

Introdução ao NAS Parallel Benchmarks (NPB)

Testes de Referência Versões: Sequencial, Memória Partilhada/Memória Distribuída

Ambiente de Operação no Cluster Search

Sérgio Caldas

Universidade do Minho

Escola de Engenharia

Departamento de Informática

Email: a57779@alunos.uminho.pt

Resumo—O *NAS Parallel Benchmark* é um ambiente de testes desenvolvido pela NASA, para medir a performance de super-computadores. Este ambiente de testes é constituído por 5 Kernels (IS, EP, CG, MG, FT) desenvolvidos em C/Fortran em três versões, versão sequencial e versões paralelas (Open-MP e Open-MPI). Para além destes 5 Kernels, este *benchmark* tem um conjunto de classes (S, W, A, B, C, D, E, F) cada uma com diferentes tamanhos de dados. No desenvolvimento deste trabalho tive de escolher 3 desses 5 Kernels e algumas classes, de forma a efectuar uma gama de testes para cada uma das versões, num ambiente de operação cluster, mais precisamente no cluster "Search".

1. Introdução

Este trabalho foi realizado no âmbito da disciplina de Engenharia de Sistemas da Computação (ESC), inserida no perfil de Computação Paralela e Distribuída (CPD) do 4º Ano do curso Mestrado Integrado em Engenharia Informática (MIEI) e tem como objectivo analisar a *performance* de um ambiente de testes, neste caso o *Cluster SeARCH*, na execução do *NAS Parallel Benchmark*.

Este *Benchmark* é constituído por 5 Kernels e 3 simulações bem como um conjunto de 8 classes de dados, destes 5 Kernels tive de escolher alguns de forma a efectuar um conjunto de testes, para posteriormente fazer uma análise/comparação detalhada desses mesmos testes, para além da escolha dos Kernels ainda foi necessário escolher um conjunto de classes de dados para os testes.

Os testes consistiram na repetição destes nas diferentes classes de arquitecturas de nós existentes, usando as versões sequenciais e paralelas (memória distribuída - OpenMPI e memória partilhada - OpenMP), nos diferentes compiladores (gnu e intel), diferentes opções de compilação, diferentes tecnologias de comunicação e diferentes dimensões de dados (classes).

A análise/comparação dos resultados obtidos deverá ser feita tendo em consideração medições precisas de tempos de execução/ocio/bloqueio, da ocupação da memória, da comutação do tempo de E/S, bem como outras métricas das ferramentas de monitorização.

Para a realização destes testes foi desenvolvido um *Script* que me permitiu fazer a execução dos testes em lotes com base no sistema PBS.

Depois da obtenção dos resultados dos testes, os dados foram tratados com *Shell Scripts* utilizando ferramentas de processamento de linguagens, como o *grep* e *sort*. Posteriormente, depois de filtrados os dados, estes foram processados com *Excel* para geração dos gráficos.

2. Caracterização do Ambiente de Testes

O ambiente de testes utilizado no decorrer deste trabalho foi o cluster *SeARCH*, este cluster faz parte do Departamento de Informática da Universidade do Minho, sendo este utilizado por uma vasta comunidade de cientistas/investigadores". O *SeARCH* é constituído por um conjunto de nós com diferentes arquiteturas. Os nós constituintes do *SeARCH* são:

- Arquitectura *Ivy Bridge*
 - 6 Nós 662
 - 2 Nós 652
 - 20 Nós 641
- Arquitectura *Sandy Bridge*
 - 6 Nós 541
- Arquitetura *Nehalem*
 - 2 Nós 432
 - 4 Nós 421
 - 10 Nós 431
- Arquitetura *Penryn*
 - 6 Nós 321
- Arquitetura *AMD Magny-Cours*
 - 2 Nós 262

De notar que todos os detalhes de cada nó, bem como o significado do *rank* (valores do tipo 641, 652, etc. apresentados em cima) podem ser consultados no site do cluster *SeARCH* [3]

System	Máquina 431
# CPUs	2
CPU	Intel® Xeon® X5650
Architecture	Nehalem
# Cores per CPU	6
# Threads per CPU	12
Clock Freq.	2.66 GHz
L1 Cache	192 KB 32 KB por core
L2 Cache	1536 KB 256 KB por core
L3 Cache	12 MB
Inst. Set Ext.	SSE4.2 e AVX
#Memory Channels	3
Memory BW	32 GB/s

Tabela 1. CARACTERIZAÇÃO DA MÁQUINA 431

System	Máquina 641
# CPUs	2
CPU	Intel® Xeon® E5-2650v2
Architecture	Ivy Bridge
# Cores per CPU	8
# Threads per CPU	16
Clock Freq.	2.6 GHz
L1 Cache	256 KB 32 KB por core
L2 Cache	2048 KB 256 KB por core
L3 Cache	20 MB
Inst. Set Ext.	AVX
#Memory Channels	4
Memory BW	59.7 GB/s

Tabela 2. CARACTERIZAÇÃO DA MÁQUINA 641

2.1. Nodos de Teste

As máquinas de teste que escolhi para a execução dos *Kernels* no *Cluster Search*, foram as máquinas 431 e 641 com arquiteturas *Nehalem* e *Ivy Bridge* respectivamente. Na tabela 1 encontra-se as especificação das características da máquina 431 e na tabela 2 as da máquina 641.

2.2. Compiladores Utilizados

No *Cluster Search* podemos encontrar uma gama de versões de compiladores da *GNU*, bem como outros compiladores como é o caso do compilador da *Intel*, neste trabalho os compiladores utilizados por mim foram:

- gnu/4.9.0 - utilizei este compilador, porque este é o compilador que está predefinido no *Cluster Search*, como tal decidi optar por esta versão;
- gnu/4.9.3 - este compilador foi usado nos meus testes, por ser o compilador mais recente instalado no *Cluster Search*;
- intel/2013.1.117 - este compilador foi usado porque é o único compilador da *Intel* instalado no *Cluster Search* e para além disso utilizei-o para poder comparar os dois compiladores, isto é, comparar entre os compiladores da *Intel* e da *GNU*.

3. Caracterização do *Benchmark*

O *NAS Parallel Benchmark* [4] consiste num conjunto de 5 *Kernels* e 3 simulações desenvolvidos em C e Fortran, bem

como um conjunto de 8 classes (cada uma com diferentes tamanhos de dados de *input*). Este *Benchmark* foi desenvolvido com o objetivo de fornecer uma medida das capacidade do hardware e software, para resolver intensivos problemas computacionais de dinâmica de fluídos importantes para a NASA.

Os *Kernels* e as simulações que constituem o *Benchmark* são os seguintes:

- **Kernels:**

- IS [1] - Ordenação de inteiros, acesso aleatório à memória.
- EP [5] - Este *Kernel* é embarrasosamente paralelo. Ele gera pares de desvio gaussianas aleatórios de acordo com um esquema específico. Sendo o objetivo estabelecer o ponto de referência para o máximo desempenho de uma determinada plataforma.
- CG [5] - Este *Kernel* utiliza um método de Gradiente Conjugado para calcular uma aproximação para o menos valor próprio de uma matriz grande, esparsa e não estruturada. Este testa cálculos de grelha não estruturados e comunicações usando uma matriz com entradas geradas aleatoriamente.
- MG [5] - Este *Kernel* calcula uma solução para a equação de *Poisson* 3-D através de um método Multigrid ciclo-V.
- FT [5] - Este *Kernel* faz a computação de uma Transformação de *Fourier* 3-D rápida com base no método espectral.

- **Simulações:**

- BT [5] - É uma aplicação de simulação CFD ¹ que utiliza um algoritmo implícito para resolver equações de *Navier-Stokes* de 3-Dimenções.
- SP [5] - É uma aplicação de simulação CFD similar a BT, a diferença finita de soluções é baseada na factorização aproximada *Beam-Warming* que dissocia as dimensões x, y e z.
- LU [5] - É uma aplicação de simulação CFD que usa o método SSOR ² para resolver um sistema *seven-block-diagonal* resultante da discretização de diferenças finitas das equações de *Navier-Stokes* em 3-D.

As classes de dados [1] presentes no *Benchmark* são:

- S - pequena, utilizada para testes rápidos;
- W - tamanho de estação de trabalho;
- A, B, C - Utilizada em testes de problemas *standard*. Aumenta aproximadamente 4X o tamanho de uma classe para a próxima.

1. Computação de fluídos dinâmicos

2. Symmetric Successive Over-Relaxation

- D, E, F - Utilizada em testes de problemas de larga escala. Aumenta aproximadamente 16X o tamanho de uma classe para a próxima.

É possível consultar informação mais detalhada de cada uma das classes, consultando o site do *NAS Parallel Benchmark* [2].

3.1. Escolha de Kernels e Classes de Dados

Depois de uma análise do *benchmark*, os kernels por mim escolhidos foram os seguintes:

- CG [1] - Gradiente conjugado, Irregular acesso à memória e comunicação irregular;
- EP [1] - Kernel embarrasosamente paralelo;
- IS [1] - Ordenação de inteiros, acesso aleatório à memória.

No que toca a classes de dados para testes, as minhas escolhas foram:

- Classe A [2] - Tem um tamanho de 50 MB ;
- Classe B [2] - Tem um tamanho de 200 MB;
- Classe C [2] - Tem um tamanho de 0.8 GB.

4. Testes Efectuados

Como foi dito anteriormente, o objetivo deste trabalho era efectuar uma gama de testes para diferentes *Kernel's* para diferentes Classes de dados, em baixo podem ser consultados quais os testes efectuados por mim, nas diferentes máquinas, para os diferentes *Kernel's*, para as diferentes classes e com diferentes compiladores. Sendo que o X diz que o teste foi realizado e - que não foi realizado.

4.1. Versão Sequencial

Na tabela 3 podemos consultar todos os testes efectuados para a versão sequencial na máquina 431 e na tabela 4 todos os testes efectuados para a versão sequencial na máquina 641.

Máquina 431	-O0	-O2	-O3
gnu/4.9.0	X	X	X
gnu/4.9.3	X	X	X
intel/2013.1.117	X	X	X

Tabela 3. TESTES EFECTUADOS PARA VERSÃO SEQUENCIAL NA MÁQUINA 431

Máquina 641	-O0	-O2	-O3
gnu/4.9.0	X	X	X
gnu/4.9.3	X	X	X
intel/2013.1.117	X	X	X

Tabela 4. TESTES EFECTUADOS PARA VERSÃO SEQUENCIAL NA MÁQUINA 641

4.2. Versão OMP

Na tabela 5 podemos consultar todos os testes efectuados para a versão OMP na máquina 431 e na tabela 6 todos os testes efectuados para a versão OMP na máquina 641.

Máquina 431	-O0	-O2	-O3
gnu/4.9.0	-	X	X
gnu/4.9.3	-	X	X
intel/2013.1.117	-	X	X

Tabela 5. TESTES EFECTUADOS PARA VERSÃO OMP NA MÁQUINA 431

Máquina 641	-O0	-O2	-O3
gnu/4.9.0	-	X	X
gnu/4.9.3	-	X	X
intel/2013.1.117	-	X	X

Tabela 6. TESTES EFECTUADOS PARA VERSÃO OMP NA MÁQUINA 641

4.3. Versão MPI

Na tabela 7 podemos consultar todos os testes efectuados para a versão MPI na máquina 431 e na tabela 8 todos os testes efectuados para a versão MPI na máquina 641. De referir que nestes testes fixei as flags em -O3, sendo que 8, 16 e 32 refere-se ao número de processos para os quais fiz os testes.

Máquina 431	8	16	32
gnu/openmpi.eth/1.8.4	X	X	X
gnu/openmpi.mx/1.8.4	-	-	-
intel/openmpi.eth/1.8.2	-	-	-
intel/openmpi.mx/1.8.2	-	-	-

Tabela 7. TESTES EFECTUADOS PARA VERSÃO MPI NA MÁQUINA 431

Máquina 641	8	16	32
gnu/openmpi.eth/1.8.4	X	X	X
gnu/openmpi.mx/1.8.4	-	-	-
intel/openmpi.eth/1.8.2	X	X	X
intel/openmpi.mx/1.8.2	-	-	-

Tabela 8. TESTES EFECTUADOS PARA VERSÃO MPI NA MÁQUINA 641

5. Análise de Resultados

5.1. Versão Sequencial

Nos testes para a versão sequencial, fiz uma análise dos tempos e dos MOPS³ para diferentes compiladores, fixando as flags, isto é, cada gráfico apresenta os tempos para diferentes compiladores com a mesma flag de compilação.

Na figura 1, 2 e 3 encontram-se os gráficos com os tempos para diferentes compiladores, com a mesma flag de compilação para as Máquinas 431.

3. Milhões de Operações Por Segundo

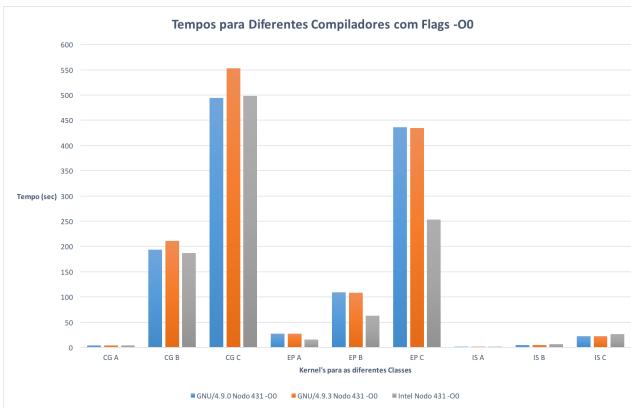


Figura 1. Tempos para Diferentes Compiladores com Flags -O0

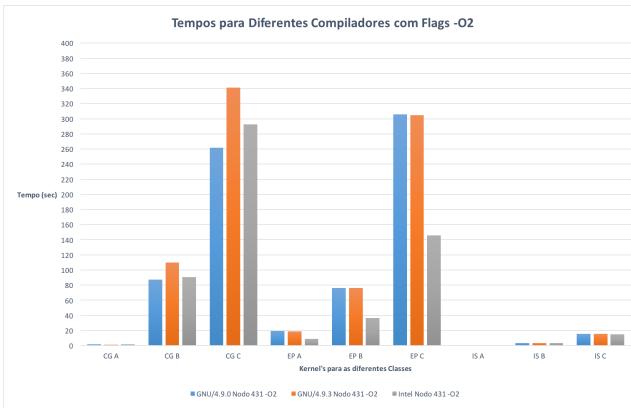


Figura 2. Tempos para Diferentes Compiladores com Flags -O2

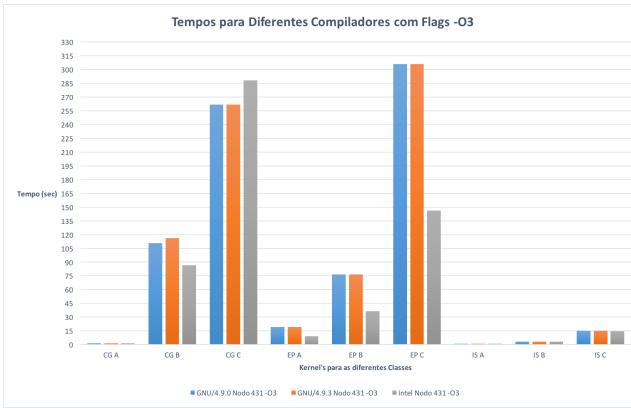


Figura 3. Tempos para Diferentes Compiladores com Flags -O3

Através da comparação dos 3 gráficos, à medida que se vai aumentando o nível de optimização, isto é, passado de -O0 para -O2 e por fim para -O3 podemos constatar que o comportamento dos compiladores é semelhante, mesmo com o aumento da Classe de dados. No caso do gráfico da figura 3 para o Kernel CG com a classe de dados B o compilador da Intel mostra-se melhor do que os outros, neste caso o Kernel apresenta um tempo inferior aos outros, mas

se verificarmos no caso do Kernel CG para a classe de dados C, ai o compilador da Intel é pior que os outros dois, isto é o Kernel compilado com o compilador da Intel apresenta tempos de execução superiores aos outros dois. No caso do Kernel EP (Embaraçosamente paralelo) podemos afirmar que o compilador da Intel é melhor que os outros dois da GNU, pois para qualquer classe de dados o Kernel apresenta sempre tempos melhores que do que quando compilado com os compiladores da GNU.

Nos gráficos das figuras 4, 5 e 6 podemos consultar os gráficos que relacionam os MOPS para os diferentes Kernel's para as diferentes classes de dados, para os diferentes compiladores, com a mesma flag de compilação.

Neste caso, mais uma vez, à medida que se aumenta o nível de compilação, o comportamento dos compiladores é semelhante, sendo que o Kernel CG é o que apresenta maior numero de MOPS, no caso do gráfico da figura 6 este Kernel, para a classe de dados A o compilador da Intel é ligeiramente menos eficiente do que os outros dois da GNU. Para a classe de dados B o compilador mais eficiente é o compilador da GNU da versão 4.9.0, tendo este aproximadamente o mesmo valor que o Kernel quando compilado com o compilador da intel. Na classe de dados C para os dois compiladores da GNU o Kernel apresenta valores semelhantes sendo estes melhores que o Kernel quando compilado com o compilador da Intel. A semelhança do que acontece nos tempos, para o Kernel EP o compilador da Intel, mais uma vez é o mais eficiente, apresentando valores sempre superiores aos outros dois compiladores da GNU, quer para a classe de dados A, B e C.

Os resultados para as máquinas 641 são praticamente semelhantes aos resultados das máquinas 431, isto é o comportamento á medida que se aumenta o nível de optimização é praticamente igual. Os seus gráficos podem ser consultados no anexo A

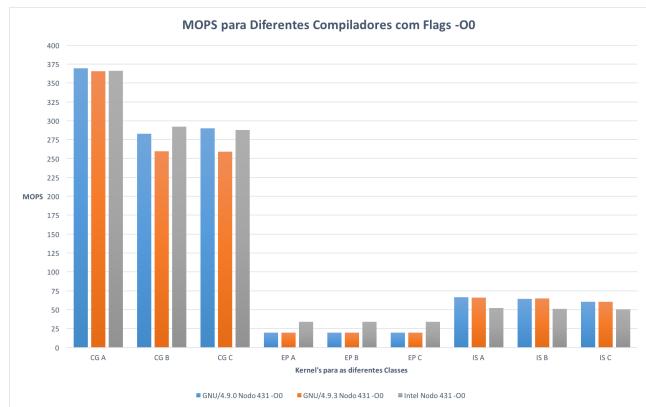


Figura 4. MOPS para Diferentes Compiladores com Flags -O0

5.2. Versão OMP

Um dos objetivos do trabalho era executar código em paralelo, neste caso com um paradigma de memória partilhada recorrendo a OpenMP, sendo que os testes foram efetuados

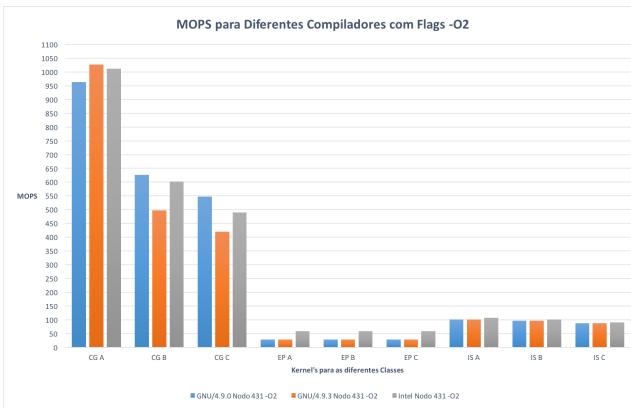


Figura 5. MOPS para Diferentes Compiladores com Flags -O2

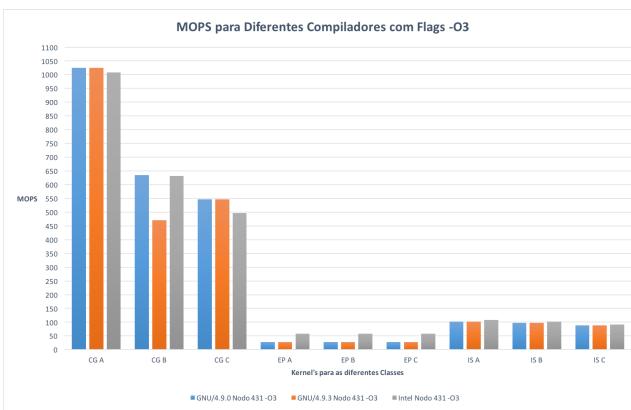


Figura 6. MOPS para Diferentes Compiladores com Flags -O3

para 1, 2, 4, 8, 10, 12, 16, 24 e 32 *Threads*. Na análise dos resultados destes testes, procurei comparar os tempos de execução, os MOPS e MOPS/Thread para os diferentes compiladores, fixando as flags de compilação para um *Kernel*. De referir que devido a um grande número de gráficos que foram gerados para a versão OMP, apenas apresento uma parte dos resultados, para isso procurei apresentar os gráficos que achei que representam os melhores resultados que obtive. Como mostra a tabela 5 e a tabela 6 para esta versão apenas efetuei testes com as flags de compilação -O2 e -O3. De notar que os gráficos apresentados dizem respeito aos testes efetuados nas máquinas 431.

Escolhi os gráficos do *Kernel IS*, pois foi o *Kernel* para o qual obtive um tempo de execução inferior. Como podemos verificar pela a análise do gráfico da figura 7 e do gráfico da figura 8 há pouca variação dos tempos quer na versão compilada com a flag -O2, quer com a flag -O3, tanto um gráfico como o outro são praticamente iguais.

No gráfico da figura 7 com flag -O2 o *Kernel IS* atinge o seu menor tempo para o compilador gnu/4.9.0 com 24 *Threads* tanto para a classe de dados A, B e C com tempos de 0.08 segundos, 0.33 segundos e 1.41 segundos respetivamente. Para o compilador gnu/4.9.3 com flag -O2 o *Kernel* atinge o menor tempo para a classe A com 24 *Threads* bem

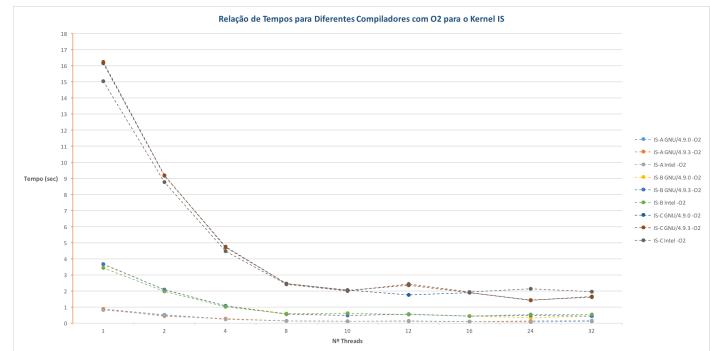


Figura 7. Tempos para Diferentes Compiladores com Flags -O2 para o Kernel IS

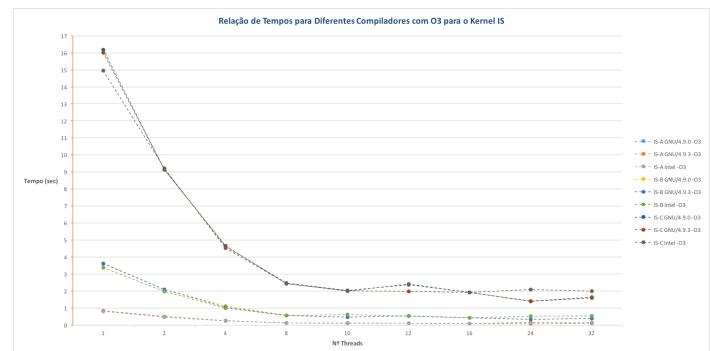


Figura 8. Tempos para Diferentes Compiladores com Flags -O3 para o Kernel IS

como para a classe C com tempos de 0.08 segundos e 1.44 segundos respetivamente, para a classe B o *Kernel* atinge o menor tempo tanto com 16 *Threads* como com 24 *Threads* com valor de 0.44 segundos. Quanto ao compilador da *Intel*, com flag -O2 o *Kernel* atinge o seu menor tempo com 16 *Threads*, quer para a classe A, B e C com tempos de 0.1 segundos, 0.43 segundos e 1.93 segundos respetivamente. Podemos assim concluir que para este *Kernel* com flag -O2, independentemente da classe de dados o compilador da *Intel* é o mais eficiente.

No gráfico da figura 8, com flag -O3, para o compilador gnu/4.9.0 o *Kernel IS* atinge o seu menor tempo para a classe A com 16 *Threads*, tendo um valor de 0.1 segundos e para as classes de dados B e C atinge o menor tempo com 24 *Threads*, com tempos de 0.34 segundos e 1.4 segundos respetivamente. Para o compilador da gnu/4.9.3, com flag -O3 o *Kernel* atinge o seu menor tempo com 24 *Threads*, quer para a classe A, B e C, com tempos de 0.08 segundos, 0.35 segundos e 1.41 segundos respetivamente. No que toca ao compilador da *Intel* com flag -O3 o *Kernel* atinge o seu menor tempo com 16 *Threads*, com tempos de 0.1 segundos, 0.43 segundos e 1.92 segundos respetivamente. À semelhança do que acontece com o caso anterior, isto é, quando o *Kernel* é compilado com -O2, aqui o compilador da *Intel* também mostra ser o mais eficiente, com flag -O3.

Quanto aos MOPS como podemos verificar nos gráficos da figura 9 e no gráfico da figura 10, estes tendem a

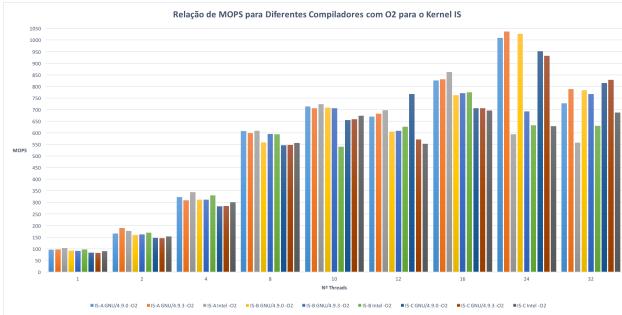


Figura 9. MOPS para Diferentes Compiladores com Flags -O2 para o Kernel IS

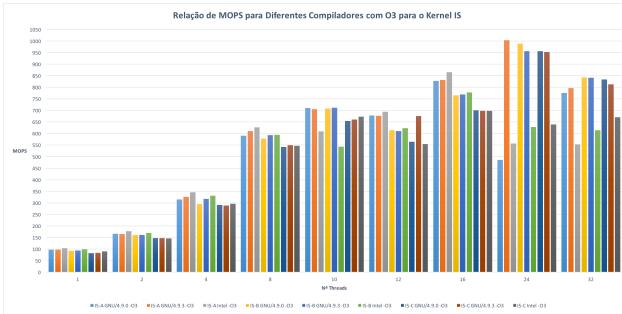


Figura 10. MOPS para Diferentes Compiladores com Flags -O3 para o Kernel IS

aumentar até às 24 *Threads*, tanto com flag -O2 como com flag -O3. Em oposição os MOPS/*Thread* tendem a diminuir com o aumento do número de *Threads* tanto com flag -O2 como com flag -O3, como mostra o gráfico da figura 11 e o gráfico da figura 12, o que já era de esperar.

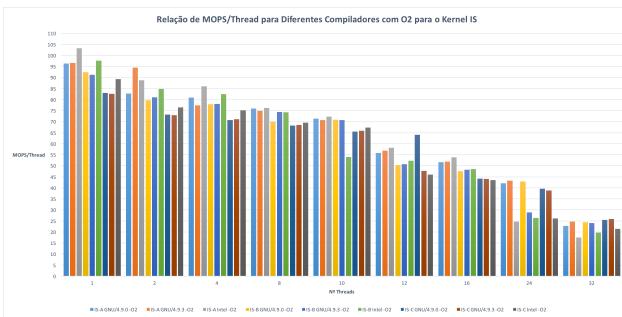


Figura 11. MOPS/Thread para Diferentes Compiladores com Flags -O2 para o Kernel IS

Os gráficos para o Kernel IS nas máquinas 641 podem ser consultados no anexo A.

5.3. Versão MPI

Para além da versão paralela OMP, era também objetivo deste trabalho efetuar testes com um paradigma de memória distribuída, neste caso MPI. Nesta versão efetuei testes para ethernet quer com compiladores da intel, quer com

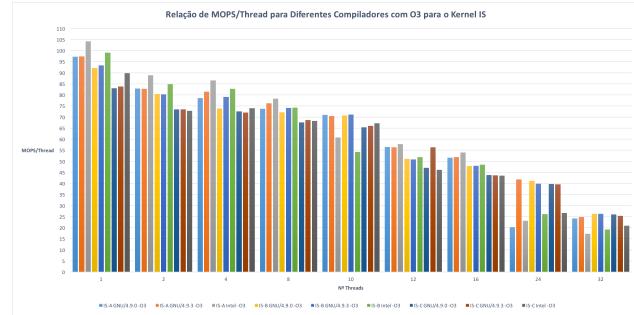


Figura 12. MOPS/Thread para Diferentes Compiladores com Flags -O3 para o Kernel IS

compiladores da gnu, testei com 8, 16 e 32 processos para cada classe de dados. De notar que os dados apresentados, dizem respeito a testes efetuados nas máquinas 641, apenas com *Ethernet* uma vez que não consegui obter dados para a execução com *Myrinet*.

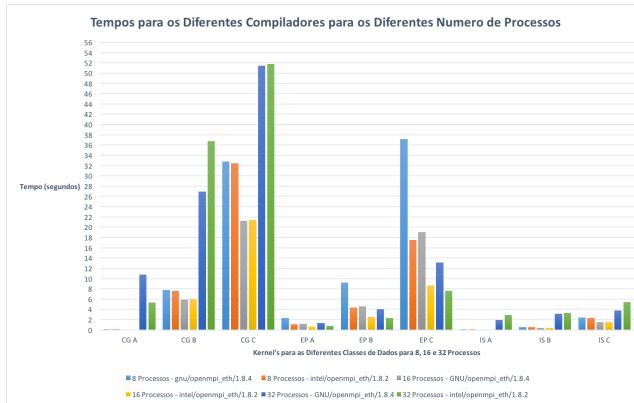


Figura 13. Tempos para os Diferentes Compiladores para os Diferentes Números de processos

No gráfico da figura 13 podemos consultar os tempos de cada *Kernel* para as diferentes classes de dados numa execução com 8 processos, 16 processos e 32 processos. Como podemos analisar o *Kernel* CG atinge o seu menor tempo, independentemente da classe de dados, com 8 processos, quer para o compilador da *GNU*, quer com o compilador da *Intel*.

No que diz respeito ao *Kernel* EP (Embaraçosamente Paralelo), este atinge o seu menor tempo com 32 processos para as classes de dados B e C com o compilador da *Intel* e para a classe A atinge o seu menor tempo com 16 processos mas também com o compilador da *Intel*. Por isto para este *Kernel* podemos concluir que o compilador da *Intel* é o mais eficiente.

Por fim o *Kernel* IS atinge o seu melhor tempo com 16 processos, qualquer que seja a sua classe de dados, sendo que os tempos de execução entre dois os compiladores, são muito próximos um do outro.

Quanto aos MOPS, podemos analisar o gráfico da figura 14, como podemos ver os MOPS para o *Kernel* CG tendem

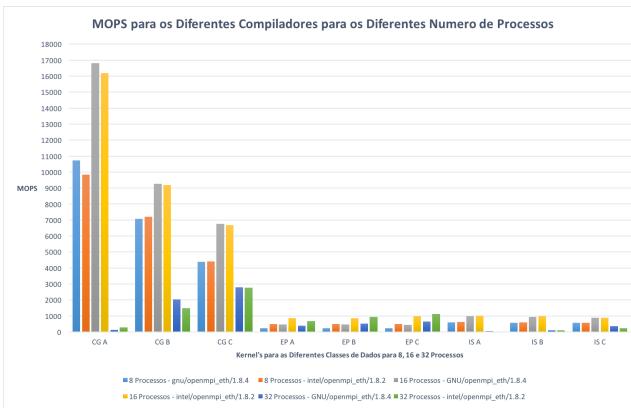


Figura 14. MOPS para os Diferentes Compiladores para os Diferentes Números de processos

a diminuir com 8 processos e com 16 processos, com o aumento da classe de dados sendo que com 32 processos, estes tendem a aumentar, ao aumentar a classe de dados. Pela a análise do gráfico podemos concluir que para o *Kernel CG* é mais eficiente executá-lo com 16 processos, uma vez que para as diferentes classes de dados, é com 16 processos que os MOPS são maiores.

No *Kernel EP* os MOPS apresentam aproximadamente os mesmos valores, quer para 8 e 16 processos, como pode ser comprovado pelo gráfico da figura 14, contudo, para 32 processos os MOPS aumentão ligeiramente com o aumento das classes de dados. Neste *Kernel* para as classes de dados B e C, o mais eficiente é executar o *Kernel* com 32 processos com o compilador da *Intel*, sendo que para a classe de dados A, o mais eficiente é executar o *Kernel* com 16 processos, com o compilador da *GNU*.

Por fim no *Kernel IS* os MOPS, mais uma vez tendem a ter aproximadamente os mesmos valores, havendo pouca variação, à medida que se aumenta a classe de dados, contudo é com 16 processos, que a execução deste *Kernel* mostra ser mais eficiente, tanto com o compilador da *GNU* como com o compilador da *Intel*.

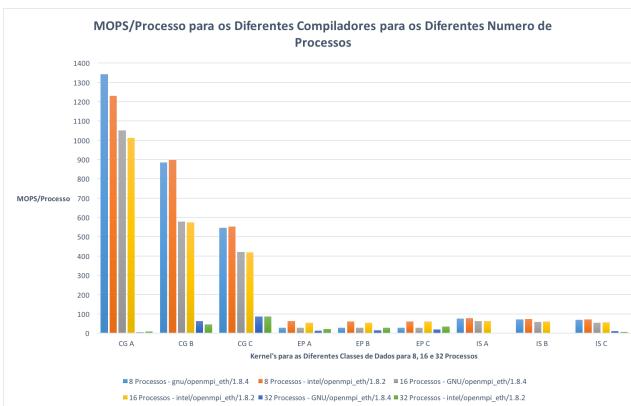


Figura 15. MOPS para os Diferentes Compiladores para os Diferentes Números de processos

O gráfico da figura 15 mostra a relação entre os diferentes *Kernel's* para as diferentes classes de dados, para 8, 16 e 32 processos. Pela a análise do gráfico, podemos verificar que para o *Kernel CG* à medida que se aumenta a classe de dados, de A até C os MOPS por processo diminuem, para 8 e 16 processos, sendo que para 32 processos, estes aumentam para os dois compiladores, o que já era de esperar uma vez que o comportamento dos MOPS é o mesmo.

No *Kernel EP* o comportamento dos MOPS por processo é semelhante ao comportamento dos MOPS, isto é, tendem a ter os mesmo valores (aproximadamente), à medida que se aumenta a classe de dados. Para o *Kernel IS*, os MOPS por processo tendem a ter os mesmos valores, à semelhança do que acontece com no gráfico da figura 14.

6. Ganhos

No que toca aos ganhos, decidi filtrar um bocado os dados uma vez que, caso não o fizesse o relatório iria ficar muito extenso. Com isto, decidi escolher o *Kernel IS* com as classes de dados A, B e C, compilado com o compilador da *Intel* com flag *-O3*.

6.1. Sequencial vs OMP

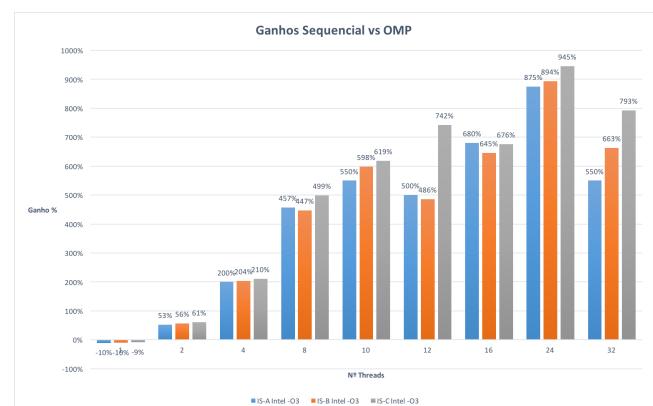


Figura 16. Percentagem de Ganhos Versão Sequencial vs Versão OMP

Como podemos verificar no gráfico da figura 16, os tempos da versão OMP começam mesmo por ter um ganho negativo, isto é, tem um tempo pior que a versão sequencial. Contudo, à medida que vai aumentando o numero de *Threads* os ganhos também vão aumentando, atingindo o seu máximo para 24 *Threads*. Para a classe de dados A, B e C com 24 *Threads* os ganhos são de 875%, 894% e 945% respectivamente, em relação à versão sequencial.

6.2. Sequencial vs MPI

Na relação de ganhos entre a versão sequencial e a versão MPI, eu fixei o compilador e as flags, sendo que apresento os ganhos para todas as classes de dados, e para o diferente numero de processos, para o qual efetuei os testes, como pode ser comprovado pela gráfico da figura 17.

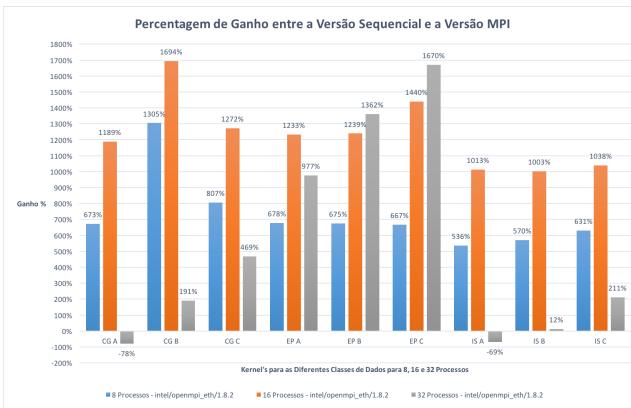


Figura 17. Percentagem de Ganhos Versão Sequencial vs Versão MPI

Como pode ser analisado pelo gráfico no *Kernel CG*, é com 16 processos que se atinge maior percentagem de ganhos da versão MPI em relação à versão sequencial, tanto para a classe de dados A, como B e como C, apresentando valores de 1189%, 1695% e 1272%. Já no *Kernel EP* a percentagem de ganho da versão MPI em relação à versão sequencial atinge o seu máximo com 32 processos para a classe B e C, com valores de 1392% e 1670% e para a classe de dados A atinge o seu máximo com 16 processos com um valor de 1233%.

Por fim, no *Kernel IS* a percentagem de ganho da versão MPI em relação à versão sequencial, atinge o seu máximo com 16 processos, independentemente da classe de dados, apresentando um valor de 1013% para a classe de dados A, 1003% para a classe de dados B e 1038% para a classe de dados C.

De referir que no caso do *Kernel CG* e do *Kernel IS*, com 32 processos para a classe de dados A, a percentagem de ganho chega a ser negativa, isto é, o tempo de execução da versão MPI é pior que o tempo de execução da versão sequencial.

7. Utilitários de Monitorização

No desenvolvimento deste trabalho, aquando de cada teste utilizei uma ferramenta de monitorização, no meu caso *dstat*, para examinar o estado do sistema de operação, neste caso o *Cluster Search*. Estas ferramentas permitem-nos visualizar a forma como o sistema é afetado em termos de utilização do *cpu*, da memória e dos discos aquando do aumento gradual da carga de computação.

Como referi, utilizei o comando *dstat* com as seguintes flags:

- c - ativa as estatísticas do *cpu* (*system, user, idle, wait, hardware interrupt e software interrupt*);
- d - ativa as estatísticas do disco (*read e write*);
- m - ativa as estatísticas de memória (*used, buffers, cache, free*).

Nesta secção irei analizar estas estatísticas para um dado *Kernel*, no meu caso para o *Kernel IS*, para o compilador da

Intel com flag *-O3* com a classe de dados C para cada uma das versões, isto é, para a versão sequencial, versão OMP, nas máquinas 431 e para a e versão MPI nas máquinas 641.

7.1. Versão Sequencial

Na versão sequencial apenas apresento os gráficos do *dstat* respetivos ao melhor tempo de execução.

No que toca ao uso do *CPU*, para o *Kernel IS* com a classe de dados C na máquina 431

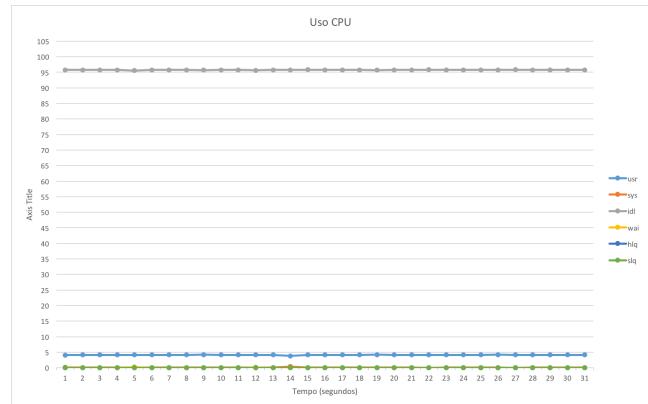


Figura 18. Uso do CPU

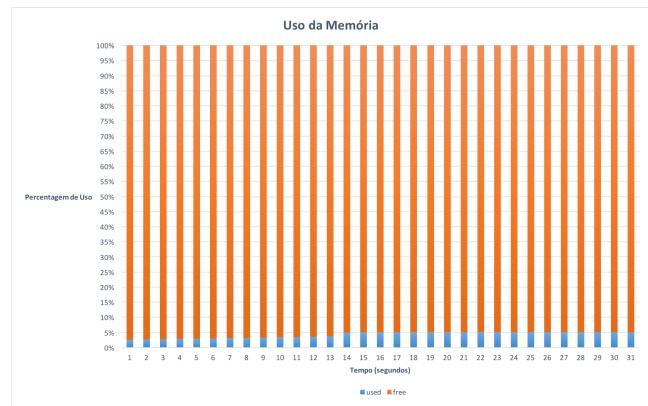


Figura 19. Uso da Memória

7.2. Versão OMP

Na versão OMP apenas apresento os gráficos do *dstat* respetivos ao melhor tempo de execução para o melhor número de *Threads*, isto é, para o número de *Threads* em que o tempo é menor.

7.2.1. CPU.

7.2.2. Memória.

7.2.3. Disco.

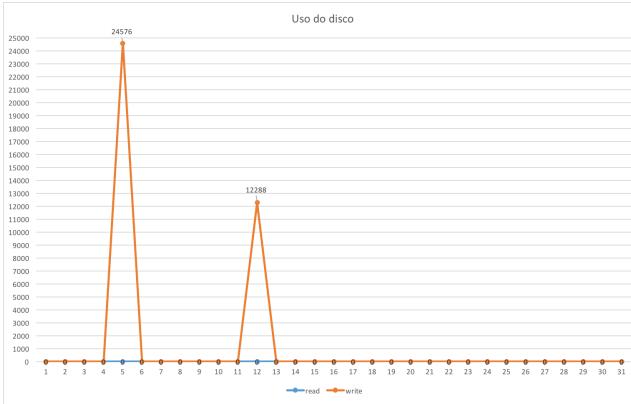


Figura 20. Uso do Disco

7.3. Versão MPI

Na versão OMP apenas apresento os gráficos do *dstat* respetivos ao melhor tempo de execução para o melhor número de Processos, isto é, para o número de Processos em que o tempo é menor.

7.3.1. CPU.

7.3.2. Memória.

7.3.3. Disco.

8. Conclusão

Depois de realizado todos os testes, do tratamento dos resultados e por fim a análise destes, posso concluir de uma maneira geral, que na versão sequencial na máquina 431, os tempos de execução são melhores, quando os *Kernel's* são compilados com o compilador da *Intel*, de notar que à medida que se aumenta os níveis de optimização, o intervalo de tempo de cada gráfico, também vai diminuindo, o que já era de esperar, uma vez que estamos a aumentar as optimizações. Quanto ao número de MOPS, também de uma maneira geral o compilador da *Intel* se mostra o mais eficiente, apresentando valores superiores em relação aos outros compiladores. De notar também que à medida que vamos aumentando os níveis de optimização, o intervalo de valores de cada gráfico vai aumentando, sendo que do nível -O2 para -O3 esse aumento não é muito significativo.

Quanto à versão OMP, podemos concluir que para o *Kernel IS* com um nível de optimização -O2 e -O3 e com a classe de dados C (classe de dados maior), que a versão 4.9.0 e a versão 4.9.3 do compilador da *GNU* são os compiladores mais eficientes, uma vez que é com a execução do *Kernel* compilado com estes compiladores que se encontra o menor tempo, sendo esse valor encontrado na execução do *Kernel* com 24 *Threads*. Quanto ao número de MOPS é com a versão do compilador 4.9.4 do compilador da *GNU* que se encontra o valor mais elevado de MOPS tanto para o nível de optimização -O2 e -O3, sendo esse valor encontrado com

24 *Threads* o que nos permite concluir que é este o melhor compilador, para a obtenção de melhores resultados desta métrica. No que toca aos MOPS/*thread* é com o compilador da *Intel* que se encontra o maior numero de MOPS/*Thread*, sendo este valor encontrado com apenas 1 *Thread*, o que podemos concluir que para a obtenção dos melhores valores desta métrica, é o compilador da *Intel* a melhor escolha. Contudo por uma análise do gráfico dos tempos, podemos concluir que com 1 *Thread* é onde se obtém o pior tempo.

No que toca à versão MPI, podemos concluir que o compilador da *Intel* é o compilador mais eficiente no que toca à obtenção dos melhores tempos, uma vez que de uma maneira geral, é com este compilador, numa execução com 16 processos (*mapped-by-core*), que se obtém os melhores tempos. Quanto ao número de MOPS, também de uma maneira geral é o compilador da *Intel* que é o melhor compilador para a obtenção dos melhores valores desta métrica, uma vez que é com este compilador que, de uma maneira geral, se encontra o maior número de MOPS, sendo estes valores encontrados com 16 processos. Por fim, quanto ao número de MOPS/Processo é o compilador da *Intel* que é o melhor, uma vez que é com este, que encontramos os melhores valores para esta métrica, contudo, ao contrário do que acontece com os tempos e com os MOPS, é com 8 processos que encontramos os melhores valores para esta métrica.

Referente à realização deste trabalho, efetuei inúmeros testes no *Cluster Search*, para a realização desses testes tive de criar uma forma de automatizar o trabalho, para isso recorri a *Shell Scripts* e ferramentas de escalonamento de tarefas, como o PBS. Depois de efetuados os testes, tive de processar os dados de forma a criar gráficos que exprimissem os meus resultados, para isso usei ferramentas de processamento de linguagens como é o caso do *gawk* e do *grep*, por fim, depois de ter todos os dados organizados, recorri ao excel para a geração dos gráficos.

Este trabalho permitiu-me ambientar de uma forma mais objetiva num ambiente de *clustering*, sendo que passei por algumas dificuldades principalmente na execução dos testes para a versão MPI, dificuldades essas que não foram totalmente ultrapassadas, pelo menos na execução em *Myrinet*. Para além dessas dificuldades, apesar de ter automatizado grande parte do trabalho com as *scripts*, penso que perdi demasiado tempo na criação dos gráficos, uma vez que os fiz todos à mão e não era esse o objetivo. Contudo, faço um balanço positivo da maior parte do trabalho, sendo que, como trabalho futuro pretendo aperfeiçoar principalmente o meu processo de automatização, de forma a não perder demasiado tempo em algumas partes, bem como conseguir efetuar todos os testes que defini ao inicio.

Referências

- [1] Nasa advanced supercomputing division. <http://www.nas.nasa.gov/publications/npb.html>.
- [2] Problem sizes and parameters in nas parallel benchmarks. http://www.nas.nasa.gov/publications/npb_problem_sizes.html.

- [3] Services and advanced research computing with htc/hpc clusters. http://search6.di.uminho.pt/wordpress/?page_id=55.
- [4] R. F. V. der Wijngaart and M. Frumkin. Nas grid benchmarks version 1.0. *NASA Technical Report NAS-02.005*, 7 2002.
- [5] H. Jin, M. Frumkin, and J. Yan. The openmp implementation of nas parallel benchmarks and its performance. *NAS Technical Report NAS-99-011*, 10 1999.

Apêndice

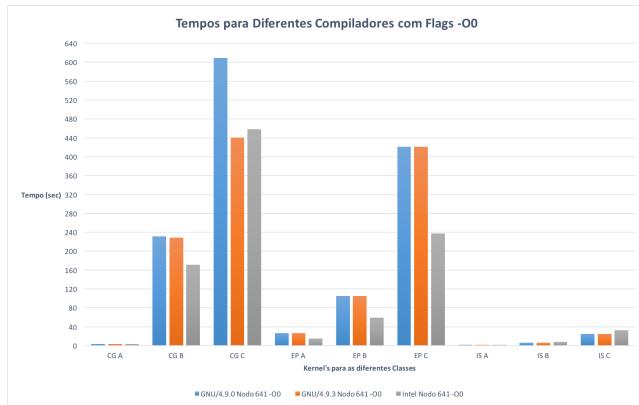


Figura 21. Tempos para Diferentes Compiladores com Flags -O0

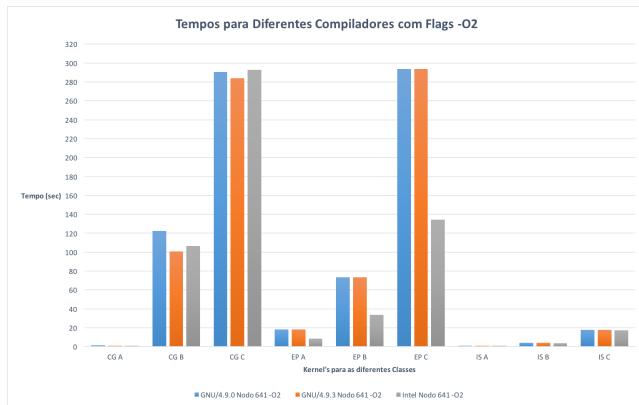


Figura 22. Tempos para Diferentes Compiladores com Flags -O2

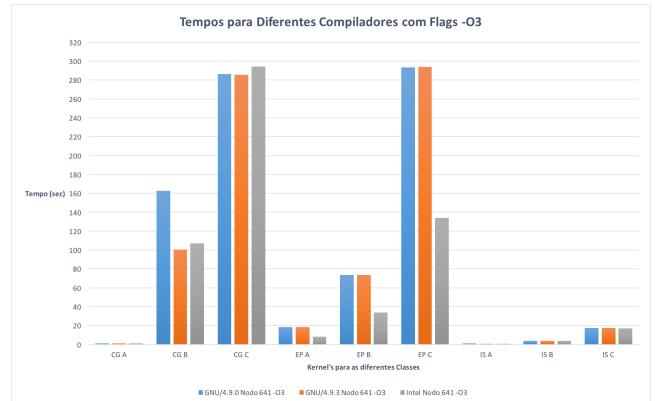


Figura 23. Tempos para Diferentes Compiladores com Flags -O3

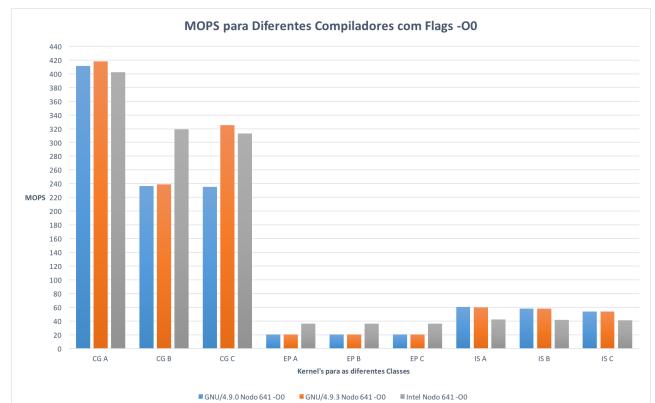


Figura 24. MOPS para Diferentes Compiladores com Flags -O0

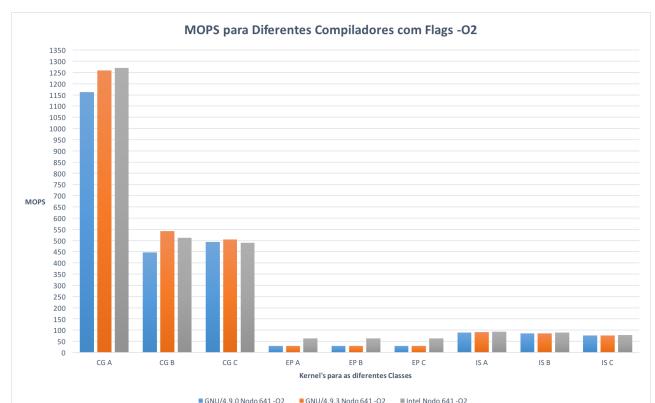


Figura 25. MOPS para Diferentes Compiladores com Flags -O2

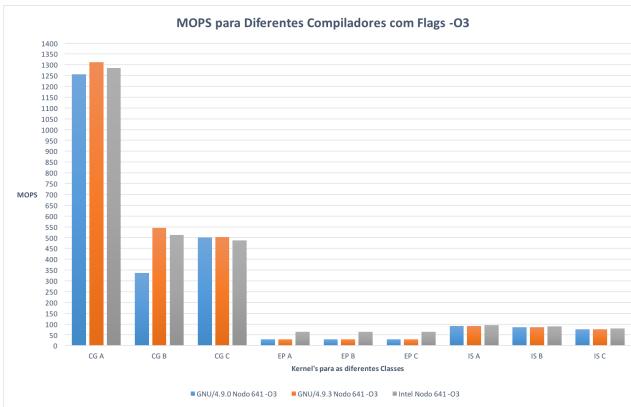


Figura 26. MOPS para Diferentes Compiladores com Flags -O3

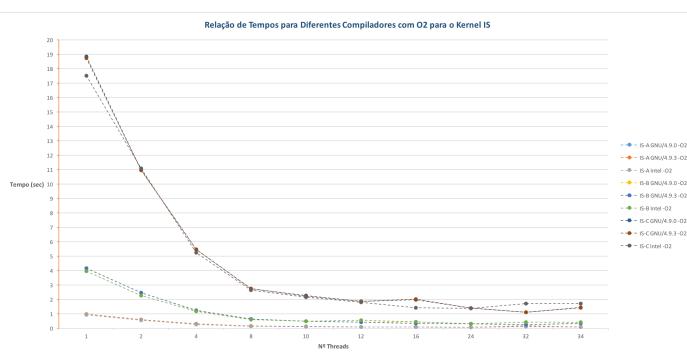


Figura 27. Tempos para Diferentes Compiladores com Flags -O2 para o Kernel IS

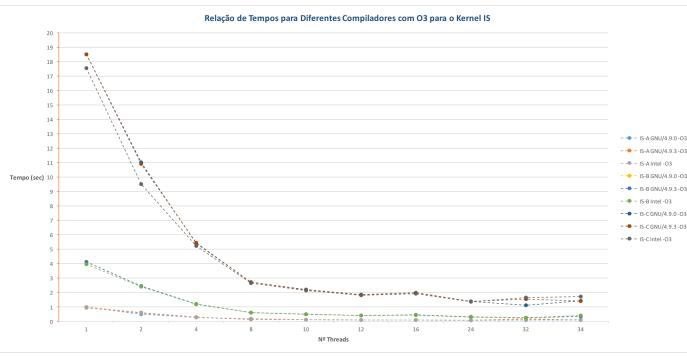


Figura 28. Tempos para Diferentes Compiladores com Flags -O3 para o Kernel IS

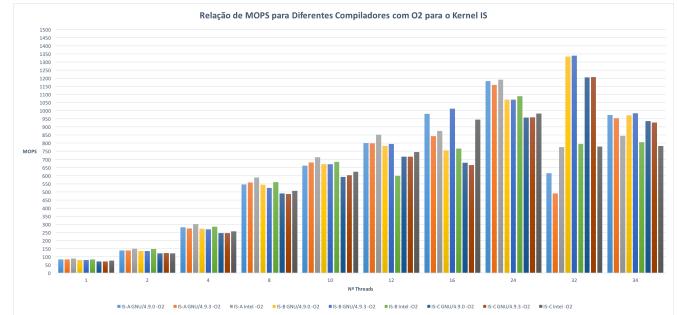


Figura 29. MOPS para Diferentes Compiladores com Flags -O2 para o Kernel IS

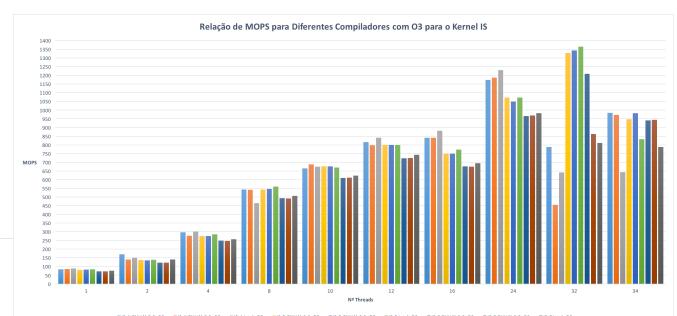


Figura 30. MOPS para Diferentes Compiladores com Flags -O3 para o Kernel IS

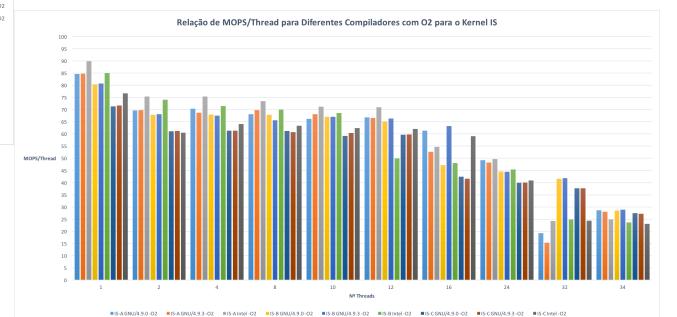


Figura 31. MOPS/Thread para Diferentes Compiladores com Flags -O2 para o Kernel IS

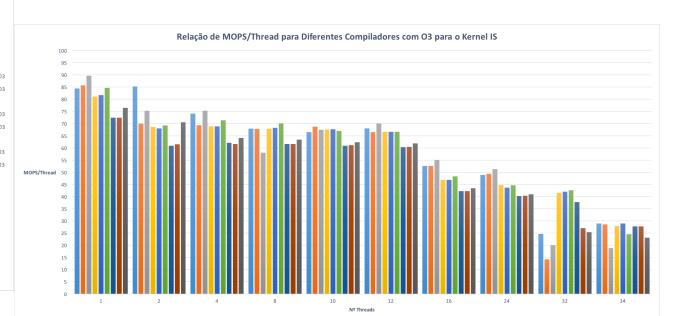


Figura 32. MOPS/Thread para Diferentes Compiladores com Flags -O3 para o Kernel IS