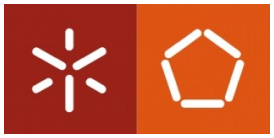
NP Benchmarks

Engenharia de Sistemas da Computação
Computação Paralela Distribuída

Mestrado em Engenharia Informática
Universidade do Minho



Duarte Nuno Ferreira Duarte
pg27715

Índice

Índice de imagens

Índice de tabelas

Índice de gráficos

Introdução

Com a realização deste trabalho pretende-se obter um ponto de comparação entre os vários nodos do *cluster*. As *benchmarks* são utilizadas tipicamente para estabelecer vários pontos de comparação entre diferentes sistemas, neste caso prende-se aplicar este conceito ao **search6**. Serão utilizados para comparação também diferentes compiladores, o compilador da Intel e duas versões de compiladores da GNU. Com este trabalho será possível por em prática vários conhecimentos adquiridos nesta unidade curricular ao longo deste semestre.

1. NAS Parallel Benchmarks

O NAS Parallel Benchmarks é um grupo de programas com o intuito de avaliar computadores, mas principalmente de avaliar máquinas paralelas, nomeadamente clusters. Os testes que podem ser utilizados são o BT, CG, DT, EP, FT, IS, LU, MG e SP. Destes serão utilizados apenas alguns. Cada um destes testes é composto por várias classes (A,B,C,D,E,F ou S,W) e à medida que a classe avança alfabeticamente maior é o teste. Estes testes estão englobados em três grupos principais, o grupo SERIAL, o MPI e ainda o OMP. Estes grupos principais estão diretamente referenciados para as versões a serem utilizadas, MPI no caso de memoria distribuída e OMP no caso de memoria partilhada.
Para estes testes foram apenas utilizados os seguintes *kernels*:

EP - *Embarrassingly Parallel;*
IS - *Integer Sort, random memory access;*
MG - *Multi-Grid on a sequence of meshes, long and short-distance communication, memory intensive.*

E ainda foi utilizada a seguinte aplicação:

LU - *Lower-Upper Gauss-Seidel solver*

Para cada um destes testes, quer fossem kernels ou aplicação, foram utilizadas três classes diferentes. A classe A, B e C: *standard test problems.*

2. Caracterização do Sistema

Para os testes feitos foi utilizado o nó 431-6 do Search, este nó apresenta a seguinte especificação:

Manufacturer **:** intel

Model **:** X5650

Architecture **:** x86-64

Clock speed **:** 2.67 GHz

Number of Cores : 12

Threads per core : 2

Total *threads* : 24

3. Definições Utilizadas

a. Compiladores

Na realização destes testes foram utilizados compiladores de duas entidades, a Intel e a GNU. Com duas versões no caso da GNU.
Compilador da Intel tem a versão 13.0.1 e os dois da GNU foram a versão 4.8.2 e a 4.9.0.
De todas as versões de compiladores disponíveis apenas não foi utilizada a versão mais antiga existente do GNU.

b. Versões

Foram utilizados para estes testes a versão Serial, que utiliza apenas um processador, a versão MPI que usa um determinado número de processos com memória distribuída. E por fim a versão do OMP que usa um determinado numero de *threads* com memoria partilhada.

c. Tamanhos

Foram utilizadas sempre três classes diferentes, a A, B e C. No caso do MPI foram testadas com 4, 8, 9 e 16 processos. No caso do OMP foram testadas com 4, 8 e 16 processos.

4. Benchmarks

a. Tempos

Os testes apresentado de seguida mostram o tempo de execução em segundos do *kernel* **EP**, **IS** e **MG** e ainda do programa **LU** para as classes A, B e C. Com as diferentes versões de compiladores e a ainda para diferente numero de *threads* e processos. Apenas para o *kernel* **MG** não existem testes de todos os tipos para a classe C porque é impossível compilar.

A seguinte legenda serve para serem perceptíveis os gráficos de tempo de execução apresentados mais adiante.

1	Serial
2	**MPI** com 1 processo
3	**MPI** com 4 processos
4	**MPI** com 8 processos
5	**MPI** com 9 processos
6	**MPI** com 16 processos
7	**MPI** com 32 processos
8	**OMP** com 1 *thread*
9	**OMP** com 4 *threads*
10	**OMP** com 8 *threads*
11	**OMP** com 16 *threads*

i. Tabelas de tempos dos testes

EP			MPI						OMP				MIN
COMP	class	Serial	1	4	8	9	16	32	1	4	8	16	
gcc4.8.2	A	15,04		5,12	2,59		1,13	0,59	19,62	4,92	2,14	1,32	0,59
gcc4.9.0		16,71		4,92	3,13		1,43	0,73	15,63	4,93	2,27	1,24	
icc		15		4,41	2,86		1,31	0,66	15,77	5,02	2,23	1,1	
gcc4.8.2	B	59,11		16,14	10,31		4,43	2,25	77,51	16,38	8,5	5,17	2,25
gcc4.9.0		61,22		16,59	10,56		5,55	2,77	63,15	16,41	8,5	4,63	
icc		59,17		17,64	10,39		5,02	2,58	62,36	16,44	8,72	5,08	
gcc4.8.2	C	240,62		65,87	41,35		17,66	8,89	245,74	66,21	34,41	18,35	8,89
gcc4.9.0		252,7		66,82	41,94		20,6	10,99	253,71	65,89	34,8	20,53	
icc		238,22		70,46	42,12		19,39	10,06	255,17	65,59	34,87	17,73	

Tabela 1 – Tempo de execução EP

IS			MPI						OMP				MIN
COMP	class	Serial	1	4	8	9	16	32	1	4	8	16	
gcc4.8.2	A	0,97		0,27	0,15		0,07	10,18	0,97	0,28	0,13	0,09	0,07
gcc4.9.0		0,82		0,3	0,16		0,11	0,83	0,83	0,28	0,14	0,09	
icc		0,77		0,24	0,16		0,08	7,77	0,84	0,27	0,13	0,09	
gcc4.8.2	B	3,22		0,94	0,62		0,32	53,05	4,18	1,1	0,58	0,4	0,3
gcc4.9.0		3,49		0,95	0,67		0,44	3,07	3,57	1,06	0,56	0,3	
icc		4,07		1,02	0,77		0,35	45,29	3,4	1	0,55	0,39	
gcc4.8.2	C	14,74		4,68	2,54		1,37	156,34	14,91	4,97	2,44	1,69	1,37
gcc4.9.0		19,74		4,3	2,98		1,9	12,33	15	5	2,47	1,73	
icc		14,37		4,27	2,69		1,48	151,01	16,09	4,42	2,44	1,72	

Tabela 2 – Tempo de execução IS

LU			MPI						OMP				MIN
COMP	class	Serial	1	4	8	9	16	32	1	4	8	16	
gcc4.8.2	A	38,01		14,91	9,33		3,72	22,31	51,2	30,98	29,98	30,8	3,72
gcc4.9.0		40,03		14,22	9,94		5,82	4,39	46,28	31,91	29,35	37,3	
icc		38,89		13,39	10,27		4,61	23,97	40,12	32,36	29,89	39,48	
gcc4.8.2	B	162,56		51,31	39,66		17,05	148,87	174,9	131,65	127,46	133,41	15,01
gcc4.9.0		181,18		62,73	43,29		23,91	15,01	186,59	134,36	123,91	156,64	
icc		183,58		57,7	43,01		21,61	108,95	181,3	137,58	121,7	132,19	
gcc4.8.2	C	680,3		254,34	160,43		117,23	790,93	713,43	533,18	466,16	518,75	58,14
gcc4.9.0		825,84		220,57	204,14		96,51	58,14	841,4	509,75	466,75	532,73	
icc		782,54		244,59	184,35		130,81	1113,47	805,63	521,65	482,08	544.93	

Tabela 3 – Tempo de execução LU

MG			MPI						OMP				MIN
COMP	class	Serial	1	4	8	9	16	32	1	4	8	16	
gcc4.8.2	A	1,7		0,55	0,43		0,16	2,97	1,96	0,46	0,27	0,28	0,16
gcc4.9.0		1,6		0,48	0,46		0,27	0,37	1,52	0,46	0,26	0,28	
icc		1,59		0,48	0,47		0,2	3,19	1,64	0,48	0,27	0,29	
gcc4.8.2	B	6,95		2,1	2		0,76	20,45	7,04	2,21	1,21	1,23	0,76
gcc4.9.0		7,5		2,5	2,21		1,25	1,67	7,47	2,16	1,29	1,16	
icc		7,69		2,23	2,21		0,95	16,13	7,55	2,21	1,27	1,12	
gcc4.8.2	C			16,91	15,46		6,84	107,6					6,84
gcc4.9.0				19,86	17,6		10,03	9,56					
icc				17,23	17,29		7,24	119,09					

Tabela 4 – Tempo de execução MG

ii. Gráficos de tempos dos testes

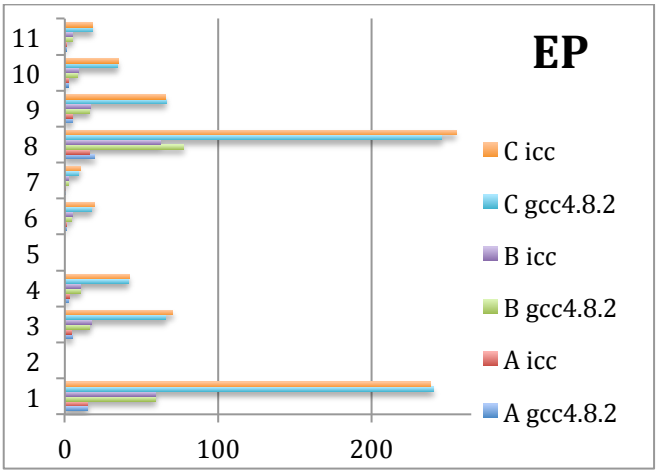


Gráfico 1 – Tempo de execução EP

Como é possível observar pelo gráfico acima (Gráfico 1) os tempos obtidos com o compilador da Intel e da GNU são muito semelhantes mas neste teste o compilador da GNU é ligeiramente melhor. Neste caso também se pode ver que o teste C é mais lento que o teste B que por sua vez é mais lento do que o teste A.

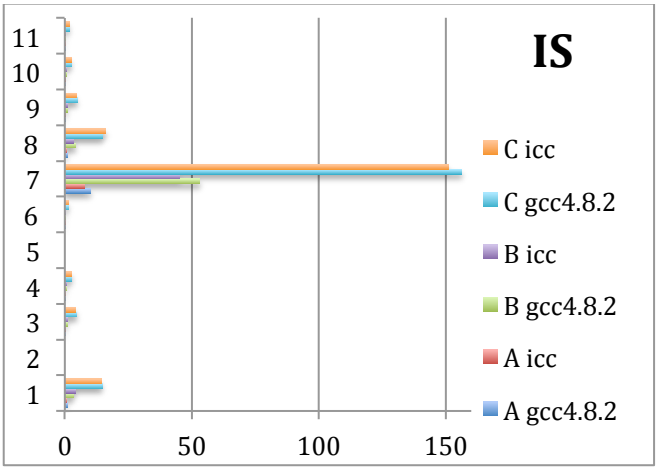


Gráfico 2 – Tempo de execução IS

Neste teste, o **IS**, o compilador da Intel é melhor, obtendo tempos melhores apesar de muito semelhantes aos da GNU. Neste caso também se pode ver que o teste C é mais lento que o teste B que por sua vez é mais lento do que o teste A.

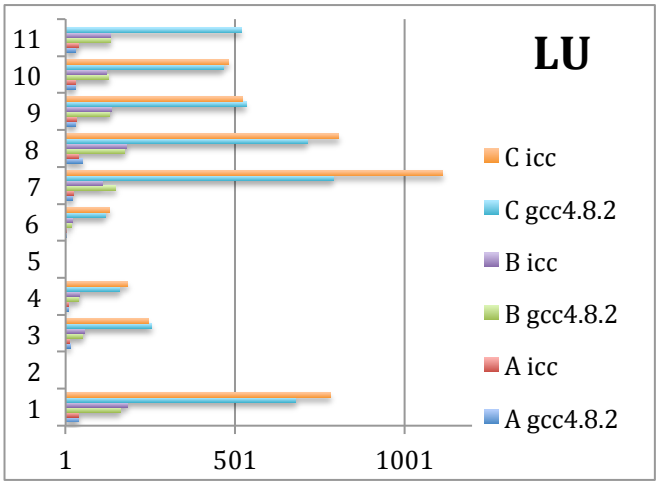


Gráfico 3 – Tempo de execução LU

No teste do programa **LU** os tempos obtidos com o compilador da GNU são muito melhores do que os obtidos com o compilador da Intel. E uma vez mais se pode ver que o teste C é mais lento que o teste B que por sua vez é mais lento do que o teste A.

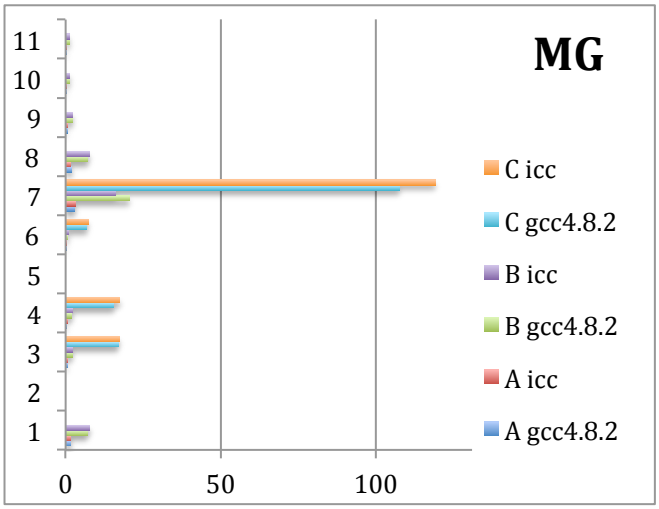


Gráfico 4 – Tempo de execução MG

No ultimo teste efectuado, o **MG**, o compilador da GNU volta a obter melhores tempos apesar de muito semelhantes aos da Intel. Neste teste apenas se podem observar tempos na classe C para os testes do MPI.

✓ Observando pelos valores obtidos é possível tirar algumas conclusões preliminares. Em quase todos os testes a versão que usa o compilador da GNU é melhor. Em termos de desempenho global por classe a que geralmente

obtém melhores tempos é a que usa MPI com 16 processos. Excepto no teste **EP** que com 32 processos é melhor.

b. Memoria

Para os gráficos desta secção a legenda utilizada é a seguinte:

i. Teste EP

Classe A

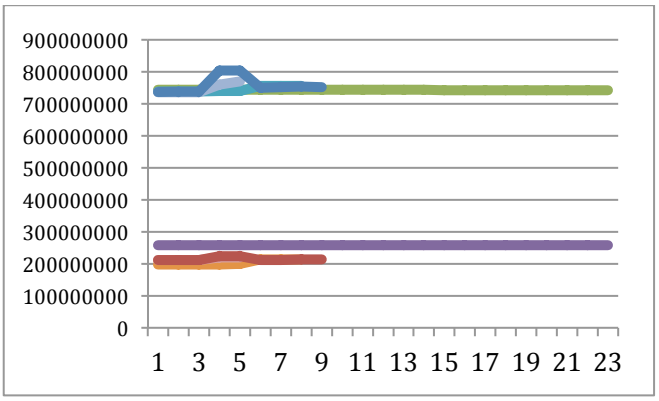


Gráfico 5 – Memória utilizada EP - A

Classe B

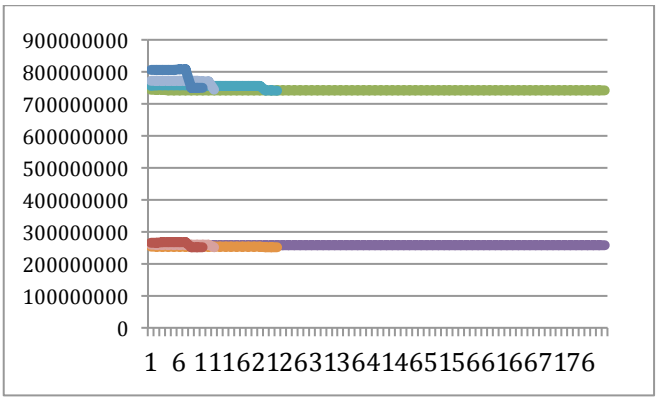


Gráfico 6 – Memória utilizada EP - B

Classe C

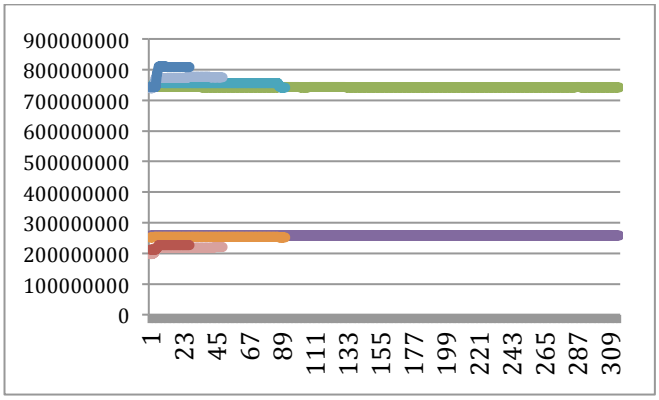


Gráfico 7 – Memória utilizada EP – C

Nos testes da classe EP com a versão serial é possível ver que a memoria utilizada pelo programa mantém-se constante ao longo de toda a execução. No caso das versões que utilizam MPI existe um aumento proporcional ao numero de processos no inicio e depois torna-se constante.

Teste IS

Classe A

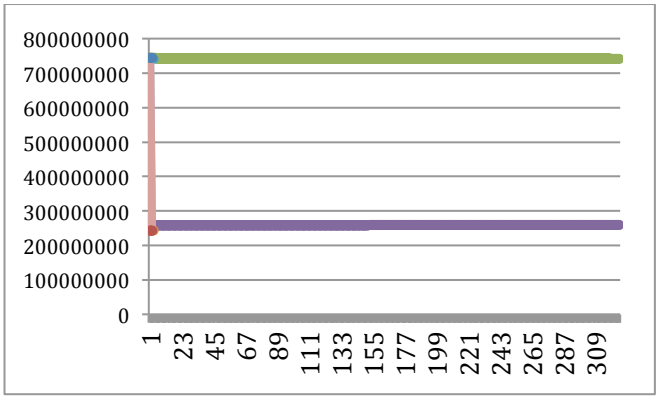


Gráfico 8 – Memória utilizada IS - A

Classe B

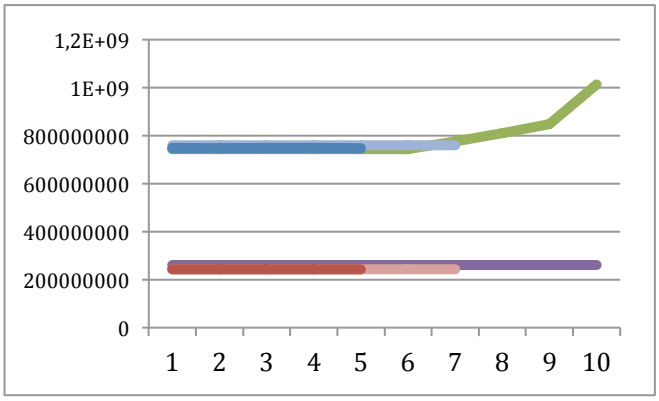


Gráfico 9 – Memória utilizada IS - B

Classe C

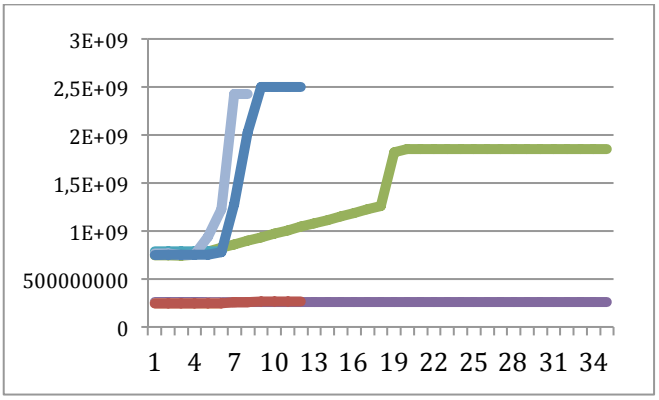


Gráfico 10 – Memória utilizada IS - C

Nos testes da classe IS com a versão serial é possível ver que a memoria utilizada pelo programa mantém-se constante ao longo de toda a execução. No caso das versões que utilizam MPI existe um aumento proporcional ao numero de processos no inicio e depois torna-se constante. Tal como no teste anterior.

ii. Teste LU

Classe A

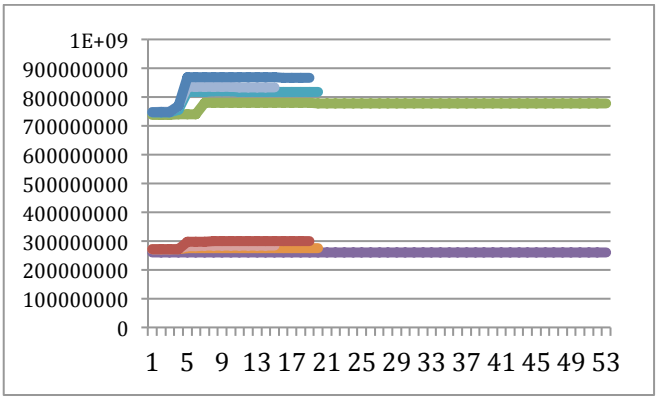


Gráfico 11 – Memória utilizada LU - A

Classe B

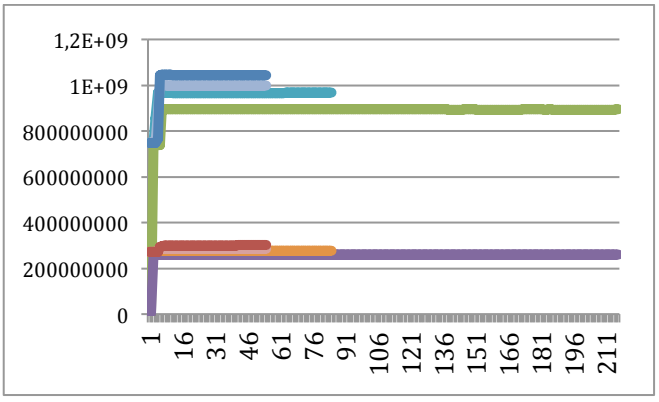


Gráfico 12 – Memória utilizada LU - B

Classe C

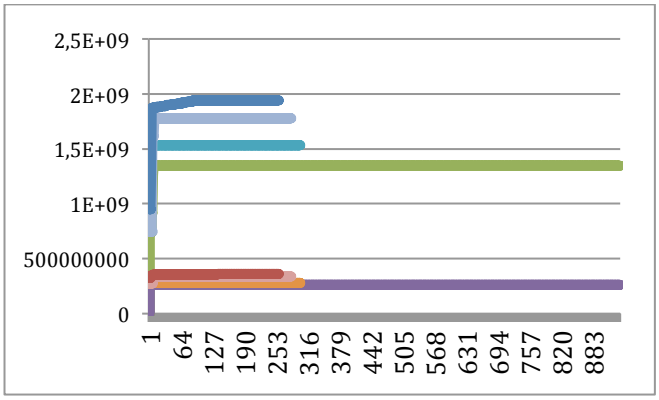


Gráfico 13 – Memória utilizada LU - C

Nos testes da classe LU com a versão serial é possível ver que a memoria utilizada pelo programa mantém-se constante ao longo de toda a execução. No caso das versões que utilizam MPI existe um aumento proporcional ao numero de processos no inicio e depois torna-se aproximadamente constante, verificando-se algumas oscilações na utilização da cache.

iii. Teste MG

Classe A

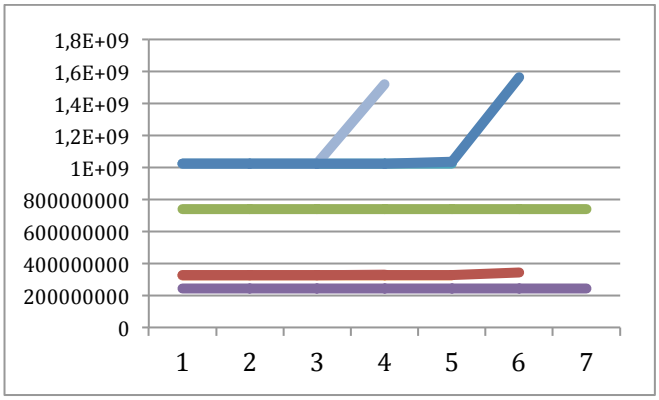


Gráfico 14 – Memória utilizada MG - A

Classe B

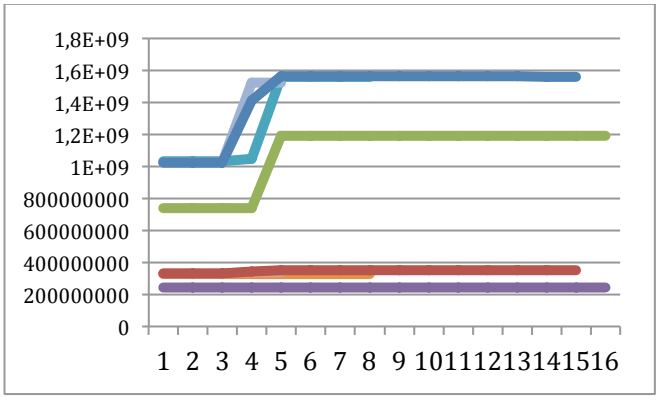


Gráfico 15 – Memória utilizada MG - B

Classe C

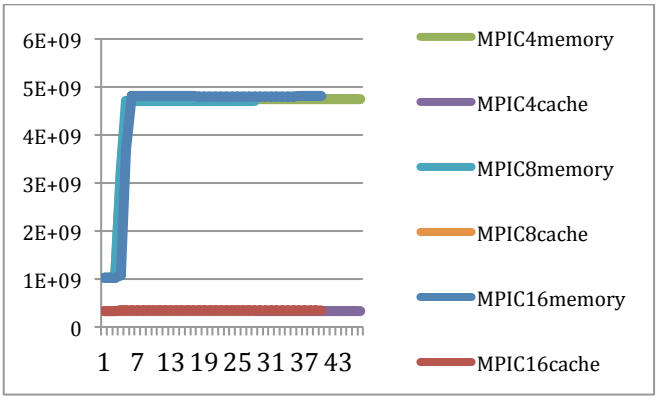


Gráfico 16 – Memória utilizada MG - C

Nos testes da classe MG com a versão serial é possível ver que a memoria utilizada pelo programa mantém-se constante ao longo de toda a execução. No caso das versões que utilizam MPI existe um aumento bastante significativo mesmo em relação aos outros testes.

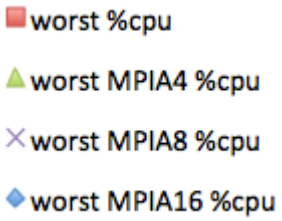
✓ Nesta observação como era esperado com o aumento do numero de processos a memoria ocupada é maior. E isso é possível ver facilmente nos gráficos apresentados. Estes gráficos podem ser vistos com mais detalhe nos documentos *excel* em anexo, na linha do meio (linha referente à memoria). Em todos estes gráficos devem ser descartados os pedaços correspondentes ao inicio pois alguns deles apresentam "barras verticais" incluídas nos gráficos que são anteriores ao inicio da execução.

c. Escritas e leituras

✓ Neste campo, na linha de cima dos anexos, pode-se observar que ao nível de escritas e leituras não se obtém grandes valores, as escritas e leituras são quase nulas não parecendo relevante os valores pontuais que aparecem. Portanto achei por bem que esses gráficos não fossem tidos em conta aqui.

d. Utilização do CPU

Neste secção será abordada a utilização do CPU para diferentes tipos de execuções. Para os gráficos desta secção a legenda utilizada é a seguinte:



i. Teste EP

Classe A

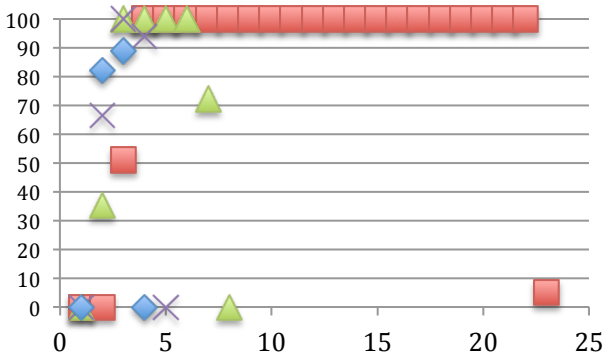


Gráfico 17 – Utilização do CPU em EP – A

Nos momentos iniciais a utilização do CPU é de 0% e depois vai subindo até atingir os 100% e mantém-se assim até ao fim da execução, isto para qualquer numero de processos.

Classe B

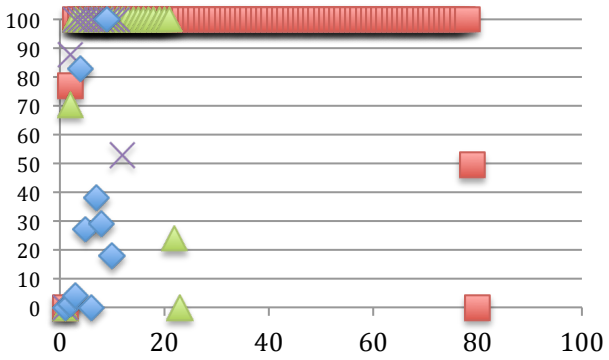


Gráfico 18 – Utilização do CPU em EP – B

Nos momentos iniciais a utilização do CPU é de 0% e depois vai subindo até atingir os 100% e mantém-se assim até ao fim da execução. Para este teste no caso da utilização do MPI com 16 processos apenas se mantém nos 100% por uns meros segundos, isto porque o programa com esta opção executa bastante rápido e por isso é bastante diluído ao longo da criação de processos e a termina-los.

Classe C

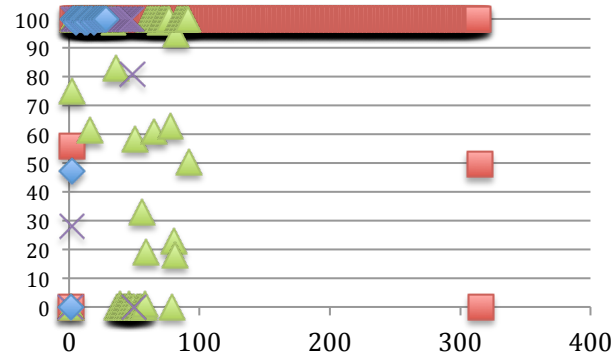


Gráfico 19 – Utilização do CPU em EP – C

Nos momentos iniciais a utilização do CPU é de 0% e depois vai subindo até atingir os 100% e mantém-se assim até ao fim da execução, isto para qualquer numero de processos. De forma análoga ao teste anterior (para a classe B) verifica-se o mesmo mas para o MPI com 4 processos.

ii. Teste IS

Classe A

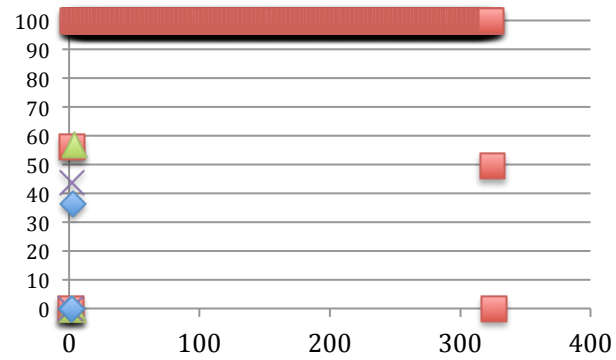


Gráfico 20 – Utilização do CPU em IS – A

Nos momentos iniciais a utilização do CPU é de 0% e depois vai subindo até atingir os 100% e mantém-se assim até ao fim da execução, isto para qualquer numero de processos.

Classe B

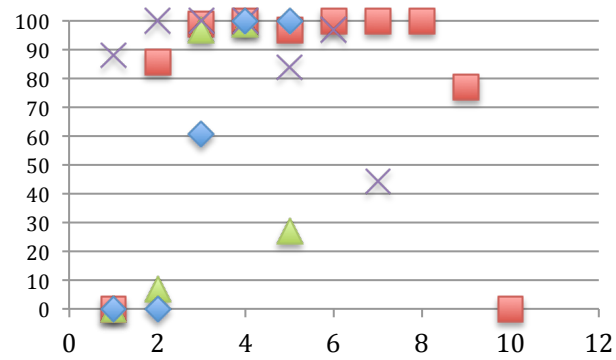


Gráfico 21 – Utilização do CPU em IS – B

Nos momentos iniciais a utilização do CPU é de 0% e depois vai subindo até atingir os 100% e mantém-se assim até ao fim da execução, isto para qualquer numero de processos. Para alguns testes vê-se que existem oscilações que são devidas à utilização de várias outras pessoas do mesmo nodo do *cluster*.

Classe C

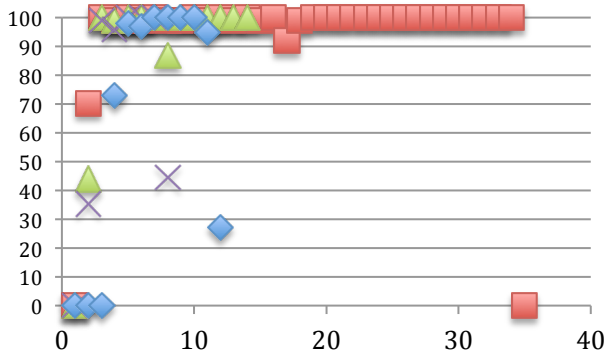


Gráfico 22 – Utilização do CPU em IS – C

Nos momentos iniciais a utilização do CPU é de 0% e depois vai subindo até atingir os 100% e mantém-se assim até ao fim da execução, isto para qualquer numero de processos.

iii. Teste LU

Classe A

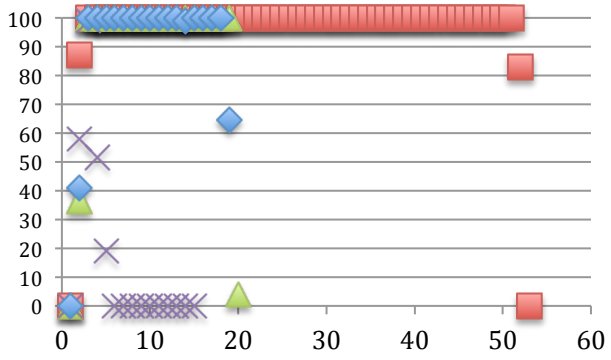


Gráfico 23 – Utilização do CPU em LU – A

Nos momentos iniciais a utilização do CPU é de 0% e depois vai subindo até atingir os 100% e mantém-se assim até ao fim da execução, isto para qualquer numero de processos. Para uma das versões aparece durante vários instantes uma utilização de 0% pois um dos CPU não estava a ser utilizado.

Classe B

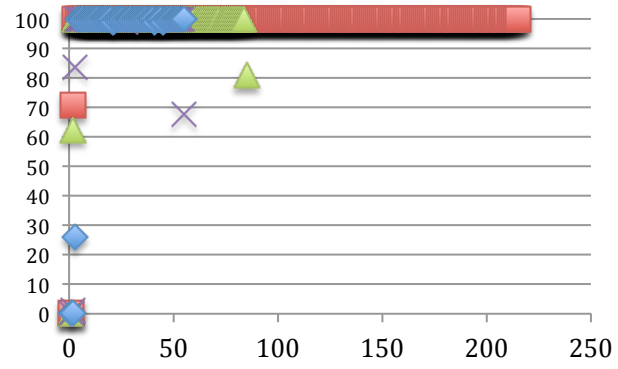


Gráfico 24 – Utilização do CPU em LU – B

Nos momentos iniciais a utilização do CPU é de 0% e depois vai subindo até atingir os 100% e mantém-se assim até ao fim da execução, isto para qualquer numero de processos.

Classe C

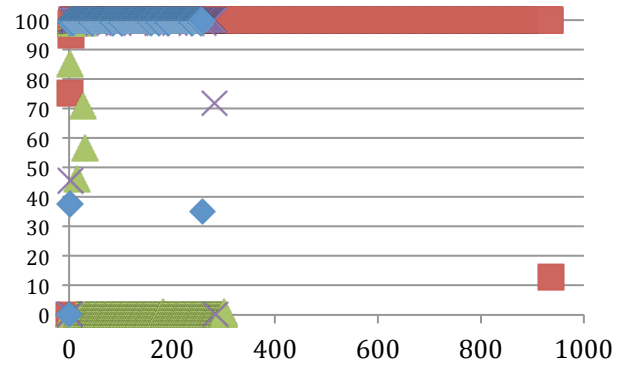


Gráfico 25 – Utilização do CPU em LU – C

Nos momentos iniciais a utilização do CPU é de 0% e depois vai subindo até atingir os 100% e mantém-se assim até ao fim da execução, isto para qualquer numero de processos. Para uma das versões aparece durante vários instantes uma utilização de 0% pois um dos CPU não estava a ser utilizado.

iv. Teste MG

Classe A

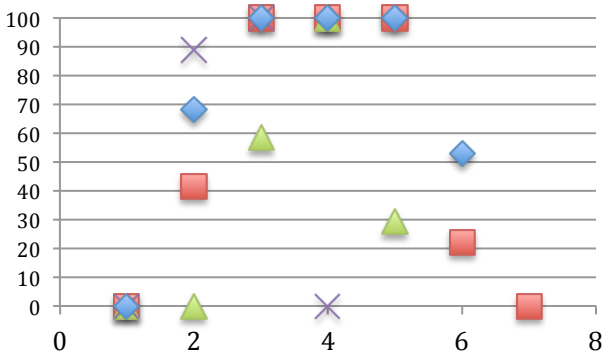


Gráfico 26 – Utilização do CPU em MG – A

De todos os resultados obtidos este é o mais estranho de todos muito porque foi extremamente difícil de medir com precisão pois este *kernel* executa muito rápido e os resultados obtidos "parecem aleatórios".

Classe B

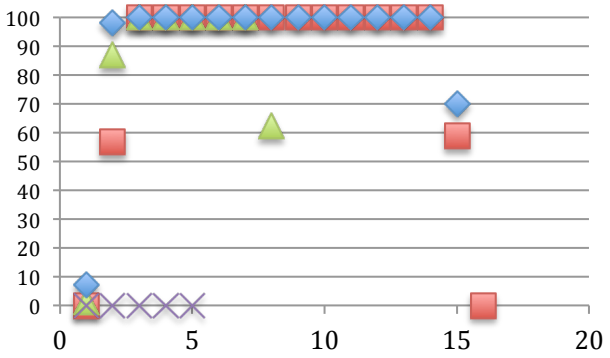


Gráfico 27 – Utilização do CPU em MG – B

Nos momentos iniciais a utilização do CPU é de 0% e depois vai subindo até atingir os 100% e mantém-se assim até ao fim da execução, isto para qualquer numero de processos

Classe C

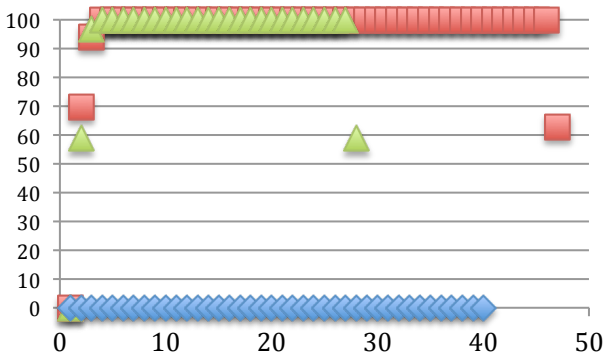


Gráfico 28 – Utilização do CPU em MG – C

Nos momentos iniciais a utilização do CPU é de 0% e depois vai subindo até atingir os 100% e mantém-se assim até ao fim da execução, isto para qualquer numero de processos. É possível observar que para este teste especifico o MPI com 16 processos tem um processo a utilizar sempre 0% do CPU.

✓ Como era expectável seriam utilizadas tantos processos para executar o programa quantas as "pedidas". E como é possível ver pelos gráficos em anexo, na linha debaixo observa-se que durante a execução a utilização de cada CPU está em 100% ou muito perto disso.

5. Comentário dos resultados

Depois de observados estes resultados pode-se chegar à conclusão que nenhum dos testes tem um impacto significativo ao nível das escritas e leituras do disco isto deve-se bastante ao programa conseguir facilmente executar apenas utilizando a memoria.
Ao nível da memoria consegue-se ver que a criação de processos tem um impacto bastante grande.
Ao nível da utilização do CPU todos os *kernels* e programas executados utilizaram sempre o máximo do CPU ou muito perto disso, salvo raras exceções.

Obtenção dos resultados

a. Esquema de funcionamento

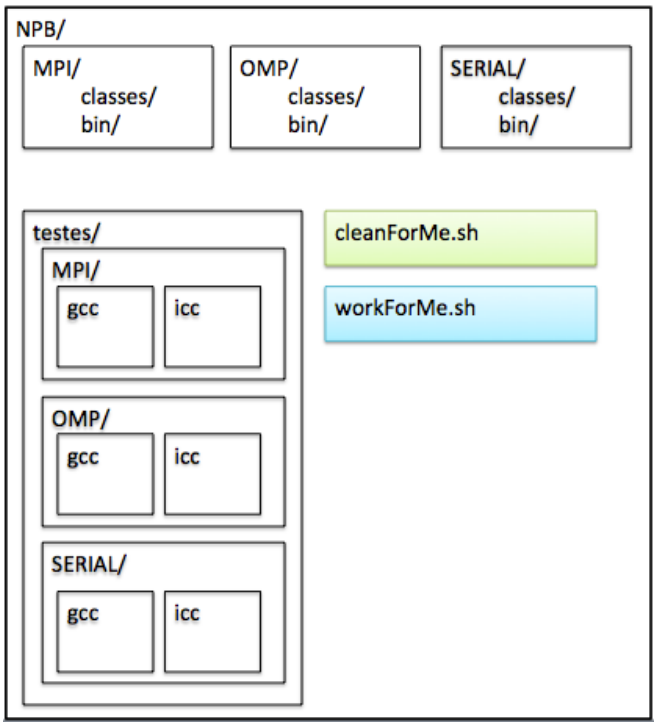


Figura 1 – Organização da pasta base

Este figura ilustra a forma como está organizado o ambiente de testes, uma pasta principal que contém quatro pastas, uma com o local onde ficam os executáveis e onde são feitos os testes. A pasta de testes contém os testes para MPI, OMP e para o SERIAL. Em cada um deles há uma pasta para os que foram compilados com o compilador da GNU e outra para os da Intel.

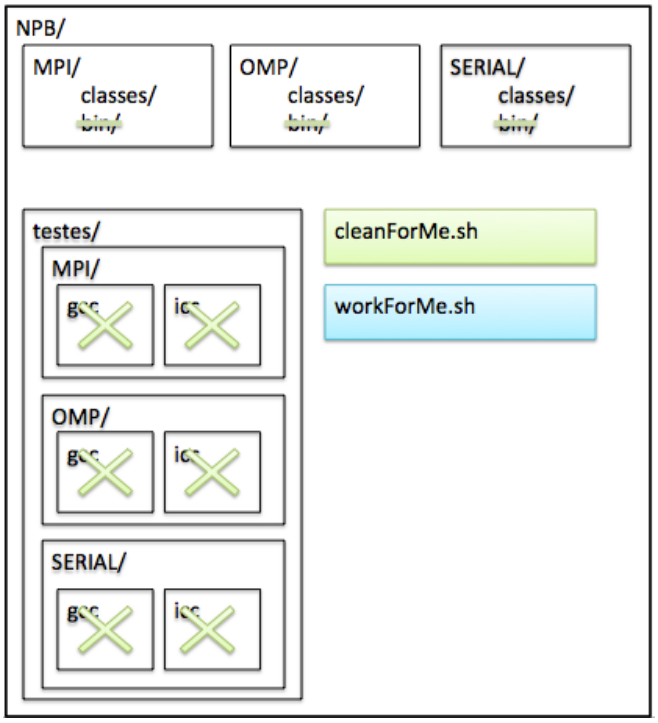


Figura 2 – Funcionamento do script 1

Nesta figura é possível ver o que acontece quando é executado o *script* "cleanForMe". São apagados os executáveis que se encontram na pasta bin dos vários tipos e ainda na pasta de testes. Este *script* é explicado em pormenor mais à frente.

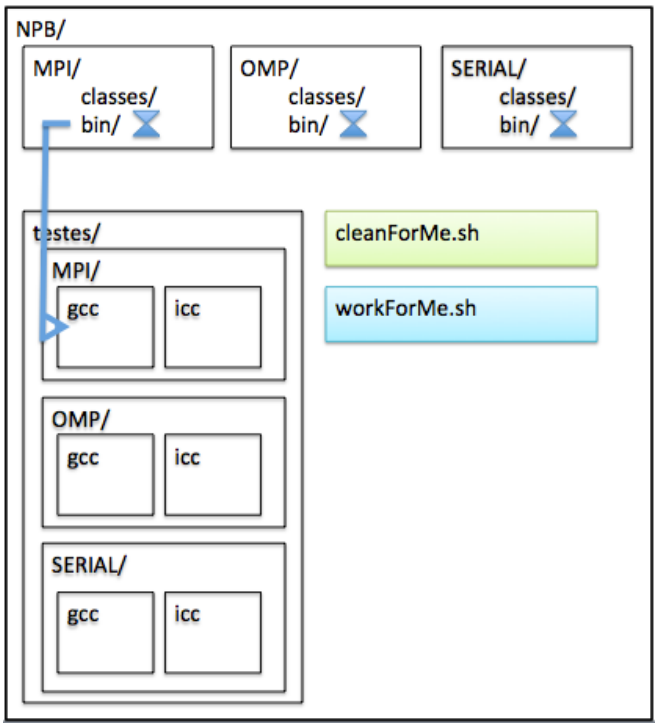


Figura 3 – Funcionamento do script 2

Nesta figura observa-se a execução do *script* "workForMe". Este script permite poupar muito trabalho pois ele compila todas os tipos para as três classes de teste com um determinado compilador e faz isto para o MPI, OMP e Serial em simultâneo.

b. Scripts utilizados

i. cleanForMe.sh

```
#!/bin/sh
rm NPB3.3-MPI/bin/*
rm NPB3.3-OMP/bin/*
rm NPB3.3-SER/bin/*

for tp in "MPI" "OMP" "SERIAL"
do
     for cl in A B C
     do
     rm NPB_tests/$tp/GCC/type$cl/*
     rm NPB_tests/$tp/ICC/type$cl/*
     done
done
```

Este é o *script* que foi apresentado de forma superficial anteriormente. Começa por remover tudo o que se encontra nas pastas bin das pastas de compilação. E depois para os tipos MPI, OMP e Serial remove nas pastas de teste quer seja compilado com o ICC ou com o GCC.

ii. workForMe.sh

```
#/bin/sh
comp=icc
 ##MPI
mv PB3.3-MPI/config/make.def-$comp
PB3.3-MPI/config/make.def

cd NPB3.3-MPI/
for ts in “mg” “lu” “is” “ep”
do
     for th in 2 4 8 9 16 32
     do
          make $ts CLASS=A NPROCS=$th
          make $ts CLASS=B NPROCS=$th
          make $ts CLASS=C NPROCS=$th
     done
done

cd bin/
mv *.A.*
../../NPB_tests/MPI/$comp/typeA/
mv *.B.*
../../NPB_tests/MPI/$comp/typeB/
mv *.C.*
../../NPB_tests/MPI/$comp/typeC/
cd ../../

mv PB3.3-MPI/config/make.def PB3.3-
MPI/config/make.def-$comp
##OMP
mv PB3.3-OMP/config/make.def PB3.3-
OMP/config/make.def-$comp

cd NPB3.3-OMP/
for ts in “mg” “lu” “is” “ep”
do
     make $ts CLASS=A
     make $ts CLASS=B
     make $ts CLASS=C
done

cd bin/
mv *.A.*
../../NPB_tests/OMP/GCC/typeA/
mv *.B.*
../../NPB_tests/OMP/GCC/typeB/
mv *.C.*
../../NPB_tests/OMP/GCC/typeC/
cd ../../

mv NPB3.3-OMP/config/make.def
NPB3.3-OMP/config/make.def-$comp

 ##SERIAL
cd NPB3.3-SER/

...
```

Este apenas tem aqui um excerto, pois para o caso do OMP e Serial funciona de forma análoga à apresentada aqui. Começa por mudar a makefile para a versão escolhida, icc ou gcc. Depois compila os varios testes, neste caso o mg, lu, is e ep para as diferentes classes com diferentes numero de processos. Por fim move os executáveis para as pastas respectivas e recoloca a makefile com o nome predefinido.

iii. cpi.pbs

```
#!/bin/bash
#PBS -l walltime=05:00:00
#PBS-j oe
#PBS -N a
#PBS -lnodes=2:ppn=20
cat $PBS_NODEFILE

cd ~/NPB3.3.1/NPB_tests/

echo "##########################"
#SER
echo "serial now ------------------"
cd SERIAL
./script_serial.sh
cd ..

echo "mpi now ---------------------"
cd MPI
./script_mpi.sh
cd ..

echo "omp now ---------------------"
cd OMP
./script_omp.sh
cd ..

cd ..

#cat /proc/cpuinfo
```

Este cpi limita-se a ser utilizado com o qsub de forma a “angariar” um nodo de execução e executar os *scripts* que se seguem.

iv. script_omp.sh

```
#!/bin/bash
comp=icc
exec > "t_omp.txt"

for th in 1 4 8 16
do
     for cl in “mg” “lu” “is” “ep”
     do
     export OMP_NUM_THREADS=$th
     echo $th "#### $comp/$cl.A"
     ./$comp/typeA/$cl.A.x
     echo $th "#### $comp/$cl.B"
     ./$comp/typeB/$cl.B.x
     echo $th "#### $comp/$cl.C"
     ./$comp/typeC/$cl.C.x
done
```

Este executa todos os testes da versão OMP dos vários tipos e coloca os resultados deles num ficheiro chamado t_omp.txt.

v. script_mpi.sh

```
#!/bin/bash
comp=icc
exec > "t_mpi.txt"

for th in 1 4 9 16 32
do
     for cl in “ep” “is” “lu” “mg”
     do
     echo $th "#### MPI $cl.A.$th"
     mpirun –np "$th"
"./$comp/typeA/ep.A.$th"
     echo $th "#### MPI $cl.B.$th"
     mpirun –np "$th"
"./$comp/typeB/$cl.B.$th"
     echo $th "#### MPI $cl.C.$th"
     mpirun –np "$th"
"./$comp/typeC/$cl.C.$th"
done
```

Este executa todos os testes da versão MPI dos vários tipos e dos diferentes numero de processos e coloca os resultados deles num ficheiro chamado t_mpi.txt.

vi. script_serial.sh

```
#!/bin/bash
comp=icc
exec > "t_serial.txt"

for tp in “ep” “is” “lu” “mg”
do
     echo "#### SERIAL $tp.A"
     ./$comp/typeA/$tp.A.x
     echo "#### SERIAL $tp.B"
     ./$comp/typeB/$tp.B.x
     echo "#### SERIAL $tp.C"
     ./$comp/typeC/$tp.C.x
done
```

Este executa todos os testes da versão Serial dos vários tipos e coloca os resultados deles num ficheiro chamado t_serial.txt.

Conclusão

Com a realização deste trabalho foi possível constatar uma vez mais que a paralelização do código faz com que o processamento seja mais eficiente e mais rápido. Este trabalho foi bastante útil na medida em que permite utilizar novas ferramentas que foram abordadas nesta unidade curricular. Nunca pensei quando comecei este trabalho que ele ia dar tanto trabalho. Uma coisa que me tinha proposto a fazer e que no fim não foi feito era utilizar o comando ***dstat*** para os testes de OMP mas devido às limitações de tempo foi impossível faze-lo.