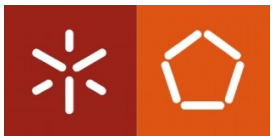
Portfolio

Engenharia de Sistemas da Computação

Computação Paralela Distribuída

Mestrado em Engenharia Informática

Universidade do Minho



Duarte Nuno Ferreira Duarte

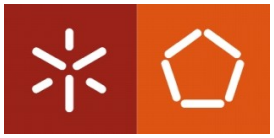
pg27715

Índice

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

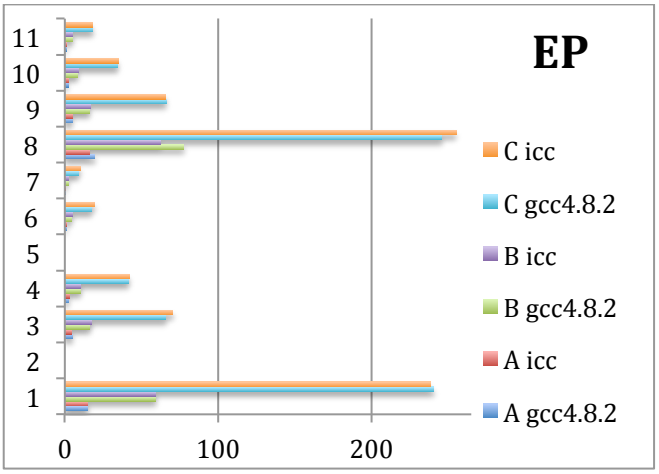


Gráfico 1 – Tempo de execução EP

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.

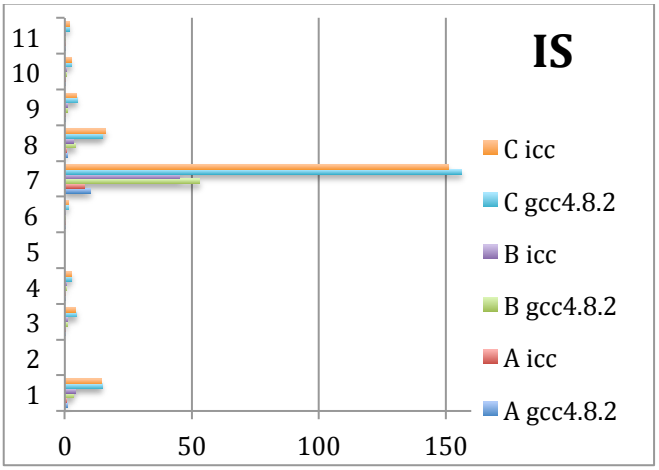


Gráfico 2 – Tempo de execução IS

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.

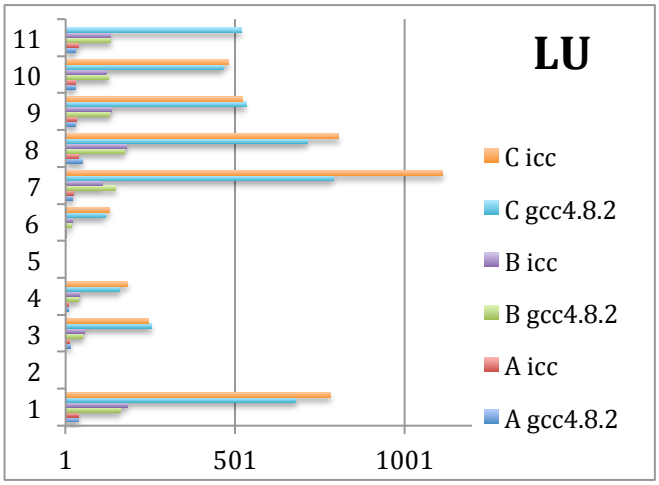


Gráfico 3 – Tempo de execução LU

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.

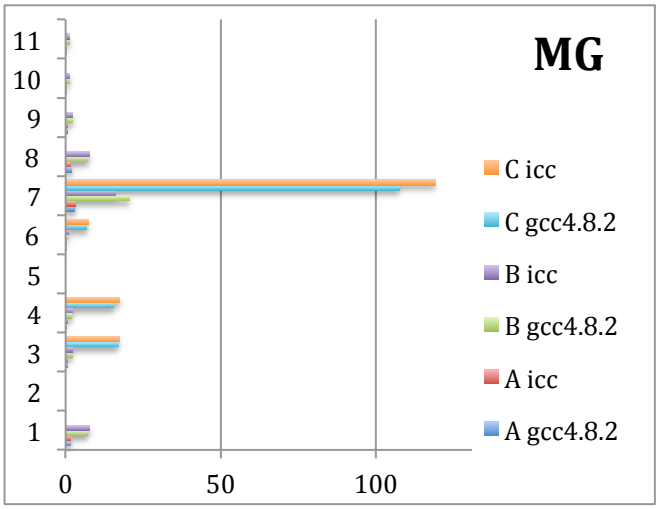


Gráfico 4 – Tempo de execução MG

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

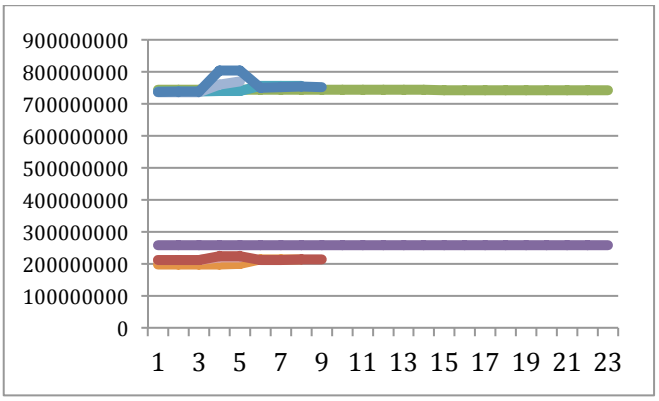


Gráfico 5 – Memória utilizada EP - A

Classe B

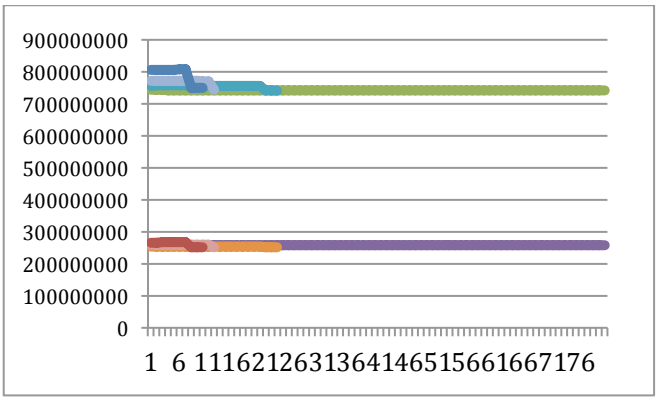


Gráfico 6 – Memória utilizada EP - B

Classe C

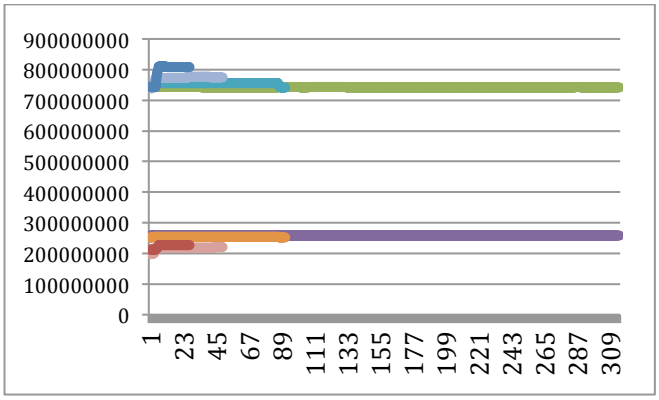


Gráfico 7 – Memória utilizada EP – C

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

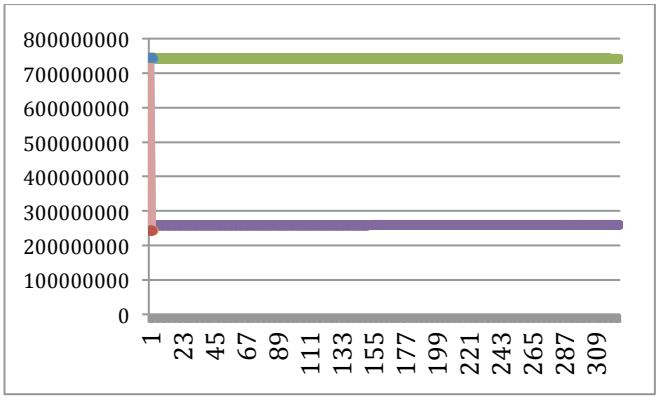


Gráfico 8 – Memória utilizada IS - A

Classe B

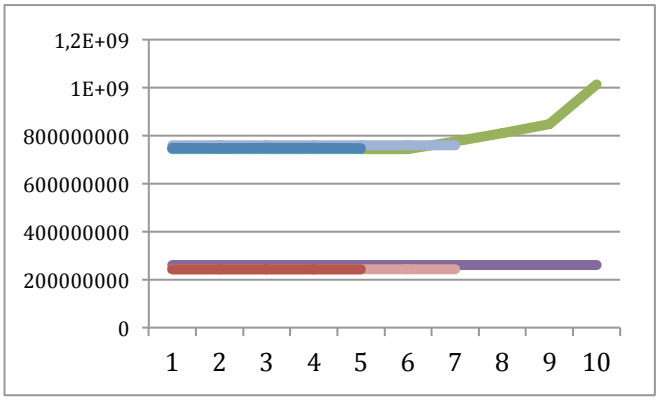


Gráfico 9 – Memória utilizada IS - B

Classe C

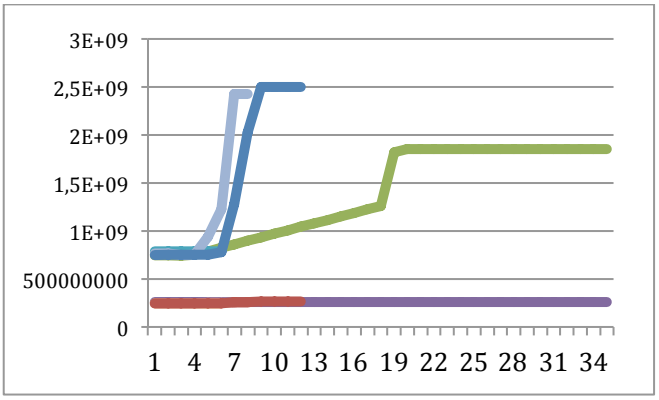


Gráfico 10 – Memória utilizada IS - C

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

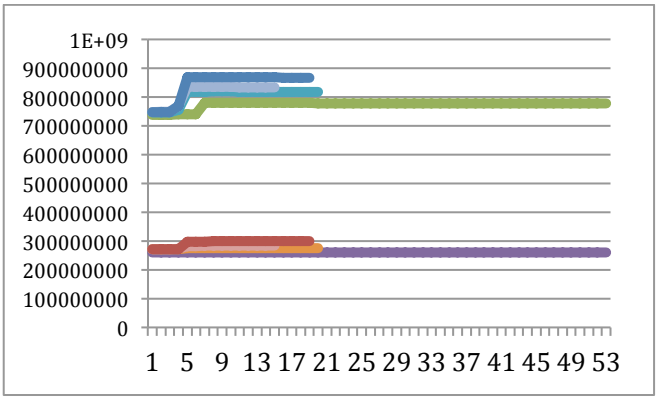


Gráfico 11 – Memória utilizada LU - A

Classe B

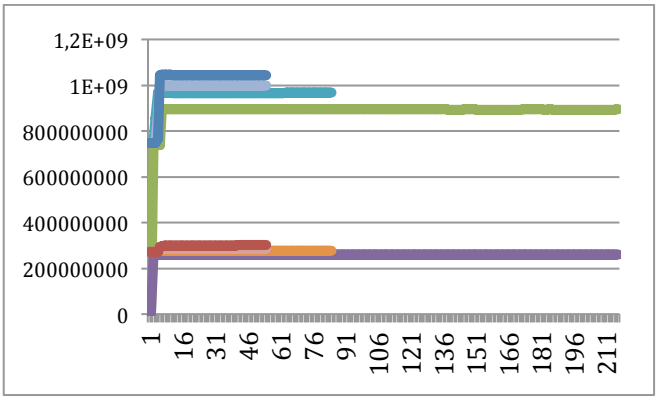


Gráfico 12 – Memória utilizada LU - B

Classe C

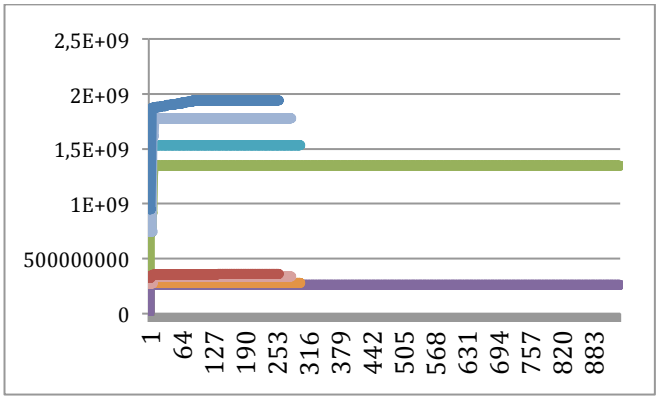


Gráfico 13 – Memória utilizada LU - C

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

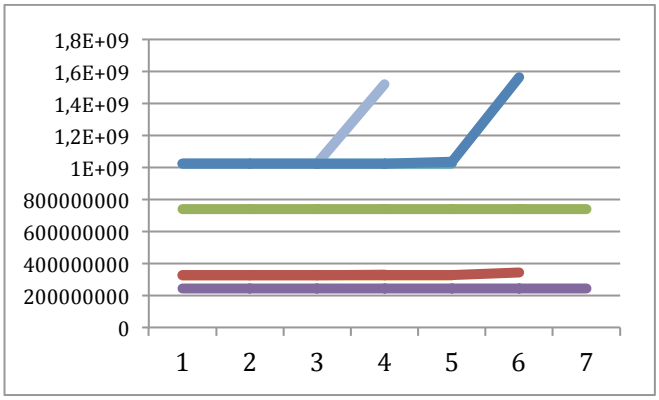


Gráfico 14 – Memória utilizada MG - A

Classe B

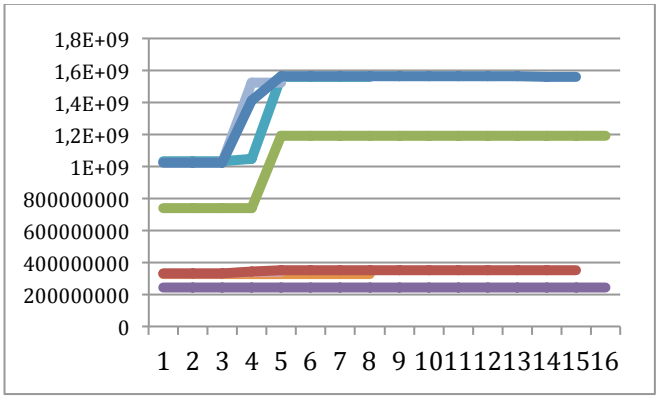


Gráfico 15 – Memória utilizada MG - B

Classe C

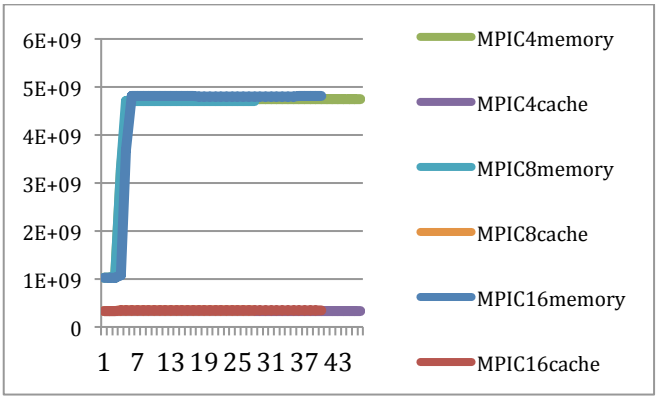


Gráfico 16 – Memória utilizada MG - C

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.

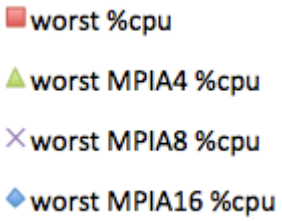
✓ Nesta observação como era esperado com o aumento do numero de processos a memoria ocupada é maior. E isso é possível ver facilmente nos gráficos apresentados. Estes gráficos podem ser vistos com mais detalhe nos documentos *excel* em anexo, na linha do meio (linha referente à memoria). Em todos estes 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

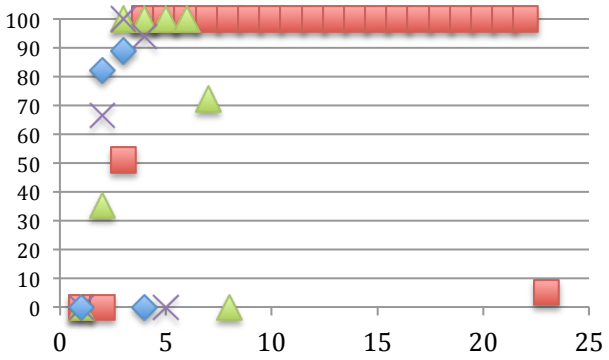


Gráfico 17 – Utilização do CPU em EP – A

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

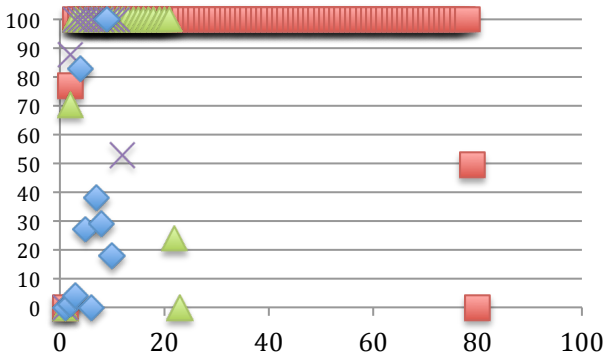


Gráfico 18 – Utilização do CPU em EP – B

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

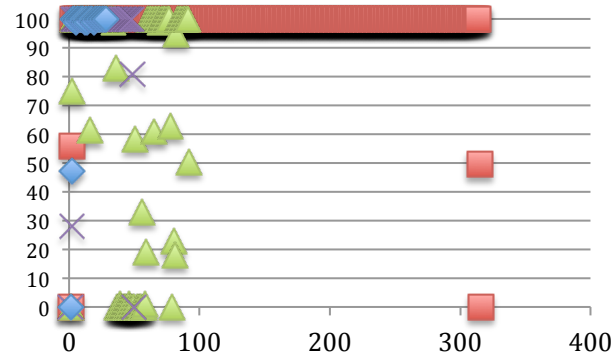


Gráfico 19 – Utilização do CPU em EP – C

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

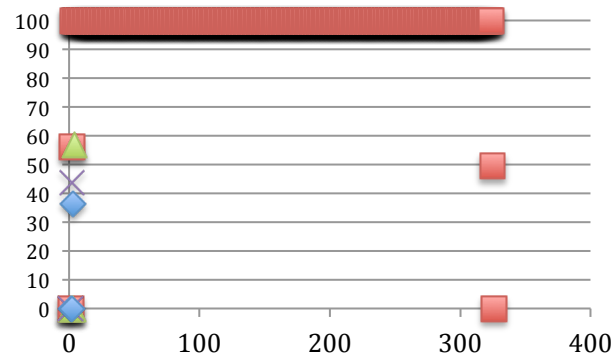


Gráfico 20 – Utilização do CPU em IS – A

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

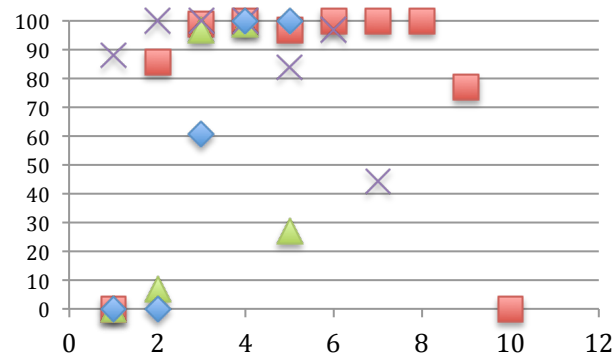


Gráfico 21 – Utilização do CPU em IS – B

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

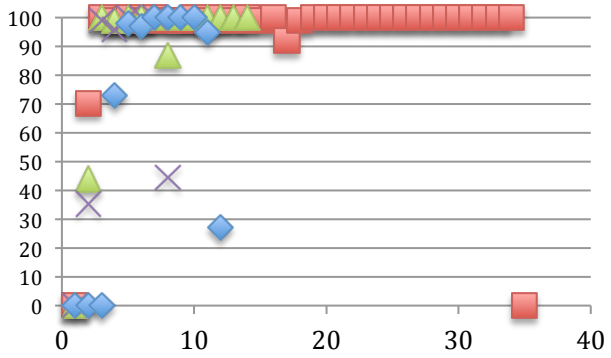


Gráfico 22 – Utilização do CPU em IS – C

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

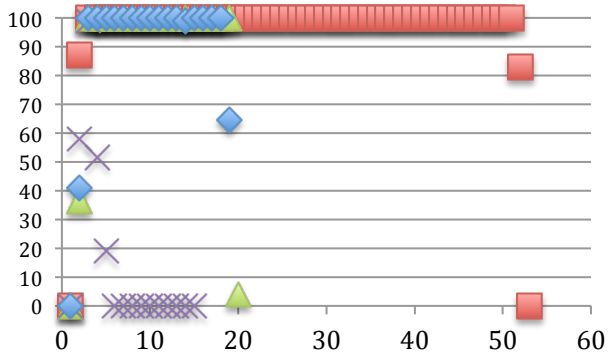


Gráfico 23 – Utilização do CPU em LU – A

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

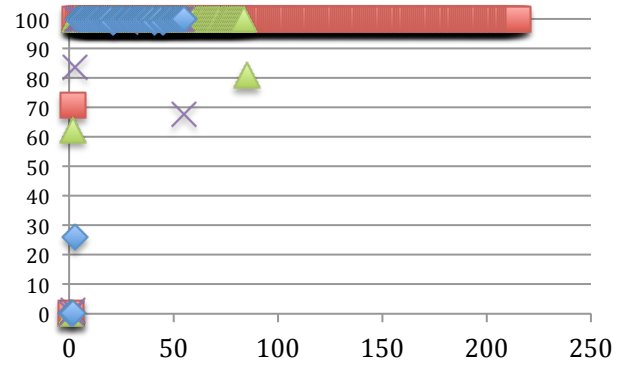


Gráfico 24 – Utilização do CPU em LU – B

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

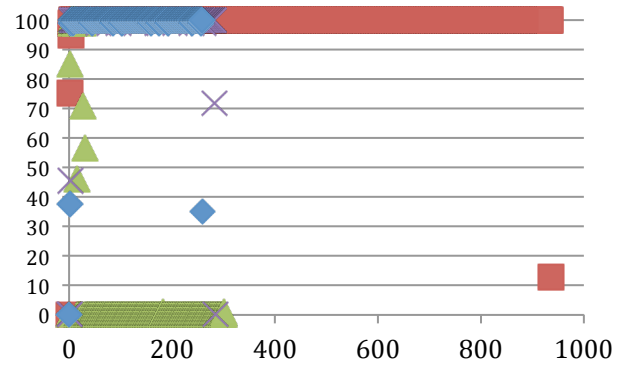


Gráfico 25 – Utilização do CPU em LU – C

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

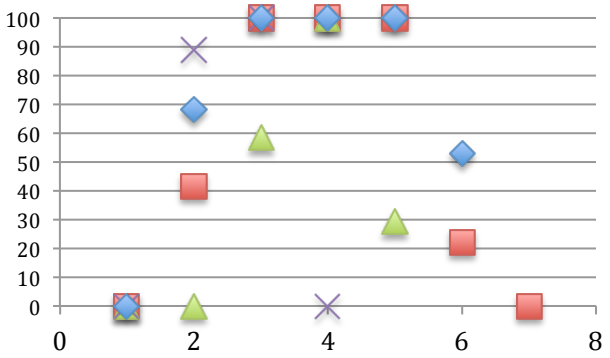


Gráfico 26 – Utilização do CPU em MG – A

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

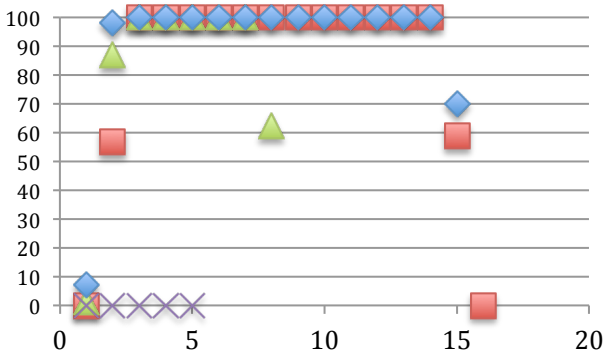


Gráfico 27 – Utilização do CPU em MG – B

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

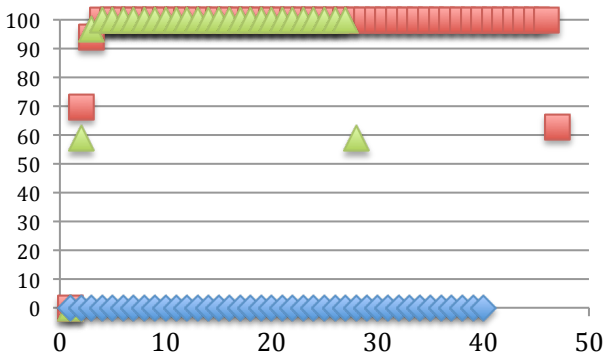


Gráfico 28 – Utilização do CPU em MG – C

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

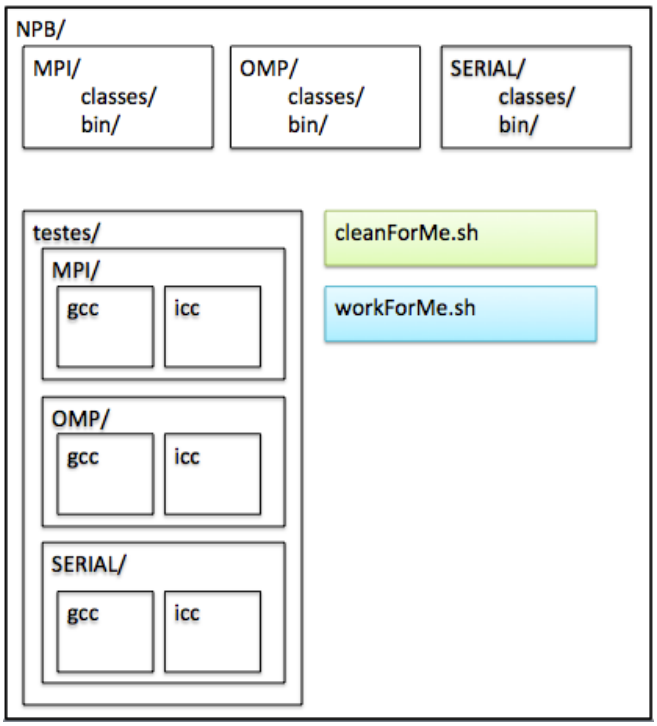


Figura 1 – Organização da pasta base

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.

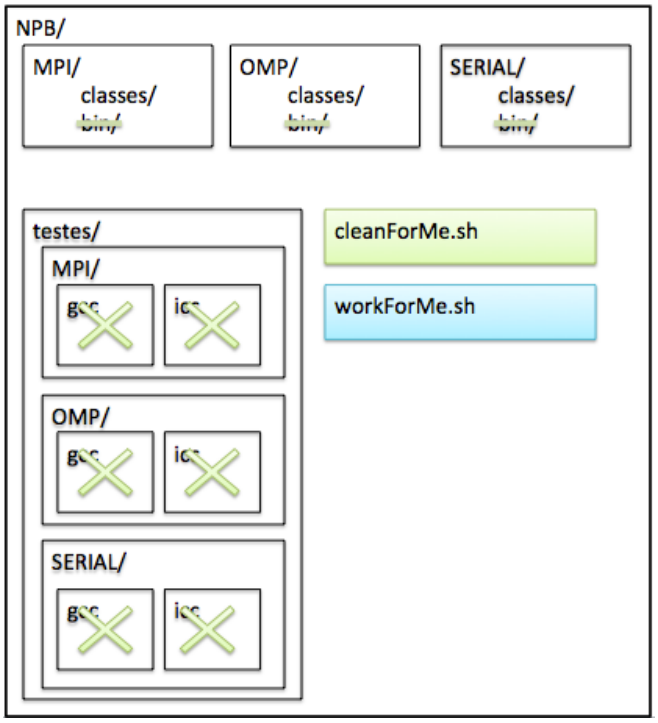


Figura 2 – Funcionamento do script 1

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.

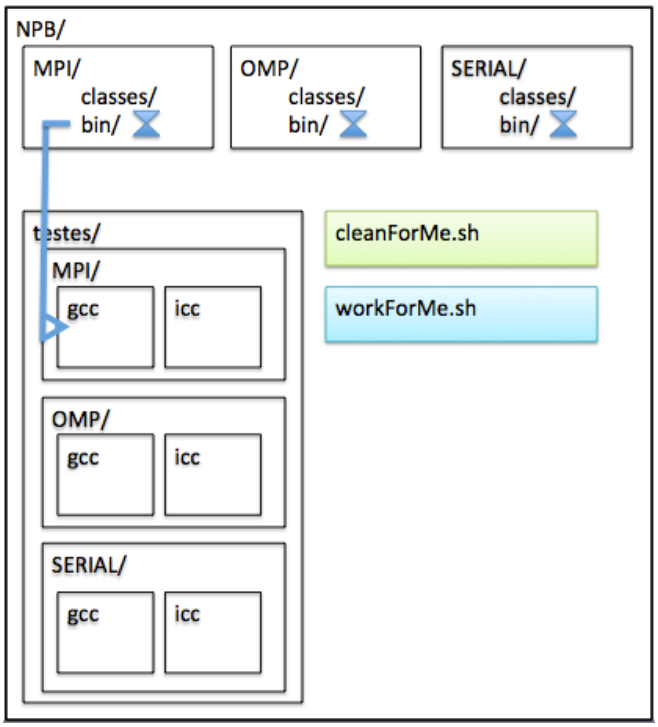


Figura 3 – Funcionamento do script 2

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.

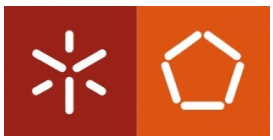
Monitorização

Engenharia de Sistemas da Computação

Computação Paralela Distribuída

Mestrado em Engenharia Informática

Universidade do Minho



Duarte Nuno Ferreira Duarte

pg27715

Índice

Introdução

Este trabalho demonstra a utilização de comandos de monitorização do sistema tais como o vmstat, top e ps. Este comandos são úteis para obter valores de desempenho das aplicações e diferentes nós do Search. Estes comandos são capazes de gerar estatísticas de utilização de uma aplicação, utilizador ou sistema e serão úteis neste caso para obter os dados de uma simples aplicação.

Código de teste

O código apresentado de seguida é o código que será utilizado pelos comandos do sistema.

```
void multiply_matrices() {
        int i, j, k ;
        for (i = 0 ; i < MSIZE ; i++) {
                for (j = 0 ; j < MSIZE ; j++) {
                        float sum = 0.0 ;
                        for (k = 0 ; k < MSIZE ; k++) {
                             sum = sum + (matrix_a[i][k] *
matrix_b[k][j]);
printf("x\n");
                }
               matrix_r[i][j] = sum ;
                }
        }
```

Como é possível ver o código é o mais básico que existe de multiplicação de matrizes não possui qualquer tipo de otimização mas faz o esperado. Apenas o printf é de estranhar mas só aparece ali para que a aplicação decorra tempo suficiente para ser monitorizada.

Dados Obtidos

São apresentados os três comandos que vão ser testados no nó `compute-431-10` e no nó `compute-321-1`.

Compute-431-10

VMSTAT

Antes da aplicação estar a decorrer os dados do VMSTAT eram os seguintes:

```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0  11200 44337260 252896 160604    0    0     0     0    0    0  4  0 96  0
```

Como é possivel observer pela tabela acima o numero de processos à espera para executar e os processos que não voltam a executar estão ambos a zero pois não existe nenhum processo por parte do utilizador. Ao nivel da memoria os valores não serão comentados nesta altura. Ao nivel de swap e IO todos os valores se encontram a zero pois não existe qualquer processo a executar. Ao nivel de CPU a utilização é apenas por parte do sistema ao executar o código do kernel.

Durante a execução o VMSTAT já devolvia os seguintes valores:

```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0  11200 44331884 253084 161636    0    0     0     0    0    0  4  0 96  0  0
```

Durante a execução é então possível ver que já existe um processo na fila de processos à espera para executar. Em termos de memória a única parte que sofreu alteração considerável foi a da memória livre e isto deve-se à execução da multiplicação de matrizes pois esta aplicação tem alguma exigência ao nível da memoria. Em swap os valores mantêm-se a zero como seria de esperar, não é suposto (para o tamanho utilizado) que haja valores a terem de ir para disco. Ao nível de IO os valores mantêm-se também a zero como seria de esperar pois a aplicação não tem nada que use chamadas de IO. Ao nível de CPU os valores mantiveram-se sendo que aqui eram esperadas mudanças pois a execução da aplicação devia alterar estes valores pelo menos um bocado.

TOP

Antes da aplicação estar a decorrer os dados do TOP eram os seguintes:

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	19348	1144	908	S	0.0	0.0	0:01.47	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.04	kthreadd
3	root	RT	0	0	0	0	S	0.0	0.0	0:03.90	migration/0
4	root	20	0	0	0	0	S	0.0	0.0	0:04.29	ksoftirqd/0
5	root	RT	0	0	0	0	S	0.0	0.0	0:00.05	migration/0
6	root	RT	0	0	0	0	S	0.0	0.0	0:05.43	watchdog/0
7	root	RT	0	0	0	0	S	0.0	0.0	0:54.14	migration/1
8	root	RT	0	0	0	0	S	0.0	0.0	0:00.05	migration/1
9	root	20	0	0	0	0	S	0.0	0.0	3:49.93	ksoftirqd/1
10	root	RT	0	0	0	0	S	0.0	0.0	0:04.10	watchdog/1
11	root	RT	0	0	0	0	S	0.0	0.0	0:32.03	migration/2
12	root	RT	0	0	0	0	S	0.0	0.0	0:00.05	migration/2
13	root	20	0	0	0	0	S	0.0	0.0	1:18.82	ksoftirqd/2
14	root	RT	0	0	0	0	S	0.0	0.0	0:04.13	watchdog/2
15	root	RT	0	0	0	0	S	0.0	0.0	0:02.49	migration/3
16	root	RT	0	0	0	0	S	0.0	0.0	0:00.05	migration/3
17	root	20	0	0	0	0	S	0.0	0.0	1:00.41	ksoftirqd/3

Como é possível observar pela tabela acima não existem processos a executar neste nó.

Durante a execução o TOP já devolvia os seguintes valores:

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
4956	pg27715	20	0	6984	3368	364	R	72.9	0.0	0:30.71	app
4646	root	20	0	27096	7372	64	R	35.5	0.0	0:15.06	pbs_mom
119	root	20	0	0	0	0	S	15.0	0.0	1:15.05	events/20
105	root	20	0	0	0	0	S	3.7	0.0	1:19.63	events/6
120	root	20	0	0	0	0	S	1.9	0.0	1:08.79	events/21
4962	pg27715	20	0	15292	1424	828	R	1.9	0.0	0:00.01	top
1	root	20	0	19348	1144	908	S	0.0	0.0	0:01.47	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.04	kthreadd
3	root	RT	0	0	0	0	S	0.0	0.0	0:03.90	migration/0
4	root	20	0	0	0	0	S	0.0	0.0	0:04.29	ksoftirqd/0
5	root	RT	0	0	0	0	S	0.0	0.0	0:00.05	migration/0
6	root	RT	0	0	0	0	S	0.0	0.0	0:05.43	watchdog/0
7	root	RT	0	0	0	0	S	0.0	0.0	0:54.14	migration/1
8	root	RT	0	0	0	0	S	0.0	0.0	0:00.05	migration/1
9	root	20	0	0	0	0	S	0.0	0.0	3:49.93	ksoftirqd/1
10	root	RT	0	0	0	0	S	0.0	0.0	0:04.10	watchdog/1
11	root	RT	0	0	0	0	S	0.0	0.0	0:32.03	migration/2

Quando foi colocada a aplicação a executar esta passou a utilizar quase todos os recursos deste nó, é possível observar na primeira linha, a linha correspondente à aplicação, que 72.9 % do CPU passou a ser utilizado pela aplicação e a memória não sofreu qualquer alteração. Um dado curioso é o associado ao VIRT que se trata da quantidade de memoria virtual utilizada pela aplicação e ainda o RES que se trata da memória permanente utilizada pela aplicação.

PS

Antes da aplicação estar a decorrer os dados do PS AUX eram os seguintes:

```
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
pg27715    4647  0.0  0.0 108580  2096 pts/0    S+   12:01   0:00 -bash
pg27715    4856  0.0  0.0 108444  1936 pts/1    S    12:10   0:00 -bash
pg27715    4999  1.0  0.0 110240  1132 pts/1    R+   12:17   0:00 ps aux
```

Observando a tabela acima conclui-se que o utilizador não tinha nenhuma aplicação dele a decorrer. As únicas tarefas da parte do utilizador eram a própria bash e ainda a chamada do comando ps aux. Esta tabela foi obtida com o 'ps aux' e depois foram filtrados os processos do utilizador para que esta não fosse muito extensa.

Durante a execução o PS AUX já devolvia os seguintes valores:

```
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
pg27715    4647  0.0  0.0 108580  2096 pts/0    S    12:01   0:00 -bash
pg27715    4856  0.0  0.0 108444  1936 pts/1    S    12:10   0:00 -bash
pg27715    5004 75.0  0.0   6984  3368 pts/0    R+   12:18   0:02 ./app
pg27715    5005  1.0  0.0 110236  1124 pts/1    R+   12:18   0:00 ps aux
```

Enquanto decorria a aplicação é possível olhar para a terceira linha de resultados e ver que a aplicação estava a utilizar 75% do CPU.
Um facto interessante a visualizar corresponde a coluna STAT que apresenta o estado da execução neste caso o S por parte da bash significa que se encontra a espera que um evento conclua e o R por parte da aplicação e da chamada do comando significam que está a executar.

Compute-321-1

VMSTAT

Antes da aplicação estar a decorrer os dados do VMSTAT eram os seguintes:

```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0  19620 7758328 109616 104912    2    2     2     3    1    1 41  1 58  0  0
```

Como é possivel observer pela tabela acima o numero de processos à espera para executar e os processos que não voltam a executar estão ambos a zero pois não existe nenhum processo por parte do utilizador. Ao nivel da memoria os valores não serão comentados nesta altura. Ao nivel de swap e IO todos os valores se encontram a valores proximos de zero pois não existe qualquer processo a executar por parte do utilizador. Ao nivel de CPU a utilização é apenas por parte do utilizador mas deve-se ao facto de ter outro comando a utilizer o CPU o resto está em idle.

Durante a execução o VMSTAT já devolvia os seguintes valores:

```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 2  0  19620 7755708 109624 104912    2    2     2     3    1    1 41  1 58  0  0
```

Durante a execução é então possível ver que já existem processos na fila de processos à espera para executar. Em termos de memória a única parte que sofreu alteração considerável foi a da memória livre e isto deve-se à execução da multiplicação de matrizes pois esta aplicação tem alguma exigência ao nível da memoria. Em swap os valores mantêm-se a iguais como seria de esperar, não é suposto (para o tamanho utilizado) que haja valores a terem de ir para disco. Ao nível de IO os valores mantêm-se também no mesmo que estavam como seria de esperar pois a aplicação não tem nada que use chamadas de IO excepto os printf referidos acima. Ao nível de CPU os valores mantiveram-se sendo que aqui eram esperadas mudanças pois a execução da aplicação devia alterar estes valores pelo menos um bocado.

TOP

Antes da aplicação estar a decorrer os dados do TOP eram os seguintes:

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
19878	nobody	20	0	154m	1416	1088	S	0.3	0.0	114:34.68	gmond
24057	pg27715	20	0	15164	1332	932	R	0.3	0.0	0:00.03	top
1	root	20	0	19348	552	328	S	0.0	0.0	0:00.94	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.04	kthreadd
3	root	RT	0	0	0	0	S	0.0	0.0	0:47.51	migration/0
4	root	20	0	0	0	0	S	0.0	0.0	2571:20	ksoftirqd/0
5	root	RT	0	0	0	0	S	0.0	0.0	0:00.03	migration/0
6	root	RT	0	0	0	0	S	0.0	0.0	0:03.45	watchdog/0
7	root	RT	0	0	0	0	S	0.0	0.0	0:31.86	migration/1
8	root	RT	0	0	0	0	S	0.0	0.0	0:00.03	migration/1
9	root	20	0	0	0	0	S	0.0	0.0	732:29.16	ksoftirqd/1
10	root	RT	0	0	0	0	S	0.0	0.0	0:02.77	watchdog/1
11	root	RT	0	0	0	0	S	0.0	0.0	1:39.93	migration/2
12	root	RT	0	0	0	0	S	0.0	0.0	0:00.03	migration/2
13	root	20	0	0	0	0	S	0.0	0.0	491:16.04	ksoftirqd/2
14	root	RT	0	0	0	0	S	0.0	0.0	0:04.74	watchdog/2
15	root	RT	0	0	0	0	S	0.0	0.0	0:26.62	migration/3
16	root	RT	0	0	0	0	S	0.0	0.0	0:00.03	migration/3
17	root	20	0	0	0	0	S	0.0	0.0	375:20.75	ksoftirqd/3
18	root	RT	0	0	0	0	S	0.0	0.0	0:02.62	watchdog/3
19	root	RT	0	0	0	0	S	0.0	0.0	0:13.27	migration/4
20	root	RT	0	0	0	0	S	0.0	0.0	0:00.03	migration/4
21	root	20	0	0	0	0	S	0.0	0.0	207:19.13	ksoftirqd/4
22	root	RT	0	0	0	0	S	0.0	0.0	0:03.00	watchdog/4
23	root	RT	0	0	0	0	S	0.0	0.0	0:11.82	migration/5
24	root	RT	0	0	0	0	S	0.0	0.0	0:00.03	migration/5
25	root	20	0	0	0	0	S	0.0	0.0	268:23.96	ksoftirqd/5
26	root	RT	0	0	0	0	S	0.0	0.0	0:02.95	watchdog/5
27	root	RT	0	0	0	0	S	0.0	0.0	0:03.03	migration/6
28	root	RT	0	0	0	0	S	0.0	0.0	0:00.03	migration/6
29	root	20	0	0	0	0	S	0.0	0.0	218:41.67	ksoftirqd/6
30	root	RT	0	0	0	0	S	0.0	0.0	0:02.51	watchdog/6
31	root	RT	0	0	0	0	S	0.0	0.0	0:06.49	migration/7
32	root	RT	0	0	0	0	S	0.0	0.0	0:00.03	migration/7
33	root	20	0	0	0	0	S	0.0	0.0	235:33.38	ksoftirqd/7
34	root	RT	0	0	0	0	S	0.0	0.0	0:02.39	watchdog/7
35	root	20	0	0	0	0	S	0.0	0.0	2:15.92	events/0

Como é possível observar pela tabela acima existem 2 processos a executar neste nó e um deles é o próprio comando.

Durante a execução o TOP já devolvia os seguintes valores:

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
24064 pg27715   20   0  6984 3368  364 R 43.3  0.0   0:03.69 app
    4 root      20   0     0    0    0 S 29.5  0.0   2571:21 ksoftirqd/0
23957 root      20   0 27836 8112   72 S 23.6  0.1   0:03.42 pbs_mom
   36 root      20   0     0    0    0 S  7.9  0.0   6:16.40 events/1
   39 root      20   0     0    0    0 S  5.9  0.0   3:10.84 events/4
   21 root      20   0     0    0    0 S  2.0  0.0 207:19.14 ksoftirqd/4
23956 root      20   0 27836 8720  680 S  2.0  0.1   0:00.29 pbs_mom
    1 root      20   0 19348  552  328 S  0.0  0.0   0:00.94 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.04 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:47.51 migration/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.03 migration/0
    6 root      RT   0     0    0    0 S  0.0  0.0   0:03.45 watchdog/0
    7 root      RT   0     0    0    0 S  0.0  0.0   0:31.86 migration/1
    8 root      RT   0     0    0    0 S  0.0  0.0   0:00.03 migration/1
    9 root      20   0     0    0    0 S  0.0  0.0 732:29.19 ksoftirqd/1
   10 root      RT   0     0    0    0 S  0.0  0.0   0:02.77 watchdog/1
   11 root      RT   0     0    0    0 S  0.0  0.0   1:39.93 migration/2
   12 root      RT   0     0    0    0 S  0.0  0.0   0:00.03 migration/2
   13 root      20   0     0    0    0 S  0.0  0.0 491:16.04 ksoftirqd/2
   14 root      RT   0     0    0    0 S  0.0  0.0   0:04.74 watchdog/2
   15 root      RT   0     0    0    0 S  0.0  0.0   0:26.62 migration/3
   16 root      RT   0     0    0    0 S  0.0  0.0   0:00.03 migration/3
   17 root      20   0     0    0    0 S  0.0  0.0 375:20.97 ksoftirqd/3
   18 root      RT   0     0    0    0 S  0.0  0.0   0:02.62 watchdog/3
   19 root      RT   0     0    0    0 S  0.0  0.0   0:13.27 migration/4
   20 root      RT   0     0    0    0 S  0.0  0.0   0:00.03 migration/4
   22 root      RT   0     0    0    0 S  0.0  0.0   0:03.00 watchdog/4
   23 root      RT   0     0    0    0 S  0.0  0.0   0:11.82 migration/5
   24 root      RT   0     0    0    0 S  0.0  0.0   0:00.03 migration/5
   25 root      20   0     0    0    0 S  0.0  0.0 268:23.98 ksoftirqd/5
   26 root      RT   0     0    0    0 S  0.0  0.0   0:02.95 watchdog/5
   27 root      RT   0     0    0    0 S  0.0  0.0   0:03.03 migration/6
   28 root      RT   0     0    0    0 S  0.0  0.0   0:00.03 migration/6
   29 root      20   0     0    0    0 S  0.0  0.0 218:41.67 ksoftirqd/6
   30 root      RT   0     0    0    0 S  0.0  0.0   0:02.51 watchdog/6
   31 root      RT   0     0    0    0 S  0.0  0.0   0:06.49 migration/7
   32 root      RT   0     0    0    0 S  0.0  0.0   0:00.03 migration/7
```

Quando foi colocada a aplicação a executar esta passou a utilizar grande parte dos recursos deste nó, é possível observar na primeira linha, a linha correspondente à aplicação, que 43.3 % do CPU passou a ser utilizado pela aplicação e a memória não sofreu qualquer alteração. Era esperada aqui uma maior utilização de CPU mas isto deve-se ao facto de a medição ter sido feita no inicio da execução. Um dado curioso é o associado ao VIRT que se trata da quantidade de memoria virtual utilizada pela aplicação e ainda o RES que se trata da memória permanente utilizada pela aplicação.

PS

Antes da aplicação estar a decorrer os dados do PS AUX eram os seguintes:

```
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
pg27715   23884  0.0  0.0 108444  1876 pts/0    S    12:43   0:00 -bash
pg27715   23958  0.0  0.0 108448  1936 pts/1    S+   12:45   0:00 -bash
pg27715   24068  1.0  0.0 110236  1116 pts/0    R+   12:48   0:00 ps aux
```

Observando a tabela acima conclui-se que o utilizador não tinha nenhuma aplicação dele a decorrer. As únicas tarefas da parte do utilizador eram a própria bash e ainda a chamada do comando ps aux. Esta tabela foi obtida com o 'ps aux' e depois foram filtrados os processos do utilizador para que esta não fosse muito extensa.

Durante a execução o PS AUX já devolvia os seguintes valores:

```
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
pg27715  23884  0.0  0.0 108444  1876 pts/0    S    12:43   0:00 –bash
pg27715  23958  0.0  0.0 108448  1936 pts/1    S    12:45   0:00 –bash
pg27715  24069 80.7  0.0   6984  3364 pts/1    R+   12:48   0:03 ./app
pg27715  24070  0.0  0.0 110240  1120 pts/0    R+   12:48   0:00 ps aux
```

Enquanto decorria a aplicação é possível olhar para a terceira linha de resultados e ver que a aplicação estava a utilizar aproximadamente 81% do CPU.
Um facto interessante a visualizar corresponde a coluna STAT que apresenta o estado da execução neste caso o S por parte da bash significa que se encontra a espera que um evento conclua e o R por parte da aplicação e da chamada do comando significam que está a executar.

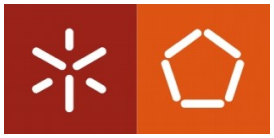
Conclusão

Foi uma boa experiencia utilizar estes comandos que estão em qualquer sistema LINUX. São comandos extremamente úteis na medida que nos podem auxiliar na monitorização de uma aplicação num ambiente. E saber se esse ambiente é o ideal para ela. São ferramentas que tem grande potencial podendo ser uma ajuda enorme no desenvolvimento de software na medida melhorar a adaptação de uma aplicação a um determinado sistema no objectivo de obter melhores desempenhos.

IOZONE

Engenharia de Sistemas da Computação
Computação Paralela Distribuída

Mestrado em Engenharia Informática
Universidade do Minho



Duarte Nuno Ferreira Duarte
pg27715

Índice

Introdução

Com a realização deste trabalho pretende-se obter um ponto de comparação entre os vários discos do *cluster*. Para tal foi utilizado o ***IOZONE*** que se trata de uma *benchmark* para sistemas de ficheiros. Em suma, o ***IOZONE*** permite que se avalie o desempenho do sitema de ficheiros para ficheiros de vários tamanhos.

1. Caracterização do Sistema

Para os testes feitos foram utilizados vários nós do *Search*, cada um associado a um disco diferente. O *Search* apresenta 9 tipos de discos diferentes e para tal seria necessário utilizar nove nós para os testes, isto não se verificou porque um dos nós que apresentava um disco diferente estava inacessível. Os 9 discos diferentes são:

INTEL_SSDSC2BW240A4
MB0500EBNCR
INTEL_SSDSC2BW120A4
INTEL_SSDSC2BW240A3F
SAMSUNG_HD502HJ
ST3120827AS
WDC_WD10EZRX-00A8LB0
WDC_WD20NPVT-00Z2TT0
SAMSUNG_HD502HI
MM0500EBKAE

Agora através do tentakel foi possivel obter a relação nó-disco usando o seguinte comando:

```
tentakel -g compute_linux
"/sbin/udevadm info -a -p
/sys/class/block/sda/sda5 -q env
| grep MODEL"
```

Assim sendo a relação obtida foi a seguinte:

Disco	Nó
INTEL_SSDSC2BW240A4	641-19
MB0500EBNCR	431-3
INTEL_SSDSC2BW120A4	662-6
INTEL_SSDSC2BW240A3F	641-8
SAMSUNG_HD502HJ	431-6
ST3120827AS	---------
WDC_WD10EZRX-00A8LB0	541-1
WDC_WD20NPVT-00Z2TT0	652-1
SAMSUNG_HD502HI	431-5
MM0500EBKAE	432-1

2. Definições Utilizadas

a. Versão

Na realização destes testes foi utilizada a versão 3.397 do **IOZONE**.

b. Esquema

Para os testes não entrarem em conflito foi necessário atribuir diferentes nomes aos ficheiros temporários criados pelo **IOZONE**. Para isto o nome atribuído foi simplesmente o nome do disco.

c. Tamanhos

Foram feitos dois testes a cada disco, um para 512 MB e um para 8GB.

Assim sendo um comando exemplo é o seguinte:

```
iozone -Ra -b
INTEL_SSDSC2BW240A3F.xls -f
INTEL_SSDSC2BW240A3F.tmp

iozone -Ra -g 8G -b
INTEL_SSDSC2BW240A3F-8gb.xls -f
INTEL_SSDSC2BW240A3F-8gb.tmp
```

d. Testes

Os testes que foram obtidos são os seguintes:

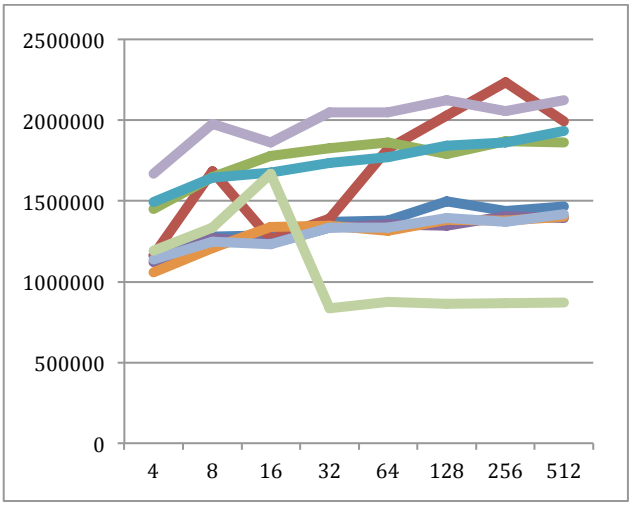
1 Write
2 Rewrite
3 Read
4 Reread
5 Random read
6 Random write
7 Backward read
8 Record rewrite
9 Stride read
10 Fwrite
11 Frewrite
12 Fread
13 Freread

3. Resutados

Doravante a legenda dos gráficos é a seguinte:

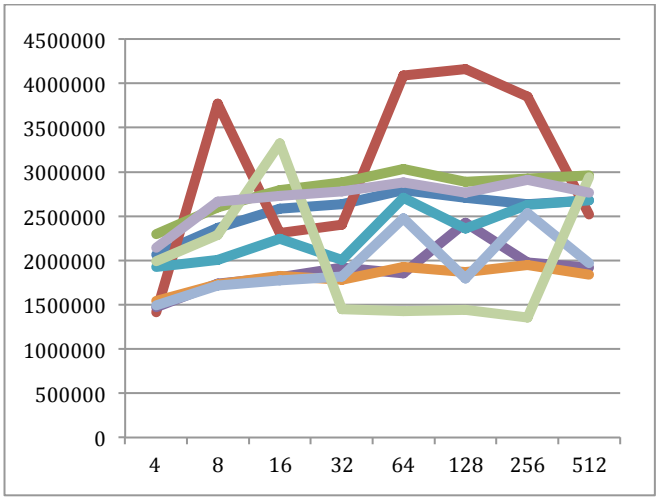
- INTEL_SSDSC2BW120A4
- INTEL_SSDSC2BW240A3F
- INTEL_SSDSC2BW240A4
- MB0500EBNCR
- MM0500EBKAE
- SAMSUNG_HD502HI
- SAMSUNG_HD502HJ
- ST3120827AS
- WDC_WD10EZRX-00A8LB0
- WDC_WD20NPVT-00Z2TT0

a. Write



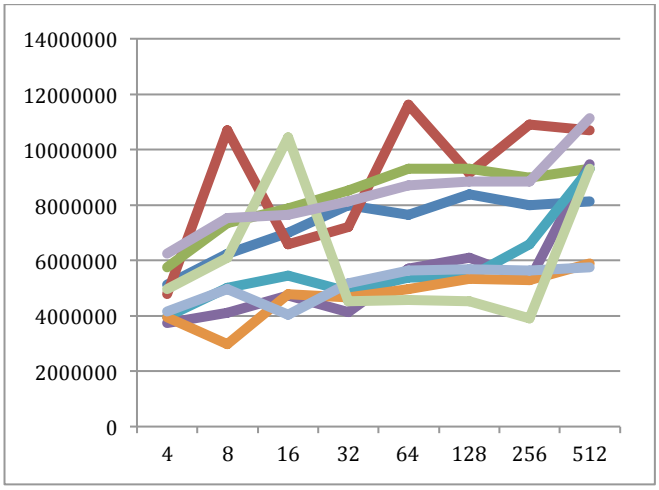
Ao nível das escritas pode-se observar recorrendo ao gráfico que o mais constante é WDC_WD20NPVT_00Z2TT0 mas que também o INTEL_SSDSC2BW240A3F à medida que o tamanho das escritas aumenta também vai "lutar" por um lugar no topo. Neste teste sem duvida que estes dois são os melhores.

b. Rewrite



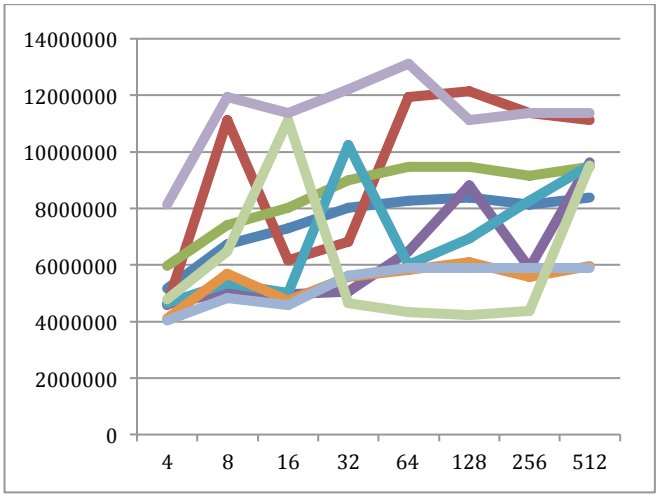
Ao nível das re-escritas pode-se observar que o INTEL_SSDSC2BW240A3F ganha por grande diferença.

c. Read



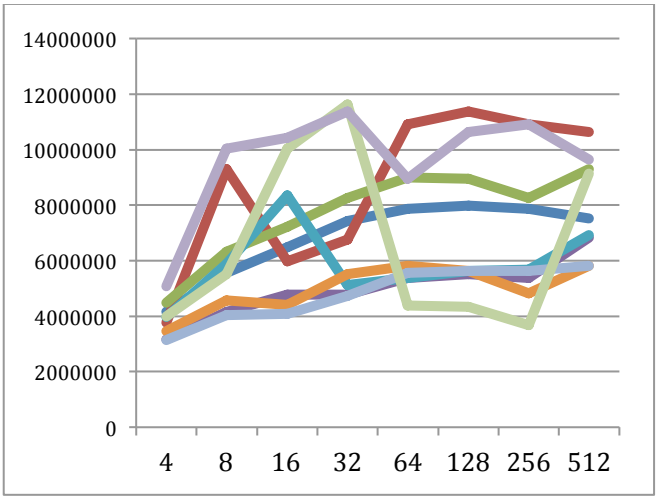
Ao nível das leituras o INTEL_SSDSC2BW240A3F ganha aos outros mas neste teste o INTEL_SSDSC2BW240A4 também obtém resultados bastante bons.

d. Reread



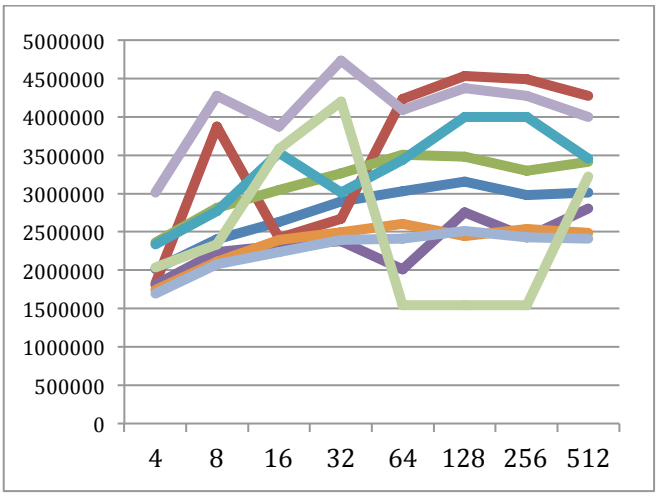
Neste teste para leituras pequenas o disco WDC_WD20NPVT_00Z2TT0 obtém resultados muito bons. Mas à medida que o tamanho das leituras aumenta uma vez mais o disco INTEL_SSDSC2BW240A3F assume uma posição de destaque.

e. Random read



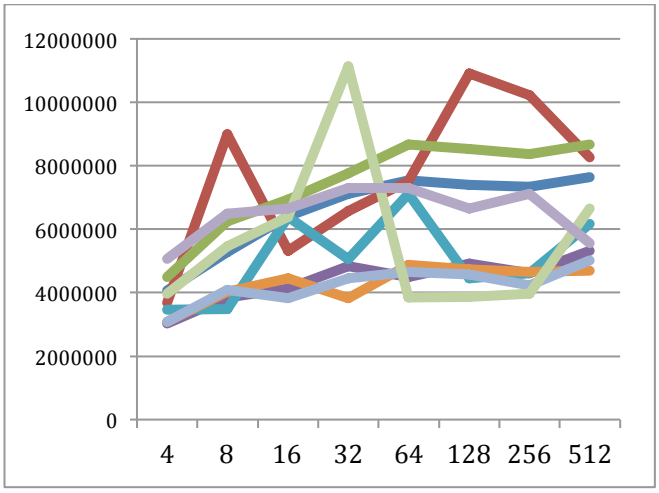
Neste teste para leituras aleatórias pequenas o disco WDC_WD20NPVT_00Z2TT0 obtém resultados bons. Mas uma vez mais à medida que o tamanho das leituras aleatórias aumenta aparece o INTEL_SSDSC2BW240A3F.

f. Random write



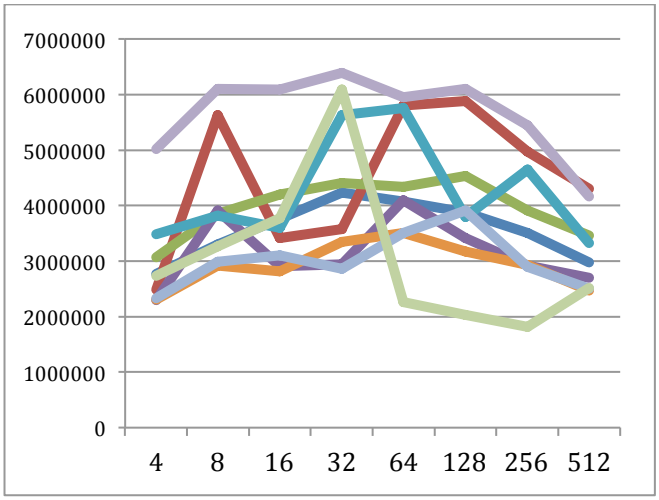
Neste teste para escritas aleatórias o disco WDC_WD20NPVT_00Z2TT0 obtém resultados bastante bons para todos os tamanhos apenas sendo ultrapassado para escritas maiores e mesmo assim por pouco pelo INTEL_SSDSC2BW240A3F.

g. Backward read



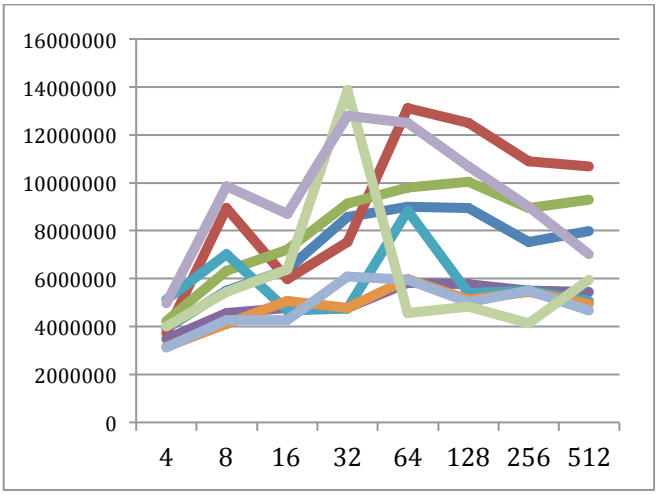
Neste teste para leituras "em sentido contrário" claramente o melhor é o INTEL_SSDSC2BW240A3F.

h. Record rewrite



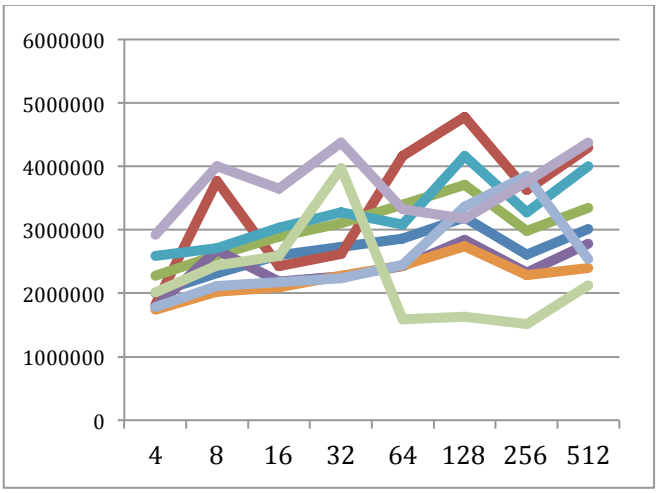
Neste teste para re-escritas claramente o melhor é o WDC_WD20NPVT_00Z2TT0 ao longo de todos os tamanhos mas é possível observar que para maiores escritas o INTEL_SSDSC2BW240A3F começa a disputar um lugar de topo.

i. Stride read



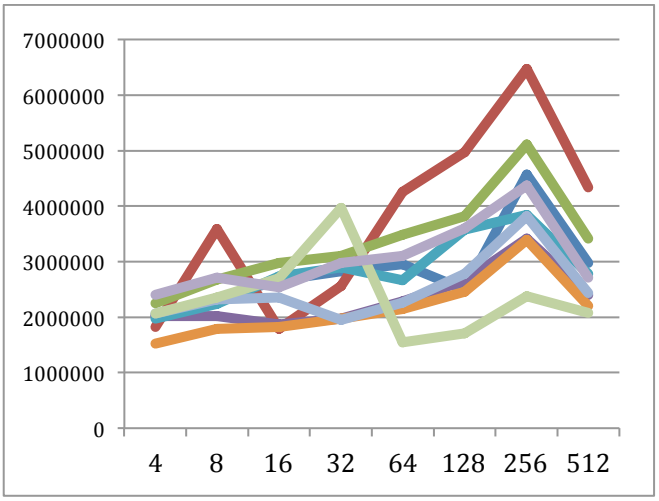
Neste teste para leituras com um paço fixo o INTEL_SSDSC2BW240A3F é o melhor.

j. Fwrite



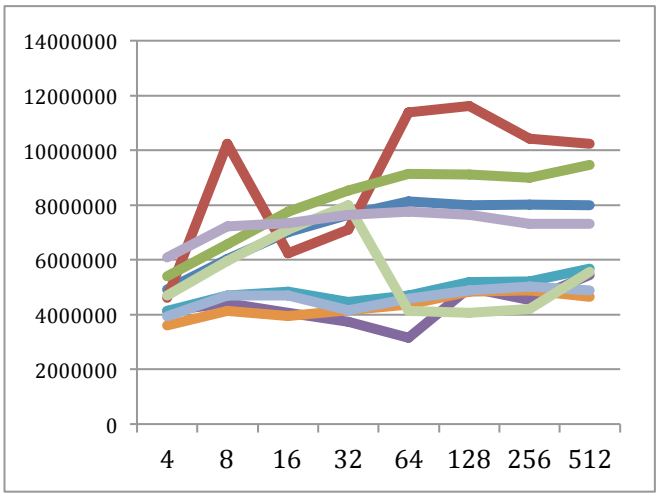
Neste teste para escritas para ficheiro o INTEL_SSDSC2BW240A3F é o melhor com o WDC_WD20NPVT_00Z2TT0 a obter também bons resultados.

k. Frewrite



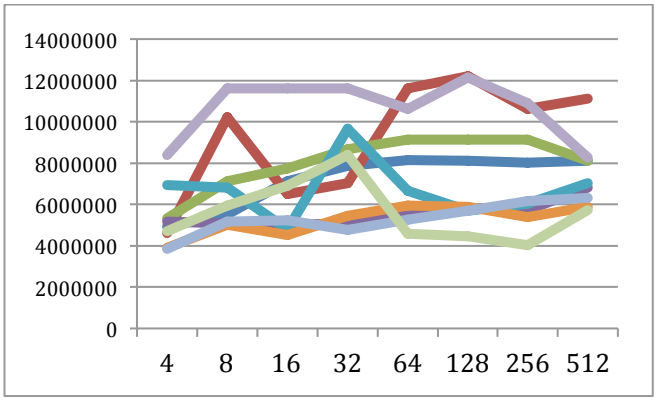
Neste teste para re-escritas para ficheiro o INTEL_SSDSC2BW240A3F é o melhor com grande destaque.

l. Fread



Neste teste para leituras de ficheiro o INTEL_SSDSC2BW240A3F é o melhor com grande diferença.

m. Freread



Para re-leituras de ficheiro o disco que obtém melhores resultados é o WDC_WD20NPVT_00Z2TT0 sendo constante para todos os tamanhos. Mas o disco INTEL_SSDSC2BW240A3F para valores mais altos obtém muito bons resultados.

4. Comentário dos resultados

Depois de observados estes resultados pode-se chegar à conclusão que os testes são relativamente homogéneos em relação aos discos. Os discos que obtêm melhores resultados são em quase todos os testes o

WDC_WD20NPVT_00Z2TT0 e o INTEL_SSDSC2BW240A3F. Como era de esperar um disco SSD teria claramente vantagem neste tipo de testes.

Obtenção dos resultados

a. Esquema de funcionamento

Os testes obtidos utilizando o IOZONE eram colocados em pastas especificas, os testes sem tamanho definido eram colocados numa pasta com o nome 0GB e os com tamanho definido para 8Gb eram colocados numa pasta com o nome 8GB.

b. Scripts utilizados

i. cpi.pbs

```
##!/bin/bash
#PBS -l walltime=03:00:00
#PBS -j oe
#PBS -N IOZONE
cd ~/esc_iozone/

/opt/iozone/bin/iozone -Ra -b
INTEL_SSDSC2BW240A3F-normal.xls -f
INTEL_SSDSC2BW240A3F-ntmp.tmp
/opt/iozone/bin/iozone -Ra -g 8G -b
INTEL_SSDSC2BW240A3F-128.xls -f
INTEL_SSDSC2BW240A3F-1tmp.tmp
```

Este cpi limita-se a ser utilizado com o qsub de forma a "angariar" um nodo de execução especifico e executar o comando que este contém.

c. Configurações

ii. conf.txt

```
set term png
set output 'file.png'
set logscale x 2
set logscale y 2
set autoscale z
set grid lt 2 lw 1
set zlabel 'Kbytes/sec'
set style data lines
set dgrid3d 80,80,3
splot 'file.gnuplot' title 'Write'
quit
```

Esta foi a configuração do gnuplot para obtenção dos gráficos.

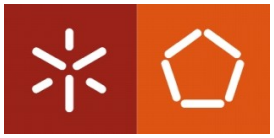
Conclusão

Com a realização deste trabalho foi possível, como era esperado, verificar que o disco SSD obtém melhores resultados. Achei estranho um dos discos SSD não ter resultados melhores dos que obteve. Este trabalho foi bastante útil na medida em que permite utilizar novas ferramentas que foram abordadas nesta unidade curricular.

Strace

Engenharia de Sistemas da Computação
Computação Paralela Distribuída

Mestrado em Engenharia Informática
Universidade do Minho



Duarte Nuno Ferreira Duarte
pg27715

Índice

Introdução

Com a realização deste trabalho pretende-se analisar o padrão de execução das aplicações ao nível dos traços que elas deixam no sistema. Monitorizar as chamadas aos sistema por parte do programa a executar.

1. Strace

Para a realização da análise foi usada a versão do search6 que é a 4.5.19. Como descrito no enunciado deste trabalho foi criado um *job* para executar o seguinte comando:

```
strace –T –ttt –o strace.out /opt/iozone/bin/iozone –R –a –i0 –i1 –i2 –i5 –g 256M
```

De seguida foi utilizado um script auxiliar para que os ficheiros fossem processados para se obter uma leitura mais fácil. Para isso foi utilizado o *refazStFd.py* desenvolvido pelo professor da Unidade Curricular. Por fim o ficheiro resultante da passagem pelo script é analisado pelo strace_analyzer de forma a serem obtidos resultados sobre a execução do strace. Estes dois últimos processos foram chamados da seguinte forma:

```
/share/apps/IOAPPS/refazStFd.py < strace.out > tmpfile
/share/apps/IOAPPS/strace_analyzer_ng_0.09.pl tmpfile > statistics.txt
```

2. Resultados obtidos

a. Tempo

A execução deste comando demorou aproximadamente 172 segundos e deste tempo 161 segundos foram de IO o que corresponde a uma percentagem de 93.5%.

```
----------------
-- Time Stats --
----------------
Elapsed Time for run: 172.335752 (secs)
Total IO Time: 161.151279 (secs)
Total IO Time Counter: 323107
   Percentage of Total Time = 93.510068%
```

b. Operações de IO

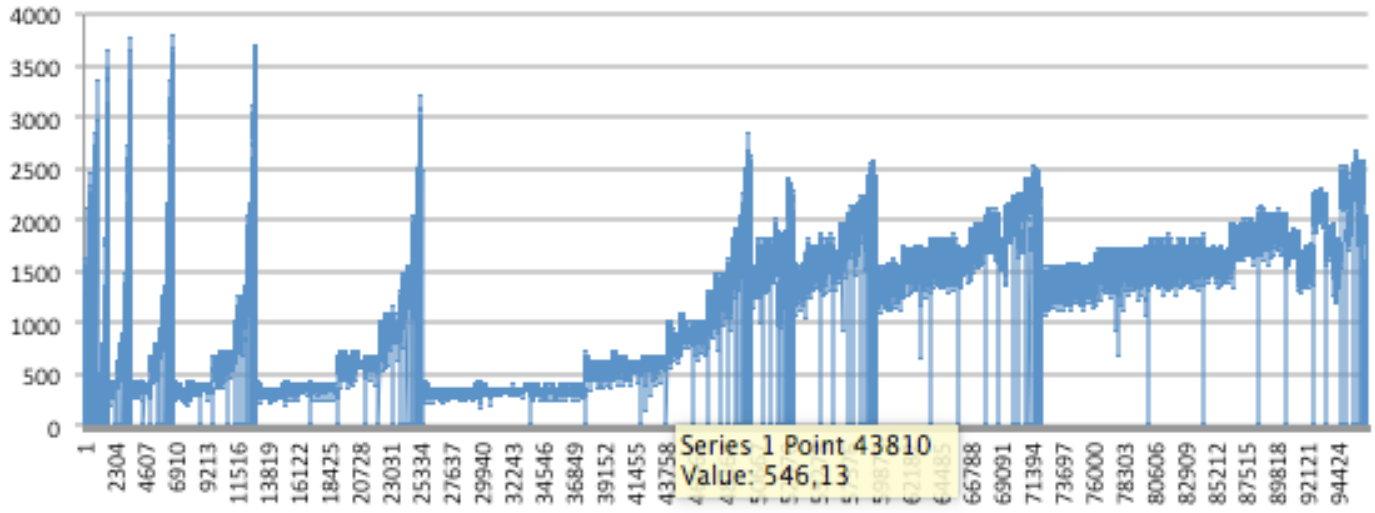
Foram obtidos resultados relativamente as chamadas de sistema, esses resultados encontram-se apresentados de seguida. Como era expectável para este comando chamadas de *read* e *write* obtiveram as contagens mais altas.

```
----------------------
-- IO Command Count --
----------------------
Command           Count
======================
access                1
lseek             95254
stat                352
unlink              234
open                941
close              1175
creat               117
fstat                 6
fsync              1404
read             126931
write             96692
```

c. Bandas utilizadas

Foram compilados os resultados relativos à utilização da banda por parte das chamadas de leitura e escrita. Estes resultados são relativos a muitas linhas nos resultados obtidos pelo que os gráficos têm valores relativos a muitas medições de tempo.

i. Banda da utilização do write



O gráfico acima ilustra a utilização da banda por parte da escrita ao longo do tempo de execução. Na tabela seguinte é possível observar o tamanho dos blocos para as chamas ao sistema no que diz respeito à escrita.

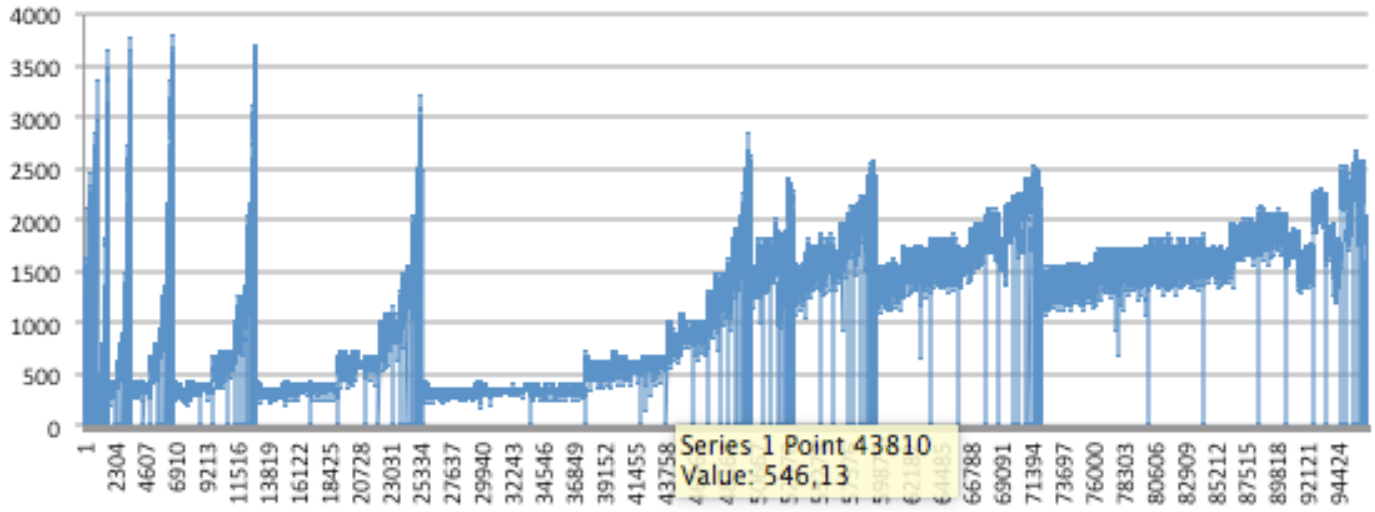
```
-- File sizes for write() syscall --

IO Size Range                       Number of syscalls
=======================================================
(  1)    0KB <   <    1KB                    1673
(  2)    1KB <   <    8KB                   24528
(  3)    8KB <   <   32KB                   18396
(  4)   32KB <   <  128KB                   27639
(  5)  128KB <   <  256KB                   12285
(  6)  256KB <   <  512KB                    6141
(  7)  512KB <   < 1000KB                    3069
(  8) 1000KB <   <   10MB                    2868
(  9)   10MB <   <  100MB                      93
( 10)  100MB <   <    1GB                       0
( 11)    1GB <   <   10GB                       0
( 12)   10GB <   <  100GB                       0
( 13)  100GB <   <    1TB                       0
( 14)    1TB <   <   10TB                       0
```

Os resultados obtidos produziram umas estatísticas em relação à chamada do write. Essas estatísticas estão apresentadas na tabela seguinte:

```
-- WRITE SUMMARY --
Total number of Bytes written = 14,796,931,660 (14,796.93166  MB)
Number of Write syscalls =   96,692

Average (mean) Bytes per syscall = 153,031.60199396 (bytes) (0.15303160199396  MB)
   Standard Deviation: 723,070.151693487 (bytes) (0.723070151693487  MB)
   Mean Absolute Deviation: 690,307.763137929 (bytes) (0.690307763137929  MB)
Median Bytes per syscall =        65,536 (bytes) (0.065536 MB)
   Median Absolute Deviation: 143,793.095974848 (bytes) (0.143793095974848  MB)

Time for slowest write syscall (secs) = 0.014625
   Line location in file: 323294

Smallest write syscall size: 1 (Bytes)
Largest write syscall size: 16777216 (Bytes)
```

Foram escritos cerca de 15GB em aproximadamente 97 mil chamadas de escrita. A média de bytes foi de 153 KB por chamada ao sistema. A mediana encontra-se nos 66 KB por chamada.

ii. Banda da utilização do read



O gráfico acima ilustra a utilização da banda por parte da leitura ao longo do tempo de execução. Na tabela seguinte é possível observar o tamanho dos blocos para as chamas ao sistema no que diz respeito à leitura.

```
-- File sizes for read() syscalls --

IO Size Range                    Number of syscalls
=======================================================
(  1)    0KB <   <    1KB                    3
(  2)    1KB <   <    8KB                32724
(  3)    8KB <   <   32KB                24564
(  4)   32KB <   <  128KB                36896
(  5)  128KB <   <  256KB                16404
(  6)  256KB <   <  512KB                 8210
(  7)  512KB <   < 1000KB                 4112
(  8) 1000KB <   <   10MB                 3884
(  9)   10MB <   <  100MB                  134
( 10)  100MB <   <    1GB                    0
( 11)    1GB <   <   10GB                    0
( 12)   10GB <   <  100GB                    0
( 13)  100GB <   <    1TB                    0
( 14)    1TB <   <   10TB                    0
```

Os resultados obtidos produziram umas estatísticas em relação à chamada do read. Essas estatísticas estão apresentadas na tabela seguinte:

```
-- READ SUMMARY --
Total number of Bytes read = 20,131,020,718 (20,131.020718  MB)
Number of Read syscalls =  126,931

Average (mean) Bytes per syscall = 158,598.141651764 (bytes) (0.158598141651764  MB)
   Standard Deviation: 749,136.288122469 (bytes) (0.749136288122469  MB)
   Mean Absolute Deviation: 716,227.68995956 (bytes) (0.71622768995956  MB)
Median Bytes per syscall =        65,536 (bytes) (0.065536 MB)
   Median Absolute Deviation: 148,001.871520747 (bytes) (0.148001871520747  MB)

Time for slowest read syscall (secs) = 0.006392
   Line location in file: 323388

Smallest read syscall size: 832 (Bytes)
Largest read syscall size: 16777216 (Bytes)
```

Foram lidos cerca de 20GB em aproximadamente 27 mil chamadas de leitura. A média de bytes foi de 159 KB por chamada ao sistema. A mediana encontra-se nos 65 KB por chamada.

iii. Open

3. Conclusão

Esta ferramenta abordada nesta semana mostrou que pode vir a ser bastante útil, é uma ferramenta muito pesada mas que mostra as chamadas todas aos sistema por parte da aplicação que monitoriza. É uma ferramenta que tem potencial e pode vir a ser utilizada no futuro no teste de outras aplicações.

4. Obtenção dos resultados

a. Scripts utilizados

i. cpi.pbs

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

strace -T -ttt -o strace.out
/opt/iozone/bin/iozone -R -a -i0 -i1
-i2 -i5 -g 256M
```

Este cpi limita-se a ser utilizado com o qsub de forma a "angariar" um nodo de execução e executar o strace.

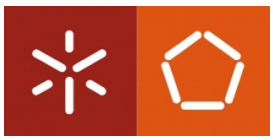
Perf

Engenharia de Sistemas da Computação

Computação Paralela Distribuída

Mestrado em Engenharia Informática

Universidade do Minho



Duarte Nuno Ferreira Duarte

pg27715

Índice

Introdução

Este trabalho demonstra a utilização do perf num programa que multiplica duas matrizes de forma muito básica e sem qualquer optimização. O perf é poderoso, consegue contadores de desempenho CPU e tracepoints entre outros. Está incluído no kernel do Linux e é regularmente atualizado.

O perf começou como uma ferramenta para usar o subsistema de contadores de desempenho no Linux, e tem tido várias melhorias.

Código de teste

O código apresentado de seguida é o código que será utilizado pelo perf.

```
void multiply_matrices() {
        int i, j, k ;
        for (i = 0 ; i < MSIZE ; i++) {
                for (j = 0 ; j < MSIZE ; j++) {
                        float sum = 0.0 ;
                        for (k = 0 ; k < MSIZE ; k++) {
                             sum = sum + (matrix_a[i][k] *
matrix_b[k][j]);
                }
               matrix_r[i][j] = sum ;
                }
        }
}
```

Como é possível ver o código é o mais básico que existe de multiplicação de matrizes não possui qualquer tipo de otimização mas faz o esperado.

Eventos

De seguida são apresentados os eventos suportados no compute-431-10.local. Existem três tipos de eventos medidos pelo perf, os eventos de hardware, os eventos de software e ainda os eventos de cache de hardware.

cpu-cycles OR cycles	[Hardware event]
instructions	[Hardware event]
cache-references	[Hardware event]
cache-misses	[Hardware event]
branch-instructions	[Hardware event]
branch-misses	[Hardware event]
stalled-cycles-frontend	[Hardware event]
stalled-cycles-backend	[Hardware event]

Esta tabela mostra os eventos de hardware suportados neste nó do cluster e a azul escuro (bold) são os eventos que serão utilizados nas medições.

cpu-clock	[Software event]
task-clock	[Software event]
page-faults OR faults	[Software event]
context-switches OR cs	[Software event]
cpu-migrations OR migrations	[Software event]
minor-faults	[Software event]
major-faults	[Software event]
alignment-faults	[Software event]
emulation-faults	[Software event]

Esta tabela mostra os eventos de software suportados neste nó do cluster.

L1-dcache-loads	[Hardware cache event]
L1-dcache-load-misses	[Hardware cache event]
L1-dcache-stores	[Hardware cache event]
L1-dcache-store-misses	[Hardware cache event]
L1-dcache-prefetches	[Hardware cache event]
L1-icache-loads	[Hardware cache event]
L1-icache-load-misses	[Hardware cache event]
LLC-loads	[Hardware cache event]
LLC-load-misses	[Hardware cache event]
LLC-stores	[Hardware cache event]
LLC-store-misses	[Hardware cache event]
dTLB-loads	[Hardware cache event]
dTLB-load-misses	[Hardware cache event]
dTLB-stores	[Hardware cache event]
dTLB-store-misses	[Hardware cache event]
iTLB-loads	[Hardware cache event]

iTLB-load-misses	[Hardware cache event]
branch-loads	[Hardware cache event]
branch-load-misses	[Hardware cache event]

Esta tabela mostra os eventos de cache de hardware.

Eventos medidos

Com os contadores de eventos que foram usados e foram vistos acima foi possível obter os seguintes resultados.

230263545	cpu-cycles	2,735 GHz
332099971	instructions	1,46 insns per cycle
84,193928	task-clock (msec)	0,802 CPUs utilized
14	context-switches	0,166 K/sec
0	cpu-migrations	0,000 K/sec
843	page-faults	0,010 M/sec
224838277	cycles	2,670 GHz
89984496	stalled-cycles-frontend	39,54% frontend cycles idle
21576052	stalled-cycles-backend	9,48% backend cycles idle
342143914	instructions	1,50 insns per cycle
40157203	branches	476,961 M/sec
69956	branch-misses	0,17% of all branches
16598576	L1-dcache-load-misses	197,147 M/sec

Como é possível observar não houve qualquer cpu-migration pois usou sempre o mesmo CPU durante a execução. A apontar também o reduzido numero de saltos por falha (branch-misses) que ocorre apenas no fim de cada multiplicação parcial.

Perfis de execução

São agora apresentados os perfis de execução. Um obtido a partir do ambiente interativo oferecido pelo perf e outro a partir do output gerado.

```
# Overhead  Command      Shared Object
# ........  .......  ..................
#
    90.33%    naive  naive
     6.95%    naive  libc-2.12.so
     2.72%    naive  [kernel.kallsyms]


# Samples: 20  of event 'faults'
# Event count (approx.): 909
#
# Overhead  Command      Shared Object
# ........  .......  ..................
#
    84.93%    naive  naive
    14.52%    naive  ld-2.12.so
     0.33%    naive  [kernel.kallsyms]
     0.22%    naive  libc-2.12.so
```

Este foi obtido através do output do perf e mostra claramente que a multiplicação naíve usa 90.33% como seria de esperar visto que é o programa a ser testado.

Overhead	Cmd	Shared	Object
89,02%	naive	naive	[.] main
3,66%	naive	libc-2.12.so	[.] __random_r
2,20%	naive	libc-2.12.so	[.] __random
1,76%	naive	[kernel.kallsyms]	[k] __do_softirq
0,73%	naive	[kernel.kallsyms]	[k] __do_page_fault
0,44%	naive	naive	[.] rand@plt
0,29%	naive	[kernel.kallsyms]	[k] clear_page_c
0,29%	naive	libc-2.12.so	[.] rand
0,15%	naive	[kernel.kallsyms]	[k] ____pagevec_lru_add
0,15%	naive	[kernel.kallsyms]	[k] __audit_syscall_exit
0,15%	naive	[kernel.kallsyms]	[k] __bitmap_empty
0,15%	naive	[kernel.kallsyms]	[k] __lru_cache_add
0,15%	naive	[kernel.kallsyms]	[k] __mem_cgroup_uncharge_common
0,15%	naive	[kernel.kallsyms]	[k] __percpu_counter_add
0,15%	naive	[kernel.kallsyms]	[k] _atomic_dec_and_lock
0,15%	naive	[kernel.kallsyms]	[k] _spin_lock
0,15%	naive	[kernel.kallsyms]	[k] anon_vma_prepare
0,15%	naive	[kernel.kallsyms]	[k] handle_pte_fault
0,15%	naive	[kernel.kallsyms]	[k] sys_open

Agora no ambiente interativo foi possível obter esta tabela que demonstra o esperado, a multiplicação de matrizes a ocupar 89,02% do tempo e de realçar os 3,66% da inicialização com valores aleatórios da matriz.

```
 23,85 │150:   movaps %xmm2,%xmm0
       │       movlps (%rdx),%xmm0
  0,33 │       movhps 0x8(%rdx),%xmm0
  0,33 │       add    $0x4,%rdx
  4,11 │       shufps $0x0,%xmm0,%xmm0
       │       mulps  (%rcx,%rax,1),%xmm0
 47,70 │       add    $0x7d0,%rax
 10,03 │       cmp    $0xf4240,%rax
       │       addps  %xmm0,%xmm1
  9,38 │     ↑ jne    150
```

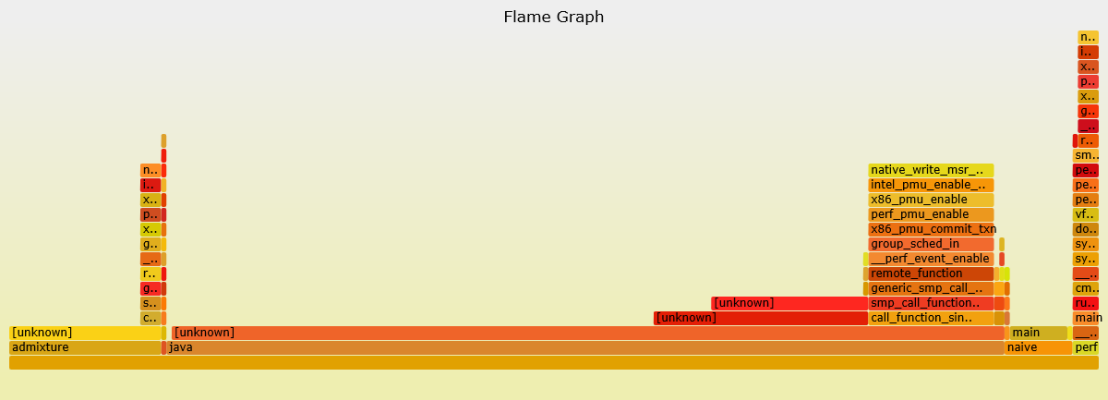
Observando mais especificamente com o perf annotate é possível ver que 47,70 é na soma dos valores da matriz que é apresentado no excerto de código seguinte.

```
sum = sum + (matrix_a[i][k] * matrix_b[k][j]) ;
```

Depois tem 10,03 na comparação para saber se termina o ciclo e 9,38 no salto caso saia do ciclo.

Flame Graph

Foi usado o software flamegraph que permite gerar gráficos a partir de medições do perf e não só. O objectivo era ver o impacto da aplicação no sistema e o impacto é agora apresentado.



Como é possível ver, com alguma dificuldade, a execução da aplicação aparece com o nome naíve na segunda linha a contar do fim do lado direito. A aplicação não tem grande impacto no sistema.

Conclusão

Foi uma boa experiencia utilizar o perf num caso prático. É uma ferramenta que tem grande potencial podendo ser uma ajuda enorme no desenvolvimento de software na medida em que contribui para ver os “hotspots” da execução. Assim é possível saber o que melhorar na aplicação de forma a obter melhores desempenhos.