Universidade do Minho
Escola de Engenharia

Universidade do Minho
Mestrado Integrado Engenharia Informática

**Paradigmas de Computação Paralela**

---

# Simulação do Processo de Difusão de Calor

**Paralelismo em Open MPI ®**

---

**Grupo 3:**
**João Lopes** **A61077**
**Nuno Moreira** **A61017**

Braga, 18 de Dezembro de 2016

# Conteúdo

# 1. Caso de Estudo

O caso de estudo desenvolvido vai em seguimento do caso de estudo desenvolvido no primeiro trabalho prático, onde, em ambos os trabalhos tem-se como objetivo paralelizar a computação do processo de difusão de calor em N_MAX iterações. A grande diferença entre ambos os trabalhos assenta no modelo de memória utilizado para efetuar a paralelização. No primeiro trabalho, o algoritmo sequencial desenvolvido foi adaptado para recorrer a diretivas OpenMP num ambiente de memória partilhada, enquanto que no presente trabalho, o mesmo código sequencial foi alterado por forma a que todos os processos recorram a memória distribuída em diretivas **OpenMPI**.

O processo de difusão de calor é efetuado sobre uma superfície que é representada por uma matriz quadrada. Cada posição da matriz possui um valor numérico representativo da temperatura nessa posição. O intervalo de valores possíveis, para os valores de temperatura de cada posição, foi de 0 a 100, onde 0 representa o valor de temperatura mínimo (frio) e 100 representa o valor de temperatura máximo (quente).

Foi considerado ainda que a superfície é aquecida apenas no seu lado superior, ou seja, apenas os valores da primeira linha da matriz sofrem o aquecimento inicial cujo valor é 100. As restantes posições da matriz são todas iniciadas a temperatura 0. Foi ainda considerado que as bordas da matriz permanecem imutáveis ao longo do processo de difusão de calor.

O calculo do valor de temperatura para cada posição da matriz é efetuado recorrendo à fórmula:

$$M\_New[i][j] = \frac{M\_Old[i-1][j] + M\_Old[i+1][j] + M\_Old[i][j-1] + M\_Old[i][j+1] + M\_Old[i][j]}{5} \quad (1)$$

Atendendo à formula utilizada, é possível observar que é utilizada outra matriz de auxílio (M_Old) que possui os valores de temperatura da ultima iteração e M_New refere-se à matriz inicial representativa da superfície.

# 2. Caracterização das Plataformas de Teste

A plataforma de testes utilizada no Search6 recorre a nodos equipados com dual-socket. Foram utilizados os nodos de computação 641 e 662 cujas características principais podem ser consultadas no anexo A. Ambos os nodos possuem ainda duas formas de comunicação entre nodos:

- Comunicação entre nodos por rede Ethernet - 1Gbps
- Comunicação entre nodos por rede Myrinet - 10Gbps

## 2.1. Compilador, Flags de Compilação e Ferramentas de Medição Utilizadas

Por forma a gerar todos os executáveis foi utilizado o compilador da gnu - **gcc 4.9.3** com as flags de compilação:

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

A execução do programa para os vários processos utilizados e para os vários tamanhos de matriz considerados está dependente do tipo de comunicação entre nodos utilizada:

- Para comunicação Ethernet:
  Foi utilizada a biblioteca **openmpi_eth/1.8.4**

```
$ mpirun -np $ppn --map-by core -mca btl self,sm,tcp --report-bindings ↩
    ./bin/heatDiff_MPI $matrix_size $n_max
```

- Para comunicação Myrinet:
  Foi utilizada a biblioteca **openmpi_mx/1.8.2**

```
$ mpirun -np $ppn --hostfile hosts_file_662 --report-bindings --mca ↩
    mtl mx --mca pml cm ./bin/heatDiff_MPI $matrix_size $n_max
```

As flags utilizadas no comando *mpirun* definem o numero de processos a utilizar fazendo o mapeamento dos mesmos nos cores disponíveis, permitem a utilização de comunicação por Ethernet e Myrinet e geram ainda uma representação visual do mapeamento dos processos nos cores dos sockets da máquina utilizada (–map-by core).

## 2.2. Tamanhos de Input

Tal como no trabalho anterior, foi necessário tomar decisões acerca do tamanho de input das matrizes a utilizar e ainda do numero máximo de iterações que vão ser efetuadas até que o processo de difusão de calor esteja concluído.

Os tamanhos de matriz escolhidos são iguais aos tamanhos do trabalho anterior: **1024*1024**, **2048*2048**, **4096*4096** e **8192*8192** de **floats**. Foram escolhidos estes valores uma vez que para o primeiro valor, as matrizes (M_New e M_Old) conseguem ser completamente inseridas na memória cache L3 (20MB cache é superior a 8MB do tamanho aproximado das matrizes). O mesmo não acontece para as restantes matrizes cujos tamanhos aproximados de 32MB, 134MB e 536MB são muito superiores ao tamanho da cache L3, levando a que ocorra misses nos vários níveis de cache e consequentes chamadas à memória RAM.

Em relação ao numero de iterações que vão ser realizadas, no trabalho de OpenMP foram escolhidos tamanhos de 1000, 2000, 4000 e 8000 iterações. Contudo, com o objetivo de reduzir ao numero de testes a efetuar no presente trabalho e tendo em conta que uma das conclusões retiradas do trabalho de OpenMP em relação ao numero de iterações efetuadas foi que não influenciava significativamente o ganho (da versão paralela em relação à versão sequencial) - à medida que o numero de iterações aumentava, os ganhos de cada matriz não sofriam aumentos nem reduções significativas de forma geral - foi decidido que para o presente trabalho o numero máximo de iterações a utilizar é o de maior esforço computacional - **8000 iterações**.

# 3. Solução OpenMPI

## 3.1. Arquitetura Geral

A figura abaixo representa uma visão geral da solução da paralelização em OpenMPI da versão sequencial do código. Todo o processo é explicado mais detalhadamente em cada uma das restantes secções.

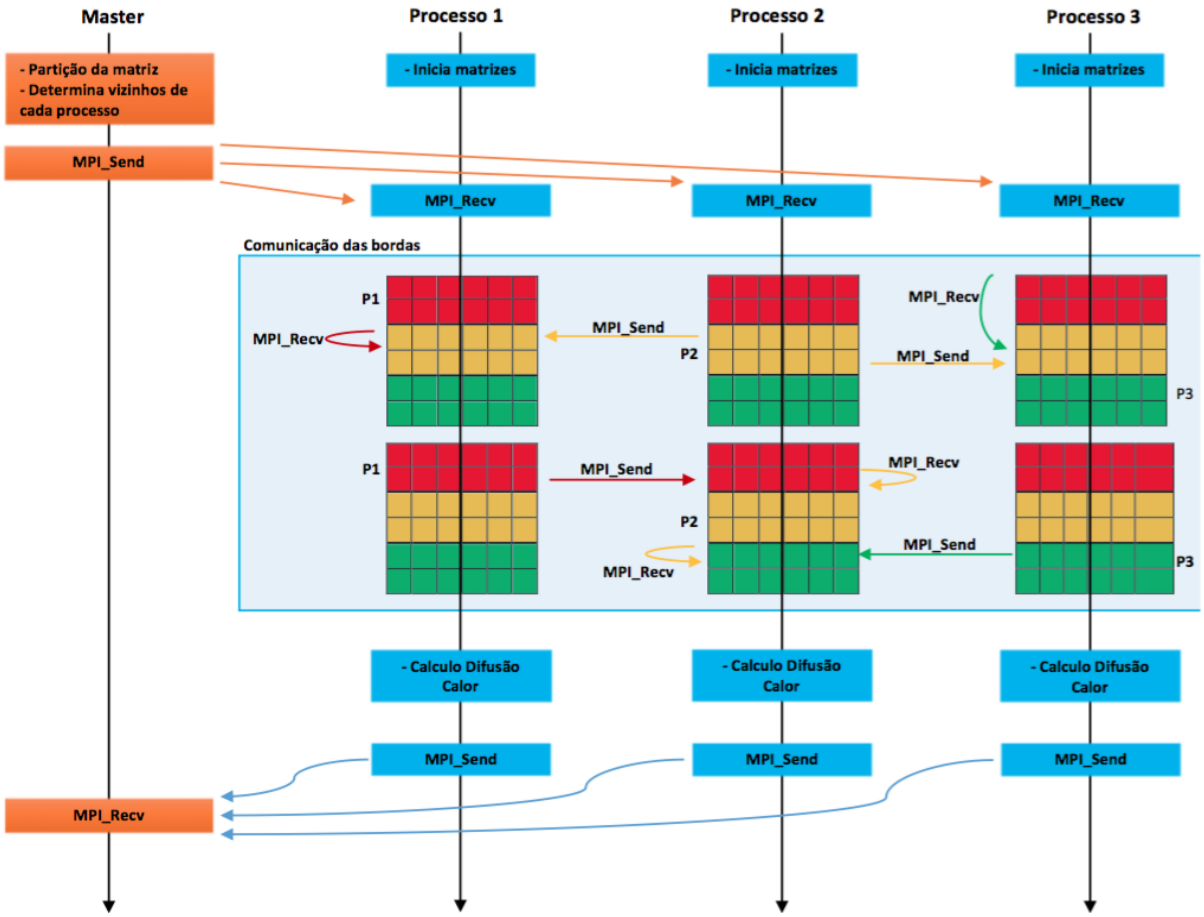



Figure 1: Arquitetura geral da solução em OpenMPI

## 3.2. Encontrar o Paralelismo

Antes de proceder à implementação do código paralelo em memória distribuída foi necessário compreender quais os métodos do algoritmo sequencial que potencialmente devem ser paralelizados. Para este efeito podem ser retirados os tempos de execução de cada método utilizado no algoritmo sequencial e analisar comparativamente os mesmos por forma a determinar qual (ou quais) deve ser paralelizado.

Dado que o algoritmo sequencial é o mesmo que foi utilizado para o primeiro caso de estudo em OpenMP, a análise de paralelismo é igual. Dos vários métodos utilizados no código sequencial, apenas um consegue ser evidentemente destacado, sendo todos os outros métodos de inicialização de variáveis apenas. Esse método é o responsável por efetuar todos os cálculos necessários para o processo de difusão de calor em N_MAX iterações e, por conseguinte, é o método que deve ser paralelizado:

```
    ...
    iterate(heatPlate_new, heatPlate_old, matrix_size, n_max_iterations);
    ...
```

A paralelização em memoria distribuída originou várias alterações no código sequencial, as quais vão ser referidas e analisadas nas secções seguintes.

## 3.3. Partição e Distribuição dos Dados

Por forma a perceber como os vários processos vão efetuar o calculo da difusão de calor sobre a matriz, foi necessário começar por analisar a forma de decomposição dos dados para que sejam distribuídos pelos processos bem como analisar a forma como o calculo da difusão é efetuado.

Na versão sequencial, apenas um processo percorre todas as linhas da matriz em cada iteração por forma a determinar os novos valores de cada posição. De modo a paralelizar em MPI, a matriz deve ser decomposta funcionalmente ou pelo domínio de dados de modo a que cada processo seja responsável pelo calculo individual de uma parte da matriz. Uma vez que a matriz está a ser acedida por linhas (por ser o método mais eficiente de percorrer uma matriz - *row major*), é aplicada uma decomposição pelo domínio de dados.

Esta decomposição pelo domínio de dados resulta na partição da matriz por blocos (conjuntos) de várias linhas onde a cada processo é atribuído um bloco e esse processo é responsável pelo processamento desse bloco. A quantidade de blocos deve ser então igual ao numero de processos a utilizar e a quantidade de linhas por bloco deve ser igual para que se consiga garantir uma correta distribuição de carga entre processos (*load balance*). Neste sentido, foi desenvolvida uma solução de partição eficiente da matriz pelos processos de forma a que cada processo possua no máximo apenas uma linha a mais para processar do que o numero de linhas mínimo (igual em todos os processos). Apresenta-se o excerto de código responsável pela determinação da partição:

```
if(procID == MASTER) {
    lineDivision = (matrix_size/numWorkers);
    lineExtra = (matrix_size%numWorkers);

    ...

    for(int procDest = 1; procDest <= numWorkers; procDest++) {
        /*** Particao eficiente e compacta da matriz ***/
        if(procDest <= lineExtra) numLines = lineDivision + 1;
        else numLines = lineDivision;
        ...
    ...
```

Começa-se por efetuar a divisão do numero de linhas da matriz pelo numero de processos responsáveis pelo processamento de cada bloco de linhas. Caso o resto da divisão inteira desta divisão fosse sempre 0 então estava assegurado que existe um numero igual de linhas a serem processadas por processo, o que nem sempre ocorre. Quando isto não acontece então existem linhas extra a serem processadas/distribuídas pelos processos. Para garantir que existe uma distribuição de carga balanceada, cada processo com id inferior ou igual ao numero extra de linhas vai processar o seu numero mínimo mais uma linha extra. Desta forma garante-se que no máximo cada processo é responsável por apenas uma linha a mais do que o numero de linhas mínimo.

## 3.4. Processos

Para que os vários processos tenham conhecimento da quantidade de linhas que cada um tem de processar e ainda qual a parte da matriz reservada para cada um foi necessário adotar uma variante do algoritmo "Master-Slave". Assim, existe apenas um processo considerado MASTER que é o processo 0. É da responsabilidade do

MASTER determinar o numero de linhas que cada um dos restantes processos vai processar (analisado na secção anterior) e ainda a parte da matriz que cada processo é responsável, ou seja, a linha de inicio de processamento na matriz que juntamente com o numero de linhas a processar representa exatamente a parte da matriz destinada a cada processo.

Atendendo à fórmula do calculo de difusão do calor utilizada, é possível observar que para cada posição de cada linha, são utilizados valores das linhas imediatamente acima e abaixo. Isto implica que os processos responsáveis pelo processamento da sua parte da matriz tenham de ter conhecimento dos valores das linhas superiores à sua linha de inicio de processamento e inferiores à sua linha de fim (com exceção para a borda superior e inferior) que pertencem aos processos vizinhos. Introduz-se assim o conceito de processo vizinho. Um processo é vizinho superior de outro se for o processo imediatamente anterior e é considerado vizinho inferior se for o processo imediatamente depois. Este conceito é necessário uma vez que, para cada iteração, os processos vizinhos devem comunicar entre si por forma a partilhar a informação das linhas que os separam, que são necessárias para o calculo da difusão de calor.

Para que cada processo possa comunicar com os seus vizinhos tem de saber qual o id dos mesmos. É da responsabilidade do MASTER determinar os vizinhos de cada processo.

É ainda da responsabilidade do MASTER enviar toda a informação de cada processo para cada um e receber o resultado final da computação de cada processo.

No excerto de código abaixo é possível observar todos os passos descritos nesta secção com os comentários correspondentes.

```
for(int procDest = 1; procDest <= numWorkers; procDest++) {
    /*** Particao eficiente e compacta da matriz ***/
    if(procDest <= lineExtra) numLines = lineDivision + 1;
    else numLines = lineDivision;

    /*** Determina os vizinhos do processo "procDest" ***/
    if(procDest == 1) processUp = NONE;
    else processUp = procDest - 1;
    if(procDest == numWorkers) processDown = NONE;
    else processDown = procDest + 1;

    /*** Envia a informacao relevante aos processos ***/
    ...

    /*** Incrementa offset pelo nr linhas para que o proximo saiba onde comecar ***/
    offset = offset + numLines;
}

/*** Espera pelos resultados de cada processo ***/
...
```

## 3.5. Comunicação entre Processos e Sincronismo

Como referido anteriormente, foi seguido um algoritmo com base em "Master-Slave" onde é da responsabilidade do processo MASTER efetuar todas as operações necessárias para poder enviar a cada processo a informação que lhe é relevante e receber o resultado final de cada processo. Os vários processos responsáveis pelo processamento na sua parte da matriz devem, a cada iteração, comunicar com os seus processos vizinhos as linhas de borda que os separam.

O excerto de código seguinte mostra o conjunto de dados enviados pelo processo MASTER para cada um dos restantes processos:

```
...
/*** Envia a informacao relevante aos processos ***/
MPI_Send(&offset, 1, MPI_INT, procDest, BEGIN_TAG, MPI_COMM_WORLD);
MPI_Send(&numLines, 1, MPI_INT, procDest, BEGIN_TAG, MPI_COMM_WORLD);
MPI_Send(&processUp, 1, MPI_INT, procDest, BEGIN_TAG, MPI_COMM_WORLD);
MPI_Send(&processDown, 1, MPI_INT, procDest, BEGIN_TAG, MPI_COMM_WORLD);
...
```

Como é possível observar, é enviado a cada processo a posição na matriz onde o mesmo deve começar a processar e o numero de linhas da matriz a processar. Estes dois valores em conjunto definem a porção da matriz reservada para cada processo. É ainda enviada informação acerca dos vizinhos do processo (caso existam) para que os processos possam comunicar entre eles.

Numa primeira implementação da solução em MPI, o processo MASTER iniciava a matriz com os valores iniciais e enviava ainda a cada processo esses valores iniciais referentes apenas à parte da matriz de cada processo. Contudo, rapidamente se verificou que era uma solução ineficiente uma vez que bastava que cada processo inicia-se também a matriz com os valores iniciais para deixar de fazer sentido que o MASTER envie esse conjunto de valores, reduzindo assim significativamente o tempo de comunicação.

O excerto de código abaixo mostra a primeira comunicação efetuada pelos restantes processos - receber os dados vindos do processo MASTER:

```
...
/*** Recebe informacao do MASTER ***/
MPI_Recv(&offset, 1, MPI_INT, 0, BEGIN_TAG, MPI_COMM_WORLD, &status);
MPI_Recv(&numLines, 1, MPI_INT, 0, BEGIN_TAG, MPI_COMM_WORLD, &status);
MPI_Recv(&processUp, 1, MPI_INT, 0, BEGIN_TAG, MPI_COMM_WORLD, &status);
MPI_Recv(&processDown, 1, MPI_INT, 0, BEGIN_TAG, MPI_COMM_WORLD, &status);
...
```

Os dados são recebidos pela mesma ordem que são enviados pelo MASTER.

Após a receção dos dados, cada processo pode então começar o processo de difusão de calor para a sua parte da matriz. Para cada iteração, começa-se por efetuar a comunicação das linhas de borda entre processos vizinhos. Este ponto foi particularmente difícil de conseguir pelo que, numa primeira implementação, foi criado um conjunto de código compacto e simples que efetuava a comunicação entre os processos vizinhos de forma eficiente, contudo, tinha um comportamento correto apenas para valores de matrizes inferiores a 1010. Após uma análise das mensagens de erro geradas pelo MPI para matrizes de dimensão superior a 1010 acreditamos que o erro estaria possivelmente relacionado com a capacidade dos *buffers* de *send* e *receive*, utilizados nos métodos do MPI, não serem suficientes. Foi alterado o código por forma a utilizar *Bsend* juntamente com todos os parâmetros adicionais de reserva de memória dos *buffers*. Contudo, esta solução também gerava erros para matrizes de dimensões grandes. Foi feita outra versão do código para fazer uso de *Isend* mas o erro persistia. Foi apenas após vários dias de análise e estudo de comportamento do MPI que o erro foi detetado, a implementação utilizada estava sujeita a *deadlocks* nos *sends* uma vez que podia atingir um ponto em que todos os processos estavam a efetuar *send* e nenhum *receive*.

A solução encontrada para contornar este problema passou por garantir que sempre que um processo efetue um *send* existe sempre outro processo que está à espera de receber essa informação. Isto foi conseguido a partir da partição do processo em duas fazes:

- Primeira:
  - Se o processo for PAR então ENVIA as suas linhas de borda para os processos vizinhos
  - Se o processo for ÍMPAR então RECEBE as linhas de borda dos seus processos vizinhos
- Segunda:
  - Se o processo for ÍMPAR então ENVIA as suas linhas de borda para os processo vizinhos
  - Se o processo for PAR então RECEBE as linhas de borda dos seus processos vizinhos

Nota: o excerto de código responsável pelo processo acima referido pode ser consultado no anexo B.

Após a troca entre processos vizinhos cada processo efetua o calculo da difusão de calor para a sua parte da matriz e o processo é repetido até N_MAX iterações após as quais cada processo envia para o MASTER o seu resultado final da sua parte da matriz como pode ser verificado pelo excerto de código:

```
...
/*** Processo envia para o MASTER, o resultado do calculo da sua parte da matriz ***/
MPI_Send(&offset, 1, MPI_INT, MASTER, END_TAG, MPI_COMM_WORLD);
MPI_Send(&numLines, 1, MPI_INT, MASTER, END_TAG, MPI_COMM_WORLD);
MPI_Send(&(heatPlate_new[offset][0]), numLines*matrix_size, MPI_FLOAT, MASTER, END_TAG, ↩
    MPI_COMM_WORLD);
...
```

### 3.5.1 Comunicações Coletivas

Para além da forma de comunicação ponto-a-ponto utilizada em todas as comunicações efetuadas, o OpenMPI possui ainda formas de comunicação coletiva: *MPI_Bcast*, *MPI_Scatter/Gather* e *MPI_Reduce*.

Uma boa análise de qual forma de comunicação é mais eficiente para o problema a resolver seria implementar várias versões do código desenvolvido, cada uma com a combinação adequada de formas de comunicação coletiva, e determinar os tempos de comunicação de cada uma.

Esta análise foi ponderada pelo grupo, contudo, tendo em conta a natureza das comunicações efetuadas e o código desenvolvido com formas de comunicação ponto-a-ponto, chegou-se à conclusão que nenhuma forma de comunicação coletiva era facilmente integrada no problema. Isto deve-se sobretudo à natureza específica das comunicações que são efetuadas - cada comunicação ocorre sempre entre dois processos com informação transmitida relevante apenas ao processo destino.

Analisando mais detalhadamente as comunicações existentes, toda a informação que o MASTER envia é distinta entre os vários processos e portanto relevante apenas para o processo destino logo não é possível utilizar MPI_Bcast ou MPI_Scatter. As comunicações entre processos (*workers*) são efetuadas apenas com os processos vizinhos com informação específica para cada um o que leva mais uma vez a que MPI_Bcast ou MPI_Scatter não consigam ser utilizados. Por fim, analisando a comunicação de resposta dos processos para o MASTER, foi ponderado a utilização de MPI_Reduce, contudo, como cada processo responde com a sua parte da matriz que tem um destino específico na matriz do processo MASTER, leva a que a utilização deste tipo de comunicação seja possível apenas pela alteração de uma grande parte de código, que provavelmente não vai trazer ganhos uma vez que todas as restantes comunicações são efetuadas de forma ponto-a-ponto.

Em suma, pela análise efetuada pelo grupo, a utilização de formas de comunicação coletivas não são viáveis para a análise do problema em questão, tendo em conta a implementação efetuada para a comunicação ponto-a-ponto, sendo necessário uma grande reestruturação do código para que possam ser utilizadas.

## 3.6. Mapeamento das Tarefas

Até aqui foi analisada a forma como a distribuição de informação de processamento é efetuada entre os vários processos, resta agora analisar a forma como as tarefas são mapeadas nas máquinas utilizadas.

Foram utilizadas até **2 máquinas** do segmento **641** que utilizam a rede **Ethernet** como meio de comunicação. Cada máquina deste segmento possui **2 sockets** de 8 cores físicos cada. Foi ainda utilizada **1 máquina** do segmento **662** com utilização da rede **Myrinet**. Cada máquina deste segmento possui também **2 sockets** com 12 cores físicos cada. Mais informação acerca de cada máquina pode ser encontrado no anexo A.

Uma vez que o ambiente de execução MPI permite a especificação da forma de mapeamento dos executáveis, foi necessário decidir qual mapeamento utilizar - mapeamento por core ou por nodo. Foi utilizado o **mapeamento por core**. Tendo em conta as especificações das máquinas utilizadas, o mapeamento por nodo leva a que seja atingido logo desde início o overhead máximo de comunicação uma vez que logo a partir da utilização de dois processos, inclusive, a comunicação é efetuada em nodos distintos. No mapeamento por core, são utilizados primeiro todos os cores de um nodo, passando para os cores de outro nodo quando os do primeiro não são suficientes para satisfazer o numero de cores pretendidos. A forma de execução do job adotada permite obter uma representação gráfica da distribuição de trabalho por core:

```
[compute-641-7.local:29458]
[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]
                                        (...)
```

Como é possível observar, cada rank está a ser distribuído nos diferentes cores de cada socket da máquina e todos eles executam em paralelo. Os processos são distribuídos uniformemente por cada core. É possível 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 da máquina.

# 4. Metodologia e Condições de Medição

O tempo de execução do algoritmo paralelo em MPI foi aproximado pela estimativa de tempo walltime do processo mais lento (*Slowest Process Time*). Para além da obtenção do tempo de execução foi ainda necessário medir:

- Tempo total gasto em comunicação (**MPI_Send + MPI_Recv**)
- Tempo total gasto em computação
- A reserva de nodos de computação foi efetuada de forma exclusiva
- Foi utilizado o contador de tempo do MPI para obter os tempos (**MPI_Wtime()**)
- A resolução de tempos utilizada é o **milissegundo (ms)**
- As áreas delimitadas para a medição dos tempos não incluem qualquer tipo de I/O
- Foram recolhidas 5 medições para cada tempo e aplicada a mediana sobre os resultados

# 5. Avaliação da Versão Paralela em OpenMPI

Por forma a poder avaliar a performance da solução em MPI é necessário estudar o grau de escalabilidade da mesma. Contudo, neste paradigma é necessário analisar um fator extra quando comparado com OpenMP - os vários processos necessitam comunicar entre si para partilhar os dados e mensagens necessárias e de acordo com a implementação do algoritmo - existe portanto um overhead adicional ao da computação paralela que é o overhead de comunicação dos dados entre processos. Isto leva a que seja necessário ter em consideração os **tempos de computação, comunicação e ócio** nos ganhos obtidos.

## 5.1. Evolução do Tempo de Comunicação - nodos 641 por Ethernet

Por forma a perceber o impacto da comunicação entre processos fez-se uma análise da evolução do tempo total gasto em comunicação (o tempo gasto por todos os processos) à medida que são utilizados mais processos. O tempo de comunicação de cada processo corresponde ao tempo gasto a enviar e a receber informação. Quando são utilizados mais que 32 processos estão a ser utilizados 2 nodos de computação.

A baixo são apresentados gráficos da evolução do tempo de comunicação. Deve-se ter especial atenção na escala de cada um.

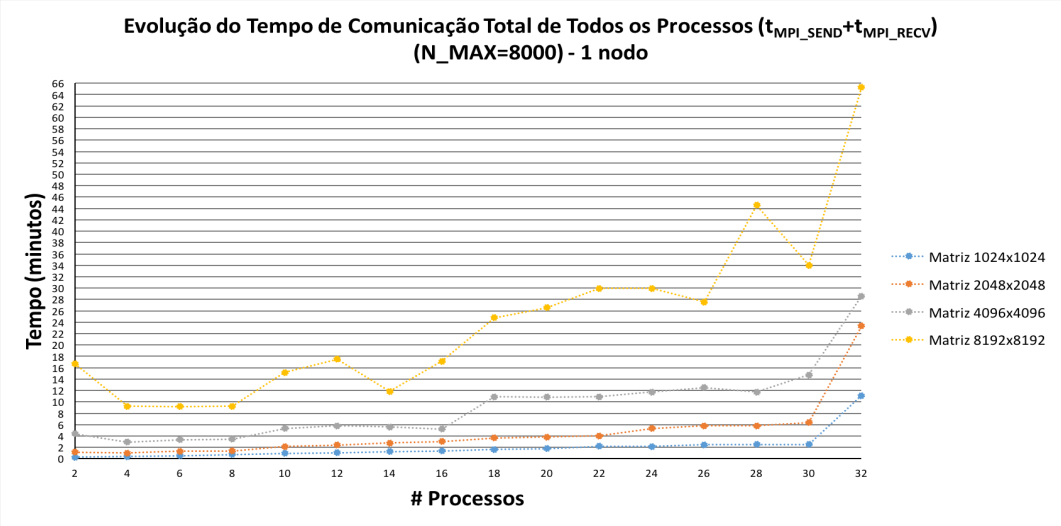



Figure 2: Evolução do tempo de comunicação em 1 nodo

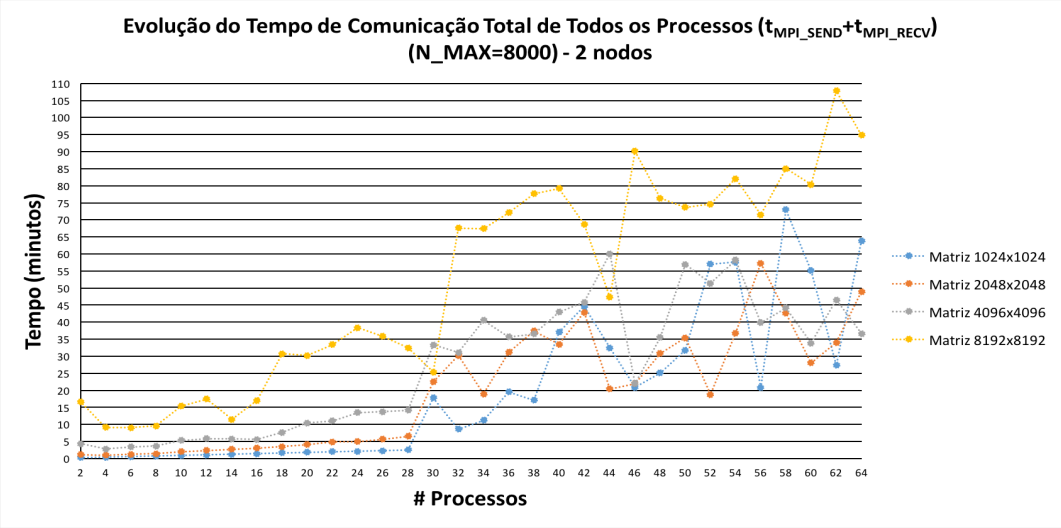



Figure 3: Evolução do tempo de comunicação em 2 nodos

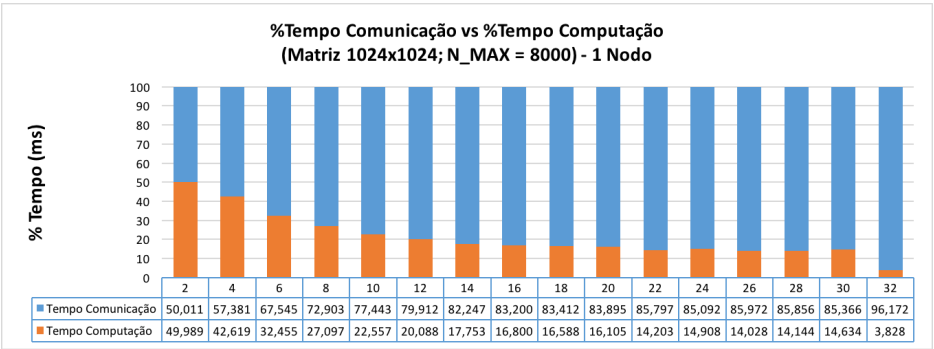
Na utilização de apenas 1 nodo, é possível observar que os tempos de comunicação não aumentam significativamente à medida que são utilizados mais processos, com a exceção da matriz 8192x8192, o que era expectável atendendo à forma como a comunicação é efetuada na solução desenvolvida - só ocorrem comunicações entre processos (*slaves*) na partilha de linhas da matriz que é sempre entre dois processos e no envio e receção de informação de e para o MASTER. A matriz 8192x8192 apresenta um comportamento mais irregular possivelmente devido à quantidade elevada de dados a serem transmitidos, principalmente quando são enviados os resultados para o MASTER que é quando ocorre o maior envio de dados e ainda pela forma como o MASTER recebe os resultados - espera pelos resultados de cada processo de forma sequencial do id de cada processo - se um processo se atrasar todos os processos com id superior vão esperar para poder enviar.

Na utilização de 2 nodos, os tempos de comunicação aumentam de forma acentuada e irregular para qualquer matriz e à medida que mais processo são utilizados. Isto acontece devido ao maior tempo necessário para efetuar a comunicação entre as duas máquinas, que está sujeita à velocidade da rede bem como a possíveis congestionamentos.

## 5.2. Evolução do Tempo de Comunicação vs Tempo Computação - nodos 641 por Ethernet

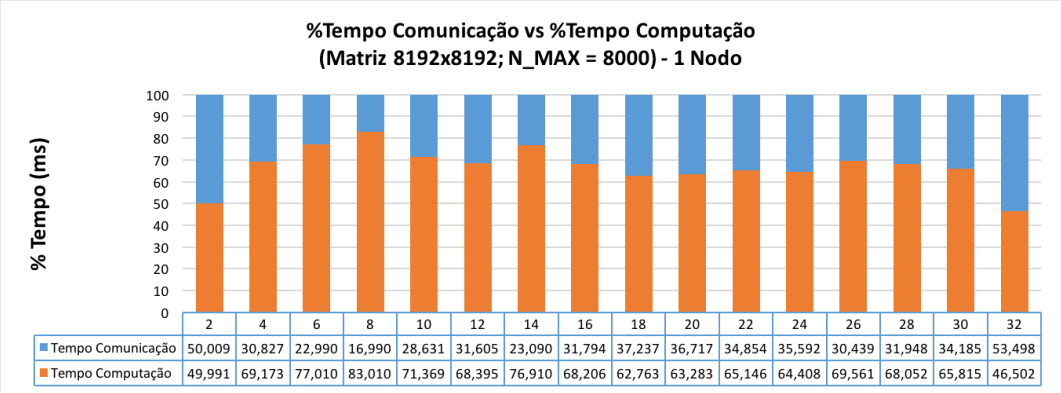
Como foi visto na secção anterior, o overhead de comunicação aumenta à medida que são utilizados mais processos. Contudo, por forma a perceber o impacto da comunicação na solução desenvolvida é necessário relacioná-la com o tempo de computação. Isto permite saber se os ganhos obtidos estão limitados pelo tempo de comunicação, computação ou ambos.

Os gráficos abaixo permitem tirar estas conclusões. Uma vez que para a utilização de 2 nodos, os resultados obtidos seguem o mesmo padrão dos resultados para 1 nodo, os gráficos para 2 nodos podem ser consultados no anexo C. São também apresentados apenas os gráficos referentes a 1 nodo para a matrizes 1024x1024 e 8192x8192 porque as restantes matrizes apresentam um comportamento incremental que vai ser explicado (os gráficos das restantes matrizes podem ser consultados no anexo C).




| | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 | 26 | 28 | 30 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Tempo Comunicação | 50,011 | 57,381 | 67,545 | 72,903 | 77,443 | 79,912 | 82,247 | 83,200 | 83,412 | 83,895 | 85,797 | 85,092 | 85,972 | 85,856 | 85,366 | 96,172 |
| Tempo Computação | 49,989 | 42,619 | 32,455 | 27,097 | 22,557 | 20,088 | 17,753 | 16,800 | 16,588 | 16,105 | 14,203 | 14,908 | 14,028 | 14,144 | 14,634 | 3,828 |

Figure 4: Tempo comunicação vs tempo computação em 1 nodo para matriz 1024x1024




| | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 | 26 | 28 | 30 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Tempo Comunicação | 50,009 | 30,827 | 22,990 | 16,990 | 28,631 | 31,605 | 23,090 | 31,794 | 37,237 | 36,717 | 34,854 | 35,592 | 30,439 | 31,948 | 34,185 | 53,498 |
| Tempo Computação | 49,991 | 69,173 | 77,010 | 83,010 | 71,369 | 68,395 | 76,910 | 68,206 | 62,763 | 63,283 | 65,146 | 64,408 | 69,561 | 68,052 | 65,815 | 46,502 |

Figure 5: Tempo comunicação vs tempo computação em 1 nodo para matriz 8192x8192

Na utilização de 1 nodo de computação, para matrizes de pequena dimensão - 1024x1024 e 2048x2048 - os ganhos estão globalmente limitados pelo tempo de comunicação à medida que vão sendo utilizados mais processos. Isto acontece porque à medida que são utilizados mais processos, e como são matrizes de pequena dimensão, menor vai ser os esforço de computação o que aumenta a percentagem do tempo de comunicação. Isto é ainda mais acentuado na utilização de 2 nodos de computação.

Para matrizes de grande dimensão - 4096x4096 e 8192x8192 - na utilização de 1 nodo de computação pode-se considerar que os ganhos estão limitados pelo tempo de computação devido à grande quantidade de dados a serem processados por cada processo. Isto tende a diminuir à medida que são utilizados mais processos - com 2 nodos de computação os ganhos são limitados tanto pelo tempo de comunicação quer pelo tempo de computação - caso fossem acrescentados mais nodos, o tempo de comunicação iria sobrepor gradualmente sobre o tempo de computação.

## 5.3. SpeedUp's - nodos 641 por Ethernet

Apresentam-se abaixo os gráficos dos ganhos obtidos, para cada matriz com 8000 iterações, na utilização de 1 e 2 nodos 641 com comunicação por Ethernet.

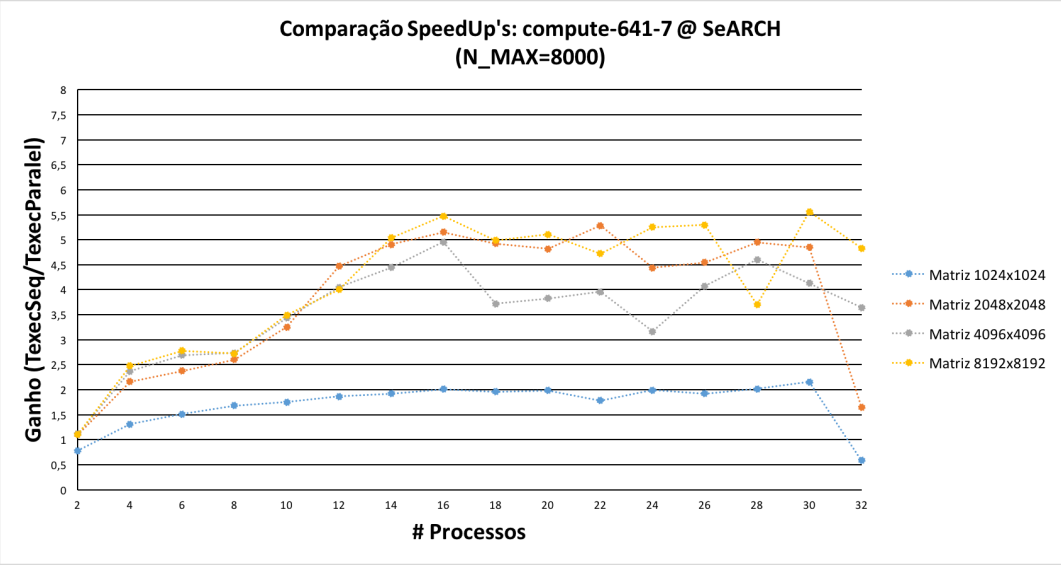



Figure 6: SpeedUp's de cada matriz em 1 nodo 641 com comunicação Ethernet

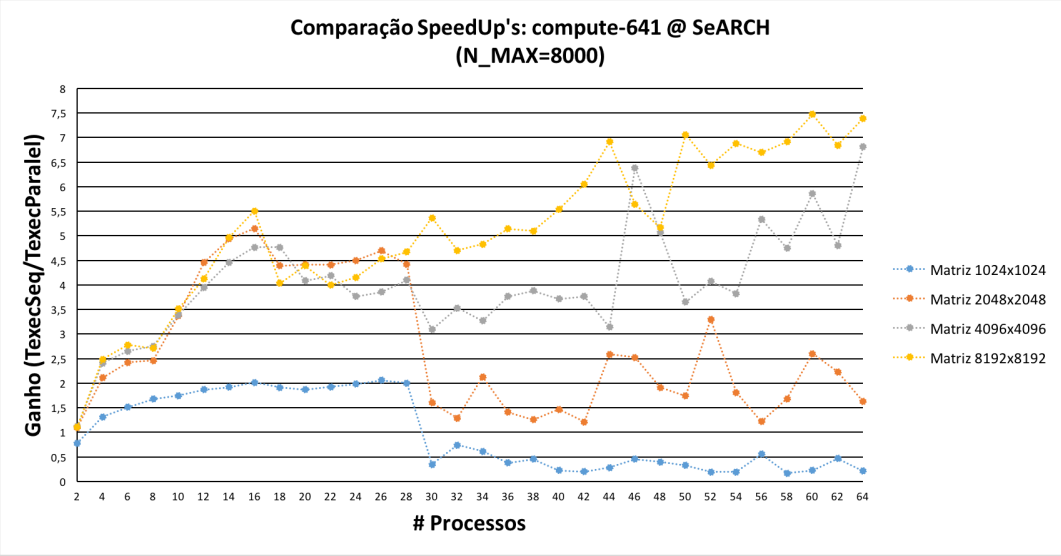



Figure 7: SpeedUp's de cada matriz em 2 nodos 641 com comunicação Ethernet

Atendendo à análise efetuada na secção anterior, os ganhos obtidos são facilmente explicáveis. Na utilização de apenas 1 nodo de computação, a matriz 1024x1024 é a que apresenta menores ganhos devido ao overhead de comunicação elevado. Era de esperar que a matriz 2048x2048 fosse a segunda com menores ganhos, o que não acontece aqui nem aconteceu na paralelização com OpenMP, possivelmente devido à forma como os dados estão guardados e são acedidos em memória. As matrizes de dimensão 4096x4096 e 8192x8192 são as que possuem maiores ganhos pois são limitadas pelo tempo de computação devido à grande quantidade de dados a serem processados em cada iteração.

É ainda possível observar que os ganhos vão crescendo até à utilização de cerca de 16 processos passando depois a serem constantes. Isto acontece devido ao peso crescente da comunicação sobre a computação e ainda devido à limitação da implementação sequencial do processamento efetuado por cada processo. De notar ainda que a partir de 16 processos até aos 32 utilizados, são processos em cores virtuais o que pode também limitar o ganho.

Quando se utiliza 2 nodos de computação, até à utilização de 32 processos, os ganhos são semelhantes à utilização de apenas 1 nodo - como seria de esperar devido ao mapeamento por core e não por nodo. Contudo, a partir da utilização de mais do que 32 processos existe uma perda inicial acentuada devido à utilização do segundo nodo de computação que leva a um aumento do overhead de comunicação. Isto é especialmente visível para as três matrizes de menor dimensão - a matriz 1024x1024 chega mesmo a ter perdas em comparação com a versão sequencial devido ao baixo tempo de computação e elevado tempo de comunicação à medida que são utilizados mais processos.

Na matriz 8192x8192 não ocorre esta perda devido ao equilíbrio do tempo de comunicação e computação. De facto, esta matriz apresenta ganhos praticamente crescentes à medida que são utilizados mais processos pois o tempo de computação amortiza o tempo de computação. Caso fossem utilizados mais nodos de computação este ganho começaria a diminuir gradualmente pois o tempo necessário para a computação seria menor, devido ao aumento de processos responsáveis pela mesma, passando a prevalecer o tempo de comunicação.

## 5.4. SpeedUp's OpenMP vs OpenMPI - nodos 641 por Ethernet

Uma vez que foi efetuada uma versão em OpenMP do mesmo problema, é relevante comparar qual das versões paralelas é mais eficiente quando comparadas com a versão sequencial. Note-se que nos gráficos abaixo, em OpenMP, são utilizadas no máximo 24 threads, que foi o numero máximo escolhido no primeiro trabalho.

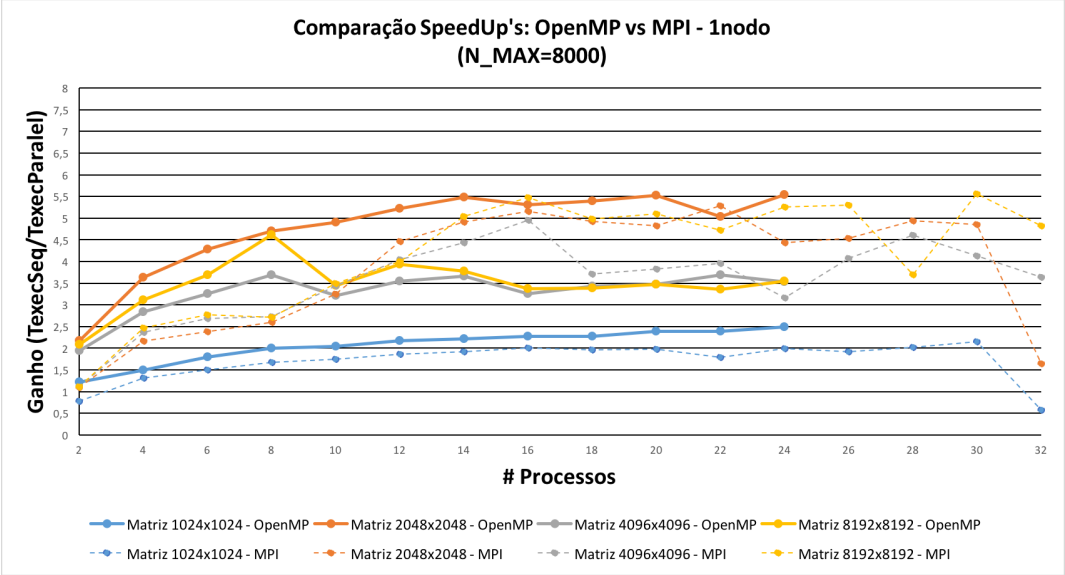



Figure 8: SpeedUp's de cada paradigma em 1 nodo 641 com comunicação Ethernet

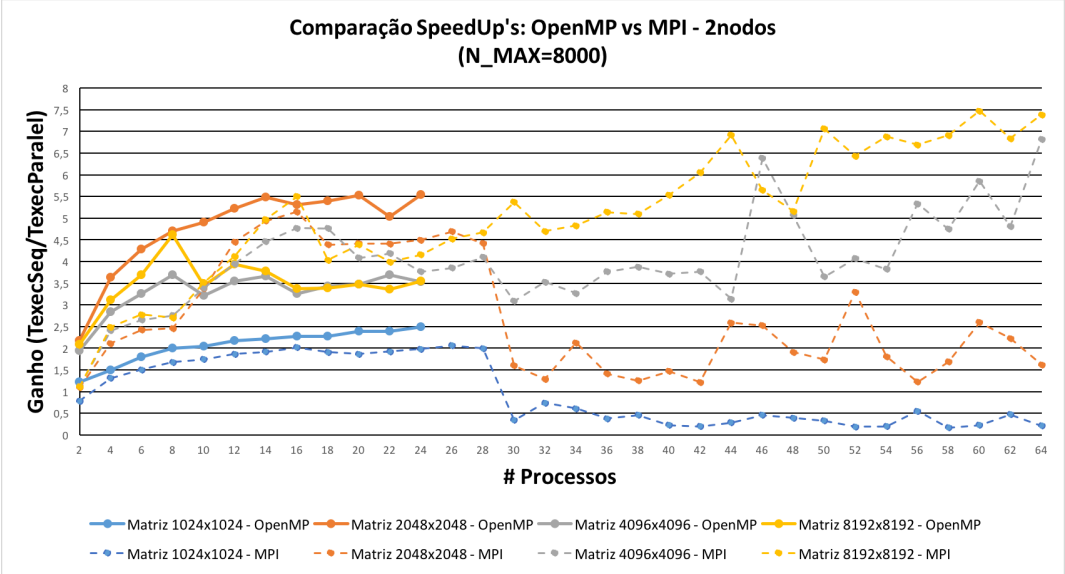



Figure 9: SpeedUp's de cada paradigma em 2 nodos 641 com comunicação Ethernet

É possível constatar que os ganhos em OpenMP são muito mais consistentes/regulares à medida que se utilizam mais processos/threads.

Comparando os ganhos na utilização de apenas 1 nodo de computação é possível verificar que a versão OpenMP tem melhores ganhos para matrizes de pequenas dimensões - 1024x1024 e 2048x2048. Para matrizes de maiores dimensões - 4096x4096 e 8192x8192 - é a versão OpenMPI que possui melhores ganhos, embora não muito maiores do que a versão em OpenMP.

O referido acima é especialmente fácil de constatar quando são utilizados 2 nodos de computação. Para as matrizes de pequena dimensão, a utilização de vários processos em OpenMPI leva a uma perda elevada de ganhos quando comparado com OpenMP (existe até mesmo perda quando comparado com a versão sequencial para a matriz 1024x1024). Contudo, o ganho pouco acentuado, nas matrizes de grande dimensão, obtido na utilização de apenas 1 nodo de computação em MPI, começa a ser gradualmente melhor à medida que são utilizados mais processos. De facto, para a matriz 8192x8192, os ganhos com a utilização de 64 processos conseguem ser praticamente duas vezes melhores do que o melhor ganho em OpenMP para essa matriz.

### 5.5. Conclusões da Utilização de OpenMPI na Solução Proposta - nodos 641 por Ethernet

Na utilização do modelo de memória distribuída implementado pelo OpenMPI é necessário ter em consideração o overhead de comunicação adicional no estudo da escalabilidade de um algoritmo. Na solução proposta para o processo de difusão de calor procurou-se minimizar ao máximo os fatores que podem comprometer a escalabilidade de uma solução em OpenMPI - solução computacional de cada processo, balanceamento de carga entre processos e tempo de comunicação.

Na solução implementada foi tido o cuidado para que o balanceamento de carga fosse o mesmo entre todos os processos responsáveis pelo calculo da difusão de calor, garantindo assim que os resultados obtidos não são limitados por este fator. De facto, atendendo à análise das várias medidas apresentadas, os fatores condicionantes dos ganhos estão relacionados apenas com o tempo de comunicação e computação.

Foi demonstrado que o overhead de comunicação aumenta de forma incremental à medida que são utilizados mais processos. A relação do overhead de comunicação com o tempo de comunicação permitiu confirmar o que era expectável para os valores dos ganhos: matrizes de pequenas dimensões estão limitadas pelo tempo de comunicação, que vai sendo cada vez maior à medida que são utilizados mais processos, devido à baixa quantidade de dados a serem processados (menor tempo de computação). Matrizes de grandes dimensões estão limitadas pelo tempo de computação quando é utilizado 1 nodo de computação e começam a possuir um equilíbrio de tempo entre comunicação e computação quando são utilizados 2 nodos. Caso fossem acrescentados mais nodos o tempo de comunicação iria começar a prevalecer sobre o tempo de computação.

Tendo em conta todos estes dados e pela análise comparativa com a solução em OpenMP pode-se concluir que para matrizes de baixa dimensão a solução em OpenMP é mais eficiente em termos de ganhos (speedup's) quando são utilizados até cerca de 32 processos/threads. Para matrizes de maior dimensão, a solução em OpenMPI é a que deve ser escolhida, quando é executada em pelo menos 2 nodos de computação - 64 processos na máquina utilizada - como é evidente pelo crescimento dos ganhos da matriz 8192x8192 quando são utilizados 64 processos.

## 6. Outras Medidas

### 6.1. SeARCH compute-641 :: 4 Nodos via Ethernet

Por forma a dar continuação ao estudo da utilização de mais nodos de computação, para o algoritmo desenvolvido, foram efetuados testes, nas mesmas condições das medições anteriores, mas com 4 nodos de computação o que equivale a um máximo de 128 possíveis processos a executar em paralelo.

Contudo, o grupo não conseguiu obter os resultados das medidas para este teste visto que aquando da entrega do relatório, o *job* a ser executado no SeARCH Cluster ainda não tinha sido escalonado. Caso fossem obtidos valores, os mesmos poderiam servir para confirmar a expetativa de que o tempo de comunicação iria sobrepor-se ao tempo de computação, à medida que são utilizados mais processos, para matrizes de grande dimensão.

### 6.2. SeARCH compute-662 :: 1 Nodo (8 processos) via Myrinet

Visto que as condicionantes de ganhos da solução desenvolvida estão relacionadas com o tempo de comunicação e computação, consoante o tamanho de matriz utilizado, caso seja possível diminuir estes tempos então são obtidos melhores ganhos. Para tal, foram efetuados testes no SeARCH compute-662, que possui uma maior capacidade de processamento quando comparado com o 641 previamente utilizado e com comunicação Myrinet que é 10 vezes mais rápida do que a comunicação por Ethernet. Com isto procurava-se obter um maior ganho bem como melhores tempos de comunicação e computação.

#### 6.2.1 Compilação e Execução

Uma vez que um nodo 662 possui 48 cores e um nodo 641 possui 32, para efetuar uma comparação de tempos equilibrada utilizando apenas 1 nodo, foram reservados os 48 cores do nodo 662 (para que a máquina fosse de execução exclusiva) mas foram utilizados apenas até 32 processos. Contudo, na utilização de um numero de processos superior a 8, ocorria um erro. Este erro deve-se ao facto de a utilização de rede Myrinet permitir apenas a comunicação máxima entre 8 processos.

#### 6.2.2 Resultados compute-662 :: 1 Nodo (8 processos) via Myrinet

Dada a limitação de na utilização de 1 nodo, com comunicação Myrinet, ser possível utilizar apenas até 8 processos e uma vez que já tinham sido recolhidos tempos na utilização de até 8 processos, decidiu-se fazer a análise desses tempos em comparação com a utilização de 1 nodo 641 até 8 processos.

O gráfico abaixo apresenta o ganho da utilização de 1 nodo 662 com comunicação Myrinet em comparação com a utilização de 1 nodo 641 com rede Ethernet até 8 processos.

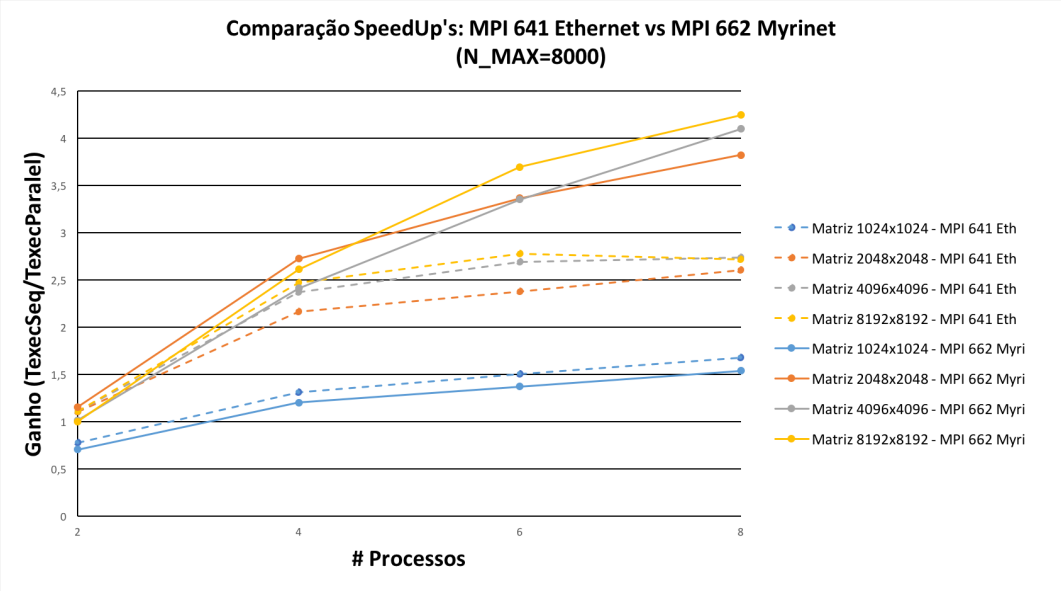



Figure 10: SpeedUp's nodo 641 com comunicação Ethernet e nodo 662 com comunicação Myrinet

É possível observar que a utilização de 1 nodo de computação 662 com comunicação Myrinet possui mais ganhos do que a utilização de 1 nodo 641 com comunicação Ethernet, para qualquer tamanho de matriz, exceto para a matriz 1024x1024.

Por forma a perceber como estes ganhos foram atingidos, se por melhoria de tempo de comunicação, computação ou ambos, foi efetuada uma análise dos ganhos de comunicação e computação, cujos valores/resultados podem ser consultados nos anexos D e E, respetivamente.

Atendendo à comparação dos ganhos obtidos entre comunicação Ethernet e Myrinet, é possível constatar que, para a utilização de 8 processos, a utilização de comunicação por Myrinet é mais lenta do que comunicação Ethernet (observando os valores recolhidos pode ser considerado que ambas as comunicações possuem tempos iguais). Apesar da a rede Myrinet ser 10 vezes mais rápida, esta "igualdade" de tempos pode ocorrer nas matrizes de pequena dimensão, que apesar de serem limitadas pelo tempo de comunicação (concluído anteriormente), não apresenta ganhos de comunicação devido à pouca quantidade de dados a serem transmitidos. Para as matrizes de grande dimensão, é possível também acontecer uma vez que são limitadas pelo tempo de computação (concluído anteriormente) logo é difícil obter melhores resultados para o tempo de comunicação.

Analisando agora os ganhos obtidos na computação é possível constatar que o nodo 662 apresenta uma grande percentagem de ganho de computação, chegando mesmo em alguns casos a ser superior a 150%. Isto leva a concluir que o maior speedup obtido para utilização de 1 nodo 662 com Myrinet deve-se exclusivamente ao ganho no tempo de computação. Uma vez que o tempo de computação melhorou significativamente e o tempo de comunicação manteve-se, então, esta solução é limitada pelo tempo de comunicação quando comparado com a utilização de 1 nodo 641 por Ethernet.

## 6.3. SeARCH compute-662 :: 4 Nodos (32 processos) via Myrinet

Atendendo aos resultados, acima apresentados, da utilização de comunicação Myrinet num nodo 662, que não obteve ganhos em relação à rede Ethernet, uma das razões para tal pode estar relacionada com o baixo numero de processos utilizados (8 processos). Desta forma, e com o intuito de efetuar uma comparação com utilização de 32 processos (nº de processos utilizados nos testes para o nodo 641) em comunicação Myrinet, foi desenvolvido um *job* que consegui-se utilizar mais que 8 processos neste tipo de comunicação.

Para tal, foram reservados especificamente 4 nodos 662 (que corresponde a 192 cores disponíveis) mas, em cada nodo foram utilizados apenas 8 cores (informação introduzida no ficheiro de *hostfile* requerido pela utilização de comunicação Myrinet). Desta forma, cada um dos 4 nodos 662 vai utilizar 8 dos seus cores o que origina um total de 32 processos que podem ser utilizados.

Contudo, mais uma vez, o grupo não conseguiu obter os resultados das medidas para este teste pois aquando da entrega do relatório, o *job* responsável ainda não tinha sido escalonado. Caso fossem obtidos valores, possivelmente começaria a ser observado melhorias nos tempos de comunicação devido à utilização da rede Myrinet bem como as previsíveis melhorias dos tempos de computação devido à utilização de nodos 662.

# A. Máquinas de Teste

| Hardware | SeARCH (compute-641) | SeARCH (compute-662) |
|---|---|---|
| Processador | Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz | Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz |
| Cache | 20MB | 30MB |
| Memória | 64GB | 64GB |
| #Cores | 16 físicos + 16 virtuais | 24 físicos + 24 virtuais |
| #Sockets | 2 | 2 |
| Sistema Operativo | Rocks 6.1 (Emerald Boa) | Rocks 6.1 (Emerald Boa) |

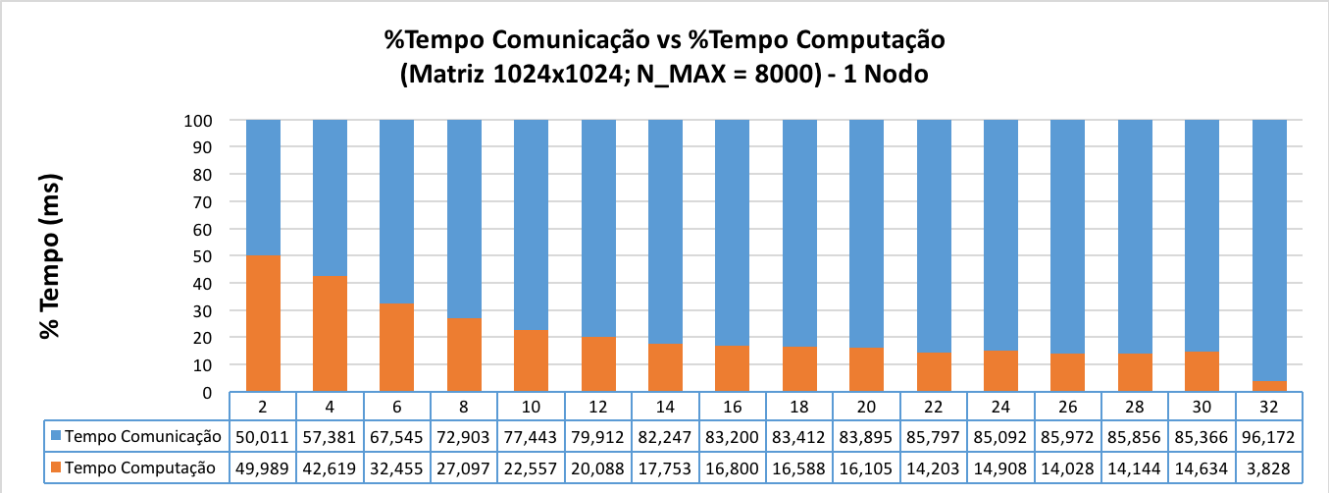
# B. Excerto - Comunicação de Linhas de Borda Entre Processos Vizinhos

```
...
/*** Efetua ate N_MAX iteracoes ***/
for(int iteration = 0; iteration < n_max_iterations; iteration++) {
    /*** Efetua a comunicacao das bordas de particao entre processos ***/
    /***     Se o processo for PAR ENVIA, se for IMPAR RECEBE      ***/
    if((procID%2) == 0) {
        if(processUp != NONE)
            MPI_Send(&(heatPlate_new[offset][0]), matrix_size, MPI_FLOAT, processUp, ↩
                PUP_TAG, MPI_COMM_WORLD);
        if(processDown != NONE)
            MPI_Send(&(heatPlate_new[offset+numLines-1][0]), matrix_size, MPI_FLOAT, ↩
                processDown, PDOWN_TAG, MPI_COMM_WORLD);
    }
    else {
        if(processUp != NONE)
            MPI_Recv(&(heatPlate_new[offset-1][0]), matrix_size, MPI_FLOAT, processUp, ↩
                PDOWN_TAG, MPI_COMM_WORLD, &status);
        if(processDown != NONE)
            MPI_Recv(&(heatPlate_new[offset+numLines][0]), matrix_size, MPI_FLOAT, ↩
                processDown, PUP_TAG, MPI_COMM_WORLD, &status);
    }
    /***     Se o processo for IMPAR ENVIA, se for PAR RECEBE      ***/
    if((procID%2) != 0) {
        if(processUp != NONE)
            MPI_Send(&(heatPlate_new[offset][0]), matrix_size, MPI_FLOAT, processUp, ↩
                PUP_TAG, MPI_COMM_WORLD);
        if(processDown != NONE)
            MPI_Send(&(heatPlate_new[offset+numLines-1][0]), matrix_size, MPI_FLOAT, ↩
                processDown, PDOWN_TAG, MPI_COMM_WORLD);
    }
    else {
        if(processUp != NONE)
            MPI_Recv(&(heatPlate_new[offset-1][0]), matrix_size, MPI_FLOAT, processUp, ↩
                PDOWN_TAG, MPI_COMM_WORLD, &status);
        if(processDown != NONE)
            MPI_Recv(&(heatPlate_new[offset+numLines][0]), matrix_size, MPI_FLOAT, ↩
                processDown, PUP_TAG, MPI_COMM_WORLD, &status);
    }

    /*** Efetua o calculo da difusao do calor para a sua parte da matrix ***/
 ...
}
```

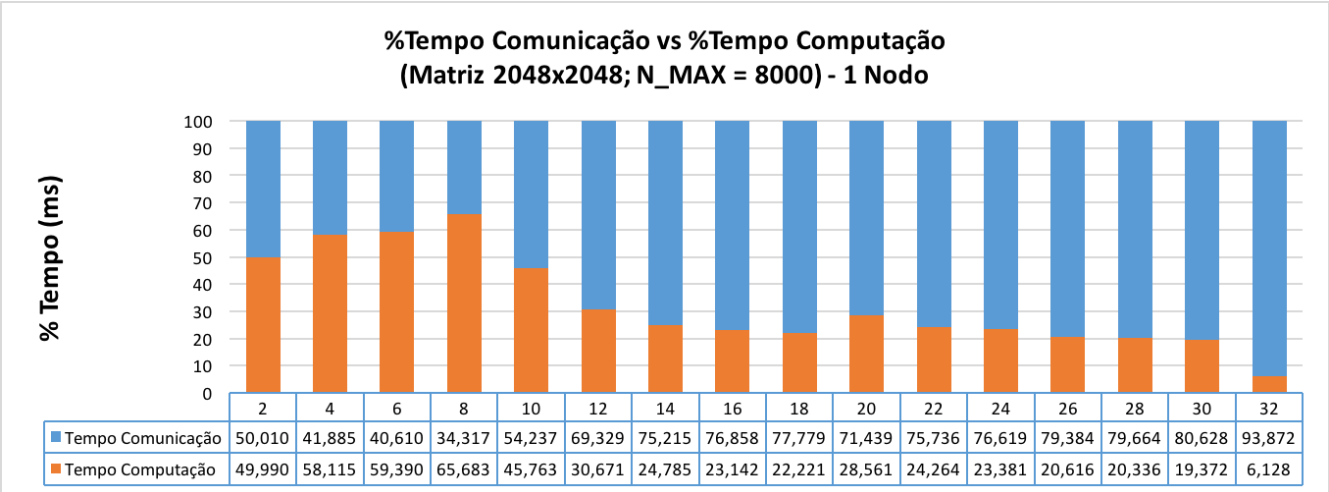
# C. %Tempo Comunicação VS %Tempo Computação - nodos 641 por Ethernet

## C.1. 1 Nodo



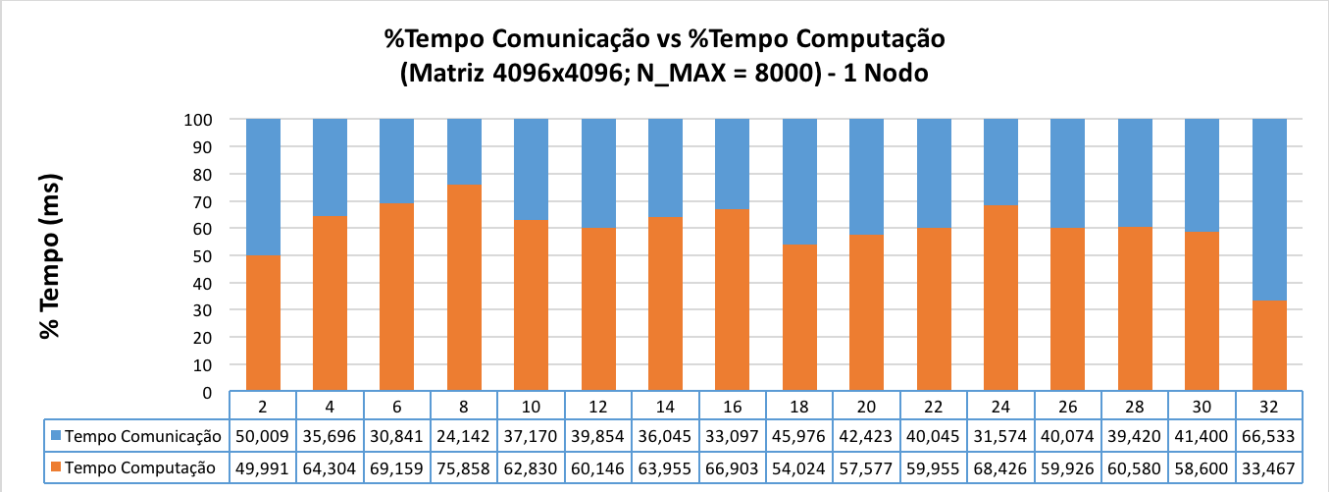

| | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 | 26 | 28 | 30 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Tempo Comunicação | 50,011 | 57,381 | 67,545 | 72,903 | 77,443 | 79,912 | 82,247 | 83,200 | 83,412 | 83,895 | 85,797 | 85,092 | 85,972 | 85,856 | 85,366 | 96,172 |
| Tempo Computação | 49,989 | 42,619 | 32,455 | 27,097 | 22,557 | 20,088 | 17,753 | 16,800 | 16,588 | 16,105 | 14,203 | 14,908 | 14,028 | 14,144 | 14,634 | 3,828 |

Figure 11: Tempo comunicação vs tempo computação em 1 nodo para matriz 1024x1024



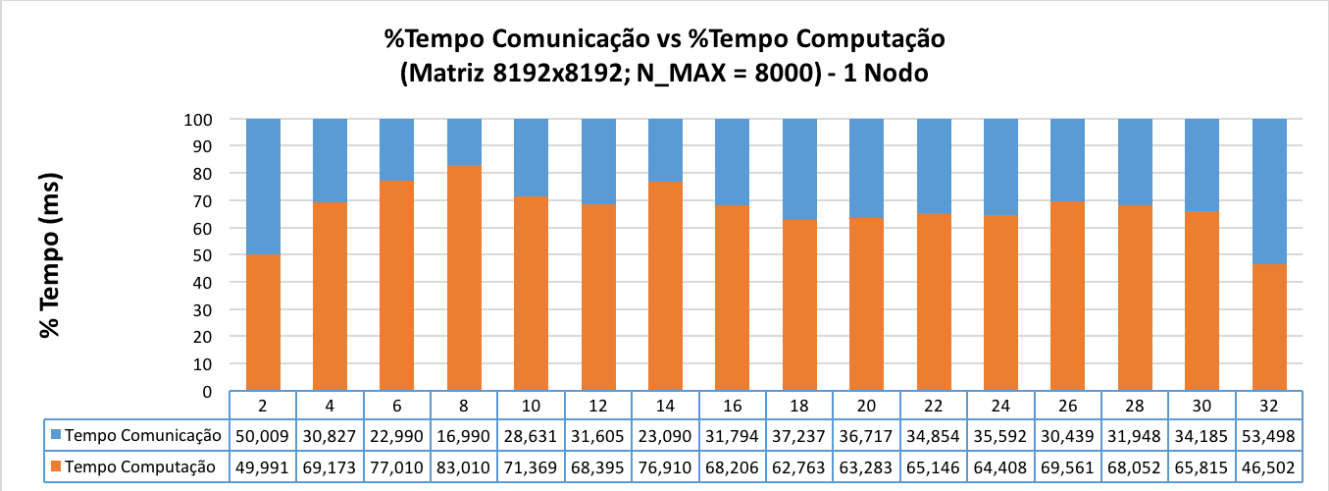

| | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 | 26 | 28 | 30 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Tempo Comunicação | 50,010 | 41,885 | 40,610 | 34,317 | 54,237 | 69,329 | 75,215 | 76,858 | 77,779 | 71,439 | 75,736 | 76,619 | 79,384 | 79,664 | 80,628 | 93,872 |
| Tempo Computação | 49,990 | 58,115 | 59,390 | 65,683 | 45,763 | 30,671 | 24,785 | 23,142 | 22,221 | 28,561 | 24,264 | 23,381 | 20,616 | 20,336 | 19,372 | 6,128 |

Figure 12: Tempo comunicação vs tempo computação em 1 nodo para matriz 2048x2048




| | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 | 26 | 28 | 30 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Tempo Comunicação | 50,009 | 35,696 | 30,841 | 24,142 | 37,170 | 39,854 | 36,045 | 33,097 | 45,976 | 42,423 | 40,045 | 31,574 | 40,074 | 39,420 | 41,400 | 66,533 |
| Tempo Computação | 49,991 | 64,304 | 69,159 | 75,858 | 62,830 | 60,146 | 63,955 | 66,903 | 54,024 | 57,577 | 59,955 | 68,426 | 59,926 | 60,580 | 58,600 | 33,467 |

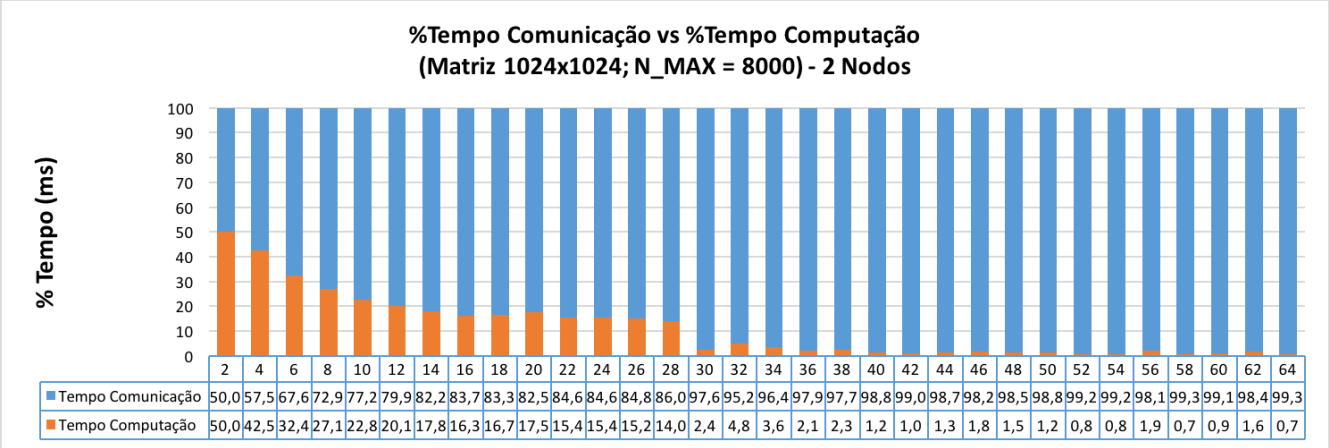
Figure 13: Tempo comunicação vs tempo computação em 1 nodo para matriz 4096x4096




| | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 | 26 | 28 | 30 | 32 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Tempo Comunicação | 50,009 | 30,827 | 22,990 | 16,990 | 28,631 | 31,605 | 23,090 | 31,794 | 37,237 | 36,717 | 34,854 | 35,592 | 30,439 | 31,948 | 34,185 | 53,498 |
| Tempo Computação | 49,991 | 69,173 | 77,010 | 83,010 | 71,369 | 68,395 | 76,910 | 68,206 | 62,763 | 63,283 | 65,146 | 64,408 | 69,561 | 68,052 | 65,815 | 46,502 |

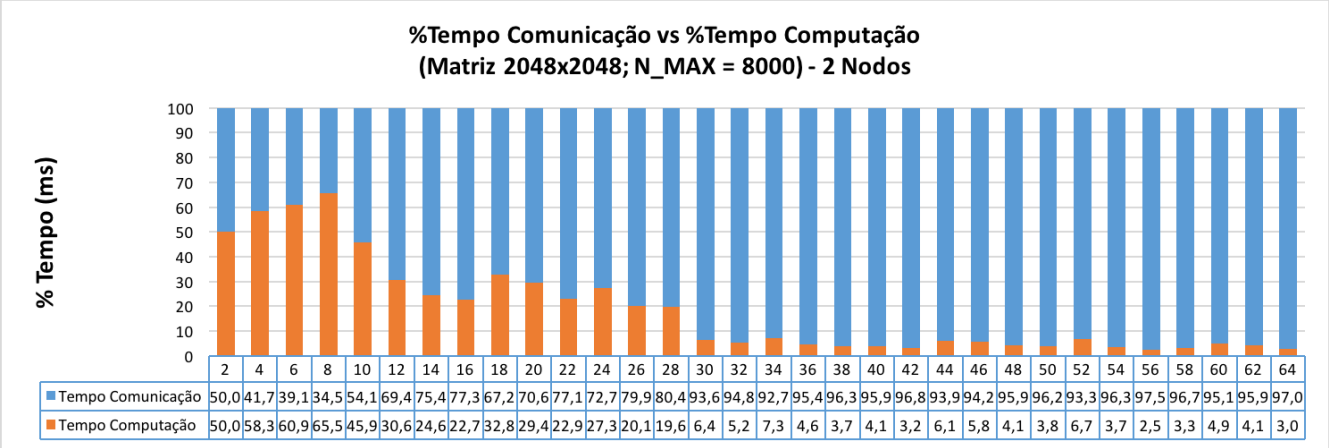
Figure 14: Tempo comunicação vs tempo computação em 1 nodo para matriz 8192x8192

## C.2. 2 Nodos



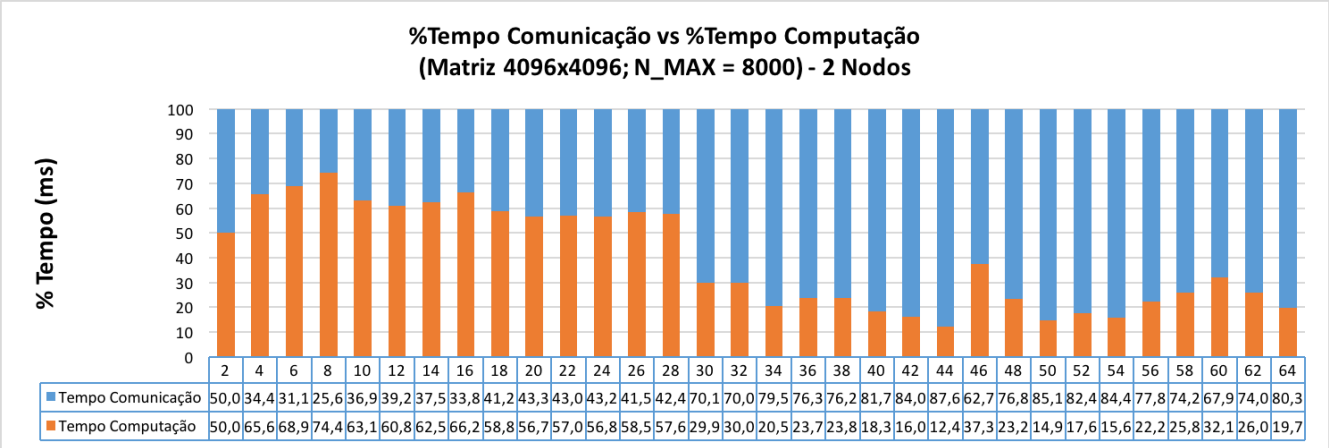

| | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 | 26 | 28 | 30 | 32 | 34 | 36 | 38 | 40 | 42 | 44 | 46 | 48 | 50 | 52 | 54 | 56 | 58 | 60 | 62 | 64 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Tempo Comunicação | 50,0 | 57,5 | 67,6 | 72,9 | 77,2 | 79,9 | 82,2 | 83,7 | 83,3 | 82,5 | 84,6 | 84,6 | 84,8 | 86,0 | 97,6 | 95,2 | 96,4 | 97,9 | 97,7 | 98,8 | 99,0 | 98,7 | 98,2 | 98,5 | 98,8 | 99,2 | 99,2 | 98,1 | 99,3 | 99,1 | 98,4 | 99,3 |
| Tempo Computação | 50,0 | 42,5 | 32,4 | 27,1 | 22,8 | 20,1 | 17,8 | 16,3 | 16,7 | 17,5 | 15,4 | 15,4 | 15,2 | 14,0 | 2,4 | 4,8 | 3,6 | 2,1 | 2,3 | 1,2 | 1,0 | 1,3 | 1,8 | 1,5 | 1,2 | 0,8 | 0,8 | 1,9 | 0,7 | 0,9 | 1,6 | 0,7 |

Figure 15: Tempo comunicação vs tempo computação em 2 nodos para matriz 1024x1024



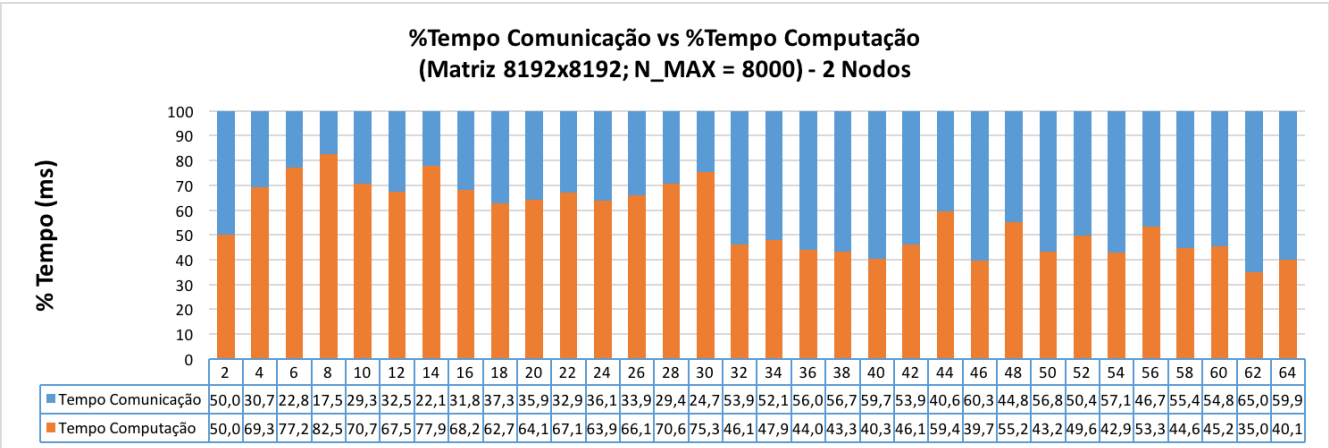

| | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 | 26 | 28 | 30 | 32 | 34 | 36 | 38 | 40 | 42 | 44 | 46 | 48 | 50 | 52 | 54 | 56 | 58 | 60 | 62 | 64 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Tempo Comunicação | 50,0 | 41,7 | 39,1 | 34,5 | 54,1 | 69,4 | 75,4 | 77,3 | 67,2 | 70,6 | 77,1 | 72,7 | 79,9 | 80,4 | 93,6 | 94,8 | 92,7 | 95,4 | 96,3 | 95,9 | 96,8 | 93,9 | 94,2 | 95,9 | 96,2 | 93,3 | 96,3 | 97,5 | 96,7 | 95,1 | 95,9 | 97,0 |
| Tempo Computação | 50,0 | 58,3 | 60,9 | 65,5 | 45,9 | 30,6 | 24,6 | 22,7 | 32,8 | 29,4 | 22,9 | 27,3 | 20,1 | 19,6 | 6,4 | 5,2 | 7,3 | 4,6 | 3,7 | 4,1 | 3,2 | 6,1 | 5,8 | 4,1 | 3,8 | 6,7 | 3,7 | 2,5 | 3,3 | 4,9 | 4,1 | 3,0 |

Figure 16: Tempo comunicação vs tempo computação em 2 nodos para matriz 2048x2048




| | 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 | 50,0 | 34,4 | 31,1 | 25,6 | 36,9 | 39,2 | 37,5 | 33,8 | 41,2 | 43,3 | 43,0 | 43,2 | 41,5 | 42,4 | 70,1 | 70,0 | 79,5 | 76,3 | 76,2 | 81,7 | 84,0 | 87,6 | 62,7 | 76,8 | 85,1 | 82,4 | 84,4 | 77,8 | 74,2 | 67,9 | 74,0 | 80,3 |
| Tempo Computação | 50,0 | 65,6 | 68,9 | 74,4 | 63,1 | 60,8 | 62,5 | 66,2 | 58,8 | 56,7 | 57,0 | 56,8 | 58,5 | 57,6 | 29,9 | 30,0 | 20,5 | 23,7 | 23,8 | 18,3 | 16,0 | 12,4 | 37,3 | 23,2 | 14,9 | 17,6 | 15,6 | 22,2 | 25,8 | 32,1 | 26,0 | 19,7 |

Figure 17: Tempo comunicação vs tempo computação em 2 nodos para matriz 4096x4096




| | 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 | 50,0 | 30,7 | 22,8 | 17,5 | 29,3 | 32,5 | 22,1 | 31,8 | 37,3 | 35,9 | 32,9 | 36,1 | 33,9 | 29,4 | 24,7 | 53,9 | 52,1 | 56,0 | 56,7 | 59,7 | 53,9 | 40,6 | 60,3 | 44,8 | 56,8 | 50,4 | 57,1 | 46,7 | 55,4 | 54,8 | 65,0 | 59,9 |
| Tempo Computação | 50,0 | 69,3 | 77,2 | 82,5 | 70,7 | 67,5 | 77,9 | 68,2 | 62,7 | 64,1 | 67,1 | 63,9 | 66,1 | 70,6 | 75,3 | 46,1 | 47,9 | 44,0 | 43,3 | 40,3 | 46,1 | 59,4 | 39,7 | 55,2 | 43,2 | 49,6 | 42,9 | 53,3 | 44,6 | 45,2 | 35,0 | 40,1 |

Figure 18: Tempo comunicação vs tempo computação em 2 nodos para matriz 8192x8192

# D. Ganhos Comunicação Myrinet Nodo 662 vs Ethernet Nodo 641

| Matriz | PPN | T. Comun. Total - 641 Ether. | T. Comun. Total - 662 Myr. | Ganho Myr. vs Ether. |
|---|---|---|---|---|
| 1024*1024 | 2 | 16244,796 | 17847,358 | -8,98% |
| | 4 | 22101,625 | 24051,835 | -8,11% |
| | 6 | 33891,911 | 37036,303 | -8,49% |
| | 8 | 43808,286 | 47534,77 | -7,84% |
| 2048*2048 | 2 | 69832,653 | 66579,094 | 4,89% |
| | 4 | 59607,498 | 55513,959 | 7,37% |
| | 6 | 78879,511 | 79662,654 | -0,98% |
| | 8 | 81148,511 | 103415,35 | -21,53% |
| 4096*4096 | 2 | 264473,178 | 289182,2 | -8,54% |
| | 4 | 177406,828 | 190206,462 | -6,73% |
| | 6 | 202302,35 | 219817,146 | -7,97% |
| | 8 | 207787,87 | 242756,98 | -14,40% |
| 8192*8192 | 2 | 999373,967 | 1103997,19 | -9,48% |
| | 4 | 552507,78 | 563098,476 | -1,88% |
| | 6 | 550700,772 | 572589,514 | -3,82% |
| | 8 | 553651,455 | 639264,802 | -13,39% |

Table 1: Comparação do ganho de utilização de comunicação Myrinet vs Ethernet em nodos 662 e 641, respetivamente. Valores em milissegundos.

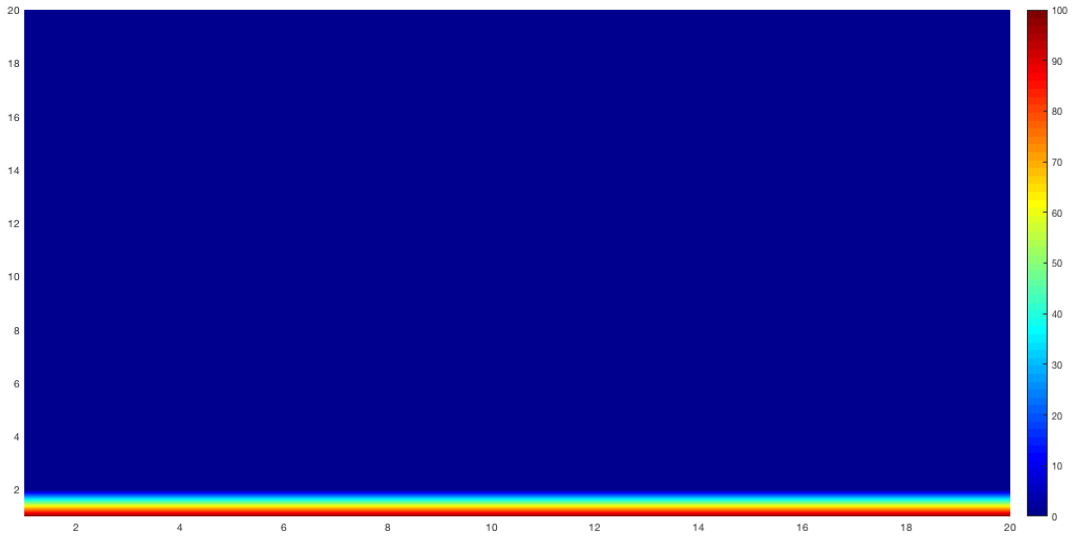
# E. Ganhos de Computação Nodo 662 vs Nodo 641

| Matriz | PPN | T. Comput. Total - 641 | T. Comput. Total - 662 | Ganho Comput. 662 vs 641 |
|---|---|---|---|---|
| 1024*1024 | 2 | 16237,91 | 17837,925 | -8,97% |
| | 4 | 16415,47 | 17855,649 | -8,07% |
| | 6 | 16285,009 | 18100,977 | -10,03% |
| | 8 | 16282,971 | 17922,032 | -9,15% |
| 2048*2048 | 2 | 69804,441 | 66548,595 | 4,89% |
| | 4 | 82704,662 | 57353,432 | 44,20% |
| | 6 | 115355,076 | 57475 | 100,70% |
| | 8 | 155315,673 | 57575,898 | 169,76% |
| 4096*4096 | 2 | 264377,971 | 289071,646 | -8,54% |
| | 4 | 319588,375 | 298039,605 | 7,23% |
| | 6 | 453651,712 | 306990,91 | 47,77% |
| | 8 | 652903,875 | 331775,577 | 96,79% |
| 8192*8192 | 2 | 999008,942 | 1103575,016 | -9,48% |
| | 4 | 1239795,337 | 1133953,991 | 9,33% |
| | 6 | 1844721,597 | 1226442,135 | 50,41% |
| | 8 | 2705090,791 | 1447961,551 | 86,82% |

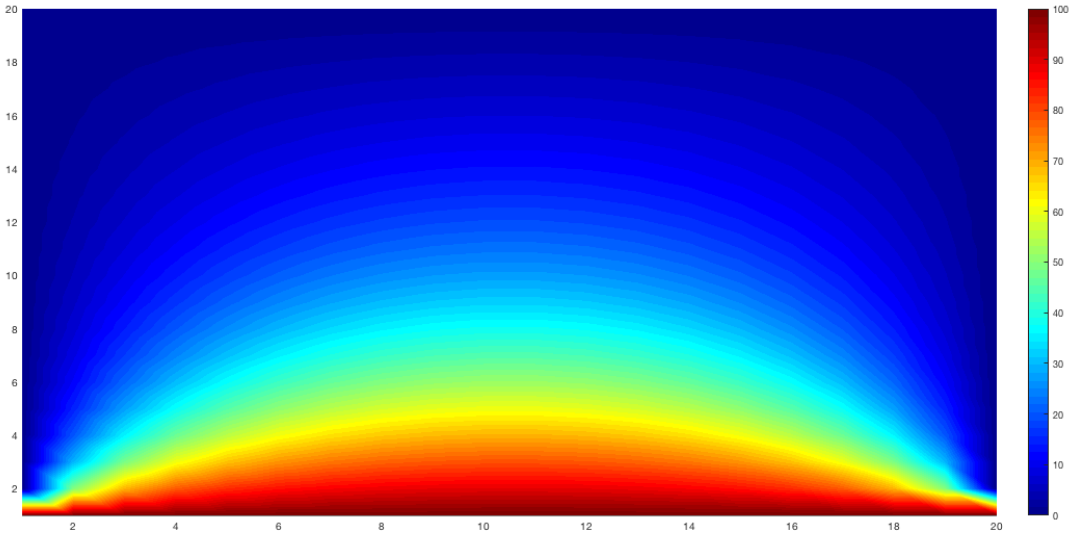
Table 2: Comparação do ganho de computação na utilização de um nodo 662 vs um nodo 641. Valores em milissegundos.

# F. Representação Gráfica do Resultado

Apresenta-se abaixo uma representação gráfica da matriz inicial e da matriz final do processo de difusão de calor efetuado 8000 vezes, para uma matriz de tamanho 20x20 (equivalente para o código sequencial e paralelo):




(a) Matriz inicial 20x20




(b) Matriz final 20x20, para 8000 N_MAX iterações

Figure 19: Representação gráfica do processo de difusão de calor para uma matriz de 20x20 com 8000 N_MAX iterações