Parallel programming practice (1-2)

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

Szeberényi Imre

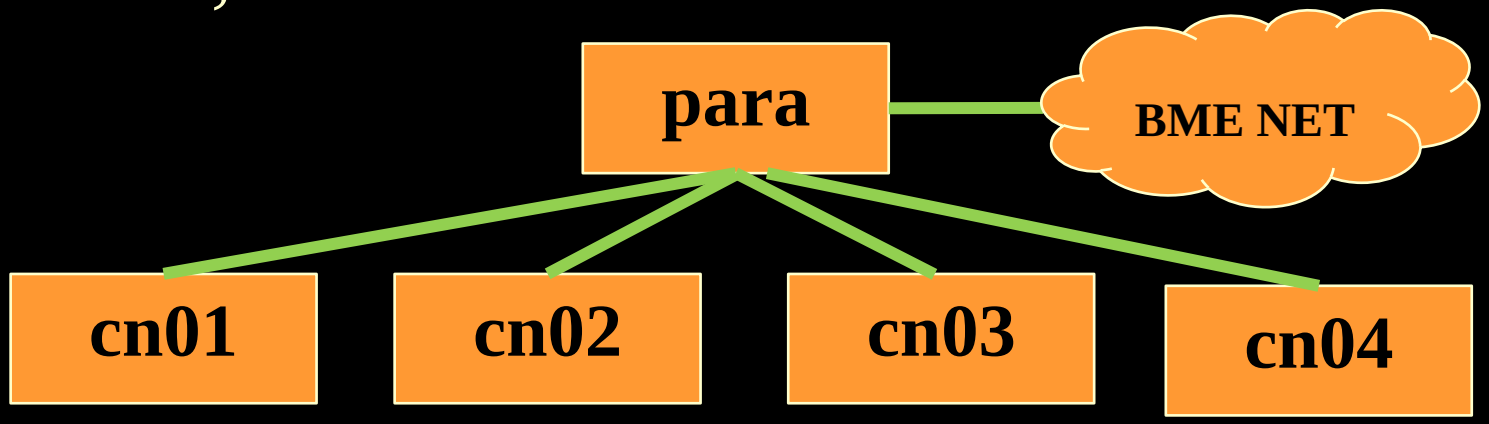
BME IIT

<szebi@iit.bme.hu>

MŰEGYETEM 1782

Environment /1

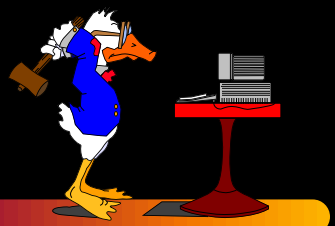
- Cluster in the CIRCLE cloud
- para.iit.bme.hu – main machine 4 vCPU
- cn01, cn02, cn03, cn04 – workers 4 vCPU
- /users, /usr – NFS



Environment /2

- module
  - Easy modification of the environment variables that different tools require.
- slurm
  - Sheduler and resource manager. This tool ensures an efficient and fair share of resources.
- compilers
- development tools

Login to the cluster



Using Linux based client (eg. from lab, win10)

ssh neptun@para.iit.bme.hu

Using Windows based free client: (putty.org)

putty

host: para.iit.bme.hu

login: neptun

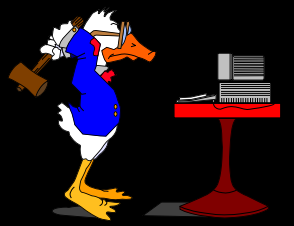
OR you can use VScode Remote ssh extension

module

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

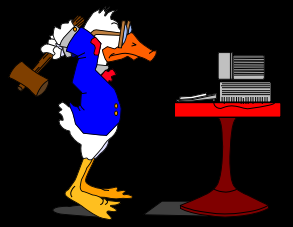
Example:

```
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

Try the linda example

```
# Login to the cluster
module load linda
cd
cp  -r ~szebi/para/linda    .  # Mind the dot !
cd  linda
clc -w -linda compile_args -m32 -o chello chello.cl
./chello

# There is no resource allocation and scheduling
# The program runs on the para headnode
# It is not efficient, Solution: see later!
```

MPI example #1 (.c)

```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)

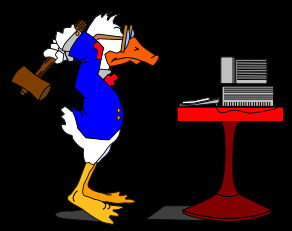
```
#include <iostream>
#include <mpi.h>
using namespace std;

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 2 on host para
Hello, world!  I am ./mpihello 0 of 2 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

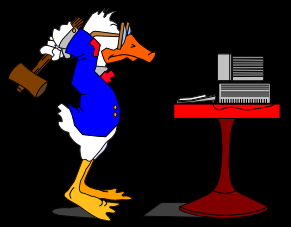
- enters in to the nodes, or
- uses daemons (eg. LAM)

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:

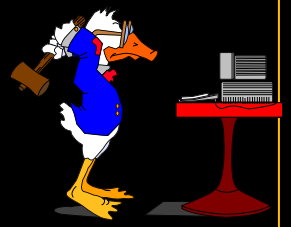
```
sbatch runhello.sh
```

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
```

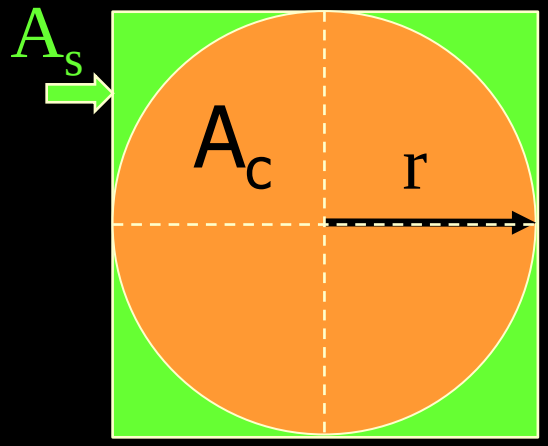
Command:

```
sbatch multi.sh
```

Major environmental variables :
SLURM_PROCID
SLURM_NODEID
SLURM_LOCALID

PI example

$$A_c = \pi r^2$$
$$A_s = (2r)^2$$
$$\pi = 4\frac{A_c}{A_s}$$




```
int isInside() {
    double x = ((double) rand()) / RAND_MAX;
    double y = ((double) rand()) / RAND_MAX;
    if ((x*x + y*y) <= 1.0) return(1);
    else return(0);
}
```

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
```

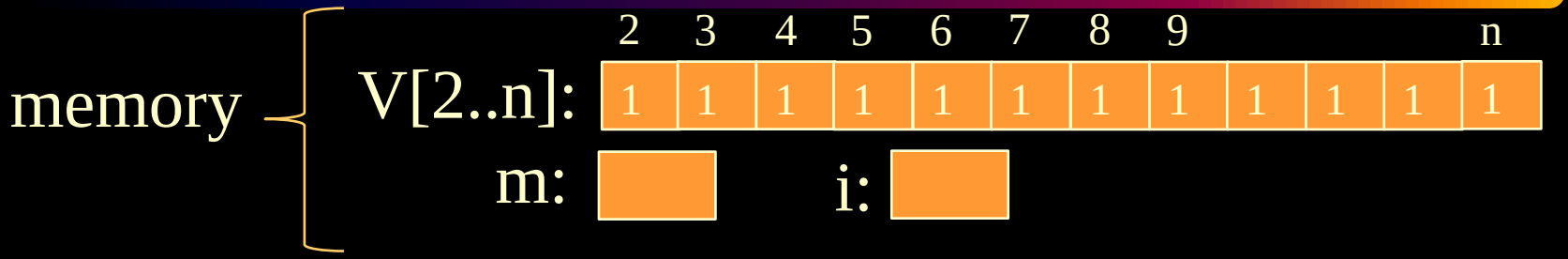
Problem: Primes in [1..n]

Init.	Pass 1	Pass 2	Pass 3
2←*m*	2	2	2
3	3←*m*	3	3
4			
5	5	5←*m*	5
6			
7	7	7	7 ← *m*
8			
9	9		
10			
11	11	11	11
12			
13	13	13	13
14			
15	15		
16			
17	17	17	17
18			
19	19	19	19
20			
21	21		
22			
23	23	23	23
24			
25	25	25	

The sieve of Eratosthenes yielding a list of primes for $n = 25$.
Multiples of marked element erased from the list.

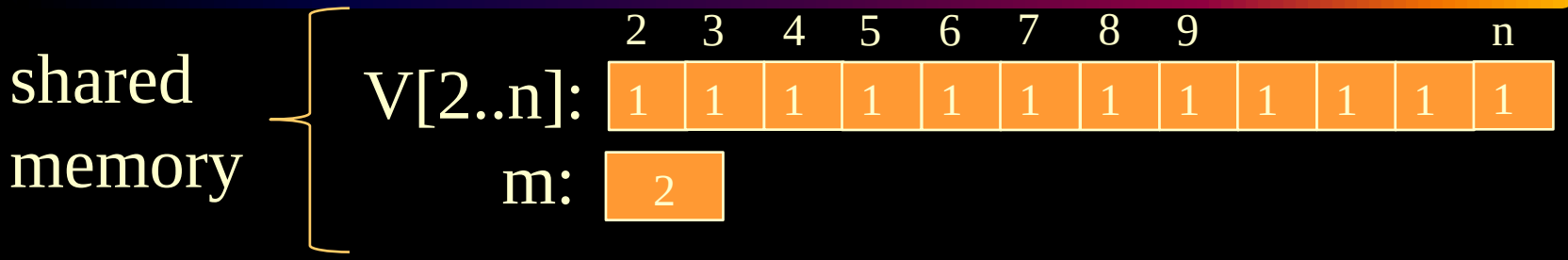
$m^2 > n$, the computation stops

Serial implementation

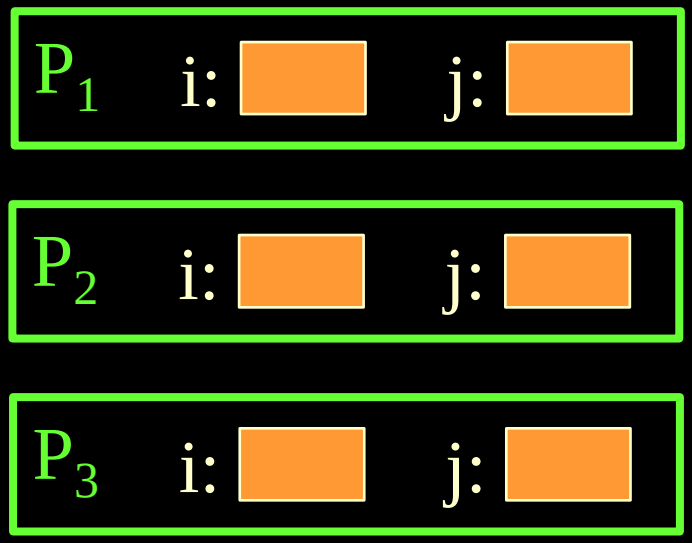



```
m:= 2
while m*m < n do
  if V[m] = 1 then
    i:= m*m
    while i <= n do
      v[i] = 0
      i:= i+m
    od
  fi
  m:= m+1
od
return all i such that V[i] = 1
```

Control-paralel implementation

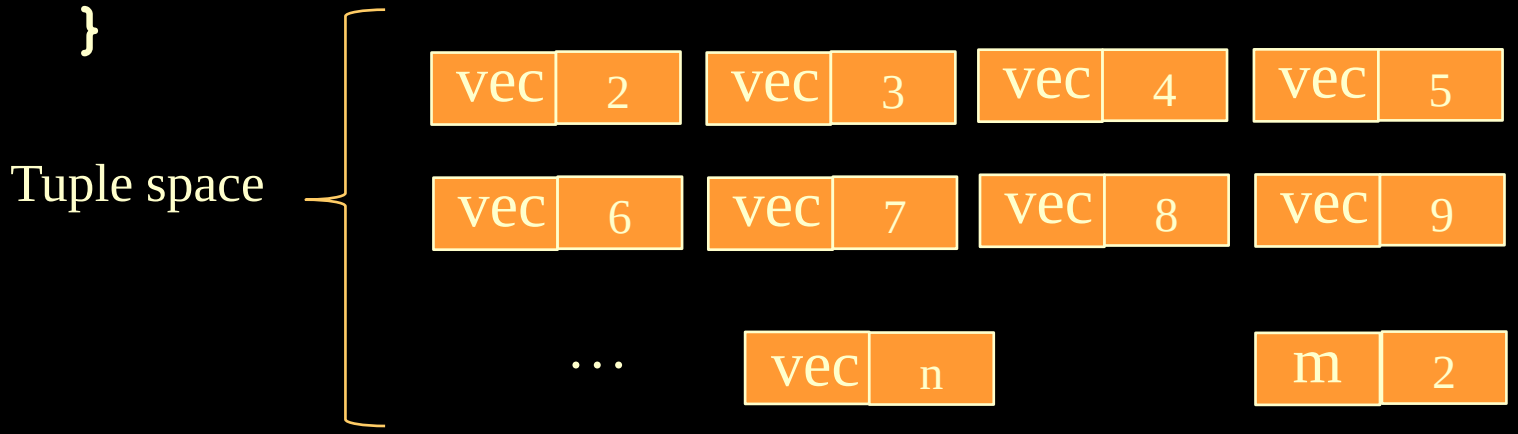



```
j:= m, m:= m+1 // atomic
while j*j < n do
  if V[j] = 1 then
    i:= j*j
    while i <= n do
      v[i] = 0
      i:= i+j
    od
  fi
  j:= m, m:= m+1 // atomic
od
wait for all and return all i such that V[i] = 1
```



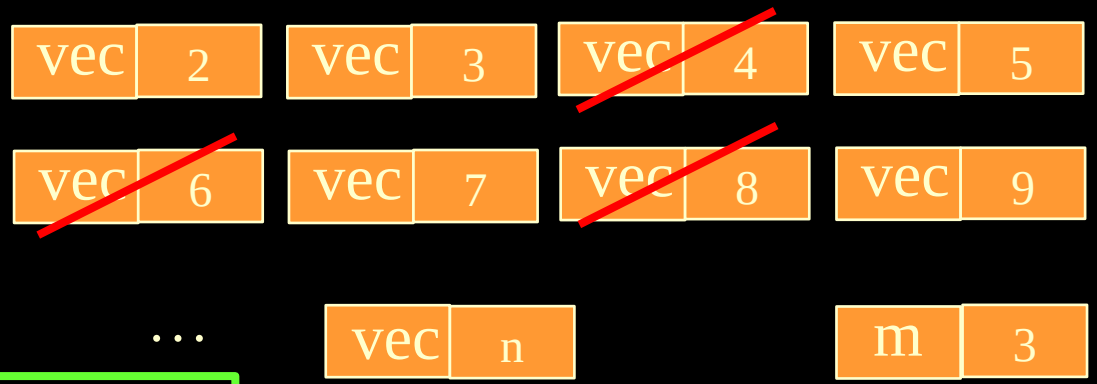
Linda implementation

```
// init the shared „variables”
in tuple space
void init(int n) {
  int i;
  for (i = 2; i <= n; i++)
    out("vec", i);
  out("m", 2);
}
```



Linda implementation #2

```
int proc(int p, int n) {
  int i, j;
  in("m", ?j);     // assume: j == 2
  out("m", j+1);
  while (j*j < n) {
    if (rdp("vec", j)) {
       i = j*j;
       while (i <= n) { inp("vec", i); i = i+j; }
    }
    in("m", ?j);
    out("m", j+1);
  }
  in("done", ?i);
  out("done", i+1);
  return 0;
}
```



for coordination only

Linda implementation #3

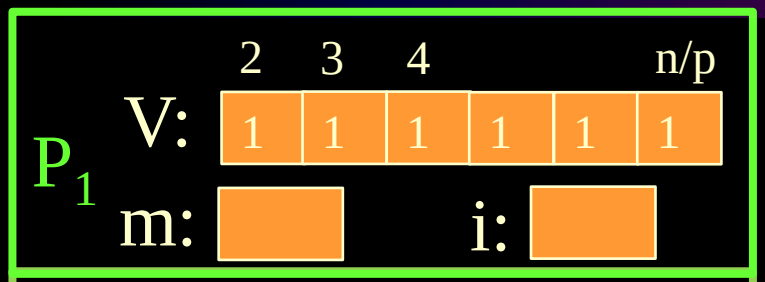
```
int real_main(int argc, char *argv[]) {
  int i,  int N = 1000, NP = 4;
  init(N);
  out("done", 0);
  for (i = 1; i <= NP; i++)
     eval(proc(i, N));

  in("done", NP);
  printf("\nAll (%d) processors done\n", NP);

  printf("Primes in [1..%d]:\n", N);
  for (i = 2; i <= N; i++) {
    if (inp("vec", i))  printf(" %d", i);
  }
  return 0;
}
```

for coordination only

Data-paralel implementation




```
m:= 2
while m <= n/p &&
              m*m < n do
  if V[m] = 1 then
    broadcast m →
    i:= m*m
    while i <= n do
      v[i] = 0, i:= i+m
    od
  fi
  m:= m+1
od
broadcast 0 and wait for results
```

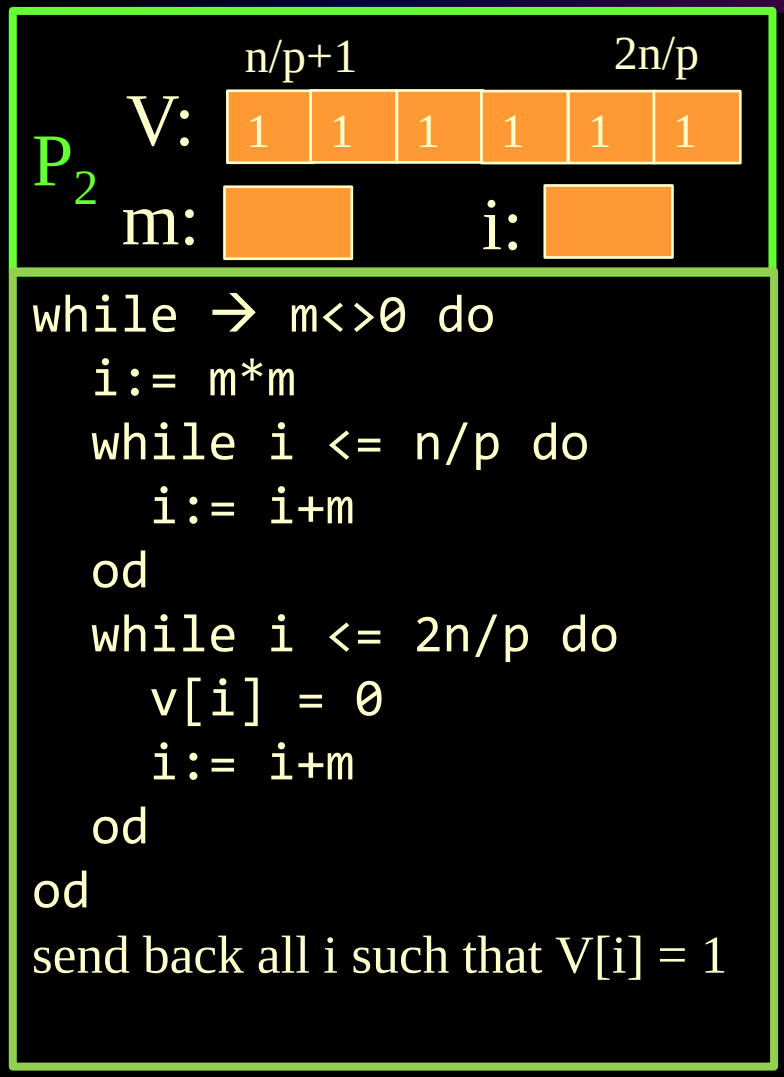
Assume that $p < \sqrt{n}$

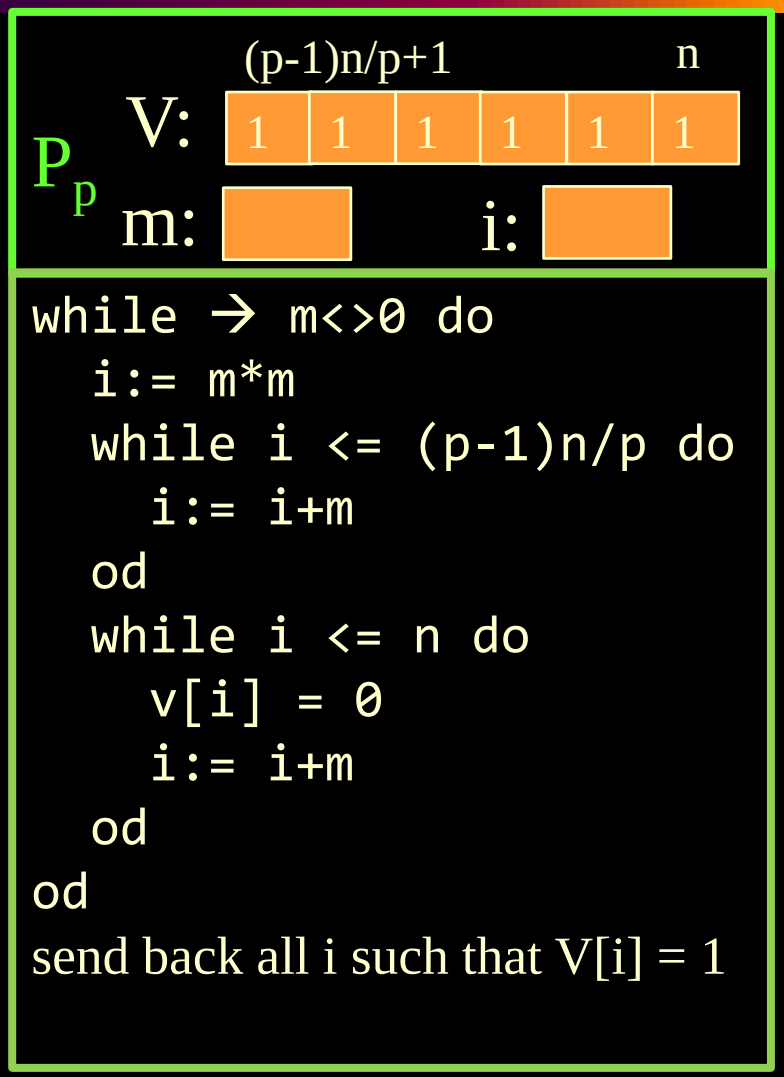
→ $\sqrt{n} < n / p$

→ All the primes that have multiples are located on the P_1 processor.

P_1 processor act as coordinator.

Data-paralel implementation #2





Assignment

1. Learn / try the environment
2. Create a MPI porgram solving prime number generation. Use the data-parallel solution.

Theory:

Check the slides 22-29

Help: ~szebi/para/MPI/myprime_serial.cc

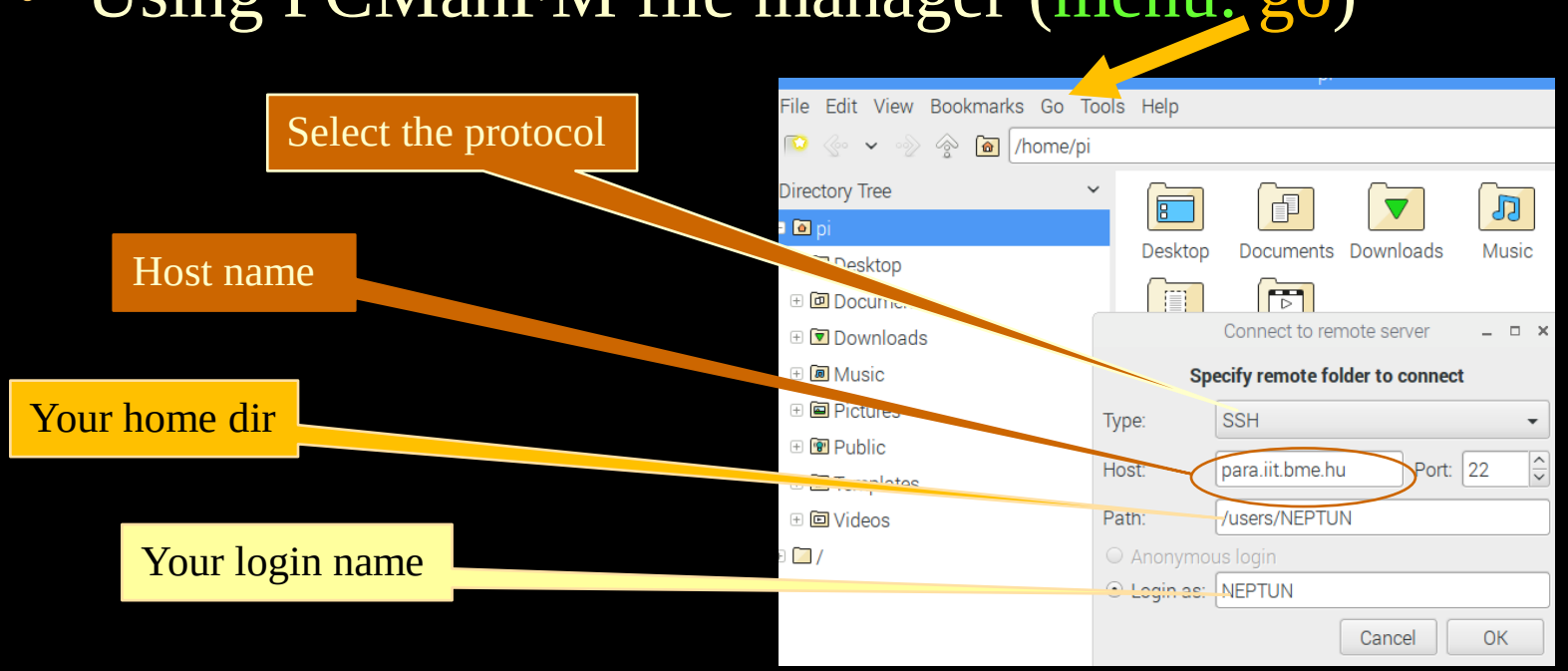
Tools/methods good to know

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

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):