UNIVERSITY OF BERGEN
SCIENTIFIC COMPUTING, IT DEPARTMENT

2018 HPC COURSE
Slurm on Hexagon

Saerda.Halifu

UNIVERSITY OF BERGEN

SLURM

Simple Linux Utility for Resource Management

60% of top 500 is using slurm

https://slurm.schedmd.com

https://docs.hpc.uib.no/wiki/Main_Page

SLURM on Hexagon

- We are running slurm-17.02.7 version on hexaogn.
- 312 nodes (312*32=9984 cores)

  2 x 16 cores Interlagos CPUs

  32 GB of RAM
- Make sure you always have “slurm “ module loaded, normally it loads by default every time you logged in to hexagon.

Example SLURM JOB SCRIPT

```
#!/bin/bash
#SBATCH -J "seqjob
#SBATCH -A CPUaccount
#SBATCH -t 60:00:00
#SBATCH –mail-type=ALL
#SBATCH –mail-user=forexample@uib.no
#SBATCH –output=serial_test.out
 cd /work/users/$USER/
 aprun -n 1 -N 1 -m 32000M ./program
```

SBATCH options

```
# Set name of job
#SBATCH --job-name=test123

# Set max wallclock time
#SBATCH –time=60:00:00
```

Mail alert at start, end and abortion of execution

#SBATCH --mail-type=ALL

(BEGIN, END, FAIL,REQUEUE)

Send mail to this address

#SBATCH --mail-user=john@gmail.com

```
# Set the number of nodes and processes per node
#SBATCH --nodes=2


# Set the number of tasks (processes) per node.
#SBATCH --ntasks-per-node=16
```

To submit your job to queue run :

sbatch job.sh

If submission is successful, it will return job ID.

Starting your program inside Sbatch

1. Srun

srun ./mpitest

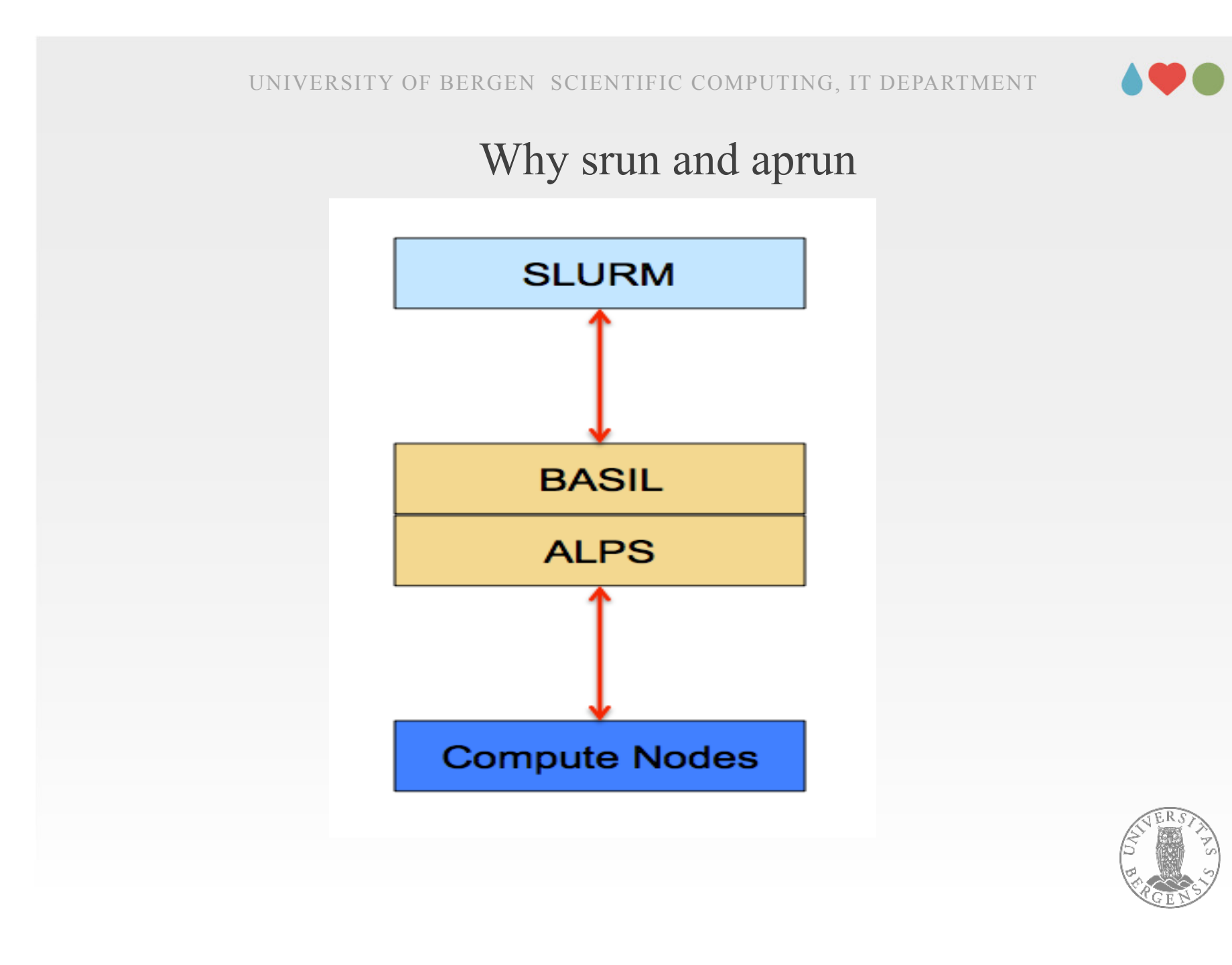
To use srun, one has to fine tune resource request with #SBATCH , number of nodes, core per node etc.

Starting your program inside Sbatch

2. aprun (recommended)

```
aprun -n32 -N32 -m1000M    ./mpitest
```

In this case one has to specify total number of cores needed in #SBATCH, rest can be tuned by aprun



Why srun and aprun

Usefull Commands

sinfo - reports the state of partitions and nodes.

squeue - reports the state of jobs or job steps.

sbatch - submit a job script for later execution.

scancel - cancel a pending or running job or job step

srun - submit a job for execution or initiate job steps in real time

apstat - Provides status information for Cray XT systems applications

xtnodestat Shows information about compute and service partition processors and the jobs running in each partition

Information on jobs

List all running jobs:

squeue
squeue --Format=jobid,username

List all current jobs for a user:

squeue -u <username>

List all running jobs for a user:

squeue -u <username> -t RUNNING (PENDING)

List detailed information for a job:

scontrol show jobid <jobid>

scontrol show jobid <jobid> -dd

To get statistics on completed jobs:

sacct -j <jobid> --format=JobID,JobName,MaxRSS,Elapsed

To view the same information for all jobs of a user:

sacct -u <username> --format=JobID,JobName,MaxRSS,Elapsed

Controlling jobs

To cancel one job:

scancel <jobid>

To temporarily hold job:

scontrol hold <jobid>

You can resume it by:

scontrol resume <jobid>

Hexagon Specific APRUN Arguments

-B use parameters from batch-system (mppwidth,mppnppn,mppmem,mppdepth used with PBS)

-N processors per node

-n processing elements

-d number of threads

-m memory per element suffix

Requested resource in SBATCH

Match the arguments for aprun.

So if you ask for "#SBATCH --mem=900mb" you will need to add the argument "-m 900M" to aprun.

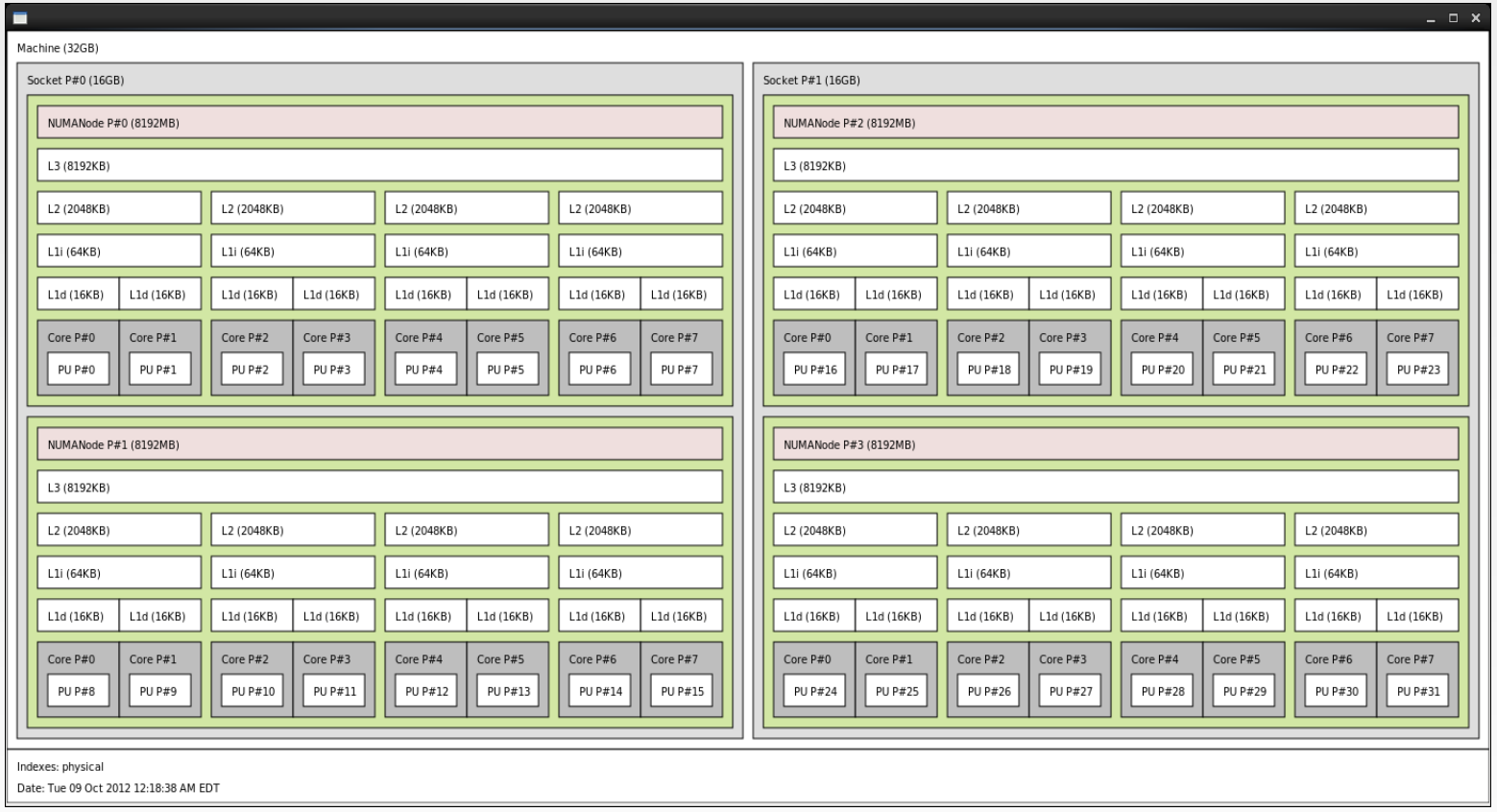
Efficiently Use Recourses

1.Nodes on hexagon can not be shared between jobs , thus if you use one core on the node you will be “charged” for whole node (32 cores), in that case your take advantage and use all the memory on the node.

2. Run with OpenMP or with precise placement

If you are using OpenMP you need to specifically map your cores

aprun -n xx -N 16 -S 4 -cc 0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30 ./mycode


Machine (32GB)
Socket P#0 (16GB)
NUMANode P#0 (8192MB)
L3 (8192KB)
L2 (2048KB)
L1i (64KB)
L1d (16KB)
Core P#0
PU P#0
Core P#1
PU P#1
Core P#2
PU P#2
Core P#3
PU P#3
Core P#4
PU P#4
Core P#5
PU P#5
Core P#6
PU P#6
Core P#7
PU P#7
NUMANode P#1 (8192MB)
PU P#8
PU P#9
PU P#10
PU P#11
PU P#12
PU P#13
PU P#14
PU P#15
Socket P#1 (16GB)
NUMANode P#2 (8192MB)
PU P#16
PU P#17
PU P#18
PU P#19
PU P#20
PU P#21
PU P#22
PU P#23
NUMANode P#3 (8192MB)
PU P#24
PU P#25
PU P#26
PU P#27
PU P#28
PU P#29
PU P#30
PU P#31
Indexes: physical
Date: Tue 09 Oct 2012 12:18:38 AM EDT

SLURM interactive job on Hexagon

salloc -n 32 -t 00:30:00

There will be warning message, you can safely ignore it.

If you want to X11 enabled on interactive mode

You just have to ssh to hexagon with –Y option

SLURM JOB ARRAY

Job arrays allow you to submit many jobs at once, as one cohesive group.

This is useful if you want to run the same program(s) on different data sets.

Instead of submitting many jobs in a loop or using multiple bash input scripts you should simply use the array syntax.

For example, if you have two input data files input0.py and input1.py, instead of using two separate submission scripts, You can simply use one submission script:

```
#!/bin/bash
#SBATCH --time=01:00:00       # walltime
#SBATCH --ntasks=1            # number of processor cores (i.e. tasks)
#SBATCH --nodes=1             # number of nodes
#SBATCH --mem-per-cpu=1024M   # memory per CPU core
#SBATCH -J "MyArrayJob"       # job name
#SBATCH --array=0-1           # job array of size 2

module load python
aprun –n1 python input${SLURM_ARRAY_TASK_ID}.py # this will expand to be: 'python input0.py' or 'python input1.py'

exit 0
```

Each index in the job array gets its own job ID, but the entire array has it's own master job ID.

For example, if the sbatch command responds Submitted batch job 36, then the environment variables will be set as follows:

```
SLURM_JOBID=36
SLURM_ARRAY_JOB_ID=36
SLURM_ARRAY_TASK_ID=0

SLURM_JOBID=37
SLURM_ARRAY_JOB_ID=36
SLURM_ARRAY_TASK_ID=1
```

Cancel SLURM ARRAY JOBS

```
# Cancel array ID 1 to 3 from job array 20
$ scancel 20_[1-3]

# Cancel array ID 4 and 5 from job array 20
$ scancel 20_4 20_5

# Cancel all elements from job array 20
$ scancel 20
```

SLURM Job dependencies

A job can be given the constraint that it only starts after another job has finished.

we have two Jobs, A and B. We want Job B to start after Job A has successfully completed.

First we start Job A by submitting it via sbatch:

```
$ sbatch jobA.sh
```

Making note of the assigned job ID for Job A, we then submit Job B with the added condition that it only starts after Job A has successfully completed:

```
$ sbatch --dependency=afterok:<jobID_A> jobB.sh
```

If we want Job B to start after several other Jobs have completed, we can specify additional jobs, using a ':' as a delimiter:

```
$ sbatch --dependency=afterok:<jobID_A:jobID_C:jobID_D> jobB.sh
```

We can also tell slurm to run Job B, even if Job A fails, like so:

sbatch --dependency=afterany:<jobID_A> jobB.sh

Useful Note

- CPU accounting is not enforced on hexagon
- Slurm starts your job from where your submission script is, PBS starts from your home directory.

- Don’t have to specify partition

- Be responsible when you open up write permission to your work folder.

Quick Guide to translate PBS/Torque to SLURM

User commands	PBS/Torque	SLURM
Job submission	qsub [filename]	sbatch [filename]
Job deletion	qdel [job_id]	scancel [job_id]
Job status (by job)	qstat [job_id]	squeue --job [job_id]
Full job status (by job)	qstat -f [job_id]	scontrol show job [job_id]
Job status (by user)	qstat -u [username]	squeue --user=[username]

Environment variables	PBS/Torque	SLURM
Job ID	$PBS_JOBID	$SLURM_JOBID
Submit Directory	$PBS_O_WORKDIR	$SLURM_SUBMIT_DIR
Node List	$PBS_NODEFILE	$SLURM_JOB_NODELIST

Job specification	PBS/Torque	SLURM
Script directive	#PBS	#SBATCH
Job Name	-N [name]	--job-name=[name] OR -J [name]
Node Count	-l nodes=[count]	--nodes=[min[-max]] OR -N [min[-max]]
CPU Count	-l ppn=[count]	--ntasks-per-node=[count]
CPUs Per Task		--cpus-per-task=[count]
Memory Size	-l mem=[MB]	--mem=[MB] OR --`mem-per-cpu=[MB]
Wall Clock Limit	-l walltime=[hh:mm:ss]	--time=[min] OR --time=[days-hh:mm:ss]
Node Properties	-l nodes=4:ppn=8:[property]	--constraint=[list]
Standard Output File	-o [file_name]	--output=[file_name] OR -o [file_name]
Standard Error File	-e [file_name]	--error=[file_name] OR -e [file_name]
Combine stdout/stderr	-j oe (both to stdout)	(Default if you don't specify --error)
Job Arrays	-t [array_spec]	--array=[array_spec] OR -a [array_spec]
Delay Job Start	-a [time]	--begin=[time]

Hexagon Support :

support@hpc.uib.no

Fram Support :

support@metacenter.no

THANK YOU!

UNIVERSITY OF BERGEN SCIENTIFIC COMPUTING, IT DEPARTMENT

Eco-Lighthouse®