Universidade Federal de São Carlos

Bacharelado em Ciência da Computação

Sistemas Operacionais 2

"Trabalho sobre Paralelização usando Threads "

Juliano Lanssarini RA: 489093

Romão Martines RA: 595071

Professor: Hélio Crestana Guardia

São Carlos, 25 de junho de 2017.

1. Introdução

O presente trabalho trata-se de um estudo sobre a escalabilidade do programa de multiplicação de matrizes realizando a paralelização com o suporte de threads por meio da OpenMP – uma interface de aplicação para programação em multi-processos e compartilhamento de memória. Compreende rotinas do compilador, variáveis de ambiente e conjunto de bibliotecas que atuam no desempenho do programa a ser executado.

Com isso, o trabalho busca observar as variações no desempenho de um programa que efetua a multiplicação de duas matrizes. Um cálculo simples, mas que pode encontrar gargalos durante a execução, especialmente quando se trata de matrizes com dimensões gigantescas.

A seguir um breve relato dos comandos de execução e resultados obtidos, juntamente com algumas imagens do ambiente de execução.

1.1 Execução do Código

O programa pode ser compilado da seguinte maneira:

gcc omp_matrix.c -o omp_matrix -fopenmp

Observe que é necessário o argumento *-fopenmp* para que a biblioteca OpenMP seja executada junto do programa.

Por padrão, as variáveis principais do programa serão setadas com os seguintes valores:

NLIN = 1500

NCOL = 1500

NTHR = 4

Para executar o programa compilado, o usuário poderá fazê-lo de diversas maneiras, de acordo com sua necessidade e interesse. Isso porque ele pode definir o número de linhas e colunas das matrizes, além de definir se a paralelização acontecerá

por linhas, por colunas ou pela operação. Por padrão, as matrizes são geradas randomicamente e o comando de execução segue o seguinte modelo:

./omp_matrix -d

Contudo, o usuário poderá definir os parâmetros de execução, atribuindo valores para as linhas, colunas, número de threads, além da forma de paralelização, conforme segue:

a) Paralelização por linhas (usar o parâmetro -l):

./omp_matrix -l <num_Linhas> <num_Colunas> <num_Threads>

b) Paralelização por colunas (usar o parâmetro -c)

./omp_matrix -c <num_Linhas> <num_Colunas> <num_Threads>

c) Paralelização por operações

./omp_matrix -o <num_Linhas> <num_Colunas> <num_Threads>

Além disso, o usuário poderá solicitar ajuda para a execução do programa por meio da seguinte linha de comando:

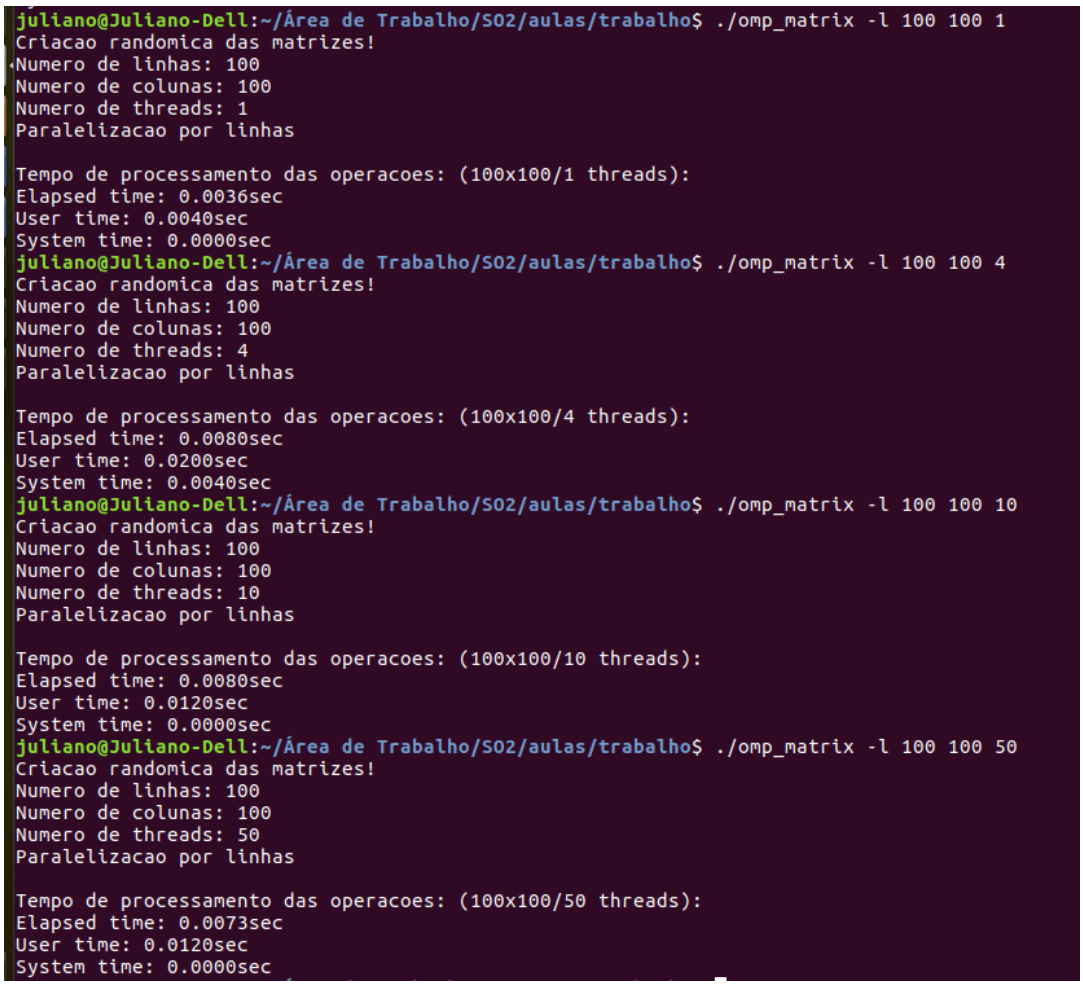
./paralelizacao_matrizes -h

2. Resultados

Foram realizados inúmeros testes para a análise do desempenho do programa. O desempenho foi analisado de acordo com as variáveis *elapsed time (tempo real do início ao fim do processo), user time (tempo em que o processador demora para realizar as operações determinadas pelo usuário)* e *system time (tempo em que o sistema assume o controle de execução para o programa, as chamadas de sistema - syscall).* Os tempos foram obtidos utilizando o comando time. O programa foi executado na mesma máquina, e todos nas mesmas condições de configuração e números de processos em execução na máquina.

Todos os testes foram feitos considerando matrizes de 100, 200 e 1500 linhas. Cada uma delas foram testadas para a paralelização com 1 thread, 4 threads, 10 threads e 50 threads, respectivamente.

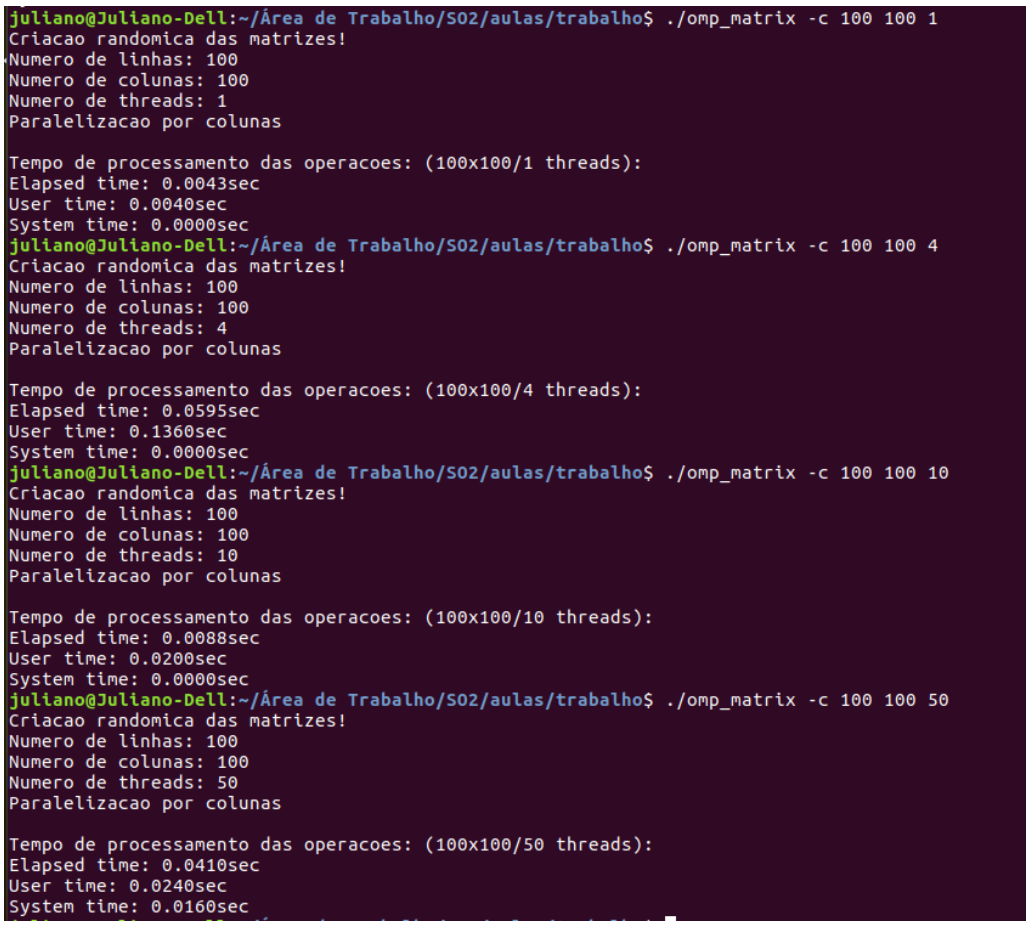
Primeiramente foram realizadas as execuções de paralelização por linhas, obtendo-se os seguintes resultados:



```
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -l 200 200 1
Criacao randomica das matrizes!
Numero de linhas: 200
Numero de colunas: 200
Numero de threads: 1
Paralelizacao por linhas

Tempo de processamento das operacoes: (200x200/1 threads):
Elapsed time: 0.0329sec
User time: 0.0320sec
System time: 0.0000sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -l 200 200 4
Criacao randomica das matrizes!
Numero de linhas: 200
Numero de colunas: 200
Numero de threads: 4
Paralelizacao por linhas

Tempo de processamento das operacoes: (200x200/4 threads):
Elapsed time: 0.0456sec
User time: 0.1040sec
System time: 0.0000sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -l 200 200 10
Criacao randomica das matrizes!
Numero de linhas: 200
Numero de colunas: 200
Numero de threads: 10
Paralelizacao por linhas

Tempo de processamento das operacoes: (200x200/10 threads):
Elapsed time: 0.0504sec
User time: 0.1080sec
System time: 0.0000sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -l 200 200 50
Criacao randomica das matrizes!
Numero de linhas: 200
Numero de colunas: 200
Numero de threads: 50
Paralelizacao por linhas

Tempo de processamento das operacoes: (200x200/50 threads):
Elapsed time: 0.0511sec
User time: 0.1160sec
System time: 0.0000sec
```

```
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -l 1500 1500 1
Criacao randomica das matrizes!
Numero de linhas: 1500
Numero de colunas: 1500
Numero de threads: 1
Paralelizacao por linhas

Tempo de processamento das operacoes: (1500x1500/1 threads):
Elapsed time: 13.4297sec
User time: 13.4080sec
System time: 0.0120sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -l 1500 1500 4
Criacao randomica das matrizes!
Numero de linhas: 1500
Numero de colunas: 1500
Numero de threads: 4
Paralelizacao por linhas

Tempo de processamento das operacoes: (1500x1500/4 threads):
Elapsed time: 17.8561sec
User time: 50.3480sec
System time: 0.1000sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -l 1500 1500 10
Criacao randomica das matrizes!
Numero de linhas: 1500
Numero de colunas: 1500
Numero de threads: 10
Paralelizacao por linhas

Tempo de processamento das operacoes: (1500x1500/10 threads):
Elapsed time: 18.2690sec
User time: 51.2960sec
System time: 0.1120sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -l 1500 1500 50
Criacao randomica das matrizes!
Numero de linhas: 1500
Numero de colunas: 1500
Numero de threads: 50
Paralelizacao por linhas

Tempo de processamento das operacoes: (1500x1500/50 threads):
Elapsed time: 19.3964sec
User time: 54.2960sec
System time: 0.1040sec
```

Posteriormente foram realizadas as execuções de paralelização por colunas, obtendo-se os seguintes resultados:



```
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -c 200 200 1
Criacao randomica das matrizes!
Numero de linhas: 200
Numero de colunas: 200
Numero de threads: 1
Paralelizacao por colunas

Tempo de processamento das operacoes: (200x200/1 threads):
Elapsed time: 0.0361sec
User time: 0.0360sec
System time: 0.0000sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -c 200 200 4
Criacao randomica das matrizes!
Numero de linhas: 200
Numero de colunas: 200
Numero de threads: 4
Paralelizacao por colunas

Tempo de processamento das operacoes: (200x200/4 threads):
Elapsed time: 0.1961sec
User time: 0.5480sec
System time: 0.0000sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -c 200 200 10
Criacao randomica das matrizes!
Numero de linhas: 200
Numero de colunas: 200
Numero de threads: 10
Paralelizacao por colunas

Tempo de processamento das operacoes: (200x200/10 threads):
Elapsed time: 0.0718sec
User time: 0.1440sec
System time: 0.0000sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -c 200 200 50
Criacao randomica das matrizes!
Numero de linhas: 200
Numero de colunas: 200
Numero de threads: 50
Paralelizacao por colunas

Tempo de processamento das operacoes: (200x200/50 threads):
Elapsed time: 0.1033sec
User time: 0.1200sec
System time: 0.0280sec
```

```
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -c 1500 1500 1
Criacao randomica das matrizes!
Numero de linhas: 1500
Numero de colunas: 1500
Numero de threads: 1
Paralelizacao por colunas

Tempo de processamento das operacoes: (1500x1500/1 threads):
Elapsed time: 13.1877sec
User time: 13.1720sec
System time: 0.0160sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -c 1500 1500 4
Criacao randomica das matrizes!
Numero de linhas: 1500
Numero de colunas: 1500
Numero de threads: 4
Paralelizacao por colunas

Tempo de processamento das operacoes: (1500x1500/4 threads):
Elapsed time: 21.4602sec
User time: 56.7920sec
System time: 0.2440sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -c 1500 1500 10
Criacao randomica das matrizes!
Numero de linhas: 1500
Numero de colunas: 1500
Numero de threads: 10
Paralelizacao por colunas

Tempo de processamento das operacoes: (1500x1500/10 threads):
Elapsed time: 19.6287sec
User time: 47.2000sec
System time: 0.3440sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -c 1500 1500 50
Criacao randomica das matrizes!
Numero de linhas: 1500
Numero de colunas: 1500
Numero de threads: 50
Paralelizacao por colunas

Tempo de processamento das operacoes: (1500x1500/50 threads):
Elapsed time: 18.6463sec
User time: 45.2560sec
System time: 0.6480sec
```

Por fim, foram realizadas as execuções de paralelização por operação, obtendo-se os seguintes resultados:

```
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -o 100 100 1
Criacao randomica das matrizes!
Numero de linhas: 100
Numero de colunas: 100
Numero de threads: 1
Paralelizacao por operacao

Tempo de processamento das operacoes: (100x100/1 threads):
Elapsed time: 0.0110sec
User time: 0.0120sec
System time: 0.0000sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -o 100 100 4
Criacao randomica das matrizes!
Numero de linhas: 100
Numero de colunas: 100
Numero de threads: 4
Paralelizacao por operacao

Tempo de processamento das operacoes: (100x100/4 threads):
Elapsed time: 0.0179sec
User time: 0.0560sec
System time: 0.0000sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -o 100 100 10
Criacao randomica das matrizes!
Numero de linhas: 100
Numero de colunas: 100
Numero de threads: 10
Paralelizacao por operacao

Tempo de processamento das operacoes: (100x100/10 threads):
Elapsed time: 0.3048sec
User time: 0.1880sec
System time: 0.3720sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -o 100 100 50
Criacao randomica das matrizes!
Numero de linhas: 100
Numero de colunas: 100
Numero de threads: 50
Paralelizacao por operacao

Tempo de processamento das operacoes: (100x100/50 threads):
Elapsed time: 1.8923sec
User time: 1.0120sec
System time: 1.8760sec
```

```
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -o 200 200 1
Criacao randomica das matrizes!
Numero de linhas: 200
Numero de colunas: 200
Numero de threads: 1
Paralelizacao por operacao

Tempo de processamento das operacoes: (200x200/1 threads):
Elapsed time: 0.0397sec
User time: 0.0320sec
System time: 0.0080sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -o 200 200 4
Criacao randomica das matrizes!
Numero de linhas: 200
Numero de colunas: 200
Numero de threads: 4
Paralelizacao por operacao

Tempo de processamento das operacoes: (200x200/4 threads):
Elapsed time: 0.2290sec
User time: 0.6280sec
System time: 0.0120sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -o 200 200 10
Criacao randomica das matrizes!
Numero de linhas: 200
Numero de colunas: 200
Numero de threads: 10
Paralelizacao por operacao

Tempo de processamento das operacoes: (200x200/10 threads):
Elapsed time: 1.3375sec
User time: 0.8760sec
System time: 1.2400sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -o 200 200 50
Criacao randomica das matrizes!
Numero de linhas: 200
Numero de colunas: 200
Numero de threads: 50
Paralelizacao por operacao

Tempo de processamento das operacoes: (200x200/50 threads):
Elapsed time: 7.7824sec
User time: 3.9040sec
System time: 7.3880sec
```

```
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -o 1500 1500 1
Criacao randomica das matrizes!
Numero de linhas: 1500
Numero de colunas: 1500
Numero de threads: 1
Paralelizacao por operacao

Tempo de processamento das operacoes: (1500x1500/1 threads):
Elapsed time: 12.4459sec
User time: 12.2280sec
System time: 0.2120sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -o 1500 1500 4
Criacao randomica das matrizes!
Numero de linhas: 1500
Numero de colunas: 1500
Numero de threads: 4
Paralelizacao por operacao

Tempo de processamento das operacoes: (1500x1500/4 threads):
Elapsed time: 31.0856sec
User time: 86.6440sec
System time: 0.9920sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -o 1500 1500 10
Criacao randomica das matrizes!
Numero de linhas: 1500
Numero de colunas: 1500
Numero de threads: 10
Paralelizacao por operacao

Tempo de processamento das operacoes: (1500x1500/10 threads):
Elapsed time: 89.6536sec
User time: 66.5760sec
System time: 72.0320sec
juliano@Juliano-Dell:~/Área de Trabalho/SO2/aulas/trabalho$ ./omp_matrix -o 1500 1500 50
Criacao randomica das matrizes!
Numero de linhas: 1500
Numero de colunas: 1500
Numero de threads: 50
Paralelizacao por operacao

Tempo de processamento das operacoes: (1500x1500/50 threads):
Elapsed time: 435.5369sec
User time: 243.6840sec
System time: 436.6120sec
```

3. Análise Crítica e Discussão

Tendo em vista os resultados obtidos, algumas conclusões se tornaram pertinentes. Na multiplicação de matrizes o *user time* é o que sofre menor alteração, pois as instruções são as mesmas. Porém, o *system time* sempre irá aumentar, pois cada vez há maior número de threads sendo criadas. Contudo, em geral, o *elapsed time* sofrerá maior alteração, ora diminuindo ora aumentando, dependendo do número de threads a serem criadas e a quantidade de linhas/colunas a serem multiplicadas. Esse é o tempo mais importante a ser analisado quando se trata de paralelização.

Por fim, o uso da paralelização é viável e traz benefícios práticos na sua utilização.

Quanto maior o custo do processamento (de acordo com a quantidade de dados), melhores são os resultados obtidos com a paralelização.

A partir de uma certa quantidade de dados, o custo de comunicação e processamento (especialmente pelo aumento de threads) aumenta significativamente. Tornando a paralelização inviável.

Outro fator importante diz respeito à lógica de programação. Neste caso, o cálculo da multiplicação das matrizes por linhas, colunas ou por operação é um fator importante na variabilidade do desempenho de execução.

Sendo assim, é importante sempre ter em mente as condições que o programa exige, bem como a lógica de programação a ser usada e, somente assim, definir qual a melhor medida de paralelização do código.

ANEXO

Este relatório acompanha o arquivo com o código da implementação. Abaixo, partes do código usado para a atividade proposta.

```
22 //Atribuicão de valores default para o número de linhas e o número de colunas
23 #define NLIN 1500
24 #define NCOL 1500
25 #define NTHR 4
26
27 //Definição do cálculo de tempo de processamento de cada atividade
28 #define toSeconds(t) (t.tv_sec + (t.tv_usec/1000000.))
29
```

```
41 int main (int argc, char *argv[])
42 {
43     //Declaração de variáveis
44     int i, j, k;                    //contadores de laço
45     int temp;                       //auxiliar de calculo que diminuirá tempo de processamento
46     i = j = k = 0;
47
48     //Estruturas para medição de processamento
49     struct timeval inic, fim;
50     struct rusage r1, r2;
51
52
53     //Passagem das linhas, colunas e threads da matriz por parametro na linha de execução
54     //  se nada for passado, utilizar valores default
55     //  caso contrario, executar apenas se todos os parametros forem passados
56     if(argc == 5)
57     {
58         //atribui os parametros apenas se forem positivos
59         _nlin = (atoi(argv[2]) >= 0) ? atoi(argv[2]) : NLIN;
60         _ncol = (atoi(argv[3]) >= 0) ? atoi(argv[3]) : NCOL;
61         _nthr = (atoi(argv[4]) >= 0) ? atoi(argv[4]) : NTHR;
62
```

```
111     //Alocação dinâmica de cada matriz utilizada no programa
112     _A = (int**) malloc(_nlin * sizeof(int*));
113     _B = (int**) malloc(_nlin * sizeof(int*));
114     _C = (int**) malloc(_nlin * sizeof(int*));
115
116     for (i=0; i<_nlin; i++)
117     {
118         _A[i] = (int*) malloc(_ncol * sizeof(int*));
119         _B[i] = (int*) malloc(_ncol * sizeof(int*));
120         _C[i] = (int*) malloc(_ncol * sizeof(int*));
121     }
122
123     //Limpeza de buffer de sementes aleatórias
124     srand(time(NULL));
125
```

```
138     //Medição de tempo e consumo de recursos antes das operações
139     gettimeofday(&inic, 0);
140     getrusage(RUSAGE_SELF, &r1);
141 
142     //Calculo da multiplicacao de matrizes
143     //Utilização de paralelismo por linhas da matriz
144     if(_linhas == 1)
145     {
146         printf("Paralelizacao por linhas\n");
147         #pragma omp parallel for private(i, j, k) num_threads(_nthr)
148         for(i=0; i<_nlin; i++)
149         {
150             for(j=0; j<_ncol; j++)
151             {
152                 _C[i][j] = 0;
153                 for(k=0; k<_nlin; k++)
154                 {
155                     temp += _A[i][j] * _B[i][j];
156                 }
157                 _C[i][j] = temp;
158                 temp = 0;
159             }
160         }
161     }
```

```
162     //paralelismo por colunas
163     else if(_colunas == 1)
164     {
165         printf("Paralelizacao por colunas\n");
166         for(i=0; i<_nlin; i++)
167         {
168             #pragma omp parallel for private(j, k) num_threads(_nthr)
169             for(j=0; j<_ncol; j++)
170             {
171                 _C[i][j] = 0;
172                 for(k=0; k<_nlin; k++)
173                 {
174                     temp += _A[i][j] * _B[i][j];
175                 }
176                 _C[i][j] = temp;
177                 temp = 0;
178             }
179         }
180     }
```

```
181     //paralelismo por operacao
182     else if(_operacao == 1)
183     {
184         printf("Paralelizacao por operacao\n");
185         for(i=0; i<_nlin; i++)
186         {
187             for(j=0; j<_ncol; j++)
188             {
189                 temp = 0;
190                 #pragma omp parallel for num_threads(_nthr) reduction (+:temp)
191                 for(k=0; k<_nlin; k++)
192                 {
193                     temp += _A[i][j] * _B[i][j];
194                 }
195                 _C[i][j] = temp;
196                 temp = 0;
197             }
198         }
199     }
200
```

```
201     //Medição de tempo e consumo de recursos depois das operações
202     gettimeofday(&fim, 0);
203     getrusage(RUSAGE_SELF, &r2);
204
205     //Exibição dos resultados
206     printf("\n");
207     printf("Tempo de processamento das operacoes: (%dx%d/%d threads): \n", _nlin, _ncol, _nthr);
208     printf("Elapsed time: %.4fsec\n", toSeconds(fim)-toSeconds(inic));
209     printf("User time: %.4fsec\n",toSeconds(r2.ru_utime)-toSeconds(r1.ru_utime));
210     printf("System time: %.4fsec\n",toSeconds(r2.ru_stime)-toSeconds(r1.ru_stime));
211
```