Trabalho (entrega – 30/10)

- Implementar dois programas para multiplicar duas matrizes, um sequencial e outro paralelo usando threads, e comparar o overhead de execução
- Na versão paralela, cada thread calcula os resultados para um subconjunto de dados e então os resultados são combinados

Multiplicação de Matrizes

```
double  a[n][n], b[n][n], c[n][n];
for (i=0; i < n; i++) {
    for (j = 0; j < n; j++ ) {
        c[i,j] = 0.0;
        for (k = 0; k < n; k++)
            c[i,j] += a[i,k] * b[k,j];
    }
}
```

Multiplicação de Matrizes

- O programa paralelo deve ser executado variando-se o número de threads de 1 ao número total de processadores (2, 4, 8, 16, 32, 64 e 80)
- Variar o tamanho da matriz (1.000.000x1.000.000; 10.000.000x10.000.000; 100.000.000x100.000.000; 1.000.000.000x1.000.000.000)
- Tirar uma média do tempo de execução (executar pelo menos três vezes cada experimento. Dica: usar a função *gettimeofday* para calcular o tempo de execução)

Multiplicação de Matrizes

- Entregar relatório com 2 tipos de gráficos:
  - Tempo de execução para os diversos tamanhos de matriz comparando o tempo execução ao aumentar o número de processadores;
  - ***Speedup*** (tempo exec. em 1 processador / tempo exec. em n procs.) variando o número de processadores. O gráfico de *speedup* é um gráfico de linha onde o eixo y contém o *speedup* e o eixo x contém o número de processadores.
- Entregar o código fonte também

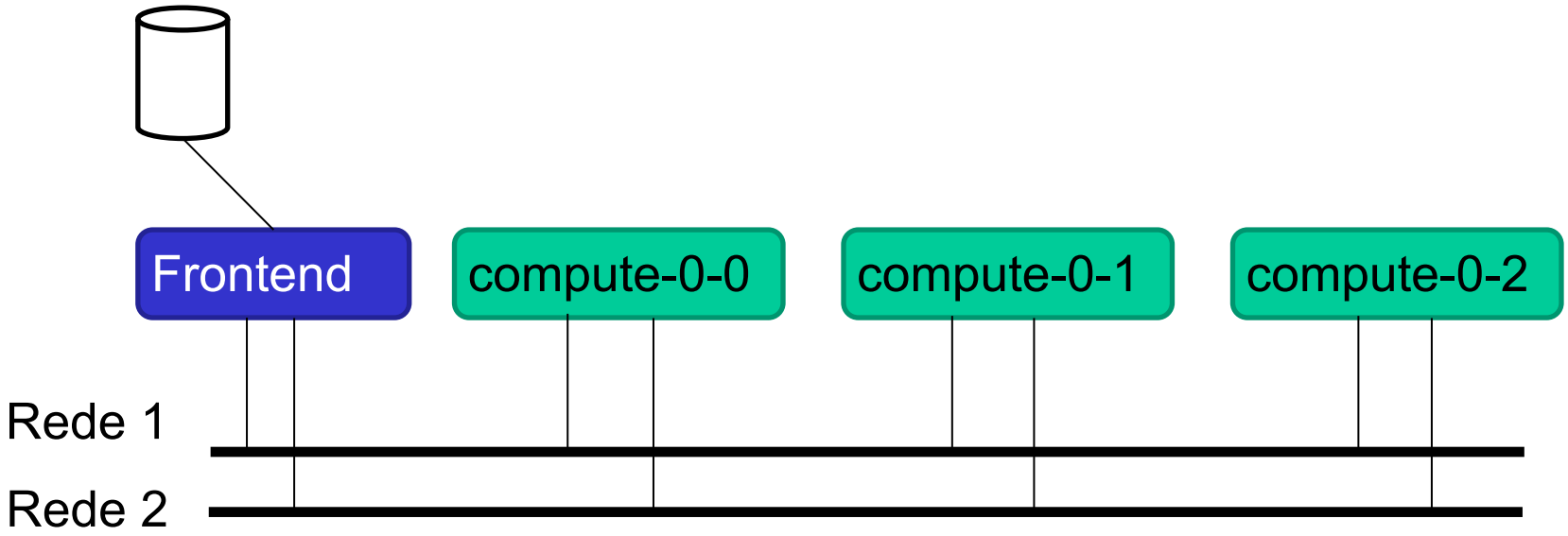
Acesso ao Cluster

- Logar no cluster via ssh:

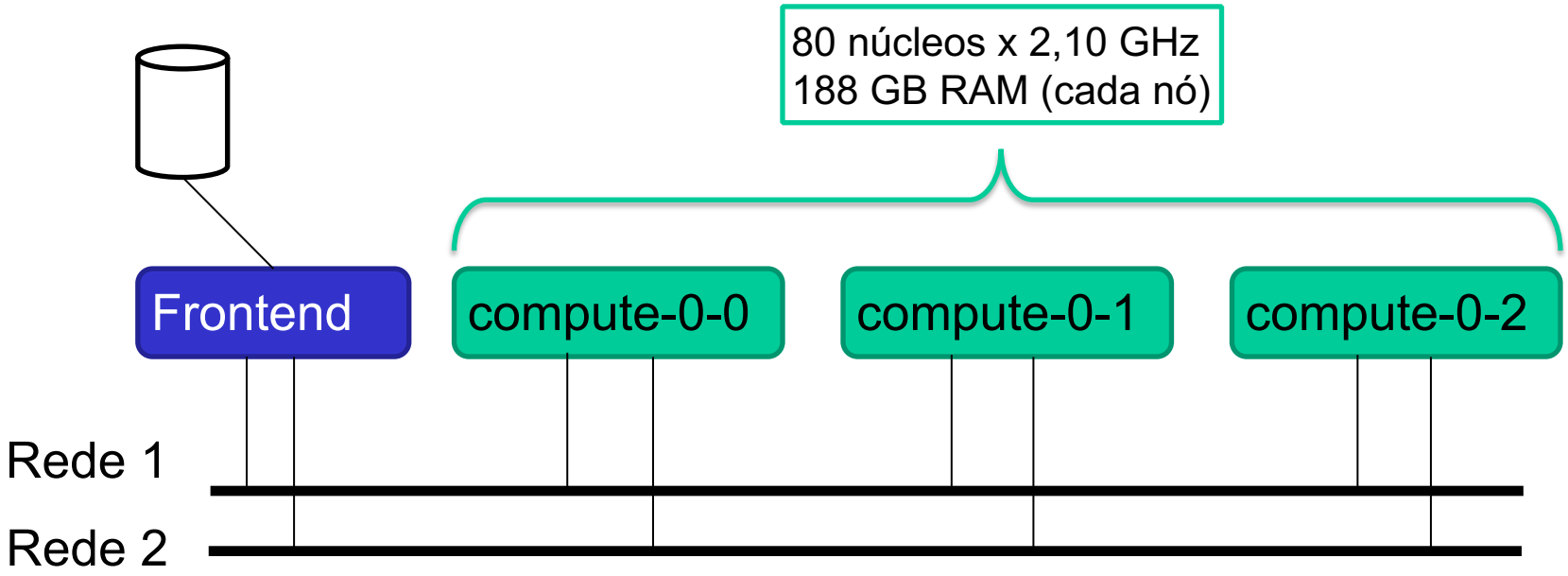
```
#ssh aluno@192.168.90.130
(senha se9@2022)
# mkdir seu_nome
```

- Visualizar no navegador informações sobre o cluster (192.168.90.130/ganglia)

Arquitetura do cluster (Rocks Cluster)



Arquitetura do cluster (Rocks Cluster)



Nosso Primeiro Programa

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

#define NUM_THREADS 40

void *PrintHello(void *threadid)
{
   long tid;
   tid = (long)threadid;
   char proc_nome[255];

   gethostname(proc_nome,255);
   printf("Hello World! It's me, thread #%ld (%s)!\n", tid,
proc_nome);
   pthread_exit(NULL);
}
```

Nosso Primeiro Programa

```
int main(int argc, char *argv[])
{
   pthread_t threads[NUM_THREADS];
   int rc;
   long t;
   for(t=0;t< NUM_THREADS;t++){
     rc = pthread_create(&threads[t], NULL, PrintHello,
(void *)t);
     if (rc){
       printf("ERROR; return code from pthread_create() is
%d\n", rc);
       exit(-1);
     }
   }

   pthread_exit(NULL);
}
```

Nosso Primeiro Programa

Copiar arquivo para o frontend do cluster estando logado em uma máquina local:

```
#scp hello.c aluno@192.168.90.130:/home/aluno/nome
```

Depois de se logar no frontend usar o comando abaixo para compilar:

```
#gcc -o hello hello.c -pthread
```

Acesso em *batch*

- O cluster usa o sistema em batch SGE (*Sun Grid Engine*) que usa os seguintes comandos:

Comando	**Descrição**
qsub	Submete o job script
qstat	Monitora as filas de job
qstat -f	qstat avançado
qstat -j *jobid*	Provê informação sobre um *job*
qdel *jobid*	Elimina o job da fila ou em execução

Exemplo de script (hello.qsub)

```
#! /bin/bash

#$ -cwd

#$ -j y

#$ -S /bin/bash


$HOME/nome-aluno/hello
```

Exemplo de script (hello.qsub)

```
#! /bin/bash
#$ -cwd
#$ -j y
#$ -S /bin/bash
```

Executar o job no diretório corrente

$HOME/nome-aluno/hello

Exemplo de script (hello.qsub)

```
#! /bin/bash
#$ -cwd
#$ -j y
#$ -S /bin/bash
```

Combina stdout com stderr

$HOME/nome-aluno/hello

Exemplo de script (hello.qsub)

```
#! /bin/bash
#$ -cwd
#$ -j y
#$ -S /bin/bash
```

Especifica o shell

$HOME/nome-aluno/hello

Submissão de job

- Para submeter um job usa-se o comando **qsub** da seguinte forma:

```
# qsub -pe smp 41 hello.qsub
```

- Neste exemplo “41" é o número de threads a serem criados
- Quando o job terminar, sua saída estará no arquivo **hello.qsub.o***