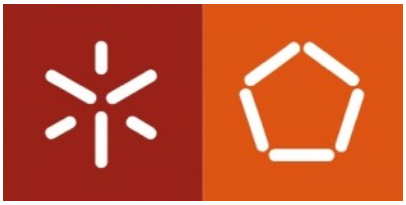


Universidade do Minho
Mestrado Integrado em Engenharia Informática

Paradigmas de Computação Paralela

Paralelismo com MPI

Paradigma de Memória Distribuída

Grupo 07

Carlos Sá - A59905
carlos.sa01@gmail.com

Ana Sousa - A69855
a69855@alunos.uminho.pt

Resumo

Este relatório é o resultado de um estudo feito sobre a performance do algoritmo de multiplicação matriz esparsa (sparse matrix) por vector comumamente conhecido como algoritmo SpMV*. No estudo realizado no trabalho anterior foi utilizado um paradigma de memória partilhada como paradigma pilar de todo o estudo utilizando OpenMP como ferramenta principal de exploração de paralelismo. Ao nível do paradigma de programação neste estudo utilizaremos o paradigma de memória distribuída. No estudo anterior utilizamos OpenMP e seguimos o paradigma de memória partilhada: o paralelismo era obtido no mesmo espaço de endereçamento.*

Neste novo trabalho o estudo do nosso algoritmo será feito com o objectivo de poder realizar computação distribuída i.e, através de actividades paralelas em espaços de endereçamento disjuntos. Para que tal seja possível, será preciso fazer uma partição dos dados, criar condições para que a computação possa ser realizada de forma independente em várias máquinas e seja produzido um único resultado que seja coerente e correto. A comunicação é por isso um aspecto fundamental. Existem diversas abordagens de paradigmas de comunicação em sistemas distribuídos como RMI (Remote Method Invocation) e passagem de mensagens. No nosso estudo utilizaremos a passagem de mensagens como paradigma de comunicação na nossa aplicação paralela com recurso a ***MPI*** *(**M**essage **P**assing **I**nterface). O MPI é uma biblioteca de funções Open Source baseada no modelo* ***SPMD*** *(**S**ingle **P**rocess **M**ultiple **D**ata) pelo que teremos um mesmo programa a correr o algoritmo SpMV com subconjuntos dos dados em diferentes máquinas em paralelo. O conjunto de máquinas alvo deste estudo serão as máquinas do cluster* ***SeARCH*** *do Departamento de Informática da Universidade do Minho.*

Uma das partes mais relevantes deste estudo está na sua componente experimental realizada após a construção do algoritmo que consiste no estudo dos tempos de execução para vários tamanhos de input e vários processos e explorar ganhos de performance nas máquinas do cluster SeARCH.

1 Caso de Estudo

O caso de estudo deste trabalho é o algoritmo de multiplicação de uma matriz esparsa no formato COO (*Coordinate list*) por um vector. Este algoritmo é também conhecido como **SpMV**.

Uma matriz esparsa (***sparse matrix***) é uma matriz em que o número de elementos cujo valor é 0 é relativamente elevado face ao tamanho da matriz. A percentagem de elementos de uma matriz esparsa que são 0 pode, contudo, variar sem perda de designação. De acordo com a definição de matriz esparsa pelo *Wikipédia*:

> *In numerical analysis, a sparse matrix is a matrix in which most of the elements are zero. By contrast, if most of the elements are nonzero, then the matrix is considered dense. The fraction of non-zero elements over the total number of elements (i.e., that can fit into the matrix, say a matrix of dimension of m x n can accommodate m x n total number of elements) in a matrix is called the sparsity (density).*
>
> Wikipédia - Sparse Matrix, *2015*

As matrizes a utilizar para testar o nosso algoritmo deverão então ser matrizes com um grande número de zeros em relação ao número total de elementos. Para tal foi necessário definir a percentagem de elementos não zero face ao número total de elementos da matriz, para que trabalhemos sempre com matrizes que são de facto esparsas e não densas. Com base nesta definição e no exemplo da matriz esparsa fornecido pelo *Wikipédia* achamos que uma boa percentagem de elementos não zero seria de 25%.

O primeiro desafio foi implementar o algoritmo sequencial que recebesse uma matriz esparsa A, um vetor não nulo x e fizesse o cálculo $y = Ax$ em que y corresponde ao resultado do cálculo efectuado. Numa fase inicial, teremos que fazer uma análise teórica do algoritmo e com recurso a uma biblioteca de funções de memória distribuída - **MPI** vamos explorar as oportunidades de paralelismo segundo um paradigma de memória distribuída. Neste paradigma um mesmo algoritmo corre em espaços de endereçamento disjuntos com um subconjunto dos dados.

Uma análise cuidadosa do algoritmo deve ser realizada para averiguar os possiveis impactos de performance. Será estudada a forma como os processos devem aceder aos valores da matriz e à forma como deve ser computado o resultado. Construímos uma versão paralela em MPI do algoritmo e realizamos alguma experimentação que nos permita tirar conclusões quanto aos tempos de execução para diferentes dados de entrada, diferentes números de processos e o ganho com o paralelismo face à versão sequencial do mesmo.

2 Análise teórica do algoritmo

Nesta seccção faremos uma pequena análise teórica do algoritmo SpMV criado pelo grupo. Farêmos a análise do algoritmo sem considerar a sua implementação no paradigma de memória distribuída. Além dos aspectos estritamente relacionados com o algoritmo e que influenciam a performance justificaremos também os Data Sets utilizados e o seu impacto na Hierarquia de Memória das máquinas de teste do algoritmo na subsecção seguinte.

Abaixo encontra-se o algoritmo puramente sequencial que realiza a multiplicação construído pelo grupo:

```
/* Begin Multiplication: result = coo * vect */
for(i=0; i<nLinhas; i++) {
    n = (int)coo[i][0];
    soma=0;
    for(j=1; j<=n; j+=2){
        soma += coo[i][j+1]*vect[(int)coo[i][j]];
    }
    result[i] = soma;
}
```

Da análise deste excerto do código pode-se perceber que para cada linha da matriz COO é feita a multiplicação dos elementos nela guardados. Tratando-se de uma matriz COO apenas são guardadas informações relativas aos elementos não zero da matriz esparsa; desta forma, a primeira posição de cada linha indica o número de elementos que se seguem, sendo que a posição seguinte indica a coluna em que aparece o primeiro elemento não zero e a posição seguinte contém o valor desse elemento, este padrão (coluna, valor) é repetido para todos os elementos não zero da linha em questão.

A questão inicial que tentamos perceber foi: *Sendo o principal objetivo diminuir o tempo de execução do algoritmo e globalmente aumentar a performance do programa, como deve ser feito o acesso aos elementos da matriz e do vetor por forma a maximizar essa performance?* Depois de estudarmos alguma bibliografia (por COD [2], Patterson, 2008) chegamos à conclusão que a melhor forma seria percorrer a matriz por linhas e não por colunas:

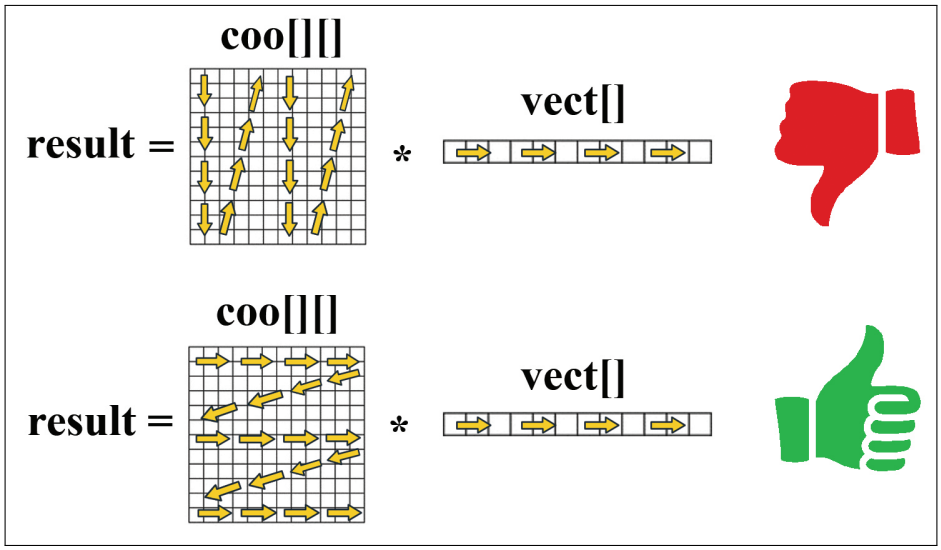


Figura 1: *Travessia da matriz por colunas vs travessia por linhas*

Percorrer a matriz por linhas como se ilustra na Figura 1 é a forma mais eficiente de correr o algoritmo uma vez que todos os valores correspondentes a uma linha da matriz estão armazenados em posições contíguas na memória.

2.1 Selecção dos Data Sets

De forma a testar o nosso algoritmo foram utilizadas 3 matrizes de diferentes tamanhos. Os tamanhos foram especificamente selecionados tendo em conta a hierarquia de memória das máquinas que vamos utilizar neste estudo. Para extrair informações sobre a hierarquia de memória da máquina utilizamos **PAPI** com recurso ao comando ***papi_mem_info***:

```
1   $ papi_mem_info >> MachineInfo/mem_hierarchy.txt
```

Além do PAPI também utilizamos uma sequência de outros comandos UNIX para retirar toda a informação do hardware do nodo de computação utilizado. Para tal foi criado um script - **get_641node_info** - que recolhe toda a informação automaticamente e que acompanha o código anexo a este relatório.

Pelo apêndice B podemos verificar que a máquina possui 20480 KB (20MB) de tamanho de cache L3 com associatividade 20-way e linhas de cache de 64bytes. Como as nossas matrizes são de **doubles**, em 20480KB de tamanho de cache L3 conseguimos guardar: $\frac{20480KB}{8Bytes} = 2560K$ doubles (assumindo 1 double = 8 Bytes). Se dividirmos este total por 2 (matriz + vector) podemos guardar no máximo cerca de 1280K doubles em cada um. Como temos linhas de cache de 64 Bytes convém que a dimensão da matriz esparsa no formato COO e do vetor

sejam múltiplos de 64 por forma a favorecer a localidade espacial. Após estes cálculos, o grupo decidiu então utilizar uma matriz de dimensão 1024×1024 para a matriz esparsa e 1024 para a dimensão do vector. Assim, temos no total um volume de dados de $\approx 1050K$ doubles que até é bastante inferior ao limite máximo de doubles que cabem na cache L3 (2560K doubles) pelo que a matriz no formato COO e o vector cabem integralmente na L3.

Para estudar o comportamento do algoritmo num enquadramento completamente oposto vamos duplicar o valor de cada dimensão para garantir que os valores da matriz e do vetor não cabem na cache e forçar o carregamento dos valores da memória principal. Assim, o segundo e terceiro tamanho de dados das matrizes escolhidos para testar o algoritmo é de 2048×2048 e 4096×4096 tendo o vetor também a mesma ordem de grandeza.

Construímos uma função **forceLoadRam** que basicamente vai enchendo a cache com valores sem significado para garantir que os valores da matriz COO e do vetor são carregados da memória principal.

3 Desenho do algoritmo paralelo

Após a realização da análise teórica do algoritmo (início da secção 2) o grupo levou a cabo um conjunto de fases de desenvolvimento com o objetivo de construir a versão paralela em **MPI** do algoritmo SpMV. Cada uma dessas fases está documentada nas subsecções deste capítulo.

3.1 Partição e distribuição dos dados

Esta fase está relacionada com a decomposição do problema, sendo que pode ser feita através da sua decomposição funcional ou da decomposição do domínio dos dados.

De acordo com a análise teórica do algoritmo feita na secção 2, a matriz deverá ser percorrida por linhas, pelo que achamos que deverá ser feita uma decomposição do domínio dos dados. Desta forma, a nossa estratégia de decomposição será partir a matriz em blocos (conjuntos) de várias linhas, sendo que a cada processo deverá ser atribuída a mesma quantidade de linhas. Apesar de tentarmos com esta partição garantir uma correta *distribuição da carga*, esta tarefa é difícil de realizar visto que o número de elementos de cada linha da matriz é diferente pois estamos a trabalhar com uma matriz esparsa no formato COO.

Na imagem 2 podemos perceber o funcionamento do algoritmo. o processo *MASTER* é responsável pela partição da matriz e pelo envio das várias linhas aos demais processos. A cada receção de uma linha, o processo efetua a multiplicação de cada linha pelo vetor. Se o número de linhas total da matriz COO não for múltiplo do número de processos, poderão sobrar uma ou duas linhas da matriz para processar ficando o processo *MASTER* responsável pelo seu tratamento.

Cada processo vai fazer a multiplicação de cada linha que recebeu pelo vetor e guardará o resultado num array local. Quando todos os processos terminarem o trabalho que lhes foi atribuído (situação garantida pela primitiva *MPI_ Barrier*) os arrays de resultados dos processos serão condensados num só (*MPI_ Reduce* - operação MAX).

3.2 Comunicação entre processos e sincronismo

O nosso algoritmo foi implementado tendo por base o algoritmo de "Master-Slave", sendo que existe apenas o processo *MASTER* envia mensagens para os restantes processos (*Slaves*), logo os *Slaves* não comunicam entre si pois não existe qualquer dependência entre eles.

Tal como já foi referido na subsecção anterior, cada processo fica encarregue de realizar a multiplicação de um subconjunto de linhas da matriz pelo vetor, guardando os resultados num array local. Como os resultados são locais aos processos é necessário fazer a junção dos vários arrays num único, só se devendo fazer esta junção quando todos os processos tiverem terminado. Para garantir essa sincronização dos processos usamos a primitiva *MPI_ Barrier*.

A necessidade de sincronização é também necessária antes de o *MASTER* começar a enviar mensagens para que se assegure que os processos estão todos funcionais para receção das mensagens e para que as medições de tempo não sejam influenciadas por algum problema num dos processos. É usada novamente uma barreira.

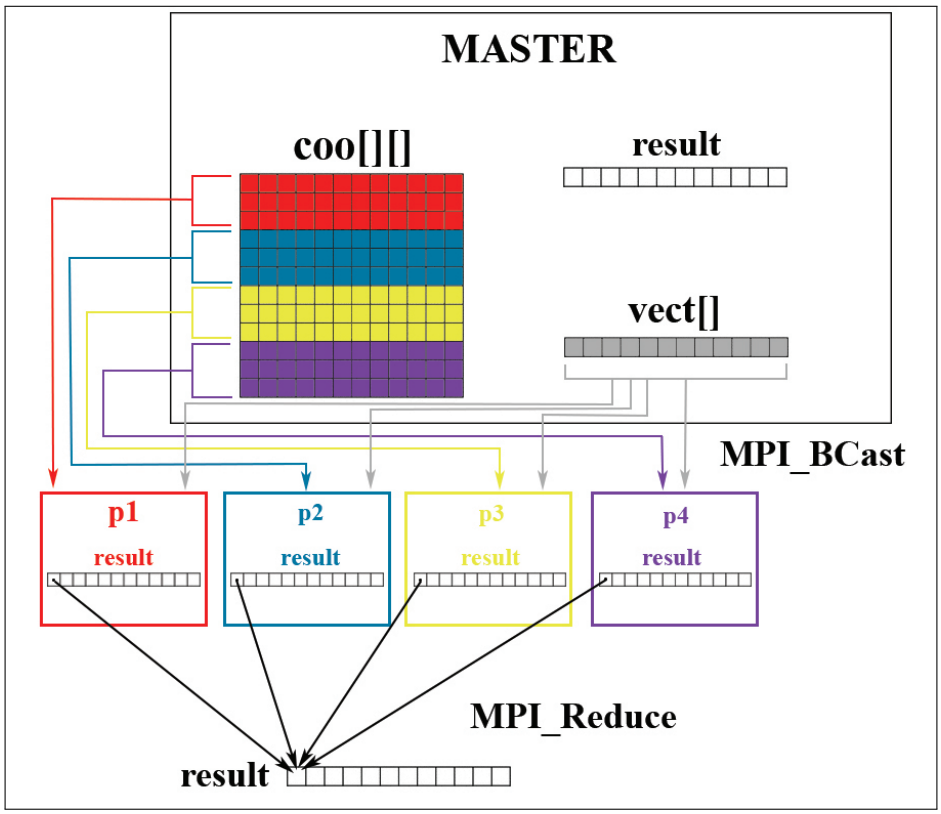


Figura 2: *Distribuição da matriz e do vetor pelos processos*

3.3 Aglomeração de computação e de comunicação

Nesta fase tentam-se arranjar formas de tornar o algoritmo mais eficiente. Uma forma seria através da agregação de mensagens para minimizar o esforço de comunicação.

O grupo implementou uma solução para a agregação de blocos de linhas numa única mensagem, contudo e apesar de a distribuição de tempo pelos vários tipos de tarefas (comunicação, computação) se tornar menos díspar, a estimativa para o tempo de execução do algoritmo piorou, pelo que decidiu-se não usar essa implementação e continuar a enviar apenas uma linha por mensagem.

3.4 Mapeamento das tarefas no sistema distribuído

Pela Figura 2 analisada anteriormente já tinhamos estudado a forma como foi feita a distribuíção da matriz e do vetor pelos diferentes processos. Após a finalizarmos a construção do algoritmo surge a necessidade de mapear as tarefas nas máquinas do *SeARCH Cluster*.

Foram utilizadas **4 máquinas** do segmento **641** para executar o nosso código nos diferentes **sockets** e recolher os tempos. Para analisar os outputs obtidos pela submissão dos jobs convém compreender o hardware que estamos a utilizar, nomeadamente quanto ao números de processadores e número de cores em cada máquina. Pela informação da tabela C podemos verificar que cada máquina deste segmento possui dois sockets de 8 cores físicos.

A abordagem inicial foi utilizar a rede **Ethernet** do sistema distribuído que interliga as máquinas. Para tal foi necessário utilizar **openmpi_eth v1.8.4** e executar **ompi_info –param mpi all** na script de submissão dos *jobs*.[1]

[1] A script de submissão dos jobs (com nome **runSearch**) encontra-se junto do código em anexo a este relatório.

Se visualizarmos a script de submissão dos jobs poderemos visualizar que o programa paralelo MPI é executado de acordo com o seguinte comando:

```
$ mpirun -np $ppn --map-by core -mca btl self,sm,tcp --report-bindings bin/tp2_mpi ↩
    $matrix_size $matrix_size $vec_size "--map-by core" "eth" 2
```

Desta forma, além do ficheiro de output com o resultado da execução do job é devolvido também um ficheiro de texto (sem extensão) com uma representação gráfica que nos permite visualizar em que socket e em que core da máquina é que os *ranks* estão a ser mapeados.

Vejamos um exemplo de um excerto desse ficheiro[2]:

```
[compute-641-6.local:29222]
[hwt 0-1]
                                (....)
  MCW rank 0 bound to socket 0[core 0]: [BB/../../../../../../..][../../../../../../../..]
  MCW rank 1 bound to socket 0[core 1]: [../BB/../../../../../..][../../../../../../../..]
  MCW rank 2 bound to socket 0[core 2]: [../../BB/../../../../..][../../../../../../../..]
  MCW rank 3 bound to socket 0[core 3]: [../../../BB/../../../..][../../../../../../../..]
  MCW rank 4 bound to socket 0[core 4]: [../../../../BB/../../..][../../../../../../../..]
  MCW rank 5 bound to socket 0[core 5]: [../../../../../BB/../..][../../../../../../../..]
  MCW rank 6 bound to socket 0[core 6]: [../../../../../../BB/..][../../../../../../../..]
  MCW rank 7 bound to socket 0[core 7]: [../../../../../../../BB][../../../../../../../..]
                                (....)
```

Neste caso, podemos verificar que cada rank está a ser distribuído nos diferentes cores do socket 0 da máquina e todos eles executam em paralelo. Os processos são distribuídos uniformemente por cada core. Podemos perceber que a partir do momento em que existem mais processos do que cores disponíveis num mesmo socket, os restantes processos são mapeados no segundo socket como se ilustra no excerto do output gerado abaixo:

```
[compute-641-6.local:29684]
[hwt 0-1]
                            (....)
 MCW rank 0 bound to socket 0[core 0]: [BB/../../../../../../..][../../../../../../../..]
 MCW rank 1 bound to socket 0[core 1]: [../BB/../../../../../..][../../../../../../../..]
 MCW rank 2 bound to socket 0[core 2]: [../../BB/../../../../..][../../../../../../../..]
 MCW rank 3 bound to socket 0[core 3]: [../../../BB/../../../..][../../../../../../../..]
 MCW rank 4 bound to socket 0[core 4]: [../../../../BB/../../..][../../../../../../../..]
 MCW rank 5 bound to socket 0[core 5]: [../../../../../BB/../..][../../../../../../../..]
 MCW rank 6 bound to socket 0[core 6]: [../../../../../../BB/..][../../../../../../../..]
 MCW rank 7 bound to socket 0[core 7]: [../../../../../../../BB][../../../../../../../..]
 MCW rank 8 bound to socket 1[core 8]: [../../../../../../../..][BB/../../../../../../..]
 MCW rank 9 bound to socket 1[core 9]: [../../../../../../../..][../BB/../../../../../..]
 MCW rank 10 bound to socket 1[core 10]: [../../../../../../../..][../../BB/../../../../..]
 MCW rank 11 bound to socket 1[core 11]: [../../../../../../../..][../../../BB/../../../..]
 MCW rank 12 bound to socket 1[core 12]: [../../../../../../../..][../../../../BB/../../..]
 MCW rank 13 bound to socket 1[core 13]: [../../../../../../../..][../../../../../BB/../..]
 MCW rank 14 bound to socket 1[core 14]: [../../../../../../../..][../../../../../../BB/..]
 MCW rank 15 bound to socket 1[core 15]: [../../../../../../../..][../../../../../../../BB]
                                (....)
```

[2]Algum conteúdo repetido em todas as linhas foi retirado

No ficheiro de output encontra-se uma evolução mais exaustiva do mapeamento dos ranks em cada core de cada socket.

Por forma a fazer alguma comparação com os tempos obtidos utilizando uma rede mais rápida um dos objetivos iniciais era fazer alguma experimentação utilizando a rede *myrinet*. Após diversas tentativas o grupo não conseguiu configurar a script de submissão dos jobs por forma a tirar partido desta rede de 10Gbps.

3.5 Implementação OpenMP vs Implementação MPI

Apesar de o OpenMP e do MPI serem aplicados em paradigmas de memória diferentes (OpenMP - memória partilhada e MPI - memória distribuída) a estratégia de implementação foi semelhante. Em ambas as implementações fez-se a divisão do trabalho por linhas da matriz esparsa no formato COO. A diferença é que na implementação com OpenMP (já realizada no trabalho anterior) a distribuição das linhas é feita pelas threads que executam numa única máquina, para memória distribuída essa distribuição é feita ao nível dos vários processos e executada num sistema distribuído em várias máquinas diferente, em paralelo.

Outra diferença relevante e que influencia o tempo de execução do algoritmo é a necessidade de comunicação entre processos através da troca de mensagens. Como iremos ver ao longo do estudo, essa comunicação assente numa rede Ethernet faz com que a a nossa implementação OpenMP para os *data sets estudados* para o nosso algoritmo SpMV seja uma melhor solução em termos de performance do que a a implementação MPI. No gráfico da figura 14 mais à frente neste estudo faremos uma comparação final dos resultados obtidos para as duas implementações.

4 Metodologia de medição

Para reservar 2 máquinas para executar o código teríamos então que reservar um nodo (-lnodes=2) de 32 "Cores"(ppn=32). Pela mesma lógica para reservar as 4 máquinas submetemos um job com (-lnodes=4) de 32 "Cores"(ppn=32). Deste modo, temos um total de 128 processos que serão mapeados em 4 máquinas do segmento 641 para três tamanhos de matrizes.

Como pretendemos executar o algoritmo para mais do que um nodo, utilizamos um total de 4 scripts diferentes para realizar as medições para os 4 nodos. As scripts de submissão dos jobs deste trabalho encontram-se no código anexo a este relatório.

Em termos de medições foram analisadas várias métricas diferentes. O tempo de execução do algoritmo paralelo foi aproximado por estimativa pelo walltime do tempo do processo mais lento (*Slowest Process Time*) como será explicado na secção 6 e utilizado para medir ganhos.

Além do **Slowest Process Time** medimos os seguintes tempos para vários *data sets*, para vários processos por nodo, em vários nodos:

- Tempo gasto a enviar dados a todos os processos (tempo gasto com **MPI_BCast**;
- Tempo total gasto em sends processo -> processo (**MPI_Send**);
- Tempo total gasto em receives (**MPI_Recv**);
- Tempo total de Computação.

4.1 Minimização de factores externos

- Sempre que foram reservados nodos para executar o algoritmo, foram reservados nodos de forma exclusiva para executar os nossos *jobs*;
- Usamos o contador de tempos do MPI para fazer a medição dos tempos (**MPI_Wtime()**);
- A resolução de tempos utilizada é o **milissegundo (ms)**, para todos os tempos;
- As áreas delimitadas para a medição dos tempos não inclui nenhum tipo de I/O (como *printf's*, por exemplo);

- Foram feitas 5 medições para cada **tempo**. Neste relatório, em tabelas de tempos como as que se encontram no apêndice F apenas consta o resultado de aplicar a mediana sobre cada tempo (atenuar maiores oscilações de valores);

5 Compilação e biblioteca openMPI

Acompanhado do respectivo relatório encontra-se uma *Makefile* e os scripts de submissão dos jobs no cluster *SeARCH*. Através da Makefile poderemos verificar que nos limitamos a usar o seguinte comando de compilação que utiliza **mpicc** com um nível de optimização -O3:

```
$ mpicc -O3 -Wall -Wextra -std=c11 -fopenmp
```

É utilizado o compilador **gcc v.4.9.3** e a biblioteca **openmpi_eth v1.8.4**

A execução do programa é feita diretamente pela script no Search (**runSearch**) que processa os resultados pretendidos para os diferentes números de processos e para os diferentes tamanhos de matrizes de uma só vez. Na execução, é utilizado o comando básico **mpirun**:

```
$ mpirun -np $ppn --map-by core -mca btl self,sm,tcp --report-bindings bin/tp2_mpi ↩
    $matrix_size $matrix_size $vec_size "--map-by core" "eth"
```

As flags de compilação utilizadas definem número de processos a utilizar, fazem o mapeamento dos processos nos cores disponiveis, permitem a utilização da rede ethernet para comunicação, e permitem gerar um ficheiro com uma representação visual com o mapeamento dos processos nos cores dos sockets da máquina.

6 Estudo dos ganhos: algoritmo em MPI vs Algoritmo Sequencial Inicial

Nesta secção iremos apresentar alguns resultados resultantes da componente experimental deste estudo. O estudo dos ganhos nesta secção refere-se aos ganhos obtidos com MPI em relação à versão sequencial construída no trabalho anterior. Para facilitar a análise dos tempos mostramos os resultados obtidos para cada experiência com 1, 2, 3 e 4 nodos separadamente.

Pela secção 6.1 é possível verificar que a distribuição da computação por várias máquinas correndo o nosso algoritmo para vários processos não traz ganhos de desempenho. À medida que aumentamos o número de processos mais a performance do algoritmo é afetada negativamente. E quando aumentamos o número de máquinas onde o algoritmo irá executar, o impacto negativo sobre os ganhos é ainda maior. O que percebemos estar a acontecer é que a performance acaba por ser completamente afetada pela criação de processos, comunicação e sincronização entre eles. Ao longo de todo o trabalho foram feitas diversas alterações no código e feitos vários ensaios calculando os tempos de diversas formas sendo que apresentamos de seguida os resultados do melhor ensaio que realizamos a nível de resultados.

Nos gráficos da secção 6.1 os ganhos são obtidos pelo quociente entre o tempo de execução do algoritmo sequencial inicial e a estimativa do tempo de execução do algoritmo paralelo: o walltime do processo mais lento (*Slowest Process Time*[3]).

6.1 Gráficos SpeedUp's

Abaixo apresentamos os gráficos obtidos na componente experimental deste trabalho. O que verificamos foi que para os tamanhos de matrizes selecionados a versão paralela MPI não nos traz nenhum tipo de ganho face à versão sequencial. Percebemos que para obter ganhos usando MPI para este algoritmo é necessário ter data sets bastante maiores do que os tamanhos de matrizes que selecionamos. Teriam que ser data sets que permitissem que

[3]Foi o slowest process time como aproximação de acordo com as recomendações do grupo de slides sobre MPI das aulas: Optimising performance (MPI)

o tempo de de computação se sobrepusesse ao tempo que é gasto na comunicação. O facto do algoritmo SpMV ser um algoritmo que realiza pouca computação torna os ganhos difíceis de obter.

Pelos gráficos podemos verificar a queda dos valores dos ganhos (que são na realidade perdas) que ocorre quando utilizamos 1 nodo (Fig. 3) e quando utilizamos 3 (Fig. 5) e 4 (Fig.6) nodos com comunicação Ethernet.

Devido a dificuldade de configuração dos jobs para utilizar a rede Myrinet, embora fosse um objetivo do grupo, acabamos por não conseguir realizar nenhum ensaio utilizando esta rede. Assim sendo, achamos que faz mais sentido discutir e fazer uma análise sobre a comunicação existente do que discutir potenciais ganhos de performance ao realizar a computação do nosso algoritmo segundo um paradigma de memória distribuída. Esta análise foi feita na secção 7.

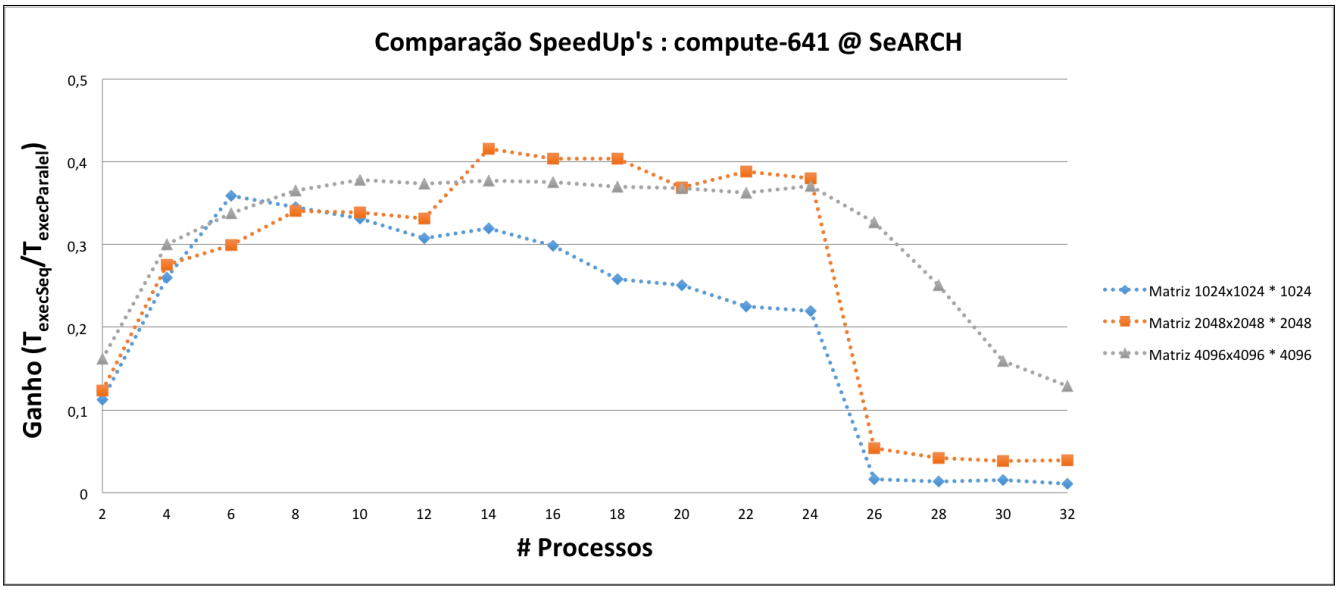


Figura 3: *Processos em apenas 1 nodo*

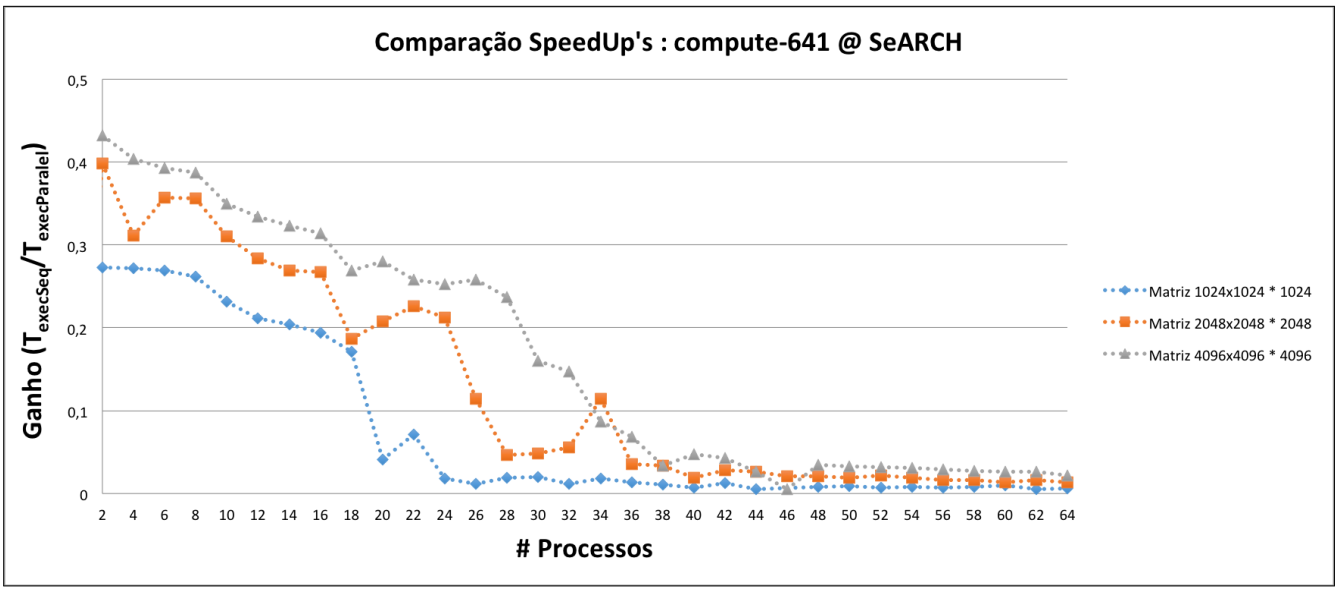


Figura 4: *Processos em 2 máquinas*

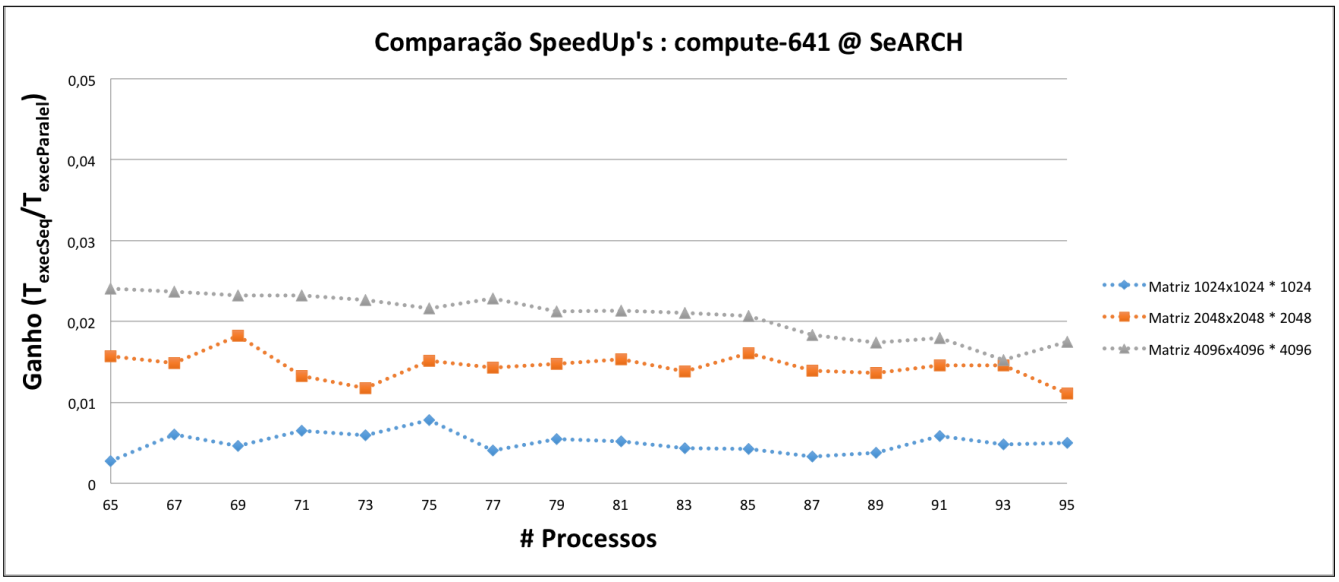


Figura 5: *Processos em 3 máquinas*

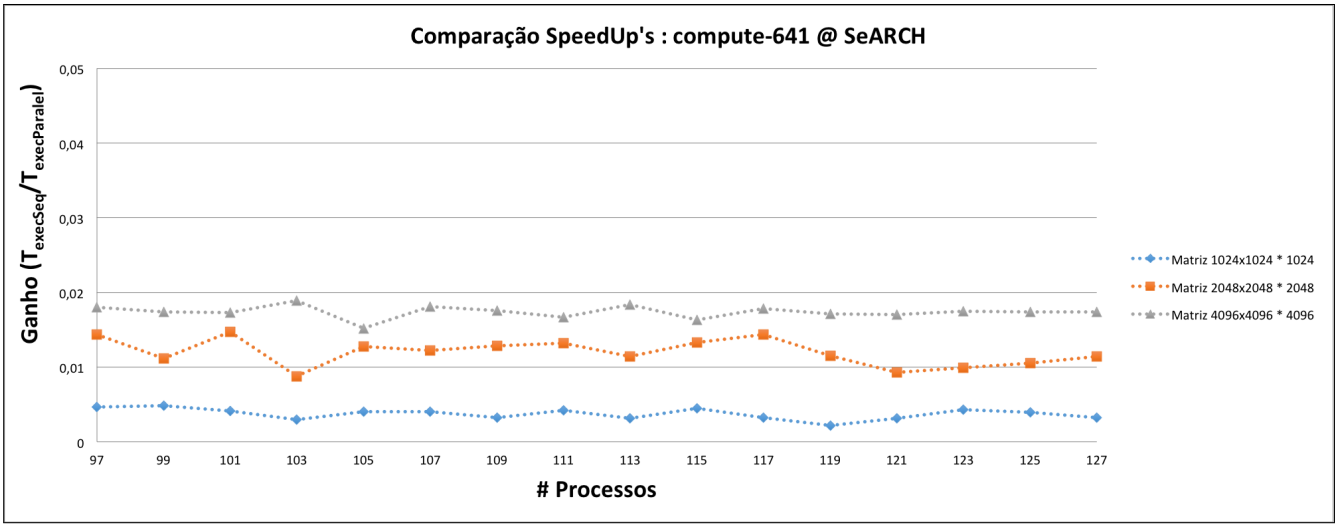


Figura 6: *Processos em 4 máquinas*

7 Comunicação

Neste trabalho, o algoritmo paralelo é um algoritmo com uma implementação MPI pelo que a fórmula do seu tempo de execução é obtida por:

$$T_{exec} = T_{Computation} + T_{Communication} + T_{Free} \tag{1}$$

No tempo de comunicação ($T_{Communication}$) está incluída a latência correspondente ao tempo gasto desde que é enviada informação aos processos até que estes iniciam a computação.

Pela experimentação que realizamos percebemos uma tendência crescente dos tempos gastos com sends e receives à medida que aumentamos o número de processos. Esses tempos podem ser consultados em anexo para os diferentes números de nodos a partir do anexo E.

Globalmente, os nossos ganhos estão completamente limitados pelo tempo gasto com a comunicação (como se pode verificar pelas figuras 8, 9 e 10) quer com o aumento do número de processos MPI quer com a expansão do número de nós 641 utilizados. O tempo de execução da versão MPI acaba por aumentar bastante e ser muito superior aos tempos da versão sequencial refletindo-se nos valores de ganhos apresentados nos gráficos da secção 6.1. Quando utilizamos mais que 32 processos, estamos a utilizar 2 nodos para realizar a computação (ver gráfico da Fig. 8), note-se a evidência do "peso"que a comunicação está a ter no nosso algoritmo. Os tempos de comunicação começam a subir drásticamente e mesmo com 4 nodos (Fig 10 para a matriz de maior dimensão (4096×4096) apresenta tempos cada vez maiores. Em resumo, quantos mais nodos utilizamos para realizar a computação, pior a performance do algoritmo do que executá-lo em apenas um nó.

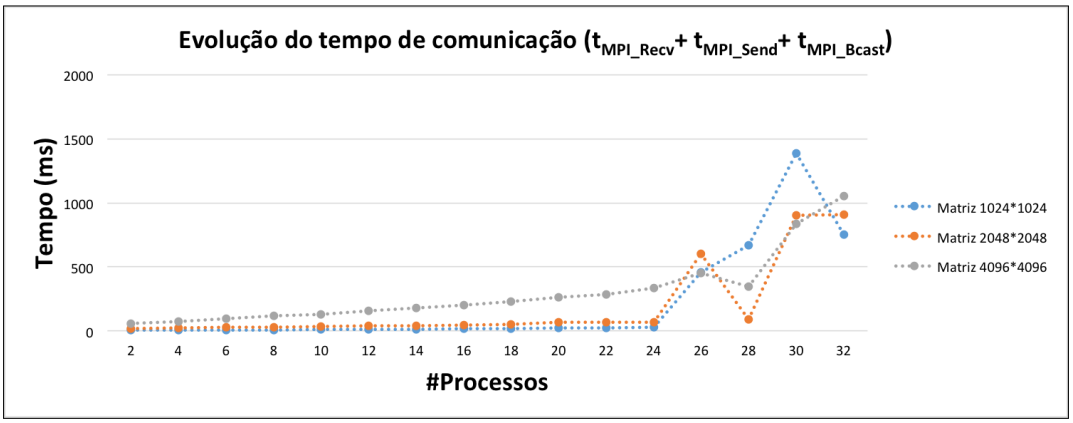


Figura 7: *Evolução dos tempos de comunicação com 1 só Nodo*

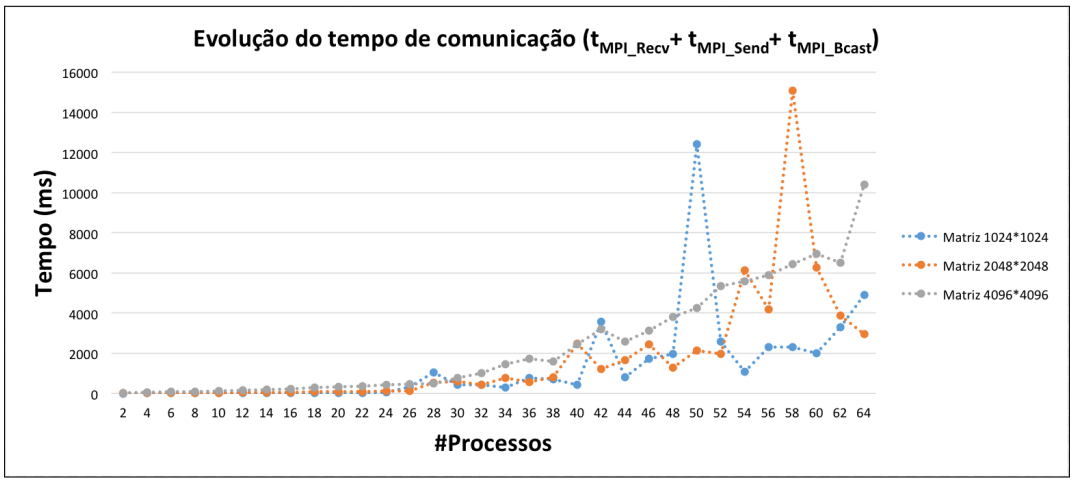


Figura 8: *Evolução dos tempos de comunicação com 2 Nodos*

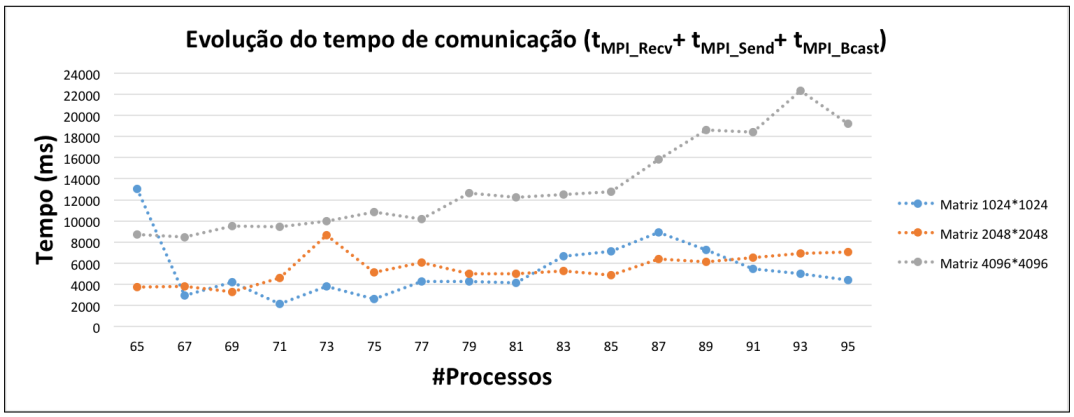


Figura 9: *Evolução dos tempos de comunicação com 3 Nodos*

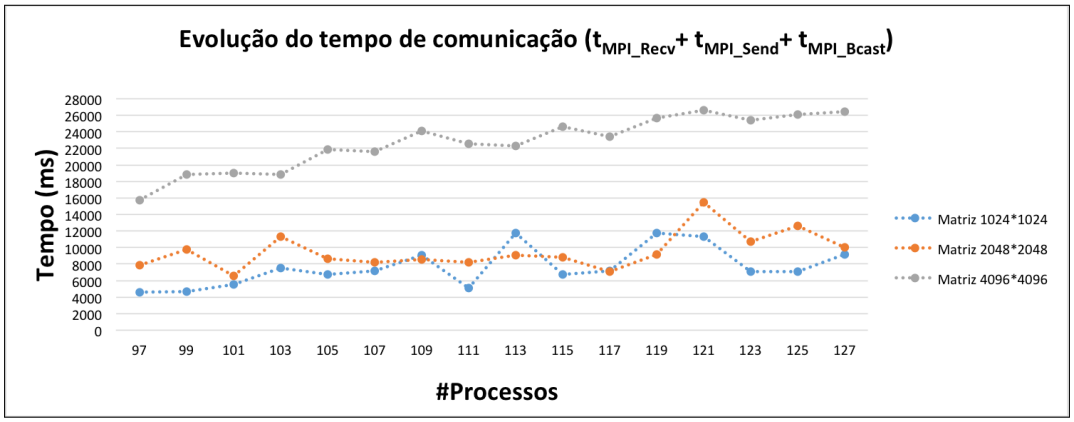


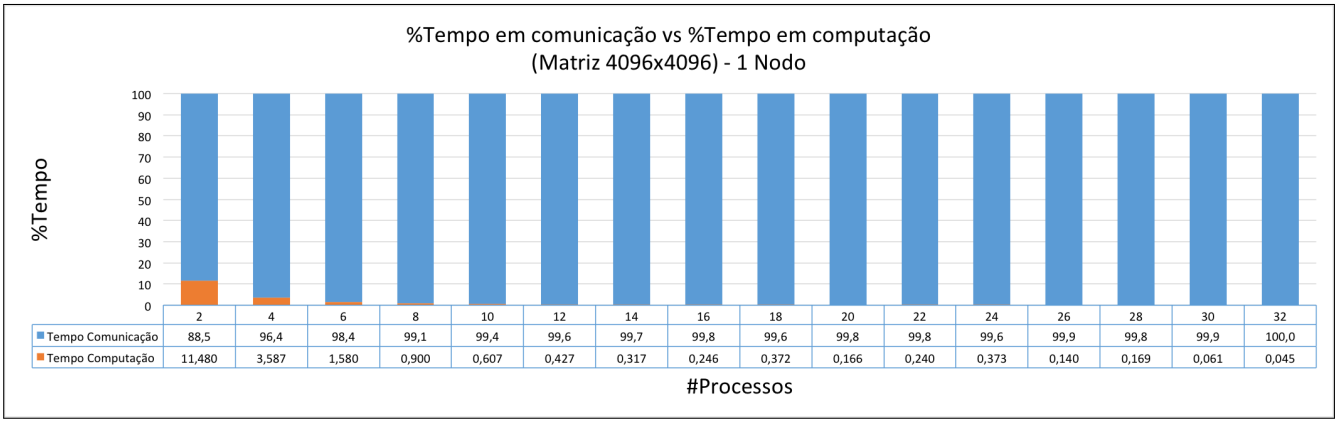
Figura 10: *Evolução dos tempos de comunicação com 4 Nodos*

8 Tempo de comunicação VS Tempo Computação

Ao analisar-se as figuras 11 12 e 13 conseguimos perceber facilmente que a maior parte do tempo total de execução de um algoritmo é gasta na comunicação, sendo que essa diferença se agrava, tal como era esperado, à medida que o número de processos e nodos aumenta. A comunicação feita entre nós recorre à rede Ethernet.

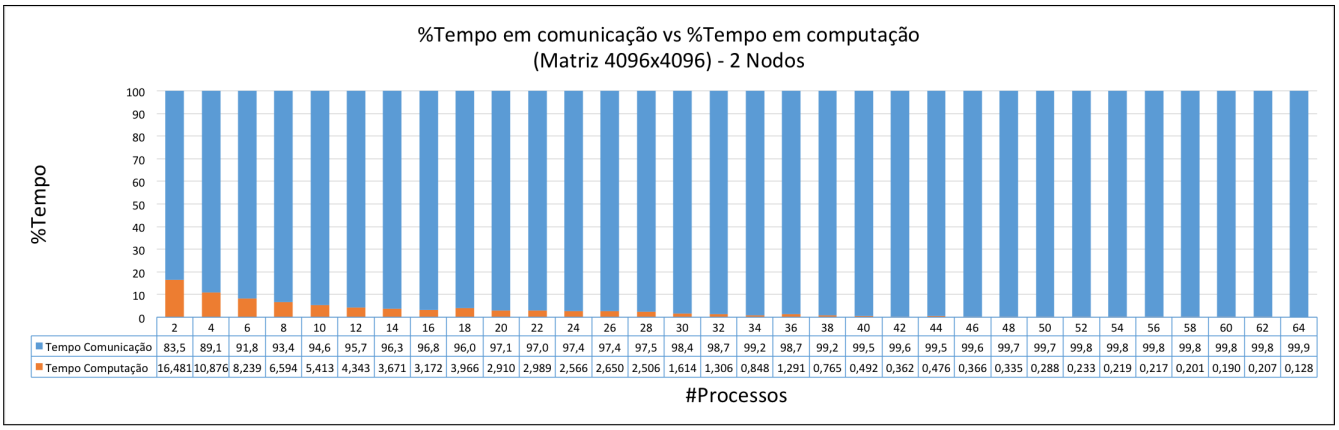
A partir de 4 nodos o tempo de computação é completamente insignificante em relação ao tempo efetivo de computação pelo que não vale a pena prolongar a análise para mais que 4 nodos.

Os restantes gráficos com a avaliação dos tempos de execução e de computação encontram-se no anexo I.



	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
Tempo Comunicação	88,5	96,4	98,4	99,1	99,4	99,6	99,7	99,8	99,6	99,8	99,8	99,6	99,9	99,8	99,9	100,0
Tempo Computação	11,480	3,587	1,580	0,900	0,607	0,427	0,317	0,246	0,372	0,166	0,240	0,373	0,140	0,169	0,061	0,045

Figura 11: *Evolução dos tempos de comunicação com 1 Nodo*



	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46	48	50	52	54	56	58	60	62	64
Tempo Comunicação	83,5	89,1	91,8	93,4	94,6	95,7	96,3	96,8	96,0	97,1	97,0	97,4	97,4	97,5	98,4	98,7	99,2	98,7	99,2	99,5	99,6	99,5	99,6	99,7	99,7	99,8	99,8	99,8	99,8	99,8	99,8	99,9
Tempo Computação	16,481	10,876	8,239	6,594	5,413	4,343	3,671	3,172	3,966	2,910	2,989	2,566	2,650	2,506	1,614	1,306	0,848	1,291	0,765	0,492	0,362	0,476	0,366	0,335	0,288	0,233	0,219	0,217	0,201	0,190	0,207	0,128

Figura 12: *Evolução dos tempos de comunicação com 2 Nodos*

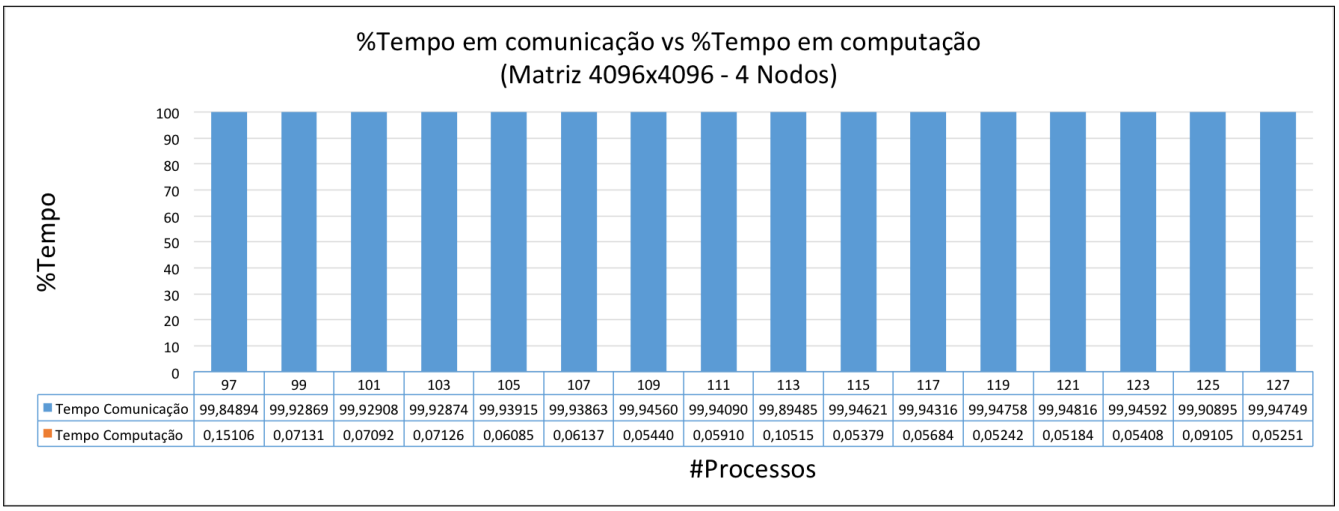


Figura 13: *Evolução dos tempos de comunicação com 4 Nodos*

9 Comparação dos ganhos entre implementação MPI vs OpenMP

Como se pode verificar ao longo da secção 6 a implementação OpenMP demonstrou ser uma solução mais eficiente a nível de desempenho para os data sets comuns em ambos os trabalhos. Pelo gráfico abaixo, podemos verificar uma análise comparativa dos ganhos obtidos em ambas as implementações comparativamente ao algoritmo sequencial utilizado como referência aos dois trabalhos:

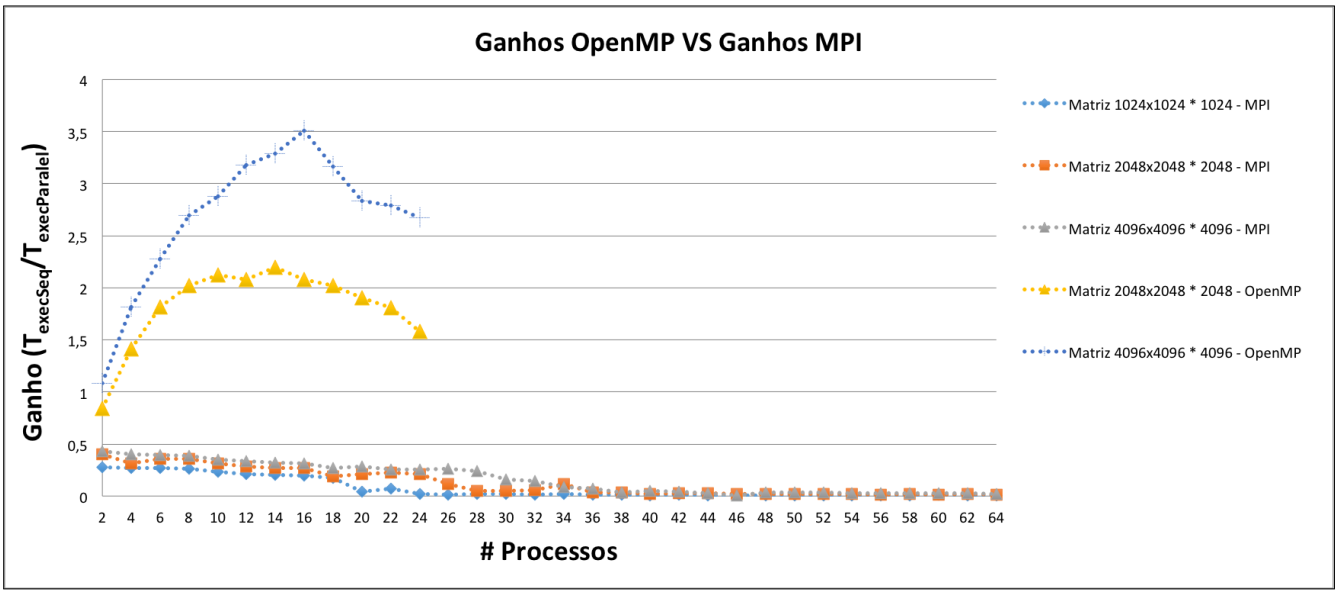


Figura 14: *Ganhos comparativos das duas implementações*

10 Conclusões do estudo

Pelo estudo feito verificamos que a implementação do algoritmo num paradigma de memória distribuída (**MPI**), para o número de nós e os data sets estudados é pior a nível de performance comparativamente à solução **OpenMP** realizada no trabalho anterior. Não conseguimos obter ganhos com a implementação **MPI** sobretudo por causa dos custos de comunicação envolvidos, pelo que percebemos do estudo.

Imaginar que um algoritmo se torna mais escalável pela ideia de distribuir a computação por vários nós pode ser uma falácia. Dependendo do tamanho dos data sets e da computação que o algoritmo realiza, os ganhos de performance poderão ser insignificantes ou até mesmo originar perdas acentuadas. Os tempos perdidos em comunicação entre processos e entre nodos podem-se, neste caso, sobrepor demasiado aos tempos em que efetivamente se realiza computação.

A execução do algoritmo MPI em apenas um nodo regista menos perdas de desempenho quando comparada com a execução para mais do que um nodo de acordo com as diferentes amostras retiradas.

Registamos algumas dificuldades quanto à medição dos tempos e na forma como estas medições deveriam ser realizadas, bem como em relação ao uso da rede Myrinet para análise de tempos.

Alguns detalhes adicionais acerca da implementação em si do algoritmo poderá ser encontrada no código em anexo a este relató
[1] [2]

Referências

[1] John L. Hennessy and David A. Patterson. *Computer Architecture, Fifth Edition: A Quantitative Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2011.

[2] David A. Patterson and John L. Hennessy. *Computer Organization and Design, Fourth Edition, Fourth Edition: The Hardware/Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design).* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 4th edition, 2008.

A Appendix

B Informações das Caches no Nodo 641

```
Memory Cache and TLB Hierarchy Information.
------------------------------------------------------------------------
TLB Information.
  There may be multiple descriptors for each level of TLB
  if multiple page sizes are supported.

L1 Data TLB:
  Page Size:              4 KB
  Number of Entries:     64
  Associativity:          4

L1 Instruction TLB:
  Page Size:              4 KB
  Number of Entries:     64
  Associativity:          4


Cache Information.

L1 Data Cache:
  Total size:            32 KB
  Line size:             64 B
  Number of Lines:      512
  Associativity:          8

L1 Instruction Cache:
  Total size:            32 KB
  Line size:             64 B
  Number of Lines:      512
  Associativity:          8

L2 Unified Cache:
  Total size:           256 KB
  Line size:             64 B
  Number of Lines:     4096
  Associativity:          8

L3 Unified Cache:
  Total size:         20480 KB
  Line size:             64 B
  Number of Lines:   327680
  Associativity:         20

mem_info.c                              PASSED
```

C CPU Info no nodo 641

```
---- RESULT of cat /proc/cpuinfo -----

processor        : 0
vendor_id        : GenuineIntel
cpu family        : 6
model               : 62
model name        : Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz
stepping        : 4
cpu MHz                : 1200.000
cache size        : 20480 KB
physical id        : 0
siblings        : 16
core id                : 0
cpu cores        : 8
apicid                : 0
initial apicid        : 0
fpu                : yes
fpu_exception        : yes
cpuid level        : 13
wp                : yes
flags                : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat pse36
clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc
arch_perfmon pebs bts rep_good xtopology nonstop_tsc aperfmperf pni pclmulqdq dtes64
monitor ds_cpl vmx smx est tm2 ssse3 cx16 xtpr pdcm dca sse4_1 sse4_2 x2apic popcnt
aes xsave avx f16c rdrand lahf_lm ida arat epb xsaveopt pln pts dts tpr_shadow vnmi
flexpriority ept vpid fsgsbase smep erms
bogomips        : 5199.77
clflush size        : 64
cache_alignment        : 64
address sizes        : 46 bits physical, 48 bits virtual
power management:

processor        : 1
vendor_id        : GenuineIntel
cpu family        : 6
model               : 62
model name        : Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz
stepping        : 4
cpu MHz                : 1200.000
cache size        : 20480 KB
physical id        : 0
siblings        : 16
core id                : 1
cpu cores        : 8
apicid                : 2
initial apicid        : 2
fpu                : yes
fpu_exception        : yes
cpuid level        : 13
```

```
wp               : yes
flags              : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat
pse36 clflush
dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon
pebs bts rep_good xtopology nonstop_tsc aperfmperf pni pclmulqdq dtes64 monitor ds_cpl
vmx smx est tm2 ssse3 cx16 xtpr pdcm dca sse4_1 sse4_2 x2apic popcnt aes xsave avx f16c
rdrand lahf_lm ida arat epb xsaveopt pln pts dts tpr_shadow vnmi flexpriority ept vpid
fsgsbase smep erms
bogomips        : 5199.77
clflush size       : 64
cache_alignment       : 64
address sizes       : 46 bits physical, 48 bits virtual
power management:
.
.
.
.
.
.
.
.
processor       : 31
vendor_id       : GenuineIntel
cpu family       : 6
model              : 62
model name       : Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz
stepping        : 4
cpu MHz              : 1200.000
cache size       : 20480 KB
physical id       : 1
siblings        : 16
core id              : 7
cpu cores       : 8
apicid              : 47
initial apicid       : 47
fpu              : yes
fpu_exception       : yes
cpuid level       : 13
wp               : yes
flags              : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat pse36 clflush
dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon
pebs bts rep_good xtopology nonstop_tsc aperfmperf pni pclmulqdq dtes64 monitor ds_cpl
vmx smx est tm2 ssse3 cx16 xtpr pdcm dca sse4_1 sse4_2 x2apic popcnt aes xsave avx f16c
rdrand lahf_lm ida arat epb xsaveopt pln pts dts tpr_shadow vnmi flexpriority ept vpid
fsgsbase smep erms
bogomips        : 5199.22
clflush size       : 64
cache_alignment       : 64
address sizes       : 46 bits physical, 48 bits virtual
power management:
```

D Informações da Memória no nodo 641

```
----- RESULT OF cat /proc/meminfo @ 641 Node -----

MemTotal:       66070988 kB
MemFree:        60148492 kB
Buffers:          302612 kB
Cached:          4030480 kB
SwapCached:          992 kB
Active:          4408048 kB
Inactive:         499648 kB
Active(anon):     567172 kB
Inactive(anon):     9928 kB
Active(file):    3840876 kB
Inactive(file):   489720 kB
Unevictable:       14708 kB
Mlocked:           14708 kB
SwapTotal:       1023992 kB
SwapFree:        1012752 kB
Dirty:             85896 kB
Writeback:             0 kB
AnonPages:        628948 kB
Mapped:            12900 kB
Shmem:                 4 kB
Slab:             232416 kB
SReclaimable:     171964 kB
SUnreclaim:        60452 kB
KernelStack:        6016 kB
PageTables:         4868 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:    34059484 kB
Committed_AS:   26820364 kB
VmallocTotal:   34359738367 kB
VmallocUsed:      402564 kB
VmallocChunk:   34323838972 kB
HardwareCorrupted:     0 kB
AnonHugePages:    591872 kB
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       2048 kB
DirectMap4k:        4540 kB
DirectMap2M:     2056192 kB
DirectMap1G:    65011712 kB
```

E Execução do algoritmo em 1 nodo

E.1 Tempos Matriz 1024x1024 Vetor 1024

Allocated computing node compute-641-5
Eth Network

Running for COO Matrix Size 1024 x1024
Running for Vector Size 1024
For each PPN we made 5 time measurements and each value presented on the following table are the median of them.
Times in milliseconds

PPN	Total Time BCast	Total Time MPI_Send	Total Time MPI_Recv	Computation Total Time	Slowest Process Time
2	0.061988830566	0.852108001709	3.793001174927	0.493049621582	4.284858703613
4	0.342130661011	0.940084457397	4.079103469849	0.166177749634	1.867055892944
6	0.089168548584	1.036882400513	4.333972930908	0.102996826172	1.344919204712
8	0.115871429443	1.129865646362	5.938768386841	0.076055526733	1.410961151123
10	0.113010406494	1.186370849609	7.402420043945	0.063180923462	1.446962356567
12	0.128030776978	1.326084136963	9.586334228516	0.050783157349	1.574039459229
14	0.143051147461	1.305818557739	11.240005493164	0.048398971558	1.554012298584
16	0.154972076416	1.404047012329	13.229846954346	0.037908554077	1.633882522583
18	0.200986862183	1.598596572876	17.309427261353	0.062704086304	1.870155334473
20	0.205993652344	1.691818237305	19.427776336670	0.065326690674	1.913070678711
22	0.216960906982	1.757860183716	22.996187210083	0.050783157349	2.024888992310
24	0.221014022827	1.935243606567	27.504205703735	0.057697296143	2.208948135376
26	0.226020812988	30.117273330688	427.457809448242	0.053405761719	40.210962295532
28	0.214099884033	39.900302886963	628.594160079956	0.052928924561	40.115118026733
30	4.564046859741	70.660829544067	1311.828851699829	0.044107437134	90.510129928589
32	5.784988403320	39.868116378784	705.554485321045	0.033617019653	40.156126022339

E.2 Tempos Matriz 2048x2048 Vetor 2048

Running for COO Matrix Size 2048 x2048
Running for Vector Size 2048
For each PPN we made 5 time measurements and each value presented on the following table are the median of them.
Times in milliseconds

PPN	Total Time BCast	Total Time MPI_Send	Total Time MPI_Recv	Computation Total Time	Slowest Process Time
2	0.411033630371	3.356933593750	13.994932174683	2.138853073120	16.133785247803
4	0.406980514526	3.293991088867	15.982151031494	0.656843185425	7.244110107422
6	0.164031982422	3.373146057129	21.878004074097	0.398874282837	6.642818450928
8	0.164985656738	3.155946731567	25.609731674194	0.255346298218	5.836963653564
10	0.164985656738	3.410339355469	30.483961105347	0.196695327759	5.857944488525
12	0.180006027222	3.646612167358	36.750793457031	0.163316726685	6.001949310303
14	0.190973281860	3.739118576050	33.683538436890	0.140905380249	4.812002182007
16	0.247001647949	3.890752792358	39.136409759521	0.128746032715	4.912853240967
18	0.262975692749	3.877162933350	46.239137649536	0.295162200928	4.956960678101
20	0.209808349609	4.729270935059	63.127517700195	0.189304351807	5.713939666748
22	0.231981277466	4.195690155029	60.373544692993	0.166893005371	5.143880844116
24	0.248193740845	4.208803176880	64.196348190308	0.152587890625	5.225896835327
26	0.236034393311	30.946731567383	572.946786880493	0.151395797729	31.864881515503
28	0.334978103638	4.585981369019	81.808328628540	0.163793563843	5.666017532349
30	0.282049179077	69.660902023315	833.823442459106	0.147581100464	70.654869079590
32	0.278949737549	50.733566284180	856.584548950195	0.129938125610	51.702976226807

E.3 Tempos Matriz 4096x4096 Vetor 4096

Running for COO Matrix Size 4096 x4096
Running for Vector Size 4096
For each PPN we made 5 time measurements and each value presented on the following table are the median of them.
Times in milliseconds

PPN	Total Time BCast	Total Time MPI_Send	Total Time MPI_Recv	Computation Total Time	Slowest Process Time
2	0.344991683960	11.039018630981	45.947074890137	7.435083389282	53.382158279419
4	0.393152236938	11.568069458008	62.547922134399	2.772092819214	28.634071350098
6	0.177860260010	11.942148208618	83.724021911621	1.538753509521	25.528907775879
8	0.213146209717	11.887311935425	102.375507354736	1.039505004883	23.648977279663
10	0.203132629395	12.620687484741	118.132114410400	0.800132751465	22.783041000366
12	0.220060348511	13.390302658081	141.453981399536	0.665664672852	23.062944412231
14	0.241041183472	13.719558715820	161.689043045044	0.558137893677	22.882938385010
16	0.267028808594	13.984918594360	184.121608734131	0.488519668579	22.929906845093
18	0.308990478516	14.117479324341	213.394403457642	0.850200653076	23.059129714966
20	0.310897827148	14.844179153442	246.621370315552	0.434160232544	23.707866668701
22	0.285863876343	14.828681945801	268.206596374512	0.681161880493	23.547887802124
24	0.336170196533	16.490221023560	315.324783325195	1.243352890015	25.238990783691
26	0.319957733154	21.250486373901	430.886030197144	0.636100769043	29.958963394165
28	0.395059585571	15.003919601440	331.309080123901	0.585317611694	23.643016815186
30	0.372886657715	41.453123092651	796.499013900757	0.508069992065	49.968004226685
32	0.315904617310	49.034833908081	1002.381086349487	0.468969345093	57.566881179810

F Execução do algoritmo em 2 nodos

F.1 Tempos Matriz 1024x1024 Vetor 1024

Allocated computing node compute-641-20
Eth Network

Running for COO Matrix Size 1024 x1024
Running for Vector Size 1024
For each PPN we made 5 time measurements and each value presented on the following table are the median of them
Times in milliseconds

PPN	Total Time BCast	Total Time MPI_Send	Total Time MPI_Recv	Computation Total Time	Slowest Process Time
2	0.061988830566	1.779794692993	1.132726669312	0.560998916626	1.784086227417
4	4.398107528687	1.791000366211	2.953052520752	0.548124313354	1.792192459106
6	0.092983245850	1.800775527954	4.756212234497	0.535726547241	1.802921295166
8	0.100851058960	1.857042312622	6.798267364502	0.526666641235	1.860141754150
10	0.112771987915	2.088069915771	8.996009826660	0.533342361450	2.092123031616
12	0.116825103760	2.289056777954	11.736869812012	0.520944595337	2.294063568115
14	0.132083892822	2.369880676270	14.303207397461	0.541925430298	2.374887466431
16	0.145196914673	2.501964569092	17.491579055786	0.554084777832	2.507925033569
18	0.164985656738	2.792835235596	23.419380187988	0.748634338379	2.796888351440
20	0.185966491699	2.868890762329	27.302742004395	0.800132751465	2.886056900024
22	0.217914581299	2.972126007080	32.126188278198	0.826597213745	2.986192703247
24	0.178098678589	3.392934799194	39.278745651245	0.917911529541	3.401994705200
26	0.177860260010	21.414995193481	305.404186248779	0.804424285889	21.435976028442
28	6.233215332031	60.057878494263	968.158245086670	0.783443450928	60.076951980591
30	0.216960906982	21.537065505981	417.823553085327	0.901937484741	21.543979644775
32	10.972023010254	20.661830902100	396.794557571411	0.922203063965	20.670890808105
34	0.275135040283	19.742965698242	281.175851821899	0.858783721924	21.684169769287
36	0.373125076294	12.304067611694	754.320383071899	0.834226608276	35.723924636841
38	0.278949737549	23.336887359619	671.180009841919	0.834941864014	36.965131759644
40	0.279903411865	22.498130798340	418.623447418213	0.773429870605	27.585983276367
42	0.257968902588	96.499919891357	3488.883972167969	0.792741775513	119.367837905884
44	0.264883041382	27.497053146362	779.661655426025	0.777006149292	31.333923339844
46	0.302076339722	67.241191864014	1661.301374435425	0.794172286987	80.328941345215
48	0.281095504761	49.320936203003	1916.205167770386	0.829935073853	62.633991241455
50	1.537084579468	353.979825973511	12062.981843948364	10.821104049683	357.506036758423
52	0.262022018433	71.553945541382	2514.448642730713	0.805854797363	83.569049835205
54	0.260114669800	33.092975616455	1053.465127944946	0.867366790771	35.936832427979
56	0.262975692749	64.245939254761	2258.786916732788	0.901699066162	66.045045852661
58	0.270128250122	61.468839645386	2240.918159484863	0.826120376587	80.538034439087
60	0.286102294922	45.231103897095	1960.934162139893	0.856161117554	57.585000991821
62	0.252962112427	88.234186172485	3209.987163543701	0.885486602783	90.759992599487
64	0.253915786743	108.519077301025	4782.235145568848	0.854969024658	128.091812133789

F.2 Tempos Matriz 2048x2048 Vetor 2048

Running for COO Matrix Size 2048x2048
Running for Vector Size 2048
For each PPN we made 5 time measurements and each value presented on the following table are the median of them
Times in milliseconds

PPN	Total Time BCast	Total Time MPI_Send	Total Time MPI_Recv	Computation Total Time	Slowest Process Time
2	0.293016433716	4.986047744751	3.174543380737	1.646757125854	4.988908767700
4	0.408887863159	6.361961364746	10.450363159180	2.100944519043	6.364107131958
6	0.149011611938	5.565166473389	14.817237854004	1.804351806641	5.568027496338
8	0.149011611938	5.594015121460	20.539522171021	1.810789108276	5.597829818726
10	0.170946121216	6.396055221558	27.167797088623	1.852273941040	6.402015686035
12	0.172853469849	7.028818130493	35.505533218384	1.894474029541	7.030963897705
14	0.186920166016	7.386922836304	43.677806854248	1.928091049194	7.392883300781
16	0.202894210815	7.436037063599	52.120685577393	1.829862594604	7.444143295288
18	0.237941741943	8.118152618408	69.486856460571	2.937793731689	8.129119873047
20	0.241041183472	8.766889572144	83.539009094238	3.089904785156	8.795022964478
22	0.269889831543	8.713006973267	91.519355773926	2.829551696777	8.731842041016
24	0.296115875244	9.663105010986	116.808891296387	3.089189529419	9.665012359619
26	0.347137451172	9.281158447266	115.749835968018	3.039360046387	9.319067001343
28	0.318050384521	43.529033660889	497.282505035400	3.135681152344	43.570995330811
30	15.689134597778	41.197061538696	536.156415939331	3.203153610229	41.227102279663
32	0.293016433716	14.352083206177	414.090156555176	3.338098526001	21.240949630737
34	0.389099121094	41.284084320068	724.958181381226	3.265380859375	42.920827865601
36	0.382900238037	47.945976257324	508.975267410278	3.236770629883	49.554824829102
38	0.404119491577	54.701805114746	767.753839492798	3.187894821167	54.725885391235
40	0.395059585571	110.469818115234	2379.361152648926	2.961635589600	111.940860748291
42	0.332117080688	81.281900405884	1133.440494537354	3.008842468262	82.725048065186
44	0.363111495972	88.250875473022	1566.877841949463	3.065347671509	88.294982910156
46	0.335931777954	103.816986083984	2336.824417114258	3.065586090088	113.602876663208
48	0.365018844604	69.975852966309	1229.619741439819	3.091096878052	80.765962600708
50	0.372886657715	103.149890899658	2038.452625274658	3.129005432129	103.217840194702
52	0.347137451172	83.549022674561	1879.071712493896	3.232240676880	86.519002914429
54	0.353097915649	200.598001480103	5939.272642135620	3.164291381836	203.438997268677
56	0.366926193237	131.736040115356	4042.609214782715	3.196716308594	134.731769561768
58	0.339031219482	360.204935073853	14734.651327133179	3.197908401489	383.588075637817
60	0.324010848999	166.877031326294	6091.384172439575	3.263235092163	178.941965103149
62	0.394821166992	114.066839218140	3764.615058898926	3.481626510620	128.888130187988
64	0.354051589966	114.495038986206	2842.991113662720	3.425598144531	116.235017776489

F.3 Tempos Matriz 4096x4096 Vetor 4096

Running for COO Matrix Size 4096 x4096
Running for Vector Size 4096
For each PPN we made 5 time measurements and each value presented on the following table are the median of them.
Times in milliseconds

PPN	Total Time BCast	Total Time MPI_Send	Total Time MPI_Recv	Computation Total Time	Slowest Process Time
2	0.355958938599	19.949913024902	13.039588928223	6.580352783203	19.953012466431
4	0.379085540771	21.484136581421	35.650014877319	7.018804550171	21.487951278687
6	0.188827514648	21.976947784424	58.525085449219	7.245302200317	21.981000900269
8	0.190973281860	22.034168243408	80.718994140625	7.267475128174	22.037982940674
10	0.195026397705	24.647951126099	106.330871582031	7.506370544434	24.652004241943
12	0.212907791138	25.750875473022	134.520053863525	7.286787033081	25.761842727661
14	0.217914581299	26.647090911865	163.280487060547	7.246732711792	26.650905609131
16	0.266075134277	27.595996856689	194.539308547974	7.286787033081	27.604103088379
18	0.287055969238	30.605077743530	263.509511947632	12.159347534180	30.667066574097
20	0.272989273071	30.502080917358	292.422294616699	9.686231613159	30.543088912964
22	0.283956527710	31.299829483032	333.662271499634	11.255264282227	31.306028366089
24	0.306129455566	34.980773925781	406.878232955933	11.646032333374	34.986972808838
26	0.375986099243	33.143043518066	420.605421066284	12.360334396362	33.189058303833
28	0.349998474121	34.938097000122	481.041908264160	13.271808624268	35.005092620850
30	0.380992889404	48.855066299438	738.741874694824	12.929677963257	48.882007598877
32	0.360965728760	60.589075088501	945.168972015381	13.309717178345	60.604095458984
34	0.407934188843	93.379020690918	1378.167390823364	12.596368789673	104.851007461548
36	0.424146652222	129.729032516479	1584.848165512085	22.425651550293	131.329059600830
38	0.380992889404	169.363021850586	1409.446716308594	12.166976928711	169.451951980591
40	0.430107116699	177.942991256714	2255.455017089844	12.023210525513	188.596963882446
42	0.410079956055	224.223136901855	2958.135366439819	11.560440063477	224.350214004517
44	0.423908233643	206.916809082031	2371.820211410522	12.325286865234	207.711935043335
46	0.388145446777	268.742084503174	2861.332893371582	11.508464813232	268.748044967651
48	0.406026840210	257.081985473633	3540.627479553223	12.779951095581	258.454799652100
50	0.447034835815	262.921094894409	3985.473394393921	12.274265289307	265.166044235229
52	0.375986099243	294.371128082275	5051.014184951782	12.459278106689	294.428110122681
54	0.463962554932	286.585092544556	5310.983657836914	12.276411056519	297.724008560181
56	0.380992889404	326.869964599609	5574.606657028198	12.848377227783	328.881978988647
58	0.411033630371	297.875165939331	6133.120298385620	12.966394424438	315.288066864014
60	0.391006469727	330.834865570068	6611.618995666504	13.242959976196	332.461118698120
62	0.380039215088	322.861909866333	6197.727203369141	13.503313064575	322.895050048828
64	0.391960144043	424.060106277466	9992.985248565674	13.352394104004	429.445981979370

G Execução do algoritmo em 3 nodos

G.1 Tempos Matriz 1024x1024 Vetor 1024

Allocated computing node compute-641-7
Eth Network

Running for COO Matrix Size 1024 x1024
Running for Vector Size 1024
For each PPN we made 5 time measurements and each value presented on the following table are the median of them
Times in milliseconds

PPN	Total Time Bcast	Total Time MPI_Send	Total Time MPI_Recv	Computation Total Time	Slowest Process Time
65	0.059843063354	250.689983367920	12776.348114013672	0.919818878174	257.061958312988
67	0.061035156250	78.516006469727	2873.333215713501	0.870227813721	81.199169158936
69	0.051975250244	100.486040115356	4085.913658142090	0.864505767822	105.410099029541
71	0.059127807617	64.931869506836	2064.159631729126	0.888347625732	78.923940658569
73	0.056028366089	80.060005187988	3683.017492294312	0.865936279297	89.170932769775
75	0.052928924561	65.239906311035	2512.350797653198	0.889539718628	66.908836364746
77	0.048875808716	134.176969528198	4153.601408004761	0.887155532837	134.207963943481
79	0.056028366089	91.809988021851	4139.129400253296	0.870466232300	102.940797805786
81	0.056028366089	93.111991882324	4002.141952514648	0.868082046509	94.732999801636
83	0.055074691772	116.336107254028	6514.759302139282	0.854969024658	121.294021606445
85	0.051975250244	129.518985748291	6965.868949890137	0.853061676025	131.057977676392
87	0.045776367188	161.128997802734	8735.558032989502	0.851631164551	167.521953582764
89	0.052213668823	129.889011383057	7128.358840942383	0.840187072754	139.951944351196
91	0.055074691772	92.207193374634	5340.785980224609	0.871181488037	97.311019897461
93	0.056982040405	102.379083633423	4865.377664566040	0.908613204956	103.668928146362
95	0.052928924561	101.705074310303	4268.366813659668	0.896930694580	103.14297676086

G.2 Tempos Matriz 2048x2048 Vetor 2048

Running for COO Matrix Size 2048 x2048
Running for Vector Size 2048
For each PPN we made 5 time measurements and each value presented on the following table are the median of them
Times in milliseconds

PPN	Total Time Bcast	Total Time MPI_Send	Total Time MPI_Recv	Computation Total Time	Slowest Process Time
65	0.123977661133	124.977111816406	3586.498498916626	13.417005538940	127.208948135376
67	0.090837478638	133.406162261963	3660.363674163818	3.338336944580	135.254859924316
69	0.086784362793	106.200933456421	3171.983957290649	3.279685974121	111.781835556030
71	0.089168548584	156.651020050049	4460.453987121582	13.555526733398	158.190965652466
73	0.088930130005	175.380945205688	8467.160940170288	3.304719924927	189.242124557495
75	0.088930130005	108.108043670654	5040.977716445923	3.243207931519	136.329889297485
77	0.090122222900	130.247116088867	5940.706014633179	3.234148025513	150.110006332397
79	0.085115432739	144.636154174805	4844.830751419067	3.399133682251	152.737855911255
81	0.091075897217	109.063863754272	4915.538787841797	3.401994705200	130.547046661377
83	0.088930130005	133.177042007446	5137.027025222778	3.312826156616	144.742012023926
85	0.094890594482	124.258995056152	4710.347652435303	3.364086151123	132.502079010010
87	0.081777572632	131.803989410400	6264.083147048950	3.315448760986	152.828931808472
89	0.093936920166	121.399164199829	5990.556716918945	3.418922424316	146.409034729004
91	0.088214874268	134.404897689819	6384.142398834229	3.486871719360	140.481948852539
93	0.106096267700	139.724016189575	6808.960199356079	3.575086593628	152.487039566040
95	0.088930130005	141.042947769165	6937.891960144043	3.592491149902	168.083190917969

G.3 Tempos Matriz 4096x4096 Vetor 4096

Running for COO Matrix Size 4096 x 4096
Running for Vector Size,4096
For each PPN we made 5 time measurements,and each value presented on the following table are the median of them
Times in milliseconds

PPN	Total Time Bcast	Total Time MPI_Send	Total Time MPI_Recv	Computation Total Time	Slowest Process Time
65	0.099182128906	367.226839065552	8332.094430923462	13.645887374878	371.241807937622
67	0.092983245850	352.380990982056	8122.689247131348	13.308048248291	362.955093383789
69	0.104904174805	380.861997604370	9155.163764953613	13.092994689941	385.777950286865
71	0.101089477539	378.108024597168	9047.718048095703	12.921571731567	380.222797393799
73	0.122070312500	386.708974838257	9604.692935943604	13.059616088867	388.176202774048
75	0.100135803223	399.849891662598	10410.023212432861	12.822389602661	399.939775466919
77	0.100135803223	375.468969345093	9796.042442321777	13.197660446167	376.592874526978
79	0.138044357300	415.529012680054	12232.102155685425	13.019800186157	415.676116943359
81	0.136137008667	401.165008544922	11830.360174179077	12.916564941406	406.935930252075
83	0.117778778076	409.304141998291	12124.305963516235	13.283729553223	411.934137344360
85	0.102996826172	428.217172622681	12358.486175537109	13.238191604614	428.436040878296
87	0.118970870972	463.840961456299	15335.238933563232	13.474225997925	469.477891921997
89	0.095129013062	497.720956802368	18090.077638626099	13.242959976196	504.014968872070
91	0.136137008667	492.412090301514	17944.641828536987	13.498067855835	492.588043212891
93	0.112056732178	549.170017242432	21762.288570404053	13.453245162964	571.171045303345
95	0.112056732178	502.207994461060	18736.204624176025	13.500690460205	503.803968429565

H Execução do algoritmo em 4 nodos

H.1 Tempos Matriz 1024x1024 Vetor 1024

Allocated computing node: compute-641-7
Eth Network

Running for COO Matrix Size: 1024 x 1024
Running for Vector Size : 1024
For each PPN we made 5 time measurements and each value presented on the following table are the median of them
Times in milliseconds

PPN	Total Time Bcast	Total Time MPI_Send	Total Time MPI_Recv	Computation Total Time	Slowest Process Time
97	0.056982040405	105.009078979492	4496.447563171387	0.929832458496	105.693101882935
99	0.050067901611	82.859992980957	4560.762405395508	0.890254974365	100.622892379761
101	0.056028366089	115.614891052246	5412.932872772217	0.924587249756	117.050886154175
103	0.054121017456	163.625955581665	7323.432922363281	0.881195068359	164.983987808228
105	0.058174133301	114.208936691284	6626.953601837158	0.881671905518	122.202157974243
107	0.056982040405	129.543781280518	7004.779815673828	0.890731811523	130.808115005493
109	0.059127807617	142.842769622803	8948.597192764282	0.854015350342	150.887966156006
111	0.077009201050	107.290983200073	5005.486011505127	0.879287719727	116.846084594727
113	0.050783157349	151.366949081421	11636.203050613403	0.880956649780	155.488967895508
115	0.054836273193	108.670949935913	6670.361042022705	0.884532928467	109.731197357178
117	0.054836273193	138.946056365967	7000.839948654175	0.864028930664	149.060010910034
119	0.051021575928	190.441131591797	11553.605079650879	0.873327255249	223.327875137329
121	0.054836273193	144.404172897339	11190.583705902100	0.904083251953	154.623031616211
123	0.051975250244	102.272033691406	6994.893312454224	0.917911529541	112.536191940308
125	0.051021575928	123.235940933228	6974.120140075684	0.897884368896	124.397993087769
127	0.055074691772	140.475988388062	9048.631668090820	0.908613204956	151.077985763550

H.2 Tempos Matriz 2048x2048 Vetor 2048

Running for COO Matrix Size: 2048 x 2048
Running for Vector Size : 2048
For each PPN we made 5 time measurements and each value presented on the following table are the median of them
Times in milliseconds

PPN	Total Time Bcast	Total Time MPI_Send	Total Time MPI_Recv	Computation Total Time	Slow
97	0.077962875366	137.064933776855	7715.757131576538	3.483533859253	1
99	0.113010406494	153.950929641724	9648.900270462036	3.489017486572	1
101	0.093936920166	127.979993820190	6398.948431015015	3.507375717163	1
103	0.087976455688	217.666149139404	11073.243141174316	3.358602523804	2
105	0.087022781372	146.350860595703	8467.471122741699	3.470420837402	1
107	0.081062316895	152.539014816284	8034.318685531616	3.397464752197	1
109	0.095129013062	141.868829727173	8417.564868927002	3.546953201294	1
111	0.108003616333	139.986991882324	8065.750122070312	3.489494323730	1
113	0.092983245850	166.033029556274	8901.688098907471	3.525733947754	1
115	0.099182128906	140.965938568115	8688.907384872437	3.479242324829	1
117	2.213954925537	125.116109848022	6992.910146713257	3.543376922607	1
119	0.091075897217	165.457010269165	9025.014400482178	3.453493118286	1
121	0.098943710327	206.755161285400	15302.939176559448	3.546714782715	2
123	0.084161758423	185.371875762939	10537.323236465454	3.475189208984	2
125	0.094890594482	177.642107009888	12472.734689712524	3.582715988159	1
127	0.077009201050	169.754981994629	9838.453054428101	3.605127334595	1

H.3 Tempos Matriz 4096x4096 Vetor 4096

Running for COO Matrix Size: 4096 x 4096
Running for Vector Size : 4096
For each PPN we made 5 time measurements and each value presented on the following table are the median of them
Times in milliseconds

PPN	Total Time Bcast	Total Time MPI_Send	Total Time MPI_Recv	Computation Total Time	Slowest Process Time
97	0.097036361694	435.060977935791	15318.155050277710	23.833036422729	437.253952026367
99	0.118970870972	485.374927520752	18353.632926940918	13.443231582642	497.411012649536
101	0.138998031616	497.081995010376	18511.007070541382	13.490200042725	499.032974243164
103	0.100851058960	452.446937561035	18421.542167663574	13.459444046021	456.515073776245
105	0.111103057861	556.113004684448	21331.169605255127	13.326883316040	567.581892013550
107	0.107049942017	475.308895111084	21119.243860244751	13.260126113892	477.134943008423
109	0.121116638184	489.038944244385	23645.787000656128	13.135910034180	490.941047668457
111	0.103950500488	518.543004989624	22019.657850265503	13.327121734619	518.636941909790
113	0.113964080811	462.255954742432	21808.678150177002	23.442506790161	468.751907348633
115	0.103950500488	521.299839019775	24083.334922790527	13.242006301880	529.546022415161
117	0.106096267700	480.462074279785	22965.704679489136	13.334989547729	482.819080352783
119	0.129938125610	491.424083709717	25142.525672912598	13.445138931274	503.305912017822
121	0.102043151855	500.808954238892	26086.586713790894	13.789415359497	506.889104843140
123	0.097036361694	491.340875625610	24920.009613037109	13.750553131104	493.556022644043
125	0.116109848022	488.961935043335	25611.621856689453	23.785829544067	497.032880783081
127	0.103950500488	493.077039718628	25956.810474395752	13.896465301514	495.365858078003

I %Tempo de Comunicação VS %Tempo Computação

- **2 Nodos**

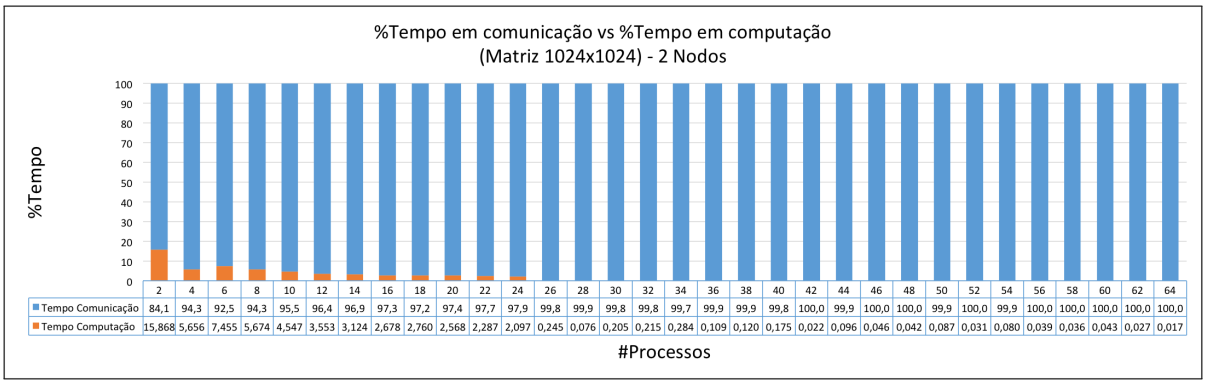


Figura 15: *Matriz e vetor dimensão 1024*

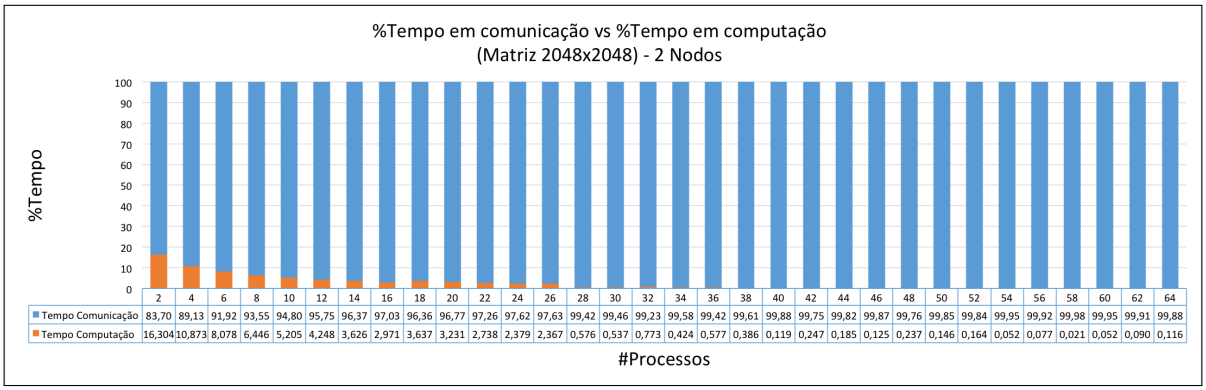


Figura 16: *Matriz e vetor dimensão 2048*

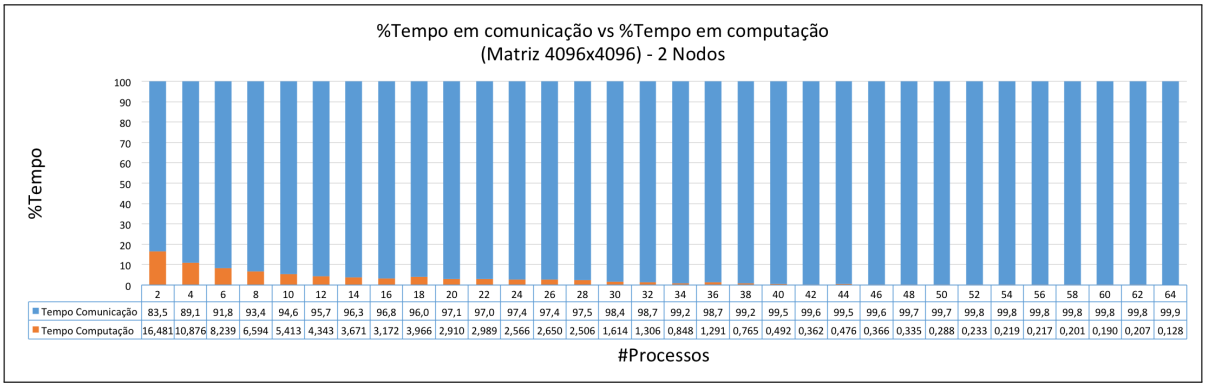


Figura 17: *Matriz e vetor dimensão 4096*

- **4 Nodos**

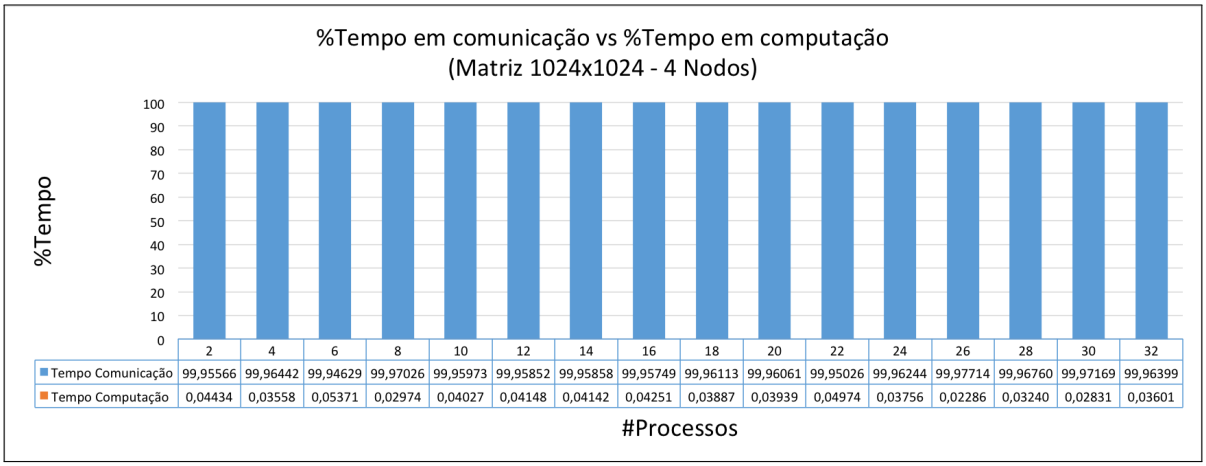
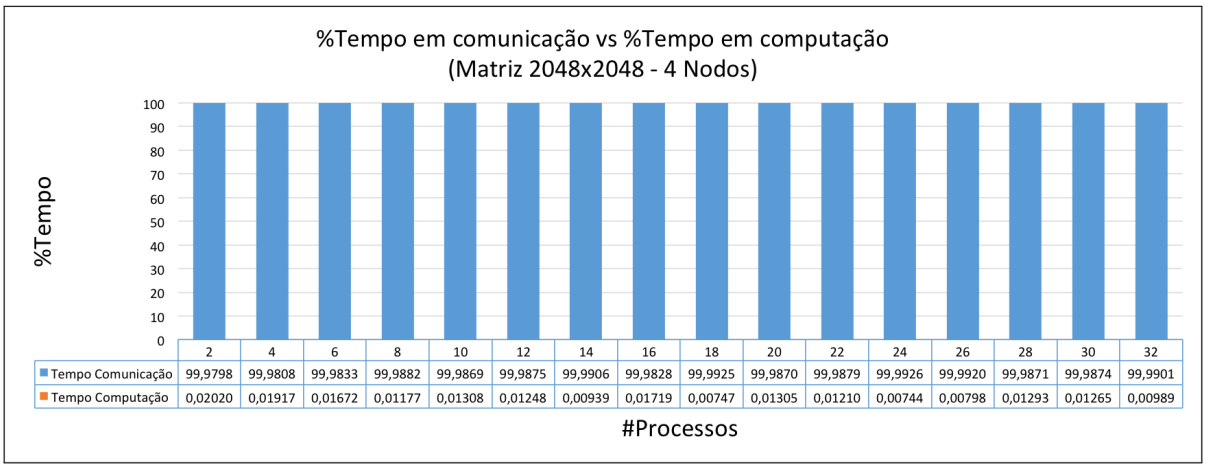
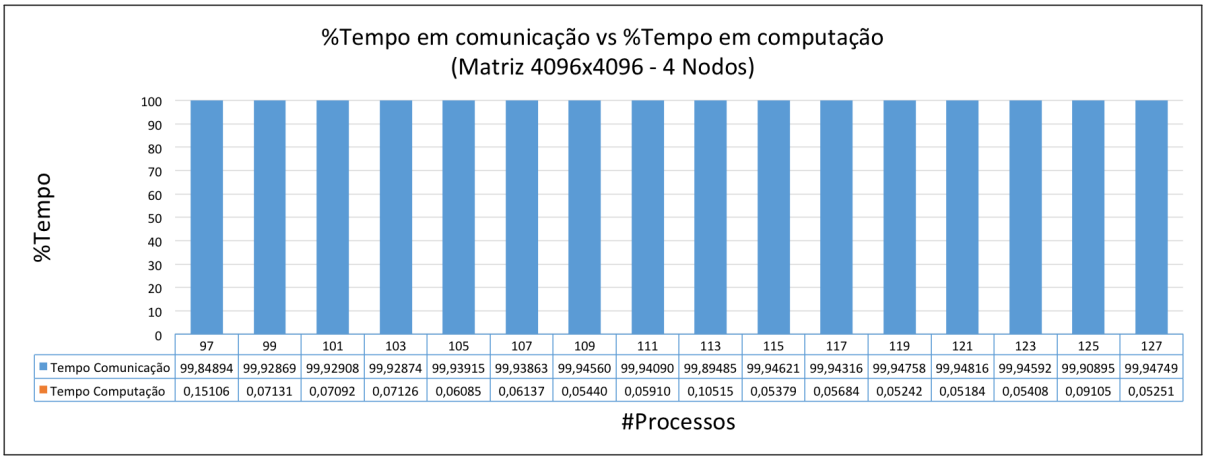


Figura 18: *Matriz e vetor dimensão 1024*



	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32
Tempo Comunicação	99,9798	99,9808	99,9833	99,9882	99,9869	99,9875	99,9906	99,9828	99,9925	99,9870	99,9879	99,9926	99,9920	99,9871	99,9874	99,9901
Tempo Computação	0,02020	0,01917	0,01672	0,01177	0,01308	0,01248	0,00939	0,01719	0,00747	0,01305	0,01210	0,00744	0,00798	0,01293	0,01265	0,00989

Figura 19: *Matriz e vetor dimensão 1024*



	97	99	101	103	105	107	109	111	113	115	117	119	121	123	125	127
Tempo Comunicação	99,84894	99,92869	99,92908	99,92874	99,93915	99,93863	99,94560	99,94090	99,89485	99,94621	99,94316	99,94758	99,94816	99,94592	99,90895	99,94749
Tempo Computação	0,15106	0,07131	0,07092	0,07126	0,06085	0,06137	0,05440	0,05910	0,10515	0,05379	0,05684	0,05242	0,05184	0,05408	0,09105	0,05251

Figura 20: *Matriz e vetor dimensão 1024*