Computação Científica de Alto Desempenho I
Projeto Final

Luis Armando Quintanilla Villon[1]

[1]Instituto de Física - UFF

luisvillon@id.uff.br

Resumo. *Neste trabalho desenvolveu-se uma versão paralela OpenMP de um programa que resolve um tipo de equação de calor. Para tal fim, a versão 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 é possível atingir, mo melhor dos casos experimentados, uma eficiência próximo a 90% e um speed up de 9.61x. Encontrou-se também que o uso de flags agressivos poderia degradar o desempenho multi-core.*

Abstract. *In this work it was developed a parallel OpenMP version of a program that solves a type of heat equation. To this end, the sequential version was previously improved by means of specific flags and software optimization techniques. The parallel program was run on 4 different computers. The results show that it is possible to achieve, at best, an efficiency close to 90 % and a speed up of 9.61x. It was also found that the use of aggressive flags could degrade multi-core performance.*

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. Na seção 8 é detalhado a implementação da versão OpenMP paralela e a sua análise de desempenho é descrita na seção 9. Na seção 10 é mostrada a maneira que a validação do programa paralelo pode validado. Finalmente, as conclusões finais são relatados na seção 11.

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.
- 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 mediante programas específicos, como o vtune da intel

3. Metodologia

3.1. Descrição

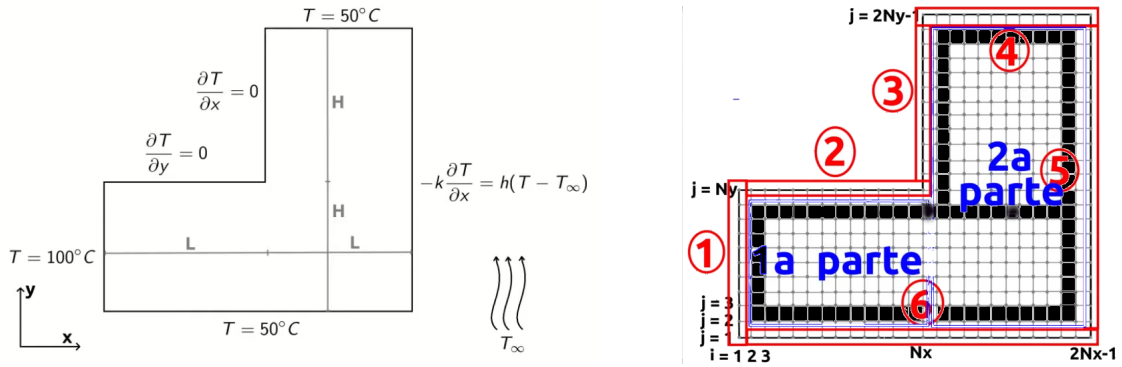
Para este trabalho se considerou em utilizar uma PDE (differential partial equation) elíptica que descreve a transferência de calor em um objeto sólido com geração de calor [Vasconcelos et al. 2015]:

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

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.



(a) Condições de contorno - modelo matemático

(b) Contornos enumerados de 1 a 6

Figura 1. Condições de contorno

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

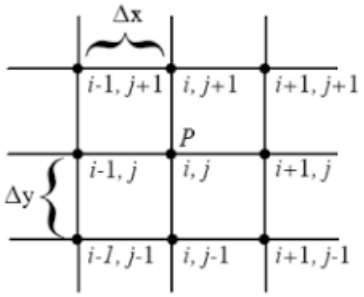


Figura 2. Ponto P - Malha bidimensional

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 :

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

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.

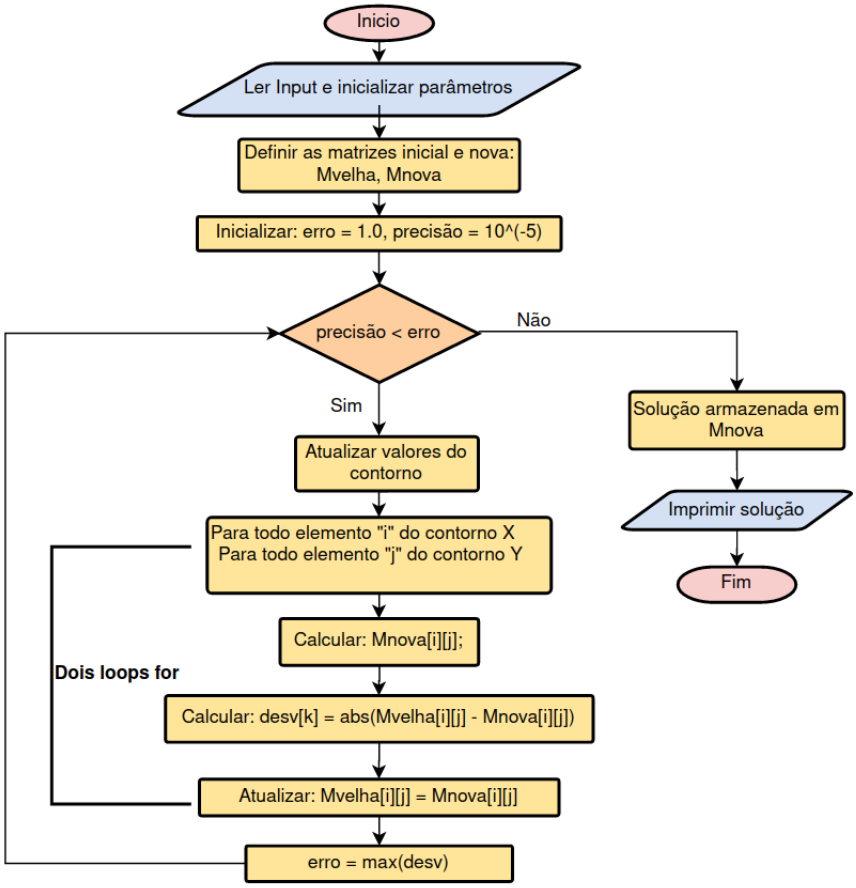


Figura 3. Fluxograma do Programa

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 1440 x 1440 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)

A primeira implementação foi realizada em um computador de escritório. O hardware utilizado (Tabla 1) é mostrado como referência do desempenho. O tempo de execução do programa foi calculado de duas formas (Figura 6) para reforçar um resultado mais robusto do intervalo de tempo solicitado (3-5 horas). Assim, o tempo de execução obtido foi de 199.95 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 80.53 minutos (Figura 7).

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 8 e

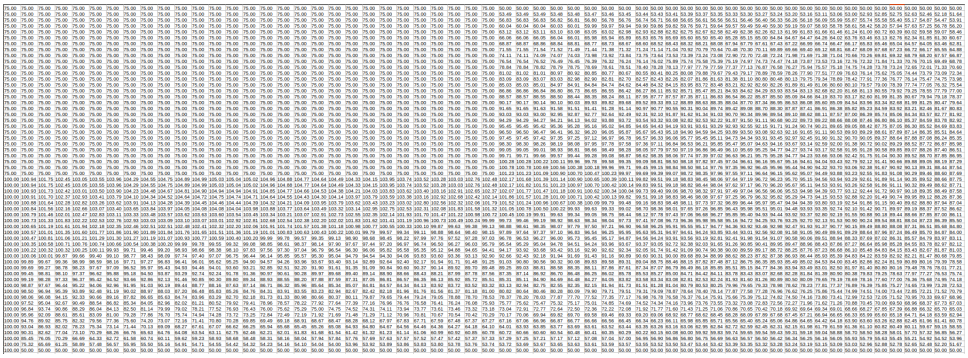


Figura 4. Matriz 60x60

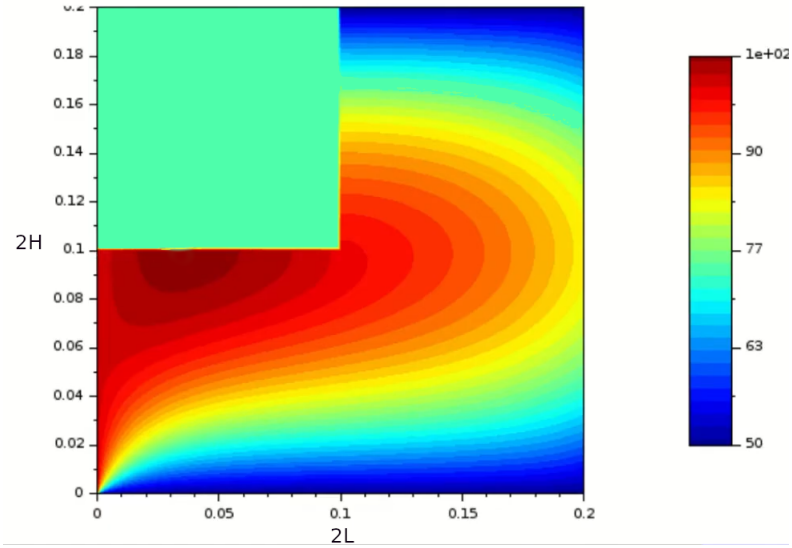


Figura 5. Mapa de calor da Matriz 60x60

```
luis@pc:~/c_c++projects/HPC-UFF/ProjetoFinal-0$ sh LuisArmando_Programa_Script.sh
Compilação Finalizada
luis@pc:~/c_c++projects/HPC-UFF/ProjetoFinal-0$ time ./LuisArmando_Programa
Tempo em segundos: 11996.922675
Tempo em horas: 3.332479

real    199m57,637s
user    199m56,842s
sys     0m0,084s
luis@pc:~/c_c++projects/HPC-UFF/ProjetoFinal-0$
```

Figura 6. Tempo de execução do programa base

```
luis@pc:~/c_c++projects/HPC-UFF/ProjetoFinal-0$ cat LuisArmando_Programa_Script.sh
#!/bin/bash

gcc  -O2 -ffinite-math-only -mtune=native -funroll-loops -foptimize-register-move -m64 LuisArmando_Programa.c -o LuisArmando_Programa

echo "Compilação Finalizada"



luis@pc:~/c_c++projects/HPC-UFF/ProjetoFinal-0$ sh LuisArmando_Programa_Script.sh
Compilação Finalizada
luis@pc:~/c_c++projects/HPC-UFF/ProjetoFinal-0$ time ./LuisArmando_Programa
Tempo em segundos: 4831.540264
Tempo em horas: 1.342095

real    80m31,876s
user    80m31,510s
sys     0m0,032s
luis@pc:~/c_c++projects/HPC-UFF/ProjetoFinal-0$
```

Figura 7. Tempo de execução do programa compilado com as melhores flags

9). 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 8. 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 9. 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 1440 x 1440 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). Os compiladores e flags empregados foram os seguintes:

- Compilador do GNU no Desktop pessoal: **gcc -O2 -ffinite-math-only -mtune=native -funroll-loops -foptimize-register-move -m64**
- Compilador da Intel no Laboratório 107C: **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 utilisado sem flags, proporciona melhor desempenho que sua contraparte GNU (Tabela 3 e Figura 10). Utilizando as melhores flags e a otimização de software, o desktop pessoal mostrou melhor desempenho. Quantitativamente, a eficiência do ganho de performance nos dois compudores, em relação ao programa base, foram de 73.73% no desktop pessoal e 42.55% no computador do Lab107C.

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	199.94	123.57
Programa com flags	80.53	97.88
Programa com Otimização	52.52	70.99

8. Implementação com OpenMP

A versão paralela do programa foi implementada com OpenMP utilizando o construtor de trabalho *#pragma omp for* antes de cada loop *for* que se encontra dentro da iteração *while* principal (Figura 11). Além 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

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

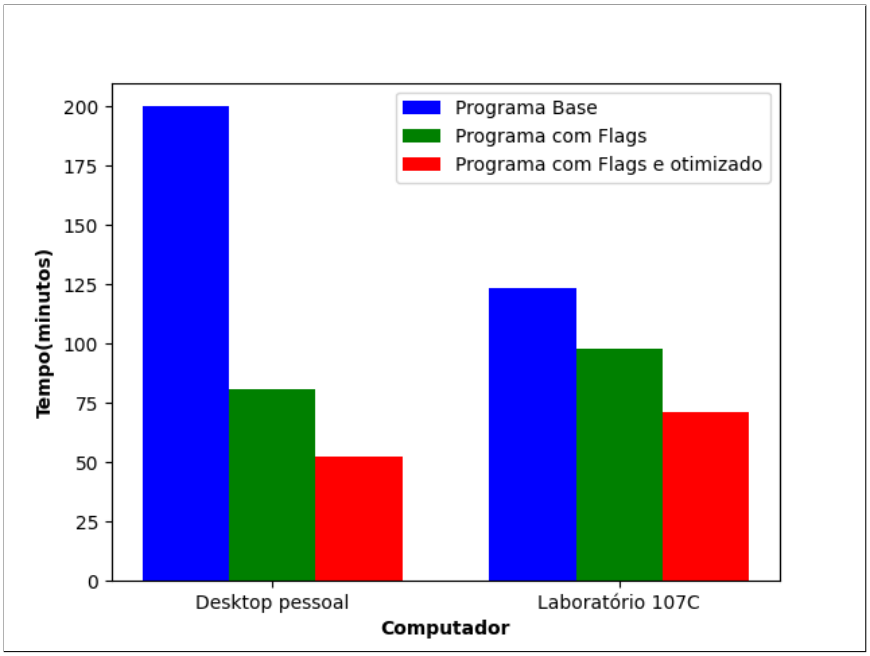


Figura 10. Comparação de tempos de execução

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. Benchmark do Programa paralelizado

9.1. 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 1440 x 1440 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 dois 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, tem-se dois nós computacionais de 24 e 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.

Desta forma, foram 4 computadores com arquiteturas diferentes que foram empregadas para o benchmark. É necessário destacar os compiladores e flags utilisados nas 4 máquinas:

- Compilador do GNU no Desktop pessoal: **gcc -O2 -ffinite-math-only -mtune=native -funroll-loops -foptimize-register-move -m64**
- Compilador da Intel no Laboratório 107C: **icc -ipo -xhost -ffinite-math-only**

```
    //-------------------------------------Loop principal para resolver o sistema--------------------.
    //Inicio da contagem de tempo.
    double start = omp_get_wtime();
    while ( precisao < erro )
    {
        #pragma omp parallel
        {
             //-------------------------------------Contorno 2.
            #pragma omp for nowait
            for(i = 1; i<=l2; i++)
            {
                Mvelha[i][l1] = Mvelha[i][l6];
                Mnova[i][l1] = Mnova[i][l6];
            }

            //-------------------------------------Contorno 3.
            #pragma omp for nowait
            for(j = Ny; j <=l3; j++)
            {
                Mvelha[l2][j] = Mvelha[Nx][j];
                Mnova[l2][j] = Mnova[Nx][j];
            }

            //-------------------------------------Contorno 5.
            #pragma omp for
            for(j = 1; j<=l3; j++)
            {
                Mvelha[l4][j] = c8*Mvelha[l7][j]+c7;
                Mnova[l4][j] = c8*Mvelha[l7][j]+c7;
            }

            //---------------------------------Cálculo da primeira parte da matriz nova.
            erro = 0.0;
            #pragma omp for private(i,j) //schedule(dynamic, chunk)
            for(i = 1; i<=l8; i++)
            {
                for(j = 1; j<=l6; j++)
                {
                    Mnova[i][j] = c4+c3*Mvelha[i-1][j]+c3*Mvelha[i+1][j]+c5*Mvelha[i][j+1]+c6*Mvelha[i][j-1];
                }
            }

            #pragma omp for reduction(max:erro) private(i,j) //schedule(dynamic, chunk)
            for(i = 1; i<=l8; i++)
            {
                for(j = 1; j<=l6; j++)
                {
                    // Calculando a diferença entre a matriz nova e velha.
                    erro = fmax(fabs(Mnova[i][j]-Mvelha[i][j]),erro);
                    Mvelha[i][j] = Mnova[i][j];
                }
            }

            //---------------------------------Cálculo da segunda parte da matriz nova.
            #pragma omp for private(i,j) //schedule(dynamic, chunk)
            for(i = Nx; i<=l8; i++)
            {
                for(j = l1; j<=l9; j++)
                {
                    Mnova[i][j] = c4+c3*Mvelha[i-1][j]+c3*Mvelha[i+1][j]+c5*Mvelha[i][j+1]+c6*Mvelha[i][j-1];
                }
            }

            #pragma omp for reduction(max:erro) private(i,j) //schedule(dynamic, chunk)
            for(i = Nx; i<=l8; i++)
            {
                for(j = l1; j<=l9; j++)
                {
                    // Calculando a diferença entre a matriz nova e velha.
                    erro = fmax(fabs(Mnova[i][j]-Mvelha[i][j]),erro);
                    Mvelha[i][j] = Mnova[i][j];
                }
            }
        }
    }
    //Contagem final do tempo.
    double end = omp_get_wtime();
```

Figura 11. Trecho do código da versão paralela com OpenPM

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

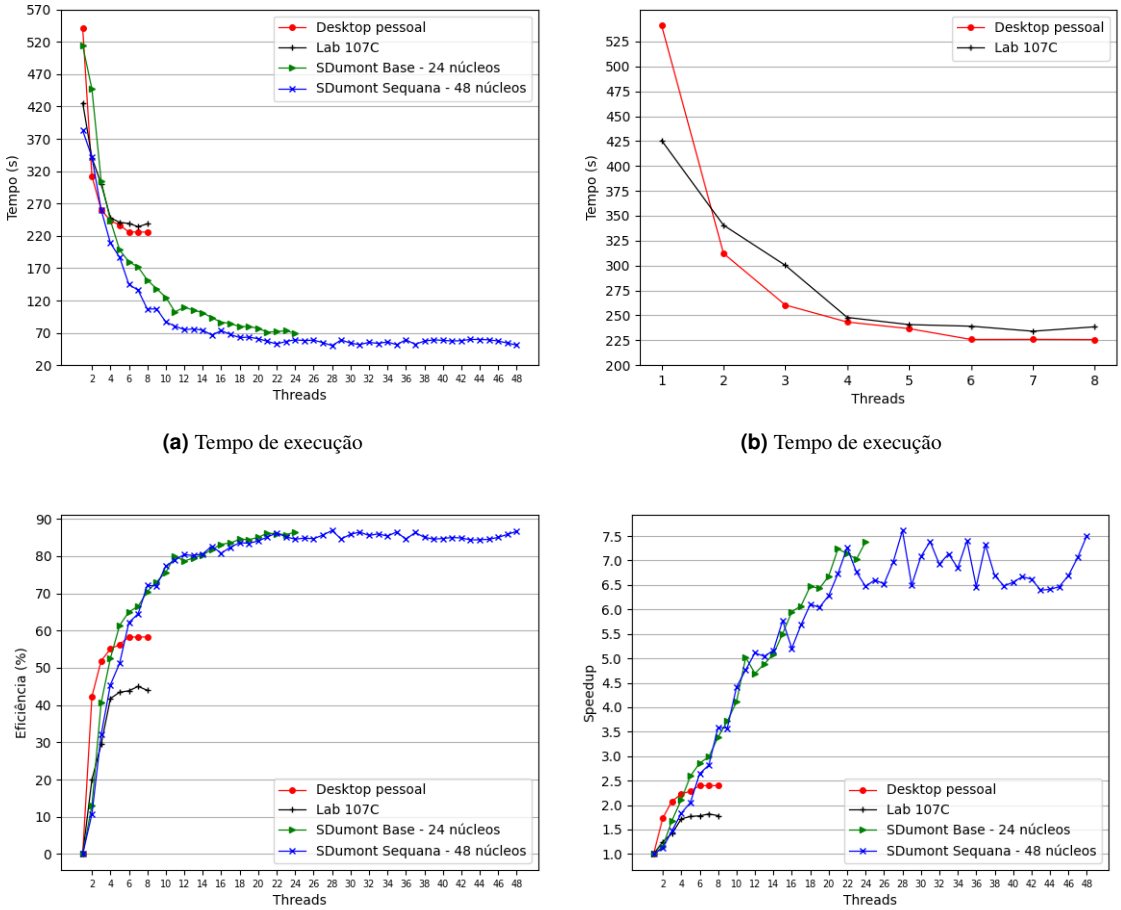
Processador	Intel® Xeon® E5-2695 v2
Clock Base	2.40 GHz
Clock Turbo Máximo	3.20 GHz
Cores físicos	12
Cache L1	12 x 32 KB
Cache L2	12 x 256 KB
Cache L3	30 MB

As métricas consideradas para a avaliação foram: o tempo de execução, eficiência e speedup (que mede a "aceleração" do tempo de execução do programa). No caso da matriz 1440 x 1440, observou-se que o tempo de execução para uma thread 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 duas matrizes menores para que o programa possa ser executado, utilizando apenas uma thread, em menos de 20 minutos no SDumont.

Sendo assim, as métricas para avaliação de desempenho foram implementadas

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



(a) Tempo de execução

(b) Tempo de execução

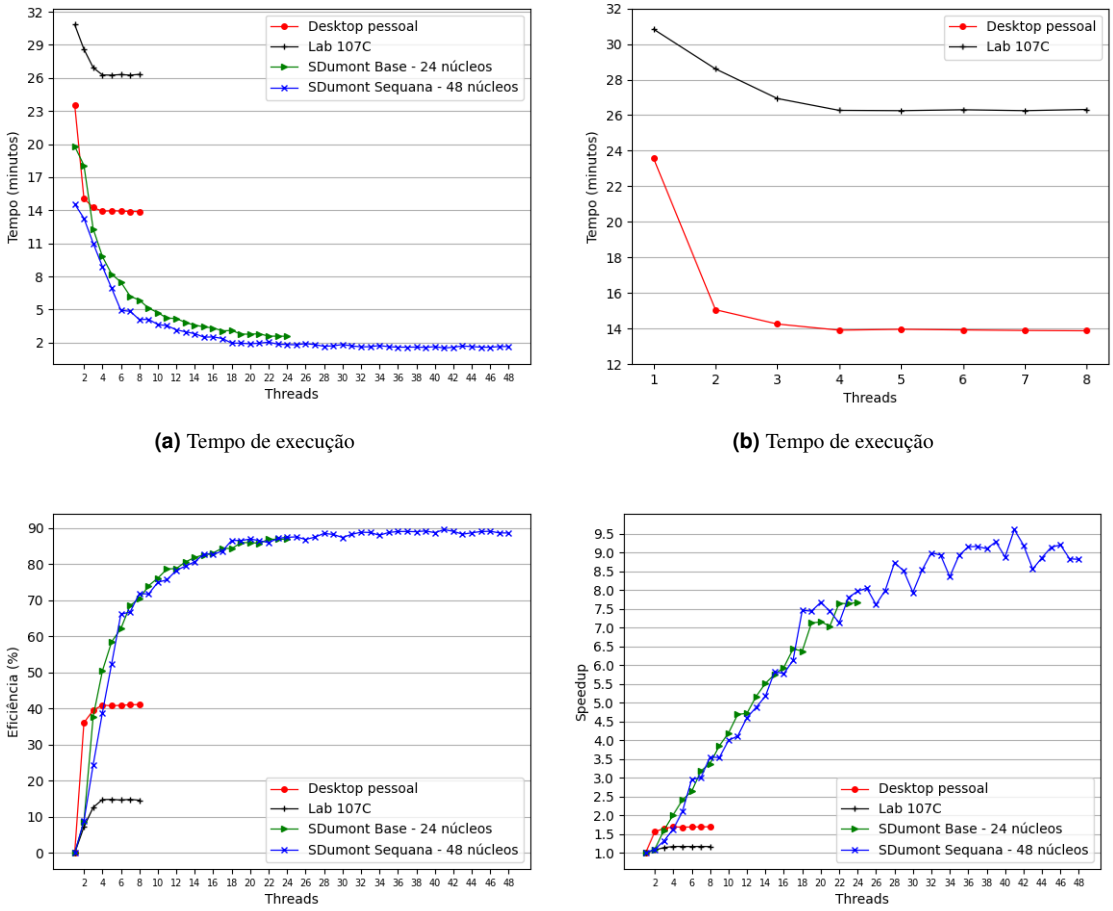
(c) Eficiência

(d) Speedup

Figura 12. Métricas avaliadas para Matriz 800 x 800

em três matrizes diferentes: 800 x 800 (Figura 12), 1080 x 1080 (Figura 13) e 1440 x 1440 (Figura 14). Na sequência, são discutidas os resultados obtidos para cada matriz considerada métrica utilizada.

- Matriz 800 x 800
  Observou-se que o tempo de execução manifesta um comportamento particular dependendo do computador utilizado. À medida que aumenta o número de threads na execução do programa, os Intel Xeon do SDumont mostram uma performance bem significativa (Figura 12a). No caso do computador do Lab107C e



(a) Tempo de execução

(b) Tempo de execução

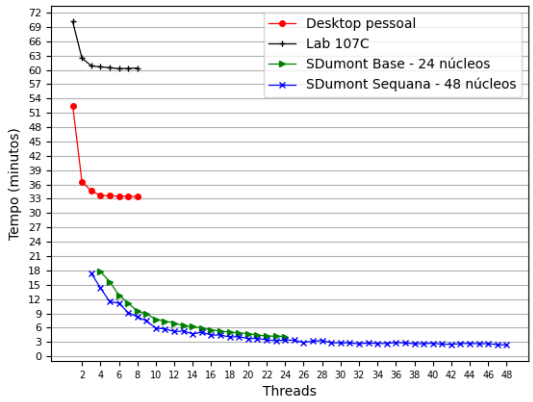
(c) Eficiência

(d) Speedup

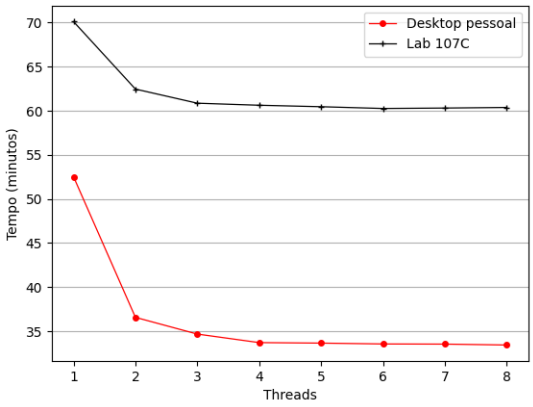
Figura 13. Métricas avaliadas para Matriz 1080 x 1080

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 12b).
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 12c).
De maneira concreta, essa diferença na eficiência implica também um maior speedup (Figura 12d). Observa-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.

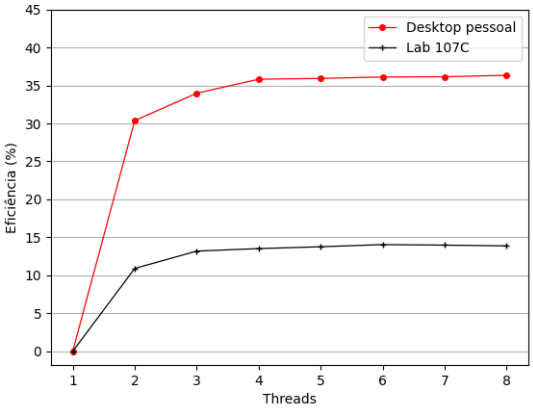
- Matriz 1080 x 1080
  De forma análoga ao observado para a matriz 800 x 800, os computadores da SDumont exibem um desempenho muito relevante à medida que mais threads são utilizados (Figura 13a). Além disso, observa-se que o rendimento do desktop pessoal é muito superior ao computador do Lab107C que mostrou a pior performance (Figura 13b).



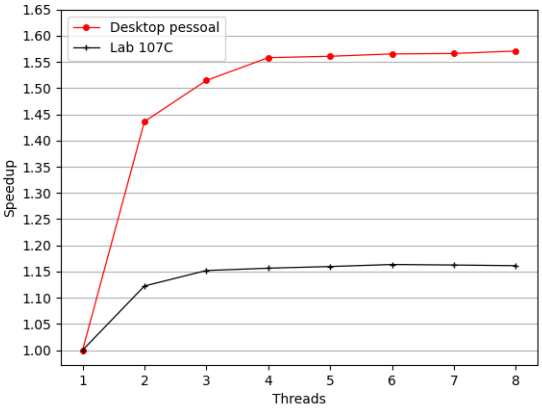
(a) Tempo de execução



(b) Tempo de execução



(c) Eficiência



(d) Speedup

Figura 14. Métricas avaliadas para Matriz 1440 x 1440

Todas as observações realizadas sobre o tempo de execução são igualmente constatadas na análise da eficiência. A eficiência dos nós computacionais do SDumont atinge quase 90% com o máximo uso das threads, enquanto que no caso do desktop pessoal e o computador do Lab107C, a eficiência alcança um pouco mais de 40% e 15% respectivamente (Figura 13c).

No gráfico do speedup, a tendencia continua sendo similiar à descrita anteriormente. Há uma grande diferença dos valor de speedup entre os nós computacionais do SDumont e os outros computadores (Figura 13d). O nó Base e o nó Sequana do SDumont alcançam speedup máximos de 7.6x e 9.6x respectivamente. No caso do desktop pessoal e do Lab107C, o speedup máximo atingido, nessa ordem, é apenas de 1.7x e 1.2x.

- Matriz 1440 x 1440

  Neste caso, a diferença nos tempos de execução entre os nós do SDumont e os outros computadores é substancialmente ampla (Figura 14a). 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. Nos outros computadores, o tempo também diminui, porém de forma pouco uniforme (Figura 14b)
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. Nos outros computadores, a eficiência e o speedup máximos alcançaram os valores, de 36% e 1.57x no do desktop pessoal, e 14% e 1.16x no Lab107C (Figuras 14c e 14d).

Em resumo, para um melhor entendimento do máximo desempenho alcançado usando todas as threads nos 4 computadores, foram agrupados os dados obtidos para cada matriz. Sendo assim, tem-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 1440 x 1440, 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 199.94 minutos, na versão paralela otimizada e com flags é realizada em apenas 2.37 minutos no melhor nó computacional do SDumont!.

Tabela 6. Melhor tempo (minutos) de execução atingido, usando todas as threads

Computador	Matriz 800 x 800	Matriz 1080 x 1080	Matriz 1440 x 1440
Desktop pessoal	3.76	13.88	33.44
Lab107C	3.90	26.25	60.26
Sdumont Base	1.16	2.58	4.16
Sdumont Sequana	0.84	1.52	2.37

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

Computador	Matriz 800 x 800	Matriz 1080 x 1080	Matriz 1440 x 1440
Desktop pessoal	58.31	41.13	36.33
Lab107C	45.02	14.85	14.05
Sdumont Base	86.45	86.97	-
Sdumont Sequana	86.86	89.59	-

Tabela 8. Máximo speedup atingido, usando todas as threads

Computador	Matriz 800 x 800	Matriz 1080 x 1080	Matriz 1440 x 1440
Desktop pessoal	2.39x	1.70x	1.57x
Lab107C	1.82x	1.17x	1.16x
Sdumont Base	7.38x	7.67x	-
Sdumont Sequana	7.61x	9.61x	-

9.2. Influencia de flags no desempenho multi-threading

Dos resultados subseção anterior, observa-se que no desktop pessoal e no computador do Lab107C, a eficiência o speedup tendem a diminuir à medida que aumenta o tamanho da matriz (Tabelas 7 e 8). Experimentalmente, encontrou-se que o uso de flags agressivos, em particular o **-ffinite-math-only**, aparentemente degrada o desempenho *multi-threading* (multi-núcleo) do processador e poderia explicar o comportamento exibido pela eficiência e o speedup.

Para verificar, de maneira mais rigorosa, se o uso da flag **-ffinite-math-only** degrada o desempneho multi-núcleo, realizou-se uma avaliação do programa paralelo utilizando o software vtune da intel[2] que se encontra instalado nos dois nós computacionais do SDumont. Para a coleta de dados, empregaram-se as opções **vtune -collect performance-snapshot** e **vtune -collect uarch-exploration**. Desta forma, são analisados a microarquitetura e memoria interna(cache L1, L2 e L3) do processador. Em síntese, para visualizar a performance do programa, neste trabalho apenas são considerados a "$Vetorização$", "$Memory\ bound$" e "$L1\ bound$". Como referência, é mostrado um trecho da avaliação realizada, na qual as métricas consideradas são definidas (Figura 15).

A análise fornecida pelo vtune foi aplicada na execução do programa, com e sem compilação da flag **-ffinite-math-only**. Para tanto, utilizou-se o máximo número de núcleos físicos disponíveis, tanto do nó *dual-socket* de 24 cores (Figura 16a), quanto do nó de 48 cores (Figura 16b). Nesta avaliação são mostrados: a fração (porcentagem) de operações vetoriais das totais utilizadas, a porcentagem em que a memória interna e a cache L3 realizam trabalho além do necessário. Claramente, o uso da flag intensifica, de forma notável a quantidade de operações vetoriais melhorando o desempenho do programa em geral. Porém há um maior uso ineficiente da memória e da cache L3, que podem ser causados pelos fenômenos conhecidos como *false-sharing* (falso compartilhamento) e *cache misses* [Hennessy and Patterson 2011]. Assim, dependendo da arquitetura e do conjunto de instruções, pode acontecer que o uso da flag **-ffinite-math-only**, aumente desnecessariamente, as latências na comunicação entre os núcleos do processador, gerando uma queda significativa no desempenho no programa paralelizado.

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

11. Conclusões

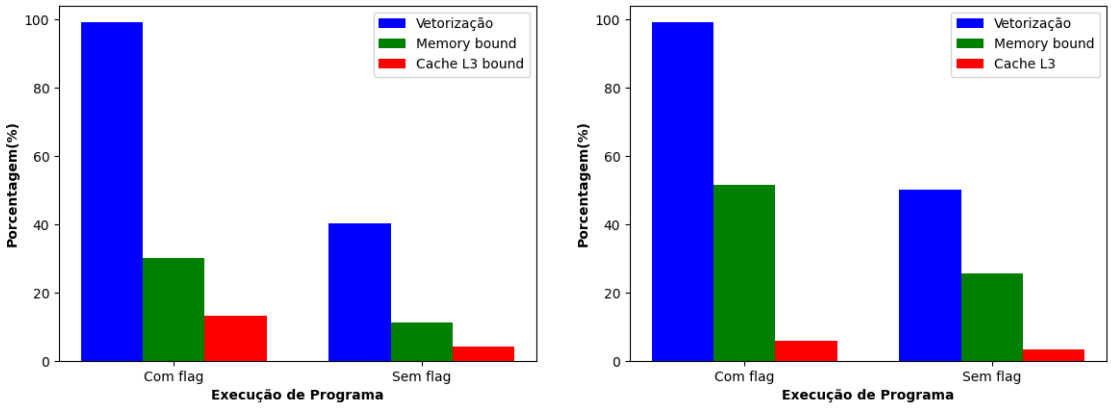
Obtiveram-se as seguinte conclusões depois de concluir este trabalho :

- É possível obter um ganho de desempenho do tempo de execução paralelizando um programa que resolve um tipo de equação de calor.

[2]https://software.intel.com/content/www/us/en/develop/documentation/vtune-help/top.html

```
Memory Bound: 30.1% of Pipeline Slots
 | The metric value is high. This can indicate that the significant fraction of
 | execution pipeline slots could be stalled due to demand memory load and
 | stores. Use Memory Access analysis to have the metric breakdown by memory
 | hierarchy, memory bandwidth information, correlation by memory objects.
 |
    L1 Bound: 7.1% of Clockticks
     | This metric shows how often machine was stalled without missing the L1
     | data cache. The L1 cache typically has the shortest latency. However, in
     | certain cases like loads blocked on older stores, a load might suffer a
     | high latency even though it is being satisfied by the L1.
     |
    L2 Bound: 0.1% of Clockticks
    L3 Bound: 13.3% of Clockticks
     | This metric shows how often CPU was stalled on L3 cache, or contended
     | with a sibling Core. Avoiding cache misses (L2 misses/L3 hits) improves
     | the latency and increases performance.
     |
    DRAM Bound: 5.7% of Clockticks
        Memory Bandwidth: 68.1% of Clockticks
    Store Bound: 12.6% of Clockticks
    NUMA: % of Remote Accesses: 65.9%
Vectorization: 99.1% of Packed FP Operations
    Instruction Mix
        SP FLOPs: 0.0% of uOps
            Packed: 0.0% from SP FP
                128-bit: 0.0% from SP FP
                256-bit: 0.0% from SP FP
            Scalar: 0.0% from SP FP
        DP FLOPs: 18.4% of uOps
            Packed: 99.1% from DP FP
                128-bit: 13.7% from DP FP
                 | A significant fraction of floating point arithmetic vector
                 | instructions is executed with a partial vector load. Make
                 | sure you compile the code with the latest instruction set or
                 | use Intel Advisor for vectorization help.
                 |
                256-bit: 85.4% from DP FP
            Scalar: 0.9% from DP FP
        x87 FLOPs: 0.0% of uOps
        Non-FP: 81.6% of uOps
    FP Arith/Mem Rd Instr. Ratio: 0.638
    FP Arith/Mem Wr Instr. Ratio: 2.035
```

Figura 15. Trecho do relatório de avaliação realizado pelo vtune



(a) Nó de 24 núcleos **(b)** Nó de 48 núcleos

Figura 16. Resumo da análise de performance realizada pelo vtune

- Os processadores do Sdumont oferecem uma maior escalabilidade para o programa paralelo proposto neste trabalho.
- O uso de flags específicos para melhorar o desempenho single-core de um programa pode não muito eficiente na versão paralela do mesmo programa.
- 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.

Referências

[Hennessy and Patterson 2011] Hennessy, J. L. and Patterson, D. A. (2011). *Computer architecture: A Quantitative Approach*. Elsevier.

[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*.