Computação Científica de Alto Desempenho II
Projeto Final: Versões sequencial, OpenMP e MPI de um programa que resolve um tipo de equação de calor

Luis Armando Quintanilla Villon[1]

[1]Instituto de Física - UFF

luisvillon@id.uff.br

Resumo. *Neste trabalho desenvolveu-se versões paralelas OpenMP e MPI de um programa que resolve um tipo de equação de calor. Para tal fim, a versões sequencial foi previamente aprimorada mediante flags específicos e técnicas de otimização de software. O programa paralelo foi executado em 4 computadores diferentes. Os resultados mostram que em um nó computacional é possível atingir, no melhor dos casos experimentados da versão OpenMP, uma eficiência próxima de 87%, e um speedup de 7.61x. Na versão MPI, os melhores obtidos foram: eficiência de 79.37% e speedup de 4.85x. Não foi possível extrair mais desempenho utilizando mais de um nó computacional.*

Abstract. *In this work, parallel OpenMP and MPI versions of a program that solve a type of heat equation were developed. To this end, the sequential version was previously enhanced through specific flags and software optimization techniques. The parallel program was run on 4 different computers. The results show that in a computational node it is possible to achieve, in the best of the cases experienced in the OpenMP version, an efficiency close to 87%, and a speed up of 7.61x. In the MPI version, the best results were: efficiency of 79.37% and speed up of 4.85x. It was not possible to extract more performance using more than one computational node.*

1. Introdução

A equação de calor, que é representada por uma equação diferencial parcial, possui grande importância na matemática, física e engenharia. Em particular, neste trabalho se considerou implementar um programa que resolve numericamente um tipo de equação de calor. Na sequencia, são relatados as diferentes seções que compõem este projeto.

Na seção 2 são descritos os objetivos deste trabalho. O método utilizado para resolver a equação é detalhado na seção 3 e a sua implementação na seção 4. O tempo de execução, com e sem flags, são mostrados na seção 5. A identificação de gargalos e as posteriores técnicas de otimização de software realizadas são relatadas na seção 6. A comparação dos tempos de execução, do programa base e otimizado, são exibidos na seção 7. Nas seções 8 e 9 a implementação das versões OpenMP e MPI são descritas respectivamente, e o benchmark com a correspondente análise de desempenho são exibidos na seção 10. Na seção 11 é mostrada a maneira que o programa paralelo pode validado. Finalmente, as conclusões finais são relatados na seção 12.

2. Objetivos

Os objetivos deste trabalho são os seguintes:

- Realizar uma avaliação de desempenho de um programa sequencial, com e sem uso flags.
- Conhecer as diferentes técnicas possíveis de diagnóstico de gargalos para logo depois otimizar a implementação.
- Quantificar qual seria o possível ganho de desempenho que poderia ser aproveitado paralelizando o programa proposto, usando as versões paralelas OpenMP e MPI.
- Utilizar várias arquiteturas para análise de desempenho do programa paralelo, incluindo nós computacionais do Supercomputador Santos Dumont.
- Ganhar experiência na utilização de métricas conhecidas em HPC, comparando versões OpenMP e MPI de um programa em particular.

3. Metodologia

3.1. Descrição

Para este trabalho se considerou em utilizar uma PDE (differential partial equation) elíptica que descreve a transferência de calor em um objeto sólido com geração de calor [Vasconcelos et al. 2015]:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + S_P T + S_C = 0 \tag{1}$$

Onde $S_P T + S_C$ é o termo que representa a geração de calor.

As condições de contorno avaliadas (Figura 1a) mostram um material com algumas extremidades mantidas a uma temperatura constante (de 50ºC e 100ºC). As derivadas parciais nulas implicam isolamento térmico. A extremidade direita com derivada parcial constante representa a exposição do material a uma corrente de fluido onde há troca de calor por convecção.

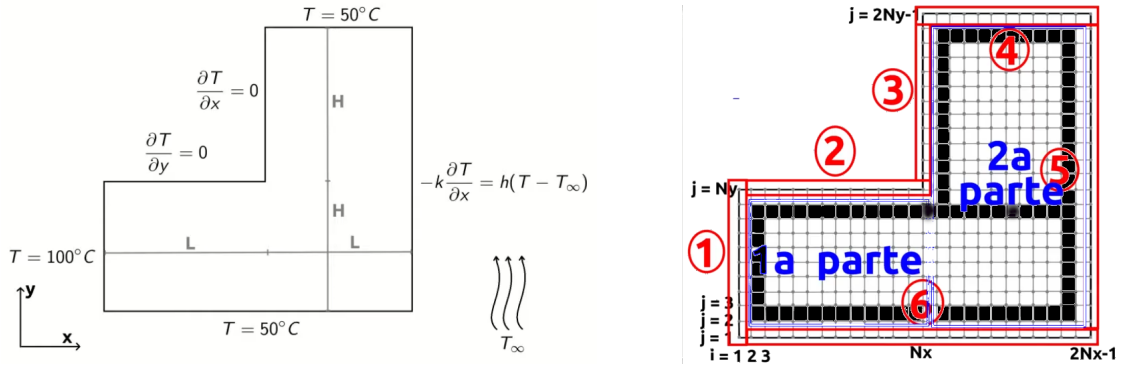
O problema consiste em encontrar a temperatura T do material descrito em regime estacionário. Para tanto, foi utilizado o método das diferenças finitas com o **método de Jacobi** para sistemas lineares.

3.2. Discretização do domínio de solução

Adotando o método das diferenças finitas [Scherer 2017], se considerou uma malha bidimensional típica (Figura 2), onde são representados o domínio de dependência e de influência, para cada ponto P do domínio de solução. A solução em todos os pontos precisam ser obtidas simultaneamente e as condições de contorno devem ser discretizadas adequadamente.

Optou-se por considerar que a largura L e a altura H do material (Figura 1a) sejam divididos em $N_x - 1$ e $N_y - 1$ partes pequenas respectivamente (Figura 1b). Deste modo, assumiu-se que :

$$dx = \Delta x = \frac{L}{N_x - 1}, \quad dy = \Delta y = \frac{H}{N_y - 1} \tag{2}$$



(a) Condições de contorno - modelo matemático

(b) Contornos enumerados de 1 a 6

Figura 1. Condições de contorno

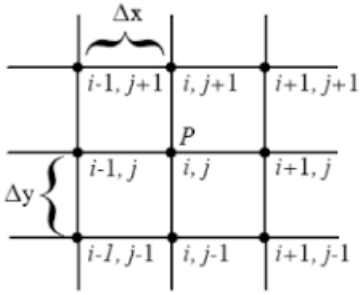


Figura 2. Ponto P - Malha bidimensional

Ademais, se observa que o material (Figura 1a) pode ter, no máximo, $2L$ de largura e $2H$ de altura. Portanto, o domínio discreto a da solução é definido da forma:

$$x(i) = (i)\Delta x, i = 0, 1, 2, 3, ..., 2N_x - 2 \tag{3}$$

$$y(j) = (j)\Delta y, j = 0, 1, 2, 3, ..., 2N_y - 2 \tag{4}$$

3.3. Discretização da equação elíptica e das condições de contorno

Da equação (1), e segundo o critério de 4 pontos cartográficos para a discretização pelo método das diferenças finitas, obtém-se:

$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta y^2} + S_P T + S_C = 0$$

Depois de algumas manipulações algébricas:

$$T_{i,j-1} + T_{i+1,j} + A_N T_{i,j+1} + A_S T_{i,j-1} + A_P T_{i,j} = S_U$$

$$T_{i,j} = -\frac{1}{A_P}(-S_U + T_{i,j-1} + T_{i+1,j} + A_N T_{i,j+1} + A_S T_{i,j-1}) \tag{5}$$

Onde:

$$\begin{aligned} A_N &= A_S = \alpha^2 = \frac{\Delta x^2}{\Delta y^2} \\ A_P &= -2 - 2\alpha^2 + S_P \Delta x^2 \\ S_U &= -S_C \Delta x^2 \end{aligned} \tag{6}$$

Assumiram-se os seguintes valores para os parâmetros deste problema:

$$\begin{aligned} H = L &= 0.1m \\ S_P &= -12m^{-2} \\ S_C &= 12500K/m^2 \\ k &= 150W/mK \\ h &= 450W/m^2K \\ T_\infty &= 0ºC \end{aligned} \tag{7}$$

4. Implementação - Fluxograma do programa

Na sequência é mostrado o fluxograma que descreve a estratégia geral para resolver o problema (Figura 3). Primeiramente, é necessário inicializar todos os coeficientes e parâmetros envolvidos. Logo depois é definida uma matriz auxiliar, $Mvelha$, com valores iniciais de fila e coluna aleatórios.

A partir daqui é utilizado um processo iterativo usando os pontos correspondentes de $Mvelha$ para resolver o sistema linear gerado. Para tanto, é criado uma matriz nova $Mnova$ que armazena a solução do sistema e são inicializados dois parâmetros adicionais, $erro = 1.0$ e $precisão = 10^{-5}$. Dentro da iteração são atualizados os valores do contorno. Adicionalmente, são criados dois loops for na qual são calculados:

a) Os valores da nova matriz $Mnova$.
b) Uma matriz ou vetor $desv$ que armazena o módulo da diferença entre $Mvelha$ e $Mnova$. Ademais, o valor de $Mvelha$ é atualizado de forma que $Mvelha = Mnova$.

Depois de finalizar os dois loops for, o paramêtro $erro$ é atualizado para o máximo valor do vetor $desv$. A iteração inicial continua até que a expressão $precisão < erro$ deixe de ser verdade. Ao final, a matriz $Mnova$ contém a solução final que é impressa para a visualização.

5. Benchmark

As 6 condições de contorno (Figura 1), bem como o loop necessário para resolver o sistema da equação (5), foram implementadas na linguagem C, utilizando o compilador gcc. Considerou-se uma matriz de 1536 x 1536 e se utilizou a temperatura de 75ºC como valor inicial da matriz, antes da iteração. Considerando uma malha de 60x60, realizou-se um esboço numérico da matriz final para visualizar os dados gerados (Figura 4), assim como um mapa de calor (Figura 5)

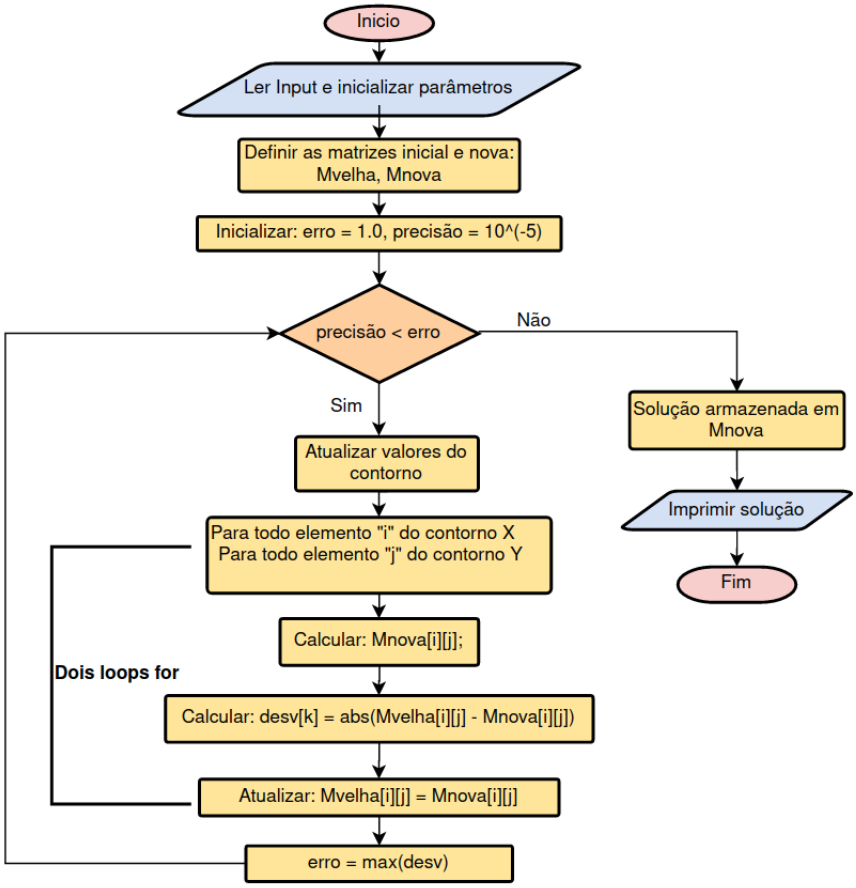


Figura 3. Fluxograma do Programa

A primeira implementação foi realizada em um computador de escritório. O hardware utilizado (Tabela 1) é mostrado como referência do desempenho. O tempo de execução obtido foi de 201.95 minutos.

Tabela 1. Informações do hardware - Desktop pessoal

Processador	Intel(R) Core(TM) i3 10100
Número de núcleos	4
Número de threads (Hyper-threading)	8
Clock Base	3.60 GHz
Clock Turbo Máximo	4.30 GHz
Cache L1	128 KB
Cache L2	1MB
Cache L3	6MB
BoboMips	7200.00
Memoria RAM	16GB DDR4-2666

5.1. Benchmark com flags

Considerando o hardware mencionado anteriormente (Tabela 1) e utilisando as flags: **-ffinite-math-only -mtune=native -funroll-loops -foptimize-register-move -m64**, conseguiu-se melhorar o desempenho do tempo de execução, obtendo 82.53 minutos.

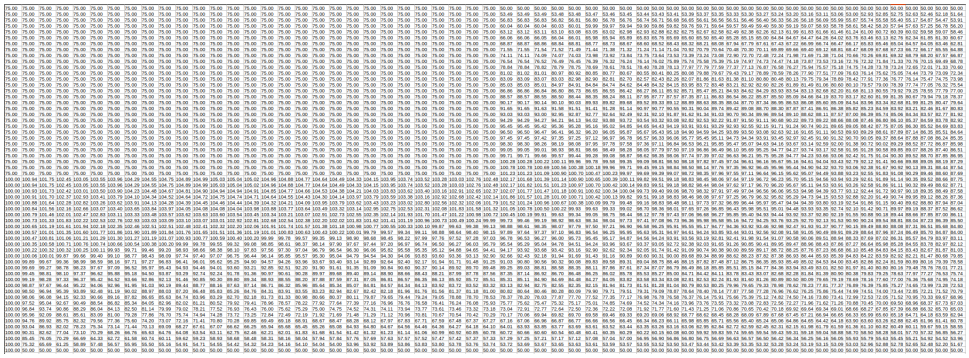


Figura 4. Matriz 60x60

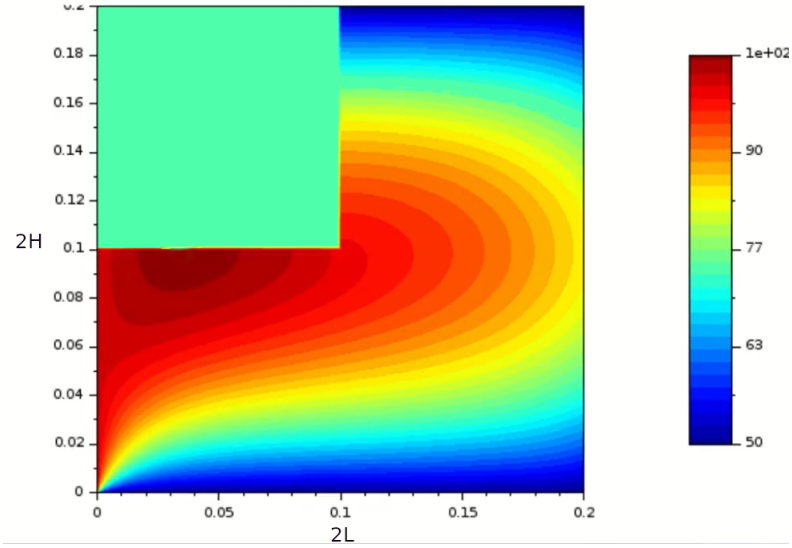


Figura 5. Mapa de calor da Matriz 60x60

6. Profiling e Otimização de software

Nesta seção se mostram os resultados obtidos da compilação e execução do programa com a função profiling, antes e depois de modificar e otimizar o código (Figuras 6 e 7). Considerou-se uma malha de 600x600 e uma precisão para o método de Jacobi de 0.00001. Os tempos de execução encontrados em segundos (sem flags), usando os mesmos dados de entrada antes e depois da otimização, foram: $t_{antes} = 908.67s$ e $t_{depois} = 612.80s$. Assim, se obteve um ganho de desempenho de $295,87s$. Isto significa um 32.56% do tempo de execução do código inicial.

Para alcançar o ganho de performance relatado, foi necessário realizar as seguintes alterações no código do programa:

a) Definir variáveis constantes auxiliares fora dos loops e redefinir as matrizes do problema com o tipo $static\ double < matriz >$. É importante destacar que se experimentou a técnica de alocação dinâmica de memória para inicializar as matrizes, porém, o desempenho não foi satisfatório. Neste caso específico, dado que as dimensões do problema são bem definidas e não são muito grandes, o tipo $static$ para definir matriz exibiu melhor performance em relação ao tempo de execução.

b) Identificar os contornos que podem ser calculados fora da iteração *while* principal. Os contornos 1, 4 e 6 são constantes (Figuras 1a e 1b) e podem ser calculados fora do *while* numa função que será chamada de *contorno_constante()*. Os contornos 2, 3 e 6 são executadas dentro do *while* na função *main*, posto que seus cálculos implicam, necessariamente, o uso de derivadas parciais.

c) Incluir a função *fmax()*, própria da linguagem C, para calcular o máximo de um par de números. Desta forma, a matriz $desv$ é a função *valormar()* são substituídas pela função *fmax()*

```
luis@pc:~/c_c++projects/HPC-UFF/ProjetoFinal-0$ gprof ./LuisArmando_Programa
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 80.38    728.42   728.42                             main
 19.89    908.67   180.25   322902   558.22   558.22  valormax

 %         the percentage of the total running time of the
time       program used by this function.
```

Figura 6. Execução do Profile antes da otimização

```
luis@pc:~/c_c++projects/HPC-UFF/ProjetoFinal-0$ gprof ./LuisArmando_Programa_Profile
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
 99.51    612.80   612.80                             main
  0.00    612.80     0.00        1     0.00     0.00  contorno_constante

 %         the percentage of the total running time of the
time       program used by this function.
```

Figura 7. Execução do Profile depois da otimização

7. Comparação dos programas base e otimizado

Executou-se o programa depois de otimizar o código e utilizando as melhores flags. A malha utilizada foi a matriz 1536 x 1536 usada anteriormente na seção 5. Consideraram-se duas configurações de hardware: o Desktop pessoal utilizado previamente nas seções anteriores (Tabela 1) e o desktop do Laboratório 107C (Lab107C) da UFF (Tabela 2). O compilador e flags empregados foram: **icc -ipo -xhost -ffinite-math-only**.

É importante ressaltar que no compilador da intel, o nível de compilação -O2 vem por *default*[1] e não precisa ser colocado explicitamente. Em geral, observou-se que o compilador da intel, quando utilizado sem flags, proporciona melhor desempenho que sua contraparte GNU (Tabela 3 e Figura 8). Utilizando as melhores flags e a otimização de software, o desktop pessoal mostrou melhor desempenho. Quantitativamente, o ganho de performance nos dois computadores, em relação ao programa base, foram de 73.73% no desktop pessoal e 42.55% no computador do Lab107C.

8. Implementação com OpenMP

A versão paralela OpenMP foi implementada utilizando o construtor de trabalho *#pragma omp for* antes de cada loop *for* que se encontra dentro da iteração *while* principal. Além

[1]https://software.intel.com/content/dam/develop/public/us/en/documents/quick-reference-guide-intel-compilers-v19-1-final-.pdf

Tabela 2. Informações do hardware - Laboratório 107C

Processador	Intel(R) Core(TM) i7-2600
Número de núcleos	4
Número de threads (Hyper-threading)	8
Clock Base	3.40 GHz
Clock Turbo Máximo	3.80 GHz
Cache L1	32 KB
Cache L2	256 KB
Cache L3	8MB
BoboMips	6784.51
Memoria RAM	4GB DDR3

Tabela 3. Tempo de execução (minutos) do programa nos dois computadores

Computador	Desktop pessoal (compilador gcc)	Laboratório 107C (compilador icc)
Programa base	201.95	133.57
Programa com flags	82.53	107.88
Programa com Otimização	53.52	71.99

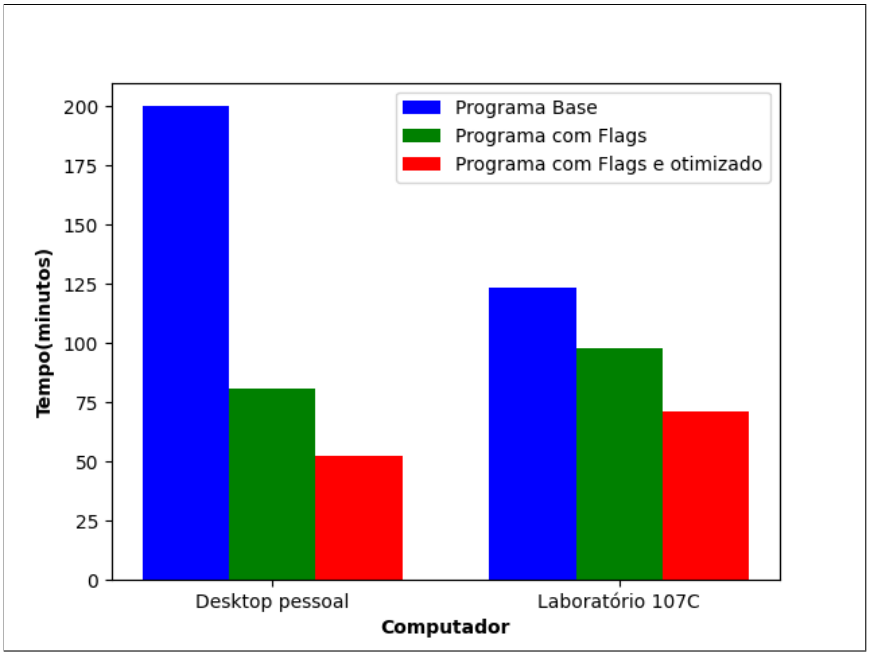


Figura 8. Comparação de tempos de execução

disso, na região paralela onde são atualizados todos os valores dos pontos da malha, é necessário utilizar um método seguro e consistente para o computo dos valores da matriz nova. Para tal fim, os índices (i, j) dos loops foram declarados como *private* e o erro é obtido mediante a cláusula $reduction$ com o operdor max.

Ademais, para o usufruir o máximo desempenho das threads no computador do Lab107C, foi necessário personalizar o constructor *#pragma omp for* estabelecendo a claúsula *schedule (static, 64)*. No caso do desktop pessoal, no foi necessário implementar outra clausula além das mencionadas até aqui.

9. Implementação com MPI

A versão paralela MPI foi implementada considerando uma decomposição de domínio horizontal. O dominio de solução foi dividido em duas partes, *A* e *B*. A comunicação entre os processos é realizada mediante as rotinas *MPI_Isend*, *MPI_Recv* e *MPI_Wait*. Para atualizar o máximo erro obtido dentro do *while*, usou-se a rotina *MPI_Allreduce*.

10. Benchmark do Programa paralelizado

10.1. Avaliação de desempenho

Realizou-se o benchmark para avaliar o desempenho do paralelismo implementado no programa. Para tanto, considerou-se a malha representada pela matriz 1536 x 1536 que foi utilizada nas seções anteriores com os dois computadores utilizados, o desktop pessoal (Tabela 1) e o do Lab107C (Tabela 2). Além do hardware mencionado, empregou-se também quatro nós computacionais do supercomputador Santos Dumont (SDumont) do Laboratório Nacional de Computação Científica (LNCC). Cada nó do SDumont considerado utiliza uma implementação *dual-socket* conformada por dois processador Intel Xeon. Sendo assim, cada nó computacionais corresponde a um dual-socket de 24 o 48 núcleos (Tabelas 4 e 5). É importante destacar que a tecnologia *Hyper-Threading* não está ativada nesses processadores. Além disso, o tempo máximo de execução permitida em cada nó é de 20 minutos.

Tabela 4. Informações do CPU, nó dual-socket de 24 núcleos

Processador	Intel® Xeon® E5-2695 v2
Clock Base	2.40 GHz
Clock Turbo Máximo	3.20 GHz
Cores físicos	12
Cache L1	12 x 32 KB
Cache L2	12 x 256 KB
Cache L3	30 MB

Tabela 5. Informações do CPU, nó dual-socket de 48 núcleos

Processador	Intel® Xeon® Gold 6252
Clock Base	2.10 GHz
Clock Turbo Máximo	3.70 GHz
Cores físicos	24
Cache L1	24 x 32 KB
Cache L2	24 x 1 MB
Cache L3	35.75 MB

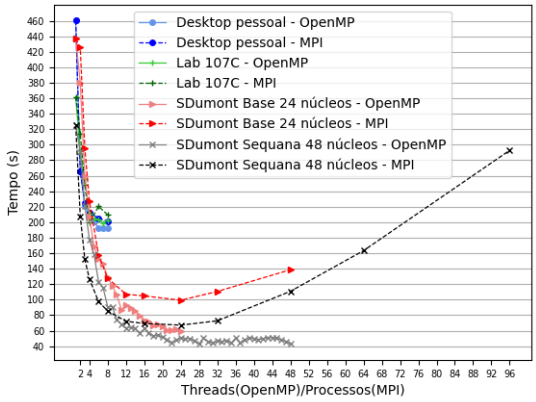
Desta forma, foram 4 computadores com arquiteturas diferentes que foram empregadas para o benchmark. É necessário destacar que o compilador e as flags utilizadas nas 4 máquinas foram: **icc -ipo -xhost -ffinite-math-only**.

As métricas consideradas para a avaliação foram: o tempo de execução, eficiência e speedup (que mede a "aceleração" do tempo de execução e a escalabilidade da aplicação). No caso da matriz 1536 x 1536, observou-se que o tempo de execução do programa sequencial em cada nó do SDumont supera o limite permitido de 20 minutos,

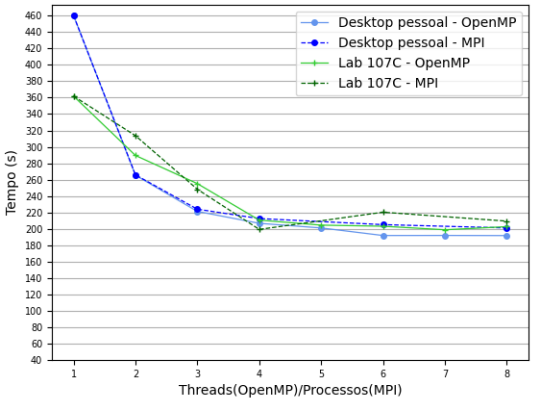
razão pela qual a eficiência e o speedup não foram avaliadas para esses dois nós computacionais. Em consequência, escolheram-se uma matriz menor para que o programa possa ser executado sequencialmente, utilizando apenas um núcleo, em menos de 20 minutos no SDumont.

Sendo assim, as métricas para avaliação de desempenho foram implementadas para as duas matrizes diferentes: 768 x 768 (Figura 9) e 1536 x 1536 (Figura 10). Na sequência, são discutidas os resultados obtidos para cada matriz considerada métrica utilizada.

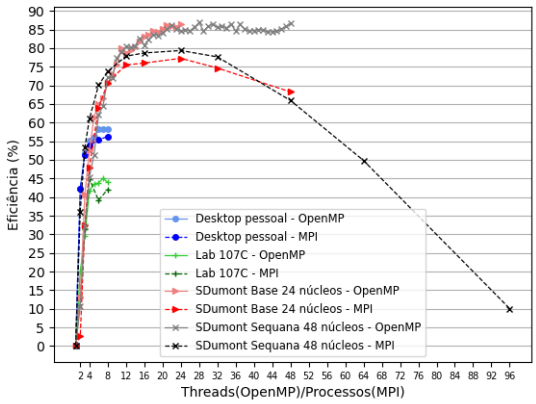
- Matriz 768 x 768
  Observou-se que o tempo de execução manifesta um comportamento particular dependendo do computador e versão paralela utilizados. No caso da versão OpemMP, à medida que aumenta o número de threads na execução do programa, os Intel Xeon do SDumont mostram uma performance bem significativa (Figura 9a). No caso do computador do Lab107C e do desktop pessoal, eles exibem uma perfomance muito similar, notando que o melhor desempenho mono-núcleo é obtido pelo maquina do Lab107C e o melhor desempenho multinúcleo pelo desktop pessoal (Figura 9b).



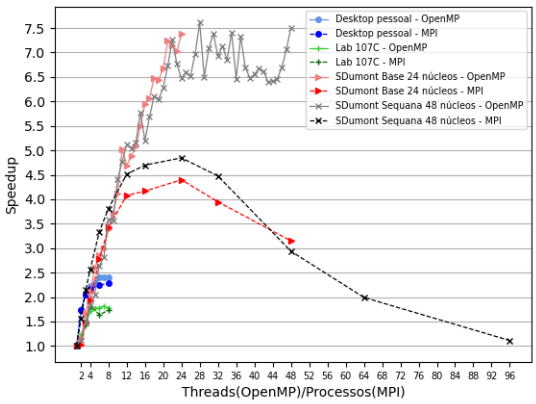
(a) Tempo de execução



(b) Tempo de execução



(c) Eficiência



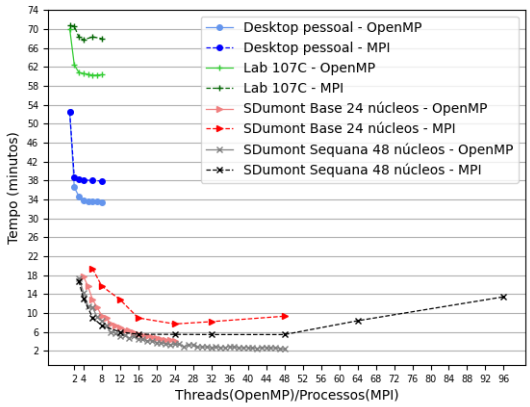
(d) Speedup

Figura 9. Métricas avaliadas para Matriz 768 x 768

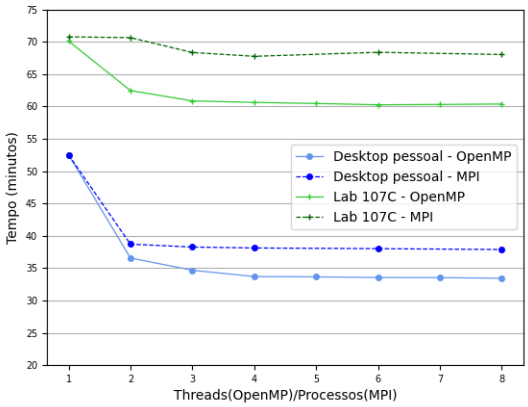
Como consequência da análise comentada no parágrafo anterior, a eficiência atingida pelos computadores do SDumont atinge um máximo de 86% que é bem superior quando comparado ao outros computadores, que alcançam valores de 58% e 45% para o desktop pessoal e Lab107C respectivamente (Figura 9c). Essa diferença na eficiência implica também um maior speedup (Figura 9d). Observou-se que os computadores do SDumont exibem um speedup máximo de 7.5x aproximadamente. No caso do desktop pessoal e da máquina do Lab107C, o máximo speedup foi de 2.4x e 1.8x respectivamente.

No caso da versão MPI, observou-se um comportamento similar à versão OpenMP detalhada nos parágrafos anteriores. Porém, quando utilizados todas os processos ou cores possíveis em um nó computacional (24 no SDumont Base e 48 no SDumont sequana), **a versão OpenMP exibe uma melhor performance em todas as métricas analisadas** (Figuras 9a - 9d).

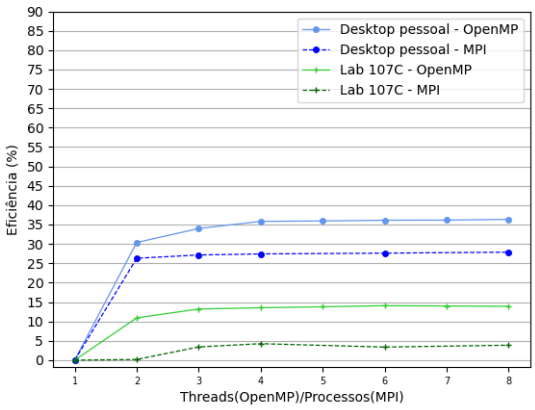
Complementarmente, constatou-se que a versão MPI não aproveitou de forma efetiva o uso de dois ou mais nós computacionais. No caso do SDumont sequana, os 96 processos utilizados que representam 2 nós computacionais, mostram um tempo de execução maior que 48 processos (que representam apenas 1



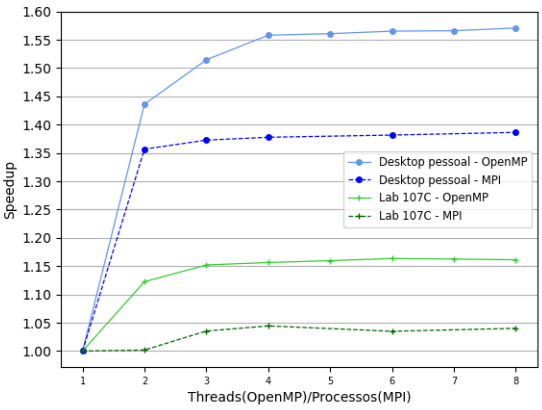
(a) Tempo de execução



(b) Tempo de execução



(c) Eficiência



(d) Speedup

Figura 10. Métricas avaliadas para Matriz 1536 x 1536

nó computacionais (Figura 9a). Em consequência, a eficiência e o speedup diminuem (Figuras 9c e 9d). O mesmo fenômeno acontece quando analisado os 48 processos utilizados no SDumont base.

- Matriz 1536 x 1536
  Neste caso, a diferença nos tempos de execução entre os nós do SDumont e os outros computadores é substancialmente ampla (Figura 10a). Convém destacar novamente que o tempo máximo para execução de um job no SDumont é de 20 minutos, razão pela qual foi impossível executar o programa usando menos de 3 e 4 threads para o nó Sequana e Base respectivamente. Apesar disso, pode-se observar uma clara tendencia na diminuição do tempo de execução à medida que aumenta o número de threads e/ou processos. Nos outros computadores, o tempo também diminui, porém de forma pouco uniforme (Figura 10b)
  Visto que é necessário dispor do tempo de execução mono-núcleo para encontrar a eficiência e o speedup, não foi viável utilizar essas métrica para os computadores do SDumont. De forma similar à matriz analisada previamente, **a versão OpenMP exibe uma melhor performance em todas as métricas** nos outros computadores (Figuras 10c e 10d).

Em resumo, para um melhor entendimento do máximo desempenho alcançado usando todas as threads (OpenMP) e/ou processos (MPI) nos 4 computadores, os dados obtidos para cada matriz foram agrupados . Sendo assim, tém-se: o melhor (mínimo) tempo de execução atingido (Tabela 6), a máxima eficiência (Tabela 7) e o máximo speedup (Tabela 8).

É importante destacar os resultados obtidos da matriz 1536 x 1536, que representa a malha inicial escolhida para a avaliação neste trabalho. Considerando os dados obtidos na seção 7 (Tabela 3) e o melhor(menor) tempo de execução da citada matriz (Tabela 6), deduz-se que, o trabalho que no programa base era executado pelo desktop pessoal em 201.95 minutos, na versão paralela otimizada e com flags é realizada em apenas 2.37 minutos (versão OpenMP) e 5.42 minutos (versão MPI) no melhor nó computacional (Sequana) do SDumont!.

Tabela 6. Melhor tempo (minutos), usando todas as threads / processos

Computador	Matriz 768 x 768		Matriz 1440 x 1440	
	OpenMP	MPI	OpenMP	MPI
Desktop pessoal	3.20	3.35	33.44	37.88
Lab107C	3.32	3.32	60.26	67.75
Sdumont Base	0.99	1.65	4.16	7.67
Sdumont Sequana	0.71	1.12	2.37	5.42

11. Validação dos Resultados

Para validar os resultados do programa paralelo otimizado e com flags, foi utilizado o comando **cmp - - silent** no terminal do linux para realizar a compararção com o resultado obtido da versão base do programa. Seja **matriz-base.csv** o nome do arquivo gerado pelo programa base e **matriz-paralela.csv** o arquivo gerado pelo programa paralelo e

Tabela 7. Máxima eficiência (%), usando todas as threads / processos

Computador	Matriz 768 x 768		Matriz 1440 x 1440	
	OpenMP	MPI	OpenMP	MPI
Desktop pessoal	58.31	56.27	36.33	27.87
Lab107C	45.02	44.89	14.05	4.25
Sdumont Base	86.45	77.27	-	-
Sdumont Sequana	86.86	79.37	-	-

Tabela 8. Máximo speedup atingido, usando todas as threads / processos

Computador	Matriz 768 x 768		Matriz 1440 x 1440	
	OpenMP	MPI	OpenMP	MPI
Desktop pessoal	2.39x	2.29x	1.57x	1.39x
Lab107C	1.82x	1.81x	1.16x	1.04x
Sdumont Base	7.38x	4.40x	-	-
Sdumont Sequana	7.61x	4.85x	-	-

com flags. Depois de dar enter ao comando **cmp - - silent matriz-base.csv matriz-paralela.csv | | echo "Os arquivos são diferentes"** no terminal, não é impresso nenhum aviso, o que demonstra que os dois dois arquivos são iguais.

12. Conclusões

Na sequência, destacam-se as principais conclusões deste trabalho :

- É possível obter um ganho de desempenho do tempo de execução paralelizando um programa que resolve um tipo de equação de calor.
- Os processadores do Sdumont oferecem uma maior escalabilidade para o programa paralelo proposto neste trabalho.
- À medida que aumenta o tamanho da matriz, a performance dos processadores Xeon do SDumont tornam-se mais eficientes e a versão paralela do algoritmo apresente melhor escalabilidade.
- Em um nó computacional, a versão paralela OpenMP exibe um melhor desempenho que a sua contraparte MPI.
- Para mais de um nó computacional, a versão paralela MPI apresenta um desempenho inferior à execução realizada em apenas um nó.

Referências

[Scherer 2017] Scherer, P. O. (2017). *Computational physics: simulation of classical and quantum systems*. Springer.

[Vasconcelos et al. 2015] Vasconcelos, M. A., COELHO, N., and PEDROSO, L. (2015). Estudo de diferenças finitas para a equação do calor em barragens de concreto. In *XXXVI Iberian Latin American Congress on Computational Methods in Engineering. Rio de Janeiro*.