 -c7 -g1 -a1 -bpacked:1 ./a.out
```

Examples

In the following examples, assume we have the following interactive allocation on Summit:

- 2 nodes
- Physical CPU cores are in **`smt4`** mode (default)
- CUDA MPS is enabled **`(-alloc_flags gpumps`**)

This can be accomplished with the following command:

```
$ bsub -P <PROJID> -nnodes 2 -W 60 -alloc_flags gpumps -Is /bin/bash
```

`-U jsrun`
(reservation only available during tutorial)

Example 1

6 resource sets (RSs) on a single node

- Each RS contains 28 HW threads (i.e., 7 physical cores) and 1 GPU
- 1 MPI rank has access to all resources in the RS and spawns 7 OpenMP threads (1 per physical core)

Using normal **`jsrun`**:

```
$ export OMP_NUM_THREADS=7

$ jsrun -n6 -c7 -bpacked:7 -g1 ./hello_jsrun | sort

---------- MPI Ranks: 6, OpenMP Threads: 7, GPUs per Resource Set: 1 ----------
MPI Rank 000, OMP_thread 00 on HWThread 001 of Node a21n04 - RT_GPU_id 0 : GPU_id 0
MPI Rank 000, OMP_thread 01 on HWThread 004 of Node a21n04 - RT_GPU_id 0 : GPU_id 0
MPI Rank 000, OMP_thread 02 on HWThread 008 of Node a21n04 - RT_GPU_id 0 : GPU_id 0
MPI Rank 000, OMP_thread 03 on HWThread 012 of Node a21n04 - RT_GPU_id 0 : GPU_id 0
MPI Rank 000, OMP_thread 04 on HWThread 016 of Node a21n04 - RT_GPU_id 0 : GPU_id 0
MPI Rank 000, OMP_thread 05 on HWThread 020 of Node a21n04 - RT_GPU_id 0 : GPU_id 0
MPI Rank 000, OMP_thread 06 on HWThread 024 of Node a21n04 - RT_GPU_id 0 : GPU_id 0

...
MPI Rank 005, OMP_thread 00 on HWThread 145 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
MPI Rank 005, OMP_thread 01 on HWThread 148 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
MPI Rank 005, OMP_thread 02 on HWThread 152 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
MPI Rank 005, OMP_thread 03 on HWThread 156 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
MPI Rank 005, OMP_thread 04 on HWThread 160 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
MPI Rank 005, OMP_thread 05 on HWThread 164 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
MPI Rank 005, OMP_thread 06 on HWThread 168 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
```

Now let's do the same with ERFs…

Example 1 (cont.)

6 resource sets (RSs) on a single node

- Each RS contains 28 HW threads (i.e., 7 physical cores) and 1 GPU
- 1 MPI rank has access to all resources in the RS and spawns 7 OpenMP threads (1 per physical core)

Using ERF with “process per RS specification”:

```
$ cat example1_processs_per_rs.erf

cpu_index_using: physical

1 : {host: 1; cpu: {0:28};   gpu: 0}
1 : {host: 1; cpu: {28:28};  gpu: 1}
1 : {host: 1; cpu: {56:28};  gpu: 2}
1 : {host: 1; cpu: {88:28};  gpu: 3}
1 : {host: 1; cpu: {116:28}; gpu: 4}
1 : {host: 1; cpu: {144:28}; gpu: 5}
```

Rank 0 has access to HW threads 0-27 and GPU 0

...

Rank 5 has access to HW threads 144-171 and GPU 5

```
$ export OMP_NUM_THREADS=7

$ jsrun --erf_input example1_process_per_rs.erf ./hello_jsrun | sort

---------- MPI Ranks: 6, OpenMP Threads: 7, GPUs per Resource Set: 1 ----------
MPI Rank 000, OMP_thread 00 on HWThread 000 of Node a21n04 - RT_GPU_id 0 : GPU_id 0
MPI Rank 000, OMP_thread 01 on HWThread 004 of Node a21n04 - RT_GPU_id 0 : GPU_id 0
MPI Rank 000, OMP_thread 02 on HWThread 008 of Node a21n04 - RT_GPU_id 0 : GPU_id 0
MPI Rank 000, OMP_thread 03 on HWThread 012 of Node a21n04 - RT_GPU_id 0 : GPU_id 0
MPI Rank 000, OMP_thread 04 on HWThread 016 of Node a21n04 - RT_GPU_id 0 : GPU_id 0
MPI Rank 000, OMP_thread 05 on HWThread 020 of Node a21n04 - RT_GPU_id 0 : GPU_id 0
MPI Rank 000, OMP_thread 06 on HWThread 024 of Node a21n04 - RT_GPU_id 0 : GPU_id 0

...
MPI Rank 005, OMP_thread 00 on HWThread 147 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
MPI Rank 005, OMP_thread 01 on HWThread 148 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
MPI Rank 005, OMP_thread 02 on HWThread 152 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
MPI Rank 005, OMP_thread 03 on HWThread 156 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
MPI Rank 005, OMP_thread 04 on HWThread 160 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
MPI Rank 005, OMP_thread 05 on HWThread 164 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
MPI Rank 005, OMP_thread 06 on HWThread 168 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
```

Example 1 (cont.)

6 resource sets (RSs) on a single node

- Each RS contains 28 HW threads (i.e., 7 physical cores) and 1 GPU
- 1 MPI rank has access to all resources in the RS and spawns 7 OpenMP threads (1 per physical core)

Using ERF with “specific rank specification”:

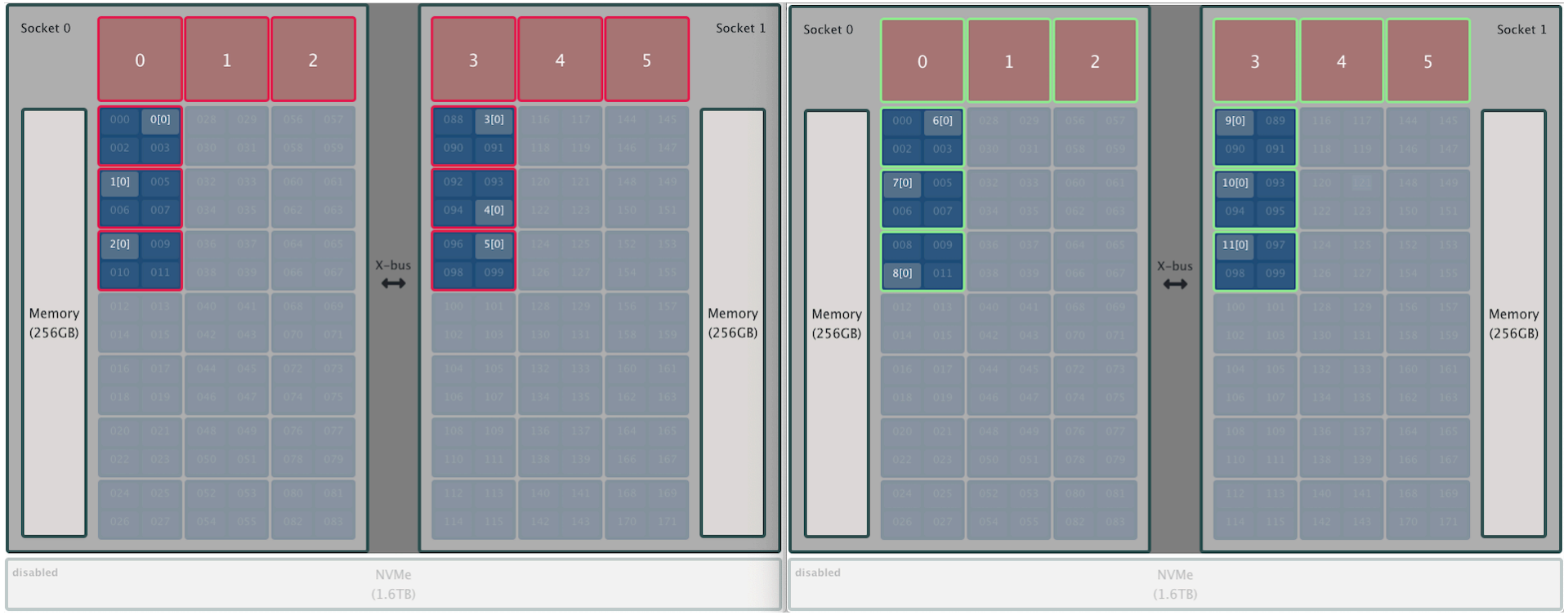
```
$ cat example1_specific_rank.erf

cpu_index_using: physical
                                                  Rank 0 has access to HW threads 0-27 and GPU 0
rank: 0 : {host: 1; cpu: {0:28};   gpu: 0}
rank: 1 : {host: 1; cpu: {28:28};  gpu: 1}
rank: 2 : {host: 1; cpu: {56:28};  gpu: 2}
rank: 3 : {host: 1; cpu: {88:28};  gpu: 3}        . . .
rank: 4 : {host: 1; cpu: {116:28}; gpu: 4}
rank: 5 : {host: 1; cpu: {144:28}; gpu: 5}
                                                  Rank 5 has access to HW threads 144-171 and GPU 5
```

```
$ export OMP_NUM_THREADS=7

$ jsrun --erf_input example1_specific_rank.erf ./hello_jsrun | sort

---------- MPI Ranks: 6, OpenMP Threads: 7, GPUs per Resource Set: 1 ----------
MPI Rank 000, OMP_thread 00 on HWThread 001 of Node a21n04 - RT_GPU_id 0 : GPU_id 0
MPI Rank 000, OMP_thread 01 on HWThread 004 of Node a21n04 - RT_GPU_id 0 : GPU_id 0
MPI Rank 000, OMP_thread 02 on HWThread 008 of Node a21n04 - RT_GPU_id 0 : GPU_id 0
MPI Rank 000, OMP_thread 03 on HWThread 012 of Node a21n04 - RT_GPU_id 0 : GPU_id 0
MPI Rank 000, OMP_thread 04 on HWThread 016 of Node a21n04 - RT_GPU_id 0 : GPU_id 0
MPI Rank 000, OMP_thread 05 on HWThread 020 of Node a21n04 - RT_GPU_id 0 : GPU_id 0
MPI Rank 000, OMP_thread 06 on HWThread 024 of Node a21n04 - RT_GPU_id 0 : GPU_id 0

...
MPI Rank 005, OMP_thread 00 on HWThread 146 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
MPI Rank 005, OMP_thread 01 on HWThread 148 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
MPI Rank 005, OMP_thread 02 on HWThread 152 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
MPI Rank 005, OMP_thread 03 on HWThread 156 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
MPI Rank 005, OMP_thread 04 on HWThread 160 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
MPI Rank 005, OMP_thread 05 on HWThread 164 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
MPI Rank 005, OMP_thread 06 on HWThread 168 of Node a21n04 - RT_GPU_id 0 : GPU_id 5
```

Example 2

2 RSs (1 per node)

- Each RS contains 24 HW threads (6 physical cores) and 6 GPUs
- 6 MPI ranks have access to each RS and each rank has a depth of (i.e., access to) 4 HW threads (1 physical core) and all 6 GPUs



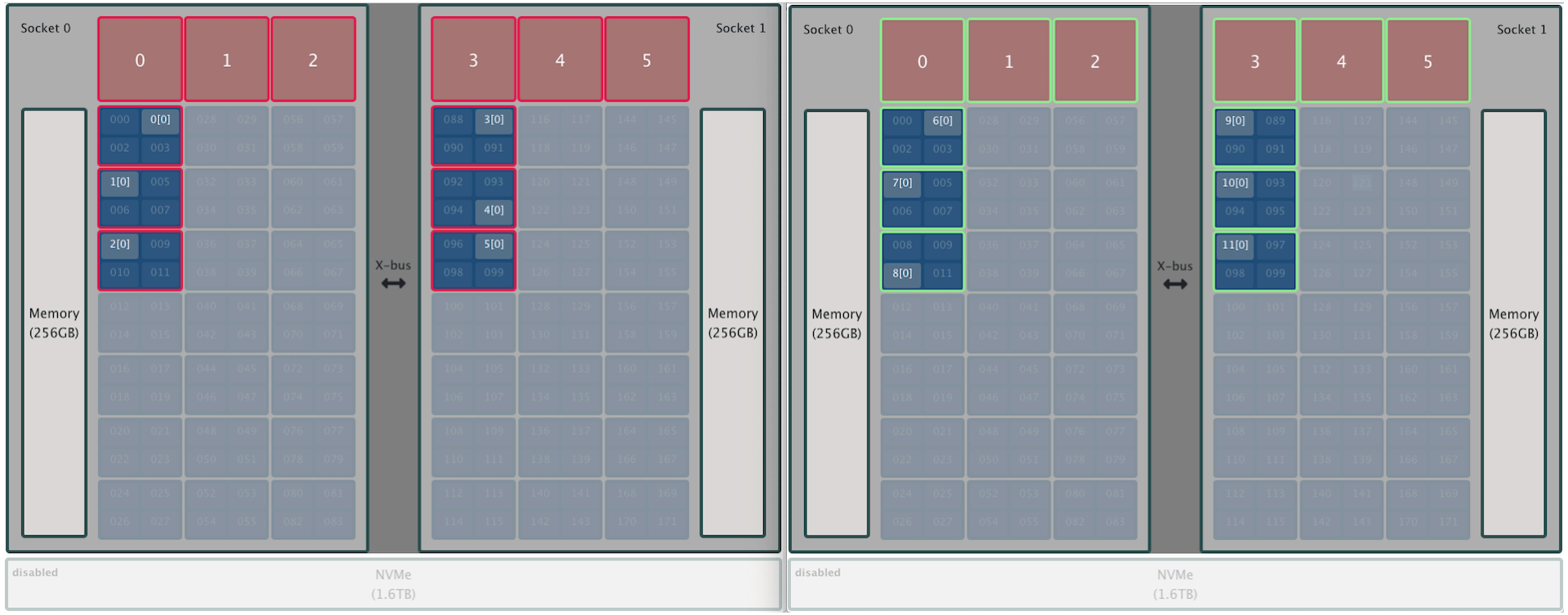
Using normal **`jsrun`**:

```
$ export OMP_NUM_THREADS=1

$ jsrun -n2 -r1 -c6 -g6 -a6 -bpacked:1 ./hello_jsrun | sort

---------- MPI Ranks: 12, OpenMP Threads: 1, GPUs per Resource Set: 6 ----------
MPI Rank 000, OMP_thread 00 on HWThread 000 of Node g17n07 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 001, OMP_thread 00 on HWThread 004 of Node g17n07 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 002, OMP_thread 00 on HWThread 008 of Node g17n07 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 003, OMP_thread 00 on HWThread 088 of Node g17n07 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 004, OMP_thread 00 on HWThread 095 of Node g17n07 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 005, OMP_thread 00 on HWThread 099 of Node g17n07 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 006, OMP_thread 00 on HWThread 000 of Node h50n06 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 007, OMP_thread 00 on HWThread 007 of Node h50n06 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 008, OMP_thread 00 on HWThread 008 of Node h50n06 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 009, OMP_thread 00 on HWThread 089 of Node h50n06 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 010, OMP_thread 00 on HWThread 093 of Node h50n06 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 011, OMP_thread 00 on HWThread 097 of Node h50n06 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
```

Example 2

2 RSs (1 per node)

- Each RS contains 24 HW threads (6 physical cores) and 6 GPUs
- 6 MPI ranks have access to each RS and each rank has a depth of (i.e., access to) 4 HW threads (1 physical core) and all 6 GPUs



To do this with an ERF:

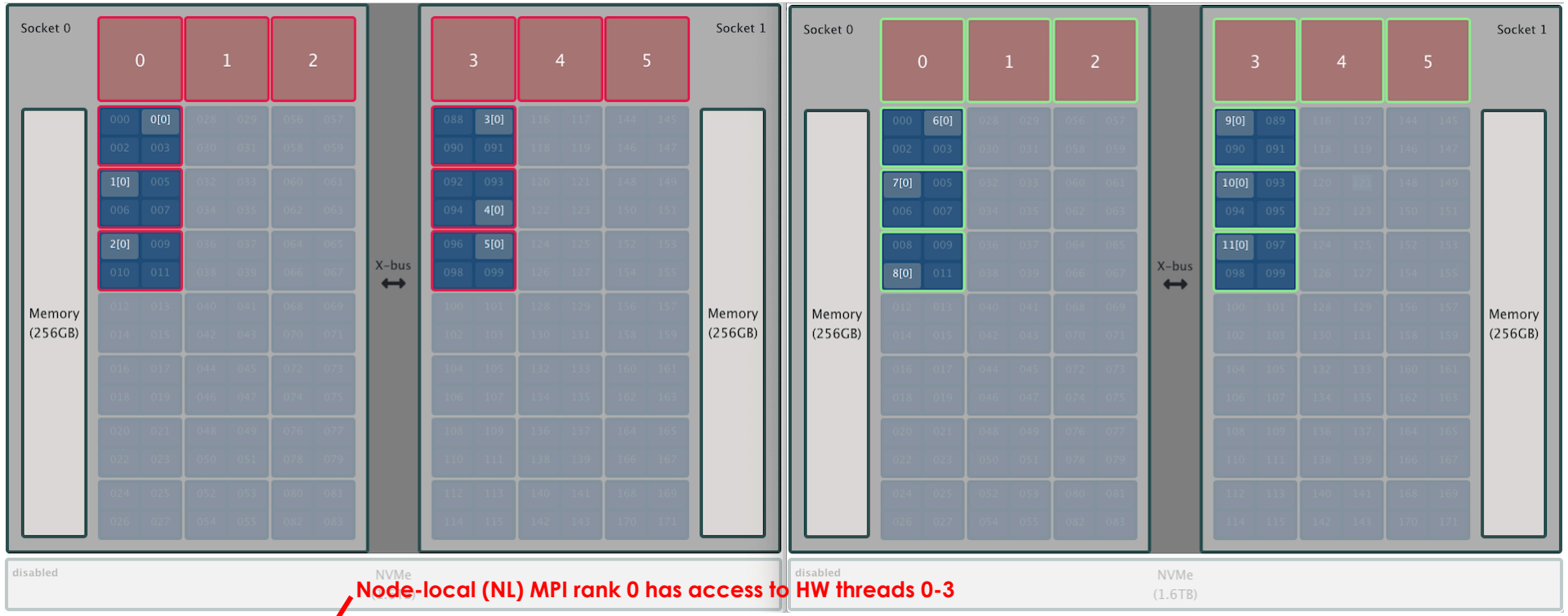
```
$ cat example2.erf

cpu_index_using: physical
6 : {host: * ; cpu: {0:4},{4:4},{8:4},{88:4},{92:4},{96:4} ; gpu: *}
```

```
$ export OMP_NUM_THREADS=1

$ jsrun --erf_input example2.erf ./hello_jsrun | sort

---------- MPI Ranks: 12, OpenMP Threads: 1, GPUs per Resource Set: 6 ----------
MPI Rank 000, OMP_thread 00 on HWThread 001 of Node g17n07 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 001, OMP_thread 00 on HWThread 005 of Node g17n07 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 002, OMP_thread 00 on HWThread 009 of Node g17n07 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 003, OMP_thread 00 on HWThread 089 of Node g17n07 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 004, OMP_thread 00 on HWThread 093 of Node g17n07 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 005, OMP_thread 00 on HWThread 098 of Node g17n07 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 006, OMP_thread 00 on HWThread 000 of Node h50n06 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 007, OMP_thread 00 on HWThread 004 of Node h50n06 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 008, OMP_thread 00 on HWThread 008 of Node h50n06 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 009, OMP_thread 00 on HWThread 091 of Node h50n06 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 010, OMP_thread 00 on HWThread 094 of Node h50n06 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 011, OMP_thread 00 on HWThread 098 of Node h50n06 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
```

Example 2

2 RSs (1 per node)

- Each RS contains 24 HW threads (6 physical cores) and 6 GPUs
- 6 MPI ranks have access to each RS and each rank has a depth of (i.e., access to) 4 HW threads (1 physical core) and all 6 GPUs



Node-local (NL) MPI rank 0 has access to HW threads 0-3

To do this with an ERF:

```
$ cat example2.erf

cpu_index_using: physical
6 : {host: * ; cpu: {0:4},{4:4},{8:4},{88:4},{92:4},{96:4} ; gpu: *}
```

NL rank 1

NL rank 2

All MPI ranks associated with this RS have access to "all" 6 GPUs

6 MPI ranks in each RS mapped to HW thread sets in a round-robin fashion

Create this RS on "all" hosts

```
$ export OMP_NUM_THREADS=1

$ jsrun --erf_input example2.erf ./hello_jsrun | sort

---------- MPI Ranks: 12, OpenMP Threads: 1, GPUs per Resource Set: 6 ----------
MPI Rank 000, OMP_thread 00 on HWThread 001 of Node g17n07 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 001, OMP_thread 00 on HWThread 005 of Node g17n07 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 002, OMP_thread 00 on HWThread 009 of Node g17n07 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 003, OMP_thread 00 on HWThread 089 of Node g17n07 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 004, OMP_thread 00 on HWThread 093 of Node g17n07 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 005, OMP_thread 00 on HWThread 098 of Node g17n07 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 006, OMP_thread 00 on HWThread 000 of Node h50n06 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 007, OMP_thread 00 on HWThread 004 of Node h50n06 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 008, OMP_thread 00 on HWThread 008 of Node h50n06 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 009, OMP_thread 00 on HWThread 091 of Node h50n06 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 010, OMP_thread 00 on HWThread 094 of Node h50n06 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
MPI Rank 011, OMP_thread 00 on HWThread 098 of Node h50n06 - RT_GPU_id 0 1 2 3 4 5 : GPU_id 0 1 2 3 4 5
```

NL rank 3

NL rank 4

NL rank 5 has access to HW threads 96-99

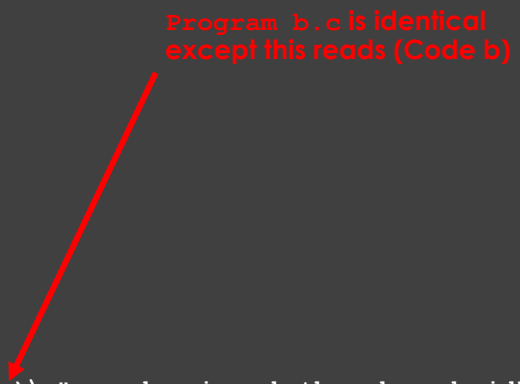
Resource set

Resource set

Example 3

Multiple-Program Multiple-Data (MPMD)

```
$ cat a.c

#define _GNU_SOURCE
#include <stdio.h>
#include <mpi.h>
#include <sched.h>

int main(int argc, char * argv[])
{
   int rank, size;
   int hwthread;
   char node_id[MPI_MAX_PROCESSOR_NAME];
   int result_length;

   MPI_Init(&argc, &argv);


   MPI_Get_processor_name(node_id, &result_length);


   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);

   hwthread = sched_getcpu();

   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);


   printf("MPI rank %02d of %02d on HW Thread %03d of Node %s - (Code a)\n", rank, size, hwthread, node_id);

   MPI_Finalize();

   return 0;
}
```



2 Programs: **`a.c`** and **`b.c`**

Both programs print each MPI rank's ID as well as the HW thread each rank ran on.

The only difference between the 2 programs is the **`printf`** line.

This program gives us the opportunity to examine the option to specify one or more applications within the preamble of the ERF

`app <#>: COMMAND LINE`

Example 3 (cont.)

Multiple-Program Multiple-Data (MPMD)

app <#>: COMMAND LINE

```
$ cat example3.erf

cpu_index_using: physical

app 0: ./run_a
app 1: ./run_b

8 : {host: 1; cpu: {0-3},{4-7},{8-11},{12-15},{16-19},{20-23},{24-27},{28-31} } : app 0
4 : {host: 1; cpu: {88-91},{92-95},{96-99},{100-103} } : app 1
```

a.c compiled to run_a
b.c compiled to run_b

Defined sets of HW threads using ranges instead of HW thread + depth

Here, we define our 2 applications (**run_a** and **run_b**) and run them under 2 different resource sets.

```
$ jsrun --erf_input example3.erf | sort

MPI rank 00 of 12 on HW Thread 000 of Node h27n01 - (Code a)
MPI rank 01 of 12 on HW Thread 005 of Node h27n01 - (Code a)
MPI rank 02 of 12 on HW Thread 010 of Node h27n01 - (Code a)
MPI rank 03 of 12 on HW Thread 013 of Node h27n01 - (Code a)
MPI rank 04 of 12 on HW Thread 016 of Node h27n01 - (Code a)
MPI rank 05 of 12 on HW Thread 021 of Node h27n01 - (Code a)
MPI rank 06 of 12 on HW Thread 026 of Node h27n01 - (Code a)
MPI rank 07 of 12 on HW Thread 029 of Node h27n01 - (Code a)
MPI rank 08 of 12 on HW Thread 089 of Node h27n01 - (Code b)
MPI rank 09 of 12 on HW Thread 092 of Node h27n01 - (Code b)
MPI rank 10 of 12 on HW Thread 096 of Node h27n01 - (Code b)
MPI rank 11 of 12 on HW Thread 101 of Node h27n01 - (Code b)
```

8 MPI ranks reporting from run_a

4 MPI ranks reporting from run_b

All 12 MPI ranks reporting under the same MPI_COMM_WORLD

Questions?