Introdução ao NAS Parallel Benchmarks
Performance Relativa de Kernels Sequenciais, em ambiente de Memória Partilhada e ambiente de Memória Distribuída

Filipe Oliveira
Departamento de Informática
Universidade do Minho
Email: a57816@alunos.uminho.pt

1. Introdução – Contextualização das Benchmarks

As "NAS Parallel Benchmarks" [1] englobam 5 kernels (EP, MG, CG, FT, IS) e 3 aplicações que simulam dinâmica de fluídos (LU,SP,BT). No nosso caso de estudo, temos por interesse os 5 kernels, dado que centraremos o nosso estudo da performance relativa via alterações no paradigma de memória e forma de comunicação entre nós, assim como a própria ferramenta de compilação e respectivas flags. Assim sendo, temos então 5 opções a analisar: EP, MG, CG, FT, IS. Resta-nos portanto analisar primeiramente quais as principais propriedades de cada um antes de qualquer avanço no trabalho.

- **EP**, tal como o próprio nome indica (Embarrassingly Parallel), que por calcular números aleatórios é um kernel implicitamente embaracosamente paralelo. Tem como propósito estabelecer o Peak Performance em "FP Operations" de um sistema de computação em teste. É então espectável obtermos os melhores resultados de performance neste kernel e, por esse mesmo motivo, este será um dos kernels com grande relevância para o nosso caso de estudo.
- **MG**, cujo kernel implementa um método numérico multigrid simplificado, numa sequência de malhas de diferentes propriedades, implicando portanto uma elevada comunicação para a resolução do algoritmo. Tanto para as versões em memória distribuída com para a versão de memória partilhada será interessante analisar o comportamento do kernel nos ambientes de teste. Será portanto também incluído no caso de estudo.
- **CG**,que recorre ao método do Conjugado do Gradiente por forma a calcular uma aproximação ao menor dos valores próprios de uma matriz esparsa de grandes dimensões. Dada o tipo de dados, este kernel testa computação e comunicação não estruturada, sendo portanto expectável uma fraca performance deste kernel quando em comparação com o **EP**. Será portanto também incluído no caso de estudo.
- **FT**, que calcula a Transformada de Fourier em 3 dimensões (3 transformadas de Fourier de uma dimensão), sendo o resultado a solução de uma equação diferencial parcial. Dado que a principal propriedade a ser estudada é comunicação, este kernel será portanto excluído do caso de estudo em detrimento do **MG**.
- **IS**, que realiza operações de sorting em inteiros. Este kernel testa tanto a capacidade de computação de um sistema em termos de operações sobre inteiros, assim como a performance de comunicação do mesmo dada a irregularidade dos acessos à memória e, quando aplicável, comunicação entre processos. Será também incluído no caso de estudo.

2. Caracterização do Hardware do ambiente de testes

Escolhidas as benchmarks, resta-me especificar os ambientes de teste nos quais pretendo realizar as benchmarks. Através da análise do hardware disponível no Search6 [1], a nossa plataforma de teste, deparamo-nos com duas realidades distintas presentes no mesmo cluster. Se por um lado temos uma grande porção dos nós de computação com configurações de hardware relativamente homogéneas (28 dos 54 nós disponíveis apresentam todos as mesmas características de comunicação, armazenamento local, memória RAM disponível e mesma família de processadores - Ivy Bridge), por outro lado temos os restantes 26 nós com características relativamente distintas entre nós (diferentes famílias de processadores, diferentes características de comunicação entre nós disponíveis, memória RAM disponível em diferente número).

Considerando a abrangência do número de nós o ponto essencial para a escolha do tipo de nós a teste, decidi incluir no mesmo os nós do tipo 662, 652, 641, e 431(apenas os nós com 48GB de memória RAM disponíveis), que passarei de seguida a caracterizar.

Denote que ao realizarmos os testes de performance nos nós acima enumerados estamos a abranger 33 dos

1. Services and Advanced Research Computing with HTC/HPC clusters

54 nós disponíveis, preservando características entre eles fundamentais para a possibilidade de comparação como por exemplo o suporte da rede Myrinet 10Gbps, e englobando como requerido mais do que uma classe de arquitectura existente no Search6.

TABLE 1. CARACTERÍSTICAS DE HARDWARE DO NÓ 662

Sistema	compute-662
# CPUs	2
CPU	Intel® Xeon® E5-2695 v2
Arquitectura de Processador	Ivy Bridge
# Cores por CPU	12
# Threads por CPU	24
Freq. Clock	2.4 GHz
Cache L1	384KB (32KB por Core)
Cache L2	3072KB (256KB por Core)
Cache L3	30720KB (partilhada)
Ext. Inst. Set	SSE4.2, AVX
#Memory Channels	4
Memória Ram Disponível	64GB
Peak Memory BW Fab. CPU	59.7 GB/s
Rede Suportada	Gigabit Ethernet, Myrinet 10Gbps

TABLE 2. CARACTERÍSTICAS DE HARDWARE DO NÓ 652

Sistema	compute-652
# CPUs	2
CPU	Intel® Xeon® E5-2670 v2
Arquitectura de Processador	Ivy Bridge
# Cores por CPU	10
# Threads por CPU	20
Freq. Clock	2.5 GHz
Cache L1	320KB (32KB por Core)
Cache L2	2560KB (256KB por Core)
Cache L3	25600KB (partilhada)
Ext. Inst. Set	SSE4.2, AVX
#Memory Channels	4
Memória Ram Disponível	64GB
Peak Memory BW Fab. CPU	59.7 GB/s
Rede Suportada	Gigabit Ethernet, Myrinet 10Gbps

TABLE 3. CARACTERÍSTICAS DE HARDWARE DO NÓ 641

Sistema	compute-641
# CPUs	2
CPU	Intel® Xeon® E5-2650 v2
Arquitectura de Processador	Ivy Bridge
# Cores por CPU	8
# Threads por CPU	16
Freq. Clock	2.6 GHz
Cache L1	256KB (32KB por Core)
Cache L2	2048KB (256KB por Core)
Cache L3	20480KB (partilhada)
Ext. Inst. Set	SSE4.2, AVX
#Memory Channels	4
Memória Ram Disponível	64GB
Peak Memory BW Fab. CPU	59.7 GB/s
Rede Suportada	Gigabit Ethernet, Myrinet 10Gbps

Da caracterização de hardware acima realizada podemos retirar já as diferentes possibilidades relativamente ao número de threads a testar em ambiente de memória partilhada, assim como propriedades adequadas aos testes em ambiente de memória distribuída. Para além do anteriormente enumerado obtive dados de extrema importância relativa à próxima decisão do nosso caso de estudo – a escolha das classes de dados a incluir no nosso caso de estudo. Será com base nas propriedades relativas à Memória RAM disponível, tamanho e forma de distribuição dos vários níveis de cache, assim como a Peak Memory Bandwidth teórica de Memória do CPU, que centrarei a minha decisão sobre quais as classes de dados relevantes.

TABLE 4. CARACTERÍSTICAS DE HARDWARE DO NÓ 431

Sistema	compute-431
# CPUs	2
CPU	Intel® Xeon® X5650
Arquitectura de Processador	Nehalem
# Cores por CPU	6
# Threads por CPU	12
Freq. Clock	2.66 GHz
Cache L1	192KB (32KB por Core)
Cache L2	1536KB (256KB por Core)
Cache L3	12288KB (partilhada)
Ext. Inst. Set	SSE4.2
#Memory Channels	3
Memória Ram Disponível	48GB
Peak Memory BW Fab. CPU	32 GB/s
Rede Suportada	Gigabit Ethernet, Myrinet 10Gbps

2.1. Inclusão de diferentes dimensões (classes) de dados

Será importante para a relevância dos testes de perfomance incluir datasets de diferentes dimensões. Ora, analisando as propriedades dos processadores presentes nos nossos nós de computação devo portante seleccionar um dataset capaz de ser compreendido na Cache L1 (menor ou igual a 32KB), um dataset capaz de ser compreendido na Cache L2 (menor ou igual a 256KB), um dataset capaz de ser compreendido na Cache L3 (menor que 12MB) e um dataset capaz de ser compreendido na Memória Ram disponível nos nós de computação (menor que 48GB).

Analisadas as dimensões dos problemas para as diferentes Classes de dados e Benchmarks a incluir no nosso caso de estudo através da tabela 5, verificamos que em nenhum dos casos os datasets são possíveis de ser compreendidos totalmente na cache L1. Assim, e focando o nosso processo de seleção com base nas benchmarks IS e MG podemos concluir que as classes de dados que melhor se associam ao nosso problema são portanto a classe S (totalmente contida na cache L2 ou L3), e as classes A, B, e C por serem consideradas classes standard das benchmarks e de fácil comparação de resultados com a comunidade do HPC.

Denote ainda que para o caso da benchmark CG a classe de dados C apresenta um tamanho de dados superior ao possível de estar contido na memória principal obrigando teoricamente a um decréscimo de perfomance derivada do aumento de IO.

TABLE 5. DIMENSÃO DO DATASET PARA AS DIFERENTES CLASSES E BENCHMARKS

Bench.	data type	S	A	B	C
EP	double	128 MB	2 GB	8 GB	32 GB
MG	double	256 KB	128 MB	128 MB	1024 MB
CG	double	15MB	1,46 GB	41,91 GB	167,64 GB
IS	integer	256 KB	32 MB	128 MB	512 MB

3. Caracterização do Software do ambiente de testes

O cluster Search6 é baseado no Rocks 6.1 cluster management distribution. Dado que um dos principais propósitos deste caso de estudo acenta na investigação da influência de diferentes ferramentas de compilação e diferentes configurações de ferramentas de comunicação, assim como dos diferentes paradigmas de memória, na perfomance global dos kernels, o próximo passo natural passa por identificar os módulos disponíveis no nosso ambiente de clustering que cumprem o objectivo anteriormente enumerado.

Relativamente aos compiladores de interesse para o caso de estudo temos portanto o **GCC compiler suite** e o **Intel compilers suite**. O último apresenta apenas a possibilidade de seleção da versão "icc version 13.0.1 (gcc version 4.4.6 compatibility)". Ora, pela própria informação presente na versão do compilador da Intel deveremos incluir no nosso leque de testes a versão do compilador da gnu 4.4.6. Irei também incluir a versão 4.9.0 do compilador da Gnu por ser considera a versão default deste mesmo compilador no nosso ambiente de clustering.

Relativamente às MPI stacks, nomeadamente à versão do OpenMPI a ser incluída, será em ambos os casos (Intel e GNU) a versão mais recente disponível para ambos, nomeadamente a versão 1.8.2 dos módulos com via de comunicação Gigabit Ethernet e Myrinet 10Gbps.

Relativamente às flags de compilação serão testas as respectivas optimizações disponibilizadas pelas flags de compilação -O2 e -O3, e sem qualquer tipo de otimização por flags de compilação.

Teremos portanto as seguintes combinações de software distintas no nosso caso de estudo:

- Compiladores distintos:
  - Kernels compilados com compilar da GNU gcc versão 4.4.6
  - Kernels compilados com compilar da GNU gcc versão 4.9.0
  - Kernels compilados com compilar da Intel icc versão 13.0.1
- Versão do OpenMPI:
  - versão 1.8.2 via comunicação Gigabit Ethernet para compilador da GNU
  - versão 1.8.2 via comunicação Gigabit Ethernet para compilador da Intel
  - versão 1.8.2 via comunicação Myrinet 10Gbps para compilador da GNU
  - versão 1.8.2 via comunicação Myrinet 10Gbps para compilador da Intel

4. Benchmarking em ambiente de testes sequencial – NPB SEQ

Nesta primeira fase do caso de estudo temos principal interesse na perfomance dos 4 kernels em teste relativamente ao tempo total para a solução, assim como o número máximo de Milhões de FP Operations alcançado por cada kernel.

Assim, foquemos a nossa atenção primeiramente no número máximo de FP Operations alcançado para a maior classe de dados em teste (classe C), para as várias opções de compiladores e flags de compilação:

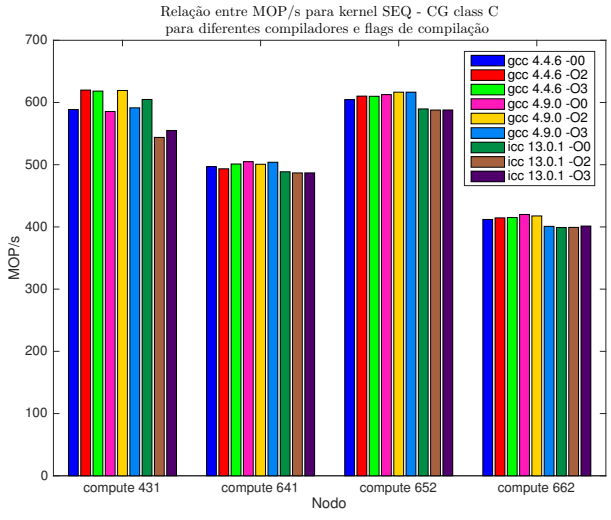

Figure 1. Milhões de FP Operations alcançado para o kernel SEQ - CG, classe de dados C para diferentes compiladores e flags de compilação

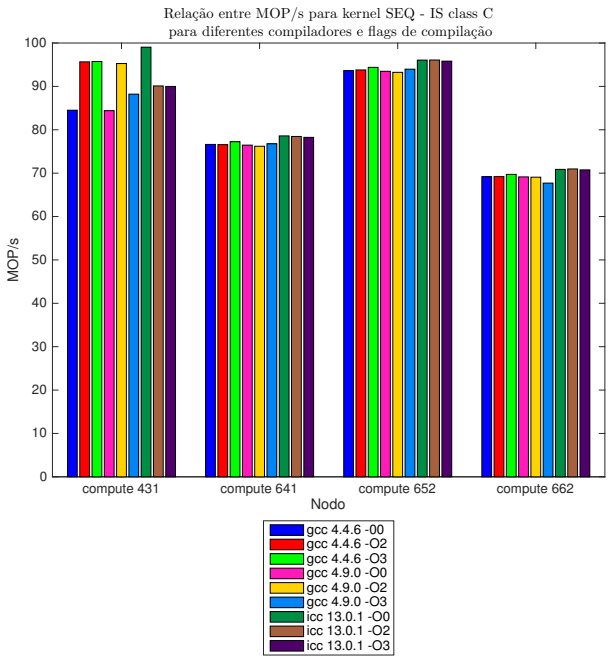

Figure 2. Milhões de FP Operations alcançado para o kernel SEQ - IS, classe de dados C para diferentes compiladores e flags de compilação

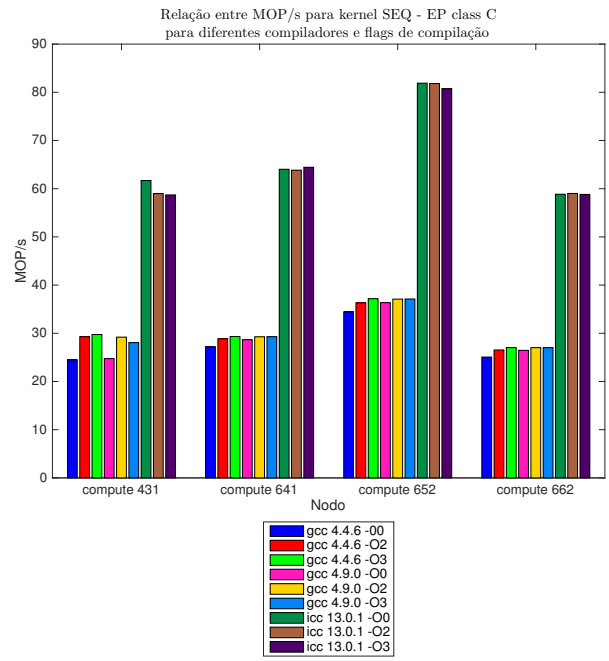

Figure 3. Milhões de FP Operations alcançado para o kernel SEQ - EP, classe de dados C para diferentes compiladores e flags de compilação

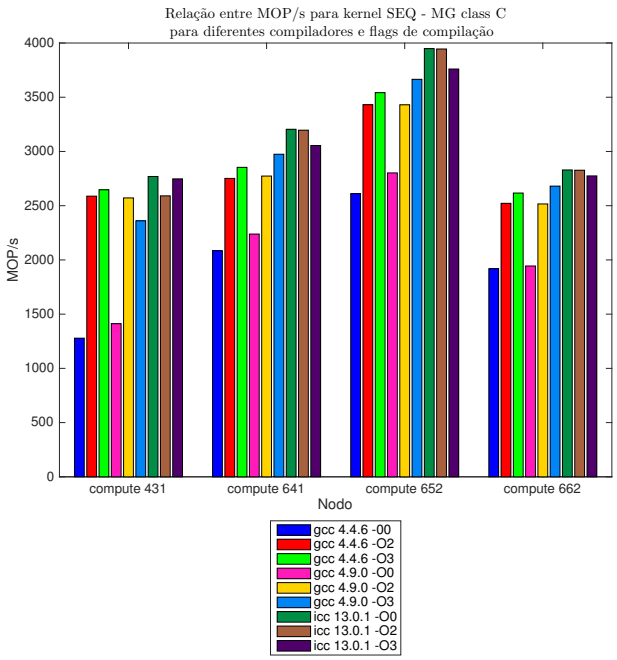

Figure 4. Milhões de FP Operations alcançado para o kernel SEQ - MG, classe de dados C para diferentes compiladores e flags de compilação

Como poderá analisar, é para os nós do tipo compute-662 que obtemos os valores mais baixos de FP Operations por segundo. Denote que estamos a centrar este primeiro caso de estudo tendo por base a versão sequencial, ou seja, o número de cores físicos disponíveis por tipo de nó não terá influência, tendo sim por exemplo, a frequência de clock base de cada CPU. Ora, tendo isso em conta, este resultado é espectável dada a frequência dos nós 662 ser inferior a qualquer um dos outros nós em teste. Podemos ainda comprovar esta afirmação pela performance dos nós compute-431 quando comparada com nós de computação que possuem CPUs de uma arquitectura mais recente.

Podemos ainda observar que relativamente à performance relativa para diferentes compiladores e flags de compilação é nos kernels compilados com o compilador gcc versão 4.9.0 com flag de compilação -03 que obtemos os melhores resultados. O compilador da Intel apenas ultrapassa o compilador da GNU em todos os nós de computação para o kernel EP, como visível na figura 3. Podemos daqui concluir que para os 4 kernels em estudo o compilador icc tem melhor performance em kernels sem dependência de dados. Contudo, o valor máximo de FP Operations não foi de todo alcançado pelo kernel EP mas sim pelo kernel MG como visível na figura 4.

Ora, analisado o máximo de FP Operations alcançado pelos kernels sequenciais, centremos a nossa atenção no tempo total para a solução destes mesmos kernels para todas as classes de dados em estudo, restringindo agora o análise gráfica para o melhor resultado obtido em termos de opção de compilador e flag de compilação – gcc 4.9.0 com flag

-O3.

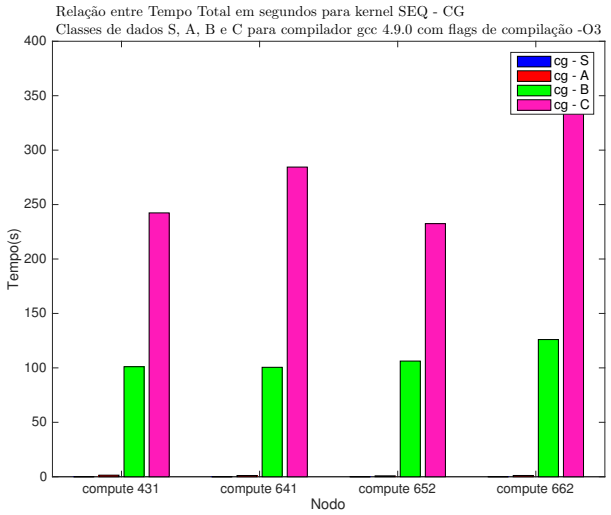

Figure 5. Relação entre tempo total para a solução para o kernel SEQ - CG, classes de dados S,A,B e C para compilador gcc 4.9.0 com flag de compilação -O3

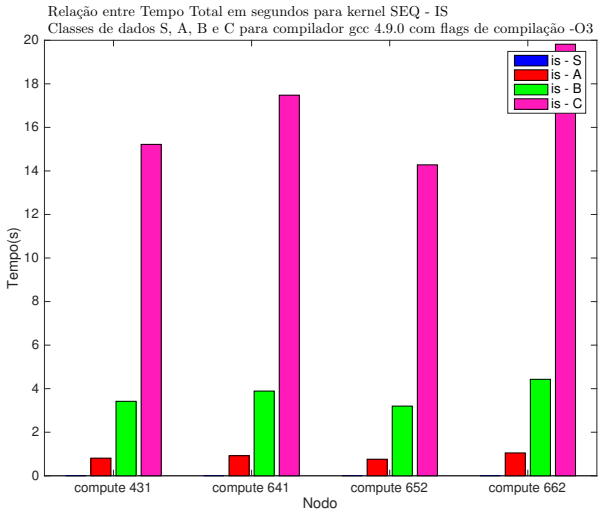

Figure 6. Relação entre tempo total para a solução para o kernel SEQ - IS, classes de dados S,A,B e C para compilador gcc 4.9.0 com flag de compilação -O3

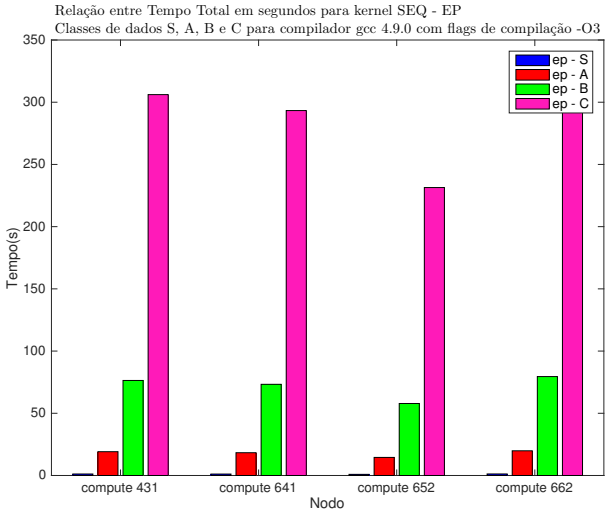

Figure 7. Relação entre tempo total para a solução para o kernel SEQ - EP, classes de dados S,A,B e C para compilador gcc 4.9.0 com flag de compilação -O3

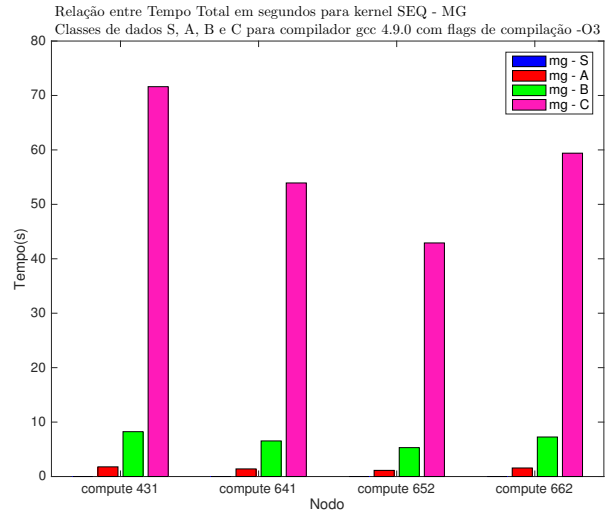

Figure 8. Relação entre tempo total para a solução para o kernel SEQ - MG, classes de dados S,A,B e C para compilador gcc 4.9.0 com flag de compilação -O3

Analisando o tempo de computação por tipo de nó voltamos a confirmar o enunciado anteriormente – os melhores resultados estão presentes para os nós do tipo compute-651 seguidos dos apresentados pelos nós compute-431. Temos portante duas arquitecturas de processadores distintas com resultados praticamente equivalentes. Comprovamos ainda que a proporção de crescimento do tempo para a solução entre classes de dados é conforme o espetado aproximadamente 4x.

Resta-nos relacionar o tempo entre os 4 kernels para os diferentes nós de computação para a maior classe de dados em estudo:

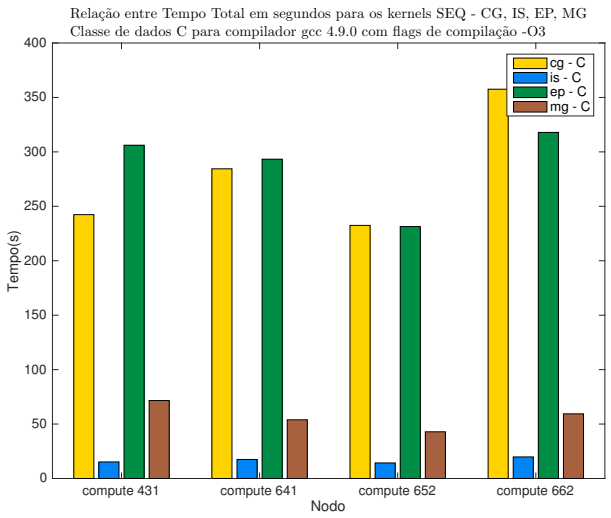

Figure 9. Relação entre tempo total para a solução para os kernels SEQ - CG, IS, EP e MG, para a classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3

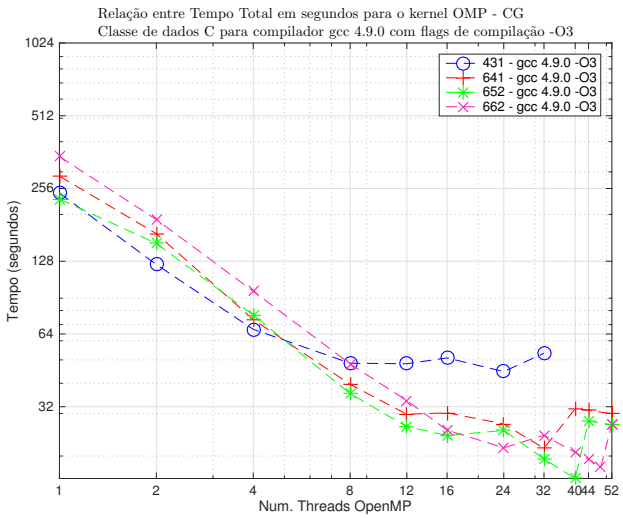

Figure 10. Relação entre tempo total para a solução para o kernel OMP - CG, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3

Podemos concluir deste último gráfico a razão do kernel EP ser tão penalizado em termos de FP Operations por segundo. Ora, como pode analisar o tempo para a solução do mesmo, apesar de ser inferior a todos os outros, não apresenta uma diferença suficiente para o que o número de operações justifique o tempo de computação. Temos no outro extremo dois kernels de complexidade superior (CG e MG) agravados pela dependência de dados mas que conseguem manter um rácio de número de operações por segundo em muito superior ao kernel que deveria definir um tecto quanto ao número de FP operations. Veremos se esta mesma razão se mantém quando adicionarmos o overhead de comunicação entre processos para os casos de ambiente de memória distribuída e memória partilhada.

5. Benchmarking em ambiente de memória partilhada – NPB OMP

Para o caso de estudo em ambiente de memória partilhada foram criados casos de teste que superassem o número de threads possíveis de correrem concorrente por tipo de CPU, ou seja, superou-se o número de threads por CPU mesmo em hyper-threading. Analisemos portanto a relação entre o tempo de computação e o aumento do número de threads openMP para a solução, para os diferentes kernels e para a maior classe de dados em teste, fixando o compilador que apresentou melhores resultados anteriormente (gcc 4.9.0. com flag de compilação -O3):

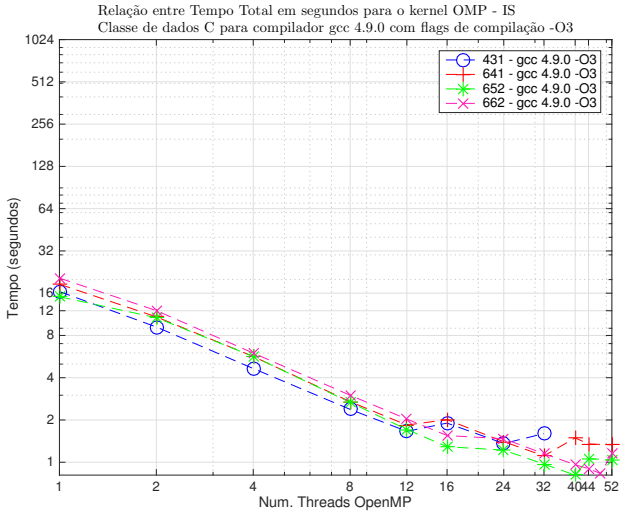

Figure 11. Relação entre tempo total para a solução para o kernel OMP - IS, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3

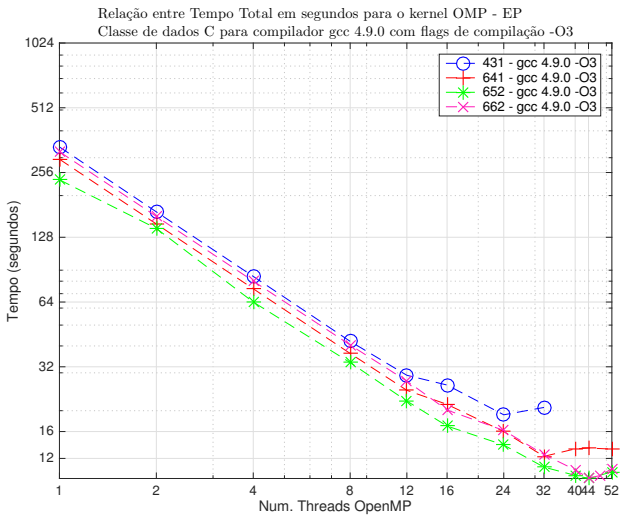

Figure 12. Relação entre tempo total para a solução para o kernel OMP - EP, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3

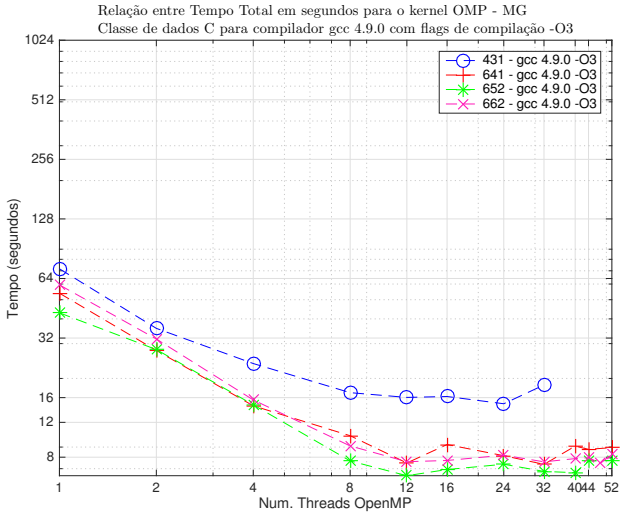

Figure 13. Relação entre tempo total para a solução para o kernel OMP - MG, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3

Ora, se anteriormente o tempo total de solução apresentado para os nós do tipo compute-662 era ultrapassado por todos os restantes tipos de nós, neste ambiente de teste tal não sucede. Isto deve-se ao número de threads disponível por CPU deste tipo de nó ser superior ao valor apresentado pelos seus ”concorrentes”, assim como o tamanho dos níveis de cache pertencer ao grupo dos que apresentam maior dimensão.

Denote que o tempo de computação apresenta uma redução para todo os incrementes no número de threads presente na solução até ao número de threads openMP superar o número de threads disponíveis por CPU. É espectável que tal suceda. Quando existem mais threads na solução que as ”fisicamente” (em hyper-threading) disponibilizadas algumas terão que ser colocadas em espera e os dados presentes em cache serão possivelmente invalidados pelos dados da thread que ganhou o tempo de processador em detrimento desta. Denote que, para todos os nós em estudo, apenas as Cache L1 e L2 são explicitamente de um Core, o nível L3 é partilhado por grupos de Cores, exponenciando o problema anteriormente descrito.

Analisados os tempos totais de solução, centremos a nossa atenção nos ganhos alcançados para as versões paralelas dos kernels quando comparados com o valores de referência sequencial, para a classe de dados C, para o compilador com melhores resultados sequenciais (gcc 4.9.0 com flag de compilação -O3).

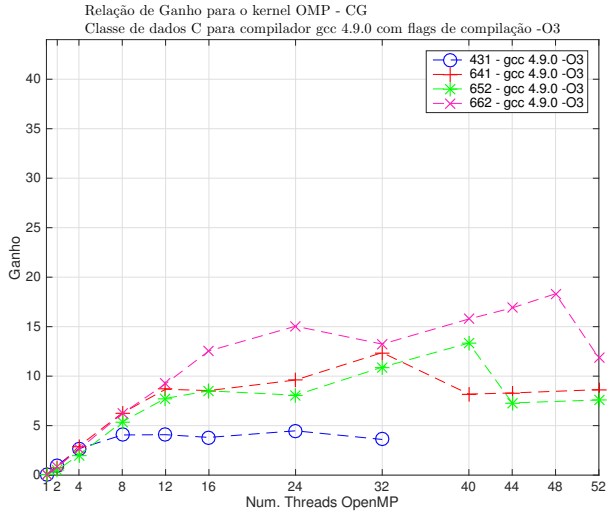

Figure 14. Relação de ganho para o kernel OMP - CG, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3

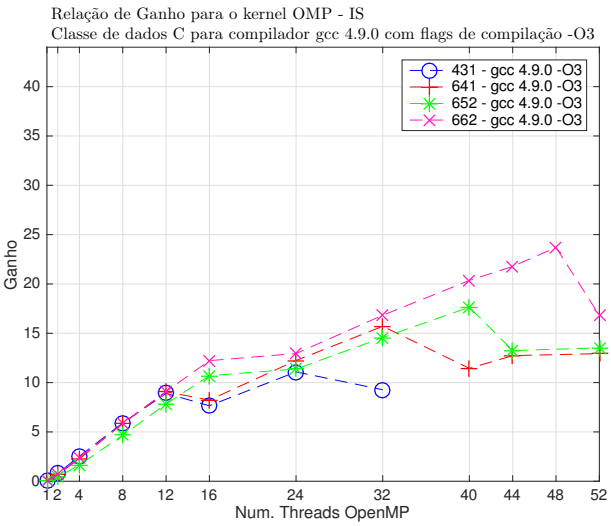

Figure 15. Relação de ganho para o kernel OMP - IS, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3

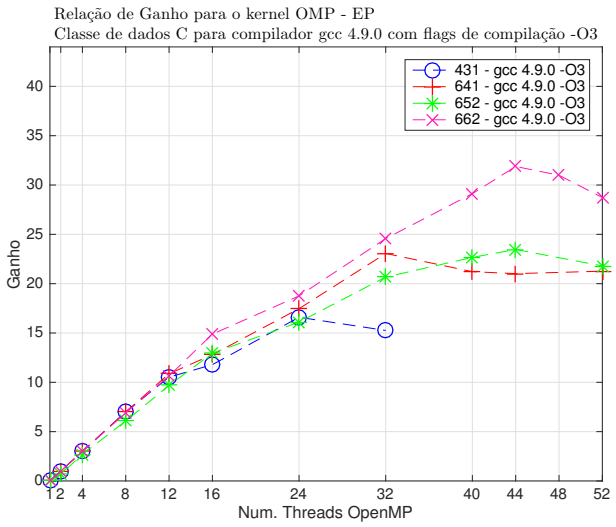

Figure 16. Relação de ganho para o kernel OMP - EP, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3

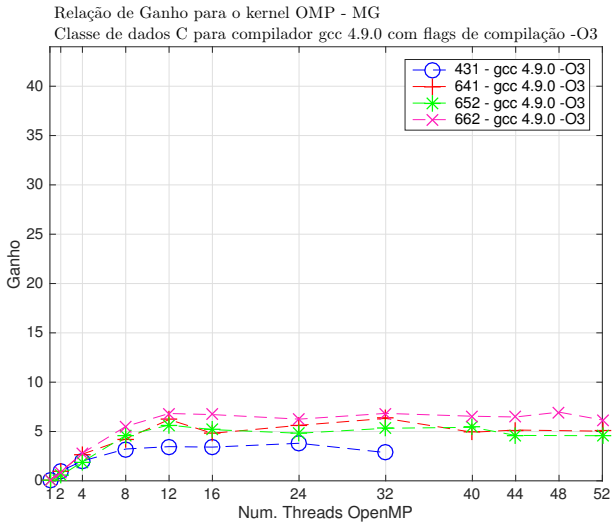

Figure 17. Relação de ganho para o kernel OMP - MG, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3

Como podemos constatar pelas figuras 14 a 17 foi para os nós compute-662 que obtivemos a maior relação de ganho. Ora isto pode ser explicado pelo anteriormente descrito e pelo facto de este mesmo nó apresentar para a versão sequencial um tempo aquém do esperado. Ora, assim em adição aos bons resultados para este tipo de nós em ambiente paralelo de memória partilhado estão os mais resultados para os testes sequenciais, elevando assim ainda mais o valor do ganho.

Por questões pedagógicas e de validação analisemos o comportamento de outra versão do compilador (gcc 4.4.6) e outra flag de compilação (-O2), restringindo-nos nesta análise aos nós com melhor resultado em kernels paralelos em memória partilhada – os nós compute-662.

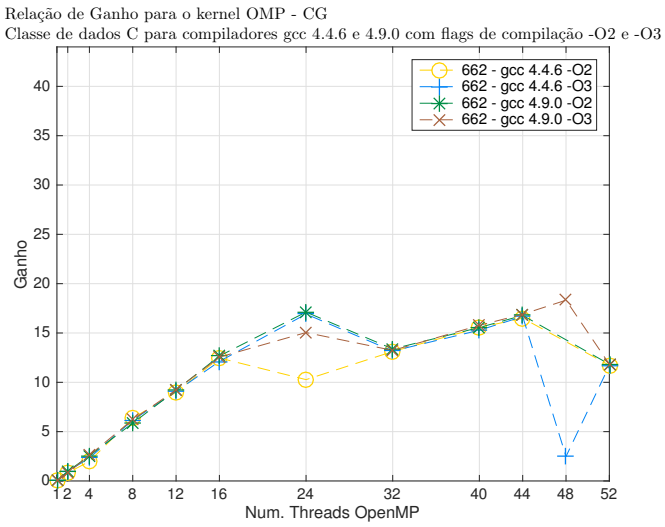

Figure 18. Relação de ganho para o kernel OMP - CG, classe de dados C para compiladores gcc 4.4.6 e 4.9.0 com flags de compilação -O2 ou -O3

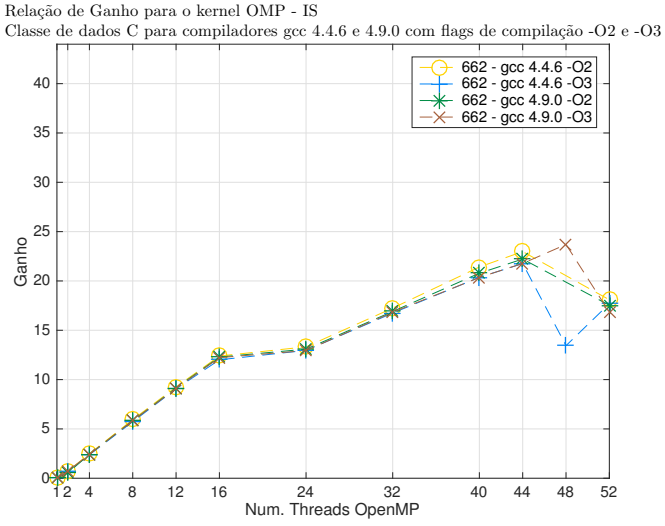

Figure 19. Relação de ganho para o kernel OMP - IS, classe de dados C para compiladores gcc 4.4.6 e 4.9.0 com flags de compilação -O2 ou -O3

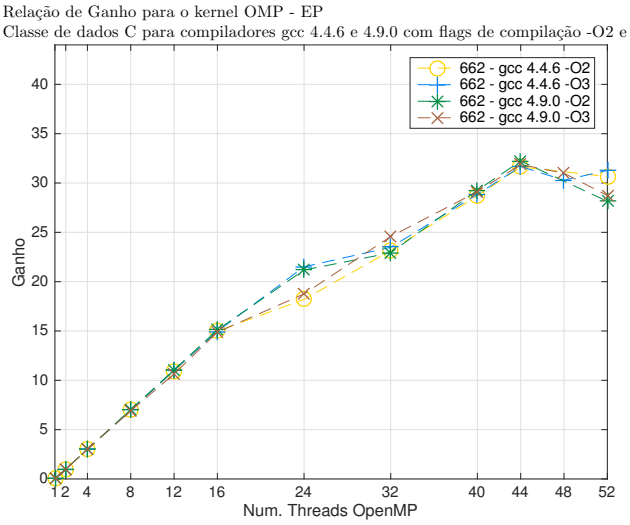

Figure 20. Relação de ganho para o kernel OMP - IS, classe de dados C para compiladores gcc 4.4.6 e 4.9.0 com flags de compilação -O2 ou -O3

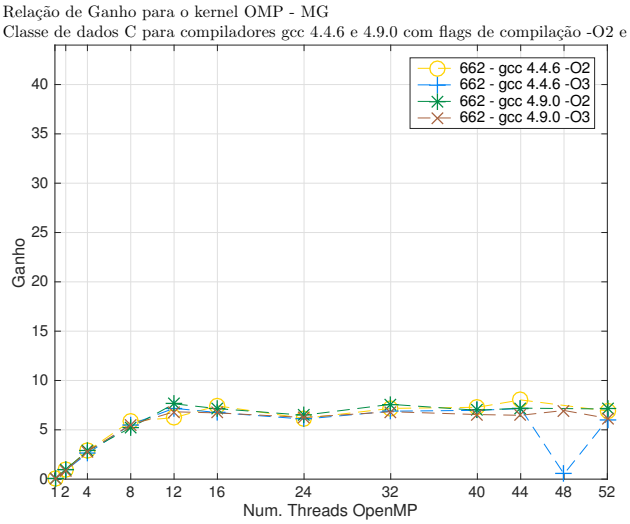

Figure 21. Relação de ganho para o kernel OMP - IS, classe de dados C para compiladores gcc 4.4.6 e 4.9.0 com flags de compilação -O2 ou -O3

Podemos concluir que o ganho relativo se manteve constante com exceção do número de threads igual a 48 para a versão do compilador gcc 4.4.6 com flag de compilação -O3. Ora, tal decréscimo de performance repercute-se em todos os kernels. Podemos afirmar com certeza que o mesmo decréscimo foi ultrapassada com a última versão do compilador tornando pouco expressivo o valor do mesmo no caso de estudo.

6. Benchmarking em ambiente de memória distribuída – NPB MPI

Para o caso de estudo em ambiente de memória distribuída foram criados casos de teste que envolveram 2 e 4 máquinas distintas do mesmo tipo. Assim para o caso de 2 máquinas envolvidas na solução teremos em teste 8 e 16 processos MPI e para o caso de 4 máquinas envolvidas na solução teremos em teste 32 processos MPI. Depois de uma sessão informativa com o Search Admin foi aconselhado que o número de processos comunicantes via Myrinet 10Gbps não ultrapassasse os 8, daí os testes anteriormente descritos. Para além das diferentes configurações relativamente ao número de processos e nós teremos ainda diferentes formas de comunicação, sendo estas Gigabit Ethernet e Myrinet 10Gbps. Ora dado necessitarmos de 4 do mesmo tipo excluímos obrigatoriamente os nós do tipo compute-652 do ambiente de teste (existem apenas 2 nós disponíveis).

Dado que o cluster utilizado para ambiente de testes é também um cluster de produção, os recursos disponíveis são também limitados, sendo que a probabilidade de alocar exclusivamente 4 dos 6 nós do tipo compute-662 era muito reduzida foi limitado o ambiente de teste em memória distribuída aos nós do tipo compute-641 e compute-431. Denote que continuamos a poder comparar duas arquitecturas de processador distintas como era nosso objectivo.

Serão então apresentados os resultados divididos por tipo de nó.

6.1. Relação entre tempo total para a solução para os kernels OMP CG, IS, EP, MG – nós compute-431

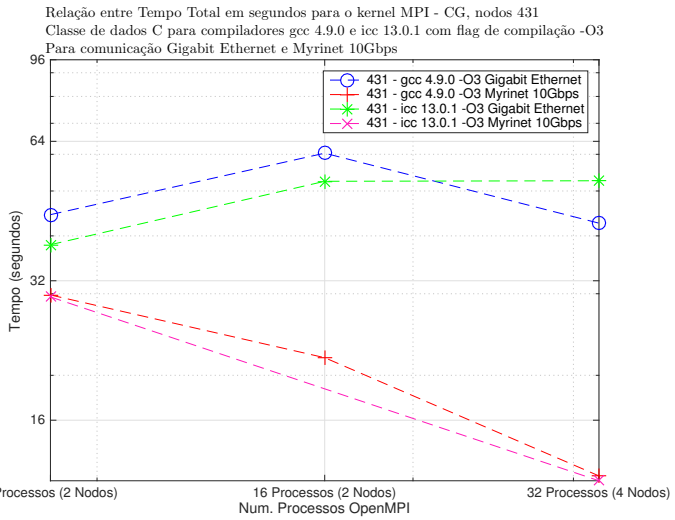

Figure 22. Relação entre tempo total para a solução para o kernel MPI - CG, nós 431, classe de dados C para compiladores gcc 4.9.0 e icc 13.0.1 com flag de compilação -O3, para comunicação Gigabit Ethernet e Myrinet 10Gbps

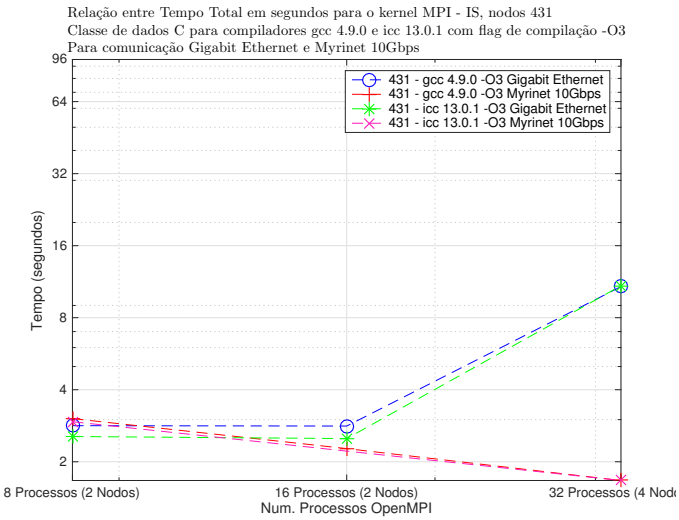

Figure 23. Relação entre tempo total para a solução para o kernel MPI - IS, nós 431, classe de dados C para compiladores gcc 4.9.0 e icc 13.0.1 com flag de compilação -O3, para comunicação Gigabit Ethernet e Myrinet 10Gbps

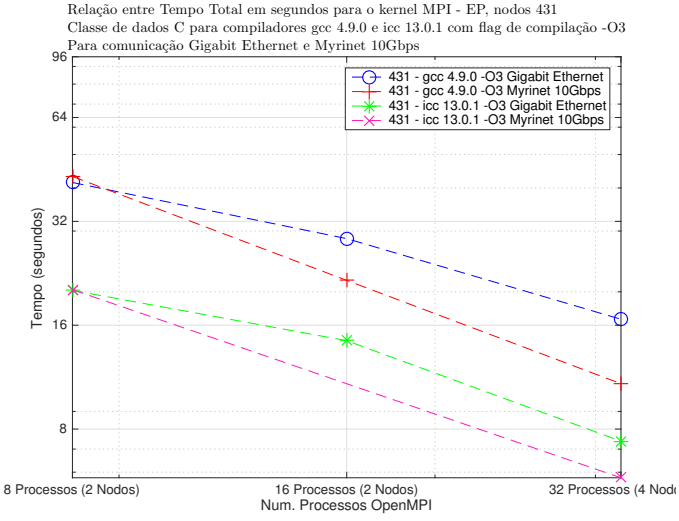

Figure 24. Relação entre tempo total para a solução para o kernel MPI - EP, nós 431, classe de dados C para compiladores gcc 4.9.0 e icc 13.0.1 com flag de compilação -O3, para comunicação Gigabit Ethernet e Myrinet 10Gbps

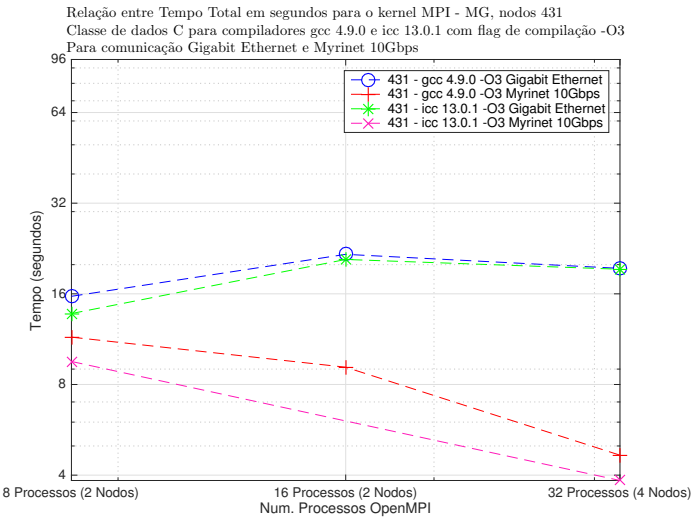

Figure 25. Relação entre tempo total para a solução para o kernel MPI - MG, nós 431, classe de dados C para compiladores gcc 4.9.0 e icc 13.0.1 com flag de compilação -O3, para comunicação Gigabit Ethernet e Myrinet 10Gbps

6.2. Relação entre tempo total para a solução para os kernels OMP CG, IS, EP, MG – nós compute-641

Denote que no momento de conclusão deste caso de estudo foi impossível recolher valores para o teste em comunicação via Myrinet 10Gbps para o compilador ICC 13.0.1 sendo portanto o mesmo tipo de teste retirado das análises gráficas.

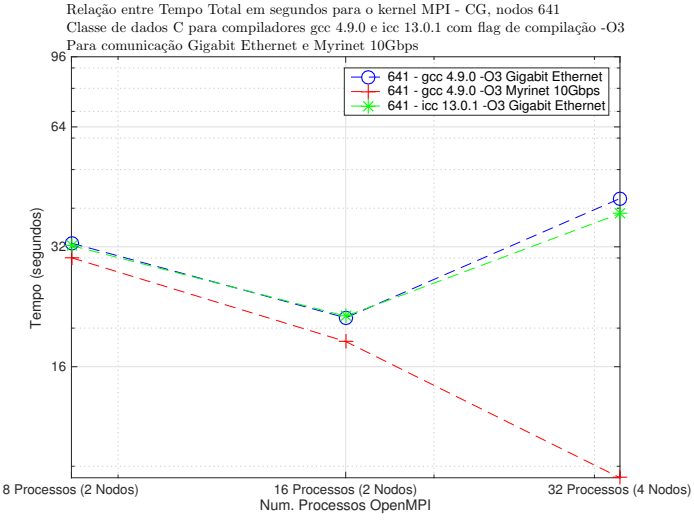

Figure 26. Relação entre tempo total para a solução para o kernel MPI - CG, nós 641, classe de dados C para compiladores gcc 4.9.0 e icc 13.0.1 com flag de compilação -O3, para comunicação Gigabit Ethernet e Myrinet 10Gbps

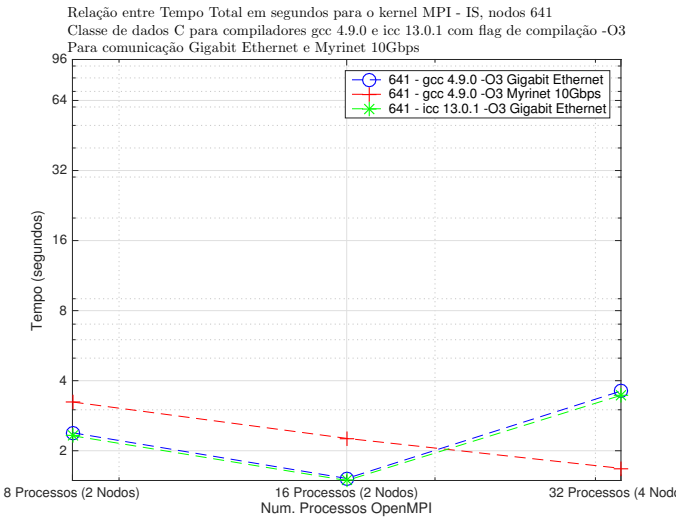

Figure 27. Relação entre tempo total para a solução para o kernel MPI - IS, nós 641, classe de dados C para compiladores gcc 4.9.0 e icc 13.0.1 com flag de compilação -O3, para comunicação Gigabit Ethernet e Myrinet 10Gbps

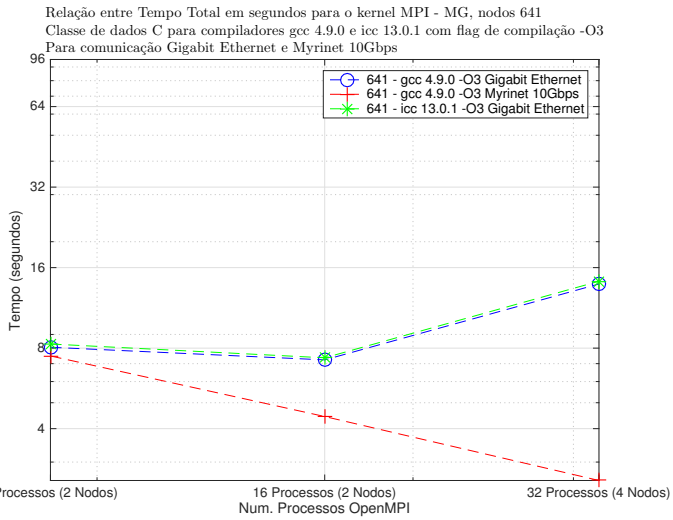

Figure 29. Relação entre tempo total para a solução para o kernel MPI - MG, nós 641, classe de dados C para compiladores gcc 4.9.0 e icc 13.0.1 com flag de compilação -O3, para comunicação Gigabit Ethernet e Myrinet 10Gbps

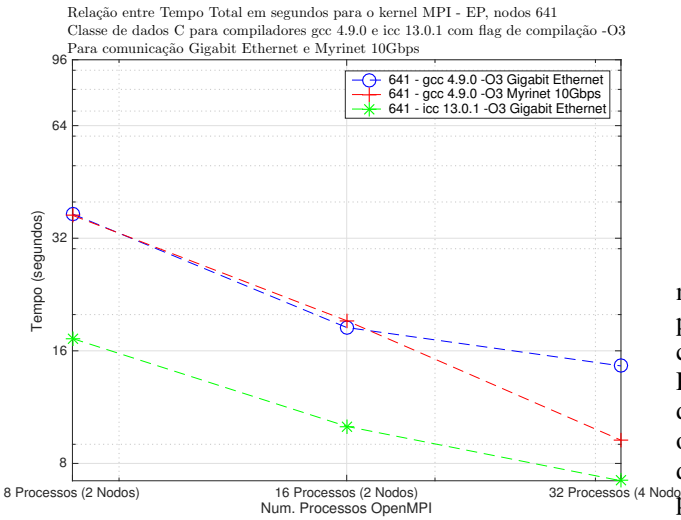

Figure 28. Relação entre tempo total para a solução para o kernel MPI - EP, nós 641, classe de dados C para compiladores gcc 4.9.0 e icc 13.0.1 com flag de compilação -O3, para comunicação Gigabit Ethernet e Myrinet 10Gbps

6.3. Características comuns aos resultados dos nós compute-431 e compute-641

Denote que para todos os kernels em estudo a adição de nós de computação e número de processos MPI envolvidos para solução diminui o tempo de computação quando a comunicação entre processos é feita via Myrinet 10Gbps. Para os casos de comunicação via Gigabit Ethernet o tempo de solução piora sendo que para o caso do kernel EP o agravamento não é tão visível dada a inexistência da dependência de dados e consequente ”independência” entre processos MPI envolvidos.

Analisemos a relação entre tempos totais para as melhores soluções SER/OMP/MPI dos kernels em teste, para ambos os nós de computação.

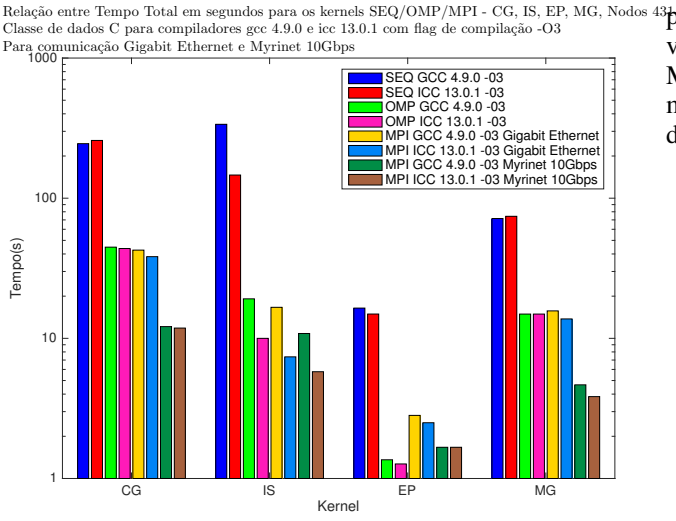

Figure 30. Relação entre tempo total em segundos para a solução para os kernels SEQ/OMP/MPI - CG, IS, EP, MG, nós 431, classe de dados C para compiladores gcc 4.9.0 e icc 13.0.1 com flag de compilação -O3, para comunicação Gigabit Ethernet e Myrinet 10Gbps

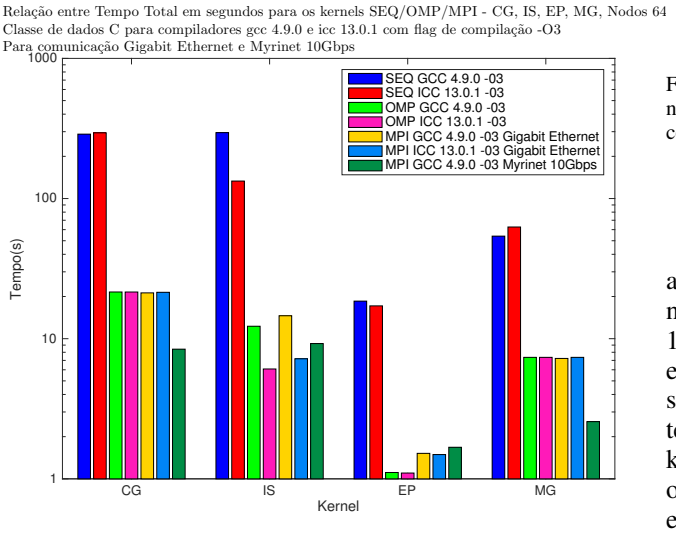

Figure 31. Relação entre tempo total em segundos para a solução para os kernels SEQ/OMP/MPI - CG, IS, EP, MG, nós 641, classe de dados C para compiladores gcc 4.9.0 e icc 13.0.1 com flag de compilação -O3, para comunicação Gigabit Ethernet e Myrinet 10Gbps

Denote que em ambos os nós de computação a relação entre tipo de de soluções se mantém, sendo para o caso dos kernels CG e MG a melhor solução num ambiente de memória distribuída com comunicação via Myrinet 10Gbps e, para o caso dos kernels EP e IS a melhor solução num ambiente de memória partilhada com paralelismo via threads openMP.

Ora, podemos ainda retirar dos gráficos anteriormente apresentados que a solução em ambiente de memória distribuída é sempre melhor com comunicação via Myrinet 10Gbps, contudo e sabendo que a latência de comunicação para o caso da comunicação via Gigabit Ethernet ronda valores entre os 10 e 100 μs e a latência de comunicação via Myrinet 10Gbps ronda valores entre os 1 e 10 μs, teoricamente teríamos para um ambiente com as mesmas condições de teste um ganho de 10. Analisemos essa suposição teórica:

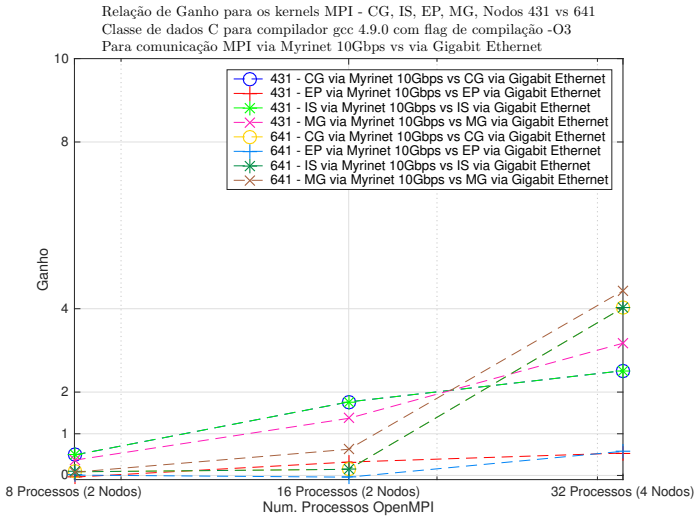

Figure 32. Relação de ganho para os kernels MPI - CG, IS, EP, MG, nós 431 e 641, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3, para comunicação Gigabit Ethernet vs Myrinet 10Gbps

Ora, apesar dos resultados serem satisfatórios não atingem o limite teórico. Tal pode ser devido aos kernels não utilizarem toda a capacidade disponível na rede Myrinet 10Gbps, por durante o período de teste a rede Myrinet se encontrar congestionada ou por os kernels simplesmente não serem IO Bound o suficiente para o tempo de comunicação ter significância suficiente – repare que para o caso de kernels onde não existe dependência de dados o ganho obtido no recurso a Myrinet é menos expressivo – como expectável.

7. Análise do ganho relativo ao kernel SER dos kernels OMP e MPI em ambiente de teste

Analisados os tempos de todas as soluções é necessário exprimir o ganho obtido para todas as soluções em teste, quando comparadas com a solução base sequencial. De seguida exprime-se o resultado gráfico das melhores soluções encontradas para os ambientes de memória distribuída e partilhada, para os nós compute-431 e compute-641.

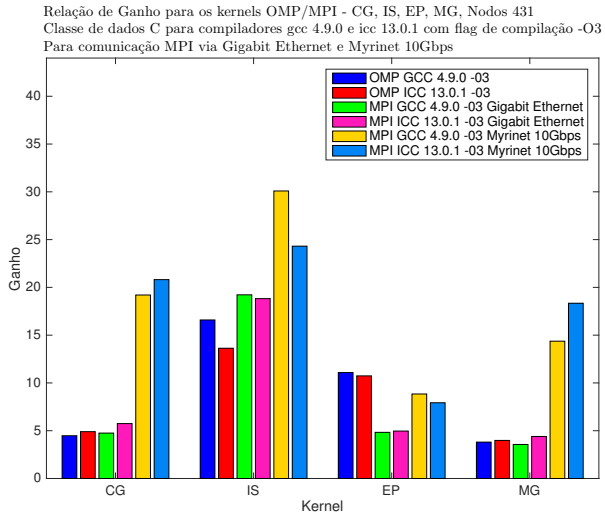

Figure 33. Relação de ganho para os kernels OMP/MPI - CG, IS, EP, MG, nós 431, classe de dados C para compilador gcc 4.9.0 e icc 13.0.1 com flag de compilação -O3, para comunicação(apenas kernels MPI) Gigabit Ethernet vs Myrinet 10Gbps

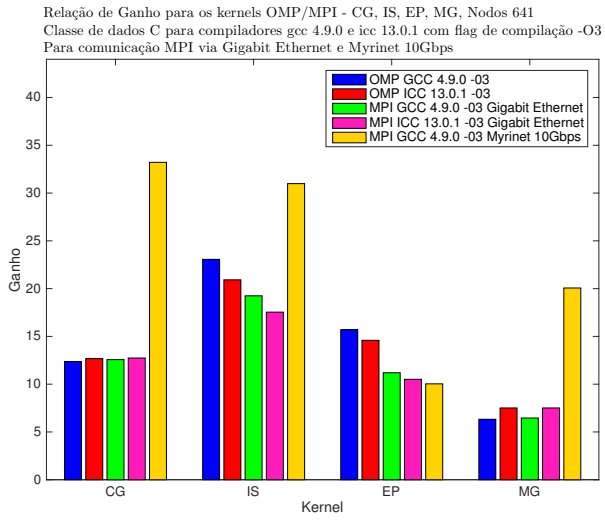

Figure 34. Relação de ganho para os kernels OMP/MPI - CG, IS, EP, MG, nós 641, classe de dados C para compilador gcc 4.9.0 e icc 13.0.1 com flag de compilação -O3, para comunicação(apenas kernels MPI) Gigabit Ethernet vs Myrinet 10Gbps

Assim, como base tanto nos resultados temporais como a nível de ganho versus a versão sequencial dos kernels em teste que as melhores soluções do conjunto de testes realizados são:

- **CG** - Kernel em memória distribuída com comunicação via Myrinet 10Gbps (4 nós 32 Processos), compilador gcc 4.9.0 flag -O3;
- **IS** - Kernel em memória partilhada com nº de processos openMP igual ao número de threads disponível, compilador gcc 4.9.0 flag -O3;
- **EP** - Kernel em memória partilhada com nº de processos openMP igual ao número de threads disponível, compilador gcc 4.9.0 ou compilador icc 13.0.1 flag -O3;
- **MG** - Kernel em memória distribuída com comunicação via Myrinet 10Gbps (4 nós 32 Processos), compilador gcc 4.9.0 flag -O3;

8. Monitorização de propriedades dos sistemas de computação para as melhores soluções do conjunto de testes

8.1. Análise de propriedades para monitorização de execução única

Obtidos os melhores kernels relativamente ao tempo de solução resta-nos analisar essas mesmas soluções relativamente à utilização dos recursos de computação e comunicação dos sistemas. Assim, e recorrendo às métricas disponibilizadas pela ferramenta **dstat** apresenta-se em anexo (dado o tamanho dos gráficos) a influência dos kernels relativamente à ocupação da memória (Memory Stats), CPU (CPU Stats), transferência de dados em rede (Net Stats), utilização dos sistemas pelos diferentes processos (System Stats), e operações de escrita/leitura de disco (Disk Stats). Atente nos 4 agrupamentos de gráficos respectivos aos 4 kernels em teste:

- **CG** - Figura 45;
- **IS** - Figura 46;
- **EP** - Figura 47;
- **MG** - Figura 48;

Pela análise das 4 figuras relativamente ao cpu usage podemos concluir que todos os kernels consomem quase a totalidade do tempo de computação da máquina, e que em adição nenhum dos kernels apresenta uma espera relevante em termos de percentagem de cpu wait. Ora, como podemos constatar pela análise dos gráficos System Stats, kernels sem dependência de dados (EP e IS) apresenta um maior número de interrupções do sistema e context swap. Tal deve-se ao facto de existirem um maior número de processos em ”luta” por CPU time. Já os kernels CG e MG apresentam um número de bytes transferidos de grandeza 10 vezes superior ao apresentado pelos kernels EP e IS. Tal é expectável dado que a melhor solução para os kernels EP e IS centra-se num nó não exigindo comunicação exterior e por contrário a melhor solução para os kernels CG e MG centra-se em 4 nós exigindo comunicação entre os mesmos.

Relativamente à utilização da memória disponível do sistema todos os kernels apresentam valores reduzidos, sendo novamente verificados os maiores valores para os kernels CG e MG em detrimento dos kernels IS e EP.

Em termos de utilização de disco nenhum dos kernels recorre intensivamente a leitura e escrita por grandes períodos, centrando os mesmos ora no início ou término do mesmo. Podemos então verificar duplamente (Disk Stats e Memory Stats) que nenhum kernel esgotou completamente

a memória principal, com recorrência ao disco para troca de dados temporários.

8.2. Análise de propriedades para monitorização de execução ”n-vezes” da melhor solução

Como se pôde constatar anteriormente, foi nos kernels IS e EP que se verificaram o maior número de interrupções do sistema e context swap quando comparados com os kernels MG e CG, mesmo tendo o segundo conjunto de kernels apresentado um maior tempo total de execução. Ora, analisemos na Figura 49 o comportamento agora para a execução ”n-vezes” do kernel EP para 1, 8 e 32 execuções consecutivas na mesma máquina. Denote que foi escolhido o kernel EP em detrimento do IS dado o tempo de computação do kernel EP ser maior, permitindo assim um análise mais alargada da influência da execução dos kernels para o sistema de computação.

Como espectável, os resultados do teste de performance mantêm-se inalterados independentemente da sequência de ”n-testes” realizados, contudo, durante o maior teste podemos observar uma maior percentagem de tempo **%CPU IDLE** em alguns dos testes **%CPU WAIT** – como se pode confirmar observando a Figura 49 – tal é devido a períodos de espera por escrita no disco. Denote que durante esse mesmo período de tempo observamos também um decaimento abrupto do número de CPU context swap e interrupções (não realizando trabalho não existe a possibilidade de interrupção).

Seria interessante alcançar comportamentos de ”ramping” nos sistemas de computação por forma a verdadeiramente forçar os sistemas de computação em teste ao peak perfomance dos vários micro-benchmarks e respectiva ”saturação” do sistema.

8.3. Análise de propriedades para monitorização de execução do mesmo kernel para diferentes implementações SEQ, OMP e MPI

Analisado o comportamento das melhores soluções para os diferentes kernels, centremos agora a nossa atenção na influência do mesmo kernel no sistema de computação para as diferentes melhores soluções – sequencial, em ambiente de memória partilhada e em ambiente de memória distribuída.

Uma vez que utilizamos até ao momento o kernel EP para base de comparação, mantenhamos essa mesma premissa, focando especial atenção nas interrupções do sistema e context swap, assim como nos dados estatísticos relativos ao uso do CPU e memória – como poderemos constatar pelas figuras 39 e 38, as métricas relativas ao uso da rede e de escrita/leitura em disco não apresentam relevância neste kernel específico (dado ser embaraçosamente paralelo).

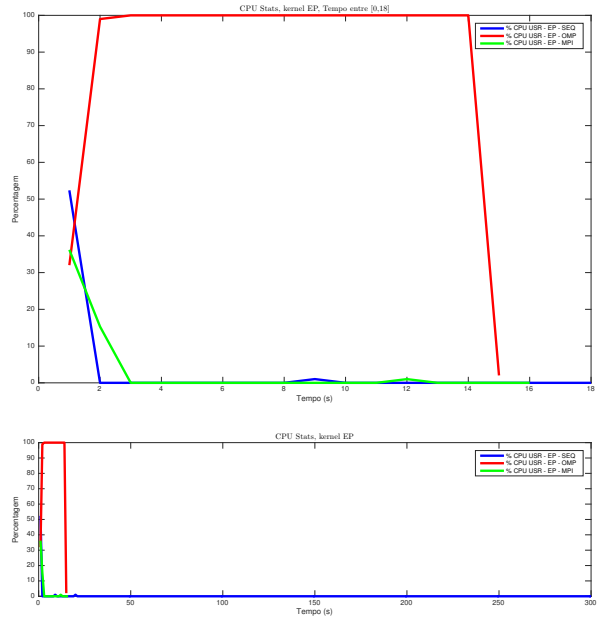

Figure 35. Relação de % de tempo de CPU para os kernels SEQ/OMP/MPI - EP, nós 641, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3, para comunicação Myrinet 10Gbps (memória distribuída)

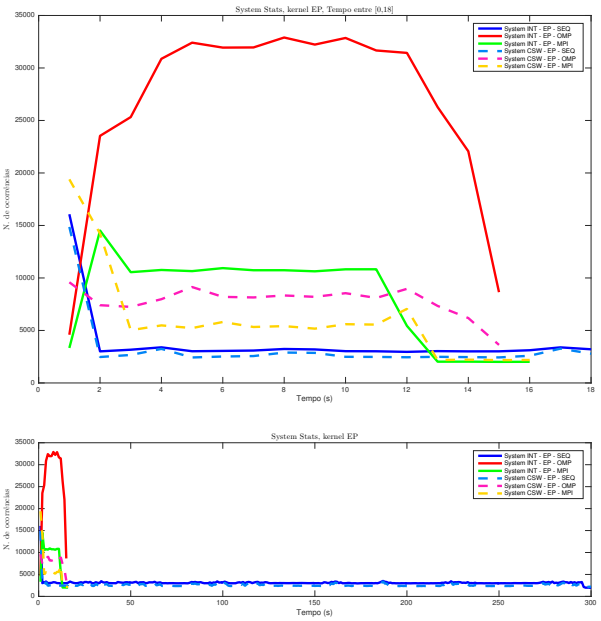

Figure 36. Relação de número de interrupções de processo e número de ”context swap” para os kernels SEQ/OMP/MPI - EP, nós 641, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3, para comunicação Myrinet 10Gbps (memória distribuída)

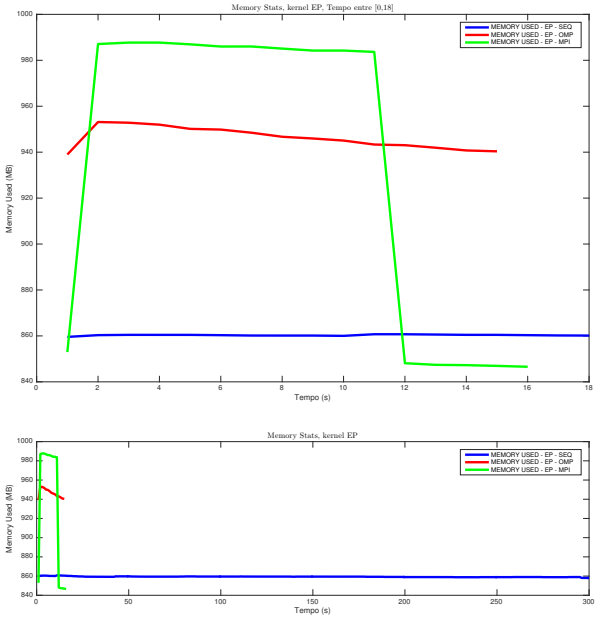

Figure 37. Relação de Memória Utilizada pelos kernels SEQ/OMP/MPI - EP, nós 641, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3, para comunicação Myrinet 10Gbps (memória distribuída)

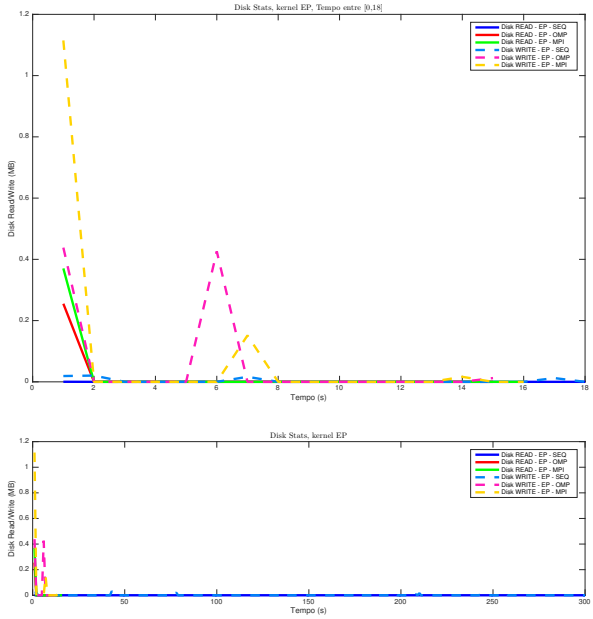

Figure 39. Relação de dados (em MB) escritos/lidos do disco pelos kernels SEQ/OMP/MPI - EP, nós 641, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3, para comunicação Myrinet 10Gbps (memória distribuída)

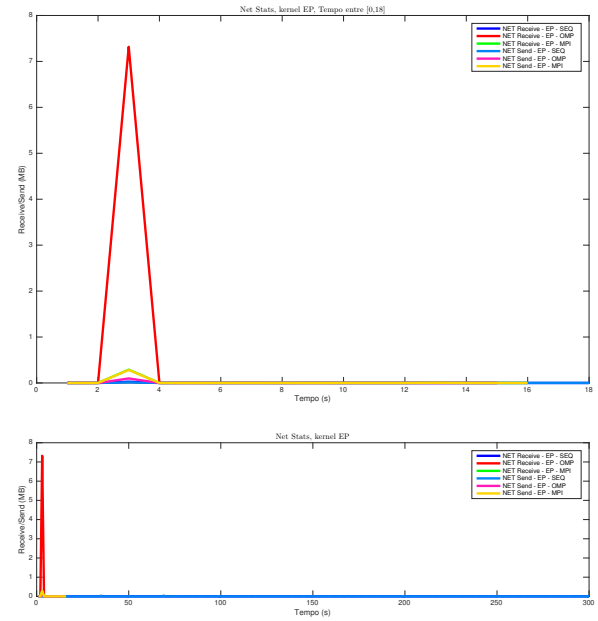

Figure 38. Relação de de dados (em MB) recebidos/enviados através da rede pelos kernels SEQ/OMP/MPI - EP, nós 641, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3, para comunicação Myrinet 10Gbps (memória distribuída)

As figuras 35 a 39 para além de confirmarem o descrito anteriormente para a melhor solução alcançada em termos de tempo total para a solução neste kernel (OMP), confirmam ainda a não existência de comunicação entre processos MPI durante a execução do kernel em memória distribuída – levando uma vez mais à conclusão que este é implicitamente paralelo. Contudo, o facto de não existir dependência de dados entre processos não permite que o kernel alcance o máximo de utilização do CPU em ambiente de memória distribuída muito pelo bottleneck verificado na figura 37 – repare que é para o kernel versão MPI que alcançamos a maior utilização de memória, e que é durante esse mesmo período de maior utilização da memória que se verifica um número elevado de ”context swap” desse mesmo kernel – dada a espera pelos dados presentes em memória.

Tanto os kernels SEQ e MPI não alcançam o máximo de proveito das capacidades de computação do CPU, sendo estas apenas alcançadas pelo kernel OMP. Em nenhuma das versões SEQ/OMP/MPI as métricas relativas ao uso do disco e rede têm especial importância. Analisemos portanto se essa mesma afirmação se mantém para um kernel que implique uma elevada comunicação para a resolução do algoritmo – kernel CG - figuras 40 a 44.

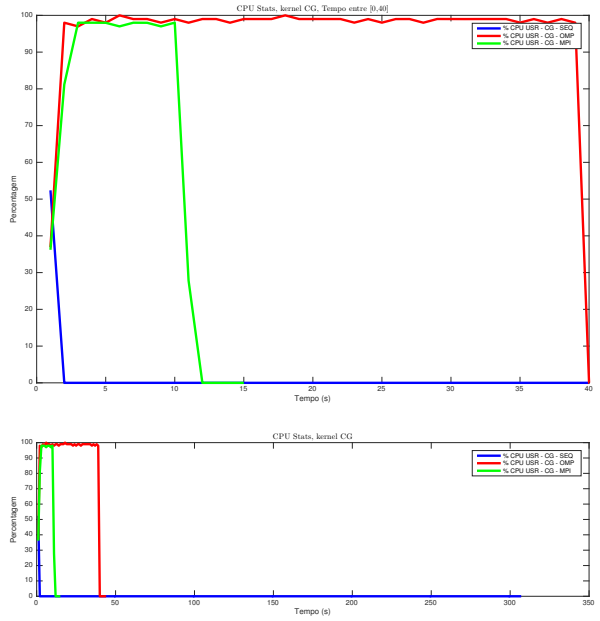

Figure 40. Relação de % de tempo de CPU para os kernels SEQ/OMP/MPI - CG, nós 641, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3, para comunicação Myrinet 10Gbps (memória distribuída)

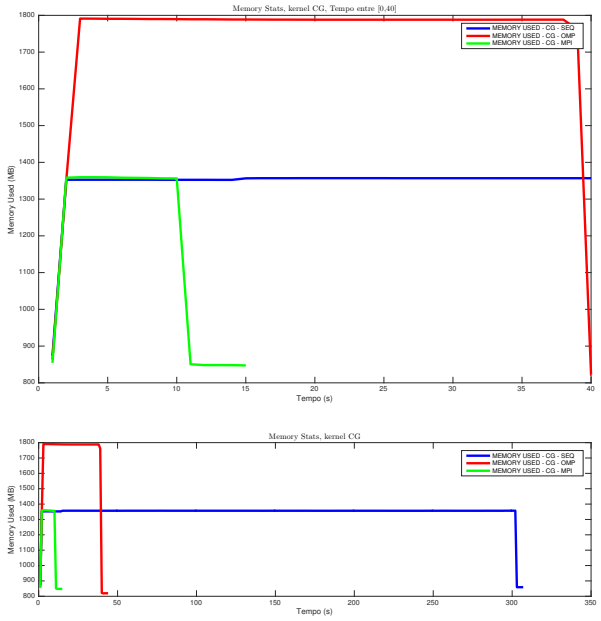

Figure 42. Relação de Memória Utilizada pelos kernels SEQ/OMP/MPI - CG, nós 641, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3, para comunicação Myrinet 10Gbps (memória distribuída)

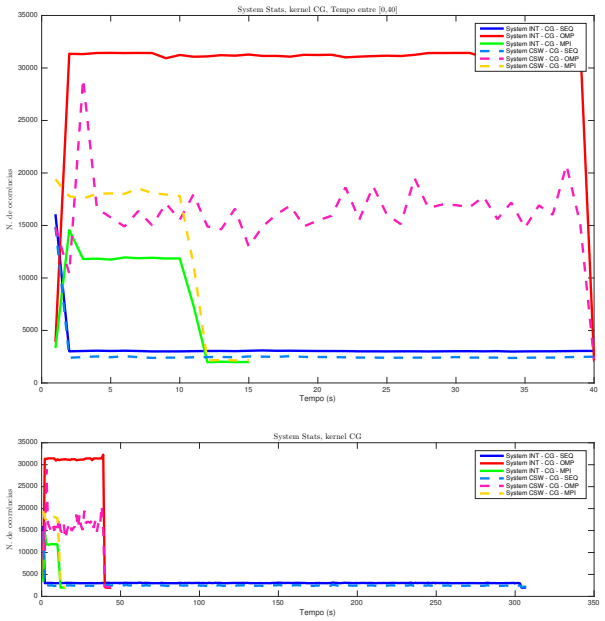

Figure 41. Relação de número de interrupções de processo e número de ”context swap” para os kernels SEQ/OMP/MPI - CG, nós 641, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3, para comunicação Myrinet 10Gbps (memória distribuída)

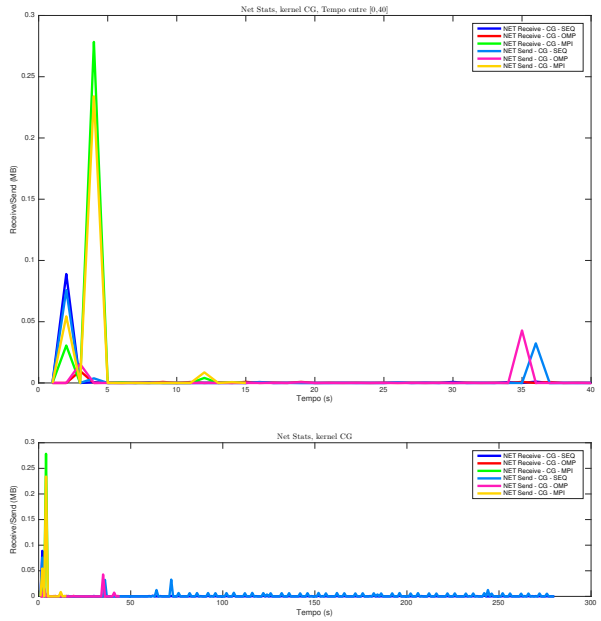

Figure 43. Relação de de dados (em MB) recebidos/enviados através da rede pelos kernels SEQ/OMP/MPI - CG, nós 641, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3, para comunicação Myrinet 10Gbps (memória distribuída)

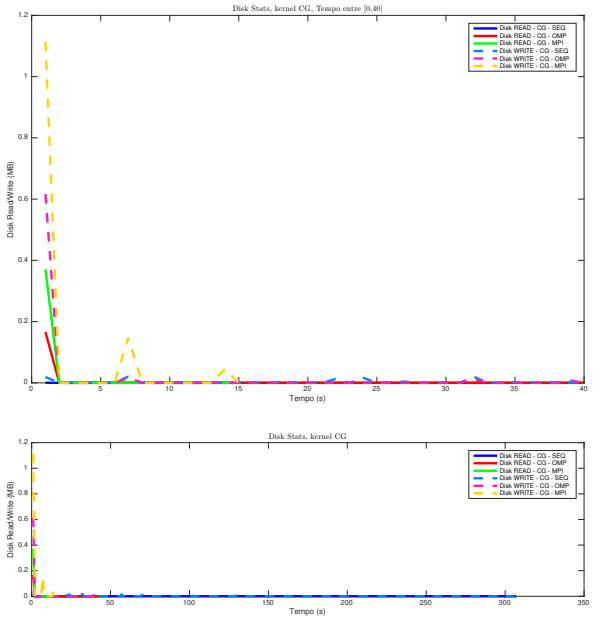

Figure 44. Relação de dados (em MB) escritos/lidos do disco pelos kernels SEQ/OMP/MPI - CG, nós 641, classe de dados C para compilador gcc 4.9.0 com flag de compilação -O3, para comunicação Myrinet 10Gbps (memória distribuída)

Tal como previsto para o kernel CG, com o melhor resultado obtido para ambiente de memória distribuída, todas as métricas corroboram o afirmado durante todo este caso de estudo. Ora, neste caso e em contrário ao que analisado para o kernel EP em ambiente de memória distribuída, o kernel CG versão MPI consegue alcançar o máximo de computação disponível por nó de computação, sendo que mantém esse valor durante toda a execução do algoritmo.

Ora, tendo o melhor tempo de execução sido obtido para uma solução com 4 máquinas é normal que o tempo de computação obtido em MPI seja 1/4 do tempo obtido em ambiente de memória partilhada (por ambos apresentarem um perfil de benchmarking idêntico e por o kernel CG OMP demorar 4 vezes mais a computar a solução) dado que dispomos de 4 vezes mais capacidade de computação (conforme demonstrado na figura 40). Tal poderia não ser verdade se o algoritmo sofre-se perdas por overhead de comunicação, contudo neste caso esse mesmo valor não foi relevante (analisar figura 43).

Uma vez mais, as métricas relativa à escrita e leitura de disco desempenharam um papel pouco significativo no profiling dos kernels. Podemos então concluir que nenhum dos 4 kernels aqui estudados é afectado significativamente por IO, sendo exponencialmente superior a dependência de capacidade de computação e peak memory bandwidth dos sistemas de computação.

9. Conclusão

Relativamente aos kernels em estudo podemos concluir que tanto as otimizações via directivas openMP (ambiente de memória partilhada) e métodos openMPI (ambiente de memória partilhada) são eficientes na otimização de kernels por si só otimizados na sua versão sequencial.

Podemos ainda inferir que kernels sem dependências de dados podem apresentar outras limitações que não o overhead das directivas e de comunicação, podendo por exemplo depender de características específicas de hardware (como o número de unidades de operações de vírgula flutuante).

O valor do benchmarking destes casos de estudo ultrapassa a comparação entre tipos de nós e prende-se com o desenvolvimento e prática de métodos de tratamento e análise de métricas de sistemas de computação de alta perfomance. Inicialmente o caso de estudo era alargado por forma a podermos rapidamente seleccionar as características de compilação e nós de maior interesse, sendo que depois de uma primeira fase foi dado maior ênfase a componentes do teste que apresentavam uma maior disparidade de resultados.

A componente prática laboratorial deste caso de estudo realça ainda mais a diferença entre valores conseguidos por experimentação e valores medidos nos nossos próprios clusters. Podemos resumir este trabalho em três fases, todas elas de extrema importância:

- **Seleção e Fundamentação** - das ferramentas de bechmarking, das métricas a recolher e das configurações a testar;
- **Execução** - das benchmarks no nosso ambiente de clustering e recolha de grandes quantidades dados;
- **Filtragem e Interpretação** - dos dados estatísticos e produção de uma apreciação (neste caso do relatório);

Todos os alunos foram sujeitos a tempo limitado para correr os benchmarks e analisar os resultados, assim como recursos de computação em ambiente de clustering limitados. Podemos considerar este caso de estudo como um misto de prioritização de objectivos, fundamentação de resultados via aplicação de conceitos teóricos e como uma forma de maior ambientação no ambiente de clustering presente no Search6.

References

[1] NAS Parallel Benchmarks, http://www.nas.nasa.gov/publications/npb.html

Appendix

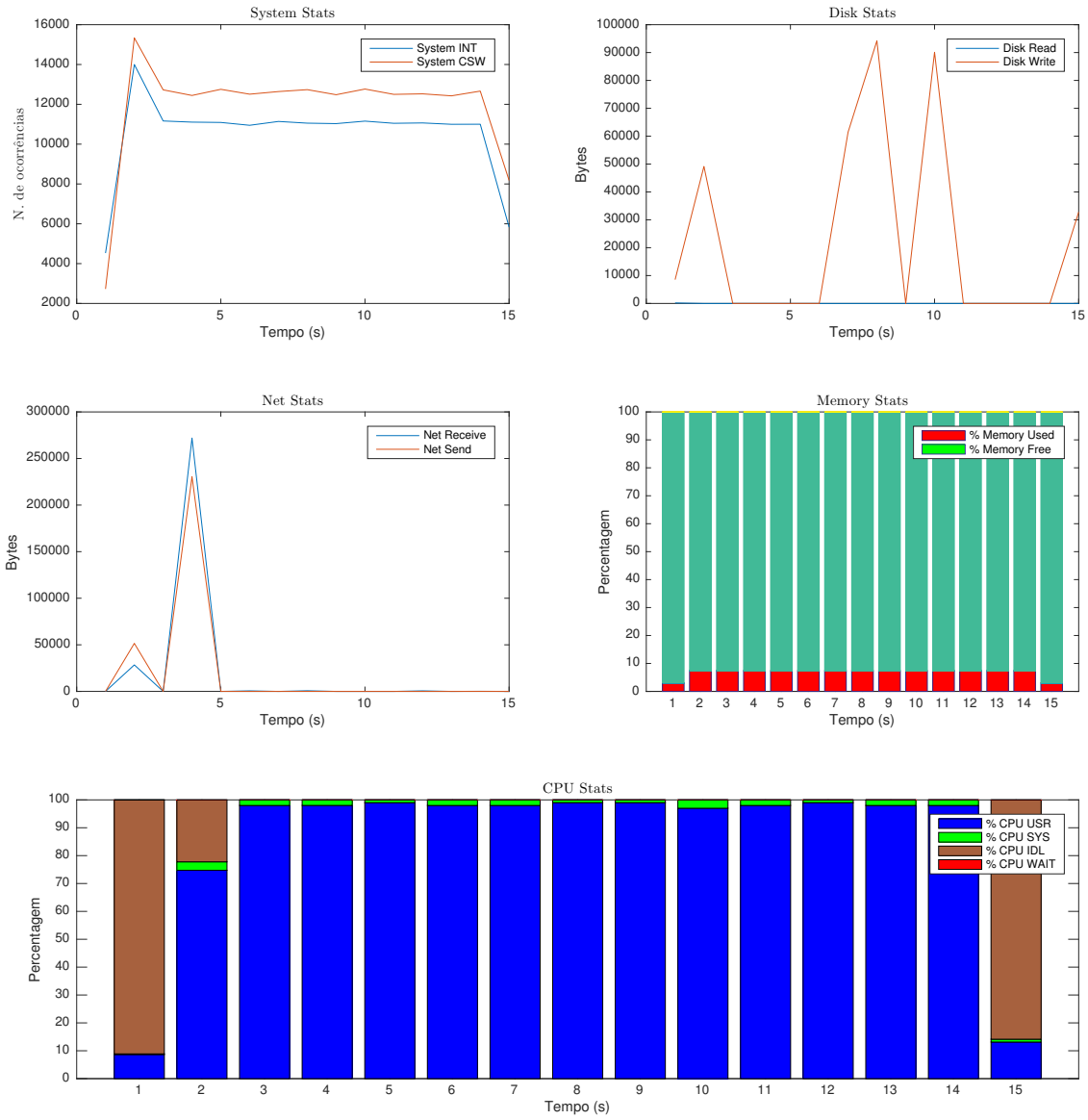

Figure 45. Monitorização de propriedades dos sistemas de computação para a melhor solução do kernel CG – Kernel em memória distribuída com comunicação via Myrinet 10Gbps (4 nós 32 Processos), compilador gcc 4.9.0 flag -O3, nó compute-641

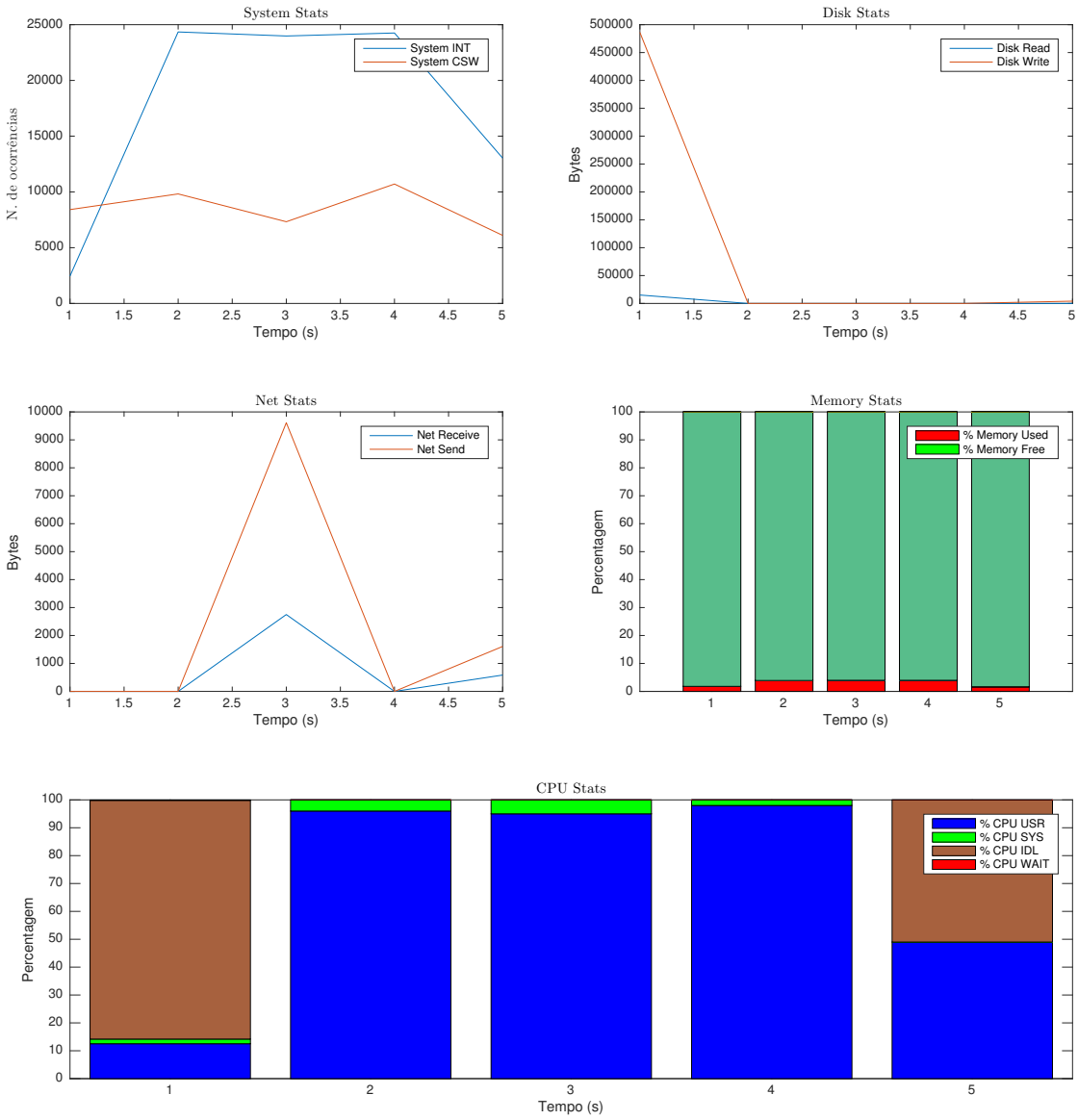

Figure 46. Monitorização de propriedades dos sistemas de computação para a melhor solução do kernel IS – Kernel em memória partilhada com nº de processos openMP igual ao número de threads disponível, compilador gcc 4.9.0 flag -O3, nó compute-641

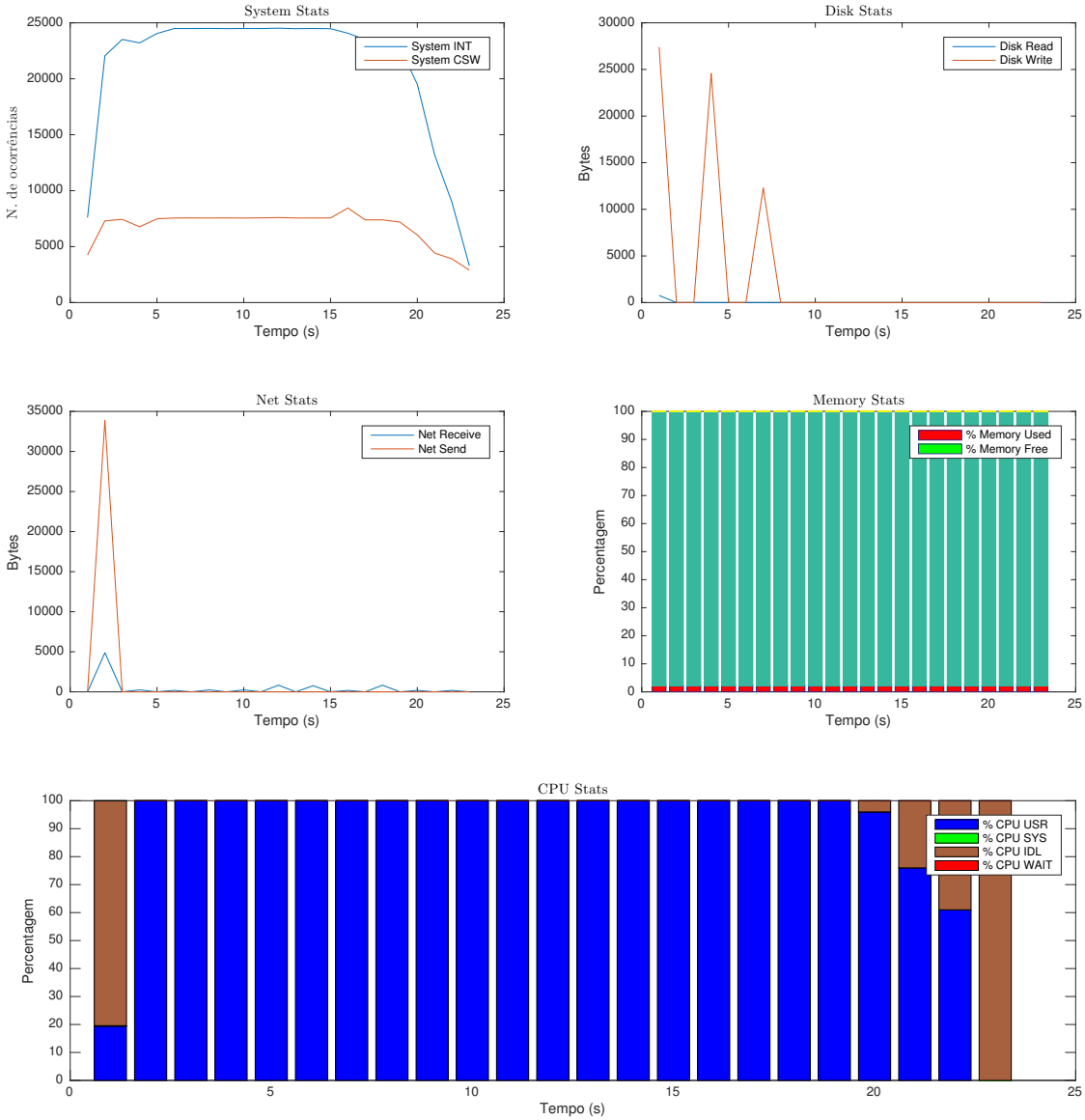

Figure 47. Monitorização de propriedades dos sistemas de computação para a melhor solução do kernel EP – Kernel em memória partilhada com nº de processos openMP igual ao número de threads disponível, compilador gcc 4.9.0 flag -O3, nó compute-641

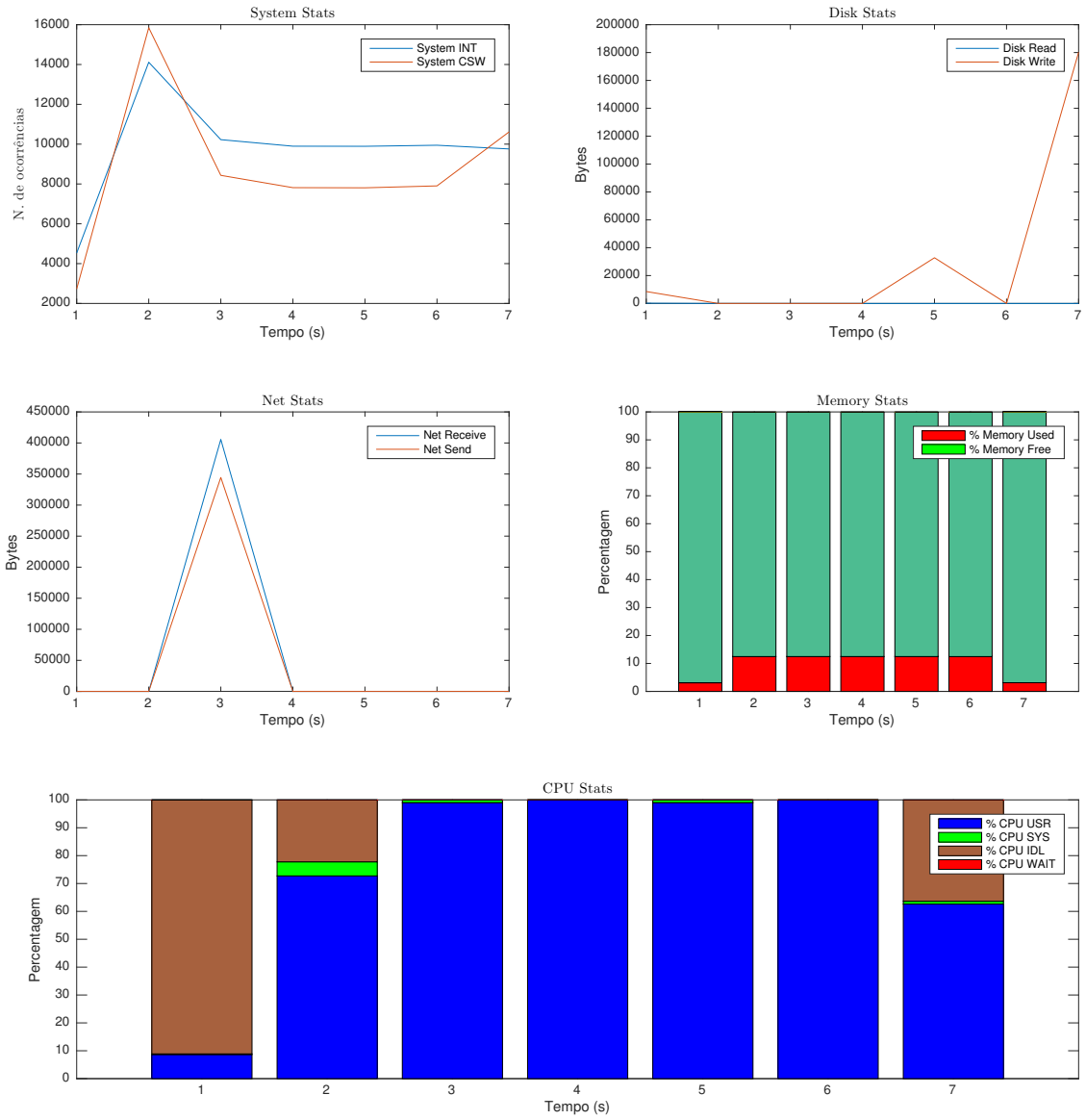

Figure 48. Monitorização de propriedades dos sistemas de computação para a melhor solução do kernel MG – Kernel em memória distribuída com comunicação via Myrinet 10Gbps (4 nós 32 Processos), compilador gcc 4.9.0 flag -O3, nó compute-641

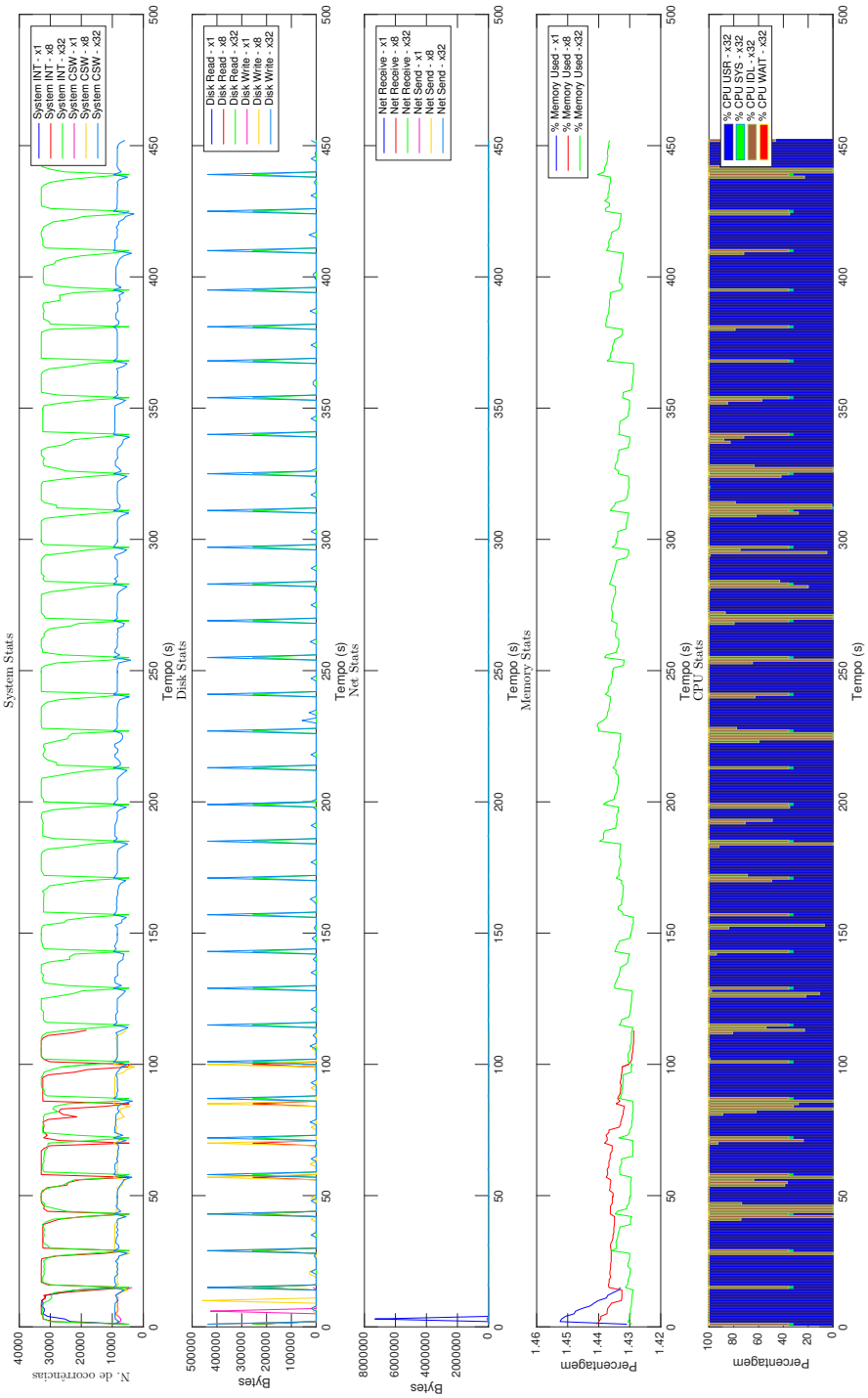

Figure 49. Monitorização de propriedades dos sistemas de computação para a melhor solução do kernel EP executada 1x,8x e 32x – Kernel em memória partilhada com nº de processos openMP igual ao número de threads disponível, compilador gcc 4.9.0 flag -O3, nó compute-641