# CFN Beowulf cluster facility

Internal report 1/2005
CFN/IST

Tiago Pereira
Jorge Ferreira
Horácio Fernandes

This document describes CFN PC Linux cluster — Orionte — in three ways: a) hardware assembling, b) software assembling, c) user operation. Thus, it is helpful to: a) physical reproduction of the cluster, b) software installation and administration of new clusters, c) operation by engineers and physicists.

Contents

# 1 Software user guide

## *Logging in to Orionte*

Orionte is located at:

```
193.136.136.59
```

or alternatively at:

```
orionte.cfn.ist.utl.pt
```

Users can reach it with any ssh client.

If you are on Windows, putty is an acceptable one, freely distributed at

```
http://www.chiark.greenend.org.uk/
~sgtatham/putty/download.html
```

If you are on Linux, you'll just need to type

```
shell > ssh orionte.cfn.ist.utl.pt
```

at the console and then type your username and password.

## *Copying file to and from Orionte*

This section explains how to copy files from one's personal computer to Orionte and vice-versa. This is accomplished by using a sftp client.

- **On windows**

If you have not done so yet, download an sftp client from the internet. Pstfp.exe is a good enough choice and is freely available at:

```
http://www.chiark.greenend.org.uk/
~sgtatham/putty/
download.html
```

After downloading, execute the file `psftp.exe`. You can now enter Orionte's IP address, your user name and password:

```
psftp: no hostname specified; use
"open host.name" to connect

psftp> open tpereira@193.136.136.59

Using username "tpereira".

Using keyboard-interactive
authentication.

Password:

Remote working directory is
/home/tpereira
```

You are now logged in to Orionte and can start uploading and downloading files.

To copy from Orionte:

```
psftp> lcd c:\myLocalDir

New local directory is
c:\myLocalDir

psftp> cd myRemoteDir

Remote directory is now
/home/tpereira/myRemoteDir

psftp> get myRemoteFile

remote:/home/tpereira/myRemoteDir/
myRemoteFile => local:myRemoteFile
```

To copy to Orionte, use:

```
psftp> put myLocalFile

local: myLocalFile =>
remote:/home/tpereira/myRemoteDir/
myLocalFile
```

- **On Linux**

Most Linux distributions have by default a sftp client. If yours does not, please install one (you can follow your distribution's documentation).

To upload and download files, just do:

```
shell > sftp tpereira@193.136.136.59

Connecting to 193.136.136.59...

Password:

sftp> lcd myLocalDir

sftp> put myFile

Uploading myFile to
/home/tpereira/myFile
```

On every sftp client, the user can, at any time, get help by typing:

```
psftp> help
...
```

## Paths

Important itens that the user should have in his PATH variable are:

```
/opt/mpich2-1.0.2/bin
/opt/sun-jdk-1.4.2.06/bin
/opt/intel/compiler80/bin
/usr/local/bin
/usr/bin
```

Important itens that the user should add to his CLASSPATH variable are:

```
.
/opt/opensourcephysics
/opt/mpiJava/lib/classes
```

This can be accomplished by appending the user startup script, `/$HOME/.bashrc`, as in the following example:

```
export PATH="$PATH:/opt/mpich2-
1.0.2/bin"
export
CLASSPATH=".:/opt/opensourcephysic
s:/opt/mpiJava/lib/classes"
```

## Physical composition

When running jobs in Orionte, please take into account that:

- there are 16 nodes, named: master, node1, ... node15,
- interconnection is provided by a 1Gbit star topology switch, so programs have equal bandwidth from any node to any node,
- only some nodes have storage, although storage is available through nfs, in all nodes:

  **1.** master node has 100 GBytes mounted at /home,

  **2.** node8 has 120 GBytes mounted at /scratch, available in all nodes through nfs.

- each node has 1GByte of RAM, except master which has 2GByte,
- the master node connects to the outside internet at 100MBit/s,
- each node has a real performance, measured by Linpack, of 4,5 Gflops,
- currently booted OS is Gentoo linux and 2.6.10 vanilla kernel.

## MPI

We describe here how to use version 2.0 of MPICH.

To run a parallel program, one must:

1. write the code
2. compile the code
3. launch mpi environment
4. launch the program

- **Writing C parallel programs**

An example C parallel program could be:

```
#include <stdio.h>
#include "mpi.h"
int main (int argc, char **argv) {
int buf[1], myRank, nProcs;
int tag;
MPI_Status *stat;
MPI_Init (&argc, &argv);
MPI_Comm_size (MPI_COMM_WORLD,
 &nProcs);
MPI_Comm_rank (MPI_COMM_WORLD,
 &myRank);
tag = 123;
buf[0] = 321;
printf ("My rank is %d of a
 total of %d processes.\n",
 myRank, nProcs);
if (myRank == 0) {
MPI_Recv (buf, 1, MPI_INT, 1,
 tag, MPI_COMM_WORLD, stat);
}
```

```
if (myRank == 1) {
MPI_Send (buf, 1, MPI_INT, 1,
 0, MPI_COMM_WORLD);
}
MPI_Finalize();
}
```

The important sections equal in all mpi parallel programs written in C are:

1. include mpi.h:

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

2. start mpi:

```
MPI_Init (&argc, &argv);
```

3. determine the id of current process:

```
MPI_Comm_rank(MPI_COMM_WORLD,
&myRank);
```

4. determine the total number of processes:

```
MPI_Comm_size(MPI_COMM_WORLD,
&nProcs);
```

5. actual code:

```
MPI_Send (...);
MPI_Recv (...);
MPI_Broadcast (...);
```

6. finalize mpi

```
MPI_Finalize();
```

- **Compiling C parallel programs**

The C compilation can be accomplished with:

```
shell > mpicc myProg.c -o
myProg.bin
```

This compiler includes flags and libraries needed to mpi functions.

- **Writing fortran parallel programs**

An example fortran parallel program could be:

```
Program Example1_1
implicit none
integer nProcs, ierr
include "mpif.h"
integer myRank, source, dest, tag,
 status(MPI_STATUS_SIZE)
real my_result
dest = 0
source = 1
tag = 123
call MPI_Init(ierr)
call MPI_Comm_rank(MPI_COMM_WORLD,
 myRank, ierr)
call MPI_Comm_size(MPI_COMM_WORLD,
 nProcs, ierr)
write(*,"('Process ',i1,' started
 of a total ',i1)") myRank, nProcs
if(myRank.eq.0) then
call MPI_Recv(my_result, 1,
 MPI_REAL, source, tag,
 MPI_COMM_WORLD, status, ierr)
write(*,"('Result was ',f10.6) ")
 my_result
endif
if(myRank.eq.1) then
my_result = 321.123
call MPI_Send(my_result, 1,
 MPI_REAL, dest, tag,
 MPI_COMM_WORLD, ierr)
endif
call MPI_Finalize(ierr)
stop
end
```

- **Compiling fortran parallel programs**

The fortran compilation can be accomplished with:

```
shell > mpif77 myProg.c -o
myProg.bin
```

This compiler includes flags and libraries needed to mpi functions.

- **Writing java parallel programs**

An example java parallel program, could be:

```
import mpi.*;
public class Poisson
{
 private int myRank, nProcs,
 dest, source, tag, root;
 private int[] cmd = new int[1];
 public Poisson(String[] args)
 throws MPIException
 {
  dest = 1;
  source = 0;
  tag = 123;
  root = 0;
  MPI.Init(args);
  nProcs=MPI.COMM_WORLD.Size();
  myRank=MPI.COMM_WORLD.Rank();
  if (myRank == 0)
  {
   try
   {
    MPI.COMM_WORLD.Send(cmd, 0, 1,
    MPI.INT, dest, tag);
   }
   catch(mpi.MPIException e){}
  }
  if (myRank == 1)
  {
   try
   {
    MPI.COMM_WORLD.Recv(cmd, 0, 1,
    MPI.INT, source, MPI.ANY_TAG);
   }
   catch(mpi.MPIException e){}
  }
  try
  {
   MPI.COMM_WORLD.Bcast(cmd, 0, 1,
   MPI.INT, root);
  }
  catch(mpi.MPIException e) {}
  try
  {
   MPI.Finalize();
  }
  catch(mpi.MPIException e) {}
 }
}
```

Important sections, in a java parallel program, are:

1. Import mpi classes:

```
import mpi.*;
```

2. Start mpi:

```
MPI.Init(args);
```

3. Determine the id of current process:

```
myRank = MPI.COMM_WORLD.Rank();
```

4. Determine the total number of processes:

```
nProcs = MPI.COMM_WORLD.Size();
```

5. Actual message passing:

```
MPI.COMM_WORLD.Send (...);
MPI.COMM_WORLD.Recv (...);
MPI.COMM_WORLD.Bcast (...);
```

6. Finalize mpi

```
MPI.Finalize();
```

The API can be viewed online at: http://www.hpjava.org/mpiJava/doc/api/.

- **Compiling java parallel programs**

Compilation of java parallel programs is done with:

```
shell > javac myProg.java
```

- **Launching the mpi environment**

To be able to run parallel programs, one must first launch the mpi environment. This consists of executing a daemon in each node, one wants to run the parallel program in.

The following procedure is required, only the first time the user uses mpi:

1. create a file named

```
/$HOME/.mpd.conf
```

2. write in this file the following line:

```
secretword=<something>
```

3. make this file accessible only to its user, with read and write permissions,

4. create a file named:

```
/$HOME/mpd.hosts
```

5. list, in this file, the hostnames of the nodes in which the user wants to launch the mpi environment:

```
node1
node2
node3
node4
```

```
...
node15
```

Exclude, from this list, the name of the master node.

Then, the user must proceed in the following manner to launch the mpi environment, in four nodes as an example:

```
shell > mpdboot -n 4
```

After having launched the mpi environment, the user can run parallel programs.

The list of hosts in which mpi environment is ready, can be viewed by typing:

```
shell > mpdtrace
```

- **Running the parallel program**

To run a C or fortran parallel program, proceed as follows:

```
shell > mpiexec -np 4 myProg
```

To run a java parallel program, proceed as follows:

```
shell > mpiexec -np 4 java
myJavaProg
```

- **Inspecting currently running parallel programs**

To inspect running programs proceed as follows:

```
shell > mpdlistjobs
```

The output will be, depending on what is running, something like the following:

```
jobid    = 3@oriente_35892
jobalias =
username = tpereira
host     = oriente
pid      = 19341
sid      = 19340
rank     = 0
pgm      = ./mpi_bcast_bench

jobid    = 3@oriente_35892
jobalias =
username = tpereira
host     = node7
pid      = 21226
sid      = 21225
rank     = 1
pgm      = ./mpi_bcast_bench
```

- **Killing a running parallel program**

To kill the previous running mpi program, do the following:

```
shell > mpdkilljob 3
```

- **Shutting down the mpi environment**

To shutdown the mpi environment, write:

```
shell > mpdallexit
```

## *Running programs with specific node allocation*

Sometimes, it may be of convenience to be able to run a simple program (not an MPI program) in a specific node. To do so, please proceed as follows, after having launched the MPI environment:

```
shell > mpiexec -host node15
~/myProg &
```

This will run the program `myProg` in `node15`.

## *Using the batch queue system*

Users are highly encouraged to use the batch queue when runnning programs.

The main differences between running a program through the batch system and in a traditional way are:

- automatic migration of processes to available nodes (nodes that are idle), according to a determined policy;
- ability for the user to logout from the cluster while jobs are running;

- both `stdout` and `stderr` of the program that the user is running are redirected to two text files created by the batch system.

To launch a program using the batch system, the user should do the folowing:

```
shell > qsub myScript
165.oriente.cfn.ist.utl.pt
```

The output line shows the number of the batch job, in this case 165.

And `myScript` is a text file like the following:

```
#!/bin/bash
/home/tpereira/tmp/myProg
```

Where `myProg` is an executable file previously compiled.

To inspect currently running jobs, the user can use the utility `qstat`:

```
shell > qstat
Job id Name User  Time Use S Queue
---------- --- -------- - - -----
169.oriente myS tpereira 0 Q batch
```

To get information about the state of all nodes, the user can use the utility `pbsnodes`:

```
shell > pbsnodes -a
oriente
 state = free
 np = 1
 ntype = cluster
 status = ...
node1
 state = job-exclusive
 np = 1
 ntype = cluster
 status = ...
...
node15
 state = down
 np = 1
 ntype = cluster
 status = ...
```

After the job has run, the user can see its output in the file named `myScript.o165`:

```
shell >  cat myScript.o165
My output. This output was
generated by myProg.
```

## *Using graphic console on windows*

If you are accessing Oriente through windows and need to use a graphic console, please proceed as follows:

1. Log on to Oriente;
2. Set a password for your vncserver:

```
shell > vncpasswd
Using password file
/home/tpereira/.vnc/passwd
Password:
Verify:
Would you like to enter a view-
only password (y/n)? n
```

3. Execute a vncserver:

```
shell > vncserver
New 'X' desktop is oriente:3
...
```

5. Note the number of the display, the vncserver is working on (in this case 3);
6. Go back to your windows desktop and download the vnc software from: www.realvnc.com;
7. Launch the vncviewer;
8. Enter the server name in the following manner:

```
oriente.cfn.ist.utl.pt:3
```

where 3 is the display number noted from step 4.

## *IDL*

IDL v. 6.1 (Interactive Data Language) from Research Systems is currently installed on Orionte under `/opt/rsi` and can be started via the `idl` command or the `idlde` for the Development Enviroment.

### Setup

Before using IDL the user should source one of the idl setup scripts located under `/opt/rsi/idl/bin/` or, even better, place the following commands in one of the shell initialization files located at the user home directory.

1. *bash* users should place:

```
/opt/rsi/idl_6.1/bin/idl_setup.bash
```

   in `~/.profile`

2. *ksh* users must include:

```
/opt/rsi/idl_6.1/bin/idl_setup.ksh
```

   in `~/.profile`

3. and *csh* users should add the following command:

```
source /opt/rsi/idl_6.1/bin/idl_setup
```

   in `~/.cshrc`

- **Graphical enviroment**

To use the Development Enviroment or produce any on-screen graphical output from IDL, the user should start an X windows server on the client side (e.g. Exceed on Windows) and use X tunneling when connecting to orionte through ssh. Alternatively, you can start a VNCserver on orionte, as explained before (section “Using graphic console on windows”), which can be accessed using a compatible VNC client on the user side.

- **Using IDL**

1. Start idl

```
shell> idl
```

2. and type commands like:

```
IDL> data =
sin(((findgen(62)/10)^2))

IDL> plot, data

IDL> exit
```

3. or run the interactive demo program

```
IDL> demo
```

4. if you prefer to use the development environment type:

```
shell> idlde
```

- **Documentation**

From the IDL command prompt you can type a question mark "?" to obtain online help. Within the Development Environment point to Help menu on the top right of the window. For additional information visit the IDL website at http://www.rsinc.com/idl/ .

## *Matlab*

Matlab R13, including Matlab 6.5, Simulink 5.2, Optimization Toolbox, Signal Processing Toolbox, and Wavelet Toolbox is available in your default path.

- **Setup**

Matlab runs without any special configuration, however, a user can create a file `.matlab` under his home directory with additional personal preferences.

- **Using Matlab noninteractively**

To run Matlab noninteractively just type:

```
shell> matlab -nodisplay
<matlabscript.m
```

- **Using Matlab interactively**

To run Matlab interactively, proceed in one of the following manners:

1. Without graphical support just type:

```
shell> matlab -nodisplay
```

2. If graphical support is needed

   To be able to interact graphically with `matlab`, the user should start a X windows server on the client side (e.g. Exceed on Windows) and use X tunneling when connecting to orionte through ssh. Alternatively, you can start a VNCserver on `oriionte`, as explained before (section "Using graphic console on windows"), which can be accessed using a compatible VNC client on the user side.

   To start matlab just type:

```
shell> matlab
```

- **Documentation**

Type `help` to get online help when using `matlab` from the command prompt or `helpwin` if a graphical environment is active. For additional information on Matlab visit Mathworks website at http://www.mathworks.com.

### *List of available software*

- Java SDK 1.4.2_06 (Sun Microsystems)
- Gnu CC 3.4.3 – c and c++ gnu compiler
- ICC e IFORT – Intel compilers
- MPICH 2-1.0.2 – MPI Chameleon distro
- MPICH 1.1 – MPI Chameleon distro
- mpiJava – MPI ported to java
- OpenSourcePhysics – java library for physicists
- MatLab 6.5.0.180913a Release 13
- Math Kernel Library 7.2.1 for Linux – Intel linear algebra library
- BLAS e ATLAS – Linear algebra libraries
- FFTW – Parallel fast fourier transform library
- GSL (gnu scientific library) – scientific library
- gnuplot – graphic program
- emacs – text editor

## 2 Production

In this section, we discuss how to build Orionte, regarding mechanical and cooling details.

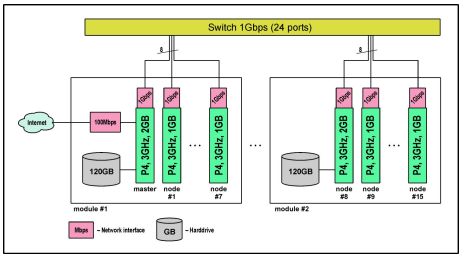
### *Components assembling*

In building Orionte, four objectives are taken into account:

- low cost,
- scalability,
- performance,
- compacting.

Thus, we have decided to use normal PC motherboards with Pentium 4 processors and Gigabit ethernet.

**Logical design**




Orionte is build by modules of eight nodes, each module having:

| | |
|---|---|
| 8 CPU | Intel P4 3GHz FSB 800MHz |
| Memory | 9 GByte (8 x 2 x 512Mb + 1GByte) |
| 8 network adapters | 1Gbit (onboard) |
| 8 motherboards | Intel D865GLC |
| 1 hardrive | Western Digital 120 Gb |
| 8 power sources | Sparkle Power FSP200-50PL |
| Water pump | 1000 l/h |

This arrangement comes from the dimensions of the 6U rack we use: 8 boards plus one harddrive fit in one rack.

The first node on the first module has an additional network interface, for access from outside and the first node in each module has an additionally 1GB RAM, for these nodes will have inevitably more usage.

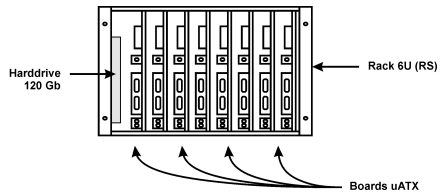
Every other node contains:

| | |
|---|---|
| 1 processor | P4, 3 GHz FSB 800MHz |
| Memory | 1 GB DDR (2x512Mb) |
| Network adapter | 1Gbit (onboard) |

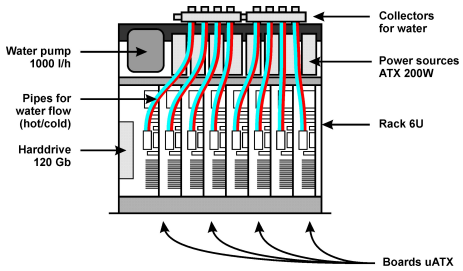
Additionally, it is needed:

| | |
|---|---|
| Cooling | Water chiller (~1000 Watts/module) |
| Network | Switch USRobotics 1 Gbit, 24 ports (sufficient for hree modules) |

**Schematic (front)**




**Schematic (top)**

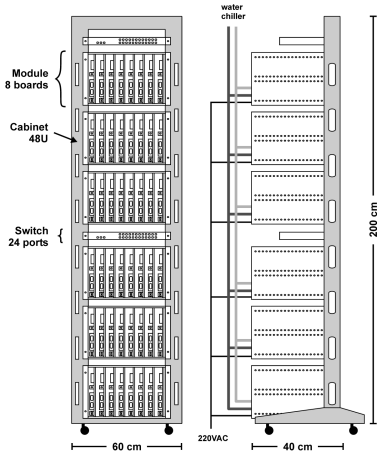



**Front**

**Top**

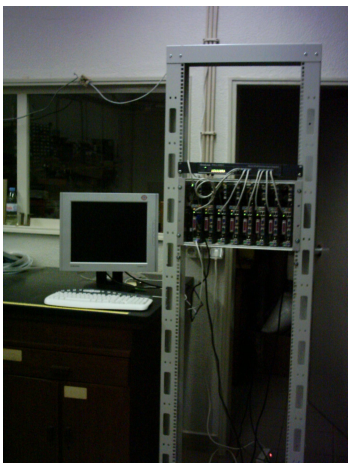
**Back**

**Schematic design of 48U cabinet**




The 6U racks can be accommodated in a 48U cabinet, for future scaling

**48U cabinet containing one module and one administration terminal**



## Choosing components

When choosing components, one must consider:

- performance,
- cost,
- assembling problems.

About the motherboard, we have:

- the factor form, for it must fit in the 6U rack,
- the processor to use,
- 1 Gbit network adapter onboard is a must.

The power sources must:

- be able to deliver enough power for one motherboard and one hardrive. Most power sources do the job,
- be small enough to fit in the rack. Most power sources aren't this small.

The water pump must also be small enough to fit in the rack.

## Cooling

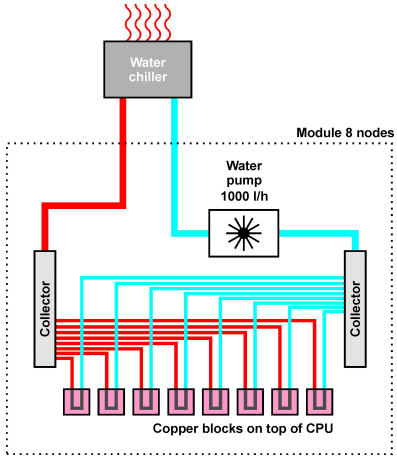
The cooling system for the processors was design with the following objectives in mind:

- efficiency,
- scalability,
- compacting.

The adopted system was:

- a thin copper block for each CPU,
- compressed air pipes for carrying water,
- 1000 l/h pump in each module,
- one water chiller for the whole cluster.

**Complete cooling system**

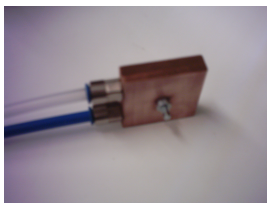


A mixture of distilled water with 5~10% of ethylene glycol is used as cooling fluid. The use of such a mixture is absolutely necessary to avoid the growing of seaweed. Other mixtures can be used, but one should avoid at all cost products that react with metals (like bleach). Car radiator cooling fluid is also a good choice.

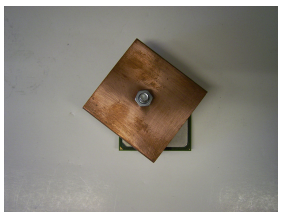
- **Copper blocks**

The copper blocks were designed to be small and efficient.

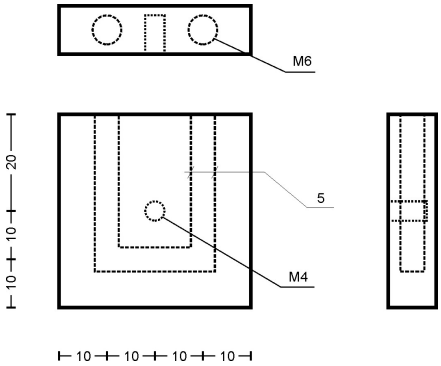
**Copper block**



**Copper block on top of processor**



**Copper block – technical design**

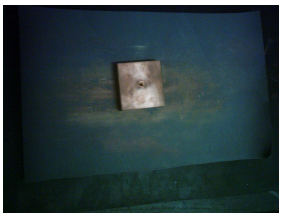



The production of these blocks has 7 phases:

1. cutting of the copper pole, in square pieces with 40 x 40 mm,
2. drilling of three 5 mm holes for water flow,
3. opening of two M6 threads,
4. closing of one hole by welding of brass pellet,
5. drilling of 3,5 mm hole for fixating the block,
6. opening M4 thread,
7. polishing face for contact with CPU with sand paper number 1200.

**Drilling of 5 mm hole in copper block with drilling tool**

**Polishing of a copper block with sandpaper**



Total time of production has been evaluated in 0,5 hours per block.

The thermal resistance of the block has been calculated from the following data:

- cpu power at idle: 51W,
- cpu power at full load: 105W,
- cpu temperature difference between idle and full load states: 13 ºC.

$$\Delta T = R_{th} \cdot P \Leftrightarrow$$

$$\Leftrightarrow \begin{cases} T = 51 \cdot R_{th} \\ T + 13 = 105 \cdot R_{th} \end{cases} \Leftrightarrow$$

$$\Leftrightarrow \quad R_{th} = \frac{13}{54} \quad \Leftrightarrow$$

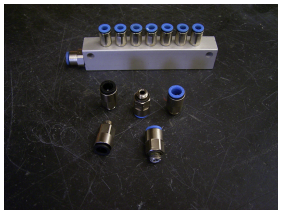
$$\Leftrightarrow \quad R_{th} = 0{,}24\ K\ J^{-1}$$

- **Pipes**

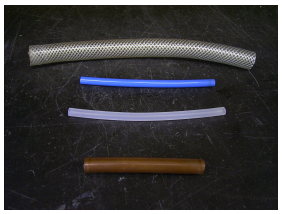
By using compressed air accessories, we have achieved a cheap, yet very reliable water flowing system.

**Raccors for pipe connection**

**Collector**



**Pipes 4 and 13 mm**



These pipes are wide enough for our needs of water flow and very malleable, which is very convenient.

- **Water pump**

Aquarium pumps are made for 24 hour operation and extremely silent. Thus, this is ideal for a cluster.

One has to keep in mind, that although the manufacturer says the pump has a flow of 1000 l/h, the effective flow of this pump will be significantly smaller when all the water pipes are assembled, and only measurable by means of a flow meter.

In our case, the effective flow is 220 l/h.

**Aquarium water pump**

- **Water chiller**

Although an expensive piece of equipment, the cost of the water chiller can be divided by a large number of nodes, seen as commercial available air conditioning water chillers easily reach 10 KW of cooling power. This is enough cooling power for one hundred processors at current processor power dissipation.

This equipment also has the advantage of not depending on atmospheric conditions.

**Water chiller for cooling of 16 nodes**

- **Sensors**

The first module is equipped with a flow meter and two temperature sensors.

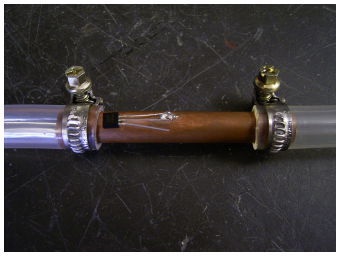
| | |
|---|---|
| Flow meter | RS 256-225 |
| Temperature sensor | LM35 |

One temperature sensor is at the cold water inlet and the other is at the hot water outlet (from the point of view of the module).

The idea of equipping one module with these sensors is to monitor hardware in a direct way. Seen as both modules are similar, there is no point in mounting sensors in both.

The LM35 is an IC in a TO-92 package. One pin of this IC has been welded to a piece of 10 mm copper pipe.

**Temperature sensor**



**Flow meter**

## *Thermal and electrical power*

The water suffers a raise in temperature of 3,3ºC, between the inlet and outlet of one module, in a situation of full load. And the flow is 220 l/hour. So, we have:

$$P = flow \times dif_{temp} \times Cv_{water}$$
$$= 0{,}061\, l\, s^{-1} \times 3{,}3\, K \times 4180\, J\, K^{-1}\, l^{-1}$$
$$= 841\, W$$

This is the power dissipated by the water chiller to the atmosphere, if it is mounted outside, or to the room if not.

In idle situation, we have 1,6ºC between cold and hot water and the same flow:

$$P = flow \times dif_{temp} \times Cv_{water}$$
$$= 0{,}061\, l\, s^{-1} \times 1{,}6\, K \times 4180\, J\, K^{-1}\, l^{-1}$$
$$= 408\, W$$

For one module, the electricity consumption is:

**Electrical power consumption for different sets of nodes**

| State | Current AC | Power |
|---|---|---|
| idle | 2,33 A | 505 W |
| 1 x 100% | 2,62A | 568 W |
| 2 x 100% | 2,92A | 634 W |
| 3 x 100% | 3,30A | 716 W |
| 4 x 100% | 3,55A | 770 W |
| 5 x 100% | 3,78A | 820 W |
| 6 x 100% | 4,07A | 883 W |
| 7 x 100% | 4,40A | 955 W |
| 8 x 100% | 4,66A | 1011 W |

Joining all data in one table, we can see how much heat is dissipated inside the room, and how much by the chiller:

**Electrical power consumption and heat dissipation for one module**

| State | Electrical power consump. | Hest dissipated by chiller | Heat dissipated by other comp. |
|---|---|---|---|
| Idle | 505 W | 408 W | 97 W |
| 100% | 1011 W | 841 W | 150 W |

### *Hardware shopping list (one module)*

- rack
  order from RS - code 500-412
- rails
  order from RS - code 259-7376
- harddrive 120 Gbytes SATA
- 8 motherboards ATX
- 8 processors
- low-profile PCI network adapter: realtek 8029
- 8 power sources ATX, Sparkle Power, Model no.: FSP200-50PL
- ~10 m compressed air pipe 4 x 6 mm, Automair, Rua Marquês de Soveral, Lisbon.
- ~0,5m copper pole 40 x 10 mm Rua da Bobadela, Carnaxide
- compressed air colectors, Automair, Rua Marquês de Soveral, Lisbon.
- raccors 6 mm, Automair, Rua Marquês de Soveral, Lisbon.
- raccors 12 mm, Automair, Rua Marquês de Soveral, Lisbon.
- water pump ~1000 l / h, aquarium store
- reinforced 12 mm hose, hardware store
- 2 water tap 3/8'', Automair, Rua Marquês de Soveral, Lisbon.
- waterchiller, lookup "Air conditioning"
- ~10 cramps 15 mm, hardware store
- ~20 screws M4 - 30 mm, hardware store
- ~8 screws M4 - 30 mm, hardware store
- aluminium pole, U profile, 13 x 8 x 8 mm, hardware store
- teflon, hardware store
- switch, 8 ports, 1Gbit
- copper pipe 10 mm, hardware store

## 3 Administrator software

In this section, we discuss installation of software necessary to cluster operation:

- linux configuration,
- remote boot,
- other cluster utilities.

It is aimed to the system administrator.

### *Linux*

We have currently installed in Orionte two kernel versions:

- 2.4.28
- 2.6.10

Version 2.4.28 is patched with OpenMosix. Version 2.6.10 is strictly vanilla and currently booted and preferred.

Regarding the configuration of the kernel, one must take into account:

- Processor type and features → Processor family → Pentium-4/Celeron(P4-based)/Pentium-4 M/Xeon
- Processor type and features → High Memory Support → 4GB
- Bus options (PCI, PCMCIA, EISA, MCA, ISA) → PCI support
- Bus options (PCI, PCMCIA, EISA, MCA, ISA) → ISA support
- Device Drivers → Block devices → Loopback device support
- Device Drivers → Block devices → Network block device support
- Device Drivers → Block devices → RAM disk support

- Device Drivers → Block devices → Default RAM disk size (Kbytes) → 65536
- Device Drivers → Block devices → Initial RAM disk (initrd) support
- Device Drivers → SCSI device support → SCSI disk support
- Device Drivers → SCSI device support → SCSI generic support
- Device Drivers → I2O device support → I2O /proc support
- Device Drivers → Networking support → Network device support → Ethernet (1000 Mbit) → Intel(R) PRO/1000 Gigabit Ethernet support
- Device Drivers → Networking support → Networking options → TCP/IP networking (INET) → IP: kernel level autoconfiguration (IP_PNP) → IP: DHCP support
- Device Drivers → Networking support → Networking options → Packet socket
- Device Drivers → I2C support → I2C device interface
- Device Drivers → I2C support → I2C Hardware Bus support → Intel 801
- Device Drivers → I2C support → I2C Hardware Bus support → Intel 810/815
- Device Drivers → I2C support → I2C Hardware Bus support → ISA Bus support
- Device Drivers → I2C support → Hardware Sensors Chip support → National Semiconductor LMxx
- File systems → Second extended fs support
- File systems → Ext3 journaling file system support
- File systems → Reiserfs support
- File systems → Kernel automounter version 4 support
- File systems → Pseudo filesystems → /proc file system support
- File systems → Pseudo filesystems → /dev file system support
- File systems → Pseudo filesystems → Virtual memory file system support (former shm fs)
- File systems → Network File Systems → NFS file system support
- File systems → Network File Systems → NFS server support
- File systems → Network File Systems → Root file system on NFS
- File systems → Native Language Support → Codepage 860 (Portuguese)

The kernel is the same for the master node and for the slaves and modules have been avoided. What this solution lacks in optimization, it gains in simplicity.

The linux distro is Gentoo. It was the chosen distro because of its robustness and portage system.

## *Openmosix*

Openmosix is an automatic process migration extension to the Linux kernel. It can be installed by patching the kernel vanilla sources with the openmosix sources found at http://openmosix. .sourceforge.net.

The OpenMosix project doesn't yet, have support for the 2.6.10 kernel.

Our experience with Openmosix tells us that:

- it is quite useful for users with programs of trivial parallelization (embarrassingly parallel),

- some crashes have occurred, especially with programs that create a lot of child processes, with short duration (example: make).

Because of this problem, openmosix is currently not available.

## *Remote boot*

The master node must boot before all other nodes. After master is fully booted, the following sequence takes place:

- slave is turned on,
- slave broadcasts dhcp request for IP, next-server IP and bootloader filename,
- master responds to slave request,
- slave requests bootloader file from tftp server in next-server IP,
- master sends bootloader file,
- slave executes bootloader file,
- bootloader downloads, from tftp server in next-server IP, kernel file and ramdisk compressed file,
- bootloader executes kernel and hands it over control,
- kernel uncompresses ramdisk, mounts ramdisk and configures system according do system configuration files in ramdisk,
- slave is ready.

This arrangement has two advantages:

- reduces cost by reducing necessary number of harddrives,
- avoids the problem of synchronizing the contents of a large number of disks.

And has one disadvantage:

- it provides reduced storage capability.

- **Motherboard bios configuration**

These are the guidelines for motherboard configuration:

Master:

- boot from harddrive,
- when power fails, motherboard should resume to “last state”,
- wake on net “magic packet” event,
- doesn’t need IDE interface, USB, floppy, graphic interface memory.

Slave:

- boot from network,
- when power fails, motherboard should resume to “off state”,
- wake on net “magic packet” event,
- doesn’t need IDE interface, USB, floppy, graphic interface memory.

- **Dhcp server**

The `dhcp` server is:

```
/usr/sbin/dhcpd
```

Configuration file is:

```
/etc/dhcp/dhcpd.conf
```

Documentation is available with:

```
shell > man dhcpd
shell > man dhcpd.conf
```

The configuration file must have one hardware address to IP address mapping for each node. Additionally, it must have the path of the bootloader file and the IP address of the tftp server.

- **Tftp server**

The `tftp` server is:

```
/usr/sbin/in.tftpd
```

Configuration file is:

```
/etc/conf.d/in.tftpd
```

Documentation is available with:

```
shell > man in.tftpd
```

- **Pxelinux**

This is the bootloader. It is configured with one file per node. These files are in the directory:

```
/tftpboot/pxelinux.cfg/
```

The bootloader file itself is:

```
/tftpboot/pxelinux.0
```

The `dhcp` and `tftp` servers must point to this file.

PxeLinux is free for download at www.kernel.org/pub/linux/utils/boot/syslinux/syslinux-3.07.tar.gz and documentation is available at the same site.

- **Nfs server**

This service is provided by the master, so that other nodes can have persistent storage.

The following files implement this server:

```
/usr/sbin/exportfs
/sbin/rpc.statd
/usr/sbin/rpc.rquotad
/usr/sbin/rpc.nfsd
/usr/sbin/rpc.mountd
```

The configuration file is:

```
/etc/exports
```

This file must contain a line for each exported directory.

Documentation is available at:

```
shell > man nfsd
shell > man exports
```

- **Ramdisks**

An argument with the path of the ramdisk file is passed to the kernel when each node boots, in the following manner:

```
kernel vmlinuz-2.6.10
append root=/dev/ram0 rw
  load_ramdisk=1
  ramdisk_size=65536
  initrd=ramdiskFS_node1.gz
```

This is an excerpt of a configuration file for PxeLinux.

Each node has to types of storage:

- volatile (ramdisk),
- non-volatile (nfs mounted directories).

The ramdisk contains the following:

```
/bin
/etc
/lib
/mnt
/sbin
/var
```

The following directories are, due to their large size, not included in the ramdisk, but instead nfs mounted by the nodes:

```
/tmp
/opt
/usr
/home
/var/log
/root
```

The contents of each ramdisk (one for each node) are located in the directories:

```
/tftpboot/ramdisks_uncompressed/no
deXX
```

where nodeXX is the node's hostname this ramdisk belongs to.

And these contents are compiled by the following script:

```
#!/bin/bash
if [ "$1" = "" ]; then
  echo"/tftpboot/mk_ramdisk<#>"
  echo "Onde <#> é o número
    do nó"
  echo "Especifique <#>"
fi
TARGET="/tftpboot/ramdiskFS_nod
  e$1"
```

```
dd if=/dev/zero of=/dev/ram0
  bs=1k count=64k

mke2fs /dev/ram0 -b 1024 65536

mount /dev/ram0 /mnt/ramdisk

cp -a
  /tftpboot/ramdisks_uncompres
  sed/node$1/* /mnt/ramdisk/

umount /dev/ram0

if test -f $TARGET.gz
  then rm $TARGET.gz
fi

dd if=/dev/ram0 of=$TARGET
  bs=1k count=64k

gzip "$TARGET"
```

The idea of using ramdisks has proven to be robust and efficient and gives the nodes a certain amount of independence from the master.

## Wakeonlan

This little program does the job of broadcasting a “magic packet” to the net.

Motherboards that are shut down, but listening, are woken up by this packet.

The selection of which motherboard one wants to wake up is made by the MAC address, in the following manner:

```
shell > /opt/wakeonlan-0.40/
  wakeonlan -i 192.168.100.255
  00:11:11:48:f5:ab

Sending magic packet to
192.168.100.255:9 with
00:11:11:48:f5:ab
```

## Ntp (network time protocol)

This protocol accomplishes the job of synchronizing the time of every node.

We have OpenNtpd-2.6.1 implementation installed.

The executable is:

```
/usr/sbin/ntpd
```

which works as server and/or client, depending on the configuration.

The configuration files are:

```
/etc/ntpd.conf

/etc/conf.d/ntpd
```

On the master node, ntpd acts as a client for an external ntp server (currently 137.120.89.224) and as a server for the other nodes.

Documentation can be found with:

```
shell > man ntpd

shell > man ntpd.conf
```

## Monitoring system temperatures

The CPU and motherboard temperatures can be monitored with the following program:

```
/usr/bin/sensors
```

Configured in:

```
/etc/sensors.conf
```

For this program to work, the kernel must provide support for I2C bus and for the sensors that the motherboard possesses, which vary from board to board.

Documentation about this program can be found at:

```
shell > man sensors

shell > man sensors.conf
```

## Adding a new user to the cluster

Adding a new user to the cluster involves 7 steps:

1. Add user in master node

```
shell > useradd -d /home/manuel
-m -G users manuel
```

2. Define his password:

```
shell > passwd manuel
```

```
New UNIX password:
Retype new UNIX password:
```

3. Define which groups he belongs to:

```
shell > usermod -G wheel manuel
```

4. Copy the following files

```
/etc/passwd
/etc/group
/etc/shadow
```

   to all directories that contain the ramdisks of all the nodes,

5. Compile the ramdisks,

6. Reboot all nodes.

## Providing free ssh traffic inside the cluster

To allow free ssh inside the cluster, two things are necessary:

1. that the user public key be in the file:

```
/$HOME/.ssh/authorized_keys
```

   and that this file be accessible with write and read permission to the user and only with read permission to others,

2. that the user private key be in the file:

```
/$HOME/.ssh/id_rsa
```

   and that this file is only accessible by the user, with read and write permissions.

For this, proceed as follows:

```
shell > ssh-keygen -t rsa
Generating public/private rsa
  key pair.
Enter file in which to save the
  key
  (/home/tpereira/.ssh/id_rsa):
Enter passphrase (empty for no
  passphrase):
Enter same passphrase again:
Your identification has been
  saved in
  /home/tpereira/.ssh/id_rsa.
Your public key has been saved
  in
/home/tpereira/.ssh/id_rsa.pub.
The key fingerprint is:
  5a:cf:18:16:0d:63:84:6c:63:
  aa:48:45:25:d3:d7:9b
```

## Providing rsh

Rsh protocol provides the same as ssh protocol, but without encryption, thus, being a lot faster.

For rsh to be available to users, two things are needed:

- that the following daemons are running:

```
/usr/sbin/in.rexecd
/usr/sbin/in.rlogind
/usr/sbin/in.rshd
```

- that the user has in his home directory a file named:

```
.rhosts
```

  identifying trusted hosts.

The three daemons are run by the super-daemon:

```
/usr/sbin/xinetd
```

Configuration files are:

```
/etc/xinetd.d/rexec
/etc/xinetd.d/rlogin
/etc/xinetd.d/rsh
```

Documentation is available at:

```
shell > man xinetd
shell > man xinetd.conf
shell > man rshd
```

### Adding a new node to the cluster

Adding a new node to the cluster involves:

- creating a new ramdisk directory with all the necessary files:

```
shell > cd /tftpboot/
  /ramdisks_uncompressed
shell > mkdir nodeXX
shell > cd nodeXX
shell > cp -a ../node1/* ./
```

- editing of certain files that differ from node to node,
- compiling the newly created ramdisk:

```
shell > cd /tftpboot
shell > ./mk_ramdisk XX
```

- creating a `/tmp` directory, on the master node, for the new node:

```
shell > mkdir
  /tmp/remote_tmps/tmp_nodeXX
```

- editing the file:

```
/etc/exports
```

  on the master node.

The necessary changes in configuration files are:

- `/etc/hosts` (all nodes)

  Add new node's hostname and IP address,

- `/etc/hostname` (new node)

  Add new node's hostname,

- `/etc/conf.d/net.eth0` (new node)

  Define new node's IP address,

- `/etc/fstab` (new node)

  Define new node's nfs mount of `/tmp` directory,

- `/etc/exports` (master node)

  Add export for new node's `/tmp` directory.

### Installed software list

- Kernel linux (vanilla 2.6.10, openmosix patched 2.4.28)
- Linux Gentoo
- Java SDK 1.4.2_06 (Sun Microsystems)
- Gnu CC 3.4.3 – c and c++ gnu compiler
- ICC e IFORT – Intel compilers
- MPICH 2-1.0.2 – MPI Chameleon distro
- MPICH 1.1 – MPI Chameleon distro
- mpiJava – MPI ported to java
- OpenSourcePhysics – java library for physicists
- MatLab 6.5.0.180913a Release 13
- Wakeonlan-0.40 – remote wakeup
- PxeLinux – bootloader for remote boot
- Math Kernel Library 7.2.1 for Linux – Intel linear algebra library
- BLAS e ATLAS – Linear algebra libraries
- FFTW – Parallel fast fourier transform library
- GSL (gnu scientific library) – scientific library
- gnuplot – graphic program
- emacs – text editor

## 4 Developer notes

- The booting scheme should be altered so that not more than 7 nodes boot from one node. This would be: node 1 to node 7 booting from node master,

node 9 to node 15 booting from node 8, etc.

- The cluster could benefit from a remote hard reset feature, controlled by the master node through the serial port, for instance.

# 5 Specifications

- weight of one module: 16.5 Kg
- weight of cabinet (48 boards): 130 Kg
- dimensions of one module (8 boards): (L x H x D): 50 x 30 x 40 cm
- dimensions of one cabinet (48 boards): (L x H x D): 60 x 200 x 40 cm
- storage per module (8 boards): 120 Gbytes
- electricity consumption (8 boards):

  400W in idle,

  840W in full load

- performance per node: 4,5 gflops