



Uni.lu HPC School 2021

PS4: HPC Management of Sequential and Embarrassingly Parallel Jobs

High Performance
Computing &
Big Data Services



Uni.lu High Performance Computing (HPC) Team
Dr. S. Varrette

University of Luxembourg (UL), Luxembourg
<http://hpc.uni.lu>



Latest versions available on Github:



UL HPC tutorials:

<https://github.com/ULHPC/tutorials>

UL HPC School:

hpc.uni.lu/education/hpcschool

PS4 tutorial sources:

ulhpc-tutorials.rtf.d.io/en/latest/sequential/basics





Summary

1 Introduction

[Serial] Task, Job and scheduling considerations
Slurm Launchers Templates for Sequential Applications

Main Objectives of this Session

- Understanding the **jobs and tasks concepts**
- Recognizing **embarrassingly parallel tasks**
 - ↳ Understanding scheduling challenges induced by their execution even on an HPC facility
- Guidelines to the **optimized** design of HPC job campaign in such case
 - ↳ prepare `run_*` task script
 - ↳ **launcher.* design** for tasklist management, exploiting Slurm & HW characteristics
 - ↳ **interactive** tests and performance analysis
 - ↳ **passive runs** and speedup analysis compared to seq. executions **up to 92% improvement**
- Discovering GNU Parallel
 - ↳ usage and exploitation of ULHPC Generic launcher for GNU parallel

Test Case for the session

- Sample “Stress Me!” parameter exploration job campaign
 - ↳ `run_stressme <N>` imposes configurable amount of *stress* on 1 core for <N> seconds

Jobs, Tasks & Local Execution



```
$> ./myprog
```



Jobs, Tasks & Local Execution



```
$> ./myprog
```



Jobs, Tasks & Local Execution



```
$> ./myprog  
$> ./myprog -n 10
```



Jobs, Tasks & Local Execution



```
$> ./myprog  
$> ./myprog -n 10
```



Jobs, Tasks & Local Execution



```
$> ./myprog  
$> ./myprog -n 10  
$> ./myprog -n 100
```



Jobs, Tasks & Local Execution



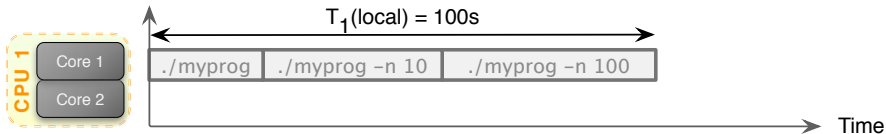
```
$> ./myprog  
$> ./myprog -n 10  
$> ./myprog -n 100
```



Jobs, Tasks & Local Execution



```
$> ./myprog  
$> ./myprog -n 10  
$> ./myprog -n 100
```



Jobs, Tasks & Local Execution



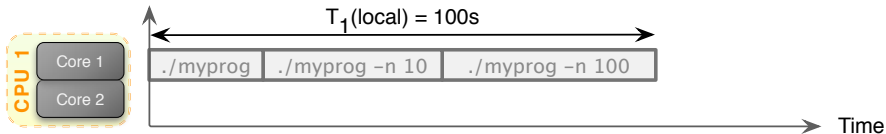
Job(s)

```
$> ./myprog
$> ./myprog -n 10
$> ./myprog -n 100
```

3

Task(s)

3



Jobs, Tasks & Local Execution



```
# launcher  
  
./myprog  
./myprog -n 10  
./myprog -n 100
```



Jobs, Tasks & Local Execution



```
# launcher  
  
./myprog  
./myprog -n 10  
./myprog -n 100
```



Jobs, Tasks & Local Execution



```
# launcher  
  
./myprog  
./myprog -n 10  
./myprog -n 100
```



Jobs, Tasks & Local Execution



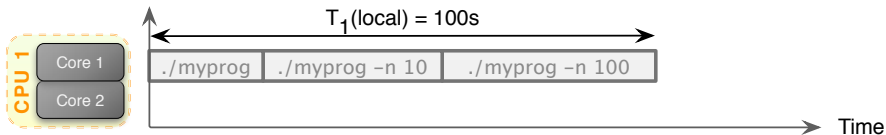
```
# launcher  
  
./myprog  
./myprog -n 10  
./myprog -n 100
```



Jobs, Tasks & Local Execution



```
# launcher  
  
./myprog  
./myprog -n 10  
./myprog -n 100
```



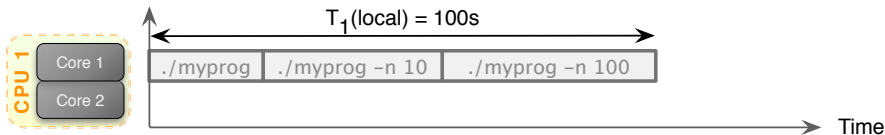
Jobs, Tasks & Local Execution



```
# launcher
./myprog
./myprog -n 10
./myprog -n 100
```

(s)Job 1

Task(s) 3



Jobs, Tasks & Local Execution



```
# launcher  
  
./myprog  
./myprog -n 10  
./myprog -n 100
```



Jobs, Tasks & Local Execution



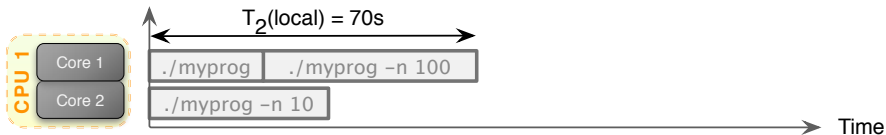
```
# launcher2
"Run in //:"
./myprog
./myprog -n 10
./myprog -n 100
```



Jobs, Tasks & Local Execution



```
# launcher2
"Run in //:"
./myprog
./myprog -n 10
./myprog -n 100
```



Jobs, Tasks & Local Execution



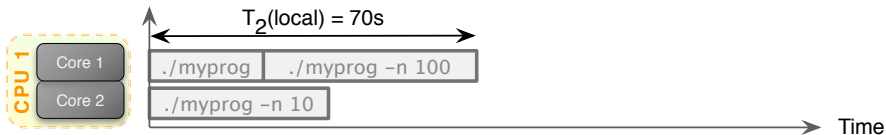
```
# launcher2
"Run in //:"
./myprog
./myprog -n 10
./myprog -n 100
```

(s)qor

1

Task(s)

3

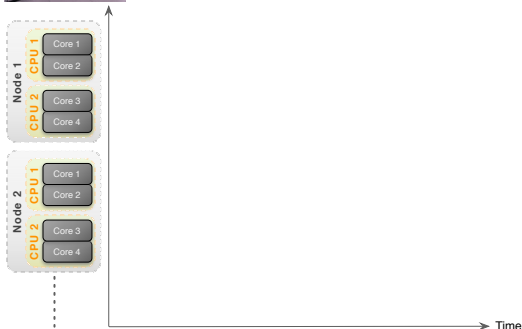


Jobs, Tasks & HPC Execution



```
# launcher
./myprog
./myprog -n 10
./myprog -n 100
```

- **Note:** Simplified example! in practice:
 - iris nodes (28c): 2 CPUs of 14 cores each
 - aion nodes (128c): (virtual) 8 CPUs of 16 cores each

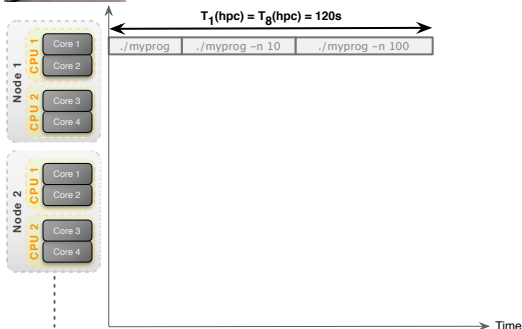


Jobs, Tasks & HPC Execution



```
# launcher
./myprog
./myprog -n 10
./myprog -n 100
```

- Note: Simplified example! in practice:
 - iris nodes (28c): 2 CPUs of 14 cores each
 - aion nodes (128c): (virtual) 8 CPUs of 16 cores each



Jobs, Tasks & HPC Execution



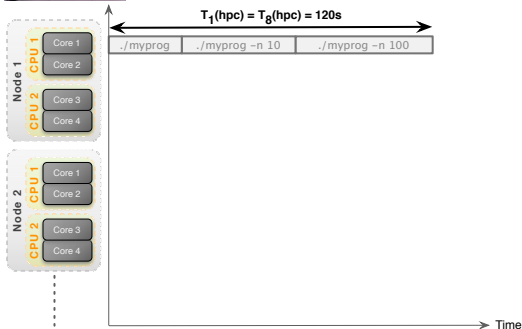
```
# launcher
./myprog
./myprog -n 10
./myprog -n 100
```

Job(s) 1

Task(s) 3

- Note: Simplified example! in practice:

- iris nodes (28c): 2 CPUs of 14 cores each
- aion nodes (128c): (virtual) 8 CPUs of 16 cores each



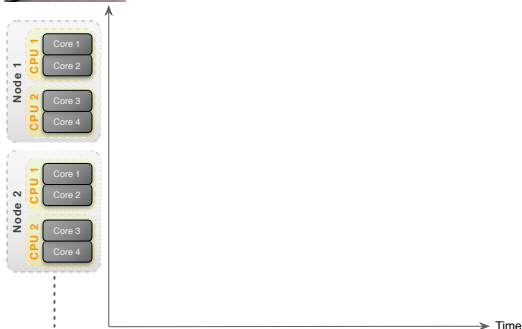
Jobs, Tasks & HPC Execution



```
# launcher2
"Run in //:"
./myprog
./myprog -n 10
./myprog -n 100
```

● **Note:** Simplified example! in practice:

- iris nodes (28c): 2 CPUs of 14 cores each
- aion nodes (128c): (virtual) 8 CPUs of 16 cores each



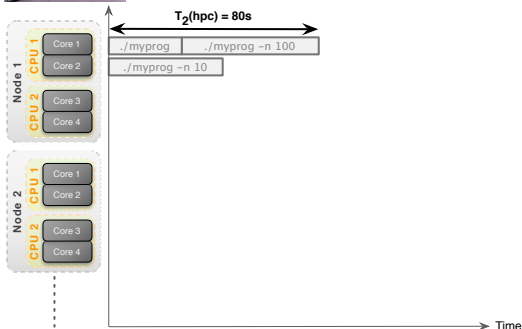
Jobs, Tasks & HPC Execution



```
# launcher2
"Run in //:"
./myprog
./myprog -n 10
./myprog -n 100
```

- Note: Simplified example! in practice:

- iris nodes (28c): 2 CPUs of 14 cores each
- aion nodes (128c): (virtual) 8 CPUs of 16 cores each



Jobs, Tasks & HPC Execution



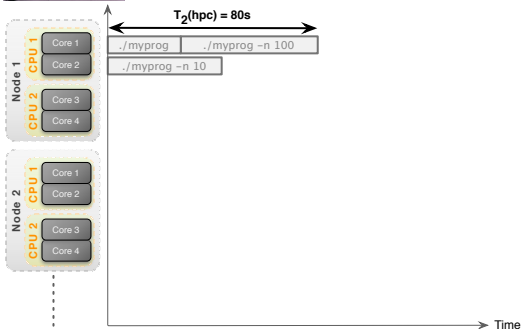
```
# launcher2
"Run in //:"
./myprog
./myprog -n 10
./myprog -n 100
```

Job(s) 1

Task(s) 3

- Note: Simplified example! in practice:

- iris nodes (28c): 2 CPUs of 14 cores each
- aion nodes (128c): (virtual) 8 CPUs of 16 cores each



Jobs, Tasks & HPC Execution



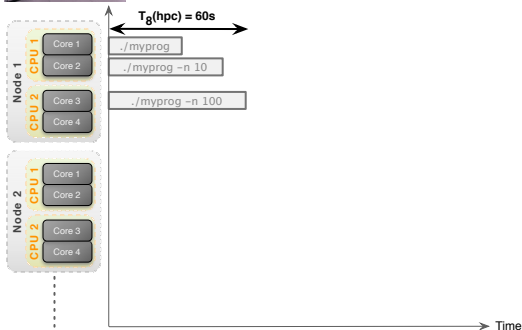
```
# launcher2
"Run in //:"
./myprog
./myprog -n 10
./myprog -n 100
```

Job(s) 1

Task(s) 3

- Note: Simplified example! in practice:

- iris nodes (28c): 2 CPUs of 14 cores each
- aion nodes (128c): (virtual) 8 CPUs of 16 cores each



Local vs. HPC Executions

Context	Local PC	HPC
Sequential	$T_1(\text{local}) = 100$	$T_1(\text{hpc}) = 120\text{s}$
Parallel/Distributed	$T_2(\text{local}) = 70\text{s}$	$T_2(\text{hpc}) = 80\text{s}$ $T_8(\text{hpc}) = 60\text{s}$

Local vs. HPC Executions

Context	Local PC	HPC
Sequential	$T_1(\text{local}) = 100$	$T_1(\text{hpc}) = 120\text{s}$
Parallel/Distributed	$T_2(\text{local}) = 70\text{s}$	$T_2(\text{hpc}) = 80\text{s}$ $T_8(\text{hpc}) = 60\text{s}$

- Sequential runs **WON'T BE FASTER** on HPC
 ↳ Reason: Processor Frequency (typically $\geq 3\text{GHz}$ vs $\geq 2\text{GHz}$)

Local vs. HPC Executions

Context	Local PC	HPC
Sequential	$T_1(\text{local}) = 100$	$T_1(\text{hpc}) = 120\text{s}$
Parallel/Distributed	$T_2(\text{local}) = 70\text{s}$	$T_2(\text{hpc}) = 80\text{s}$
		$T_8(\text{hpc}) = 60\text{s}$

- Sequential runs **WON'T BE FASTER** on HPC
 - Reason: Processor Frequency (typically $\geq 3\text{GHz}$ vs $\geq 2\text{GHz}$)
- Parallel/Distributed runs **DO NOT COME FOR FREE**
 - runs **will be sequential** even if you reserve ≥ 2 cores/nodes
 - you have to **explicitly** adapt your jobs to benefit from the multi-cores/nodes

Serial Task script Launcher

```
#!/bin/bash -l      # <--- DO NOT FORGET '-l'
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
print_error_and_exit() { echo "***ERROR*** $*"; exit 1; }
module purge || print_error_and_exit "No 'module' command"
# C/C++: module load toolchain/intel # OR: module load toolchain/foss
# Java: module load lang/Java/1.8
# Ruby/Perl/Rust...: module load lang/{Ruby,Perl,Rust...}
# /\ ADAPT TASK variable accordingly - absolute path to the (serial) task to be executed
TASK=${TASK:=${HOME}/bin/app.exe}
OPTS=$*

${TASK} ${OPTS}
```

Serial Python Slurm Launcher

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -c 1
#SBATCH --time=0-01:00:00
#SBATCH -p batch
print_error_and_exit() { echo "***ERROR*** $*"; exit 1; }
module purge || print_error_and_exit "No 'module' command"
# Python 3.X by default (also on system)
module load lang/Python
# module load lang/SciPy-bundle
# and/or: activate the virtualenv <name> you previously generated with
#     python -m venv <name>
source ./<name>/bin/activate

python [...]
```

R Slurm Launcher

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -c 28
#SBATCH --time=0-01:00:00
#SBATCH -p batch

print_error_and_exit() { echo "***ERROR*** $*"; exit 1; }
module purge || print_error_and_exit "No 'module' command"
module load lang/R
export OMP_NUM_THREADS=${SLURM_CPUS_PER_TASK:-1}

Rscript <script>.R |& tee job_${SLURM_JOB_NAME}.out
```



MATLAB Slurm Launcher

```
#!/bin/bash -l
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH -c 28
#SBATCH --time=0-01:00:00
#SBATCH -p batch

print_error_and_exit() { echo "***ERROR*** $*"; exit 1; }
module purge || print_error_and_exit "No 'module' command"
module load base/MATLAB

matlab -nodisplay -nosplash < INPUTFILE.m > OUTPUTFILE.out
```

Sample “Stress Me!” parameter exploration

Your Turn!

Hands-on Pre-requisites

► url ◀ | github | src

- Access to ULHPC facility ssh
- (eventually) Open Tmux session (Aion/Iris) `~/.tmux.conf, tmux new -s <name>`
 ↪ OR (eventually) Open GNU Screen (Iris) `~/.screenrc, screen -S <name>`
- Clone/Pull [ULHPC/tutorials](#) repository `~/git/github.com/ULHPC/tutorials`
- Prepare dedicated directory `~/tutorials/sequential` for this session

```
(access)$> mkdir -p ~/tutorials/sequential
(access)$> cd ~/tutorials/sequential
# create a symbolic link to the reference material
(access)$> ln -s ~/git/github.com/ULHPC/tutorials/sequential/basics ref.d
```

Sample “Stress Me!” parameter exploration

- **Objective:** mimic (integer) parameter exploration in $[1 \dots N]$ for serial (1-core) tasks
 - ↳ each task last different amount time (here ranging ranging from 1 to n seconds)
 - ↳ see `scripts/run_stressme`
 - ✓ stress-based workload generator that imposes a configurable amount of stress on the system

```
./scripts/run_stressme -h
```

NAME

`run_stressme`: A sample workload generator that imposes a stress on the computing node using the '`stress`' command.

USAGE

`./scripts/run_stressme [-n] [N]`: run serial stress during N second (Default: 20)

OPTIONS

`-n --noop --dry-run`: Dry run mode

Sample “Stress Me!” parameter exploration

N=#tasks	Expected Seq. time (T_1)	Optimal time (T_∞)
1	1s	1s
10	55s (~1 min)	10s
30 (default)	465s (7 min 45s)	30s
100	5050s (1h 24 min 10s)	100s

Hands-on: Single task run (interactive)

► [url](#) ◀ | [github](#) | [src](#)

- Get an interactive job
- Open another terminal (or another screen tab/windows)
 - ↪ connect to your running job
 - ↪ monitor it's load with `htop`
- Execute `run_stressme` tasks for various parameters
 - ↪ monitor its effect on second window in `htop`

```
si
sjoin <JOBID>
htop
```

Sample “Stress Me!” parameter exploration

Hands-on: A First launcher (1 job, 1 task on one core)

► [url](#) ◀ | [github](#) | [src](#)

- Copy and adapt **ULHPC** Generic serial launcher `launcher.stressme-serial.sh`
- Get an **interactive** job `si`
 - ↪ dry-run vs real execution
- Test **passive** job execution `sbatch`

Sample “Stress Me!” parameter exploration

- Previous approach works but will **induces an non-necessary overload of the scheduler** when scaling the number of tasks (Ex: job campaign to 1000 or 10000 test cases)
 - ↪ **To complete N (serial) tasks, you need to submit N jobs**
 - ✓ **DO NOT make a for loop!!!**: better efficient alternatives **MUST** be used
 - ✓ You will likely wait (sometimes very long time) to have all your jobs completed
 - ↪ Node coverage is sub-optimal
 - ✓ **your serial jobs can be spread on up to N different nodes**
 - ↪ Very often (90% of the cases), your serial tasks are short-lived (couple of seconds)
 - ✓ You loose more time to schedule jobs than actually running them

Sample “Stress Me!” parameter exploration

- Previous approach works but will **induces an non-necessary overload of the scheduler** when scaling the number of tasks (Ex: job campaign to 1000 or 10000 test cases)
 - ↪ **To complete N (serial) tasks, you need to submit N jobs**
 - ✓ **DO NOT make a for loop!!!**: better efficient alternatives **MUST** be used
 - ✓ You will likely wait (sometimes very long time) to have all your jobs completed
 - ↪ Node coverage is sub-optimal
 - ✓ **your serial jobs can be spread on up to N different nodes**
 - ↪ Very often (90% of the cases), your serial tasks are short-lived (couple of seconds)
 - ✓ You loose more time to schedule jobs than actually running them
- You **SHOULD** restrict to 1 node **YET** exploit at best the many cores available
 - ↪ Recall: 128 cores per Aion nodes, 28 cores per Iris nodes
 - ↪ Make 1 jobs (for a full nodes) and use it to schedule **inside** your tasks
 - ✓ From scheduler point of view: **only 1 job to schedule** (faster)
 - ✓ **Efficient**: exploit at best the available cores

Embarrassingly Parallel Jobs

(1 node, 1 job, #cores tasks per node)

- **VERY BAD / To avoid:** Fork [bomb] for loop of sbatch/srun
 - ↳ (dangerous) ampersand & to spawns the command in the background
 - ↳ optimize wait synchronization barrier YET prefer GNU Parallel

```
TASK=run_stressme
ncores=${SLURM_NTASKS_PER_NODE:-$(nproc --all)} # Number of cores per node
for i in {1..1000}; do # DON'T DO THAT - just to illustrate
    srun -nl --exclusive -c 1 --cpu-bind=cores ${TASK} $i & # NO!! Fork 1000 sub-processes
done
wait
```

Embarrassingly Parallel Jobs

(1 node, 1 job, #cores tasks per node)

- **VERY BAD / To avoid:** Fork [bomb] for loop of sbatch/srun
 - ↳ (dangerous) ampersand & to spawns the command in the background
 - ↳ optimize wait synchronization barrier YET prefer GNU Parallel

```
TASK=run_stressme
ncores=${SLURM_NTASKS_PER_NODE:-$(nproc --all)} # Number of cores per node
for i in {1..1000}; do # DON'T DO THAT - just to illustrate
    srun -nl --exclusive -c 1 --cpu-bind=cores ${TASK} $i &
    [[ $((i%ncores)) -eq 0 ]] && wait # Better but suboptimal
done
wait
```

Embarrassingly Parallel Jobs

(1 node, 1 job, #cores tasks per node)

- **VERY BAD / To avoid:** Fork [bomb] for loop of sbatch/srun
 - ↳ (dangerous) ampersand & to spawns the command in the background
 - ↳ optimize wait synchronization barrier YET prefer GNU Parallel
- **BAD / To avoid:** Job arrays `sbatch --array 1-1000%128 [...]`
 - ↳ in all cases, job array capabilities are drastically reduced (on purpose)
 - ✓ `scontrol show config | grep -i maxarray`

Job Array Variable	Description
<code>\$SLURM_ARRAY_JOB_ID</code>	first job ID of the array
<code>\$SLURM_ARRAY_TASK_ID</code>	current job array step index value.
<code>\$SLURM_ARRAY_TASK_COUNT</code>	number of tasks in the job array.

Embarrassingly Parallel Jobs

(1 node, 1 job, #cores tasks per node)

- **VERY BAD / To avoid:** Fork [bomb] for loop of sbatch/srun
 - ↳ (dangerous) ampersand & to spawns the command in the background
 - ↳ optimize wait synchronization barrier YET prefer GNU Parallel
- **BAD / To avoid:** Job arrays sbatch --array 1-1000%128 [...]
 - ↳ in all cases, job array capabilities are drastically reduced (on purpose)
 - ✓ `scontrol show config | grep -i maxarray`

Job Array Variable	Description
<code>\$SLURM_ARRAY_JOB_ID</code>	first job ID of the array
<code>\$SLURM_ARRAY_TASK_ID</code>	current job array step index value.
<code>\$SLURM_ARRAY_TASK_COUNT</code>	number of tasks in the job array.

`${TASK} ${SLURM_ARRAY_TASK_ID} # Simpler syntax YET induce same (worse) load on scheduler`

Embarrassingly Parallel Jobs

(1 node, 1 job, #cores tasks per node)

- **VERY BAD / To avoid:** Fork [bomb] for loop of sbatch/srun
 - ↳ (dangerous) ampersand & to spawns the command in the background
 - ↳ optimize wait synchronization barrier YET prefer GNU Parallel
- **BAD / To avoid:** Job arrays sbatch --array 1-1000%128 [...]
 - ↳ in all cases, job array capabilities are drastically reduced (on purpose)
 - ✓ `scontrol show config | grep -i maxarray`

Job Array Variable	Description
\$SLURM_ARRAY_JOB_ID	first job ID of the array
\$SLURM_ARRAY_TASK_ID	current job array step index value.
\$SLURM_ARRAY_TASK_COUNT	number of tasks in the job array.

- **Always prefer GNU Parallel for Embarrassingly Parallel Serial Tasks!**
 - ↳ **Good, efficient and Optimized**
 - ↳ Transparent through the provided ULHPC launcher

GNU Parallel



- A shell tool to execute **independent** tasks in parallel using one or more nodes
 - ↪ A task can be a single command or script
- Typical input could be a list of files, a list of parameters
 - ↪ Recall that in bash, "{1..10}" is expanded to "1 2 3 4 5 6 7 8 9 10"

Hands-on: GNU Parallel Build and Installation

► [url](#) ◀ | [github](#) | [src](#)

- Setup your HOME to support GNU Stow installation
 - ↪ [GNU Stow manual - tutorial](#)
- Download, build and install parallel
- Use stow to enable/disable the installed version

`stow [-D]`

On `stow -D`, you can ignore the 'BUG in find_stowed_path?' message due to recent [Issues #65](#)

GNU Parallel Syntax

GNU Parallel syntax can be a little distributing, but basically it supports two modes:

- 1 Reading command arguments **on the command line**:

```
parallel [-j N] [OPTIONS] COMMAND {} ::: TASKLIST
```

- 2 Reading command arguments **from an input file**:

```
parallel -a TASKLIST.LST [-j N] [OPTIONS] COMMAND {}  
parallel [-j N] [OPTIONS] COMMAND {} :::: TASKLIST.LST
```

- **Tips & Tricks:**

- If your COMMAND embed a pipe stage, you have to escape the pipe symbol as follows \
- **ALWAYS** echo your commands until you're satisfied

GNU Parallel Syntax

Hands-on: Discovering the parallel command

► [url](#) ◀ | [github](#) | [src](#)

- play with a TASKLIST from the command line
- play now with a TASKLIST from an input file.

GNU Parallel Syntax

```
# Default string {} --> path/to/filename.ext
parallel echo {} ::: path/to/filename.ext
# Remove extension {.} --> path/to/filename
parallel echo {.} ::: path/to/filename.ext
# Remove path {/} --> filename.ext
parallel echo {/} ::: path/to/filename.ext
# Remove path and extension {/.} --> filename
parallel echo {/.} ::: path/to/filename.ext
# Change extension and path --> output/filename.out
parallel echo output/{/.}.out ::: path/to/filename.ext
```

Back to “Stress Me!” parameter exploration

Hands-on: Best launcher based on GNU Parallel

► url ◀ | github | src

- Copy and adapt **ULHPC Generic GNU Parallel launcher**
 - ↪ `launcher.stressme.${ULHPC_CLUSTER}.sh`
 - ↪ only difference optimized default `--ntasks-per-node` for aion/iris
 - ↪ convenient symlink `launcher.stressme.sh`
- Get an **exclusive interactive** job `si --exclusive`
 - ↪ dry-run vs test vs real execution
- Test **passive** job execution `sbatch`
- Embarrassingly [GNU] parallel tasks across multiples nodes
 - ↪ Concepts and Safeguards
 - ↪ Slurm dependency mechanism `--dependency=singleton`
 - ↪ sample submission script `scripts/submit_stressme_multinode`

Back to “Stress Me!” parameter exploration

```
(access)$> ./scripts/submit_stressme_multinode -h
Usage: submit_stressme_multinode [-x] [-N MAXNODES]
    Sample submission script across multiple nodes
    Execution won't spread on more than 4 nodes (singleton dependency)
    -x --execute      really submit the jobs with sbatch
    -N --nodes MAXNODES  set max. nodes
# Target restriction to 2 running nodes max
(access)$> ./scripts/submit_stressme_multinode -N 2
(access)$> ./scripts/submit_stressme_multinode -N 2 -x
# queue -u $(whoami)
  JOBID PARTIT   QOS      NAME  NODE  CPUS ST  TIME  TIME_LEFT NODELIST(REASON)
  2175780 batch normal StressMe_0    1   28 PD  0:00   1:00:00 (Dependency)
  2175781 batch normal StressMe_1    1   28 PD  0:00   1:00:00 (Dependency)
  2175782 batch normal StressMe_0    1   28 PD  0:00   1:00:00 (Dependency)
  2175779 batch normal StressMe_1    1   28  R  0:02   59:58 iris-064
  2175778 batch normal StressMe_0    1   28  R  0:05   59:55 iris-047
```



Thank you for your attention...

Questions?

ulhpc-tutorials.rtfld.io/en/latest/sequential/basics



High Performance Computing @ Uni.lu

University of Luxembourg, Belval Campus
Maison du Nombre, 4th floor
2, avenue de l'Université
L-4365 Esch-sur-Alzette
mail: hpc@uni.lu

1

Introduction

[Serial] Task, Job and scheduling considerations
Slurm Launchers Templates for Sequential Applications

Uni.lu HPC School 2021 Contributors



Dr. Xavier Bessoron
Research Scientist



Hyacinthe Cartiaux
Infra. & HPC Arch. Engineer



Dr. Aurelien Ginohac
Research Scientist



Dr. Emmanuel Kieffer
Research Scientist



Dr. Loizos Koutsantonis
Postdoctoral Researcher



Dr. Ezhimathi Krishnasamy
Postdoctoral Researcher



Abatcha Ollah
Infra. & HPC Arch. Engineer



Dr. Tiago C. Pessoa
Postdoctoral Researcher



Sarah Peter
Infra. & Arch. Engineer



Teddy Valette
Infra. & HPC Arch. Engineer



Dr. Sebastien Varrette
Research Scientist

... and additional help (Survey, session tests)

Arylne Vandeventer
Project Manager

hpc.uni.lu