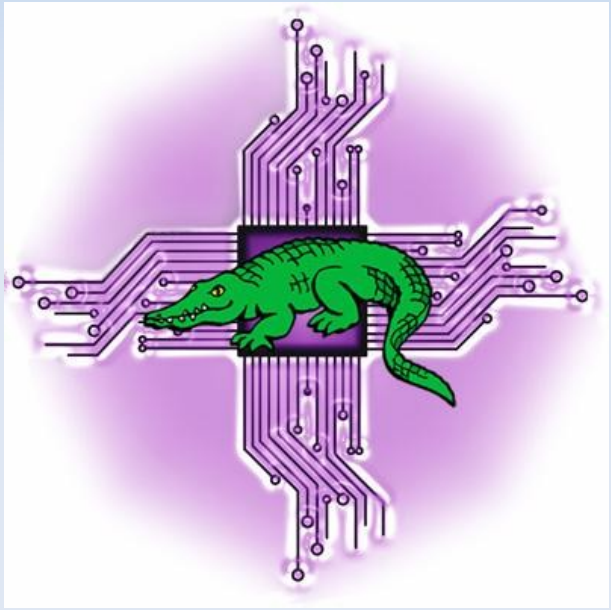
The Spartan HPC System at the University of Melbourne

COMP90024 Cluster and Cloud Computing



University of Melbourne, March 22, 2022

lev.lafayette@unimelb.edu.au

Outline of Lecture

"This is an advanced course but we get mixed bag: students that have 5+ years of MPI programming on supercomputers, to students that have only done Java on Windows."

- Some background on supercomputing, high performance computing, parallel computing, scientific computing (there is overlap, but they're not the same thing).
- An introduction to Spartan, University of Melbourne's HPC general purpose HPC system.
- Logging in, help, and environment modules.
- Job submission with Slurm workload manager; simple submissions, multicore, job arrays, job dependencies, interactive jobs.
- Parallel programming with shared memory and threads (OpenMP) and distributed memory and message passing (OpenMPI) and use of MPI4Py

Why Supercomputers?

Data size and complexity is growing much faster than the capacity of personal computational devices to process that data[1][2], primarily to the heat issues associated with increased clock-speed (Dennard scaling).

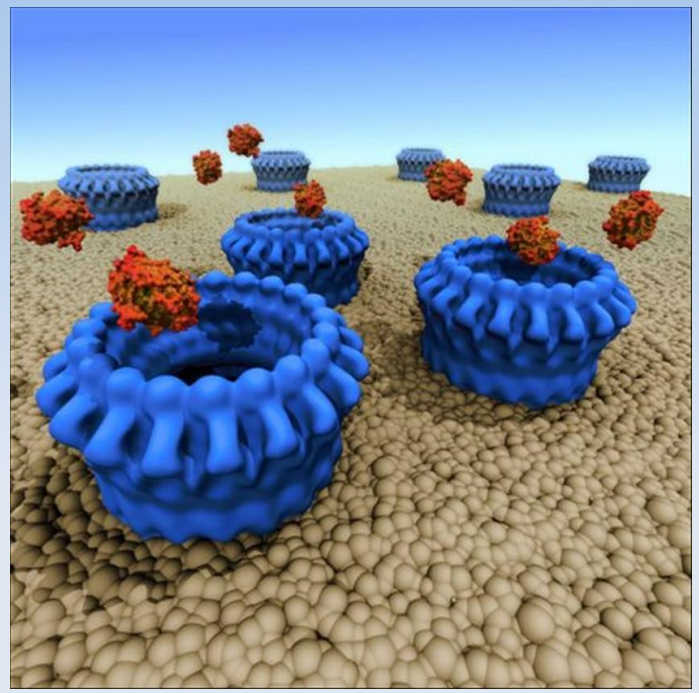
Various technologies to mitigate this problem; multi-processor systems, multi-core processors, shared and distributed memory parallel programming, general purpose graphics processing units, non-volatile memory (the PDP-11 gets the last laugh), RoCE/RDMA and network topologies, and so forth.

All of these are incorporated in various HPC systems and at scale. Increased research output[3], exceptional return on investment (44:1 profits or cost-savings [4]).

Supercomputers are critical for the calculations involved in climate and meteorological models, geophysics simulations, biomolecular behaviour, aeronautics and aerospace engineering, radio telescope data processing, particle physics, brain science, etc.

Some Local Examples

Researchers from Monash University, the Peter MacCallum Cancer Institute in Melbourne, the Birkbeck College in London, and VPAC in 2010 unravelled the structure the protein perforin to determine how pathogenic cells are attacked by white blood cells [5].



In 2015 researchers from VLSCI announced how natural antifreeze proteins bind to ice to prevent it growing which has important implications for extending donated organs and protecting crops from frost damage [6].

In 2016 CSIRO researchers successfully manipulated the behaviour of Metallic Organic Frameworks to control their structure and alignment which provides opportunities for real-time and implantable medical electric devices [7].

In 2019 a research team, including University of Melbourne on Spartan, broke the Zodiac cipher 360 which had eluded criminal and legal teams in the United States for over fifty years.

Supercomputers

'Supercomputer” arbitrary term. In general use it means any single computer system (itself a contested term) that has exceptional processing power for its time. One metric is the number of floating-point operations per second (FLOPS) such a system can carry out.

Supercomputers, like any other computing system, have improved significantly over time. The Top500 list is based on FLOPS using LINPACK - HPC Challenge is a broader, more interesting metric.

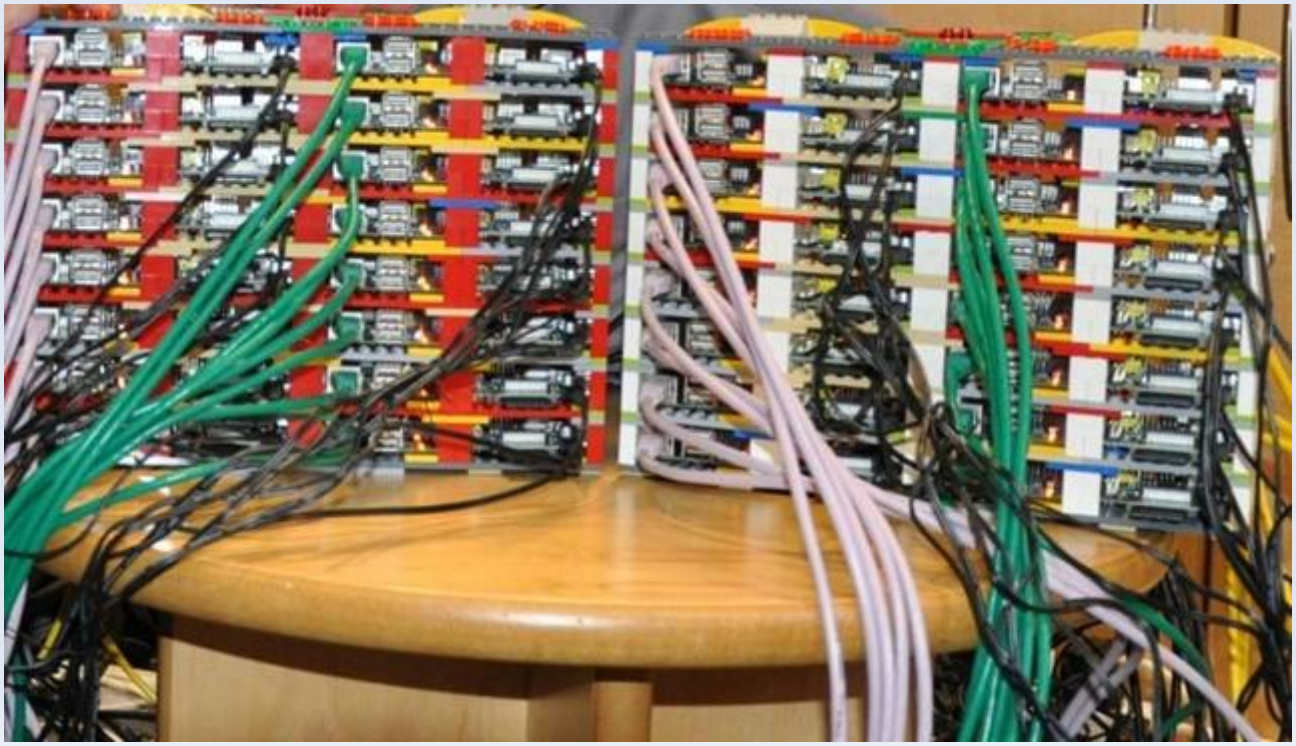
1994: 170.40 GFLOPS 1996: 368.20 GFLOPS 1997: 1.338 TFLOPS 1999: 2.3796 TFLOPS 2000: 7.226 TFLOPS 2004; 70.72 TFLOPS 2005: 280.6 TFLOPS 2007: 478.2 TFLOPS 2008: 1.105 PFLOP 2009: 1.759 PFLOPS
2010: 2.566 PFLOPS 2011: 10.51 PFLOPS
2012: 17.59 PFLOPS 2013: 33.86 PFLOPS
2014: 33.86 PFLOPS 2015: 33.86 PFLOPS
2016: 93.01 PFLOPS 2017: 93.01 PFLOPS
2018: 143.00 PFLOPS 2019: 148.60 PFLOPS
2020: 442.01 PFLOPS 2021: 442.01 PFLOPS
2022: 1.102 EFLOPS

The Exaflop benchmark was reached in 2022 by the Frontier system at Oak Ridge National Laboratory in the United States.

High Performance Computing

High-performance computing (HPC) is any computer system whose architecture allows for above average performance. A system that is one of the most powerful in the world, but is poorly designed, could be a "supercomputer".

Clustered computing is when two or more computers serve a single resource. This improves performance and provides redundancy; typically a collection of smaller computers strapped together with a high-speed local network (e.g., Myrinet, InfiniBand, 10 Gigabit Ethernet). Even a cluster of Raspberry Pi with Lego chassis (University of Southampton, 2012)!



Horse and cart as a computer system and the load as the computing tasks. Efficient arrangement, bigger horse and cart, or a teamster?

The clustered HPC is the most efficient, economical, and scalable method, and for that reason it dominates supercomputing.

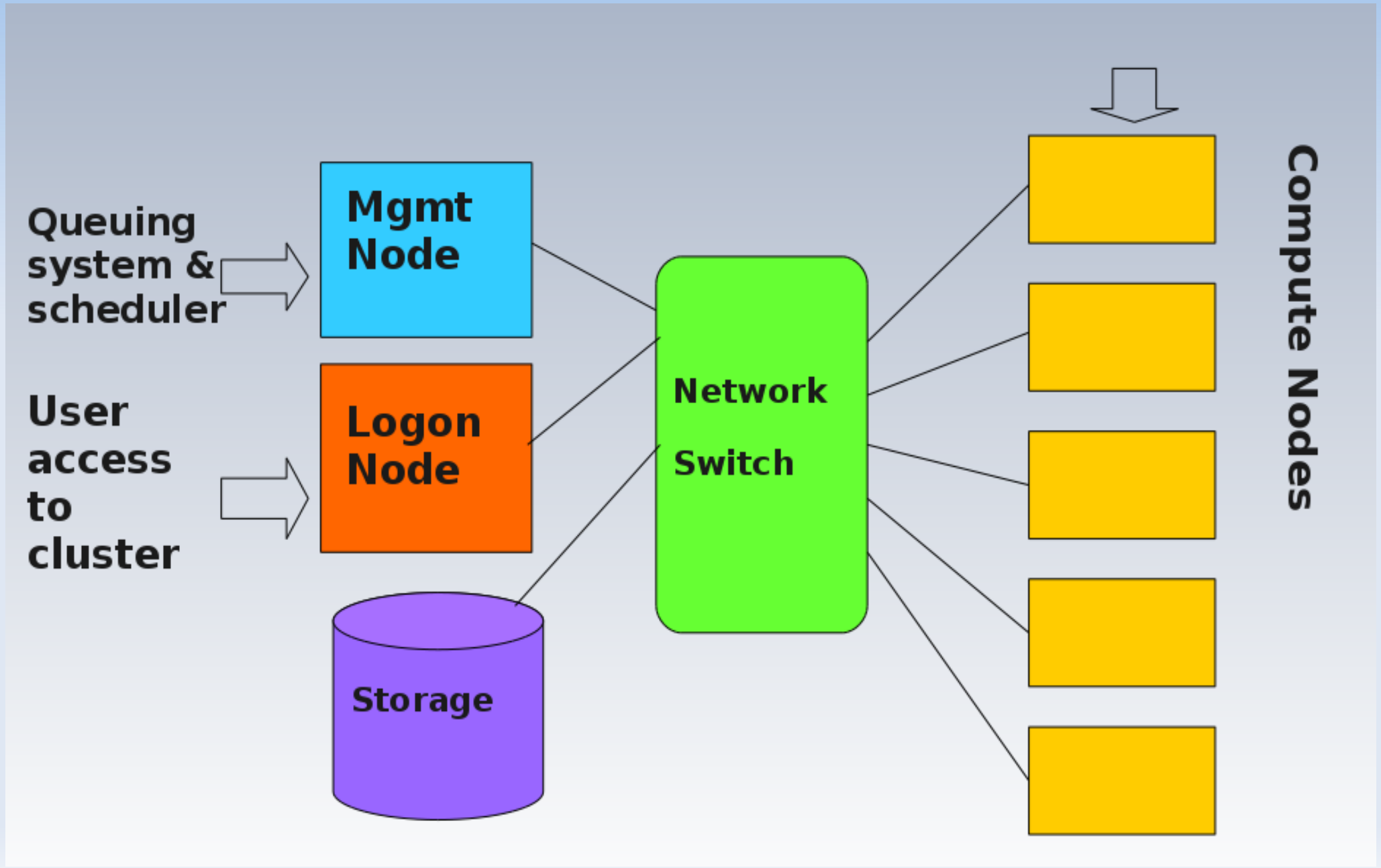
Parallel and Research Programming

With a cluster architecture, applications can be more easily parallelised across them. Parallel computing refers to the submission of jobs or processes over multiple processors and by splitting up the data or tasks between them (random number generation as data parallel, driving a vehicle as task parallel).

Research computing is the software applications used by a research community to aid research. This skills gap is a major problem and must be addressed because as the volume, velocity, and variety of datasets increases then researchers will need to be able to process this data.

Reproducibility in science is a ***huge*** issue! Many of the problems relate to inattentiveness to software versions, compilers and options, etc. all of which can be very site-specific in HPC facilities. See: The Ten Year Reproducibility Challenge (https://www.nature.com/articles/d41586-020-02462-7). Also, the limitations of number systems used in computing.

HPC Cluster Design

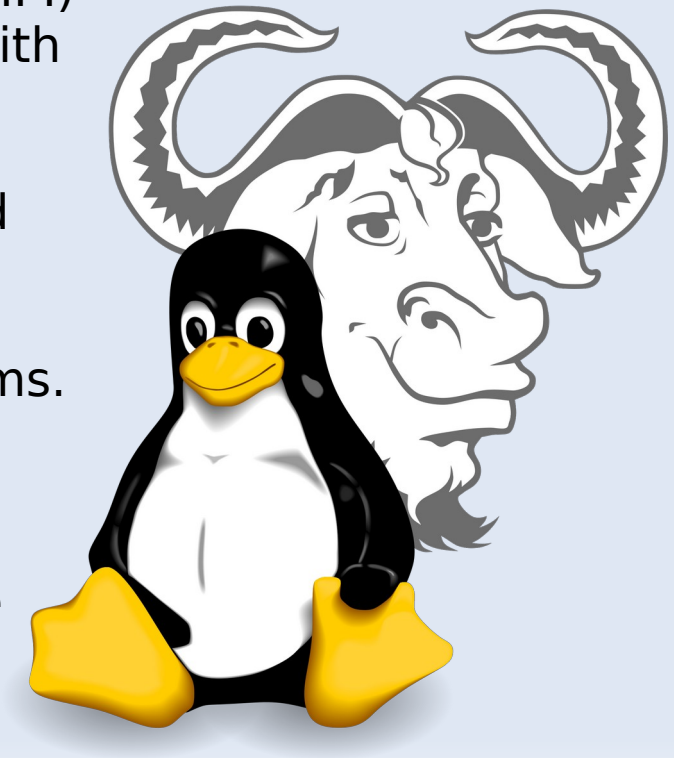


It's A GNU/Linux World

From November 2017 onwards of the Top 500 Supercomputers worldwide, *every single machine* used Linux.

The command-line interface provides a great deal more power and is very resource efficient.

GNU/Linux scales and does so with stability and efficiency. Critical software such as the Message Passing Interface (MPI) and nearly all scientific programs are designed to work with GNU/Linux.



The operating system and many applications are provided as "free and open source", which means that not only are there are some financial savings, were also much better placed to improve, optimize and maintain specific programs.

Free or open source software (not always the same thing) can be can be compiled from source for the specific hardware and operating system configuration, and can be optimised according to compiler flags. There is necessary where every clock cycle is important.

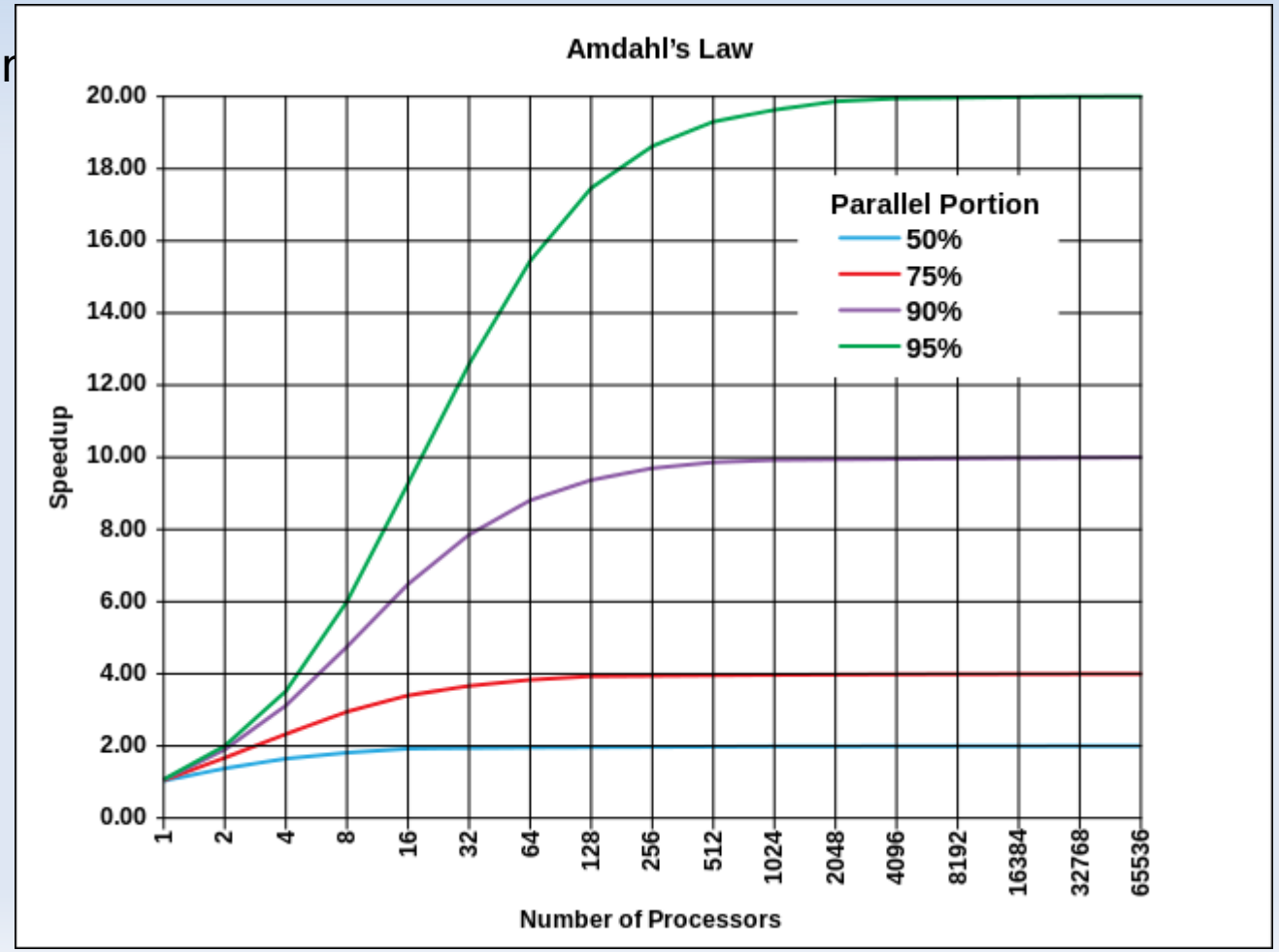
Limitations of Parallel Computation

Parallel programming and multicore systems should mean better performance. This can be expressed a ratio called speedup

Speedup (p) = Time (serial)/ Tir

Correctness in parallelisation requires synchronisation. Synchronisation and atomic operations causes loss of performance, communication latency.

Amdahl's law, establishes the maximum improvement to a system when only part of the system has been improved.



Stepping Around the Limits

Gustafson and Barsis (1988) noted that Amadahl's Law assumed a computation problem of fixed data set size.

If one increases the size of the dataset, the benefit of parallelisation increases!

Consider driving to Sydney from Melbourne via the coastline, with a “multicore engine” installed in Mallacoota, but then increasing this to other cities.

Cores	Mallacoota	Sydney	Total Time	Brisbane	Cairns	Total Time
1	8 hours	+8 hours	16 hours	+8 hours	+8 hours	32 hours
2	8 hours	+4 hours	12 hours	+4 hours	+4 hour	20 hours
4	8 hours	+2 hours	10 hours	+2 hours	+2 hours	14 hours
8	8 hours	+1 hour	9 hours	+1 hour	+1 hour	11 hours
.. ..	..	..	..	..	..	..
Inf	8 hours	+nil	8 hours	+nil	+nil	8 hours

If you can make use of parallelisation you should make use of it! It will always generate some benefit, and the larger the problem the bigger the gain.

UniMelb's HPC System: Spartan

A detailed review was conducted in 2016 looking at the infrastructure of the Melbourne Research Cloud, High Performance Computing, and Research Data Storage Services. University desired a 'more unified experience to access compute services'

Recommended solution, based on technology and usage, is to make use of existing NeCTAR Research cloud with an expansion of general cloud compute provisioning and use of a smaller "true HPC" system on bare metal nodes.

Since then Spartan has taken up a large GPGPU partition, moving from a small, experimental system, to a world-class facility. We have also moved all our partitions to physical nodes and have introduced interactive graphical tools with FastX and OpenOnDemand.

Complete list of current partitions and storage at :

https://dashboard.hpc.unimelb.edu.au/status_specs/

Spartan is ~~Small but~~ Important

Spartan as a model of an HPC-Cloud Hybrid has been featured at Multicore World, Wellington, 2016, 2017; eResearchAustralasia 2016 and several European HPC centres, including the European Organization for Nuclear Research (CERN), 2016, and the OpenStack Summit, Barcelona 2016.
https://www.youtube.com/watch?v=6D1lobuCZqE

Also featured in OpenStack and HPC Workload Management in Stig Telfer (ed), The Crossroads of Cloud and HPC: OpenStack for Scientific Research, Open Stack, 2016
http://openstack.org/assets/science/OpenStack-CloudandHPC6x9Booklet-v4-online.pdf

- Architecture also featured in:
Spartan and NEMO: Two HPC-Cloud Hybrid Implementations. 2017 IEEE 13th International Conference on e-Science, DOI: 10.1109/eScience.2017.70

The Chimera and the Cyborg, Hybrid Compute Advances in Science, Technology and Engineering Systems Journal Vol. 4, No. 2, 01-07, 2019

Other presentations on Spartan include use of the GPGPU partition at eResearch 2018, its development path at eResearchAU 2020, interactive HPC at eResearchNZ 2021, and more! Over 150 papers cite Spartan as a contributing factor their research.

Setting Up An Account and Training

Spartan uses its own authentication that is tied to the university Security Assertion Markup Language (SAML). The login URL is
https://dashboard.hpc.unimelb.edu.au/karaage

Users on Spartan must belong to a project. Projects must be led by a University of Melbourne researcher (the "Principal Investigator") and are subject to approval by the Head of Research Compute Services. Participants in a project can be researchers or research support staff from anywhere.

The University, through Research Platforms, has an extensive training programme for researchers who wish to use Spartan.

University of Melbourne is a major contributor to the International HPC Certification Program. https://www.hpc-certification.org/

University of Melbourne also contributes to the Easybuild software build system repository
https://easybuild.io/

Logging In and Help

To log on to a HPC system, you will need a user account and password and a Secure Shell (ssh) client. Linux distributions almost always include SSH as part of the default installation as does Mac OS 10.x, although you may also wish to use the Fugu SSH client. For MS-Windows users, the free PuTTY client is recommended. To transfer files use scp, WinSCP, Filezilla, and especially rsync.

Logins to Spartan are based on POSIX identity for the system

```
ssh your-username@spartan.hpc.unimelb.edu.au
```

To consider making an SSH config file, and using passwordless SSH. See: https://dashboard.hpc.unimelb.edu.au/ssh/

For help go to http://dashboard.hpc.unimelb.edu.au or check man spartan.

Lots of example scripts at /usr/local/common

Need more help? Problems with submitting a job, need a new application or extension to an existing application installed, if job generated unexpected errors etc., an email can be sent to: `hpc-support@unimelb.edu.au`

The Linux Environment and Modules

Everyone will need exposure to the GNU/Linux command line. If not, you'd better get some! At least learn the twenty or so basic environment commands to navigate the environment, manipulate files, manage processes. Plenty of good online material available (e.g., my book "Supercomputing with Linux", https://github.com/VPAC/superlinux)

Environment modules provide for the dynamic modification of the user's environment (e.g., paths) via module files. Each module contains the necessary configuration information for the user's session to operate according according to the modules loaded, such as the location of the application's executables, its manual path, the library path, and so forth.

Modulefiles also have the advantages of being shared with many users on a system and easily allowing multiple installations of the same application but with different versions and compilation options.

Modules Commands

Some basic module commands include the following:

`module help`
The command module help , by itself, provides a list of the switches, subcommands, and subcommand arguments that are available through the environment modules package.

`module avail`
This option lists all the modules which are available to be loaded.

`module whatis <modulefile>`
This option provides a description of the module listed.

`module display <modulefile>`
Use this command to see exactly what a given modulefile will do to your environment, such as what will be added to the PATH, MANPATH, etc. environment variables.

More Modules Commands

`module load <modulefile>`
This adds one or more modulefiles to the user's curre
environment (some modulefiles load other modulefile

module unload <modulefile>
This removes any listed modules from the user's current environment.

module switch <modulefile1> <modulefile2>
This unloads one modulefile (modulefile1) and loads another (modulefile2).

module purge
This removes all modules from the user's environmen

In the lmod system as used on Spartan there is also “module spider” which will search for all possible modules and not just those in the existing module path and provide descriptions.

(Image from NASA, Apollo 9 “spider module”)

Batch Systems and Workload Managers

The Portable Batch System (or simply PBS) is a utility software that performs job scheduling by assigning unattended background tasks expressed as batch jobs among the available resources.

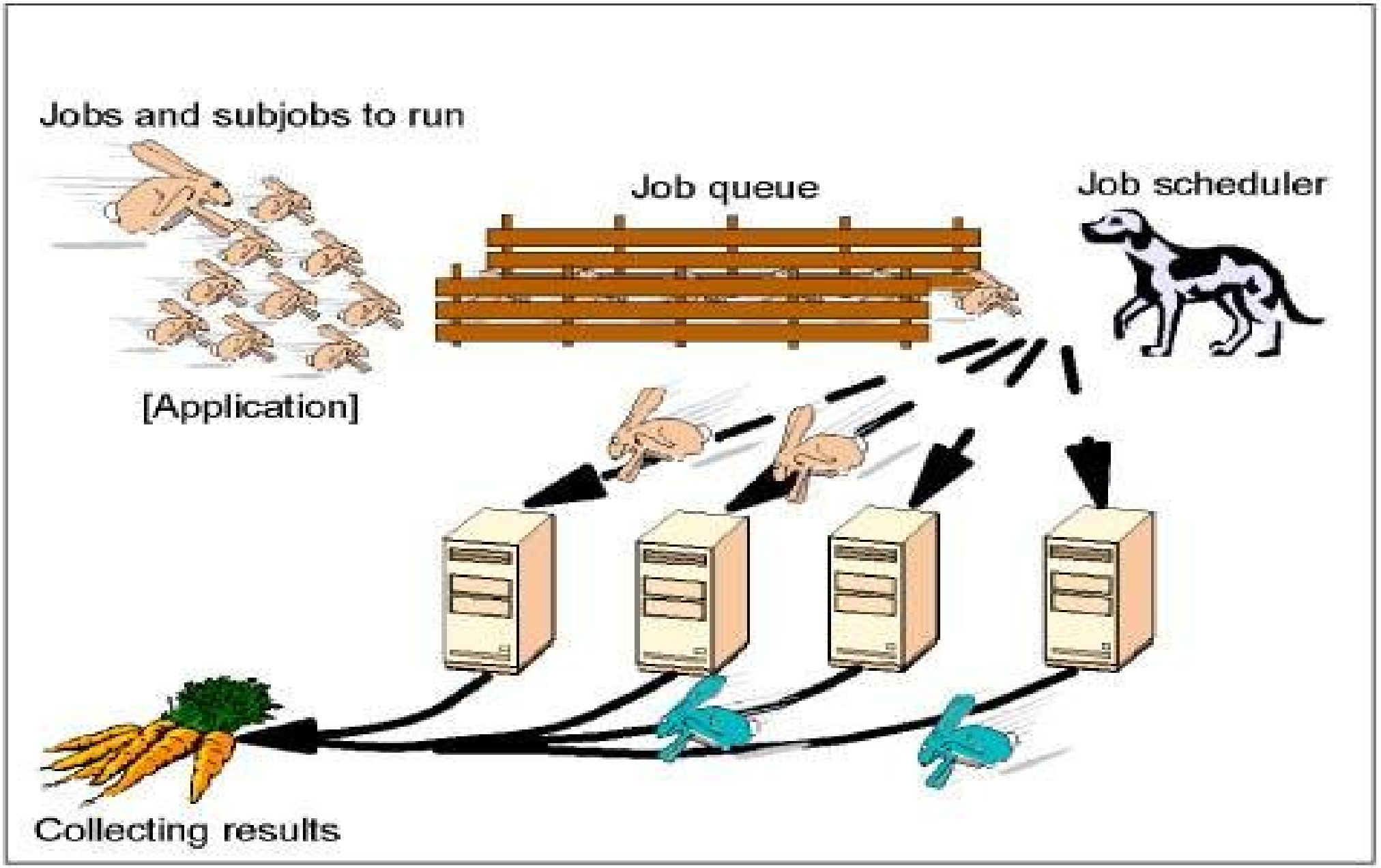
The original Portable Batch System was developed by MRJ Technology Solutions under contract to NASA in the early 1990s. In 1998 the original version of PBS was released as an open-source product as OpenPBS. This was forked by Adaptive Computing (formally, Cluster Resources) who developed TORQUE (Terascale Open-source Resource and QUEue Manager). Many of the original engineering team is now part of Altair Engineering who have their own version, PBSPro.

In addition to this the popular job scheduler Slurm (originally “Simple Linux Utility for Resource Management”), now simply called Slurm Workload Manager, also uses batch script where are very similar in intent and style to PBS scripts.

Spartan uses the Slurm Workload Manager. A job script written on one needs to be translated to another (handy script available pbs2slurm https://github.com/bjpop/pbs2slurm)

In addition to this variety of implementations of PBS different institutions may also make further elaborations and specifications to their submission filters (e.g., site-specific queues, user projects for accounting). (Image from the otherwise dry IBM 'Red Book' on Queue Management)

Submitting and Running Jobs



Submitting and Running Jobs

Submitting and running jobs is a relatively straight-forward process consisting of:

1) Setup and launch
2) Job Control, Monitor results
3) Retrieve results and analyse.

Don't run jobs on the login node! Use the queuing system to submit jobs.

1. Setup and launch consists of writing a short script that initially makes resource requests and then commands, and optionally checking queueing system.

Core command for checking queue:	squeue | less
Alternative command for checking queue:	showq -p physical | less
Core command for job submission:	sbatch [jobscript]

2. Check job status (by ID or user), cancel job.

Core command for checking job in Slurm:	squeue -j [jobid]
Detailed command in Slurm:	scontrol show job [jobid]
Core command for deleting job in Slurm:	scancel [jobid]

3.Slurm provides an error and output files They may also have files for post-job processing. Graphic visualisation is best done on the desktop.

Simple Script Example

```
#!/bin/bash
#SBATCH --partition=physical
#SBATCH --time=01:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
module load my-app-compiler/version
my-app data
```

The script first invokes a shell environment, followed by the partition the job will run on (the default is 'physical' for Spartan). The next four lines are resource requests, specifically for one compute node, one task. Note default values don't need to be included.

After these requests are allocated, the script loads a module and then runs the executable against the dataset specified. Slurm also automatically exports your environment variables when you launch your job, including the directory where you launched the job from. If your data is a different location this has to be specified in the path!

After the script is written it can be submitted to the scheduler.

```
[lev@spartan]$ sbatch myfirstjob.slurm
```

Multithreaded, Multicore, and Multinode Examples

Modifying resource allocation requests can improve job efficiency.

For example shared-memory multithreaded jobs on Spartan (e.g., OpenMP), modify the --cpus-per-task to a maximum of 8, which is the maximum number of cores on a single instance.

```
#SBATCH --cpus-per-task=8
```

For distributed-memory multicore job using message passing, the multinode partition has to be invoked and the resource requests altered e.g.,

```
#!/bin/bash
#SBATCH -partition=physical
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=4
module load my-app-compiler/version
srun my-mpi-app
```

Note that multithreaded jobs *cannot* be used in a distributed memory model across nodes. They can however exist be conducted on distributed memory jobs which include a shared memory component (hybrid OpenMP-MPI jobs).

Arrays and Dependencies

Alternative job submissions include specifying batch arrays, and batch dependencies.

In the first case, the same batch script, and therefore the same resource requests, is used multiple times. A typical example is to apply the same task across multiple datasets. The following example submits 10 batch jobs with myapp running against datasets dataset1.csv, dataset2.csv, ... dataset10.csv

```
#SBATCH --array=1-10
myapp ${SLURM_ARRAY_TASK_ID}.csv
```

In the second case a dependency condition is established on which the launching of a batch script depends, creating a conditional pipeline. The dependency directives consist of `after`, `afterok`, `afternotok`, `before`, `beforeok`, `beforenotok`. A typical use case is where the output of one job is required as the input of the next job.

```
#SBATCH --dependency=afterok:myfirstjobid mysecondjob
```

Interactive Jobs

For real-time interaction, with resource requests made on the command line, an interactive job is called. This puts the user on to a compute node.

This is typically done if they user wants to run a large script (and shouldn't do it on the login node), or wants to test or debug a job. The following command would launch one node with two processors for ten minutes.

```
[lev@spartan interact]$ sinteractive --nodes=1 --ntasks-per-node=2
srun: job 164 queued and waiting for resources
srun: job 164 has been allocated resources
[lev@spartan-rc002 interact]$
```

PBS, SLURM Comparison

User Commands	PBS/Torque	SLURM
Job submission	qsub [script_file]	sbatch [script_file]
Job submission	qdel [job_id]	scancel [job_id]
Job status (by job)	qstat [job_id]	squeue [job_id]
Job status (by user)	qstat -u [user_name]	squeue -u [user_name]
Node list	pbsnodes -a	sinfo -N
Queue list	qstat -Q	squeue
Cluster status	showq, qstatus -a	squeue -p [partition]
Environment		
Job ID	$PBS_JOBID	$SLURM_JOBID
Submit Directory	$PBS_O_WORKDIR	$SLURM_SUBMIT_DIR
Submit Host	$PBS_O_HOST	$SLURM_SUBMIT_HOST
Node List	$PBS_NODEFILE	$SLURM_JOB_NODELIST
Job Array Index	$PBS_ARRAYID	$SLURM_ARRAY_TASK_ID

PBS and SLURM Comparison

Job Specification	PBS	SLURM
Script directive	#PBS	#SBATCH
Queue	-q [queue]	-p [queue]
Job Name	-N [name]	--job-name=[name]
Nodes	-l nodes=[count]	-N [min[-max]]
CPU Count	-l ppn=[count]	-n [count]
Wall Clock Limit	-l walltime=[hh:mm:ss]	-t [days-hh:mm:ss]
Event Address	-M [address]	--mail-user=[address]
Event Notification	-m abe	--mail-type=[events]
Memory Size	-l mem=[MB]	--mem=[mem][M|G|T]
Proc Memory Size	-l pmem=[MB]	--mem-per-cpu=[mem][M|G|T]

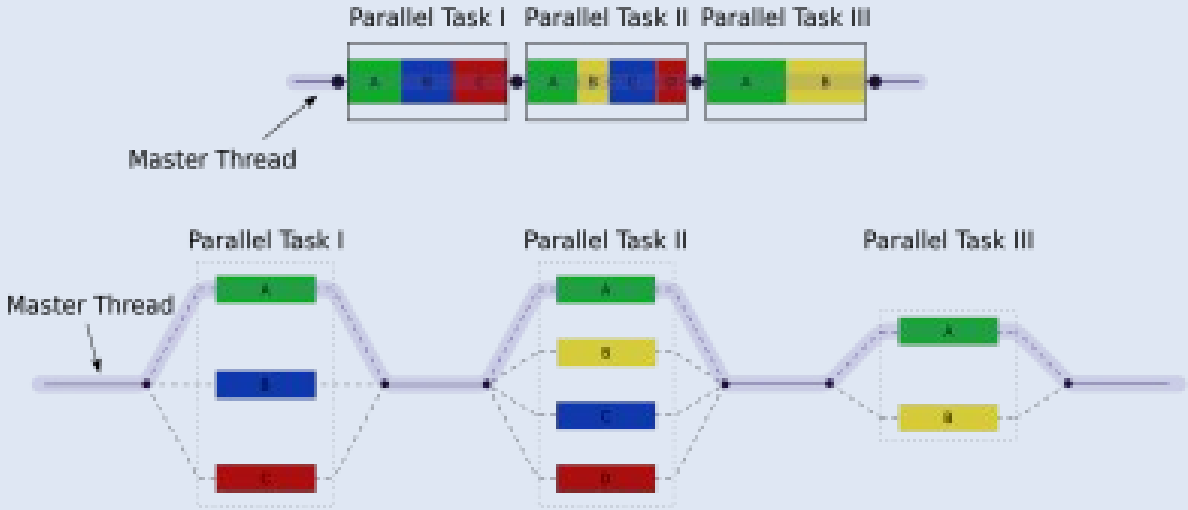
Shared Memory Parallel Programming

One form of parallel programming is multithreading, whereby a master thread forks a number of sub-threads and divides tasks between them. The threads will then run concurrently and are then joined at a subsequent point to resume normal serial application.

One implementation of multithreading is OpenMP (Open Multi-Processing). It is an Application Program Interface that includes directives for multi-threaded, shared memory parallel programming. The directives are included in the C or Fortran source code and in a system where OpenMP is not implemented, they would be interpreted as comments.

There is no doubt that OpenMP is an easier form of parallel programming, however it is limited to a single system unit (no distributed memory) and is thread-based rather than using message passing. Many examples in `/usr/local/common/OpenMP`.

(image from: User A1, Wikipedia)



Shared Memory Parallel Programming

```
#include <stdio.h>
#include  "omp.h"
int main(void)
{
    int id;
    #pragma omp parallel num_threads(8) private(id)
    {
    int id = omp_get_thread_num();
    printf("Hello world %d\n", id);
    }
return 0;
}
```

```
program hello2omp
    include "omp_lib.h"
    integer :: id
    !$omp parallel num_threads(8) private(id)
      id = omp_get_thread_num()
           print *, "Hello world", id
    !$omp end parallel
end program hello2omp
```

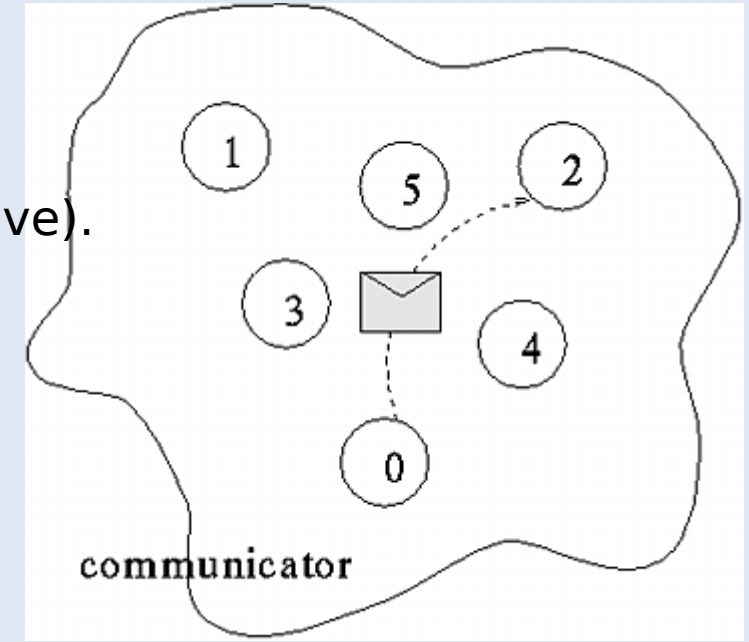
Distributed Memory Parallel Programming

Moving from shared memory to parallel programming involves a conceptual change from multi-threaded programming to a message passing paradigm. In this case, MPI (Message Passing Interface) is one of the most well popular standards and is used here, along with a popular implementation as OpenMPI.

The core principle is that many processors should be able cooperate to solve a problem by passing messages to each through a common communications network.

The flexible architecture does overcome serial bottlenecks, but it also does require explicit programmer effort (the "questing beast" of automatic parallelisation remains somewhat elusive).

The programmer is responsible for identifying opportunities for parallelism and implementing algorithms for parallelisation using MPI.



Distributed Memory Parallel Programming

```
#include <stdio.h>
#include "mpi.h"
int main( argc, argv )
int  argc;
char **argv;
{
    int rank, size;
    MPI_Init( &argc, &argv );
    MPI_Comm_size( MPI_COMM_WORLD, &size );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    printf( "Hello world from process %d of %d\n", rank, size );
    MPI_Finalize();
    return 0;
}
```

```
!       Fortran MPI Hello World
        program hello
        include 'mpif.h'
        integer rank, size, ierror, tag, status(MPI_STATUS_SIZE)

        call MPI_INIT(ierror)
        call MPI_COMM_SIZE(MPI_COMM_WORLD, size, ierror)
        call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierror)
        print*, 'node', rank, ': Hello world'
        call MPI_FINALIZE(ierror)
        end
```

MPI Compilation and Job Scripts

The OpenMP example needs to be compiled with OpenMP directives. The OpenMP example cannot run across compute nodes; therefore it is best run on the “cloud” partition. The OpenMPI compilation needs to call the MPI wrappers.

```
module load OpenMPI/1.10.0-GCC-4.9.2
gcc -fopenmp helloomp.c -o helloompc
mpigcc mpihelloworld.c -o mpihelloworld

#!/bin/bash
#SBATCH -p cloud
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=16
export OMP_NUM_THREADS=16
module load GCC/4.9.2
mpiexecu helloompc

#!/bin/bash
#SBATCH -p physical
#SBATCH --nodes=2
#SBATCH --ntasks=16
module load OpenMPI/1.10.2-GCC-4.9.2
mpiexec mpi-helloworld
```

MPI4Py for Python

Python too has various MPI bindings available. The most common used is MPI4Py. Examples are in the Spartan directory, `/usr/local/common`. As a package it can be simply imported (e.g., `from mpi4py import MPI`). The “Hello World” example has the core routines from MPI.

```
from mpi4py import MPI
import sys
size = MPI.COMM_WORLD.Get_size()
rank = MPI.COMM_WORLD.Get_rank()
print("Helloworld! I am process %d of %d.\n" % (rank, size))
```

But remember! With environment modules with extensions you do not necessarily get all the packages/libraries/extensions that you might expect. See the README file for an explanation of how to review the extensions already installed.

Examples for MPI4Py (and others) will be conducted at the workshop!

References

[1] Hilbert M, Lopez P. "The world's technological capacity to store, communicate, and compute information". Science. 332 (6025) 2011
[2] Guo, Huadong, et al. "Scientific big data and digital earth." Chinese Science Bulletin 59.35 (2014): 5066-5073.
[3] Apon, Amy., et al., High Performance Coputing Instrumentation and Research Productivity in U.S. Universities, Journal of
Information Technology Impact, Vol 10, No 2, pp87-98, 2010
[4] Joseph, Earl., et al., Creating Economic Models Showing the Relationship Between Investments in HPC and the Resulting Financial
ROI and Innovation — and How It Can Impact a Nation's Competitiveness and Innovation, IDC Special Study, October 2013
[5] Law, R., Lukoyanova, N., Voskoboinik, I. et al. The structural basis for membrane binding and pore formation by lymphocyte perforin. Nature 468, 447–451 (2010). https://doi.org/10.1038/nature09518
[6] Kuiper, Michael J et al. “The biological function of an insect antifreeze protein simulated by molecular dynamics.” eLife vol. 4 e05142. 7 May. 2015, doi:10.7554/eLife.05142
[7] Rubio-Martinez, Marta, et al. "New synthetic routes towards MOF production at scale." Chemical Society Reviews 46.11 (2017): 3453-3480.


THANKS FOR WATCHING
& LISTENING PATIENTLY
makeameme.org