Execuções em Ambientes de Clustering

Universidade do Minho
Nuno Silva
A78156

***Abstract*— Com este trabalho prático pretende-se, apresentar uma análise sobre a execuções em ambientes de *Clustering*, neste caso irei utilizar SeARCH da Universidade do Minho. De forma a proceder a uma análise mais completa e aprofundada, irei utilizar *benchmarks* disponibilizados pelo NAS Parallel Benchmarks. Mais concretamente vou utilizar dois kernels e uma pseudo-aplicação, correndo os em duas classes distintas, sobre os quais nos permitem observar os comportamentos deste benchmarks em ambientes de memória partilhada, distribuída e numa vertente heterogénea.**

I. Introdução

No mundo da investigação, é comum encontrar problemas que sejam computacionalmente intensos e que exijam uma quantidade enorme de recursos. Um administrador de sistemas de computação, tem conhecimentos para proceder a uma análise do desempenho de programas, tendo em conta as características dos sistemas e o que elas influenciam no resultado final.

Assim sendo, tendo o intuito de medir as performances de supercomputadores submetendo os a diversos cenários de teste, a divisão de supercomputação avançada da NASA, desenvolveu as NAS Parallel Benchmarks. Onde estes, benchmarks, são derivados de uma dinâmica computação fluídos[6], atualmente o NAS disponibiliza benchmarks ”para todos os gostos”, em que cada uma deles poderá ser corrida em diferentes ambientes/classes.

Com isto decidi analisar dois kernels e uma pseudo-aplicação MZ(Multi-zone), possibilitando testar uma implementação híbrida (**OpenMP + MPI**). Posto isto, os benchmarks selecionados foram os seguintes:

- **IS** (Ordenação inteiros) - Neste kernel é gerado um array de grandes dimensões e ordena-lo; Onde neste cenário se poderá testar um ambiente memória partilhada;[7]
- **EP** (”Embaraçosamente paralelo”) - Calcula números aleatórios, de uma forma paralela, sobre o qual poderá ser ideal para testar o comportamento do sistema num ambiente de memória partilhada;[7]
- **SP-MZ** - Aplicações como SP ou BT, resolvem versões de equações instáveis de Navier-Stokes em três dimensões espaciais. No entanto uma única malha não é suficiente para descrever o domínio e então são usadas várias malhas/zonas para cobri-lo. Então esta pseudo-aplicações permitem testar cenários híbridos;[8]

II. Especificações Hardware

Primeiro de tudo, realizei uma análise sobre o hardware que foi disponibilizado, com isto pretende-se identificar qualquer *bottleneck* ou qualquer limitação que possa levar a perdas de desempenho. Posto isto, decidi utilizar os nodos 662 e 652, com um *Intel Xeon Processor E5-2695-V2* e *E5-2670v2*. As especificações destes nodos foram obtidas diretamente do cluster e do website das unidades processamento.[1] [2] [3][4][5].

Especificações Cluster

	Processador	
Nodo	**662**	**652**
Nome código	Ivy Bridge	Ivy Bridge
Modelo	Intel Xeon E52695v2	Intel Xeon E52670V2
# Cores	12	10
# Threads	24	20
Frequência base	2.4 GHz	2.5 GHz
Frequência turbo	3.2 GHZ	3.3 GHz

	Memória
Nodo	**662**
Cache L1	12 x 32 Kib para Dados (8-way) 12 x 32 kib para Instruções (8-way)
Cache L2	12 x 256 Kib
Cache L3	30 Mib
Max Memory Bandwidth	59.7 GiB/s
Main Memory	64 GiB

	Memória
Nodo	**652**
Cache L1	10 x 32 Kib para Dados (8-way) 10 x 32 kib para Instruções (8-way)
Cache L2	10 x 256 Kib
Cache L3	25 Mib
Max Memory Bandwidth	59.7 GiB/s
Main Memory	64 GiB

III. Especificações Software

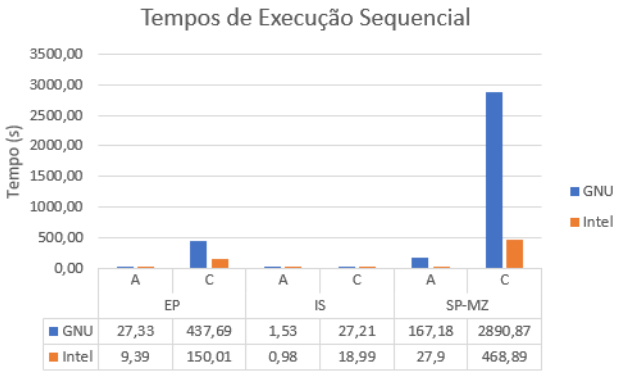
De forma a proceder a testes mais específicos e por sua vez uma conclusão mais corroborada. Decidi realizar testes tendo em conta também o software, neste caso o compilador, daí é pretendido avaliar a performance de um kernel.

Assim sendo verifiquei, o que cada um dos nodos dispõe, sendo assim concluí seguir com os compiladores **GNU** e **Intel**, bem como diferentes níveis de otimização, **O2** e **O3**.

Para um olhar mais profundo sobre o sistema, e como se comporta com a execução de uma aplicação, utilizei o **dstat**, **sar** e **mpstat** dado que permite analisar métricas mais especificas para avaliar performance.

IV. Análise de Classes

Para uma melhor compreensão dos resultados obtidos, é preciso analisar os comportamentos de cada aplicação tendo em conta a classe onde foi executada. Assim sendo, forma escolhidas duas classes **A** e **C**, onde estas apresentam testes *"standard"*, e em que apresentam tamanhos diferentes para o problema.



	EP A	EP C	IS A	IS C	SP-MZ A	SP-MZ C
GNU	27,33	437,69	1,53	27,21	167,18	2890,87
Intel	9,39	150,01	0,98	18,99	27,9	468,89

Fig. 1: Tempos de execução para uma configuração utilizando **GNU** e **Intel**, sem otimizações, no nodo 662

Como podemos comprovar a partir da figura 1, verificamos uma disparidade entre os tempos obtidos entre as classes **A** e **C**, comprovando a diferença de tamanho entre os problemas. Justificando assim a utilização de ambas as classes, uma vez que abordamos a evolução deles em ambientes de maior e menor dimensão.

V. Becnhmarking num Ambiente Sequencial

Um dos cenários sobre o qual será feita uma análise neste trabalho, será a execução de aplicações em ambientes sequenciais, nos diversos nodos e combinações de software.

A. *Tempos Sequenciais em Diferentes Nodos*

Tal como indicado previamente, os testes foram executados em diferentes arquiteturas e com diferentes combinações de otimização. Onde evidentemente, será relevante averiguar o quão impactante será a arquitetura e as otimizações, relativamente aos tempos de execução.

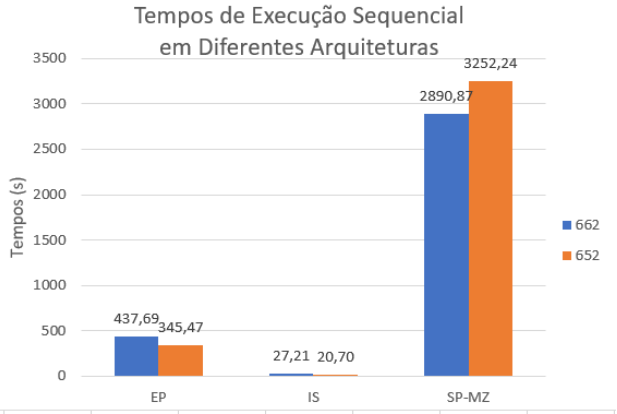


Fig. 2: Tempos de configuração pelos diferentes nodos, para uma configuração utilizando **GNU** e sem otimizações

A partir do gráfico anteriormente exposto, verificamos que na maior parte dos casos o nodo 652 apresenta os menores tempos de execução. Onde estes mesmo estão influenciados devido a este nodo apresentar uma frequência maior, relativamente ao 662.

B. *GNU vs Intel*

Aqui serão expostas e analisadas as diferenças entre os compiladores utilizados, o da **GNU** e da **Intel**. Tendo em conta que um deles é *open source* e o outro proprietário, foram corridos e registados os resultados dos kernels escolhidos, no nodo 662.

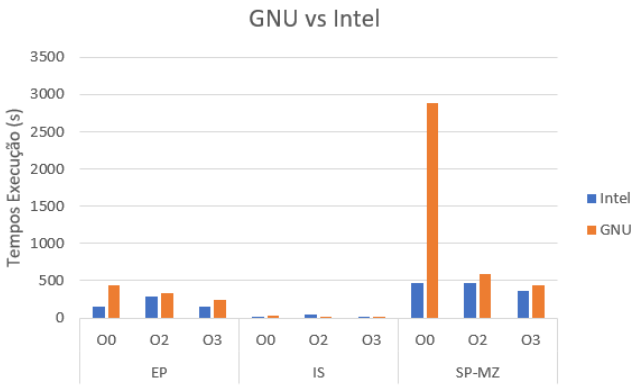


Fig. 3: Tempos de execução dos kernels, utilizando **GNU** ou **Intel**, no nodo 662

Após uma análise sobre a figura 3, verificamos que o compilador da **Intel** apresenta de uma forma geral, os melhores resultados, em relação ao da **GNU**. Isto devido a um processador da **Intel** possuir *features* que melhoram a performance geral de um kernel, tais como, vectorização SIMD (*Single Instruction Multiple Data*) e integração com as suas bibliotecas performance. Para além da possibilidade de otimizar e vetorizar o código conforme a arquitetura em questão.[12]

C. Ganhos com O2 e O3

Nesta subsecção pretendemos analisar, o impacto das otimizações sobre os testes em questão. Tendo em conta os registos verificados na figura 2, de seguida apresentamos os ganhos utilizando as otimizações disponíveis no nodo 652.

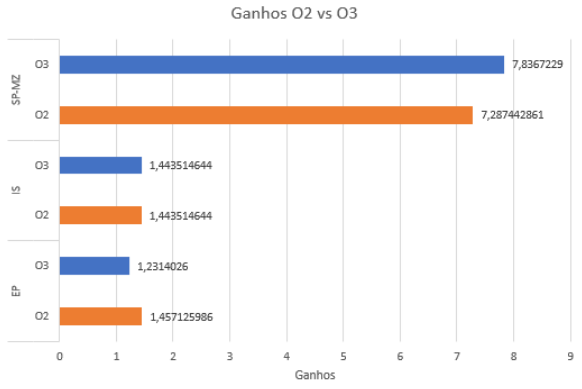


Fig. 4: Ganhos das versões otimizadas utilizando a classe C e **GNU**, no nodo 652

Tal como podemos averiguar, a partir da figura anterior, na maior parte dos casos, a utilização da flag **O3**, não apresenta ganhos superiores a **O2**. Isto devido a **O3**, apesar de implementar *loop unrolling* e *vectorização* que reduzem o número de instruções, aumenta o CPI.

D. % CPU utilizado

Através do gráfico que de seguida será apresentado, verificamos a exposição da % de CPU consumido por parte do kernel, ao longo da sua execução.

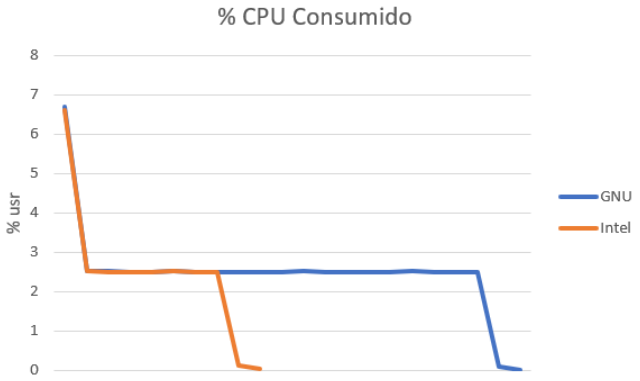


Fig. 5: Consumo CPU por parte do kernel **EP** na classe **A** com otimizações **O3**, em cada um dos compiladores utilizados

Como podemos verificar, o kernel **EP**, na classe **A** apresenta sensivelmente as mesmas percentagens ao longo do tempo, variando claro está, e tal como mencionado anteriormente, o tempo de execução.

Na secção seguinte, iremos estudar os consumos de memória por parte do mesmo kernel e classe. E tentar compreender de que forma esse consumo poderá influenciar os valores do CPU utilizado.

E. Consumo Memória

Tendo em conta o % CPU utilizado, decidimos analisar os consumos de memória de forma a concluir se estes consumos estão diretamente correlacionados com os do CPU.

Como podemos verificar pela figura 6, notamos que o consumo de memória pelos compiladores **GNU** e **Intel**, não distam por muito entre si, para além de que estes consumos mantêm-se quase constantes ao longo do tempo. Porém, o kernel compilado com **Intel**, consome mais recursos, levando-nos a querer que processa mais instruções por segundo, ao qual iremos verificar de seguida.

F. MOP/s : GNU vs Intel

Com o intuito de compreender a quantidade de instruções executadas, por compilador, mais concretamente pelos kernel e otimizações a ele aplicadas, foi

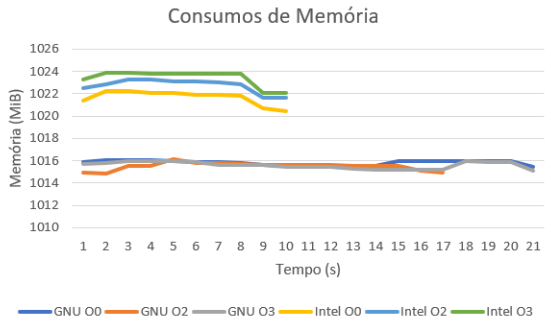


Fig. 6: Consumo de memória no nodo 652, para o kernel **EP** e classe **A**

gerado o seguinte histograma. Ao qual se nota que para a maior parte dos casos, uma aplicação compilada com **GNU** apresenta menos milhões de operações por segundo (MOP/s). Evidenciando que implica menos saturação sobre o CPU, e por sua vez piores tempos de execução.

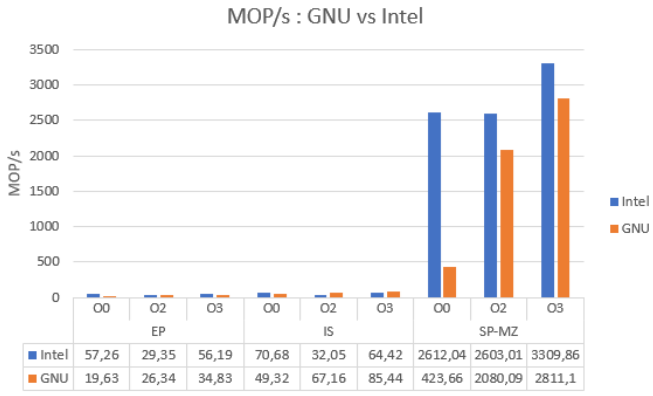


Fig. 7: Registos de MOP/s obtidos para as diferentes combinações de compilação, no nodo 662

VI. Benchmarking num ambiente de memória partilhada

Após uma análise sobre o comportamento dos kernels, com várias combinações de compilação num ambiente sequencial, procedemos a uma nova fase, neste caso num ambiente de memória partilhada.

Assim sendo iremos seguir com a mesma abordagem, enunciadas anteriormente para execuções sequenciais, contudo neste ambiente analisaremos também os ganhos na paralelização dos kernels para os mesmo cenários de compilação.

A. GNU vs Intel

Nesta secção pretende-se observar os tempos obtidos num ambiente de memória partilhada. Assim é demonstrado os tempos de execução, dos kernels selecionados na classe C ao longo das otimizações implementadas.

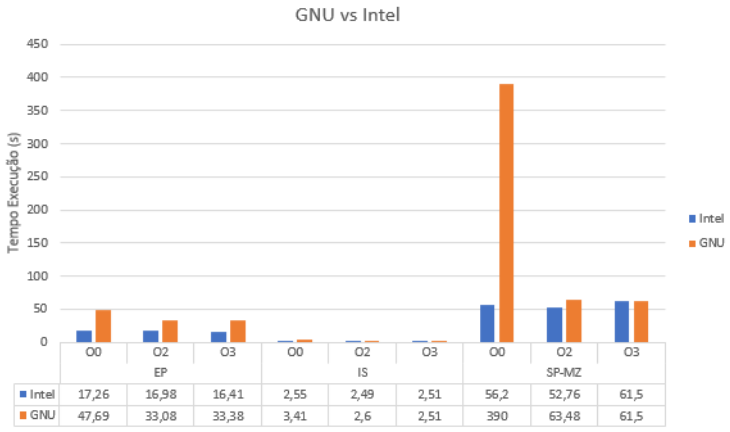


Fig. 8: Tempos de execução dos kernels, utilizando 8 *threads* em **GNU** e **Intel**, no nodo 652

Como podemos observar, e tal como podemos verificar o compilador da **Intel** apresenta os melhores resultados em praticamente a totalidade dos casos recolhidos, o mesmo se sucedeu num ambiente sequencial. Isto sucede-se pois os compiladores da **Intel** estão integrados com as mais recentes atualizações de bibliotecas como *OpenMP*, permitindo um multiprocessamento simétrico, e por sua vez a possibilidade da utilização tecnologias como *Hyper-Threading*[13].

B. Ganhos com O2 e O3

Tal como a sua homologa num ambiente sequencial, nesta secção pretendemos expor o impacto das flags **O2** e **O3**, juntamente com a paralelização do kernel em questão.

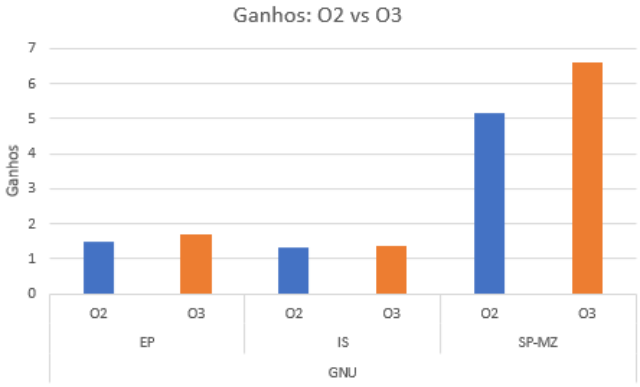


Fig. 9: Ganhos das versões otimizadas para 4 Threads, utilizando a classe C, **GNU**

Como podemos constatar, a utilização de flags de otimização **O2** e **O3**, apresentam ganhos relativamente

a não utilizar a uma flag como estas. Comparativamente as duas flags utilizadas, **O3** apresenta os melhores resultados para todos os kernels. De seguida iremos, analisar o comportamento destas flags, compiladas juntamente com o compilador da **Intel**.

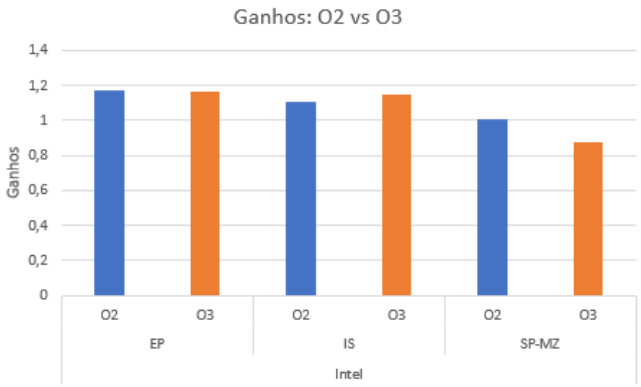


Fig. 10: Ganhos das versões otimizadas para 4 Threads, utilizando a classe C, **Intel**

Neste caso os ganhos não apresentam ser muito significativos, chegando a ser prejudicial como é o caso do kernel SP-MZ com a flag **O3**, que leva a um aumento do CPI. Em relação as flags, comparando as duas notamos que apresentam ganhos muito próximas. Assim sendo, a utilização deste tipo de flags para um compilador deste tipo não apresenta ser muito vantajoso.

C. % CPU utilizado

Tendo em conta o cenário estudado nesta mesma secção será de esperar que as taxas de utilização do CPU aumentem significativamente, visto utilizarmos mais recursos.

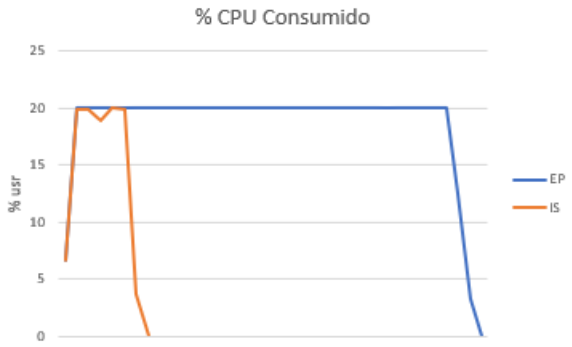


Fig. 11: Utilização do CPU por parte dos kernels **EP**, **IS**, num compilador **GNU** e utilizando **O3**, no nodo 652

Tal como podemos ver, e comparando o gráfico do ambiente sequencial, o CPU consumido pelos kernels **EP** e **IS** quando executados com 8 threads, apresentam aumentos de até 8%. Assim sendo, e tendo também em conta o ocorrido no ambiente sequencial, será expectável que os mesmos aumentos se sucedam com o compilador da **Intel**.

D. Consumo Memória

Dado estarmos a analisar um ambiente de memória partilhada, é compreensível a exposição dos consumos de memória neste de ambiente.

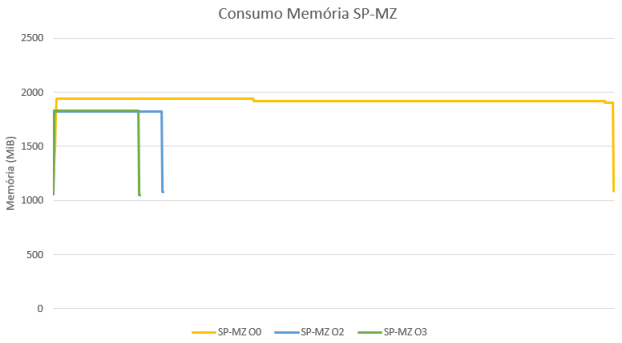


Fig. 12: Consumo memória por parte do kernel **SP-MZ**, compilado com **GNU**, no nodo 652

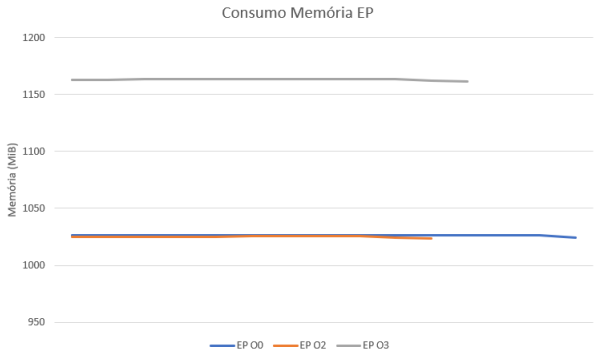
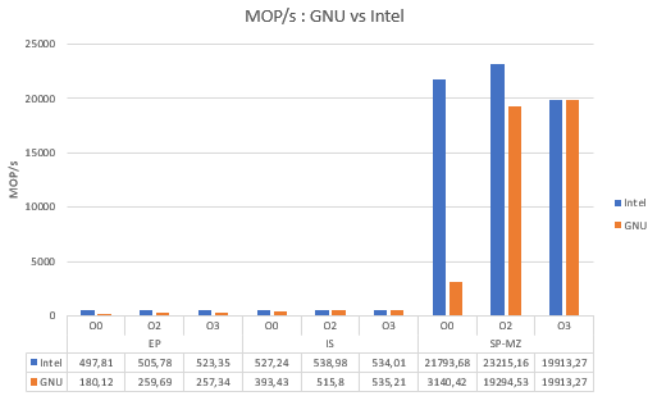


Fig. 13: Consumo memória por parte do kernel **EP**, compilado com **GNU**, no nodo 652

Pela figura 13, notamos num aumento de memória consumido por parte do kernel **EP**, este mesmo foi executado utilizando 4 *threads*, evidenciando que um ambiente paralelo consome mais memória, para este caso particular o aumento chega a 200MiB. Relativamente ao kernel **SP-MZ**, verificamos um consumo muito elevado, quase o dobro, comparativamente ao kernel **EP**, onde este consumo se mantém quase constante por longos períodos de tempo.

E. MOP/s: GNU vs Intel

Tendo analisado previamente os consumos de memória e do cpu, para este tipo de ambiente, e constatado o aumento destes mesmos, nada mais será expectável do que o aumento do número de instruções executadas por segundo.



	EP O0	EP O2	EP O3	IS O0	IS O2	IS O3	SP-MZ O0	SP-MZ O2	SP-MZ O3
Intel	497,81	505,78	523,35	527,24	538,98	534,01	21793,68	23215,16	19913,27
GNU	180,12	259,69	257,34	393,43	515,8	535,21	3140,42	19294,53	19913,27

Fig. 14: Registos de MOP/s obtidos para as diferentes combinações de compilação, no nodo 652

Tal como esperado, houve um aumento significativo no número de instruções, devido a tomarmos partido da paralelização, permitindo executar mais operações num mesmo período de tempo.

F. Escalabilidade

Um fator muito relevante para a performance de uma aplicação, e por sua vez também para o caso de estudo em questão, é a escalabilidade das implementações. Neste tipo de ambiente, previamente estudado, é importante compreender o comportamento das aplicações, em cenários de *multi-thread*. Assim sendo é possível visualizar o melhor cenário de execução, de forma a obter a melhor performance de um kernel.

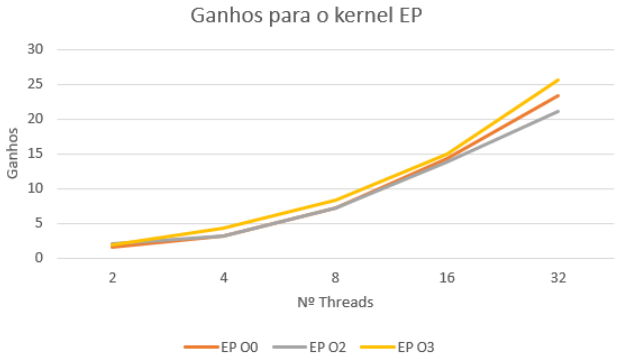


Fig. 15: SpeedUps para o kernel EP na classe C, no nodo 652

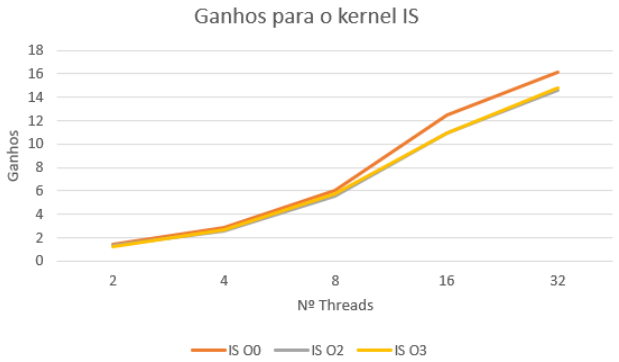


Fig. 16: SpeedUps para o kernel IS na classe C, no nodo 652

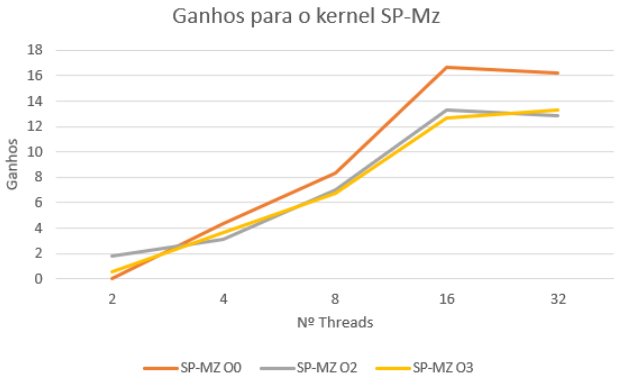


Fig. 17: SpeedUps para o kernel SP-MZ na classe C, no nodo 652

A partir das figuras 7, 8 e 9, aqui apresentadas, verificamos que os kernels escolhidos apresentam um boa escalabilidade, para um único socket, nos diferentes cenários de compilações em **GNU**.

VII. Benchmarking num ambiente de memória distribuída

Concluída uma análise sobre o comportamento dos kernels selecionados, em ambientes sequenciais e de memória partilhada, nada mais seria previsível do que estudar num ambiente de memória distribuída, através de **Open MPI**.

A utilização deste tipo de biblioteca, traz-nos regalias como por exemplo correr em qualquer tipo de arquiteturas de memória, pode ser utilizado numa maior variedade problemas do que OpenMP, porém esta limitado pelas redes de comunicação entre nodos. [10]

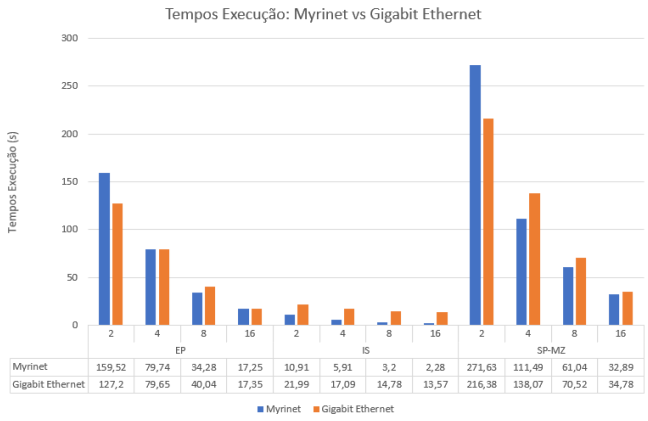
Assim sendo, é relevante entender, como a distribuição de carga afeta o poder computacional, comparativamente a outros ambientes, previamente estudados.

Para proceder a uma análise sobre este ambiente, foi utilizado o nodo 662, possuindo tecnologias distintas, *ethernet* e *myrinet*. No momento de execução, o mapeamento dos processos é feito pelos nodos, especificando assim a opção *-map-by node*.

A. *Tempos de Execução: Gigabit Ethernet vs Myrinet*

Como mencionado anteriormente, o nodo escolhido para o processamento dos kernels selecionados, esta conectado por *Gigabit Ethernet* e *Myrinet*, contendo uma largura de banda de 1Gb/s e 10Gb/s (baixa latência), respetivamente.[1][11]

Com uma diferencia tão elevada entre as larguras de banda que cada um disponibiliza, é de esperar que as execuções que utilizem *Myrinet*, apresentem os melhores resultados.



	EP				IS				SP-MZ			
	2	4	8	16	2	4	8	16	2	4	8	16
Myrinet	159,52	79,74	34,28	17,25	10,91	5,91	3,2	2,28	271,63	111,49	61,04	32,89
Gigabit Ethernet	127,2	79,65	40,04	17,35	21,99	17,09	14,78	13,57	216,38	138,07	70,52	34,78

Fig. 18: Tempos de execução dos kernels na classe C, usando Gigabit ethernet e Myrinet com otimizações **O3**, no nodo 662

Como indicado anteriormente, os kernels executados com *Myrinet* apresentam os melhores resultados na maior parte dos casos, fazendo-se de notar a diferença de tempos no kernel **IS**.

Como podemos verificar, com o aumento de processos, os tempos de execução em ambos os sistemas decrescem, sendo *Myrinet* a que apresenta os decréscimos mais acentuados.

B. *OpenMP vs Open MPI*

Neste tipo de ambientes que estamos a analisar, o tempo de execução de um programa comparativamente com um desenvolvido para memória partilhada, é prejudicada devido a distribuição de carga pelos processadores, e pela quantidade de comunicações existentes.

Dessa forma o gráfico seguinte, tem como objetivo entender o peso que os tempos de comunicação entre os processadores e por sua vez a distribuição de cargas, têm no tempo execução final.

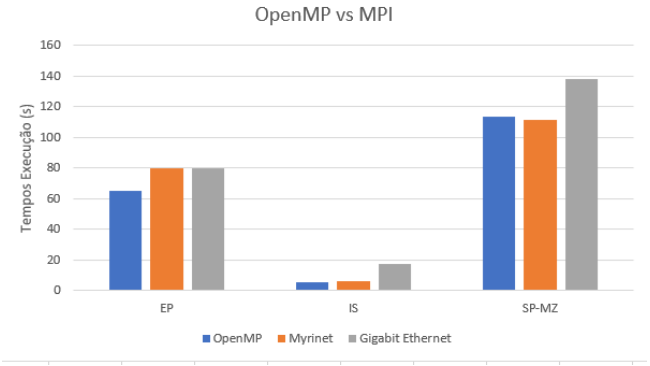


Fig. 19: Tempos de execução OpenMP vs Open MPI, no nodo 662 e com otimizações **O3**

Olhando para o gráfico da figura 19, verificamos que os kernels executados em memória partilhada, apresentam os menores tempos de execução. Isto se deve, pois, os custos de comunicação entre processos são desprezíveis.

C. *Ganhos utilizando O2 e O3*

Tendo visualizado anteriormente, o peso das comunicações e as da distribuição das cargas, existe uma necessidade de compensar esses gastos, dessa forma poderemos recorrer a flags de otimização, tal como mencionado em secções anteriores.

O gráfico seguinte, representa os ganhos do kernel **SP-MZ** utilizando as flags **O2** e **O3**, ao longo dos processos utilizados.

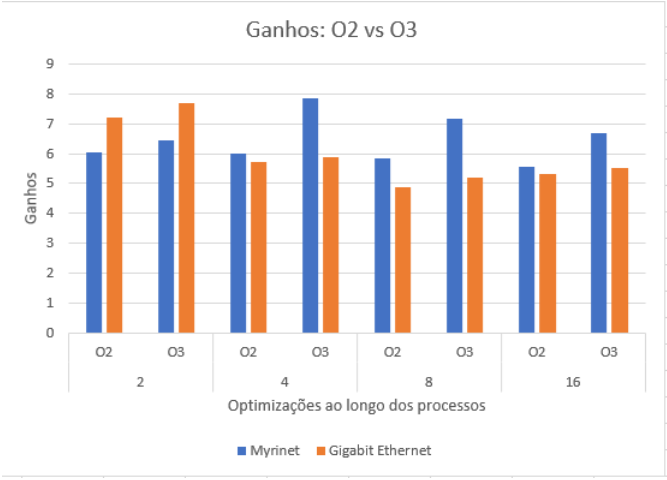


Fig. 20: Ganhos obtidos pelas flags O2 e O3, no nodo 662 para o kernel SP-MZ

Como podemos constatar, os ganhos são significativos para o kernel **SP-MZ**. Relativamente aos ganhos respetivos por cada sistema de rede, a *Myrinet* apresenta os melhores resultados a partir dos quatro processos.

D. Utilização da Rede

Com uma mudança na tecnologia, cuja influência do sistema de rede é elevada, é importante entender o peso que os tempos de comunicação têm sobre cada kernel. Assim sendo os gráficos seguintes, representam graficamente a quantidade de pacotes enviados/recebidos, por sistema de rede, ao longo da sua execução, para o kernel **EP** na classe **C**, com otimizações **O3** e 8 processos.

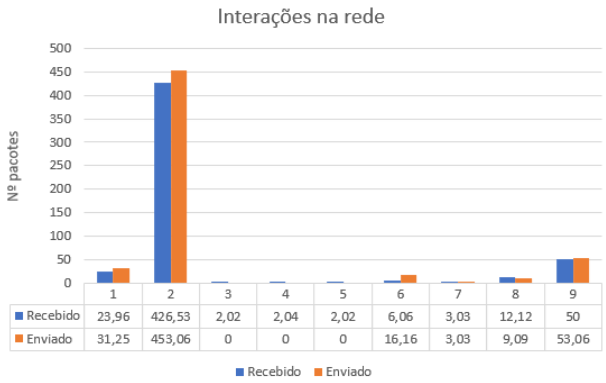


Fig. 21: Pacotes enviados/recebidos, para um sistema *Gigabit Ethernet*

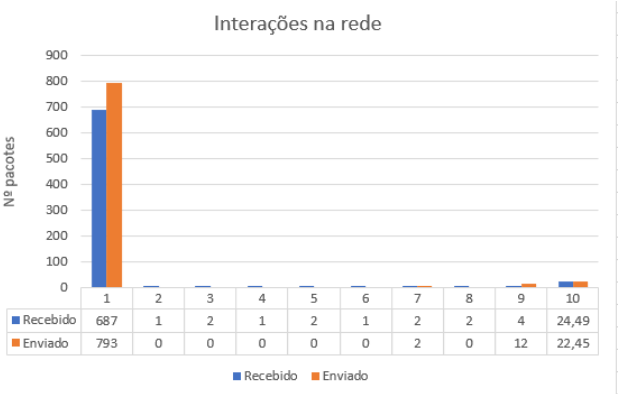


Fig. 22: Pacotes enviados/recebidos, para um sistema *Myrinet*

Como podemos ver, um sistema de rede como o *Myrinet*, consegue enviar e receber mais pacotes no inicio da execução. Isto é devido as discrepâncias entre as larguras de banda entre *Myrinet* e *Gigabit Ethernet*. Ou seja, num mesmo período de tempo uma rede *Myrinet* é capaz de enviar e receber mais pacotes, promovendo assim um tempo de execução possivelmente menor.[14]

E. % CPU Consumido

Tendo por base os tempos registados na figura 19, verificamos que uma implementação em MPI, não possui muitas vantagens relativamente a um ambiente de memória partilhado. Tendo em conta que neste tipo ambiente, os recursos se encontrarem repartidos, é de esperar que a % de CPU utilizado seja menor do que por exemplo em OpenMP.

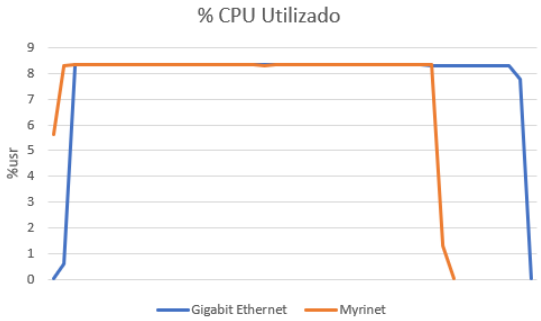


Fig. 23: CPU Consumido

Como podemos verificar, uma implementação neste tipo de ambiente utiliza cerca de 10% menos de utilização de CPU, comparativamente ao consumo numa implementação OpenMP. Olhando para este resultado, poderemos esta presentes perante um caso de desaproveitamento dos recursos disponibilizados, assim sendo, será caso de estudo, posteriormente, um paradigma híbrido.

F. Consumo Memória

Relativamente ao consumo de memória do sistema, é expectável que o consumo seja maior do que aquele registado num ambiente de memória partilhado, visto que os dados utilizados se encontram dispersados pelos cores.

Tal como o esperado, o consumo de memória foi superior aos registados num ambiente de memória partilhada, chegando mesmo no caso de um implementação *Gigabit Ethernet* a apresentar o dobro do consumo. Entre as duas implementações de sistemas de redes, é compreensível que *Myrinet* apresente valores mais baixos de consumo, visto que a sua largura de banda permite a transferência de uma maior quantidade de dados, num mesmo período de tempo, tal como mencionado anteriormente.

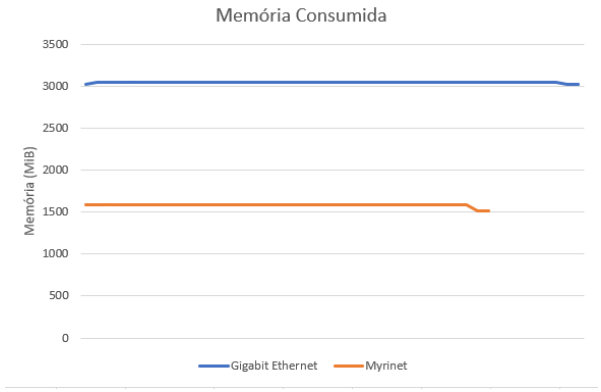


Fig. 24: Consumo memória registado pela aplicação **EP**, na classe C e com otimizações **O3**

VIII. Benchmarking num ambiente Híbrido: OpenMP + MPI

Com a evolução dos sistemas e com o intuito de obter o máximo de performance, caminhamos, então, cada vez mais para arquiteturas de tipo NUMA com múltiplos nós. Uma solução possível para tirar partido desta mesma arquitetura, será um paradigma híbrido, entre **OpenMP** e **MPI**, pretendendo assim reduzir os custos de comunicação entre processos, e simultaneamente aproveitar os vários cores disponíveis.

A. *Gigabit Ethernet vs Myrinet*

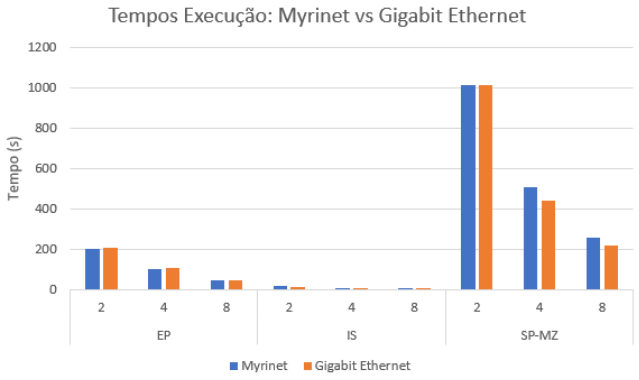


Fig. 25: Tempos de execução de um paradigma híbrido, utilizando *Myrinet* e *Gigabit Ethernet*

A partir do gráfico anteriormente exposto, verificamos que os sistemas de conexão não apresentam uma grande diferença temporal entre sistemas. Relativamente para kernels menos complexos, a utilização de *Myrinet* apresenta ser mais vantajosa, apesar da diferença ser mínima. Por outro lado, para uma aplicação como **SP-MZ**, a utilização de *Gigabit Ethernet*, apresenta diferenças mais acentuadas, e por sua vez mais vantajosas para a sua utilização.

B. *Tempos Execução*

De seguida é apresentado dois testes, comparando os ambientes previamente estudados com este paradigma ao qual estamos a analisar de momento. Assim sendo o objetivo passa por assimilar as vantagens em utilizar este paradigma em vez de um destes ambientes.

1) Híbrido vs MPI: Tendo em conta os resultados obtidos num ambiente de memória distribuída, e com o propósito de melhorar esses mesmo através de um paradigma híbrido. Foi predefinido, a utilização de 4 threads sobre o qual apenas irá modificar o número processos neste ambiente ao qual estamos a analisar de momento.

De seguida é apresentado dois histogramas, cada um deles com os resultados obtidos em sistemas de rede *Myrinet* ou *Gigabit Ethernet*, utilizando a classe **C** e otimizações **O3**.

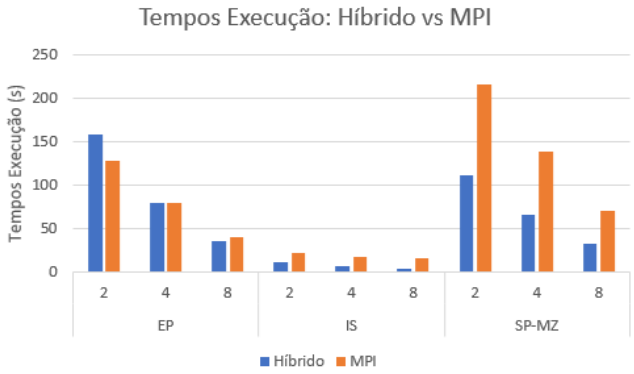


Fig. 26: Tempos de execução, utilizando um sistema Gigabit Ethernet

Como podemos verificar pelas figuras 25 e 26, a utilização deste tipo de paradigma só se justifica para kernel mais complexos, como é o caso do **SP-MZ**, independentemente do sistema de redes utilizado. Contudo de uma forma mais geral, apesar do *Myrinet* apresentar tempos mais baixos, não existe grande diferença ente a utilização de **MPI** ou **Híbrido**, com este sistema. Caso que não se verifica na utilização de *Gigabit Ethernet*, onde as diferenças temporais são mais evidentes para a maior parte dos casos.

2) Híbrido vs OpenMP: Tal como verificado na figura 19, onde é mostrado as diferenças temporais entre um ambiente de memória distribuída e partilhada. Onde é notório o impacto dos tempos de comunicação,

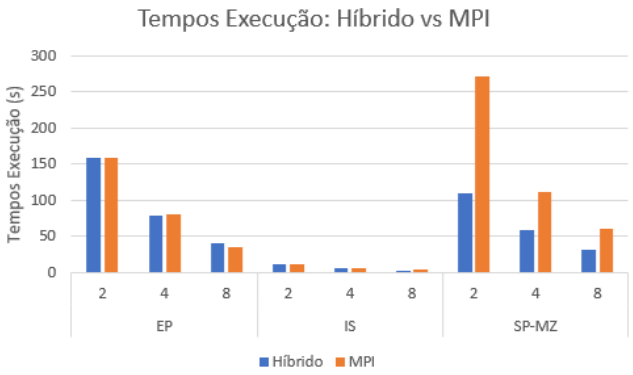


Fig. 27: Tempos de execução, utilizando um sistema Myrinet

nos tempos de execução. Como tal, utilizando um paradigma híbrido, tentando tirar partido das vantagens de um ambiente distribuído e dos cores disponíveis.

Assim sendo o histograma seguinte tem como objetivo comparar os tempos de execução entre um ambiente de memória partilhada e um paradigma híbrido. Sobre o qual é feita uma exposição utilizando os dois sistemas de redes, *Myrinet* e *Gigabit Ethernet*, e os obtidos utilizando *OpenMP*.

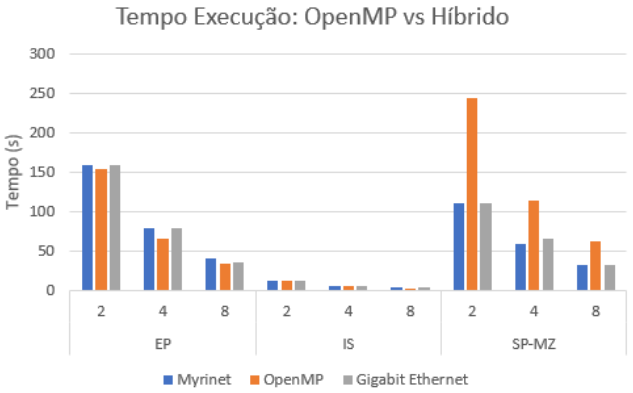


Fig. 28: Tempos de execução OpenMP vs Open MPI, com um compilador da **GNU**

Como podemos ver a partir da figura 27, a utilização de um paradigma híbrido só compensa para kernels mais complexos como é o caso do **SP-MZ**. No entanto as diferenças entre os tempos para outros tempos são mínimas, chegando a serem quase equivalentes, tal como acontece para aplicação **IS**.

C. Ganhos com O2 e O3

Tendo em conta os resultados, obtidos anteriormente, verificamos que as diferenças mais notórias em termos de tempo de execução, verifica-se no kernel **SP-MZ**. Assim sendo, a análise seguinte, terá como caso estudo esse mesmo kernel e os ganhos que apresenta com diferentes flags de otimização, ao longo do número de processos.

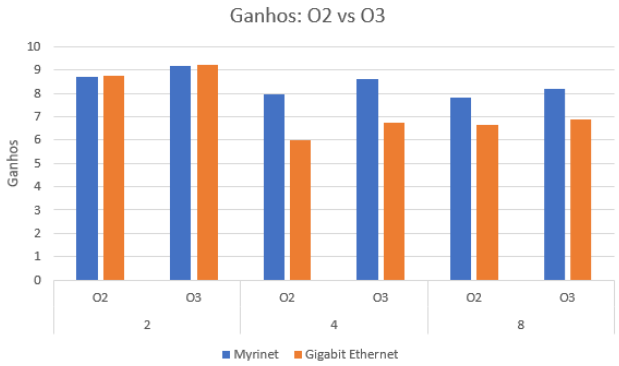


Fig. 29: Ganhos com a utilização das flags **O2** e **O3**, no nodo 662

Como podemos verificar os ganhos são muito similares para 2 processos, porém com o aumentar dos processos, um sistema que utilize *Myrinet*, é o que apresenta os ganhos mais altos e consistentes. De seguida e tendo em conta o paradigma que se esta a estudar, iremos analisar de que forma a rede do nodo foi utilizada.

D. Utilização da Rede

Com uma mudança de paradigma e tendo em conta o peso que os tempos comunicação têm sobre os tempos de execução. É relevante entender como a utilização simultânea de *OpenMP* e *MPI*, poderá afetar a utilização da rede, comparativamente com o registado para uma implementação *MPI*.

Assim os gráficos seguintes, referem-se ao kernel **SP-MZ**, com otimizações **O2**, 8 processos e 4 threads, no nodo 662.

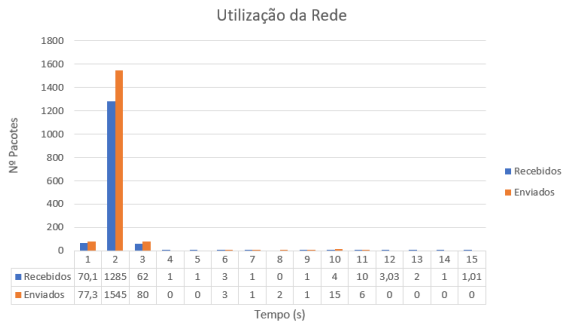


Fig. 30: Utilização de uma rede *Myrinet*, no nodo 662

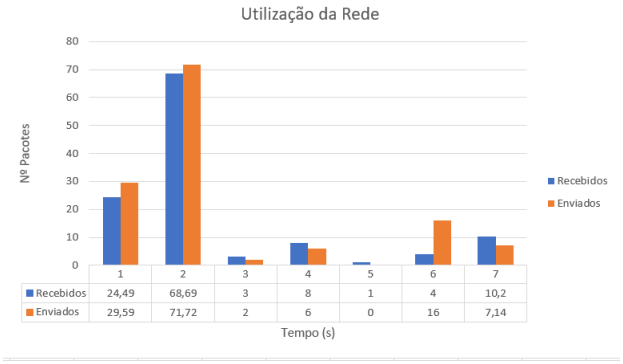


Fig. 31: Utilização de uma rede *Gigabit Ethernet*, no nodo 662

Comparativamente com a utilização da rede, em implementações de memória distribuída, podemos verificar um aumento na quantidade de pacotes enviados/-recebidos, por instante de tempo.

E. % CPU Consumido

Com o objetivo de analisar a performance obtida pelas versões híbridas, será analisada no gráfico seguinte a utilização do CPU de cada uma das versões. Sabendo que este tipo de paradigma utiliza implementações com **OpenMP** e **MPI**, será expectável um aumento dos valores atingidos, relativamente com por exemplo os atingidos no ambiente memória partilhada. O caso em análise refere-se a versão híbrida do kernel **SP-MZ**, na classe **C**, e utilizando 8 processos, para além das 4 threads.

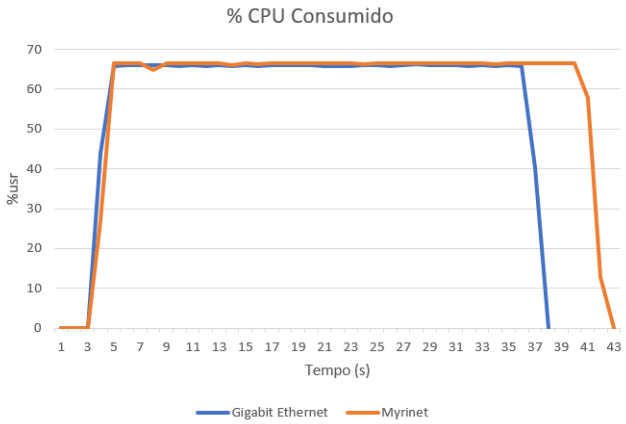


Fig. 32: % CPU Consumido pelos kernel **SP-MZ** na máquina 662

A partir do gráfico anteriormente exposto, notamos um aumento da utilização do CPU por este paradigma, cerca de 3.5 vezes mais, relativamente ao registado numa implementação *OpenMP*.

F. Consumo Memória

Dado o aumento visualizado, na quantidade de CPU Consumido, será expectável que os consumos de memória também aumentem.

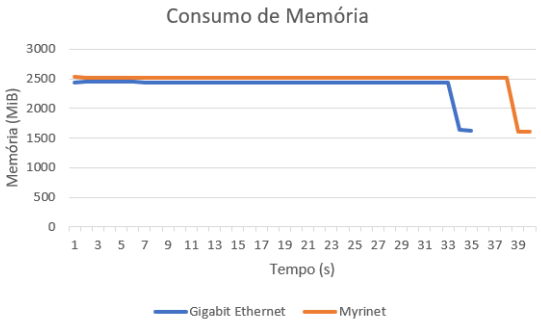


Fig. 33: Consumo Memória por um paradigma híbrido, com utilização diferentes sistemas redes, no nodo 662

A partir da figura 33, verificamos tal como previsto um aumento na quantidade de memória consumida por parte do sistema *Gigabit Ethernet*, um aumento ligeiro, face ao registado numa implementação de memória partilhada. Este consumo vai-se mantendo constante ao longo da execução, o mesmo se verificando para um sistema de *Myrinet*.

IX. Conclusão

De modo a atingir a melhor performance de uma aplicação, é preciso ter em conta o panorama geral. Este panorama, visam característica do sistema de computação utilizado, neste caso o cluster SeARCH, onde se tiveram em conta aspetos como os compilador, diferentes níveis de otimização e topologias de conexão da rede. Assim sendo, é exigida a geração de diversos gráficos, de forma a possibilitar a resposta de questões como os consumos de CPU ou memória, tempos de execução com alteração de compilador e topologias de rede.

Por outro lado, e de modo a facilitar o processo de submissão e captura dos resultados, este trabalho exigiu a utilização *scripts*, e adquirição de novos conhecimentos relativamente ao cluster utilizado.

References

[1] Search6.di.uminho.pt. 2020. Search — Services And Advanced Research Computing With HTC/HPC Clusters. [online] Available at: ¡http://search6.di.uminho.pt/wordpress/?page_id=55¿ [Accessed 7 April 2020].

[2] Ark.intel.com. 2020. Intel® Xeon® Processor E5-2695 V2 (30M Cache, 2.40 Ghz) Product Specifications. [online] Available at: ¡https://ark.intel.com/content/www/us/en/ark/products/75281/intel-xeon-processor-e5-2695-v2-30m-cache-2-40-ghz.html¿ [Accessed 7 April 2020].

[3] Ark.intel.com. 2020. Intel® Xeon® Processor E5-2670 V2 (25M Cache, 2.50 Ghz) Product Specifications. [online] Available at: ¡https://ark.intel.com/content/www/us/en/ark/products/75275/intel-xeon-processor-e5-2670-v2-25m-cache-2-50-ghz.html¿ [Accessed 9 April 2020].

[4] Cpu-world.com. 2020. Intel Xeon E5-2670 V2 - CM8063501375000 / BX80635E52670V2. [online] Available at: ¡http://www.cpu-world.com/CPUs/Xeon/Intel-Xeon%20E5-2670%20v2.html¿ [Accessed 9 April 2020].

[5] Cpu-world.com. 2020. Intel Xeon E5-2695 V2 - CM8063501288706 / BX80635E52695V2. [online] Available at: ¡http://www.cpu-world.com/CPUs/Xeon/Intel-Xeon%20E5-2695%20v2.html¿ [Accessed 9 April 2020].

[6] Hardman, J., 2020. NAS Parallel Benchmarks. [online] Nas.nasa.gov. Available at: ¡https://www.nas.nasa.gov/publications/npb.html#url¿ [Accessed 7 April 2020].

[7] H. Bailey, D., Barszcz, E., J.T, B., D.S, B., R.L, C., D, D., R.A, F., Frederickson, P., T.A, L., Schreiber, R., D. Simon, H., Venkatakrishnan, V. and K, W., 2020. The Nas Parallel Benchmarks. [online] ResearchGate. Available at: ¡https://www.researchgate.net/publication/236135736_The_Nas_Parallel_Benchmarks¿ [Accessed 7 April 2020].

[8] Van der Wijngaart, R. and Jin, H., 2020. NAS Parallel Benchmarks, Multi-Zone Versions. [online] Nas.nasa.gov. Available at: ¡https://www.nas.nasa.gov/assets/pdf/techreports/2003/nas-03-010.pdf¿ [Accessed 7 April 2020].

[9] Kili, A. and Posts, V., 2020. Dstat - A Resourceful Tool To Monitor Linux Server Performance In Real-Time. [online] Tecmint.com. Available at: ¡https://www.tecmint.com/dstat-monitor-linux-server-performance-process-memory-network/¿ [Accessed 9 April 2020].

[10] Schwarz, S., 2020. Pros And Cons Of Openmp And MPI. [online] Dartmouth.edu. Available at: ¡https://www.dartmouth.edu/ rc/classes/intro_mpi/parallel_prog_compare.html¿ [Accessed 14 April 2020].

[11] Search6.di.uminho.pt. 2020. Cluster Architecture — Search. [online] Available at: ¡http://search6.di.uminho.pt/wordpress/?page_id=274¿ [Accessed 14 April 2020].

[12] Software.intel.com. 2020. Intel® C++ Compiler. [online] Available at: ¡https://software.intel.com/en-us/c-compilers¿ [Accessed 16 April 2020].

[13] Software.intel.com. 2020. Openmp* Support. [online] Available at: ¡https://software.intel.com/en-us/cpp-compiler-developer-guide-and-reference-openmp-support¿ [Accessed 16 April 2020].

[14] Techterms.com. 2020. Bandwidth Definition. [online] Available at: ¡https://techterms.com/definition/bandwidth¿ [Accessed 21 April 2020].