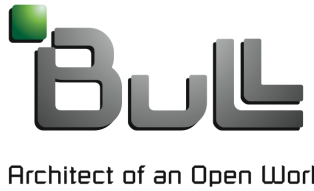
Getting Started on the Bullx HPC Environment



Fisnik Kraja
HPC Services and Software
science + computing ag

Outline

- SLURM Commands
  - sinfo, squeue, sbatch, srun, scancel
  - scontrol show job, scontrol show partition
  - sbatch script structure
  - Important Environment Variables
- CPU Binding
  - With MPI
    - For example with Bullx MPI
  - With SLURM (srun)
    - Intel MPI and Bullx MPI Integration with SLURM
    - Using or not logical cores
    - Pure MPI and hybrid MPI+OpenMP Codes
- LUSTRE
  - Lustre Components
  - File Striping

SLURM Commands

- Sinfo - Information on nodes and partitions
  - sinfo
  - sinfo –p partition_name
  - sinfo –summarize
- squeue - information about jobs on the queue
  - squeue –u user_name
- scontrol show job=job_id
- scontrol show partition=partition_name
- Scontrol show node=node_name
- scancel job_id

sinfo - Information on nodes and partitions

- sinfo –p partition_name

```
PARTITION   AVAIL  TIMELIMIT  NODES   STATE    NODELIST
B71010c*    up     infinite       2   drain*   sid[618,627]
B71010c*    up     infinite       1   alloc    sid622
B71010c*    up     infinite      87   idle     sid[619-621,623-626,628-689]
```

- sinfo –summarize

```
sinfo -s
PARTITION     AVAIL     TIMELIMIT          NODES(A/I/O/T)     NODELIST
batch         up        infinite           2/6/0/8            adev[8-15]
debug*        up        30:00              0/8/0/8            adev[0-7]
```

squeue and scancel

- squeue –u xkrajaf

```
JOBID   PARTITION   NAME   USER      ST   TIME      NODES   NODELIST(REASON)
8703    B71010c     test   xkrajaf   R    1:12:36   20      sid[330-337,628-639]
```

- scancel 8703

```
salloc: Job allocation 8703 has been revoked.
Hangup
```

scontrol show job=job_id

```
[xkrajaf@sid4 ~]$ scontrol show job=8703
JobId=8703 Name=test
   UserId=xkrajaf(51220) GroupId=guest(28893)
   Priority=88 Account=bull QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=0 ExitCode=0:0
   RunTime=00:00:10 TimeLimit=04:00:00 TimeMin=N/A
   SubmitTime=2014-07-04T14:21:45 EligibleTime=2014-07-04T14:21:45
   StartTime=2014-07-04T14:21:45 EndTime=2014-07-04T18:21:45
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   Partition=B71010c AllocNode:Sid=sid4:14059
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=sid[330-337,628-639]
   BatchHost=sid342
   NumNodes=20 NumCPUs=960 CPUs/Task=1 ReqS:C:T=*:*:2
   MinCPUsNode=16 MinMemoryNode=0 MinTmpDiskNode=0
   Features=(null) Gres=(null) Reservation=(null)
   Shared=OK Contiguous=0 Licenses=(null) Network=(null)
   Command=
   WorkDir=/home_nfs/xkrajaf
```

scontrol show partition=partition_name

```
[xkrajaf@sid4 ~]$ scontrol show partition=B510F
PartitionName=B510F
   AllocNodes=ALL AllowGroups=ALL Default=NO
   DefaultTime=04:00:00 DisableRootJobs=NO GraceTime=0 Hidden=NO
   MaxNodes=UNLIMITED MaxTime=UNLIMITED MinNodes=1
MaxCPUsPerNode=UNLIMITED
   Nodes=sid[300-317]
   Priority=1 RootOnly=NO ReqResv=NO Shared=NO PreemptMode=CANCEL
   State=UP TotalCPUs=576 TotalNodes=18 SelectTypeParameters=N/A
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
```

Starting a Job

- Interactive
  - srun -N2 -n32 ./binary
- Batch Job
  - sbatch script
    - sbatch parameters
    - srun/mpirun
- srun/sbatch options
  - --cpu-freq=<kHz>
  - --mem=<MB>
  - --mem-per-cpu=<MB>
  - --dependency
  - --kill-on-bad-exit[=0|1]
- srun/sbatch
  - Node Selection
    - -N, --nodes
    - -p, --partition
    - --sockets-per-node
    - --threads-per-core
    - --cores-per-socket
    - -c, --cpus-per-task
  - Task Distribution
    - -n, --ntasks
    - --ntasks-per-node
    - --ntasks-per-socket
    - --ntasks-per-core
    - -m, --distribution
    - --cpu_bind

SLURM Submission Script – sbatch command

- "sbatch" submits a batch script to SLURM
  - sbatch <arguments> script
- Arguments can also be specified inside the script:

```
#SBATCH -J job_name
#SBATCH --time=1:00:00
#SBATCH –exclusive
#SBATCH --begin=20:00:00
```

Job details

```
#SBATCH --partition=B71010c
#SBATCH --nodes=2
#SBATCH --sockets-per-node=2
#SBATCH --cores-per-socket=12
#SBATCH --threads-per-core=2
```

Node Selection

```
#SBATCH --ntasks=48
#SBATCH --ntasks-per-node=24
#SBATCH --ntasks-per-socket=12
#SBATCH --cpus-per-task=2
```

Allocation of CPUs from Selected Nodes

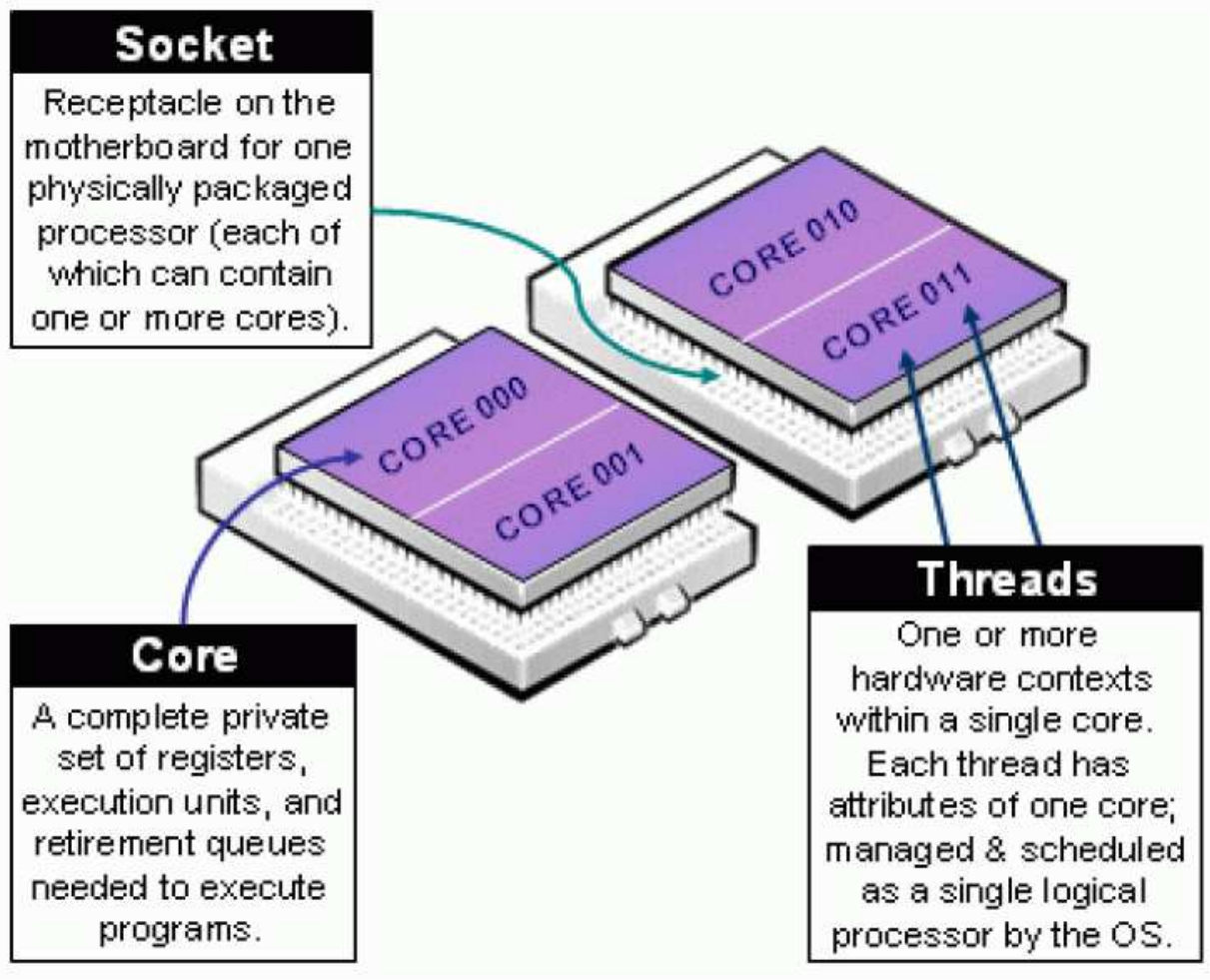
```
#SBATCH --distribution=cyclic:block
#SBATCH --cpu_bind=cores
```

Taks Distribution and Binding

SLURM Environment Variables

VARIABLE	DESCRIPTION
SLURM_NODELIST	List of reserved Nodes
SLURM_SUBMIT_DIR	Starting Directory
SLURM_NNODES	Number of Reserved Nodes
SLURM_NTASKS_PER_NODE	Number of MPI Tasks per Node
SLURM_JOBID	ID of the SLURM Job
SLURM_NTASKS	Total number of MPI Tasks

CPU Binding Terminology in SLURM



CPU Binding with MPI

- If you want the MPI library to do CPU binding, you need to tell slurm not to do it:

  sbatch --cpu_bind=none script_name.slurm

- Examples with Bullx MPI
  - mpirun -bycore -bind-to-core -npernode 24 –npersocket 12 -n 240 ./binary
  - mpirun –bysocket --bind-to-socket -npernode 24 –npersocket 12 -n 240 ./binary
  - mpirun --report-bindings
    - mpirun will display a view of the binding.

CPU Binding with with srun

- Intel MPI integration with SLURM
  - export I_MPI_PMI_LIBRARY=/usr/lib64/***libpmi.so***
  - srun --cpu_bind=rank -n 240 -K1 ./binary
- Bullx MPI integration with SLURM
  - srun ***--resv-ports*** --cpu_bind=rank -n 240 –K1 ./binary

--cpu_bind=

rank | map_cpu:<list> | mask_cpu:<list> | rank_ldom | map_ldom:<list> | mask_ldom:<list> | sockets | cores | threads | ldoms

sbatch/srun --cpu_bind=

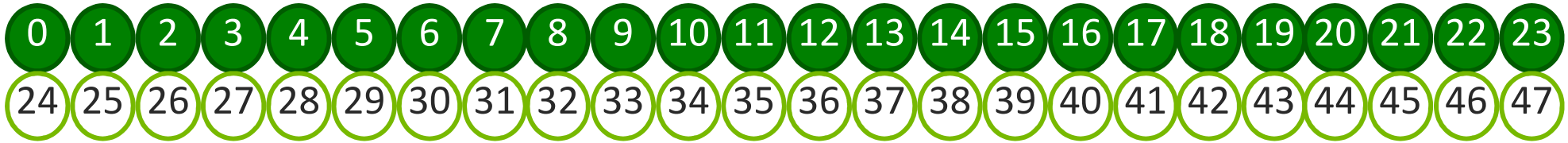
Option	Description
rank	Bind by task rank
map_cpu:<list>	Bind by mapping CPU IDs to tasks as specified where <list> is <cpuid1>,<cpuid2>,...<cpuidN>
mask_cpu:<list>	Bind by setting CPU masks on tasks as specified where <list> is <mask1>,<mask2>,...<maskN>
rank_ldom	Bind to a NUMA locality domain by rank
map_ldom:<list>	Bind by mapping NUMA locality domain IDs to tasks as specified where <list> is <ldom1>,<ldom2>,...<ldomN>
mask_ldom:<list>	Bind by setting NUMA locality domain masks on tasks as specified where <list> is <mask1>,<mask2>,...<maskN>
sockets	Automatically generate masks binding tasks to sockets
cores	Automatically generate masks binding tasks to cores
threads	Automatically generate masks binding tasks to threads
ldoms	Automatically generate masks binding tasks to NUMA locality domains
boards	Automatically generate masks binding tasks to boards

- Important Environment Variables
  - SLURM_CPU_BIND_VERBOSE
  - SLURM_CPU_BIND_TYPE
  - SLURM_CPU_BIND_LIST

srun : Task placement without using logical cores

- For Pure MPI applications:

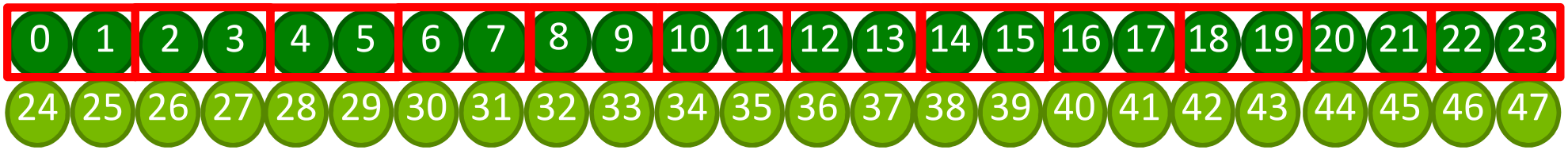
–cpu_bind=map_cpu:…




```
if [ $OMP_NUM_THREADS == 1 ] ; then
  export SRUN_BINDING="--cpu_bind=map_cpu:0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23"
  srun  -n $tasks  $SRUN_BINDING  -K1  ./binary
```

srun : Task placement without using logical cores

- PPN=12 and OMP_NUM_THREADS=2

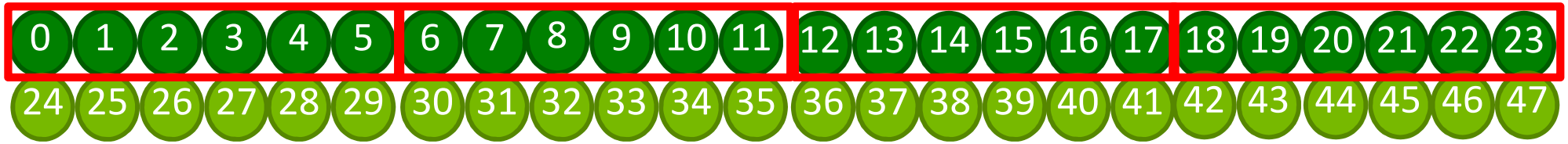



```
elif [ $OMP_NUM_THREADS == 2 ] ; then
  export SRUN_BINDING="--cpu_bind=mask_cpu:0x3,0xC,0x30,0xC0,0x300,0xC00,0x3000,0xC000,0x30000,0xC0000,0x300000,0xC00000"
  srun  -n  $tasks  -c  $OMP_NUM_THREADS  $SRUN_BINDING  ./binary
```

Rank	Bind to Cores	Hex	Cores					
			23-20	19-16	15-12	11-8	7-4	3-0
0	0,1“	0x3						0011
1	2,3	0xC						1100
2	4,5	0x30					0011	0000
3	6,7	0xC0					1100	0000
4	8,9	0x300				0011	0000	0000
5	10,11	0xC00				1100	0000	0000
6	12,13	0x3000			0011	0000	0000	0000
7	14,15	0xC000			1100	0000	0000	0000
8	16,17	0x30000		0011	0000	0000	0000	0000
9	18,19	0xC0000		1100	0000	0000	0000	0000
10	20,21	0x300000	0011	0000	0000	0000	0000	0000
11	22,23	0xC00000	1100	0000	0000	0000	0000	0000

srun : Task placement without using logical cores

- PPN=4 and OMP_NUM_THREADS=6

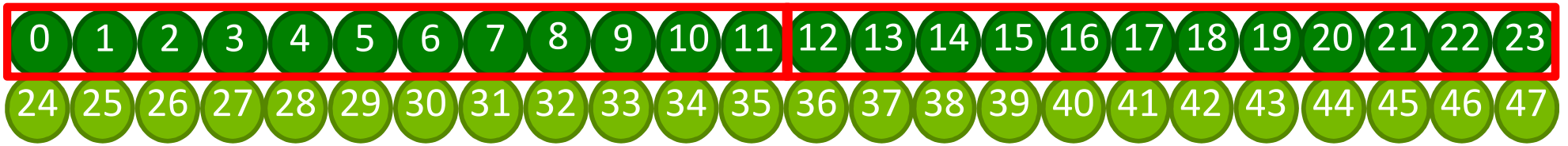



```
elif [ $OMP_NUM_THREADS == 6 ] ; then
  export  SRUN_BINDING="--cpu_bind=mask_cpu:0x3F,0xFC0,0x3F000,0xFC0000"
  srun –n $tasks -c $OMP_NUM_THREADS  $SRUN_BINDING   ./binary
```

Rank	Bind to Core	Hex	Cores					
			23-20	19-16	15-12	11-8	7-4	3-0
0	0,1,2,3,4,5	3F					0011	1111
1	6,7,8,9,10,11	FC0				1111	1100	0000
2	12,13,14,15,16,17	3F000		0011	1111	0000	0000	0000
3	18,19,20,21,22,23	FC0000	1111	1100	0000	0000	0000	0000

srun : Task placement without using logical cores

- PPN=2 and OMP_NUM_THREADS=12

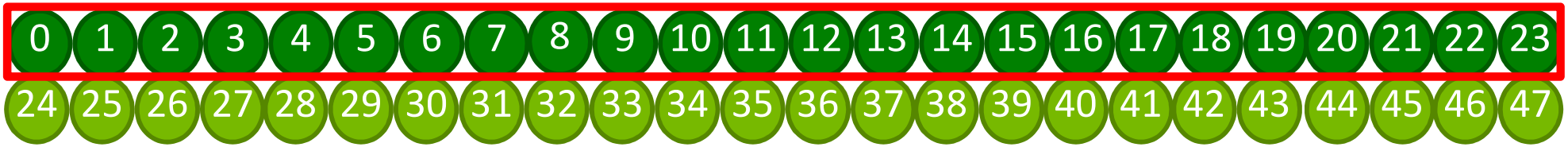



```
elif [ $OMP_NUM_THREADS == 12 ] ; then
  export SRUN_BINDING="--cpu_bind=mask_cpu:0xFFF,0xFFF000"
  srun  -n $tasks  -c $OMP_NUM_THREADS   $SRUN_BINDING  ./binary
```

Rank	Bind to Core	Hex	Cores					
			23-20	19-16	15-12	11-8	7-4	3-0
0	0-11	FFF				1111	1111	1111
1	12-23	FFF000	1111	1111	1111	0000	0000	0000

srun : Task placement without using logical cores

- PPN=1 and OMP_NUM_THREADS=24

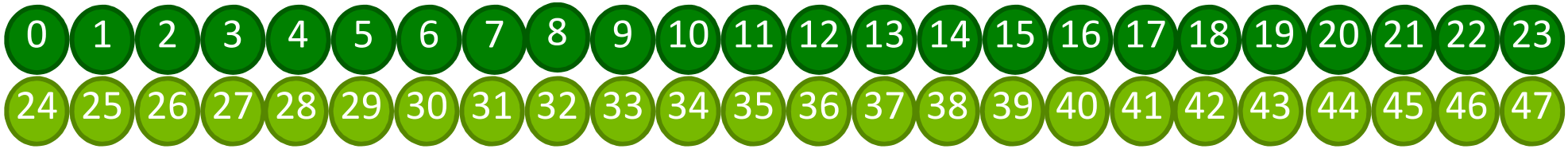



```
elif [ $OMP_NUM_THREADS == 24 ] ; then
  export SRUN_BINDING="--cpu_bind=mask_cpu:0xFFFFFF"
  srun -n  $tasks  -c  $OMP_NUM_THREADS  $SRUN_BINDING  ./binary
```

Rank	Bind to Core	Hex	Cores					
			23-20	19-16	15-12	11-8	7-4	3-0
0	0-23	FFFFF	1111	1111	1111	1111	1111	1111

srun : Task placement when using logical cores

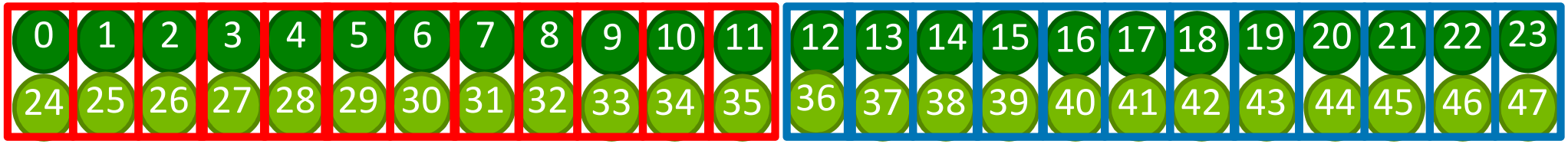
- Is it really efficient for Pure MPI Applications?




```
if [ $OMP_NUM_THREADS == 1 ] ; then
  export PHYSICAL_CORES=0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23
  export LOGICAL_CORES=24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47
  export SRUN_BINDING="--cpu_bind=map_cpu:$PHYSICAL_CORES,$LOGICAL_CORES"
  srun -n $tasks  $SRUN_BINDING  ./binary
```

srun : Task placement when using logical cores

- PPN=24 and OMP_NUM_THREADS=2

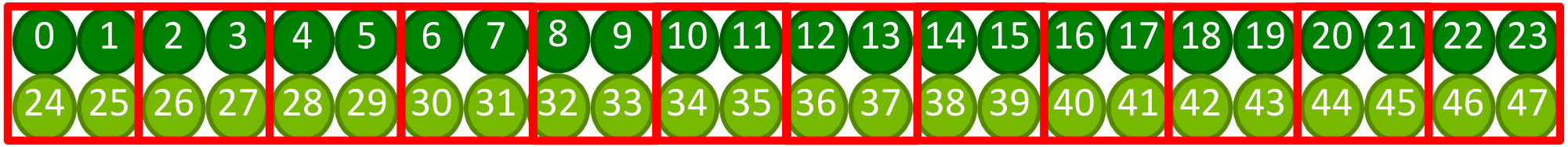



```
elif [ $OMP_NUM_THREADS == 2 ] ; then

  export SRUN_BINDING="--cpu_bind=mask_cpu: 0x1000001,0x2000002,0x4000004,0x8000008,
        0x10000010,0x20000020,0x40000040,0x80000080,0x100000100,0x200000200,0x400000400,0x800000800,
        0x1000001000,0x2000002000,0x4000004000,0x8000008000,0x10000010000,0x20000020000,0x40000040000,0x80000080000,
        0x100000100000,0x200000200000,0x400000400000,0x800000800000"

srun  -n $tasks  -c $OMP_NUM_THREADS  $SRUN_BINDING ./binary
```

Rank	Hex	Cores								
		35-32	31-28	27-24	23-20	19-16	15-12	11-8	7-4	3-0
0	1000001			0001	0000	0000	0000	0000	0000	0001
1	2000002			0010	0000	0000	0000	0000	0000	0010
2	4000004			0100	0000	0000	0000	0000	0000	0100
3	8000008			1000	0000	0000	0000	0000	0000	1000
4	10000010		0001	0000	0000	0000	0000	0000	0001	0000
5	20000020		0010	0000	0000	0000	0000	0000	0010	0000
6	40000040		0100	0000	0000	0000	0000	0000	0100	0000
7	80000080		1000	0000	0000	0000	0000	0000	1000	0000

srun : Task placement when using logical cores

- PPN=12 and OMP_NUM_THREADS=4

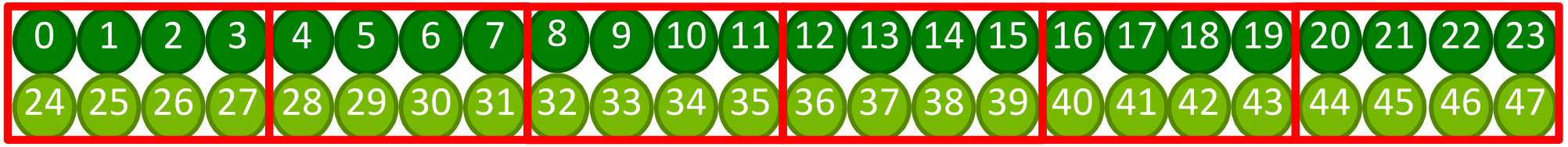



```
elif [ $OMP_NUM_THREADS == 4 ] ; then
  export SRUN_BINDING=" --cpu_bind=mask_cpu: 0x3000003,0xc00000c,0x30000030,0xc00000c0,0x300000300,0xc00000c00,
        0x3000003000,0xc00000c000,0x30000030000,0xc00000c0000,0x300000300000,0xc00000c00000"
  srun  -n $tasks  -c $OMP_NUM_THREADS  $SRUN_BINDING  ./binary
```

Rank	Hex	Cores											
		47-44	43-40	39-36	35-32	31-28	27-24	23-20	19-16	15-12	11-8	7-4	3-0
0	3000003						0011	0000	0000	0000	0000	0000	0011
1	C00000C						1100	0000	0000	0000	0000	0000	1100
2	30000030					0011	0000	0000	0000	0000	0000	0011	0000
3	C00000C0					1100	0000	0000	0000	0000	0000	1100	0000
4	300000300				0011	0000	0000	0000	0000	0000	0011	0000	0000
5	C00000C00				1100	0000	0000	0000	0000	0000	1100	0000	0000
6	3000003000			0011	0000	0000	0000	0000	0000	0011	0000	0000	0000
7	C00000C000			1100	0000	0000	0000	0000	0000	1100	0000	0000	0000
8	30000030000		0011	0000	0000	0000	0000	0000	0011	0000	0000	0000	0000
9	C00000C0000		1100	0000	0000	0000	0000	0000	1100	0000	0000	0000	0000
10	300000300000	0011	0000	0000	0000	0000	0000	0011	0000	0000	0000	0000	0000
11	C00000C00000	1100	0000	0000	0000	0000	0000	1100	0000	0000	0000	0000	0000

srun : Task placement when using logical cores

- PPN=6 and OMP_NUM_THREADS=8

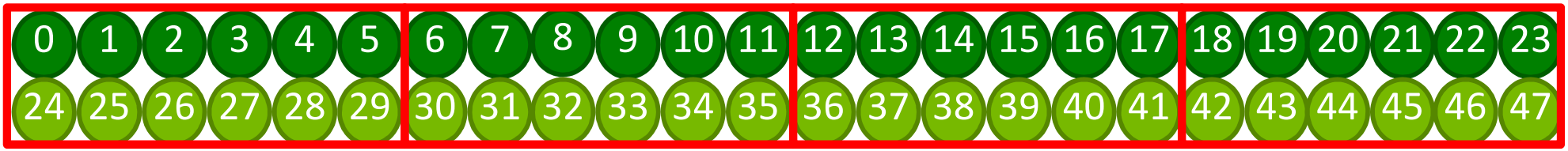



```
elif [ $OMP_NUM_THREADS == 8 ] ; then
  export SRUN_BINDING=" --cpu_bind=mask_cpu: 0xf00000f,0xf00000f0,0xf00000f00,0xf00000f000,0xf00000f0000,0xf00000f00000"
  srun  –n $tasks   -c $OMP_NUM_THREADS  $SRUN_BINDING  ./binary
```

Rank	Hex	Cores											
		47-44	43-40	39-36	35-32	31-28	27-24	23-20	19-16	15-12	11-8	7-4	3-0
0	F00000F						1111	0000	0000	0000	0000	0000	1111
1	F00000F0					1111	0000	0000	0000	0000	0000	1111	0000
2	F00000F00				1111	0000	0000	0000	0000	0000	1111	0000	0000
3	F00000F000			1111	0000	0000	0000	0000	0000	1111	0000	0000	0000
4	F00000F0000		1111	0000	0000	0000	0000	0000	1111	0000	0000	0000	0000
5	F00000F00000	1111	0000	0000	0000	0000	0000	1111	0000	0000	0000	0000	0000

srun : Task placement when using logical cores

- PPN=4 and OMP_NUM_THREADS=12

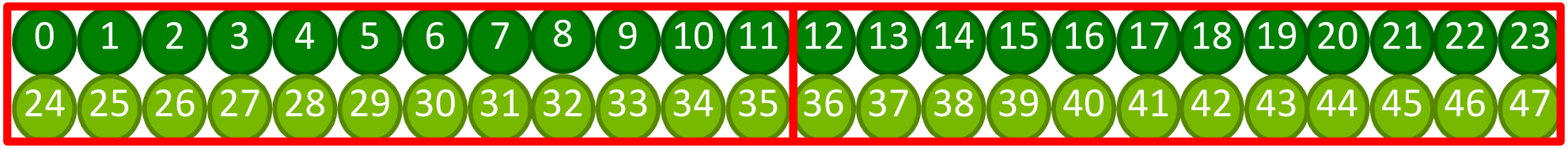



```
elif [ $OMP_NUM_THREADS == 12 ] ; then
  export SRUN_BINDING=" --cpu_bind=mask_cpu:0x3f00003f,0xfc0000fc0,0x3f00003f000,0xfc0000fc0000"
  srun  –n $tasks  -c $OMP_NUM_THREADS  $SRUN_BINDING  ./binary
```

Rank	Hex	Cores											
		47-44	43-40	39-36	35-32	31-28	27-24	23-20	19-16	15-12	11-8	7-4	3-0
0	3F00003F					0011	1111	0000	0000	0000	0000	0011	1111
1	FC0000FC0				1111	1100	0000	0000	0000	0000	1111	1100	0000
2	3F00003F000		0011	1111	0000	0000	0000	0000	0011	1111	0000	0000	0000
3	FC0000FC0000	1111	1100	0000	0000	0000	0000	1111	1100	0000	0000	0000	0000

srun : Task placement when using logical cores

- PPN=2 and OMP_NUM_THREADS=24

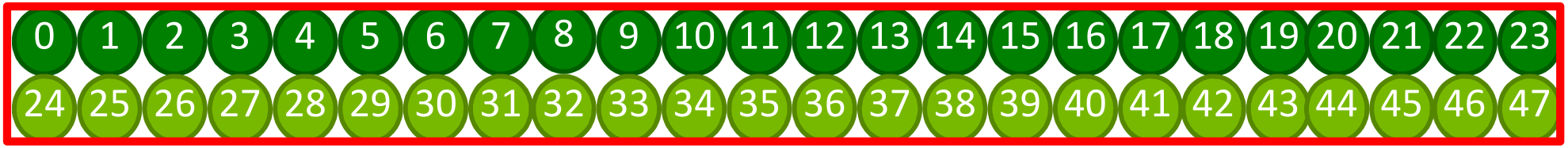



```
elif [ $OMP_NUM_THREADS == 24 ] ; then
  export SRUN_BINDING=" --cpu_bind=mask_cpu:0xfff000fff,0xfff000fff000"
  srun –n $tasks -c $OMP_NUM_THREADS   $SRUN_BINDING   ./binary
```

Rank	Hex			Cores									
		47-44	43-40	39-36	35-32	31-28	27-24	23-20	19-16	15-12	11-8	7-4	3-0
0	FFF000FFF				1111	1111	1111	0000	0000	0000	1111	1111	1111
1	FFF000FFF000	1111	1111	1111	0000	0000	0000	1111	1111	1111	0000	0000	0000

srun : Task placement when using logical cores

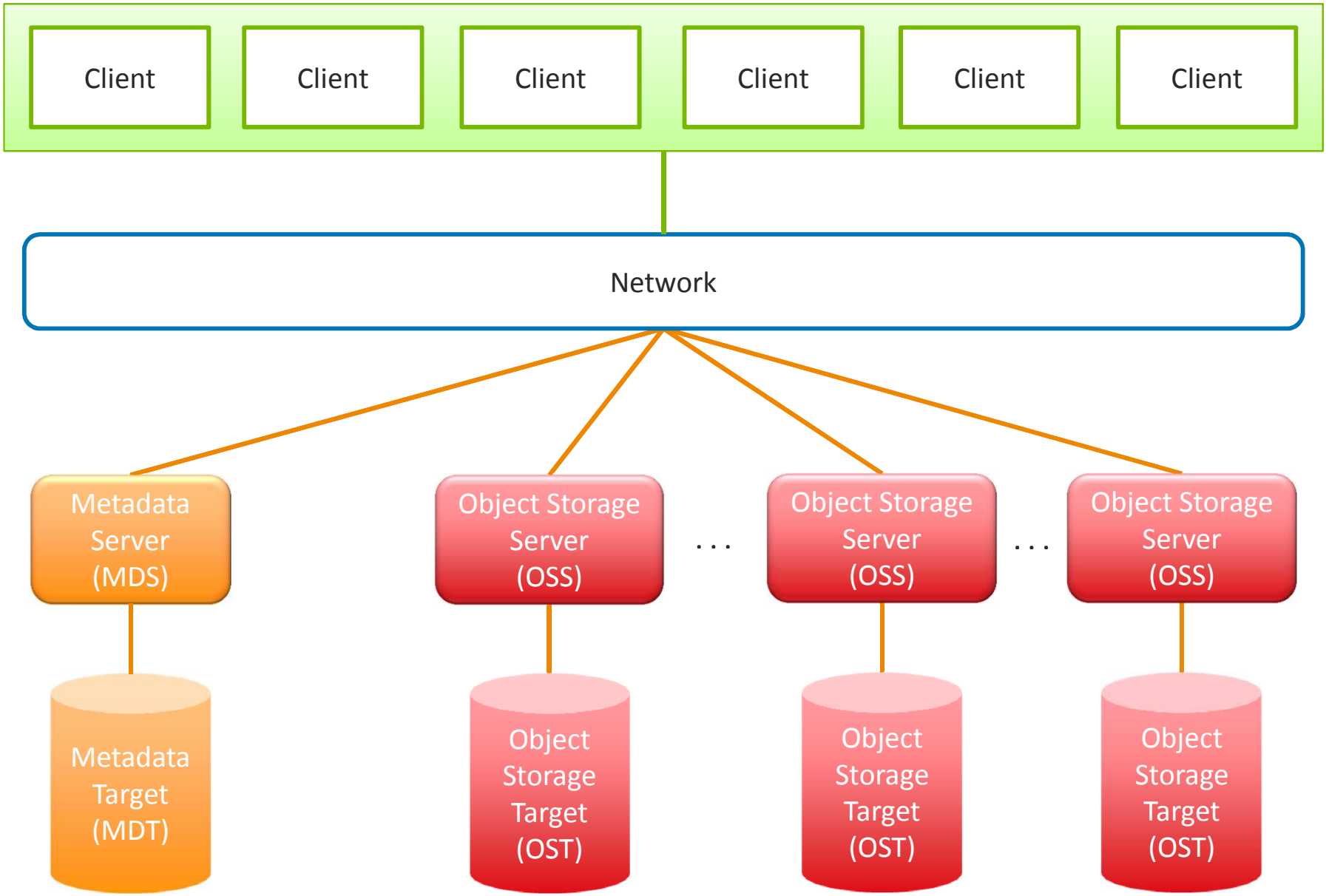
- PPN=1 and OMP_NUM_THREADS=48




```
elif [ $OMP_NUM_THREADS == 48 ] ; then
  export SRUN_BINDING=" --cpu_bind=mask_cpu:0xffffffffffff "
 srun  -n $tasks   -c $OMP_NUM_THREADS  $SRUN_BINDING   ./binary
```

Rank	Hex	Cores											
		47-43	43-40	39-36	35-32	31-28	27-24	23-20	19-16	15-12	11-8	7-4	3-0
0	FFFFFFFFFFFF	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111	1111

Lustre Components



LUSTRE: File Striping

- At the application level, it is possible to change the number of OST used for read or write
- In this case Lustre stripes the file data across the OSTs in a round robin fashion.
- The stripe size must be a multiple of the page size. The smallest recommended stripe size is 1 MB because Lustre tries to batch data into 1 MB blocks.

COMMAND	DESCRIPTION
lfs getstripe <directory>	Displays information on the current stripe count and size
lfs setstripe –c <nb OST> <directory>	Define the number of OSTs used to access this directory
lfs setstripe –c -1 <directory>	Use the maximum number of OSTs
lfs setstripe -c 1 -s $((2*1024*1024)) <dir>	Sets the stripe count to 1 and the stripe size to 2 MB

LUSTRE: Set Stripe Count – Existing File

- If the file still exists, it is not possible to change the stripe count, but you can create a new file, copy the original file into the new one, and then rename the new file with the original file name, as shown in the example commands below

```
lfs setstripe -c 20 /mnt/lustre/lfs_file.new
cp /mnt/lustre/lfs_file /mnt/lustre/lfs_file.new
mv /mnt/lustre/lfs_file.new /mnt/lustre/lfs_file
lfs getstripe -c /mnt/lustre/lfs_file
```

LUSTRE: Rules to set the Stripe Count

- If more than one compute node access the same file (one single file), then set the stripe count to a number that is a multiple of the number nodes(NCN), but equal to or less than the number of OSTs(NOST):
  - If NCN=10 and NOST=15, set the stripe count to 10
  - If NCN=10 and NOST=32, set the stripe count to 30
  - If NCN=100 and NOST=32, set the stripe count to 32

- If the number of files the application is accessing:
  - is smaller than the number of OSTs, select the appropriate stripe count so that you use all the OSTs
    - For example, if you have 8 OSTs and write 4 files at the same time, then set the stripe count to 2
  - is bigger than the number of OSTs, then set the stripe count to 1.

LUSTRE: Rules to set the Stripe Count

- If more than one compute node access the same file (one single file), then set the stripe count to a number that is a multiple of the number nodes(NCN), but equal to or less than the number of OSTs(NOST):
  - If NCN=10 and NOST=15, set the stripe count to 10
  - If NCN=10 and NOST=32, set the stripe count to 30
  - If NCN=100 and NOST=32, set the stripe count to 32

- If the number of files the application is accessing:
  - is smaller than the number of OSTs, select the appropriate stripe count so that you use all the OSTs
    - For example, if you have 8 OSTs and write 4 files at the same time, then set the stripe count to 2
  - is bigger than the number of OSTs, then set the stripe count to 1.

Architect of an Open World™

DKRZ
DEUTSCHES
KLIMARECHENZENTRUM

SC
science + computing
A Bull Group Company