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 Híbrido - OpenMP & OpenMPI ®

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

Braga, 8 de Janeiro de 2017

Conteúdo

1. Caso de Estudo

O caso de estudo desenvolvido vai em seguimento dos dois trabalhos já desenvolvidos. O presente trabalho tem por objetivo tirar partido em conjunto de diretivas **OpenMP** e **OpenMPI**, com memória partilhada e distribuída respetivamente, por forma a paralelizar o processo de difusão de calor em N_MAX iterações.

Tal como nos trabalhos anteriores, 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).

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. As bordas da matriz permanecem imutáveis ao longo do processe de difusão de calor.

2. Caracterização da Plataforma de Teste

A plataforma de teste utilizada no Search6 possui nodos equipados com *dual-socket*. Foi utilizado o nodo de computação 641 cujas características principais podem ser consultadas no anexo A. Foi ainda utilizada a rede Ethernet - 1Gbps - como meio de comunicação entre nodos.

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:

- 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 $mp_threads
```

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 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 nos trabalhos anteriores, 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.

O numero máximo de iterações é **8000** (valor utilizado no trabalho de OpenMPI). Em relação aos tamanhos de matrizes escolhidos, ao contrário dos trabalhos anteriores, que foram utilizados tamanhos de 1024*1024, 2048*2048, 4096*4096 e 8192*8192, foram apenas escolhidos os tamanhos **1024*1024** e **8192*8192**.

Foram escolhidos estes tamanhos de matriz uma vez que tendo em consideração os resultados obtidos no trabalho de OpenMPI, a matriz 8192*8192 foi a que obteve melhores ganhos tanto quando são utilizados 1 ou 2 nodos e a matriz 1024*1024 foi a que obteve piores ganhos (ocorreram perdas) independentemente do numero de nodos utilizados. Procura-se assim obter ainda mais ganhos na matriz 8192*8192 e melhorar os ganhos da matriz 1024*1024 para que deixem de ocorrer perdas.

Os restantes tamanhos de matrizes foram descartados uma vez que a partir do trabalho de OpenMPI foi possível observar que a matriz 2048*2048 tem um comportamento semelhante à matriz 1024*1024 e a matriz 4096*4096 tem um comportamento semelhante à matriz 8192*8192.

3. Solução Híbrida

Para a solução híbrida foram mantidas todas as considerações tomadas no trabalho de OpenMPI, nomeadamente a arquitetura geral (anexo B), distribuição dos dados e comunicação entre processos. A solução híbrida passou então apenas pelo acréscimo de primitivas OpenMP por forma a melhorar o tempo de computação da solução.

Tal como no trabalho de OpenMP, as primitivas utilizadas foram aplicadas à função responsável pelo calculo da difusão de calor:

```
void iterate(float **M_New, float **M_Old, int N, int offset, int numLines, int threads_OMP) {
 int startRow;
 int endRow;
 int i, j;

 if(offset == 0) startRow = offset + 1;
 else startRow = offset - 1;
 if((offset + numLines) == N) endRow = N-1;
 else endRow = offset + numLines + 1;

 #pragma omp parallel num_threads(threads_OMP)
 {
  #pragma omp parallel shared(M_New, M_Old) private(i, j)
  {
   #pragma omp for
   for(i = startRow; i < endRow; i++) {
    for(j = 1; j < N-1; j++) {
     M_Old[i][j] = M_New[i][j];
    }
   }
  }
 }

 /*************************************************************/

 if(offset == 0) startRow = offset + 1;
 else startRow = offset;
 if((offset + numLines) == N) endRow = N-1;
 else endRow = offset + numLines;

 #pragma omp parallel num_threads(threads_OMP)
 {
  #pragma omp parallel shared(M_New, M_Old) private(i, j)
  {
   #pragma omp for
   for(i = startRow; i < endRow; i++) {
    for(j = 1; j < N-1; j++) {
     M_New[i][j] = (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;
    }
   }
  }
 }
}
```

As primitivas utilizadas são iguais às primitivas utilizadas no trabalho de OpenMP. Estão a ser criadas um conjunto de threads a partir da diretiva #pragma omp parallel num_threads(threads_OMP). É utilizado ainda shared(M_New, M_Old) para que ambas as matrizes sejam partilhadas pelas várias threads e ainda private(i,j) para que cada thread possua o seu próprio valor das variáveis i, j, que são responsáveis por fazer o controlo dos ciclos.

Existe ainda mais uma diretiva, #pragma omp for, que é colocada antes dos ciclos *for* externos que leva à paralelização desse ciclo.

4. Medições

Tal como no trabalho de OpenMPI, foram utilizados 1 e 2 nodos 641. Tendo em conta as especificações da máquina 641 utilizada, as medições ideais a serem efetuadas seria a utilização de até 32 processos MPI com até 32 threads OpenMP, para 1 nodo de computação, e a utilização de até 64 processos MPI com até 32 threads OpenMP, para 2 nodos de computação.

Contudo, o grupo não conseguiu recolher dados completos dessas medições devido ao fator de tempo necessário para que o código seja executado. Foram submetidos *jobs* de vários dias (até 5 dias) para a execução das medições de cada matriz, contudo, esses períodos de tempo não foram suficientes para obter resultados completos das medições. Em alguns casos ocorriam até erros relacionados com a disponibilidade de recursos: *"libgomp: Thread creation failed: Resource temporarily unavailable"* ao longo das medições.

Tendo isto em conta, e depois de várias submissões de *jobs* para tentar contornar o problema, foram conseguidos resultados para as medições:

- 1 Nodo - Matriz 1024*1024
  - De 2 a 20 processos MPI (de dois em dois)
  - De 2 a 32 threads OpenMP (de dois em dois)
- 1 Nodo - Matriz 8192*8192
  - De 4 a 20 processos MPI (de quatro em quatro)
  - De 2 a 32 threads OpenMP (de dois em dois)
- 2 Nodos - Matriz 8192*8192
  - De 4 a 40 processos MPI (de quatro em quatro)
  - 16 e 32 threads OpenMP

Apesar dos problemas acima referidos (principalmente o tempo de execução elevado), as medições retiradas permitem tirar conclusões tendo em conta os resultados obtidos no trabalho de OpenMPI. Na secção seguinte vai ser explicado com detalhe essas conclusões e o relacionamento com os resultados obtidos no trabalho de OpenMPI.

4.1. Metodologia e Condições de Medição

O tempo de execução 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
- Ao contrário dos trabalhos anteriores, que foram recolhidas 5 medições e aplicada a mediana dos tempos, agora foi recolhida apenas 1 medição devido ao tempo elevado de execução.

5. Avaliação da Versão Híbrida

5.1. Evolução do Tempo de Execução - nodos 641 por Ethernet

Por forma a perceber o impacto da associação de threads OpenMP a cada processo OpenMPI, no tempo de execução total, foi feita uma análise desse impacto. Quando são utilizados mais que 32 processos MPI estão a ser utilizados 2 nodos de computação.

A baixo são apresentados gráficos da evolução do tempo de execução para cada tamanho de matriz considerado e numero de nodos de computação utilizados. Deve-se ter especial atenção na escala de cada um. Entradas que estejam a branco significa que a execução do programa terminou antes de obter o resultado.

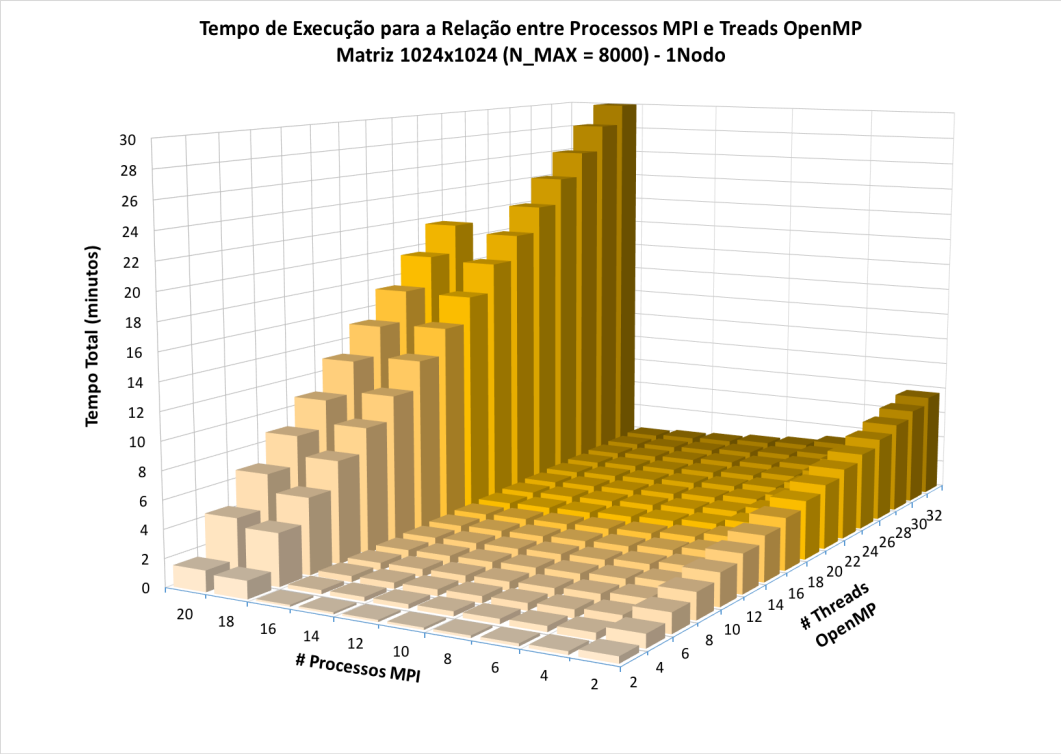

Figure 1: Evolução do tempo de execução em 1 nodo para matriz 1024*1024 para cada processo MPI e respetivas threads OpenMP

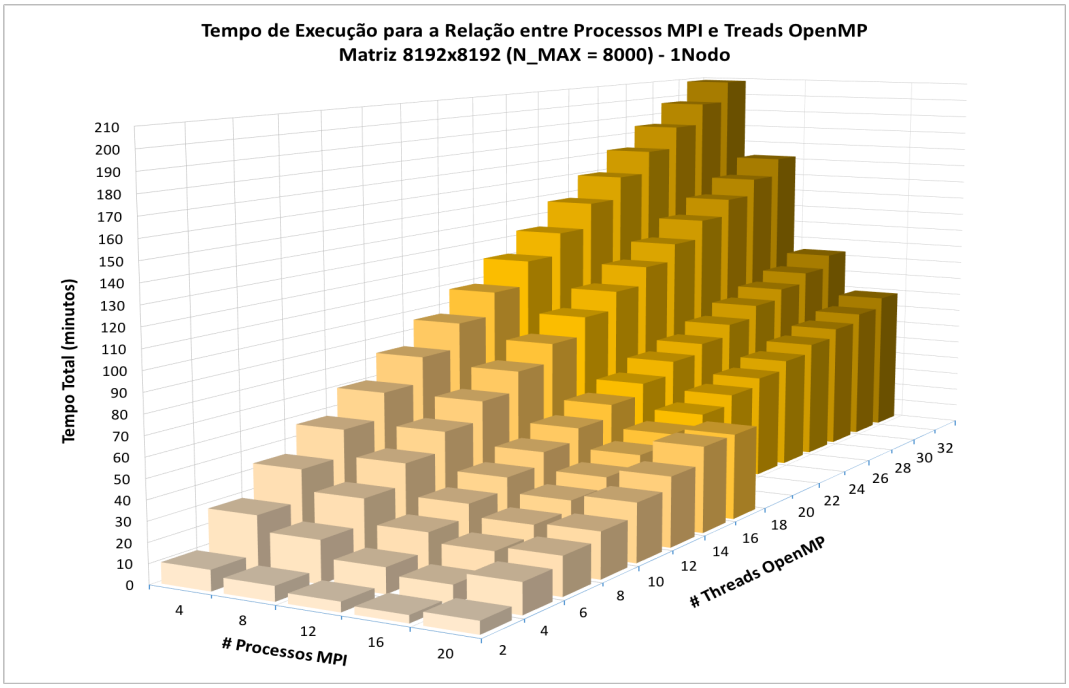

Figure 2: Evolução do tempo de execução em 1 nodo para matriz 8192*8192 para cada processo MPI e respetivas threads OpenMP

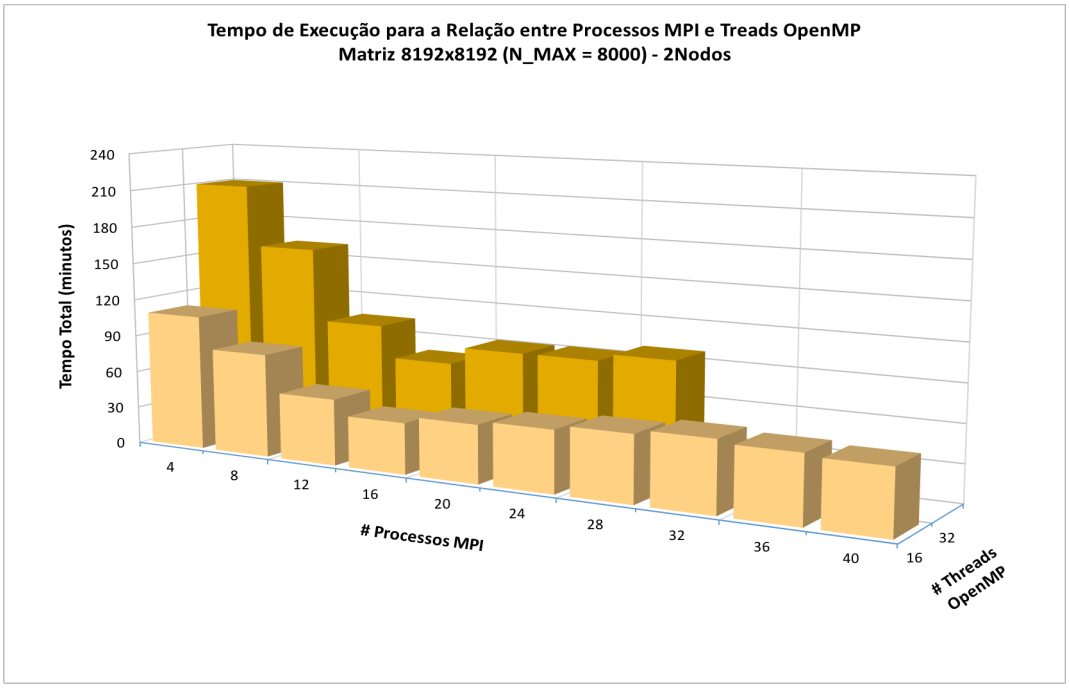

Figure 3: Evolução do tempo de execução em 2 nodos para matriz 8192*8192 para cada processo MPI e respetivas threads OpenMP

Como é possível observar, à medida que são utilizados até 16 processos MPI, o tempo de execução diminui de uma forma geral (com algumas exceções) para qualquer tamanho de matriz. Contudo, para cada processo MPI à medida que são utilizados mais processos OpenMP, o tempo de execução aumenta exponencialmente, ao contrário do que deveria acontecer teoricamente.

A partir da utilização de 16 processo MPI, o tempo de execução começa a aumentar, para a matriz 1024*1024 este aumento é muito significativo, uma vez que é a partir dos 16 processos que passam a ser utilizados os dois processadores do nodo.

O aumento exponencial do tempo de execução que ocorre à medida que são utilizados mais processos OpenMP deve-se à possível inviabilidade de dados para threads no mesmo processador e a overhead provocado por data races e troca de mensagens entre processos e threads no mesmo nodo.

Quando são utilizados dois nodos de computação (mais que 32 processos OpenMP) todo o overhead da solução MPI, efetuada no trabalho anterior, é agora exponenciado pelo overhead da utilização das primitivas OpenMP.

5.2. SpeedUp's - nodos 641 por Ethernet

Apresentam-se a baixo os gráficos dos ganhos obtidos, para cada tamanho de matriz considerado e numero de nodos de computação utilizados entre a versão sequencial e híbrida.

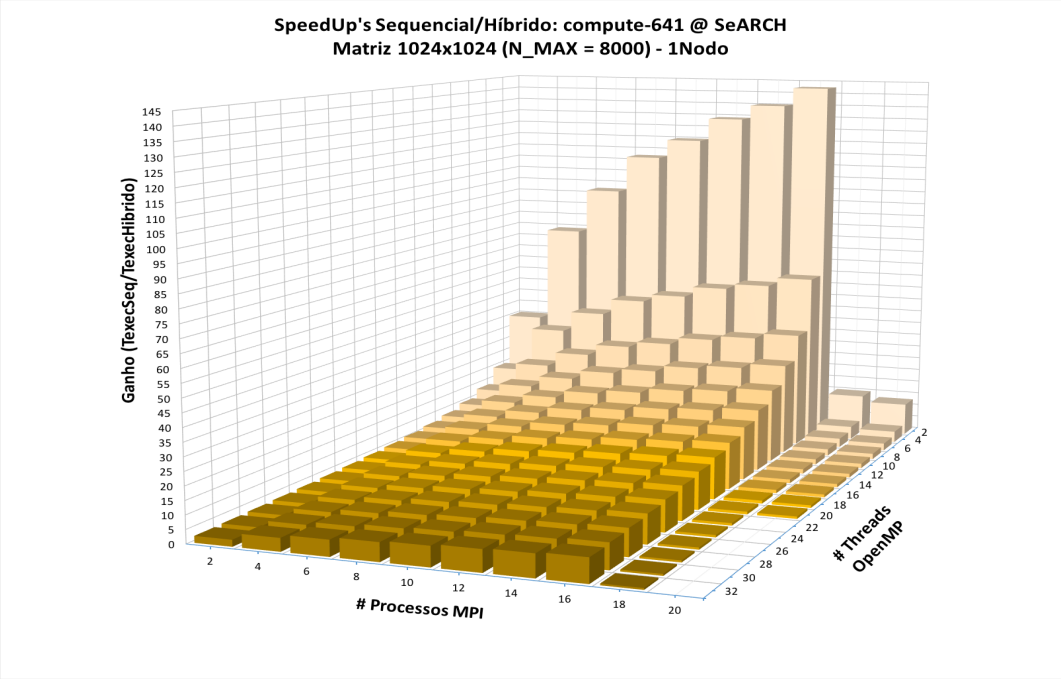

Figure 4: SpeedUp's da matriz 1024*1024 em 1 nodo de computação para cada processo MPI e respetivas threads OpenMP

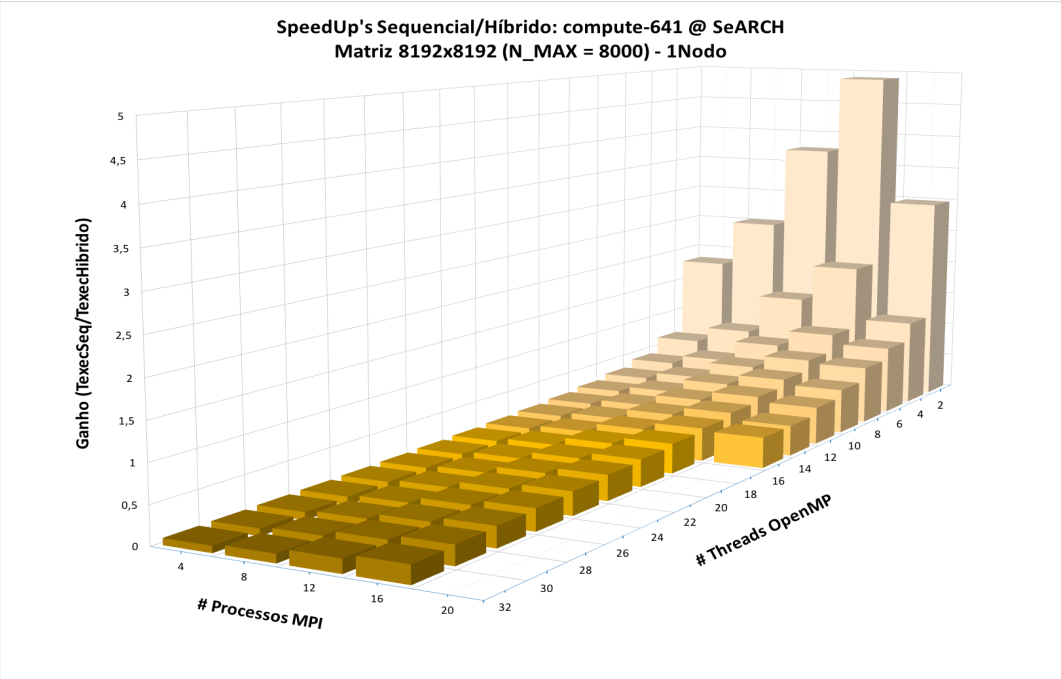

Figure 5: SpeedUp's da matriz 8192*8192 em 1 nodo de computação para cada processo MPI e respetivas threads OpenMP

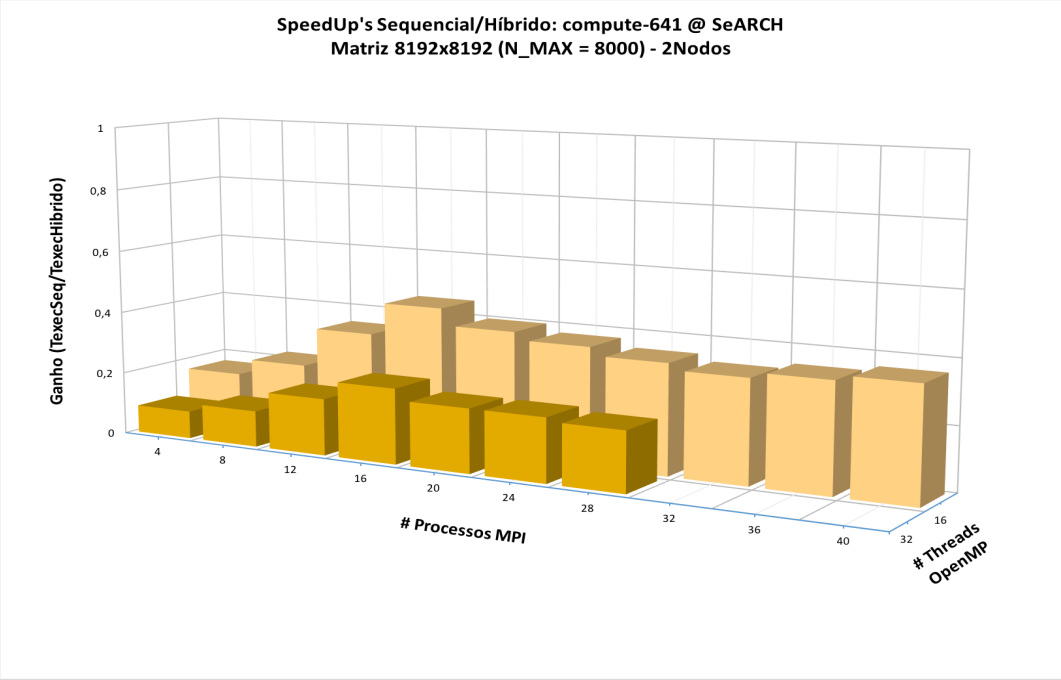

Figure 6: SpeedUp's da matriz 8192*8192 em 2 nodos de computação para cada processo MPI e respetivas threads OpenMP

Para a matriz 1024*1024 são obtidos ganhos crescentes até aos 16 processos MPI. Após os 16 processos os ganhos decrescem acentuadamente (que já foi explicado na secção anterior). Contudo, ao contrário do que seria expectável, os maiores ganhos acontecem para um numero de threads OpenMP baixo, diminuindo à medida que são utilizados mais processos OpenMP.

Os ganhos obtidos na utilização de até 16 processos MPI e 2 threads OpenMP são de facto muito bons uma vez que se encontram entre 100 e 145.

A matriz 8192*8192 tem o mesmo comportamento à medida que se utilizam mais processos MPI e OpenMP. Na utilização de apenas 1 nodo de computação, para 2 e 4 threads OpenMP ainda são obtidos ganhos, contudo, para valores superiores começam a ocorrer perdas e na utilização de 2 nodos de computação, não são obtidos quaisquer ganhos.

5.3. SpeedUp's OpenMPI vs Híbrido - nodos 641 por Ethernet

Uma vez que no presente trabalho foram adicionadas primitivas OpenMP ao código em OpenMPI anteriormente desenvolvido, torna-se necessário comparar os ganhos obtidos nas duas soluções - OpenMPI e Híbrida.

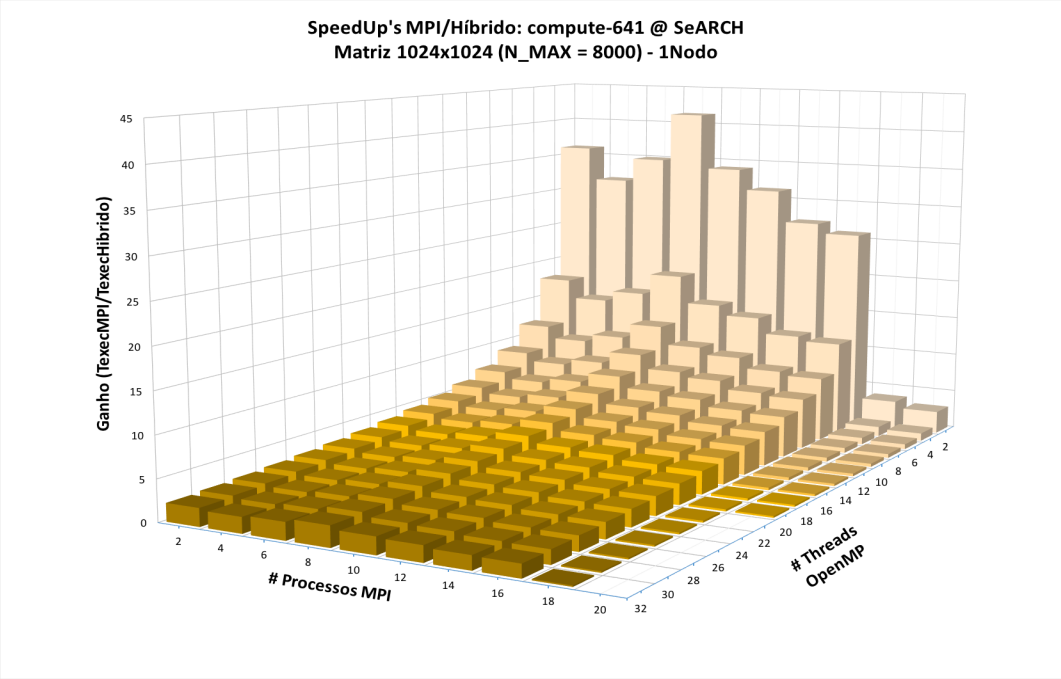

Figure 7: SpeedUp's MPI/Híbrido da matriz 1024*1024 em 1 nodo de computação para cada processo MPI e respetivas threads OpenMP

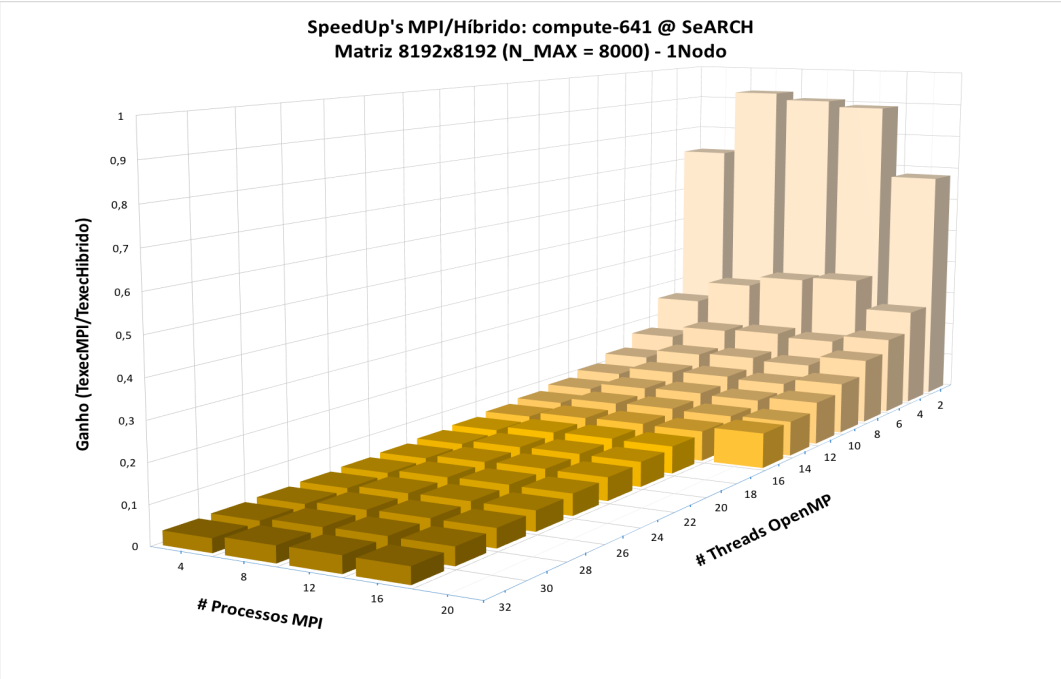

Figure 8: SpeedUp's MPI/Híbrido da matriz 8192*8192 em 1 nodo de computação para cada processo MPI e respetivas threads OpenMP

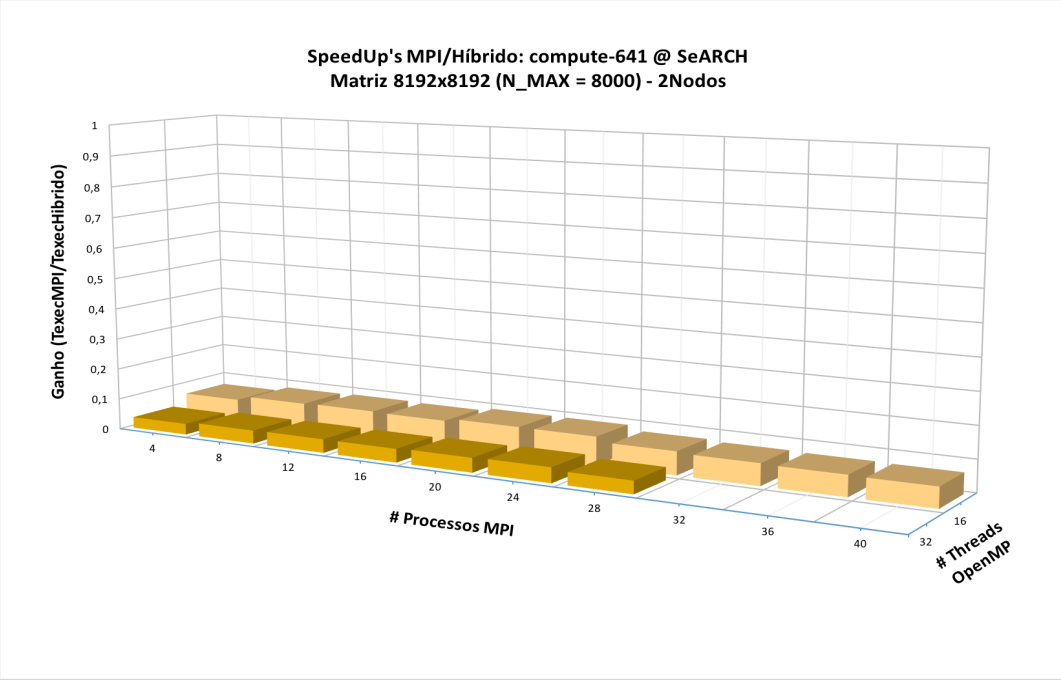

Figure 9: SpeedUp's MPI/Híbrido da matriz 8192*8192 em 2 nodos de computação para cada processo MPI e respetivas threads OpenMP

Tal como nos resultados apresentados até agora, os ganhos aumentam até à utilização de 16 processos MPI e decrescem à medida que são utilizadas mais threads OpenMP.

Para a matriz 1024*1024 existem sempre ganhos, até à utilização de 16 processos MPI, para qualquer numero de threads OpenMP, contudo, o maior ganho acontece para 2 threads OpenMP.

Na matriz 8192*8192 não foram obtidos quaisquer ganhos em comparação com a versão MPI desenvolvida no trabalho anterior, quer sejam usados 1 ou 2 nodos de computação. De facto, na utilização de 2 nodos de computação, é possível observar que os ganhos são muito inferiores do que na utilização de apenas 1 nodo de computação.

6. Conclusão

A utilização de paradigmas de computação paralela é um método muito eficaz para conseguir paralelizar um código sequencial. Contudo, a sua aplicabilidade nem sempre é trivial para todos os casos. Os resultados expectáveis/desejáveis muito raramente são obtidos devido a várias limitações de hardware bem como à implementação de código paralelo.

De facto, o exposto acima é bem visível nos resultados obtidos neste trabalho. Seria de esperar que à medida que fossem adicionadas threads de processamento OpenMP aos processos MPI, o ganho fosse cada vez maior, contudo, ocorreu exatamente o contrário. Tal deve-se ao elevado overhead de comunicação e distribuição/troca de dados entre os vários processos/threads.

Tendo em conta os resultados obtidos, para matrizes de grandes dimensões - 8192*8192 - deve ser mantida a implementação OpenMPI desenvolvida no trabalho anterior uma vez que para esta dimensão de matriz, no algoritmo híbrido, não foram obtidos quaisquer ganhos em comparação com OpenMPI. Contudo, para matrizes de pequena dimensão - 1024*1024 - foram obtidos ganhos consideravelmente superiores na versão híbrida em comparação com a versão OpenMPI, especialmente para numero de threads OpenMP baixos.

Globalmente, dos resultados obtidos nos três trabalhos desenvolvidos, pode-se concluir que para matrizes de grande dimensão deve ser utilizada uma solução OpenMPI com 2 nodos de processamento, que é a que apresenta maiores ganhos. Para matrizes de pequena dimensão, os resultados do segundo trabalho mostraram que era a versão OpenMP com maiores ganhos. Como na versão híbrida desenvolvida foram obtidos ganhos muito bons para a matriz 1024*1024 quer em comparação com a versão sequencial como OpenMPI, deveria ser agora efetuada uma compararção dos ganhos híbridos com os ganhos em OpenMP, por forma a determinar qual solução seria a mais adequada para matrizes de pequena dimensão.

A. Máquina de Teste

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

B. Arquitetura Geral da Solução em OpenMPI

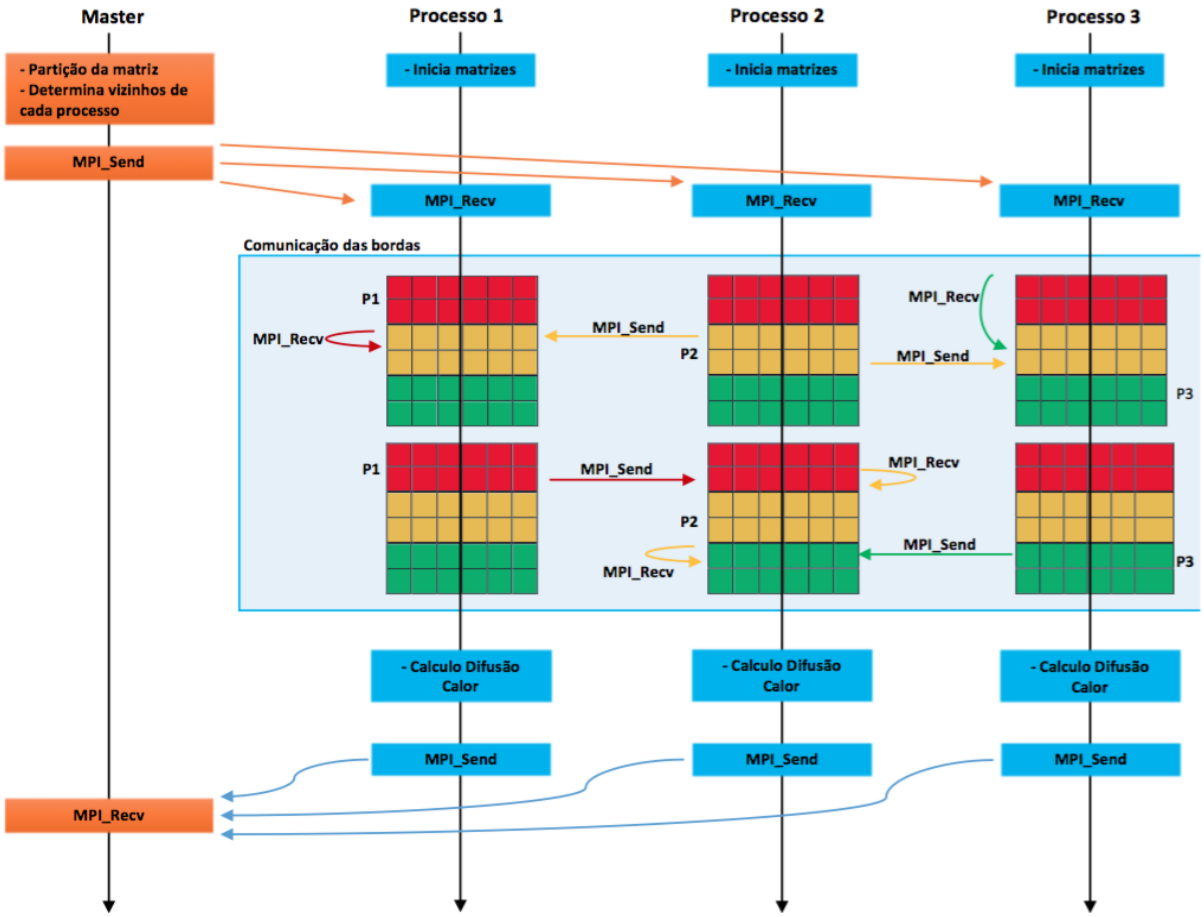

Figure 10: Arquitetura geral da solução em OpenMPI