SLOT 1

Introduction to Basics of High Performance Computing

Laboratory of Statistical and Computing Services
INSPEM, UPM 2016

Outcome

- Basic concepts & understanding of HPC and cluster computing.
- INSPEM HPC Cluster.
- Essential things required for HPC deployment from Windows.

What is HPC?

- Definition depends on individual person
  * HPC is when I care how fast I get an answer…
- Happens on (deployment):
  * Personal computer
  * Supercomputer
  * Cluster of computers (Linux) ⟵ our focus / GPU & GPGPU
  * Grid or cloud
  * Hybrid (any combination of the above)
- Thus, ‘P’ in HPC: Performance? Precision? Productivity?
- Formal definition: Refers to the use of computing systems or resources comprised of multiple processors linked together in a single system.
- Supercomputers: Most visible manifestation of HPC

Who may need HPC?

- My problem is big…
- My problem is complex…
- My computer is too slow and too small in terms computing power…
- My software is not efficient and/or not parallel?

A High Performance Problem

A Parallel Solution!

HPC Cluster (briefly)

- Most desktops and laptops today are parallel (multi-core processors)
- A cluster needs:

  * Several computers, nodes, hardware similar to a workstation, but in special cases for rack mounting

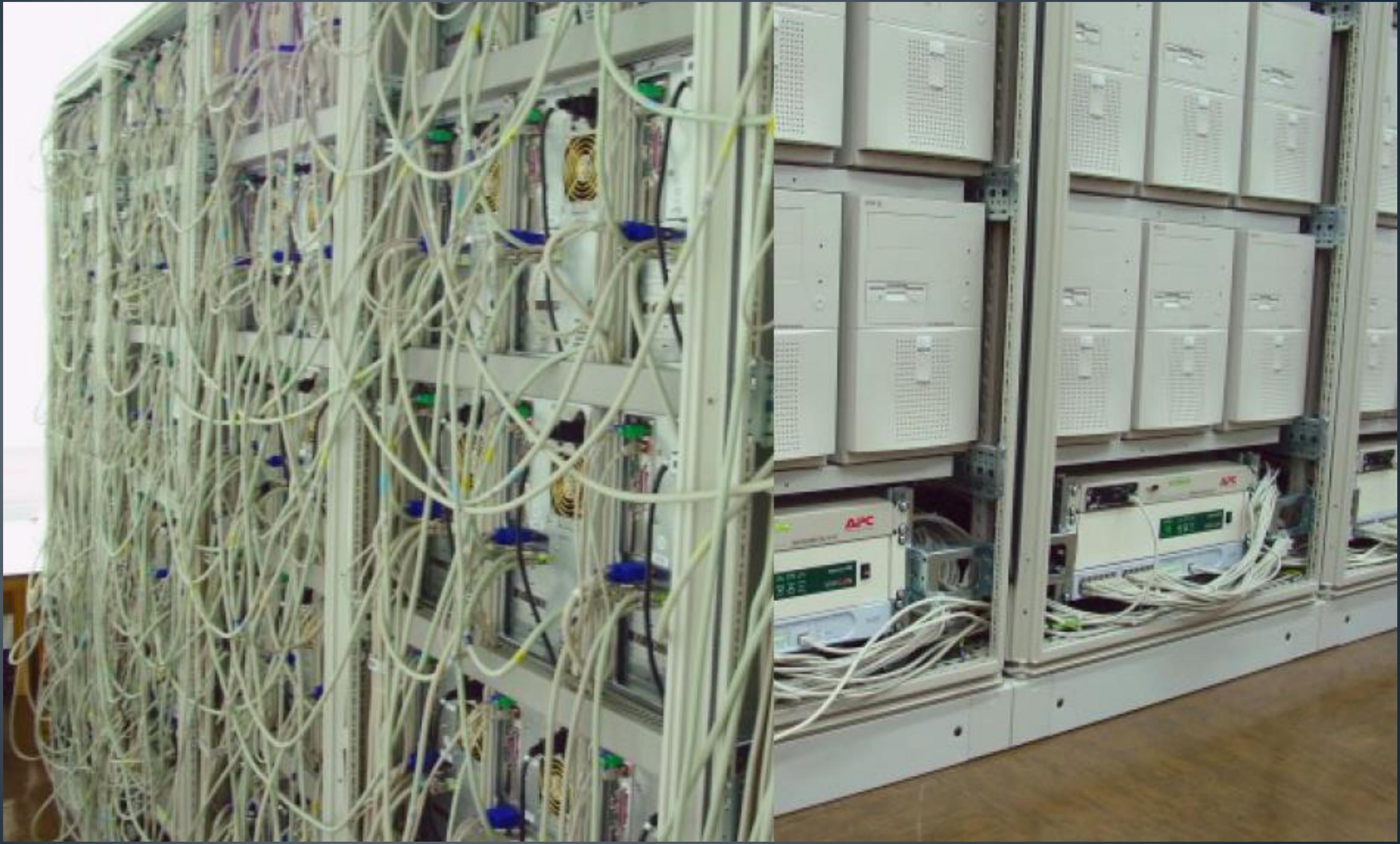
  * One or more networks (interconnects) for inter-node communication and accessing common resources

  * Software that allows the nodes to communicate with each other (usually via calls to an MPI library)

  *Software that reserves resources to individual users
- A cluster is all of those components working together to form one parallel computing device.

The Good Side (2002)

The Ugly Side (2002)



The Beautiful (2012)

Data Centre

Supercomputers (Large scale HPC Clusters)

IBM POWER5

HP/Compaq Alphaserver

Intel IA32

AMD Opteron

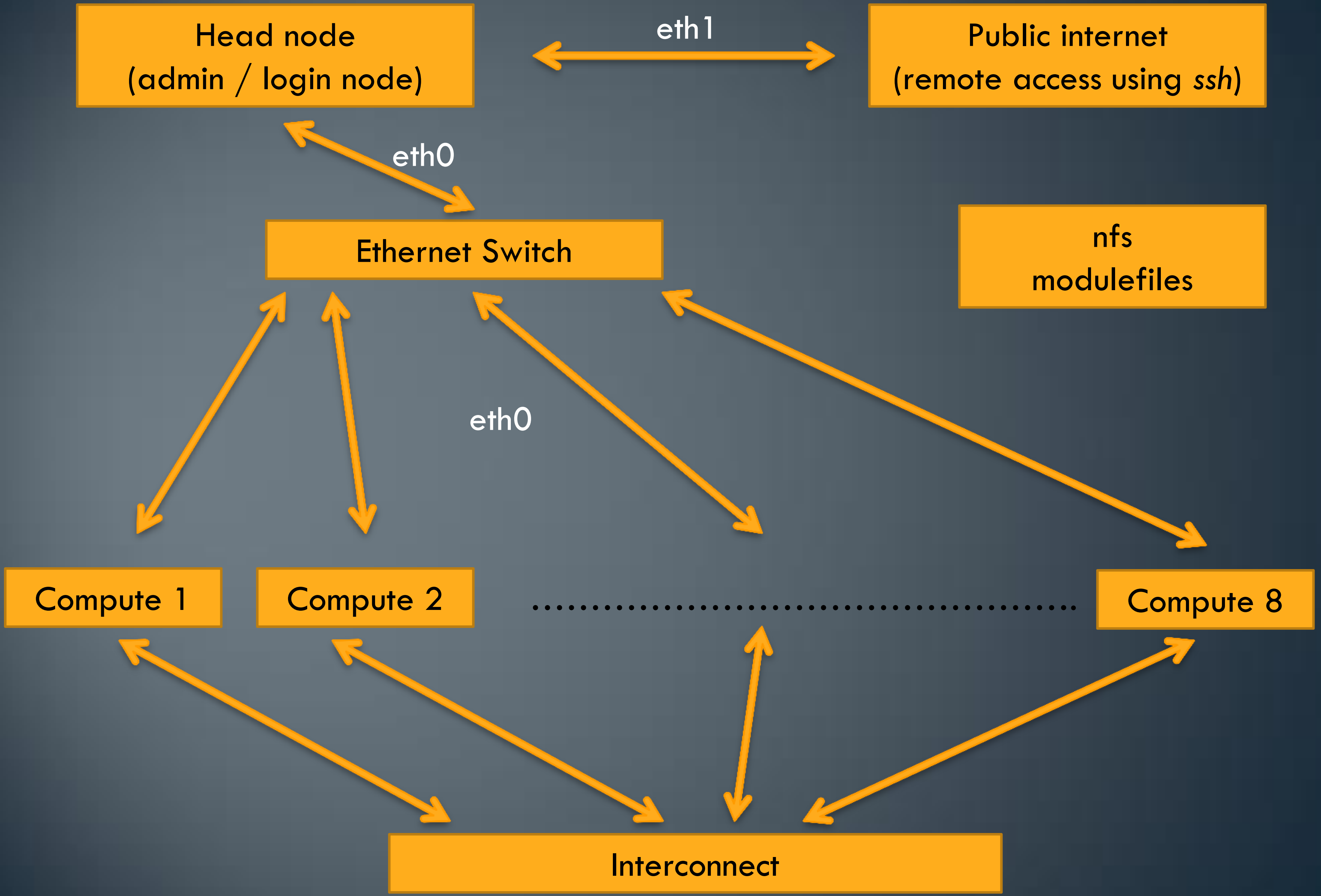
Cray XT3

IBM BG/L

JAGUAR

sgi

INSPEM HPC Cluster



Head node
(admin / login node)

eth1

Public internet
(remote access using *ssh*)

eth0

Ethernet Switch

nfs
modulefiles

eth0

Compute 1

Compute 2

……………………………………

Compute 8

Interconnect

INSPEM HPC Cluster Resources List

- IBM System X3650 M4 Server
- Head node hostname: ce.inspemhpc.upm.edu.my
- Head node IP address: 172.16.241.116
- 8 compute nodes
  * Cores : 96 (12 cores each)
  * Processors: 4 x Intel Xeon 6C @ 2.0 Ghz
  * Memory Capacity: 384GB (48GB each)
  * Storage Capacity: 2.3TB (292GB each)
  * Interconnect: Ethernet (Gigabit)
  * OS: Scientific Linux 6.2
  * MPI library: OpenMPI
  * Job Scheduler: MAUI
  * Resource Manager: PBS / Torque

$1 \leq$ nodes ≤ 8

$1 \leq$ ppn ≤ 12

Available software

- OpenFOAM *(Computational Fluid Dynamics)*
- NWChem *(Computational Chemistry)*
- GROMACS *(Molecular Dynamics)*
- Scilab *(Numerical Computing)*
- BLAST *(Bioinformatics)*
- AMBER *(Molecular Dynamics)*
- ClustalW *(Bioinformatics)*
- MATLAB R2013A *(Numerical Computing)*
- Mathematica 9 *(Numerical Computing)*
- *EasyCluster (Bioinformatics)*
- *GMAP (Bioinformatics)*
- *TGI-CL (Bioinformatics)*
- *Maker (Bioinformatics)*
- *Mothur (Bioinformatics)*

- CHARMM *(Molecular Dynamics)*
- R / R Studio Server *(Computational Statistics)*
- FastQC *(Bioinformatics)*
- Bowtie 2 *(Bioinformatics)*
- Cufflinks *(Bioinformatics)*
- Velvet *(Bioinformatics)*
- Oases *(Bioinformatics)*
- TopHat *(Bioinformatics)*
- SAMtools *(Bioinformatics)*
- Artemis *(Bioinformatics)*
- FASTX-Toolkit *(Bioinformatics)*
- COMSOL *(Computational Physics)*
- Quantum ESPRESSO *(Computational Chemistry & Computational Physics)*
- OM NeT++ *(Network Simulation)*
- NS 2 / NS 3 *(Network Simulation)*

What do you need from Windows (mostly)?

- Secure Shell (SSH) Terminal
  e.g.: PuTTY
- Secure Copy (SCP) Client
  e.g.: WinSCP
- X Emulator (*optional for graphics display)
  e.g.: Xming
- Some useful (basic) Linux commands
  e.g.: to view / create / edit files
- PBS & Maui specific commands
  e.g.: for job monitoring

SLOT 2

Introduction to Basics of High Performance Computing

Laboratory of Statistical and Computing Services
INSPEM, UPM 2016

Outcome

- Familiar with remote access and file transfer.
- Familiar with basic Linux, PBS & Maui commands.
- Brief understanding on parallel programming & message passing interface.
- OpenMPI
- Execute serial & parallel program in C

Tasks

- Login to WinSCP
- Transfer files
- Login to PuTTY
- Play around with Linux, PBS & Maui commands
- Run *hw_serial.c* and *hw_parallel.c*

*** Massively parallel, embarrassingly parallel, Flynn's taxonomy, SPMD, MPMD

Parallel Computing / Programming (briefly)

- Traditionally, software has been written for serial computation: to be run on a single computer having a single Central Processing Unit (CPU) / processor/ processor with single core;
- A problem is broken into a discrete series of instructions.
- Instructions are executed one after another.
- Only one instruction may execute at any moment in time. Only one processor / core is used
- **Parallel computing** is the simultaneous use of multiple compute resources to solve a computational problem: to be run using multiple processors
- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different processors
- An overall control/coordination mechanism is employed

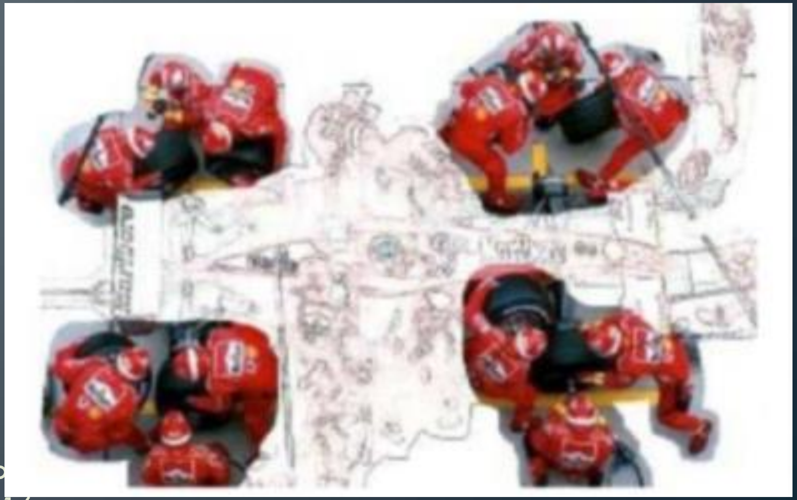
Two types of parallelism

- Functional parallelism:

  Different people are performing different tasks at the same time. Tasks typically have different duration.

- Data parallelism:

  Different people are performing the same task, but on different equivalent and independent objects. Same duration for tasks.



Message Passing Interface (MPI)

- Multiple and separate processes concurrently that are coordinated exchange data through "messages"
- A standard, i.e. there is a document describing how the API (constants & subroutines) are named and should behave; multiple "levels", MPI-1 (basic), MPI-2 (advanced), MPI-3 (new)
- An implementation of the standard:
  * Open source and commercial versions
  * Vendor specific versions for certain hardware
- Our focus: OpenMPI
- Implemented earlier in C & Fortran, later on C++ and now extended to many languages like Python, R, Matlab…

MPI Implementation

- A fully functional MPI program can be written by using only 6 MPI functions:
- MPI_Init() - initialisation
- MPI_Comm_size() – communicator size (how many MPI tasks?)
- MPI_Comm_rank() – process rank (ID of processor in the group)
- MPI_Bcast() or MPI_Send()
- MPI_Reduce() or MPI_Recv()
- MPI_Finalize()
- ** OpenMP is another type of parallel implementation (shared memory) – less programming ← not our focus
- MPI + OpenMP = Hybrid

C Hello World Serial

```
#include <stdio.h>
int main ()
{
printf ("Hello, world!\n");
return 0;
}
```

C Hello World Parallel

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

SLOT 3

Introduction to Basics of High Performance Computing

Laboratory of Statistical and Computing Services
INSPEM, UPM 2016

Outcome

- Familiar with PBS job script & job management commands.
- Familiar with job submission & output retrieval.
- Able to write job scripts for different jobs.
- If time permits: familiar with interactive job submission, interactive node request and X11 forwarding.

PBS Job script

- Shell scripts (usually with .sh extension) used to execute programs in compute nodes from head node.
- Very easy to write… if you follow the standard format

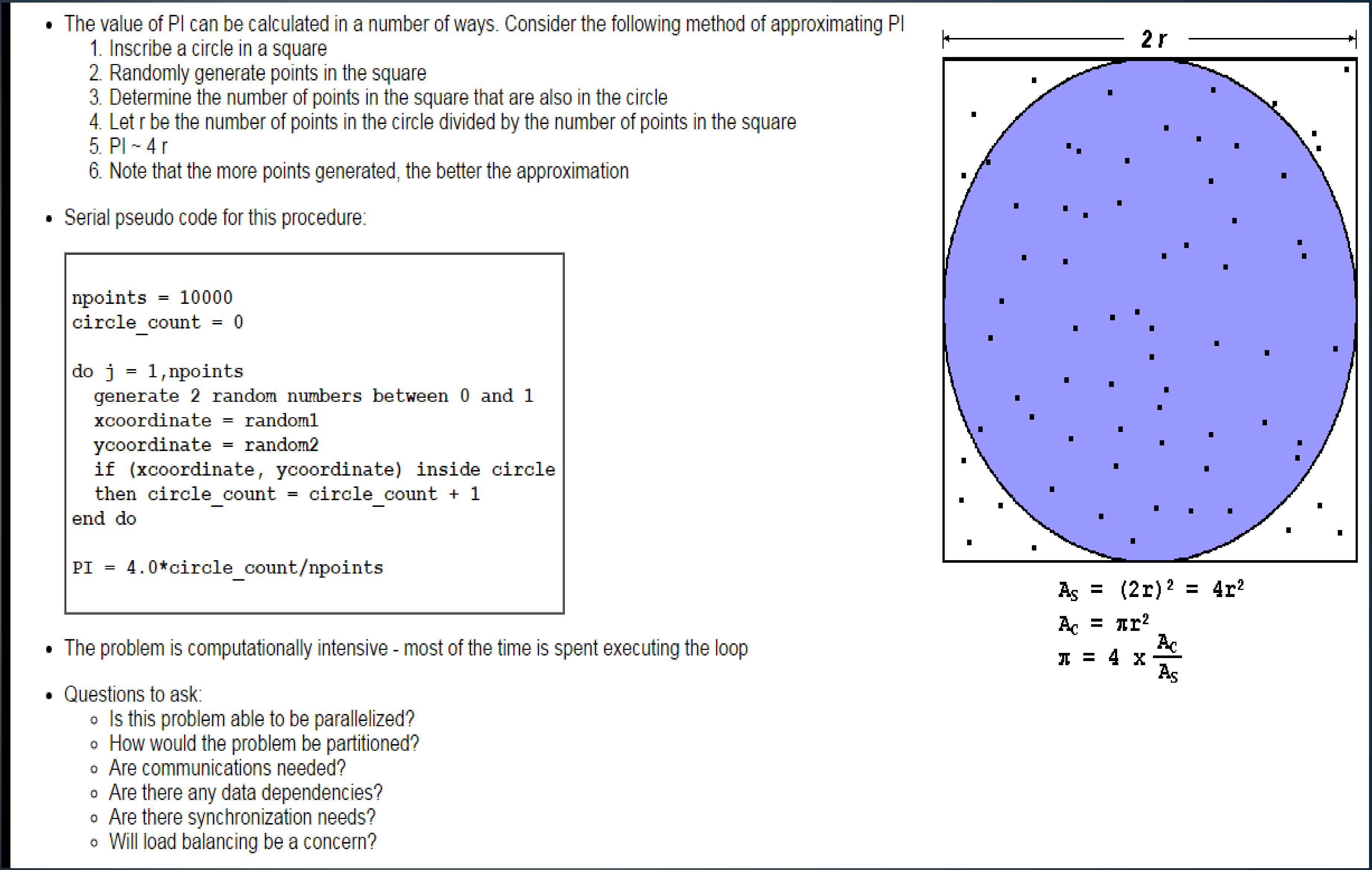
Standard job script

-
  ```
  #!/bin/sh
  #PBS -N JobName
  #PBS -l nodes=1:ppn=1
  cd PBS_O_$WORKDIR
  commands to execute
  ```
- **Optional flags:**
  ```
  #PBS -o stdout_file
  #PBS -e stderr_file
  #PBS -j oe
  #PBS -l cput = hh:mm:ss
  #PBS -l mem =512mb
  #PBS -l walltime = hh:mm:ss
  #PBS -m ae
  #PBS -M user_email_address
  #PBS –q queue_name
  ```

Tasks

- Create job scripts
- Run C , Python, MATLAB and R programs
- View outputs
- If time permits: interactive job and X11 forwarding
- MPI pi Calculation Example:
  * This program calculates pi using a "dartboard" algorithm. (See Fox et al.(1988) Solving Problems on Concurrent Processors, vol.1 * page 207)
  * All processes contribute to the calculation, with the master averaging the values for pi. This version uses mpc_reduce to collect results

Pi calculation



- The value of PI can be calculated in a number of ways. Consider the following method of approximating PI
  1. Inscribe a circle in a square
  2. Randomly generate points in the square
  3. Determine the number of points in the square that are also in the circle
  4. Let r be the number of points in the circle divided by the number of points in the square
  5. PI ~ 4 r
  6. Note that the more points generated, the better the approximation

- Serial pseudo code for this procedure:

```
npoints = 10000
circle_count = 0

do j = 1,npoints
  generate 2 random numbers between 0 and 1
  xcoordinate = random1
  ycoordinate = random2
  if (xcoordinate, ycoordinate) inside circle
  then circle_count = circle_count + 1
end do

PI = 4.0*circle_count/npoints
```

$$A_S = (2r)^2 = 4r^2$$

$$A_C = \pi r^2$$

$$\pi = 4 \times \frac{A_C}{A_S}$$

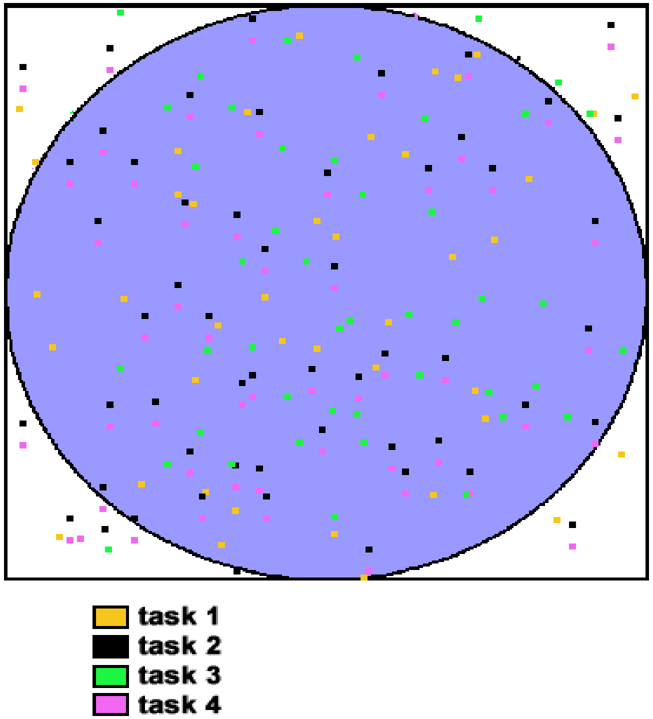
- The problem is computationally intensive - most of the time is spent executing the loop

- Questions to ask:
  - Is this problem able to be parallelized?
  - How would the problem be partitioned?
  - Are communications needed?
  - Are there any data dependencies?
  - Are there synchronization needs?
  - Will load balancing be a concern?

Parallel solution:

- Another problem that's easy to parallelize:
  - All point calculations are independent; no data dependencies
  - Work can be evenly divided; no load balance concerns
  - No need for communication or synchronization between tasks
- Parallel strategy:
  - Divide the loop into equal portions that can be executed by the pool of tasks
  - Each task independently performs its work
  - A SPMD model is used
  - One task acts as the master to collect results and compute the value of PI
- Pseudo code solution: **red** highlights changes for parallelism.

```
npoints = 10000
circle_count = 0

p = number of tasks
num = npoints/p

find out if I am MASTER or WORKER

do j = 1,num
  generate 2 random numbers between 0 and 1
  xcoordinate = random1
  ycoordinate = random2
  if (xcoordinate, ycoordinate) inside circle
  then circle_count = circle_count + 1
end do

if I am MASTER

  receive from WORKERS their circle_counts
  compute PI (use MASTER and WORKER calculations)

else if I am WORKER

  send to MASTER circle_count

endif
```


🟨 task 1
⬛ task 2
🟩 task 3
🟪 task 4