Introduction to GNU Parallel

Parallelizing Massive Individual Tasks

Siva Prasad Kasetti
HPC User Services
LSU HPC & LONI
sys-help@loni.org

Louisiana State University
Baton Rouge
09 April 2025

http://www.hpc.lsu.edu/training/archive/tutorials.php

Outline

- ❑ **Introduction**
  - ❑ What is GNU Parallel?
    - ❑ A command-line tool used for **parallelizing massive individual tasks**
  - ❑ When and Why to use GNU Parallel?
    - ❑ Examples: 1 million small MD simulations, 10k protein analysis
- ❑ **Basic Usage of GNU Parallel**
  - ❑ GNU Parallel Syntax and Options
  - ❑ Introducing Running 3 Types of jobs using GNU Parallel:
    - ❑ Serial Tasks - Run each LAMMPS task in serial
    - ❑ Multi-Threaded Tasks - Run each LAMMPS task using multiple threads
    - ❑ Small MPI Jobs - Run each LAMMPS task using multiple MPI processes
- ❑ **Proper usage of GNU Parallel**
  - ❑ Memory consideration
  - ❑ Task granularity

What is GNU Parallel?

What is GNU Parallel?

Parallelizing Massive Individual Tasks

What is GNU Parallel?

Parallelizing Massive Individual Tasks

- **Parallelize**
  - This refers to the process of dividing a workload into multiple smaller tasks and executing them simultaneously. It allows for better utilization of computational resources, such as multi-core CPUs or multiple nodes in a cluster.

What is GNU Parallel?

Parallelizing **Massive** Individual Tasks

- **Parallelize**
  - This refers to the process of dividing a workload into multiple smaller tasks and executing them simultaneously. It allows for better utilization of computational resources, such as multi-core CPUs or multiple nodes in a cluster.
- **Massive**
  - Indicates the scale or volume of tasks being handled. It often refers to workloads that consist of hundreds, thousands, or even millions of tasks, which would take a long time to process sequentially.

What is GNU Parallel?

Parallelizing Massive **Individual** Tasks

- **Parallelize**
  - This refers to the process of dividing a workload into multiple smaller tasks and executing them simultaneously. It allows for better utilization of computational resources, such as multi-core CPUs or multiple nodes in a cluster.
- **Massive**
  - Indicates the scale or volume of tasks being handled. It often refers to workloads that consist of hundreds, thousands, or even millions of tasks, which would take a long time to process sequentially.
- **Individual (Independent)**
  - Each task operates independently of the others. This means there’s no dependency between tasks, allowing them to be executed in parallel without waiting for one another.

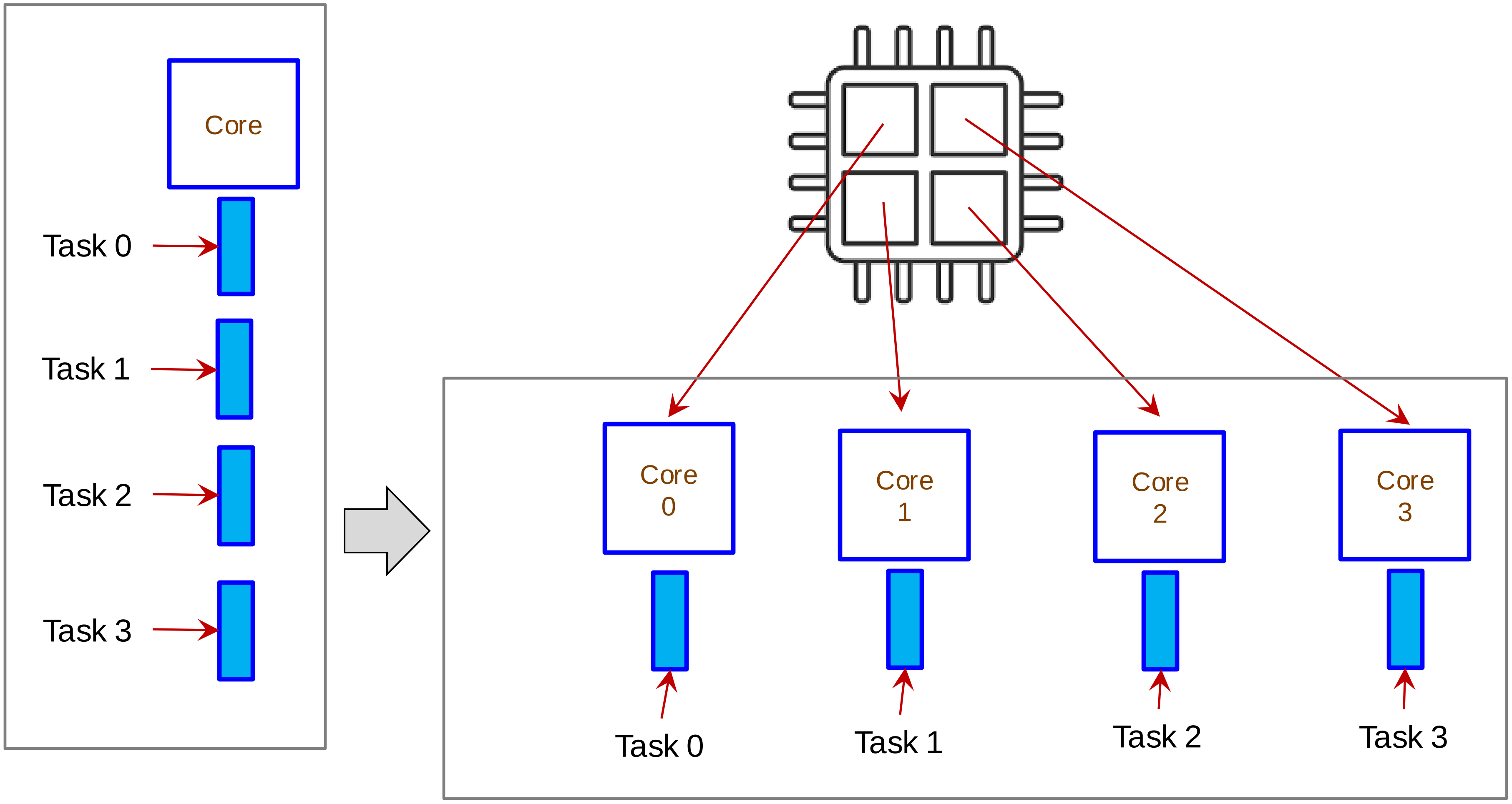
What is GNU Parallel?

Parallelizing Massive Individual Tasks

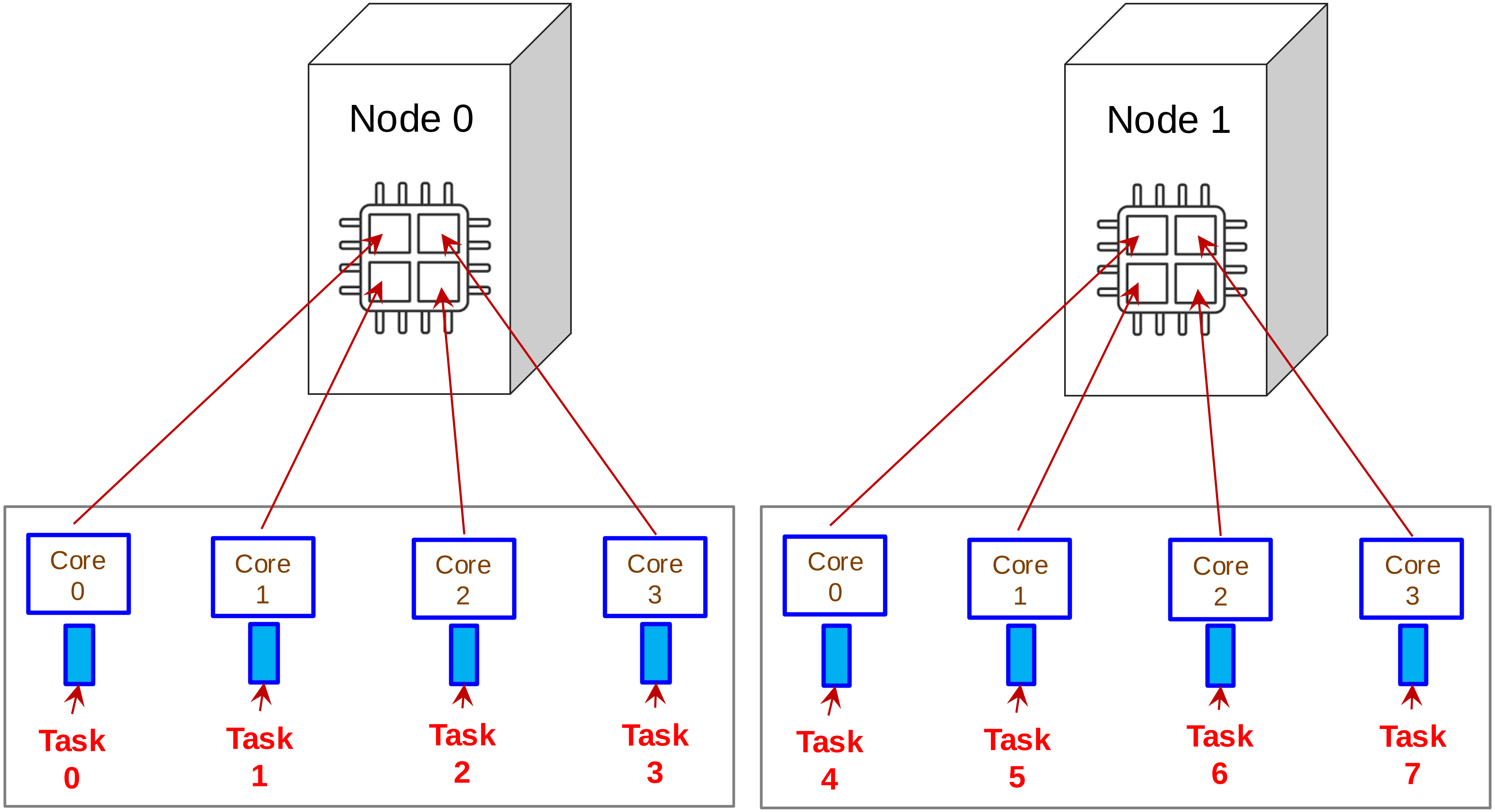
- **Parallelize**
  - This refers to the process of dividing a workload into multiple smaller tasks and executing them simultaneously. It allows for better utilization of computational resources, such as multi-core CPUs or multiple nodes in a cluster.
- **Massive**
  - Indicates the scale or volume of tasks being handled. It often refers to workloads that consist of hundreds, thousands, or even millions of tasks, which would take a long time to process sequentially.
- **Individual (Independent)**
  - Each task operates independently of the others. This means there's no dependency between tasks, allowing them to be executed in parallel without waiting for one another.
- **Tasks**
  - These are the units of work or commands you want to execute. Each task could be a program, script, or function that processes a specific set of inputs.

What do we want to accomplish?

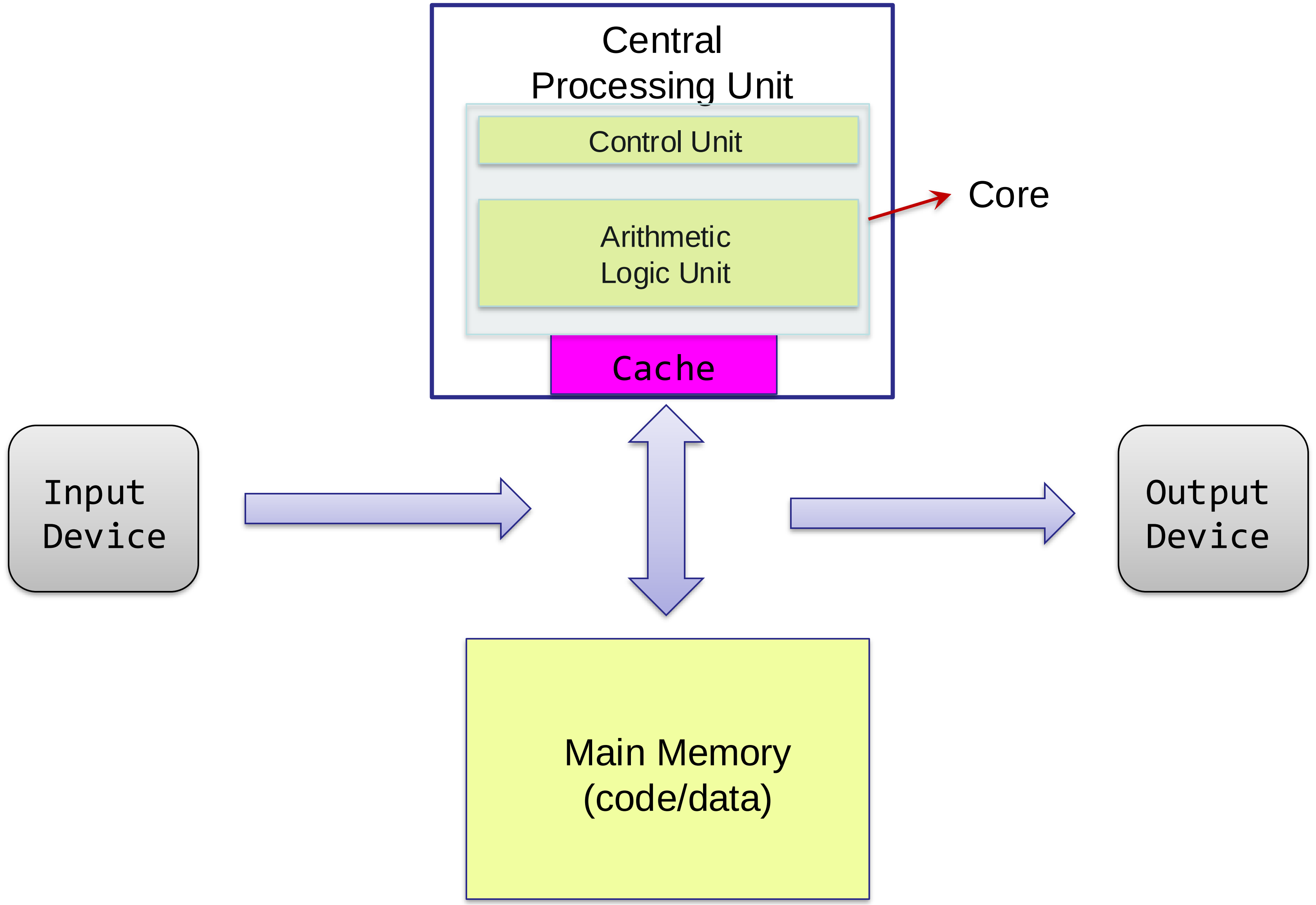
➢ **Parallize lots of small independent tasks on a multi-core platform (compute nodes)**



Background and Distribute



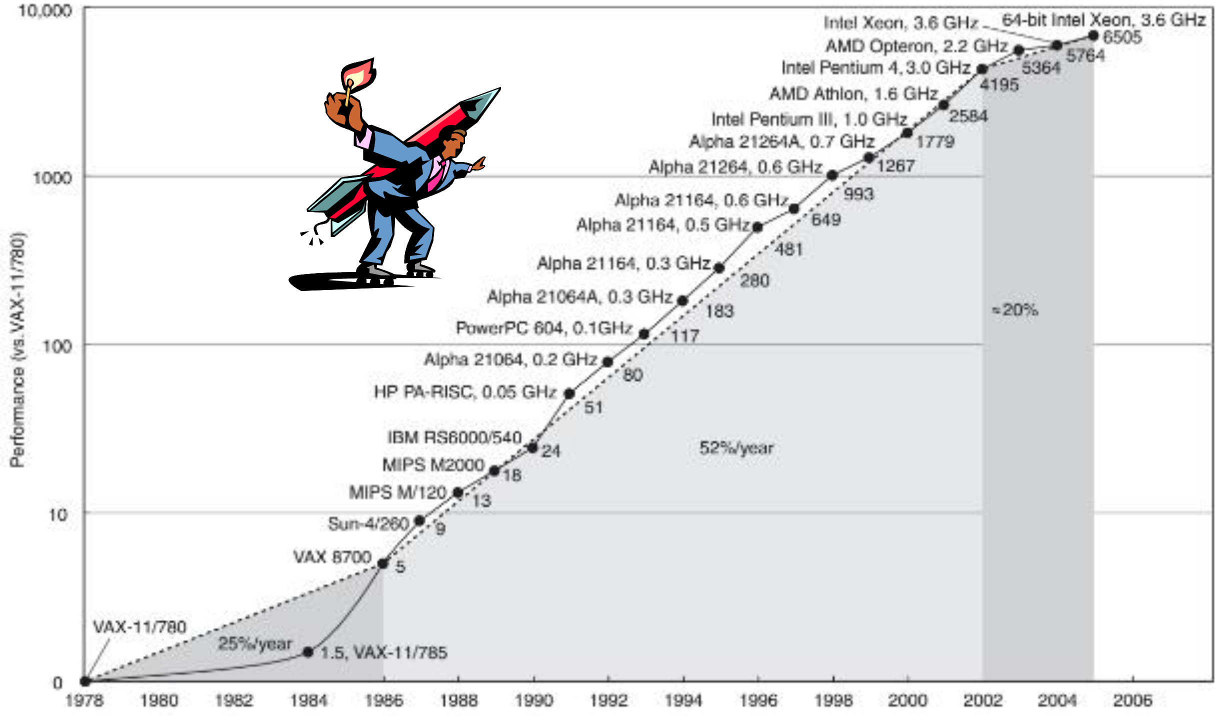
The von Neumann Architecture



Changing Times

- From 1986 - 2002, microprocessors were speeding like a rocket, increasing in performance an average of 50% per year.
- Since then, it's dropped to about 20% increase per year.

History of Processor Performance



Moore's Law states that transistor counts on chips double roughly every two years.

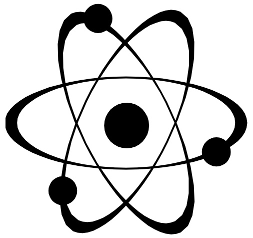
Limitation:

2 GHz Consumer
4 GHz Server

Source:
http://www.cs.columbia.edu/~sedwards/classes/2012/3827-spring/

A Little Physics Problem

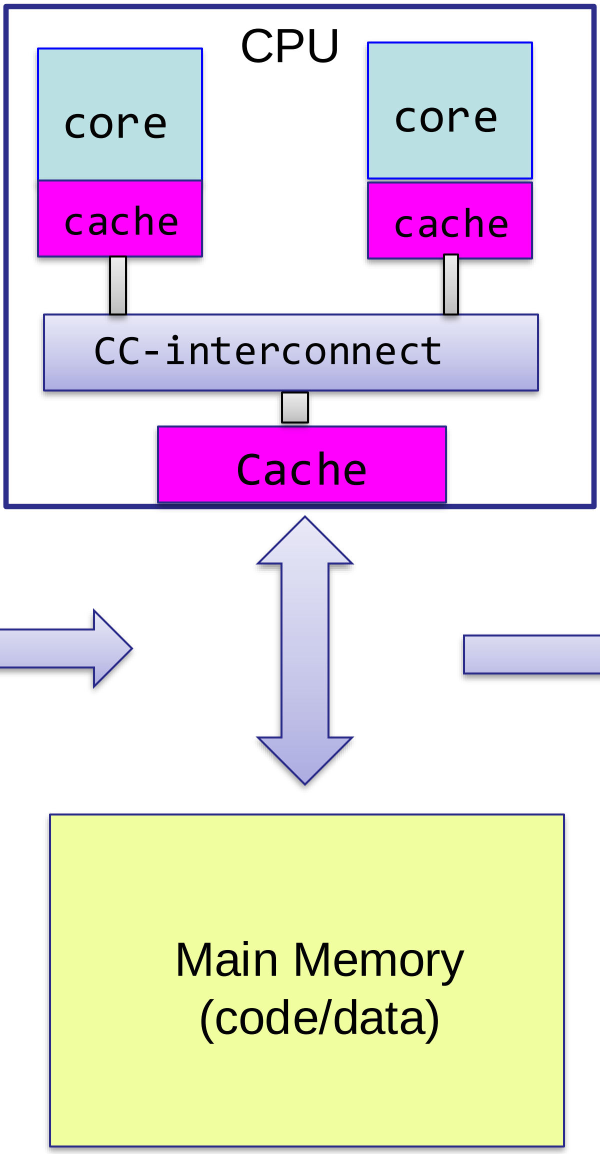
- **Smaller transistors = faster processors.**
- **Faster processors = increased power consumption.**
- **Increased power consumption = increased heat.**
- **Increased heat = unreliable processors.**

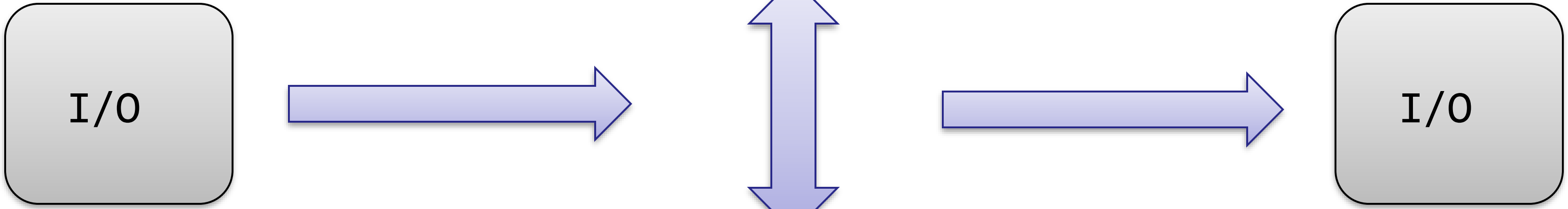


- **Solution:**
  - Move away from single-core systems to multicore processors.
  - “core” = central processing unit (CPU)
  - Introducing parallelism
    - *What if your problem is also not CPU dominant?*

The von Neumann Architecture

Around year 2005





GNU Parallel

- **GNU parallel is a shell tool for executing "embarrassingly parallel" tasks using one or more computers (*compute nodes*).**
- **Terminology:**
  - **Task**: Each ***small, independent*** piece of computation work to be finished, e.g., a single genome computation, a single MD simulation
  - **Job**: The list of tasks to be completed on a set of nodes (cores)
- **A task can be a *single command or a small script* that must be run for each of the lines in the input.**
- **The typical input is a *list of files*, a list of hosts, a list of users, a list of URLs, or a list of tables.**
- **See more at: https://www.gnu.org/software/parallel/**

GNU Parallel Syntax

Adding GNU Parallel to Environment

- **On SuperMike3:**

```
[fchen14@mike139 ~]$ module av parallel
---- /usr/local/packages/Modules/default/modulefiles/linux-rhel8-icelake ----
parallel-netcdf/1.12.2/intel-2021.5.0
parallel/20210922/intel-2021.5.0
parallel-netcdf/1.12.2/intel-2021.5.0-intel-mpi-2021.5.1
# load the module to environment
[fchen14@mike139 ~]$ module load parallel/20210922/intel-2021.5.0
[fchen14@mike139 ~]$ which parallel
/usr/local/packages/parallel/20210922/hrsviur/bin/parallel
[fchen14@mike139 ~]$ parallel –version # check GNU parallel version
GNU parallel 20210922
Copyright (C) 2007-2021 Ole Tange, http://ole.tange.dk and Free Software
Foundation, Inc.
...
```

- **Or add the below line to** `~/.modules`**:**

```
    module load parallel/20210922/intel-2021.5.0
```

GNU Parallel Syntax

- **Reading commands to be run in parallel from an input file:**

```
parallel [OPTIONS] < CMDFILE
```

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

```
parallel [OPTIONS] COMMAND [ARGUMENTS] ::: ARGLIST
```

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

```
parallel [OPTIONS] COMMAND [ARGUMENTS] :::: ARGFILE
```

ARGLIST from command line

- **`parallel [OPTIONS] COMMAND [ARGUMENTS] ::: ARGLIST`**
- **Examples:**

```
[fchen14@mike139 ~]$ parallel echo ::: A B C
A
B
C
[fchen14@mike139 ~]$ parallel echo ::: `seq 1 3`
1
2
3
[fchen14@mike139 ~]$ parallel echo ::: {A..Z}
A
B
...
Z
[fchen14@mike139 test]$ ls -1 | parallel echo
2013-06-18.tgz
backups.sh
bigmem_test.pbs
...
```

ARGLIST from file

- **parallel [OPTIONS] COMMAND [ARGUMENTS] :::: ARGFILE**

```
[fchen14@mike139 GNU_PARALLEL]$ pwd
/project/fchen14/GNU_PARALLEL
[fchen14@mike139 GNU_PARALLEL]$ cat input.lst | head
01.lj
02.lj
03.lj
...
[fchen14@mike139 GNU_PARALLEL]$ head input.lst -n 5 | parallel echo
01.lj
02.lj
03.lj
04.lj
05.lj
[fchen14@mike139 GNU_PARALLEL]$ parallel echo :::: input.lst
01.lj
02.lj
...
```

Replacement Strings

- '{}' returns a full line read from the input source.

```
[fchen14@mike139 GNU_PARALLEL]$ parallel echo {} ::: data/in.lj
data/in.lj
```

- '{/}' removes everything up to and including the last forward slash:

```
[fchen14@mike139 GNU_PARALLEL]$ parallel echo {/} ::: data/in.lj
in.lj
```

- '{//}' returns the directory name of input line.

```
[fchen14@mike139 GNU_PARALLEL]$ parallel echo {//} ::: data/in.lj
data
```

- '{.}' removes any filename extension:

```
[fchen14@mike139 GNU_PARALLEL]$ parallel echo {.} ::: data/in.lj
data/in
```

- '{/.}' returns the basename of the input line without extension. It is a combination of `{/}` and `{.}`:

```
[fchen14@mike139 GNU_PARALLEL]$ parallel echo {/.} ::: data/in.lj
in
```

- See "`man parallel`" for more detailed explanation.

Replacement String Example

- **Print the full path of the input file, and then print the desired output file name, e.g.:**
  - Input file: `data/lj.in`
  - Output file name: `output/lj.out`

```
# Process data/lj.in and send result to output/lj.out
$ parallel echo {} output/{/.}.out ::: data/lj.in
data/lj.in output/lj.out
```

Parallelize Job Script

- **GNU parallel is often called as this:**

```
cat input_file | parallel command
parallel command ::: foo bar
```

- **If command is a script, parallel can be combined into a single file so this will run the script in parallel:**

```
parallel [OPTIONS] script [ARGUMENTS] ::: ARGLIST
```

  – or

```
parallel [OPTIONS] script [ARGUMENTS] :::: ARGFILE
```

- **See next slide for example...**

Parallize Script Example

- This is the script we want to parallize "cmd_ex.sh":

```
#!/bin/bash
# print the input, on which host, which working directory
echo "This script uses input: $1 on $HOSTNAME:$PWD"
```

- Parallize the script using ARGLIST from command line:

```
[fchen14@mike139 GNU_PARALLEL]$ parallel --wd $PWD ./cmd_ex.sh ::: A B C
This script uses input: A on mike139:/project/fchen14/GNU_PARALLEL
This script uses input: B on mike139:/project/fchen14/GNU_PARALLEL
This script uses input: C on mike139:/project/fchen14/GNU_PARALLEL
```

- Parallize the script using ARGFILE:

```
[fchen14@mike139 GNU_PARALLEL]$ cat argfile
A
B
C
[fchen14@mike139 GNU_PARALLEL]$ parallel --wd $PWD ./cmd_ex.sh :::: argfile
This script uses input: A on mike139:/project/fchen14/GNU_PARALLEL
This script uses input: B on mike139:/project/fchen14/GNU_PARALLEL
This script uses input: C on mike139:/project/fchen14/GNU_PARALLEL
```

- Can parallize Python/Perl scripts, see "`man parallel`" for details

Common OPTIONS `--jobs (-j)`

- **`--jobs N (-j N)`**
  - Number of jobslots on each machine (`node`). Run up to N jobs in parallel. 0 means as many as possible. Default is 100% which will run one job per CPU core on each machine.
  - On HPC/LONI clusters, **N** is number of jobslots per node.
  - Make sure you use GNU Parallel version **`>=20161022`** to avoid a "`Max jobs to run`" bug

```
[fchen14@mike139 test]$ parallel --version
GNU parallel 20210922
...
```

- **`-j +N`**
  - Add N to the number of CPU cores. Run this many jobs in parallel.
- **`-j -N`**
  - Subtract N from the number of CPU cores. Run this many jobs in parallel. If the evaluated number is less than 1 then 1 will be used.

Common OPTIONS --slf (PBS)

- **--slf filename (--sshloginfile filename)**
  - File with sshlogins. The file consists of sshlogins on separate lines. Empty lines and lines starting with '#' are ignored.
  - Look at "man parallel" for more detailed explanation.
  - A typical example on PBS clusters while running batch jobs:

    ```
    --slf $PBS_NODEFILE
    ```
  - Recall what is inside $PBS_NODEFILE?

    ```
    [fchen14@smic139 laplace]$ cat $PBS_NODEFILE # assume we use SuperMIC
    smic139
    smic139
    ...
    smic139
    smic429
    smic429
    ...
    smic429
    ```

    20 repeats (cores) on smic139

    20 repeats (cores) on smic429

Common OPTIONS --slf (Slurm)

- **`--slf filename (--sshloginfile filename)`**
  - To get the sshloginfile on Slurm job session, use the below command:

```
scontrol show hostname $SLURM_NODELIST > nodefile
```

  - A typical example on HPC/LONI clusters while running batch jobs:

```
--slf nodefile
```

  - Look at `$SLURM_NODELIST` and `nodefile`

```
# start an interactive job requesting 2 nodes (64x2=128 cores)
[kasetti@mike4 ~]$ srun -N2 -n128 -p workq --cpu-bind none --pty bash
srun: Job is in held state, pending scheduler release
srun: job 39474 queued and waiting for resources
Interactive job 39474 waiting:
srun: job 39474 has been allocated resources # we got mike157 and mike158
[kasetti@mike157 ~]$ echo $SLURM_NODELIST
mike[157-158]
[kasetti@mike157 ~]$ scontrol show hostname $SLURM_NODELIST > nodefile
[kasetti@mike157 ~]$ cat nodefile
mike157
mike158
```

Common OPTIONS `--sshdelay`

- **If many tasks are started on the same compute node, sshd can be overloaded. On SuperMike3/QB3, some of the tasks might fail to start, e.g., starting all 64/48 tasks at the same time.**
- **GNU parallel can insert a delay between each task run on the same server:**

```
[fchen14@mike139 GNU_PARALLEL]$ parallel --sshdelay 0.1 echo ::: A B C
A
B
C
```

Common OPTIONS `--wd`

- **`--wd mydir (--workdir mydir)`**
  - Designate the working directory of your commands.
  - A typical value can be:
    - `$PBS_O_WORDIR (PBS)`
    - `$SLURM_SUMBIT_DIR (Slurm)`

Common OPTIONS `--env`

- **`--env ENV_VAR`**
  - --env to tell GNU parallel to transfer an environment variable to the remote system.
  - A typical usage:

```
export OMP_NUM_THREADS=5
parallel --env OMP_NUM_THREADS cmd ::: ARGLIST
```

Common OPTIONS `--progress`

- **`--progress`**
  - Show progress of computations. (**Not recommended for batch jobs**)
  - List the computers involved in the task with number of CPU cores detected and the max number of jobs to run.
  - After that show progress for each node: number of running jobs, number of completed jobs, and percentage of all jobs done by this computer.
  - Example:

```
[fchen14@mike139 ~]$ parallel --progress echo ::: A B C

Computers / CPU cores / Max jobs to run
1:local / 64 / 3

Computer:jobs running/jobs completed/%of started jobs/Average seconds to complete
local:3/0/100%/0.0s A
local:2/1/100%/1.0s B
local:1/2/100%/0.5s C
local:0/3/100%/0.3s
```

- **See also `--bar`**

Common OPTIONS `--joblog`

- **`--joblog logfile`**
  - Creates a record for each completed subjob (task) to be written to LOGFILE, with info on how long they took, their exit status, etc.
  - Can be used to identify failed jobs, e.g.:

```
[fchen14@mike139 misc]$ parallel --joblog logfile exit ::: 1 2 0 0
[fchen14@mike139 misc]$ cat logfile
Seq     Host    Starttime       JobRuntime      Send    Receive Exitval Signal  Command
1       :       1477514132.358       0.019      0       0       1       0       exit 1
2       :       1477514132.375       0.003      0       0       2       0       exit 2
3       :       1477514132.376       0.002      0       0       0       0       exit 0
4       :       1477514132.377       0.003      0       0       0       0       exit 0
```

Common OPTIONS `--timeout`

- **`--timeout secs`**
  - Time out for command. If the command runs for longer than secs (seconds) it will get killed.
  - If secs is followed by a % then the timeout will dynamically be computed as a percentage of the median average runtime. Only values > 100% will make sense.

❖ Useful if you know the command has failed if it runs longer than a threshold.

Serial Jobs Example

LAMMPS Introduction

- **LAMMPS is a classical molecular dynamics code with a focus on materials modeling. https://www.lammps.org/**
- **You don’t need any background in molecular dynamics/LAMMPS to understand today’s example.**
- **Typical LAMMPS Syntax**
  - Serial run

    ```
    lmp_serial -in in.script
    ```
  - Multi-Threaded run

    ```
    env OMP_NUM_THREADS=4 lmp_omp -sf omp -in in.script # use OMP_NUM_THREADS
    lmp_omp -sf omp -pk omp 5 -in in.script # use -pk omp to specify threads
    ```
  - MPI run

    ```
    srun --overlap -n 4 lmp_mpi -in in.script # Slurm version, --overlap only
    needed for interactive job
    mpirun -n 4 lmp_mpi -in in.script # PBS version
    ```
  - Custom command:

    ```
    lmp_serial -var nsteps 200 –in in.script # we defined a custom variable in
    the input file to let nsteps control total steps to run,
    The above command will run our input for 200 steps
    ```

LAMMPS Input File Used Today

```
# 3d Lennard-Jones melt
variable  x index 1
variable  y index 1
variable  z index 1
variable  xx equal 80*$x
variable  yy equal 80*$y
variable  zz equal 80*$z
units                  lj
atom_style             atomic
lattice                fcc 0.8442
region                 box block 0 ${xx} 0 ${yy} 0 ${zz}
create_box             1 box
create_atoms           1 box
mass                   1 1.0
velocity  all create 1.44 87287 loop geom
pair_style             lj/cut 2.5
pair_coeff             1 1 1.0 1.0 2.5
neighbor  0.3 bin
neigh_modify           delay 0 every 20 check no
fix                    1 all nve
# ${nsteps} is the parameter passing from LAMMPS command line "-var nsteps 200"
run                    ${nsteps}
```

Distribute Serial Jobs LAMMPS (Slurm)

```
#!/bin/bash
#SBATCH -N 2                           # request two nodes
#SBATCH -n 128                         # specify 128 process
#SBATCH -t 2:00:00
#SBATCH -p checkpt
#SBATCH -A hpc_hpcadmin8
#SBATCH -o gp-serial.out

TASKS_PER_NODE=16
SECONDS=0
scontrol show hostname $SLURM_NODELIST > nodefile
parallel --joblog lmp.serial.log \
         -j $TASKS_PER_NODE \
         --slf nodefile \
         --workdir $SLURM_SUBMIT_DIR \
         --sshdelay 0.1 \
         `which lmp_serial` -in {} -var nsteps 200 :::: input.lst
echo "took $SECONDS sec"
```

```
[fchen14@mike2 GNU_PARALLEL]$ head input.lst
data/01.lj.in
data/02.lj.in
data/03.lj.in
data/04.lj.in
data/05.lj.in
...
```

In case the task command is too long (complex)

- Use a script for each task to be distributed, example here (`call_lmp.sh`)
- GNU Parallel will distribute each task script

```
TASKS_PER_NODE=16

scontrol show hostname $SLURM_NODELIST > nodefile
parallel --joblog lmp.serial.log \
         -j $TASKS_PER_NODE \
         --slf nodefile \
         --workdir $SLURM_SUBMIT_DIR \
         --sshdelay 0.1 \
         ./call_lmp.sh {} 200 :::: input.lst
```

```
# content of call_lmp.sh
#!/bin/bash
echo "\$1=$1,\$2=$2"
lmp_serial -in $1 -var nsteps $2
```

$2 is the input parameter `200`

$1 is the input from `input.lst`, e.g. `data/01.lj.in`

Distribute Serial Jobs LAMMPS (PBS)

```
#!/bin/bash
#PBS -l nodes=2:ppn=20
#PBS -l walltime=1:00:00
#PBS -q checkpt
#PBS -A hpc_hpcadmin8
#PBS -j oe
#PBS -o gp-serial-pbs.out

module load parallel
module load lammps
TASKS_PER_NODE=20
SECONDS=0

cd $PBS_O_WORKDIR
parallel --joblog lmp.serial.log \
         -j $TASKS_PER_NODE \
         --slf $PBS_NODEFILE \
         --workdir $PBS_O_WORKDIR \
         --sshdelay 0.1 \
         `which lmp_mpi` -in {} -var nsteps 200 :::: input.lst
echo "took $SECONDS sec"
```

```
[fchen14@smic1 GNU_PARALLEL]$ head input.lst
data/01.lj.in
data/02.lj.in
data/03.lj.in
data/04.lj.in
data/05.lj.in
data/06.lj.in
...
```

Multi-Threaded Example

Distribute Multi-Threaded Jobs

- **Distribute Multi-Threaded jobs is very similar to the pure serial job example, the only difference is `TASKS_PER_NODE`:**
  - `TASKS_PER_NODE=CPU_CORES_PER_NODE / NUM_THREADS_PER_TASK`
- **If each job uses 4 threads, each node on SuperMike3 has 64 cores, then**
  - `TASKS_PER_NODE=64/4=16`
- **Slurm script (`#SBATCH comments` omitted):**

```
TASKS_PER_NODE=16
export OMP_NUM_THREADS=4
SECONDS=0
scontrol show hostname $SLURM_NODELIST > nodefile
parallel -j $TASKS_PER_NODE \
         --slf nodefile \
         --workdir $WDIR \
         --sshdelay 0.1 \
         --env OMP_NUM_THREADS \
         `which lmp_omp` -sf omp -in {} -var nsteps 200 :::: input.lst
echo "took $SECONDS sec"
```

Put 64/4=16 tasks per node

Pass the environmental variable OMP_NUM_THREADS to each task

OpenMP switch in lammps

Multi-Threaded LAMMPS (Slurm)

```
#!/bin/bash
#SBATCH -N 2                         # request two nodes
#SBATCH -n 128                       # specify 128 process
#SBATCH -t 2:00:00
#SBATCH -p checkpt
#SBATCH -A hpc_hpcadmin8
#SBATCH -o gp-omp.out

TASKS_PER_NODE=16
export OMP_NUM_THREADS=4
SECONDS=0
scontrol show hostname $SLURM_NODELIST > nodefile

parallel --joblog lmp.omp.log \
         -j $TASKS_PER_NODE \
         --slf nodefile \
         --workdir $SLURM_SUBMIT_DIR \
         --sshdelay 0.1 \
         --env OMP_NUM_THREADS \
         `which lmp_omp` -sf omp -in {} -var nsteps 200 :::: input.lst
echo "took $SECONDS sec"
```

This script is on SuperMike3, 64 cores per node, so TASKS_PER_NODE=64/4=16

Use 4 OMP threads per task

OpenMP switch in lammps

Multi-Threaded LAMMPS (PBS)

```
#!/bin/bash
#PBS -l nodes=2:ppn=20
#PBS -l walltime=1:00:00
#PBS -q checkpt
#PBS -A hpc_hpcadmin8
#PBS -j oe
#PBS -o gp-omp-pbs.out

module purge
module load parallel
module load lammps
TASKS_PER_NODE=4
SECONDS=0
cd $PBS_O_WORKDIR
parallel --joblog lmp.omp.pbs.log \
         -j $TASKS_PER_NODE \
         --slf $PBS_NODEFILE \
         --workdir $PBS_O_WORKDIR \
         --sshdelay 0.1 \
         `which lmp` -sf omp -pk omp 5 -in {} -var nsteps 200 :::: input.lst
echo "took $SECONDS sec"
```

This script is on SuperMIC, 20 cores per node, so TASKS_PER_NODE=20/5=4

Use 5 OpenMP threads per task

Multi-Process (MPI) Example

Distribute MPI Jobs - LAMMPS

- **This section describes how to distribute small MPI jobs.**
- **Example problem - LAMMPS MPI**
  - Using the same input file, but with multiple MPI process for each task.
  - For simplicity, each MPI process will use only one thread

Distributing MPI Jobs (Slurm)

```
#!/bin/bash
#SBATCH -N 2          # request two nodes
#SBATCH -n 128       # specify 128 process
#SBATCH -t 2:00:00
#SBATCH -p checkpt
#SBATCH -A hpc_hpcadmin8
#SBATCH -o gp-mpi.out

TASKS_PER_NODE=16
PROC_PER_TASK=4
SECONDS=0
scontrol show hostname $SLURM_NODELIST > nodefile

parallel --joblog lmp.mpi.log \
         -j $TASKS_PER_NODE \
         --slf nodefile \
         --workdir $SLURM_SUBMIT_DIR \
         --sshdelay 0.1 \
         srun --overlap -n $PROC_PER_TASK `which lmp` -in {} -var nsteps 200 ::::
input.lst
echo "took $SECONDS sec"
```

```
[fchen14@mike4 GNU_PARALLEL]$ head input.lst
data/01.lj.in
data/02.lj.in
data/03.lj.in
data/04.lj.in
data/05.lj.in
...
```

This script is on SuperMike3, 64 cores per node, so TASKS_PER_NODE=64/4=16

Use 4 MPI processes per task

Distributing MPI Jobs (PBS)

```
#!/bin/bash
#PBS -l nodes=2:ppn=20
#PBS -l walltime=1:00:00
#PBS -q checkpt
#PBS -A hpc_hpcadmin8
#PBS -j oe
#PBS -o gp-omp-pbs.out

module purge
module load parallel
module load lammps
TASKS_PER_NODE=4
SECONDS=0
cd $PBS_O_WORKDIR
parallel --joblog lmp.mpi.pbs.log \
         -j $TASKS_PER_NODE \
         --slf $PBS_NODEFILE \
         --workdir $PBS_O_WORKDIR \
         --sshdelay 0.1 \
         mpirun -np 5 `which lmp` -in {} -var nsteps 200 :::: input.lst
echo "took $SECONDS sec"
```

```
[fchen14@mike4 GNU_PARALLEL]$ head input.lst
data/01.lj.in
data/02.lj.in
data/03.lj.in
data/04.lj.in
data/05.lj.in
...
```

This script is on SuperMIC, 20 cores per node, so TASKS_PER_NODE=20/5=4

Use 5 MPI processes per task

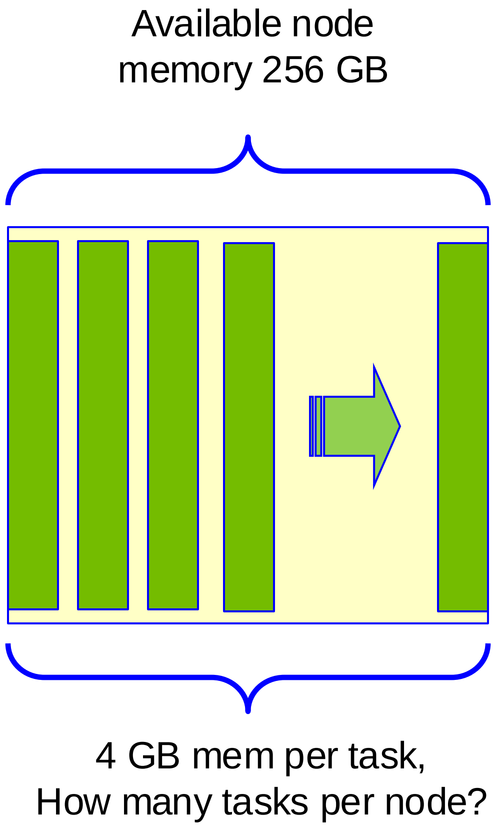
Proper Usage of GNU Parallel

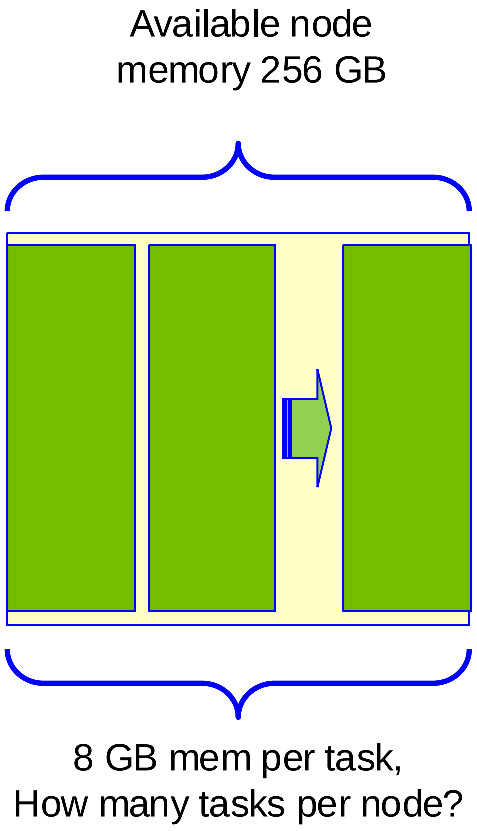
Common Rules

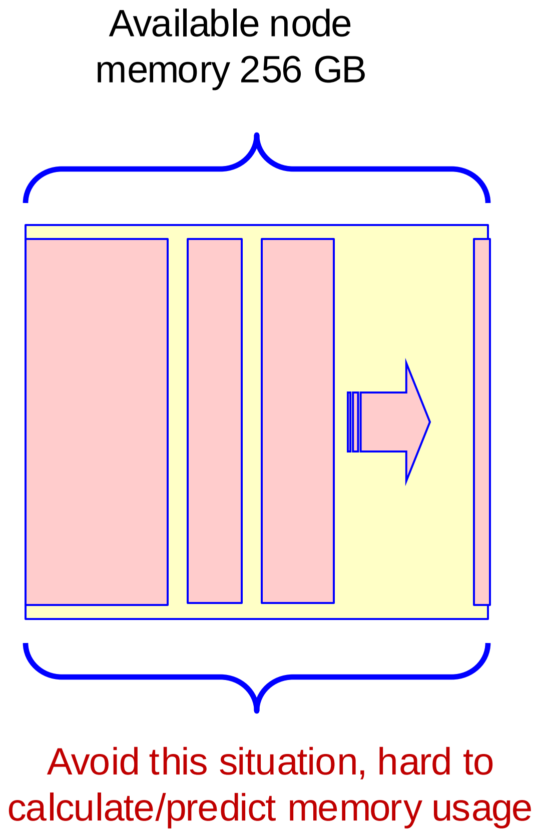
- **Don't use more than one node when you are debugging/testing your code**
- **Know the performance of single task**
- **Start with only a few tasks in your `input.lst`**
- **After you are comfortable with one node job, start with two node job first before jumping to more than three nodes**
- **Typically use no more than 5 nodes.**

Memory Consideration

- **Relationship between node memory and cores**
  - Rule of Thumb: cannot exceed the available memory on a node

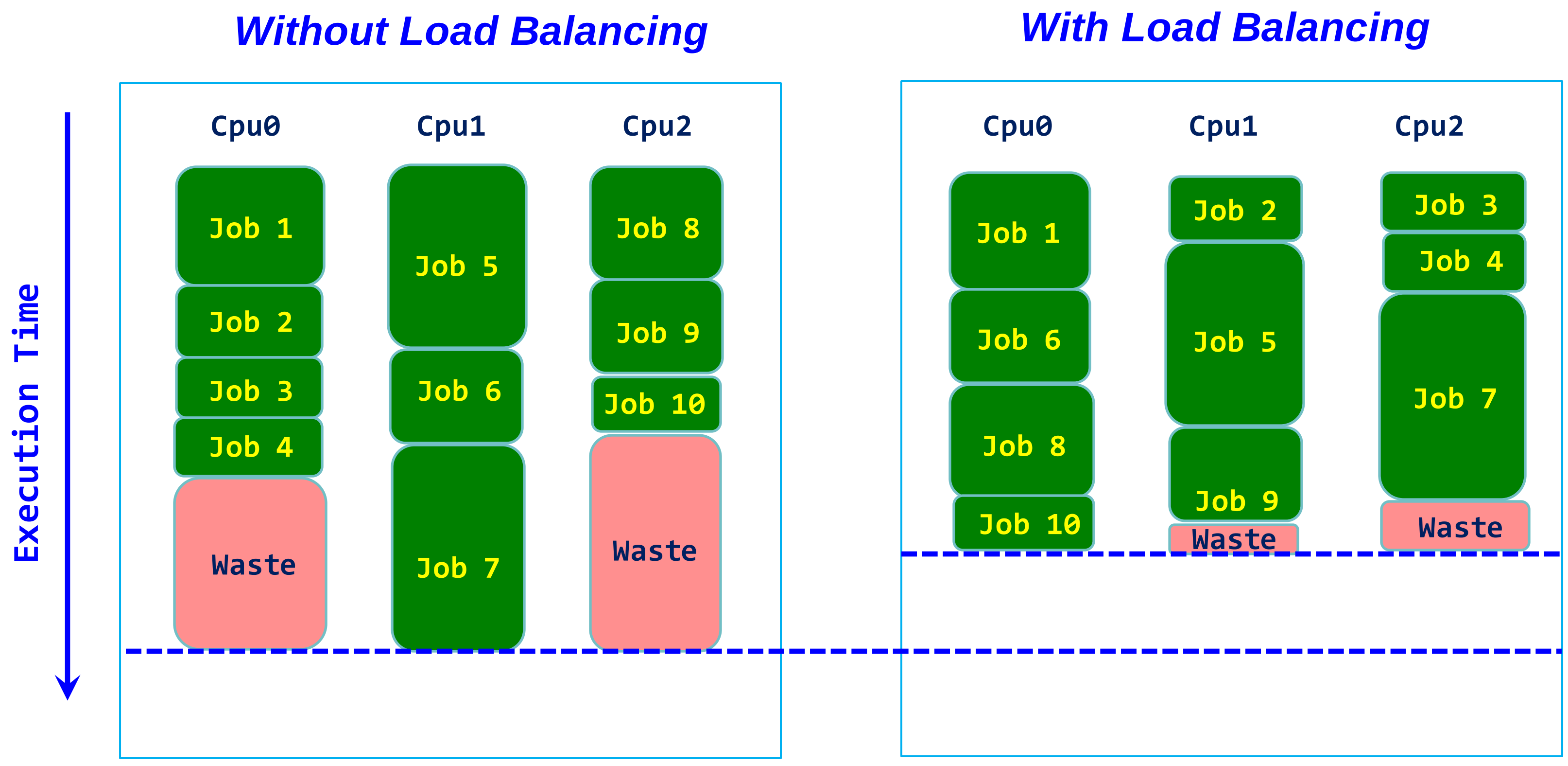





Available node memory 256 GB
Avoid this situation, hard to calculate/predict memory usage

Load Balancing in GNU Parallel

- **GNU Parallel spawns the next job when one finishes - keeping the CPUs active and thus saving time.**

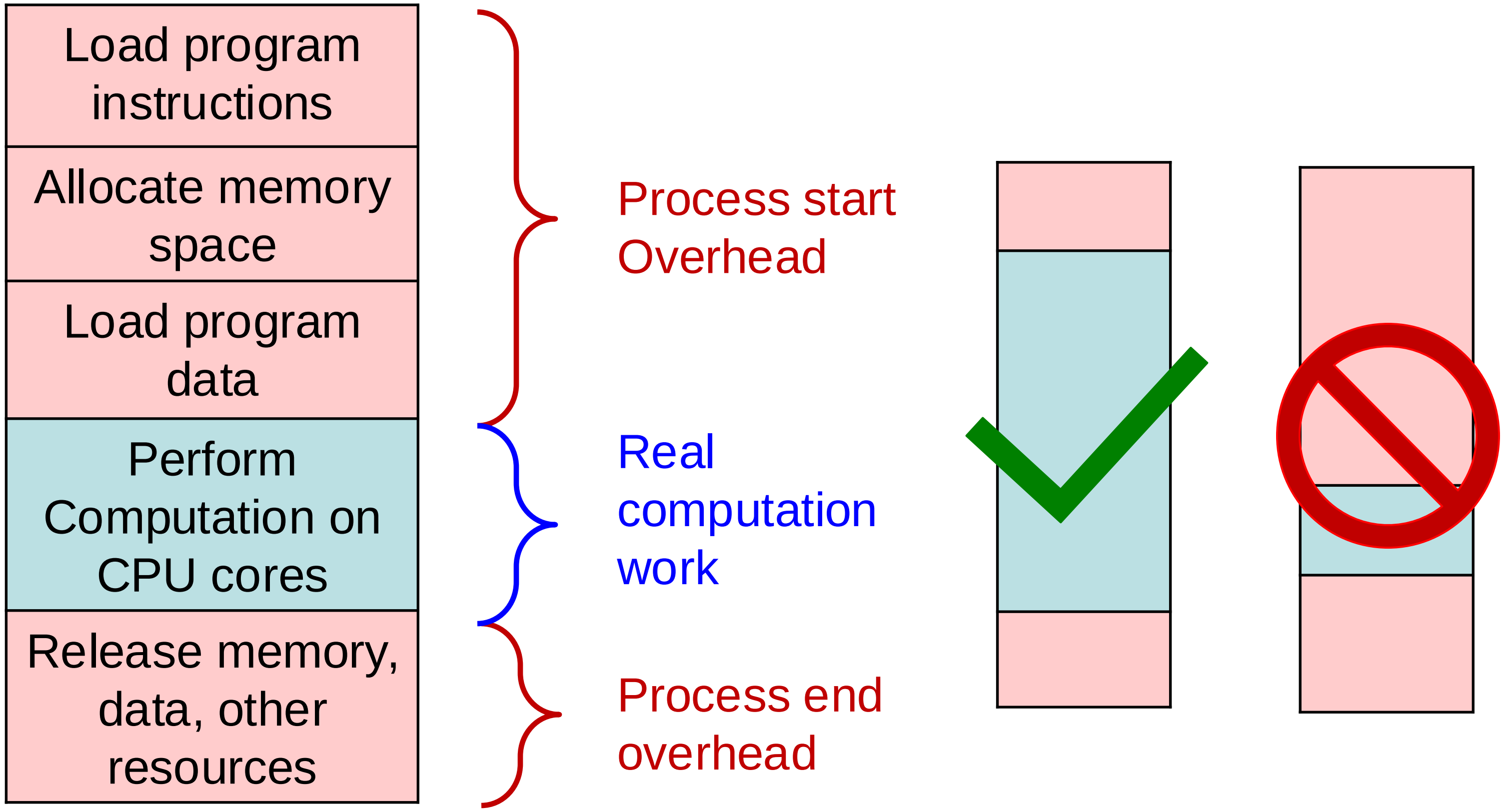


Task Granularity

- **In parallel computing, granularity (or grain size) of a task is a measure of the amount of work (or computation) which is performed by that task.**
- **Impact of granularity on performance**
  - Using fine grains or small tasks results in more parallelism and hence increases the speedup. However, synchronization overhead, scheduling strategies etc. can negatively impact the performance of fine-grained tasks.
  - Simply increasing parallelism alone cannot give the best performance.
  - In order to reduce the communication overhead, granularity can be increased. Coarse grained tasks have less communication overhead but they often cause load imbalance. Hence optimal performance is achieved between the two extremes of fine-grained and coarse-grained parallelism.

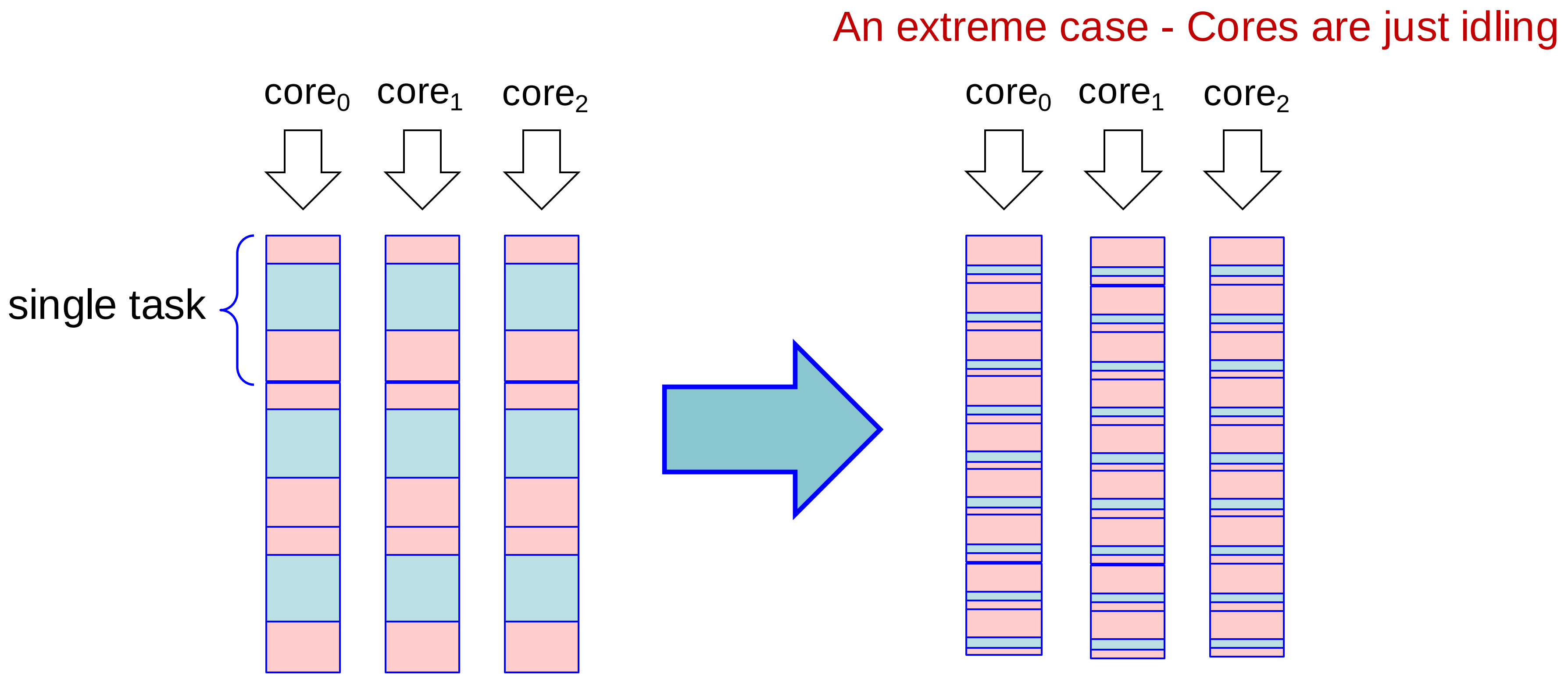
Ref: https://en.wikipedia.org/wiki/Granularity_(parallel_computing)

Components of a Task (Process)



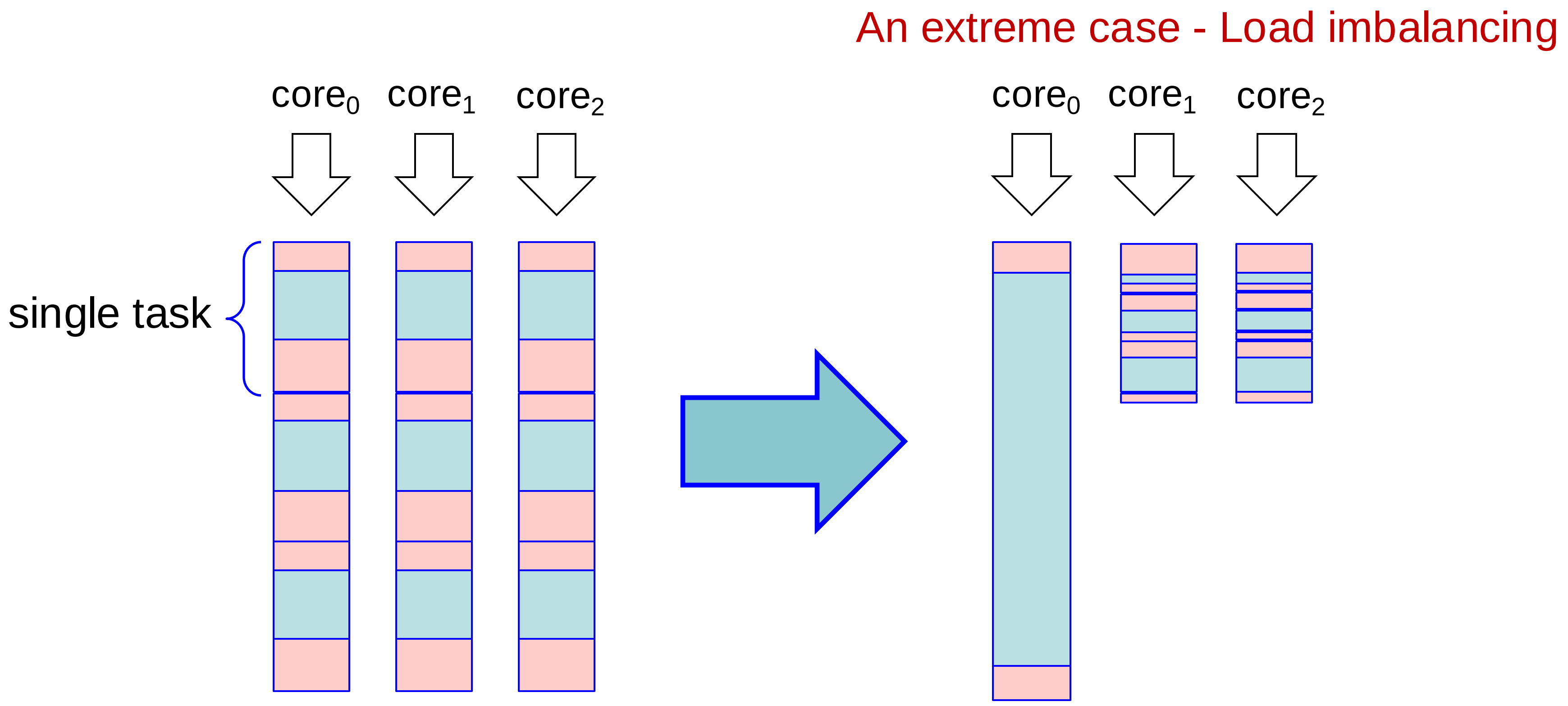
Typical Misuse - Tiny grain size case

- **Tiny grain size**
  - E.g., each task takes little time (e.g., less than a second)
  - Most time will be spent on overhead



Typical Misuse - Large grain size case

- **Large grain size**
  - Some tasks are much longer than the rest
  - Load balancing can never be achieved



Summary

- Why need GNU Parallel?
- Basic syntax of GNU Parallel and examples
- Ho to use it wisely

- For more information about GNU Parallel, refer to:
  - https://www.gnu.org/software/parallel/parallel_tutorial.html