ESC - Benchmarking SeARCH with NAS Parallel Benchmarks

Pedro Barbosa - A82068 and José Pinto - A81317

I. Introdução

Para a grande maioria das linguagens de programação encontram-se disponíveis vários compiladores. Para além da acessibilidade(preço, plataforma, etc) uma questão importante é o desempenho do programa compilado.

Em norma, cada compilador disponibiliza um conjunto de optimizações. A eficácia das optimizações de cada compilador pode variar conforme vários fatores, como a arquitetura do *hardware* e as características do programa.

No caso do Fortran, dois dos compiladores mais usados são o IFORT da Intel e o gFortran da GNU.

A primeira parte deste trabalho lida com estes aspetos e analisa o impacto destes no desempenho dos *benchmarks*.

É importante avaliar métricas para além do tempo de execução. A monitorização da utilização de recursos físicos, desde o processador à rede, permite avaliar se estes estão a ser usados de uma forma óptima e encontrar possíveis gargalos de desempenho.

A segunda parte lida com a utilização de ferramentas para obter e processar estas métricas.

II. Benchmarks

Os *benchmarks* testados são os SP-MZ, BT-MZ e LU-MZ da versão NPB 3.3.1-MZ.

III. SER

A. Flags

As *flags* -xHost (IFORT) e -march=native (gfortran) foram utilizadas com o intuito de optimizar o código para o processador onde foi compilado.[https://software.intel.com/en-us/fortran-compiler-developer-guide-and-reference-xhost-qxhost][https://gcc.gnu.org/onlinedocs/gcc/x86-Options.html]

A *flag* -ipo, apenas disponivel para o compilador IFORT, permite ao compilador aplicar a *Interprocedural Optimization* (IPO). Esta funcionalidade analisa o programa na totalidade, em vez de um segmento de código específico, permitindo optimizações como *inlining* (substituir chamada da função pelo código da função) e análise de *alias* (determinar se dois apontadores apontam para sítios distintos, o que permite executar instruções fora de ordem). [https://software.intel.com/en-us/fortran-compiler-developer-guide-and-reference-ipo-qipo] [https://software.intel.com/en-us/fortran-compiler-developer-guide-and-reference-interprocedural-optimization-ipo]

B. Medições

As seguintes tabelas apresentam o desempenho de cada compilador, com as diferentes *flags*, para os diferentes *benchmarks*. Os resultados obtidos são uma média de 5 execuções obtidas nos nodos 641 e 431 do SeARCH.

SP	**gfortran**			**ifort**		
	-O0	-O3	-O3 -march=native	-O0	-O3	-O3 -xHost -ipo
431	914	139	139	923	114	114
641	619	102	94	783	79	74

TABLE I
Medições feitas em segundos para o *benchmark* SP

BT	**gfortran**			**ifort**		
	-O0	-O3	-O3 -march=native	-O0	-O3	-O3 -xHost -ipo
431	1023	221	223	1052	194	179
641	719	192	158	789	151	123

TABLE II
Medições feitas em segundos para o *benchmark* BT

LU	**gfortran**			**ifort**		
	-O0	-O3	-O3 -march=native	-O0	-O3	-O3 -xHost -ipo
431	888	149	150	1451	147	144
641	648	115	111	1314	144	112

TABLE III
Medições feitas em segundos para o *benchmark* LU

Como é possível verificar através das tabelas, os resultados obtidos com a utilização de *flags* chegam a ser 11,7 vezes mais rápidos, para o caso do compilador IFORT da versão LU-MZ no nodo 641, em relação a versões sem qualquer optimização.

A eficácia das *flags* extra varia conforme o *benchmark*. Para qualquer aplicação é sempre importante testar diferentes combinações uma vez que o desempenho relativo entre estas não é fixo.

Decidimos verificar se a diferença entre -O0 e -O3 é perceptível através das ferramentas de monitorização.

Numa abordagem inicial foi utilizada a ferramenta mpstat para avaliar a utilização do processador. Independentemente da versão, mpstat indica que um dos cores está 100% ocupado com o *benchmark* durante a sua execução. É necessário ter cuidado com a interpretação deste valor. Estar 100% do tempo ocupado com o processo não significa que o processador esteve a efetuar trabalho útil. Uma parte significativa deste tempo é gasta à espera de leituras e escritas da memória, entre outros. Uma ferramenta mais apropriada nesta situação seria o perf. [http://www.brendangregg.com/blog/2017-05-09/cpu-utilization-is-wrong.html]

Como as métricas dos *stalled cycles* não são suportadas na arquitetura Ivy Bridge, as medições com o *perf* apresentadas foram efetuadas nos nodos 431.

LU	-O0	-O3	-O3 -xHost -ipo
unhalted-cycles	4.22e12	4.27e11	4.25e11
stalled-cycles-frontend	1.55e12	1.93e11	1.93e11
stalled-cycles-backend	1.62e11	9.96e10	9.65e10
instructions	9.92e12	5.84e11	5.8e11
CPI	2.35	1.37	1.36

TABLE IV
RESULTADOS DO *perf stat* PARA DIFERENTES *flags* DO IFORT PARA O *benchmark* LU-MZ

BT	-O0	-O3	-O3 -xHost -ipo
unhalted-cycles	3.05e12	5.66e11	5.20e11
stalled-cycles-frontend	1.14e12	1.96e11	1.6711
stalled-cycles-backend	7.10e10	4.35e10	3.75e10
instructions	7.3e12	1.11e12	1.05e12
CPI	2.39	1.95	2.02

TABLE V
RESULTADOS DO *perf stat* PARA DIFERENTES *flags* DO IFORT PARA O *benchmark* BT-MZ

SP	-O0	-O3	-O3 -xHost -ipo
cycles	2.69e12	3.33e11	3.30e11
stalled-cycles-frontend	7.95e11	1.43e11	1.39e11
stalled-cycles-backend	6.36e10	4.84e10	4.78e10
instructions	7.26e12	5.00e11	5.02e11
CPI	2.70	1.50	2.02

TABLE VI
RESULTADOS DO *perf stat* PARA DIFERENTES *flags* DO IFORT PARA O *benchmark* SP-MZ

A utilização destas *flags* leva a que sejam feitas otimizações, como a vetorização, substituição de várias instruções simples por uma instrução SIMD (single instruction multiple data), que provocam uma diminuição no CPI (número de instruções por ciclo), que é frequentemente usado como métrica de desempenho, tornando a métrica algo enganadora neste contexto.

As métricas que melhor indicam os efeitos das optimizações são simples mas importantes. O número de instruções diminuiu significativamente, tal como o número de *stalled cycles*, ciclos em que o processador não efectuou trabalho útil.

Ao utilizar ferramentas para avaliar o desempenho é necessário interpretar correctamente o significado das métricas e o ambiente de execução, de forma a não serem obtidas conclusões erradas.

IV. OMP

Todas as medições efetuadas para a versão OMP foram realizadas utilizando apenas as *flags* que obtiveram melhores resultados nos testes da versão sequencial. Para o compilador gFortran são *-O3 -march=native* e para o IFORT *-O3 -xHost -ipo*. Para além disso, foi ainda necessário adicionar as *flags* -qopenmp e -fopenmp, para os compiladores IFORT e gFortran respetivamente, de forma a que suportassem a paralelização do código.

Assim como na versão sequencial fora utilizados os nodos 641 e 431 do SeARCH. Uma vez que os nodos apresentam processadores com diferentes quantidades de *cores* a quantidade de *threads* máxima foi alterada, ambos realizam testes para 1, 2, 4, 8 e 16 *threads*. No entanto, o nodo 641 termina com 32 *threads* e o 431 apenas com 24.

Apesar de termos tentado utilizar diferentes módulos e *flags* de compilação não conseguimos executar o *benchmark* LU-MZ com mais de uma *thread*. , pelo que o resultado obtido vai ser omitido da discussão no resto da secção assim como não vai estar presente nos gráficos.

Os seguintes gráficos representam os tempos de execução dos *benchmarks* SP-MZ e BT-MZ, nos diferentes nodos, para ambos os compiladores e com quantidades diferentes de *threads*.

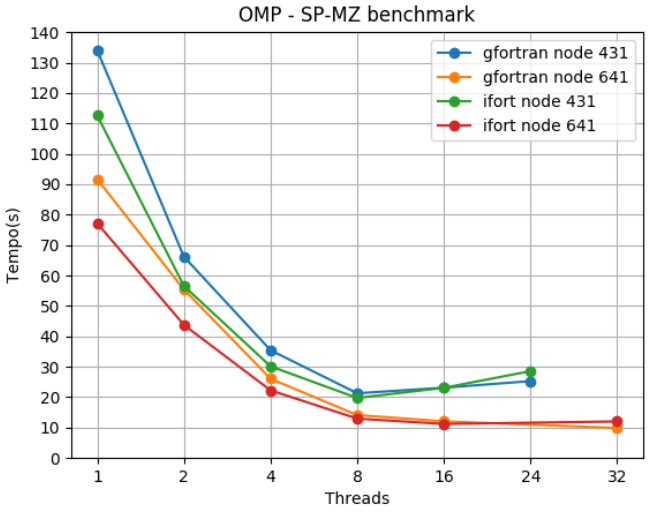

Fig. 1. Tempos de execução do *benchmark* SP-MZ

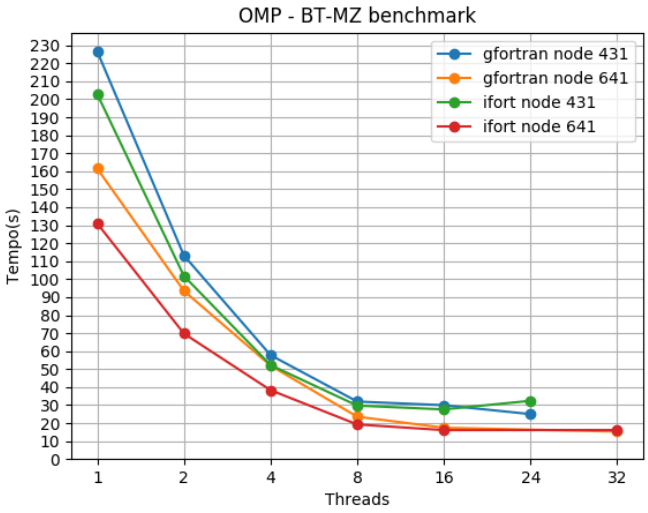

Fig. 2. Tempos de execução do *benchmark* BT-MZ

Como é comum na utilização de OpenMP, os ganhos não escalam idealmente com o número de *threads*.

A melhor combinação varia conforme o *benchmark* utilizado e o nodo.

- Nodo 431 do SP o melhor é o IFORT (8 threads)
- Nodo 641 do SP o melhor é o gFortran (32 threads)
- Nodo 431 do BT o melhor é o gFortran (24 threads)
- Nodo 641 do BT o melhor é o IFORT (32 threads) (apresentam valores bastante semelhantes)

Com esta análise podemos concluir que apesar de muitas vezes ignorado, o compilador é um fator importante na escalabilidade de uma aplicação.

V. MPI+OMP

Na versão híbrida, foram utilizadas as *flags* -O3 para ambos os compiladores e as *flags* -qopenmp e -fopenmp para o compilador IFORT e gFortran respetivamente. Todas as medições foram realizadas em 2 máquinas idênticas em simultâneo, nos nodos 641 e 431 do SeARCH.

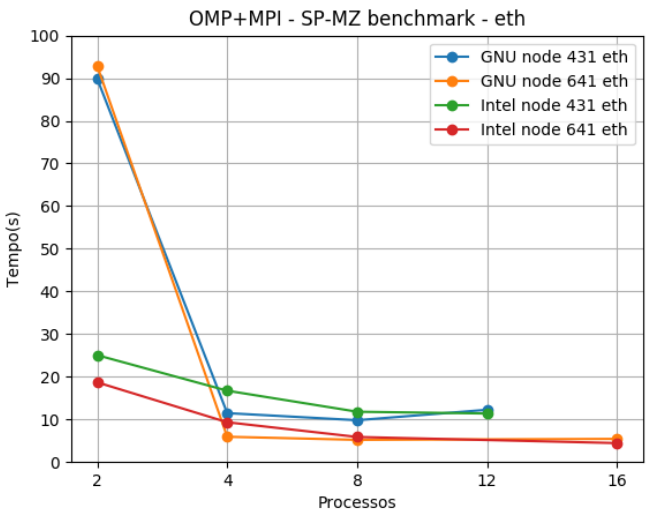

Fig. 3. Tempos de execução do *benchmark* SP-MZ

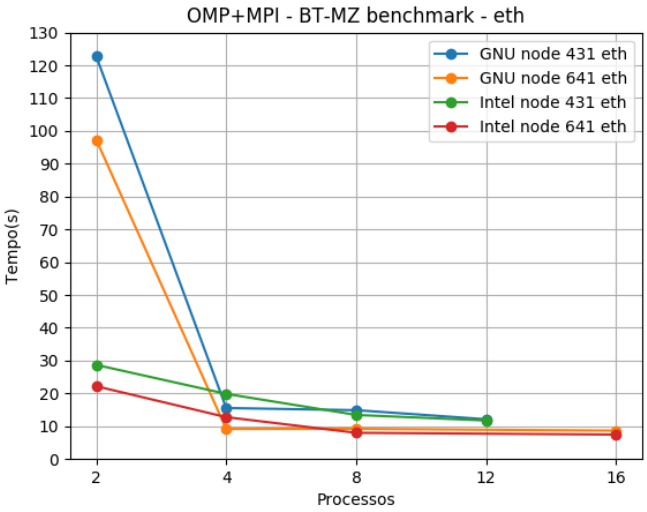

Fig. 4. Tempos de execução do *benchmark* BT-MZ

É possível verificar que a execução com o OpenMPI da GNU apresentou resultados anómalos quando eram utilizados 2 processos. No entanto, para 4 ou mais processos os tempos de execução são conforme o esperado. A primeira ação tomada foi a repetição dos testes, porém a anomalia manteve-se. Decidimos então utilizar o -mpstat para perceber melhor o que se passava.

Os gráficos seguintes representam a média da utilização dos *cores* por aplicações do utilizador durante a execução do *benchmark*. Os valores são referentes a apenas uma das máquinas utilizadas.

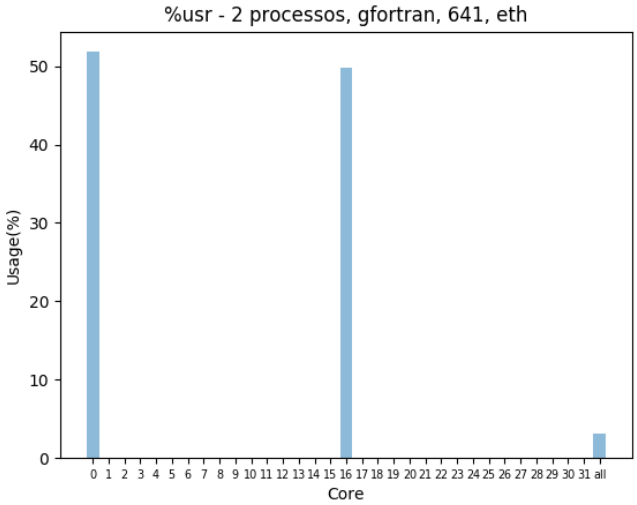

Fig. 5. Utilização dos *cores* para 2 processos (1 por máquina)

Em cada máquina apenas eram usados 2 cores, mesmo com o valor correto da variável de ambiente OMP_NUM_THREADS.

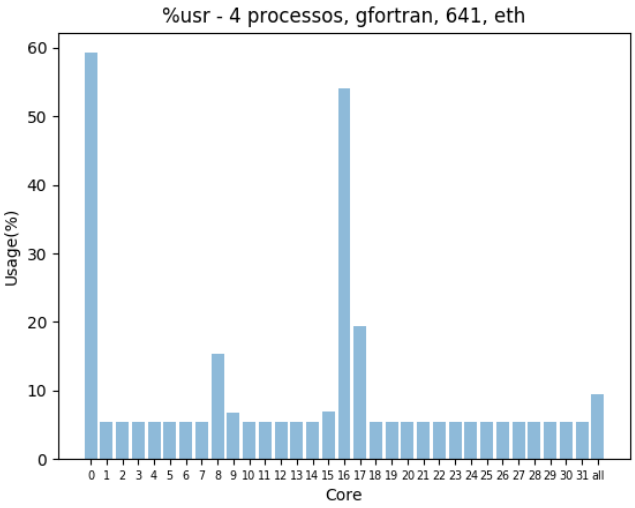

Fig. 6. Utilização dos *cores* para 4 processos (2 por máquina)

No entanto com 2 processos em cada máquina(4 no total), todos os cores lógicos eram usados.

Ao contrário da versão da GNU, a versão do OpenMPI da Intel com um processo por máquina apresentava resultados normais. Por uma questão de curiosidade decidimos também observar a utilização dos *cores*.

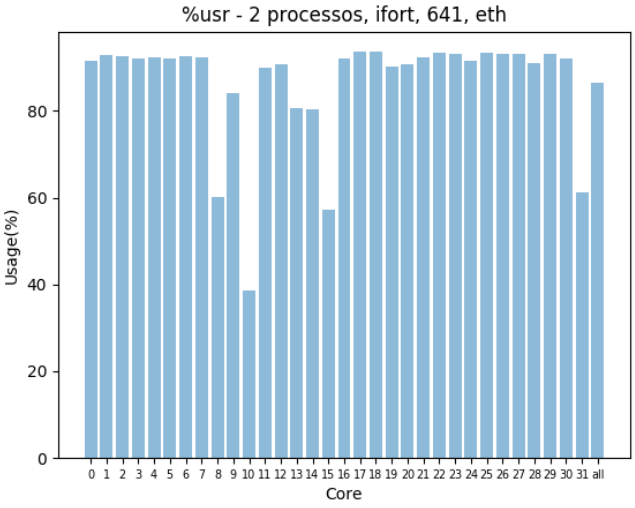

Fig. 7. Utilização dos *cores* para 2 processos (1 por máquina)

A versão da Intel utiliza todos os *cores*, o que nos leva a suspeitar da existência de alguma incompatibilidade entre o *benchmark* e a versão da GNU usada.

Inicialmente, os testes que utilizam ethernet como forma de comunicação entre máquinas foram realizados recorrendo às versões mais recentes de OpenMPI para ambos os compiladores, openmpi_eth/2.0.0 para o gFortran e openmpi_eth/1.8.2 para o IFORT.

Para a versão que utilizava *myrinet* foram utilizadas as versões openmpi_mx/1.8.4 para ambos os compiladores. No entanto, após executados os testes podemos observar que os valores obtidos eram piores que a versão *ethernet* do compilador GNU. Uma das principais causas poderia ser a versão mais desatualizada dos módulos da *myrinet*, pelo que decidimos utilizar a versão openmpi_eth/1.8.4 para o compilador da GNU na versão *ethernet*.

Após novas medições, a versão openmpi_eth/1.8.4 não apresenta melhores resultados que a versão myrinet. No entanto a diferença é pouco significativa, por isso achamos que o problema pode não ter ficado completamente resolvido.

Para além dos módulos de MPI também é necessário importar o módulo gcc/5.3.0 para o compilador da GNU e o módulo intel/2019 para o compilador da Intel, tanto na versão *ethernet* como na *myrinet*.

Os testes realizados foram corridos com 2, 4, 8 e 16 processos para os nodos 641 e 2, 4, 8, 12 processos para os nodos 431. O número de *threads* utilizado varia consoante o número de processos, de forma a utilizar todos os recursos das máquinas. Para cada uma das execuções, este valor era calculado através da divisão do número máximo de *cores lógicos* pelo número de processos na mesma máquina, por exemplo, para os nodos 641 (com 32 *cores* lógicos) e 4 processos totais (2 por máquina), vão ser utilizados 16 *threads* por máquina.

Foram utilizadas as *flags* -report-bindings e I_MPI_DEBUG 4 para verificar se o mapeamento estava de acordo com as expectativas.

VI. Utilitários de Monitorização

Para processar os dados das ferramentas foram desenvolvidos vários scripts em Python que agregam as estatísticas da execução do programa. Para além disso, os scripts foram utilizados para visualizar graficamente valores como médias, máximos, e variações ao longo do tempo. Apesar de ser possível visualizar graficamente todas as métricas recolhidas, nas subsecções seguintes apenas são apresentadas as que consideramos mais pertinentes.

Devido ao tamanho reduzido da classe A, apenas foi possível realizar medições para a versão sequencial. Tanto na versão OMP como na MPI, o tempo de execução é de tal maneira reduzido que torna irrelevante qualquer valor obtido com as ferramentas utilizadas. Todos os testes nesta fase utilizaram o compilador da Intel com as flags de optimização mencionadas, e foram executados no nodo 641.

A. CPU

Para obter estatísticas de utilização do CPU foi utilizado o comando mpstat -P ALL 1. A flag -P ALL permite a obtenção de informação sobre todos os cores lógicos da máquina. [https://linux.die.net/man/1/mpstat]

No caso dos benchmarks serial, apenas um core é utilizado e a sua utilização é sempre próxima dos 100%. usr, independentemente da classe do problema. Verificou-se que ocasionalmente a execução migra entre cores.

No caso das versões OMP e híbrida, as classes maiores correspondem a alguns aumentos da utilização global dos cores

Os gráficos seguintes correspondem à média da percentagem de utilização correspondente a aplicações *user level*.

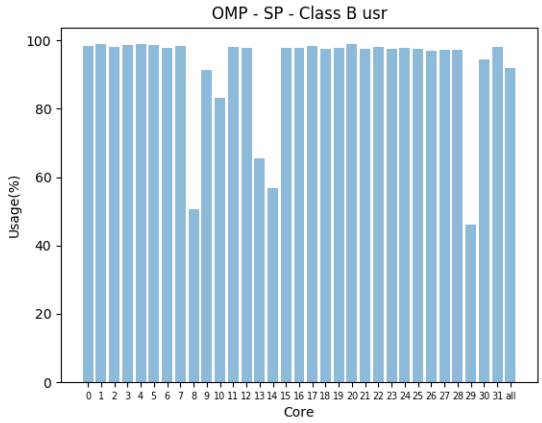

Fig. 8. Utilização dos cores lógicos pelo benchmark SP-MZ da versão OMP da classe B

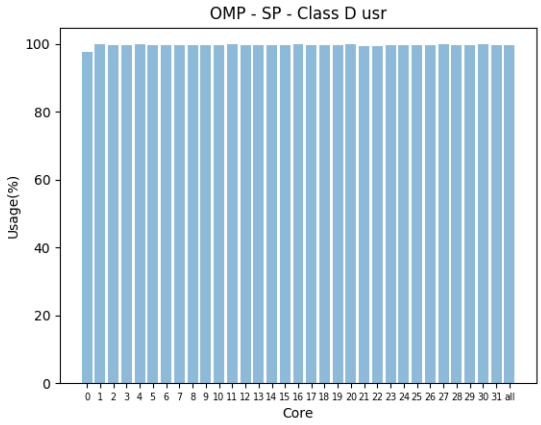

Fig. 10. Utilização dos cores lógicos pelo benchmark SP-MZ da versão OMP da classe D

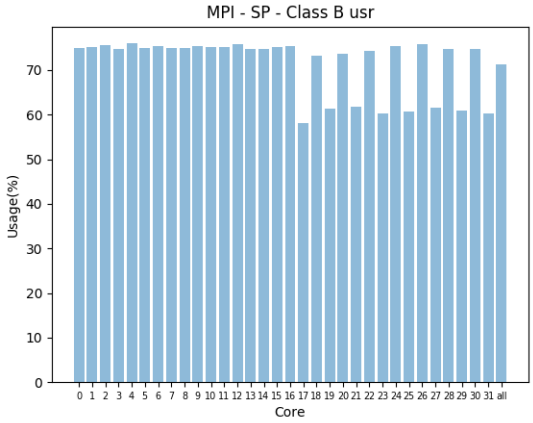

Fig. 11. Utilização dos cores lógicos pelo benchmark SP-MZ da versão MPI+OMP da classe B

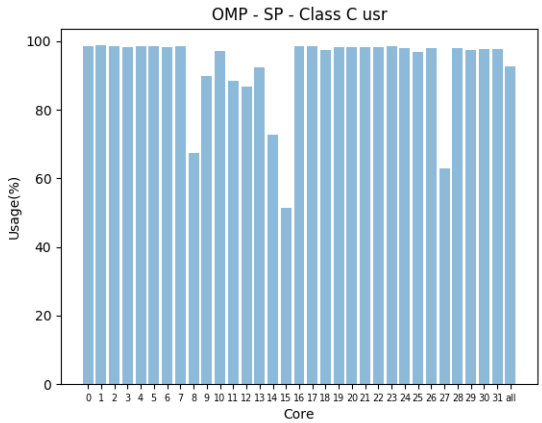

Fig. 9. Utilização dos cores lógicos pelo benchmark SP-MZ da versão OMP da classe C

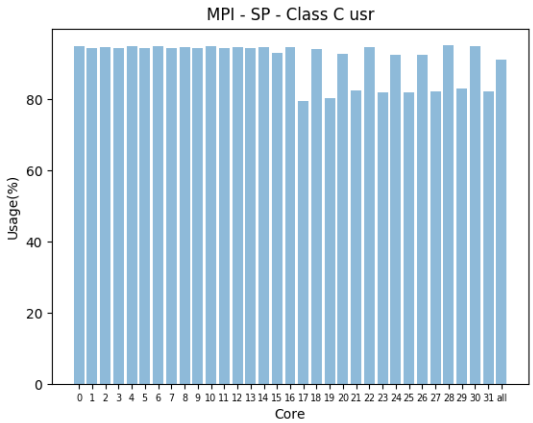

Fig. 12. Utilização dos cores lógicos pelo benchmark SP-MZ da versão MPI+OMP da classe C

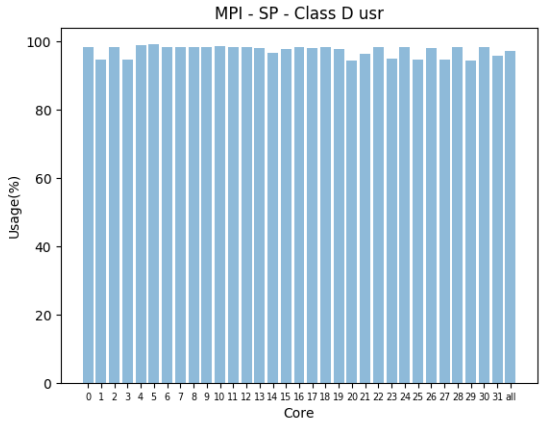

Fig. 13. Utilização dos cores lógicos pelo benchmark SP-MZ da versão MPI+OMP da classe D

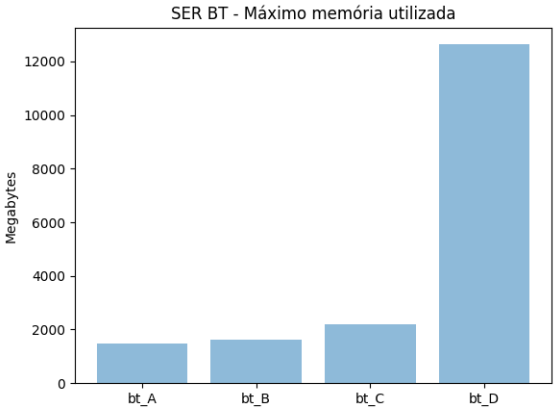

Fig. 14. Memória utilizada pela versão SER do benchmark BT-MZ para as diferentes classes

B. *Memória*

As medições foram feitas através do uso da ferramenta free. [http://man7.org/linux/man-pages/man1/free.1.html]

O comando específico é free -m -s 1, onde a flag -m converte os valores para MBytes e a flag -s 1 leva a que as medições sejam feitas com intervalos de 1 segundo.

Também foram efetuadas medições com a ferramenta vmstat e desenvolvido o script correspondente. No entanto, demos maior preferência à ferramenta free, por acharmos as suas métricas mais úteis e simples. Um aspeto interessante da ferramenta vmstat é observar a utilização da swap. Porém, não foi necessário o uso desta em nenhum dos testes efetuados. Foi efetuada uma tentativa de usar a *swap* num nodo com com apenas 8GB mas sem sucesso.

No caso da memória, consideramos mais importante medir o valor máximo do que a média, uma vez que este valor nos vai permitir determinar se a máquina onde esta a ser testado consegue correr o programa sem problemas. Por exemplo, um *benchmark* que tenha utilização média de 5gb de memória facilmente corre numa máquina com 10gb de memória, no entanto, se ao longo da execução por algum motivo existir algum momento em que o programa necessite de 15gb a máquina não tem capacidade para executar o mesmo levando a que o processo seja cancelado ou à utilização de swap.

As medições para os diferentes tipos de *benchmark*, SP-MZ e BT-MZ, apresentam resultados muito próximos pelo que apenas são apresentados gráficos referentes apenas a um dos *benchmarks*, neste caso o BT-MZ.

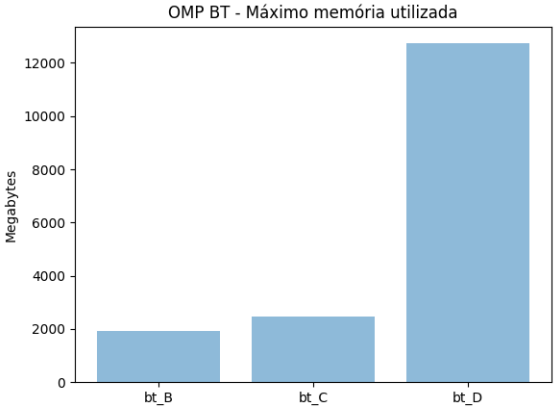

Fig. 15. Memória utilizada pela versão OMP do benchmark BT-MZ para as diferentes classes

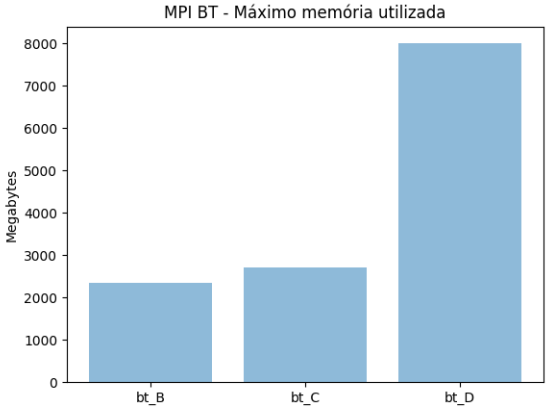

Fig. 16. Memória utilizada pela versão MPI do benchmark BT-MZ para as diferentes classes

Os valores máximos de utilização da memória são próximos aos mencionados na documentação.

Problem Class	Memory Requirement (approx.)
Class S	1 MB
Class W	6 MB
Class A	50 MB
Class B	200 MB
Class C	0.8 GB
Class D	12.8 GB
Class E	250 GB
Class F	5.0 TB

Fig. 17. Valores aproximados indicados pela documentação oficial

Os valores obtidos para as versões SER e OMP são aproximadamente iguais entre si, e a versão MPI é cerca de metade das duas, o que faz sentido tendo em conta que está a ser executado em duas máquinas.

C. Disco

Os resultados do mpstat indicam que o tempo que o cpu espera pelo disco não é significativo. No entanto, é sempre útil desenvolver métodos de analisar o desempenho do disco, pois este pode ser um fator limitante noutras aplicações.

Para medir a utilização do disco foi utilizado o comando iostat -mdx 1. De notar o uso da flag -x que permite o acesso a estatísticas mais detalhadas. [https://linux.die.net/man/1/iostat]

Achamos o impacto no desempenho de muitos dos valores absolutos fornecidos pelo iostat difícil de avaliar, principalmente devido a uma falta de referência de valores normais para o *hardware* utilizado. Como tal focámo-nos na métrica %util que, segundo a documentação, permite ter uma da saturação do disco.

Os gráficos seguintes representam a variação desta métrica nos primeiros 50 segundos de execução.

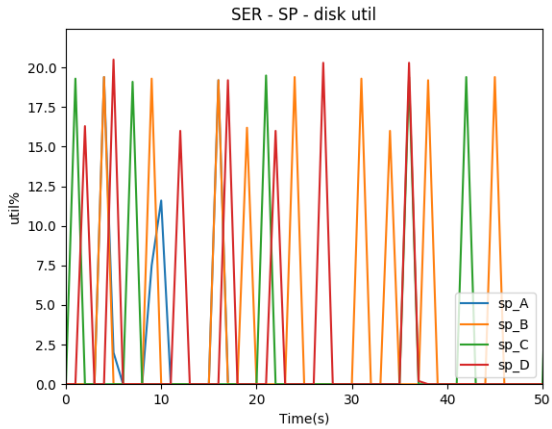

Fig. 18. Utilização do disco pelo benchmark SP-MZ para as diferentes classes da versão SER

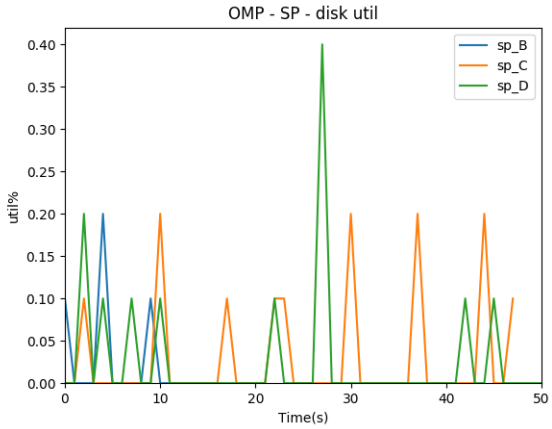

Fig. 19. Utilização do disco pelo benchmark SP-MZ para as diferentes classes da versão OMP

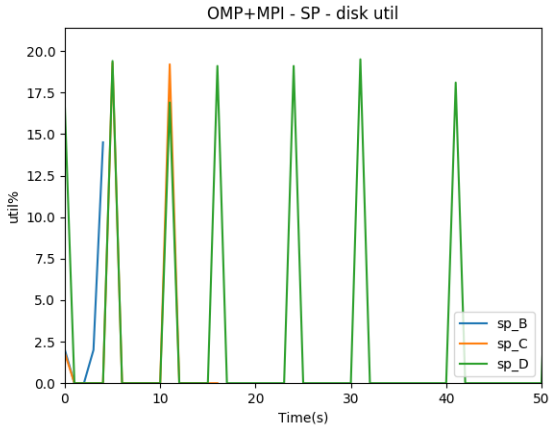

Fig. 20. Utilização do disco pelo benchmark SP-MZ para as diferentes classes da versão MPI+OMP

Os resultados da versão OMP apresentam picos de utilização significativamente menores que as restantes versões.

Como %util indica apenas a percentagem de tempo em que existe algum pedido I/O a ser servido e não a quantidade destes, é possível que várias *threads* executem pedidos paralelamente num curto espaço de tempo. [https://brooker.co.za/blog/2014/07/04/iostat-pct.html]

De qualquer forma, os resultados indicam que o disco não é um gargalo de desempenho dos *benchmarks* testados.

D. Rede

As medições sobre a rede foram feitas através do comando sar -n TCP,ETCP,DEV 1. As *flags* utilizadas disponibilizam estatísticas sobre a rede. [https://linux.die.net/man/1/sar]

O seguinte gráfico representa a quantidade de informação em kilobytes enviada através da interface *eth0* para a segunda máquina.

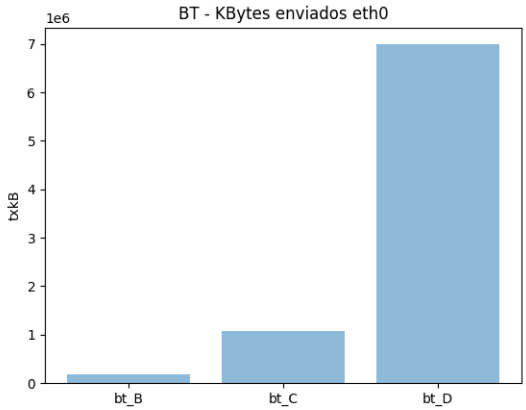

Fig. 21. Quantidade de dados enviados para a segunda máquina

O total de dados transmitidos corresponde a aproximadamente 120% da metade dos dados da classe. O excedente pode corresponder a custos de comunicação, tais como cabeçalhos, e a transmissões de informação não relacionadas com o *benchmark*.

VII. Trabalho futuro

Na maioria dos casos, o compilador da Intel apresentou melhores resultados. Seria interessante correr os *benchmarks* num processador não Intel e verificar se as diferenças de desempenho dos compiladores sofriam alterações.

Também seria interessante testar *benchmarks* que fossem *memory-bound* ou *I/O bound*, assim como *benchmarks* em C e verificar se a utilização dos diferentes compiladores e *flags* têm resultados comparáveis.