Parallel programming practice (1-2)

MPI example, environment: para.iit.bme.hu

Szeberényi Imre

BME IIT

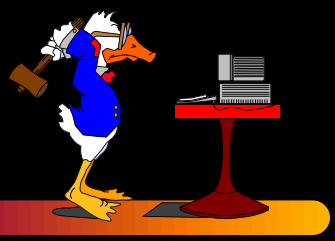
<szebi@iit.bme.hu>

MŰEGYETEM 1782

Environment

- Cluster in the CIRCLE cloud
- para.iit.bme.hu – main machine 4 vCPU
- cn01, cn02, cn03 – workers 4 vCPU
- /users, /usr – NFS
- module – flexibile handling of environment parameters
- slurm – sheduler and resourcemanager

Login to the cluster



Using Linux based client (eg. from lab)

ssh NEPTUN@para.iit.bme.hu

Using Windows based free client: (putty.org)

putty

host: para.iit.bme.hu

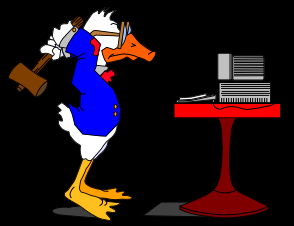
login: NEPTUN

module

- A bit forgotten tool for quick and maintainable change of environment variables to select alternative program variants.

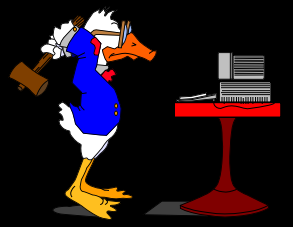
Example:

```
module load mpi
printenv | grep MPI_LIB
MPI_LIB=/usr/local/openmpi/lib
module unload mpi
module load mvapich2
printenv | grep MPI_LIB
MPI_LIB=/usr/local/mvapich2/lib
```



http://modules.sourceforge.net

module example 2



```
module load mpi
module load cuda
module load dot
module list
Currently Loaded Modulefiles:
  1) mpi    2) cuda   3) dot
```

The dot inserts the current directory into PATH

MPI example #1 (.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

int main(int argc, char **argv) {
  int rank, size, len;
  char hostname[MPI_MAX_PROCESSOR_NAME];

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  MPI_Get_processor_name(hostname, &len);
  printf("I am %s %d of %d on host %s\n",
              argv[0], rank, size, hostname);
  MPI_Finalize();
  return 0;
}
```

MPI example #1 (.cc)

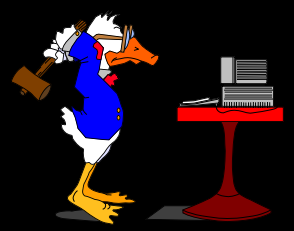
```
#include <iostream>
#include <mpi.h>
using namespace std;

int main(int argc, char *argv[]) {
  MPI::Init(argc, argv);

  int rank = MPI::COMM_WORLD.Get_rank();
  int size = MPI::COMM_WORLD.Get_size();
  int len;
  char hostname[MPI_MAX_PROCESSOR_NAME];
  MPI::Get_processor_name(hostname, len);

  cout << "I am " << rank << " of " << size
       << " on host " << hostname << endl;
  MPI::Finalize();
  return 0;
}
```

MPI compile & run



```
module load mpi
cd
cp  -r ~szebi/para/MPI   .  # Mind the dot !
cd  MPI
mpicc -o mpihello mpihello.c
mpirun -np 2 mpihello
Hello, world!  I am ./mpihello 1 of 3 on host para
Hello, world!  I am ./mpihello 0 of 3 on host para

 # There is no resource allocation and scheduling
 # The mpirun starts as many instances as we request.
```

MPI running on the cluster

The mpirun

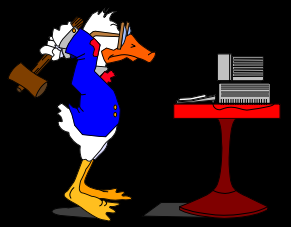
- enters in to the nodes, or
- uses daemons (eg. LAM)

to launch the program instances, but does not do any resource allocation, nor scheduling.
In multi-user environment the resource allocation is essential. There are several tools for this (condor, pbs, sge, slurm, ...)

We use SLURM.

MPI example using SLURM

```
run_mpi.sh:
#!/bin/bash
#SBATCH -o std.out
mpirun $@
```

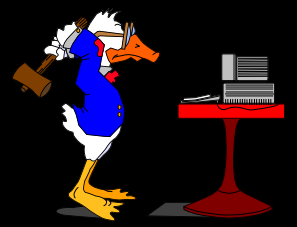


```
sbatch –n 3 run_mpi.sh ./mpihello
more std.out
Hello, world!  I am ./mpihello 0 of 3 on host cn01
Hello, world!  I am ./mpihello 2 of 3 on host cn01
Hello, world!  I am ./mpihello 1 of 3 on host cn01
```

SLURM

- Simple Linux Utility for Resource Management
- Resources are divided into partitions that can overlap. On para.iit.bme.hu machine we have 3 partitions:
  prod, debug, misi
- Major commands:
  - sinfo – node and partition info
  - srun – running a parallel job
  - sbatch – start a batch script
  - squeue – query the queue
  - salloc – resource reservation for interactive usage
  - scancel – delete job from the queue

SLURM example /1



sinfo
- Information about the resources

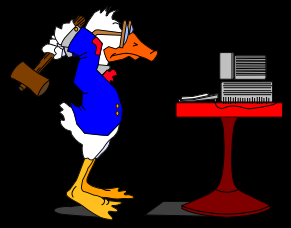
srun -n 2 /bin/hostname
- Allocate 2 CPU's and start the command /bin/hostnae on each.

srun -N 2 /bin/hostname
- Allocate 2 node's and start the command on each.

salloc -n 4 /bin/bash
- Allocate 4 CPU's and start an interactive bash command on the first one.

SLURM example /2

```
cd
cp  -r ~szebi/para/slurm    .  # Mind the dot !
cd  slurm
sbatch -n 3 simple.sh
```



- Creates a job requesting 3 CPUs and returns the command prompt.
- When the resources are available, it allocates them and launches simple.sh on one. Pass the names of the reserved resources by environment variables.

simple.sh:

```
#!/bin/bash
printenv | grep SLURM
hostname
```

Most important switches

-n number of tasks to run
--ntasks-per-node=n number of tasks per node
-N number of nodes (N = min[-max])
-o location of stdout
-p partition requested
--prolog=program run "program" before job step
-i location of stdin
-t time limit
--mincpus=n minimum number of logical CPUs

Switches in the script

Switches can be integrated with simple syntax.Pl:

```
#!/bin/bash
#SBATCH -n 3
#SBATCH -o output_file
#SBATCH -D working_dir
#SBATCH --mail-type=end
#SBATCH --mail-user=xyz@xyz.de
#SBATCH -t 01:00:00
srun /bin/hostname    # like srun –n 3
```

Running SPMD programs

File: run.sh

```
#!/bin/bash
mpirun $@
```

Command:

```
sbatch –n 4 run.sh ./mpihello
```

File: runhello.sh

```
#!/bin/bash
#SBATCH -o  hello.txt
#SBATCH -n 4
mpirun ./mpihello
```

Command:

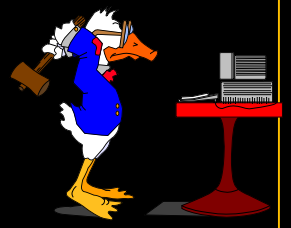
```
sbatch runhello.sh
```

Running MPMD programs

File: multi.sh

```
#!/bin/bash
#SBATCH -N 2
#SBATCH -n 8
#SBATCH -o multi.out
srun --multi-prog ./multi.conf
```

File: multi.conf



```
0 ./master.sh
## slaves
* ./slave.sh
```

Command:

```
sbatch multi.sh
```

Major environmental variables :
SLURM_PROCID
SLURM_NODEID
SLURM_LOCALID

PI example (serial version)

cd ~/MPI

more pi.c

gcc -o pi pi.c

time ./pi 5000000

PI_MPI example

```
module load mpi
cd ~/MPI
mpicc -o pi_mpi pi_mpi.c

# Measuring run times on  vCPU's
sbatch -o pi.out run_mpi.sh ./pi_mpi 5000000
# run times with 4 processors
sbatch -o pi4.out -n 4 run_mpi.sh ./pi_mpi 5000000
```

Assignment

1. Learn / try the environment
2. Paralellise a simple program :

The filt.c in the ~szebi/para/filt directory performs an edge enhancement on a 128x128 pixel grayscale image.

This should be parallelised using MPI

Action Steps

1. Understand the program!
2. Modify the serial code so that it can handle any size PGM format images. You can also check the impoved version in ~szebi/para/filt3 directory.
3. Check the operation of the modified serial program with the specified sample files!
   - Compare the result with cmp
   - Check out the result graphically!
     To do this, traser the pgm files to the local machine and chek it using PGM capable viewer (xnView, gpicview, ...). (see next slides)

Action Steps cont.

4. Parallelize the program using data domain parallelization.
5. Check the operation of the parallel program with the specified sample files!
   - Compare the result with cmp
   - Check out the result graphically!
     To do this, traser the pgm files to the local machine and chek it using PGM capable viewer (xnView, gpicview, ...).
6. Check the Speed Up

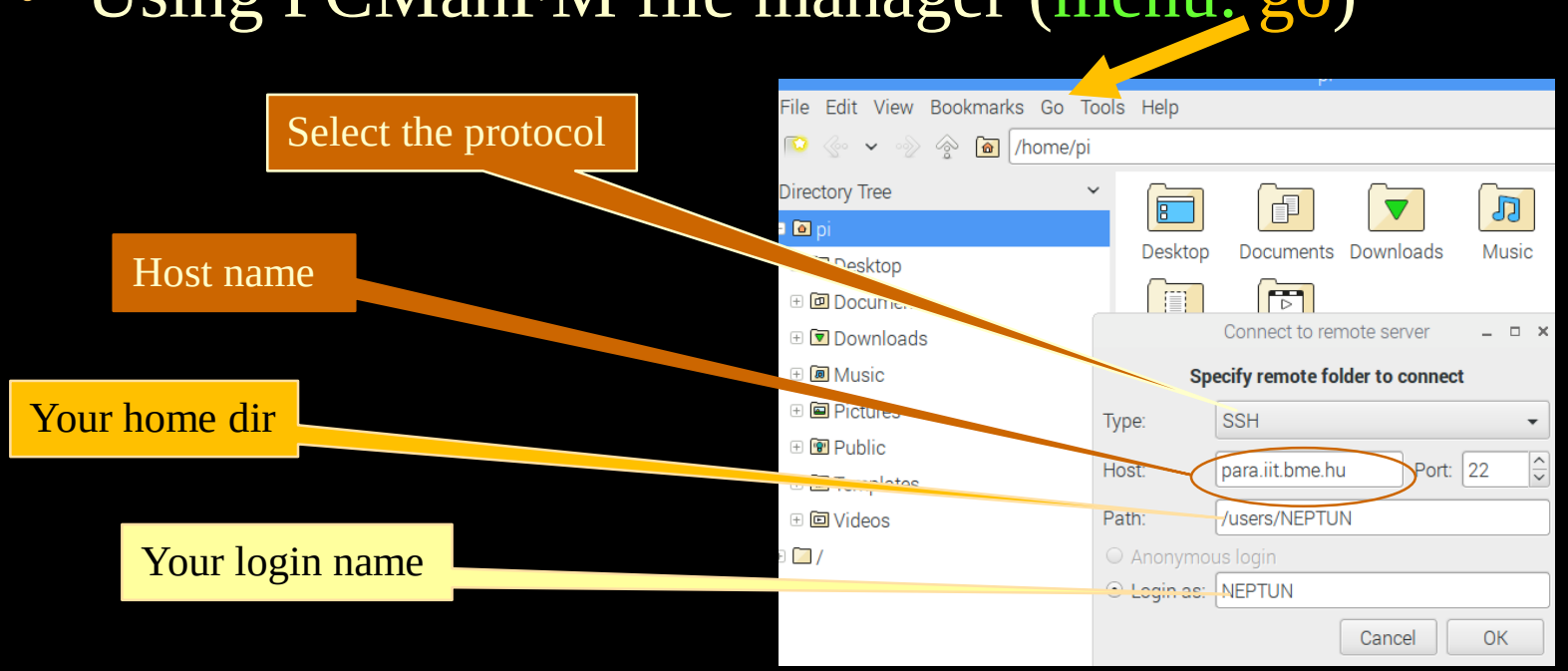
Tools/methods good to know

- General UNIX/LINUX commands
- File transfers between the local and remote machine.
- C/C++ development
- File transfer between tha local and remote sites
- .png file format
- MPI C/C++ API
  - MPI Python also available

Transferring files

Transferring files from para.iit.bme.hu to a Linux machine:

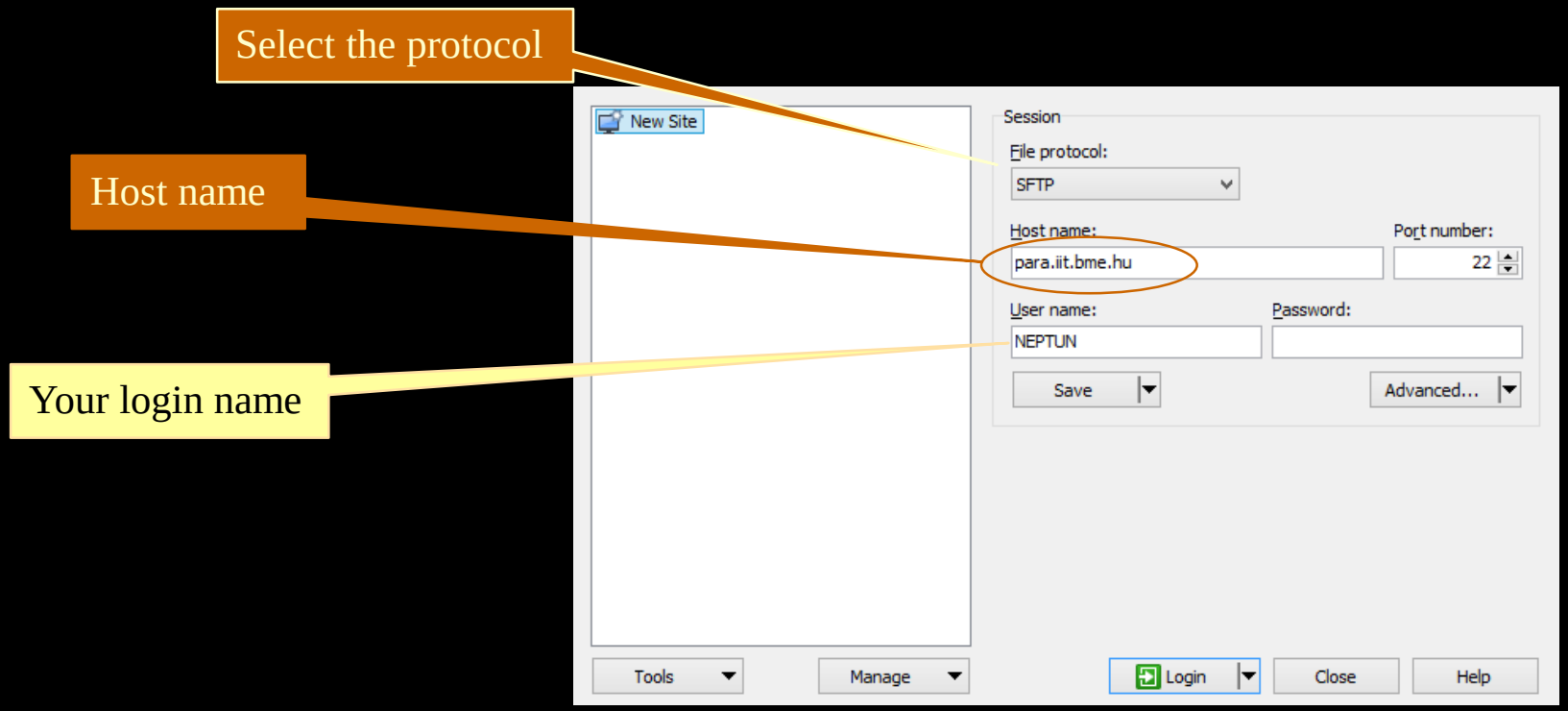
- Using Linux command line: (scp from to)
  > scp NEPTUN@iit.bme.hu: . # mind the dot
- Using PCManFM file manager (menu: go)



Transferring files

Transferring files from para.iit.bme.hu to a Windows machine:

- Using WinSCP freeware utility (winscp.net):



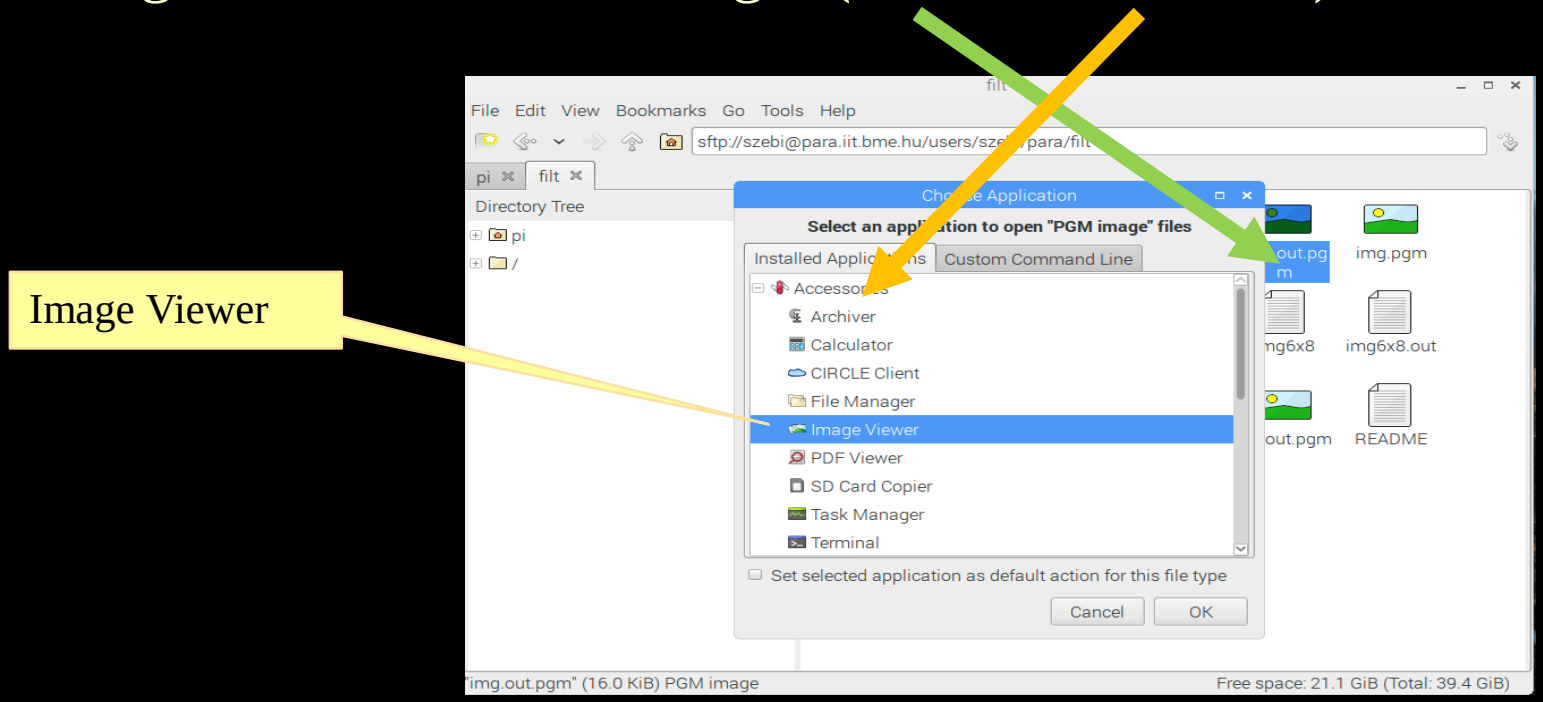
Viewing the .pgm image

1. Viewing on remote machine (para.iit.bme.hu)
   - You should have a standard Linux graphical interface (X window server) on local OS.
   - You should enable the X protocol between the remote and local machine: E.g. logging in by ssh –X NEPTUN@para.iit.bme.hu command to the remot server.
   - use the gpicview pic.out.pgm

2. Or the images should be transferred to the local machine.

Viewing the lcal .pgm image

- Using gpicview Linux utility:
  > gpicview pic.out.pgm
- Using PCManFM file manager (click, Accessories)



- Using XnView or IntraWiev on MS Windows.