



Universidade do Vale do Itajaí – UNIVALI
Centro de Ciências Tecnológicas da Terra e do Mar – CTTMar
Ciência da Computação – Sistemas Operacionais– Felipe Viel

Avaliação 02 – Threads e Paralelismo

Acadêmicos: André Fragas e Pamela Bandeira Gerber

08/04/22

Introdução

O presente trabalho apresentado é um resumo técnico em que foi realizado uma implementação em C sendo que no primeiro projeto é de uma multiplicação entre matrizes utilizando o sistema single thread e multithread, no qual o último foi feito usando as bibliotecas Pthread e OpenMP e no segundo projeto é da soma de todas as posições com uma dimensão (um array 1D) utilizando também o sistema single thread e multithread, que logo após concluí-los, foram analisados e discutidos os resultados finais obtidos.

O thread é uma divisão do processo principal de um programa. Todavia, nem todos os processos são divididos em múltiplos threads, assim como nem todos os processadores são capazes de trabalhar “tranquilamente” com uma enormidade de threads.

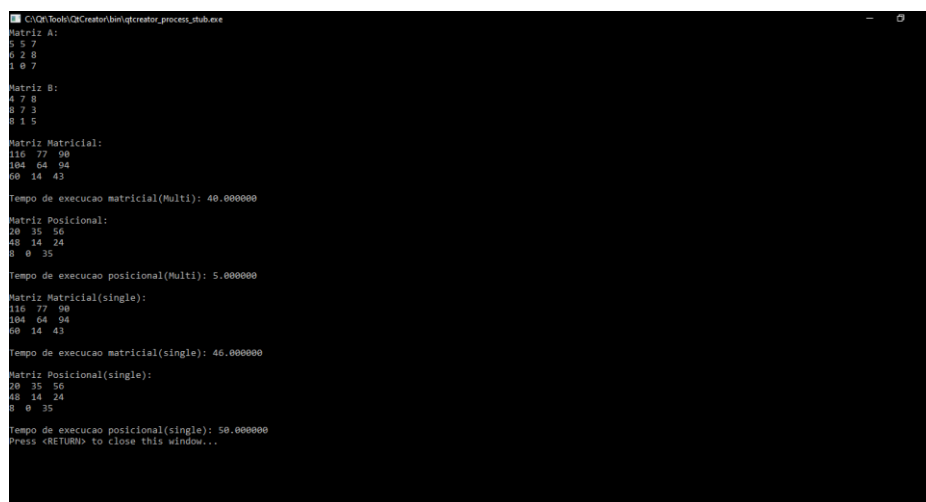
O paralelismo é sobre a execução paralela de tarefas, ou seja, de forma simultânea, a depender da quantidade de núcleos do processador. Quanto mais núcleos, mais tarefas paralelas podem ser executadas, sendo uma forma de distribuição no processamento. Também pode-se dizer que é uma forma de atingir concorrência e que cada linha de execução paralela também é concorrente, pois os núcleos estarão sendo disputados por várias outras linhas de execução e quem gerencia o que o núcleo vai executar em dado momento do tempo é o escalonador de processos.

Desenvolvimento

Projeto 1:

Conseguimos notar diferenças em questão do processamento. Na multiplicação matricial observa-se que pela lógica de multiplicação LINHA X COLUNA, que o número total de termos será o número de linhas na matriz A vezes o número de colunas em B e somamos os valores reservados desta multiplicação obtendo o valor final, já na multiplicação posicional é feito pela mesma lógica linha x coluna mas sem somar os valores da multiplicação para obter um novo resultado final. O processamento da implementação com multithread foi mais rápida que single thread. Pthread foi mais lento, porém tem a vantagem de o usuário ter mais controle na operação, podendo modificá-lo, no openmp, o código inteiro teria que ser modificado, porém foi mais rápido, pois suporta multiprocessamento de memória compartilhada em CPU para desenvolvimento de aplicações paralelas que utilizem ao máximo a capacidade do computador.

Compilação usando Pthread:



```
C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe
Matriz A:
5 5 7
8 2 0
1 0 7

Matriz B:
4 7 8
8 7 3
8 1 5

Matriz Matricial:
116 77 90
104 64 94
80 14 43

Tempo de execucao matricial(Multi): 40.000000

Matriz Posicional:
20 35 56
48 14 24
8 0 35

Tempo de execucao posicional(Multi): 5.000000

Matriz Matricial(single):
116 77 90
104 64 94
80 14 43

Tempo de execucao matricial(single): 46.000000

Matriz Posicional(single):
20 35 56
48 14 24
8 0 35

Tempo de execucao posicional(single): 50.000000
Press <RETURN> to close this window...
```

Compilação usando OpenMP:



```
main.c @ Matriz_OpenMP - Qt Creator
File Edit Build Debug Analyze Tools Window Help
Projects: Matriz_OpenMP
Sources: main.c
main.c
54 }
55
56 printf("\n");
57 printf("Matriz B: \n\n");
58 for (int i = 0; i < tam2; i++){

C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe
Tempo de execucao multi: 0.000000
Tempo de execucao single: 0.000000

Matriz A:
1 2 3
2 3 4
3 4 5

Matriz B:
2 2 2
3 3 3
4 4 4

Matriz Resultado:
20 20 20
29 29 29
38 38 38

Press <RETURN> to close this window...
```

Projeto 2:

Conseguimos notar diferenças em questão do processamento. Na soma do array, foi adicionado seus elementos antes de iniciar o tempo de execução percorrendo todo o array. Mas quando o número de elementos é muito grande, isso pode levar muito tempo, então usamos o multi-threading, onde cada núcleo do processador é usado. No nosso caso, cada núcleo avaliará a soma de uma parcela e por fim somaremos a soma de toda a parcela para obter a soma final. O processamento da implementação com multithread foi mais rápida que single thread. Pthread foi mais lento também, porém tem a vantagem de o usuário ter mais controle na operação, podendo modificá-lo, no openmp, o código inteiro teria que ser modificado, porém foi mais rápido, pois suporta multiprocessamento de memória compartilhada em CPU para desenvolvimento de aplicações paralelas que utilizem ao máximo a capacidade do computador.

Compilação usando Pthread:

```
main.c @ Array_Pthread - Qt Creator
File Edit Build Debug Analyze Tools Window Help
Projects Array_Pthread Array_Pthread.pro Sources main.c
main.c
1 #include <stdio.h>
2 #include <time.h>
3 #include <pthread.h>
4 #include <unistd.h>
5 #include <stdlib.h>
6
7 #define TamanhoArray 10

C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe
Array:
978 584 759 0 24 183 4 326 114 213
Valor da Soma: 3185
Tempo de execucao soma array: 5.000000
Array:
978 584 759 0 24 183 4 326 114 213
Valor da Soma: 3185
Tempo de execucao soma array(multi): 3.000000
Press <RETURN> to close this window...
```

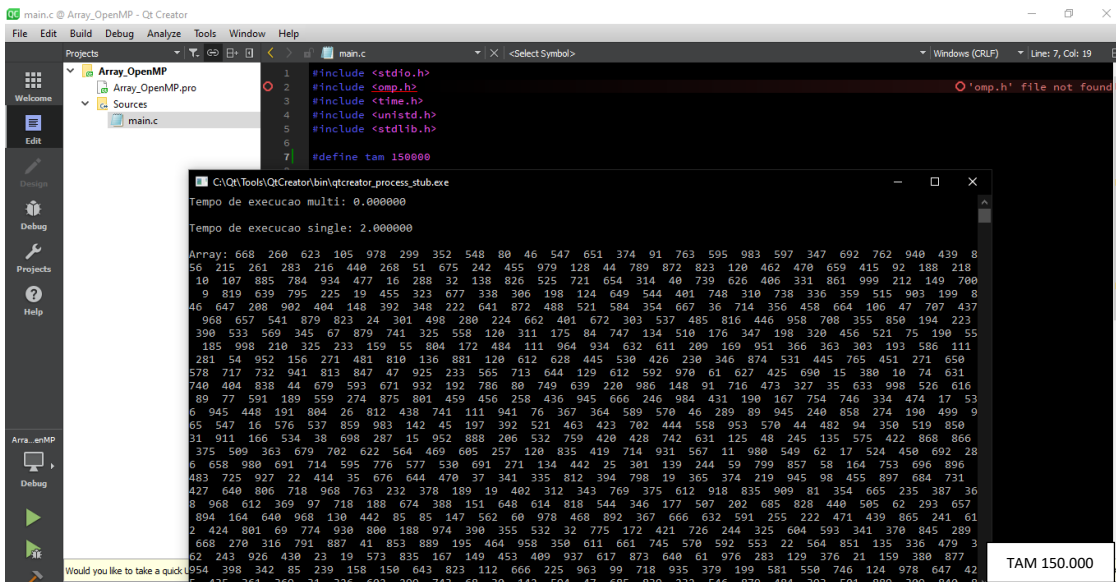
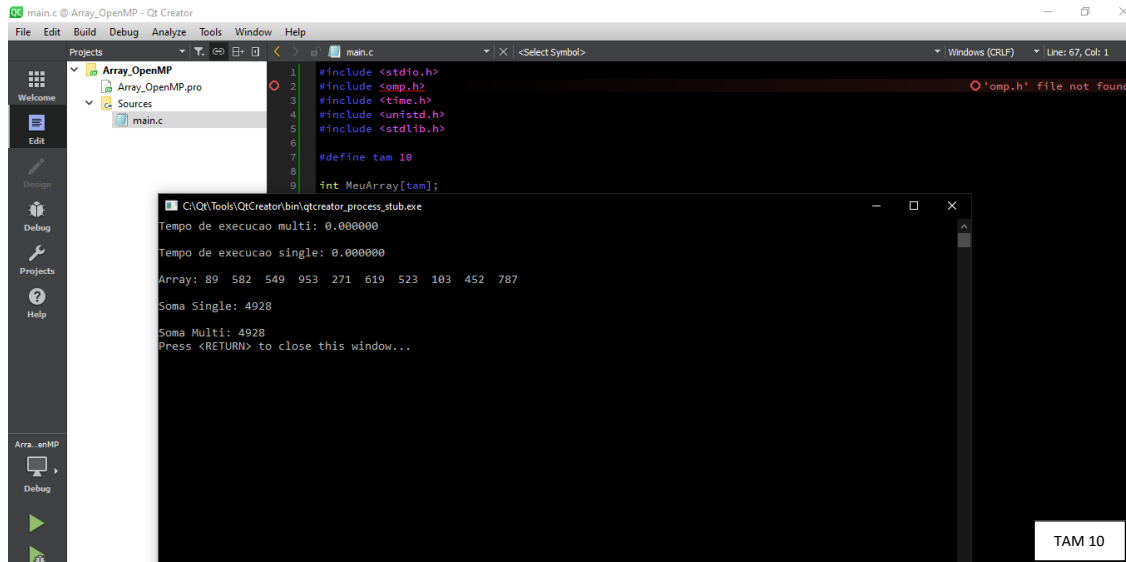
TAM 10

```
main.c @ Array_Pthread - Qt Creator
File Edit Build Debug Analyze Tools Window Help
Projects Array_Pthread Array_Pthread.pro Sources main.c
main.c
1 #include <stdio.h>
2 #include <time.h>
3 #include <pthread.h>
4 #include <unistd.h>
5 #include <stdlib.h>
6
7 #define TamanhoArray 150000

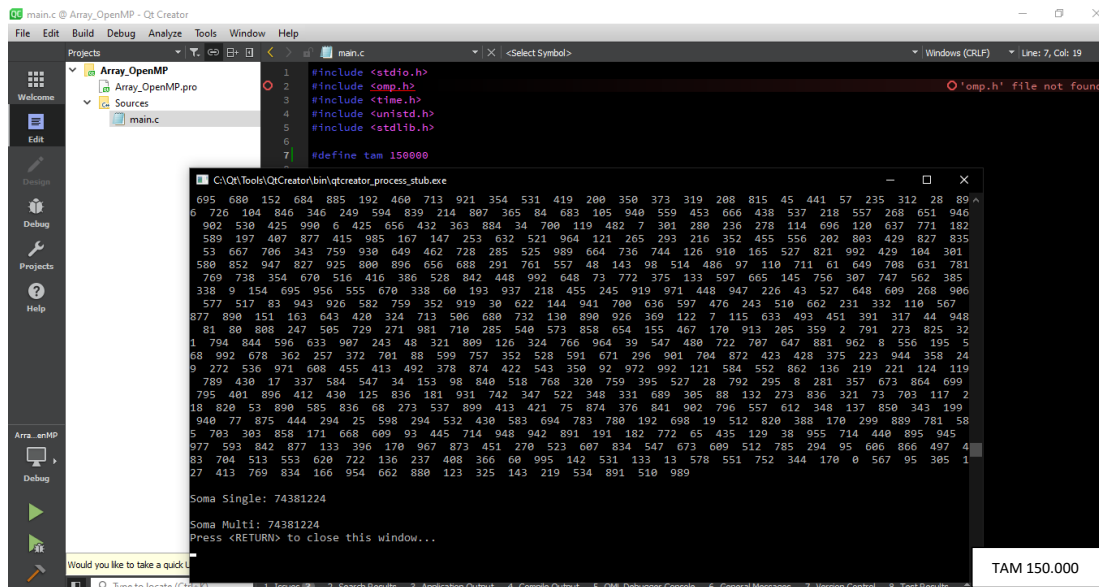
C:\Qt\Tools\QtCreator\bin\qtcreator_process_stub.exe
Valor da Soma: 74442012
Tempo de execucao soma array: 1.000000
Valor da Soma: 74442012
Tempo de execucao soma array(multi): 2.000000
Press <RETURN> to close this window...
```

TAM 150.000

Compilação usando OpenMP:



Continuação...



Códigos importantes da implementação

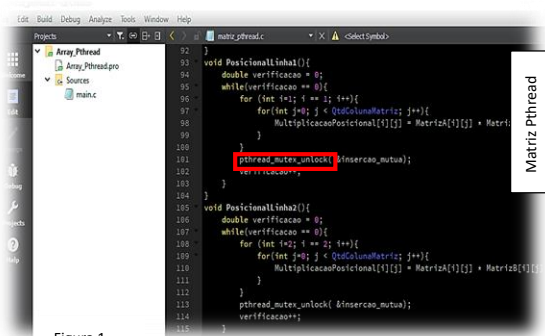


Figure 1 shows a code editor window with the file 'matriz_pthread.c'. The code is implementing a matrix multiplication using pthreads. A red box highlights the call to `pthread_mutex_unlock(&inverso_mutua);` on line 102. A vertical label 'Matriz Pthread' is on the right.

Figura 1.

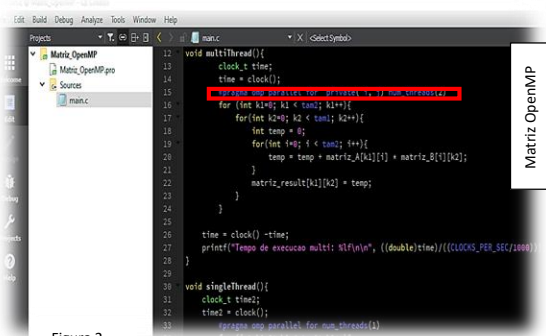


Figure 2 shows a code editor window with the file 'matriz_omp.c'. The code is implementing a matrix multiplication using OpenMP. A red box highlights the directive `#pragma omp parallel for private(i, j) num_threads(4)` on line 15. A vertical label 'Matriz OpenMP' is on the right.

Figura 2.

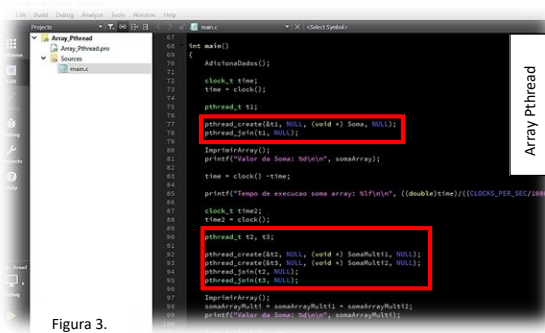


Figure 3 shows a code editor window with the file 'matriz_pthread.c'. The code is implementing a matrix multiplication using pthreads. Two red boxes highlight the calls to `pthread_create(&t1, NULL, (void *) soma, NULL);` on line 77 and `pthread_join(t1, NULL);` on line 79. A vertical label 'Array Pthread' is on the right.

Figura 3.

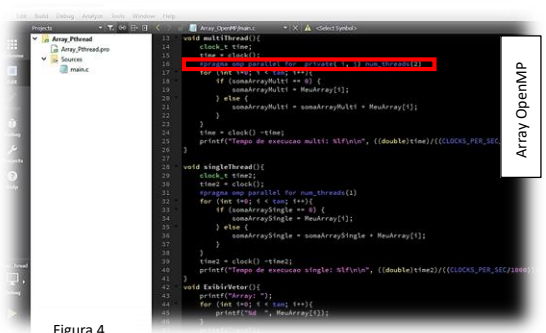


Figure 4 shows a code editor window with the file 'matriz_omp.c'. The code is implementing a matrix multiplication using OpenMP. A red box highlights the directive `#pragma omp parallel for private(i, j) num_threads(4)` on line 15. A vertical label 'Array OpenMP' is on the right.

Figura 4.

A parte mais importante é a parte de thread que cuida da sincronização, que deixa apenas uma thread executar por vez.

Figura 1. Pthread_mutex_unlock é uma função que executa tudo e depois desbloqueia a thread para outro recurso poder utilizá-lo.

Figura 2. Pragma omp instrui explicitamente o compilador a paralelizar o bloco de código escolhido.

Figura 3. Pthread_create é usado para criar uma nova thread, se valor for NULL, os atributos padrão serão usados. Se os atributos especificados por attr forem modificados posteriormente, os atributos do thread não serão afetados. Após a conclusão bem-sucedida, pthread_create() armazena o ID do thread criado no local referenciado pelo thread.

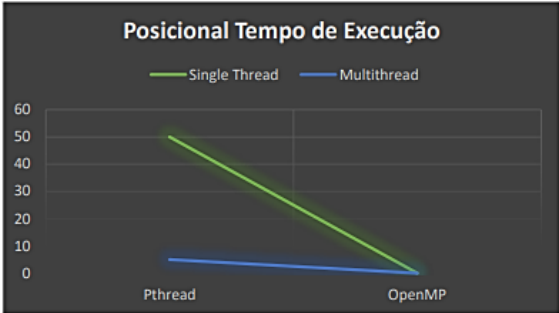
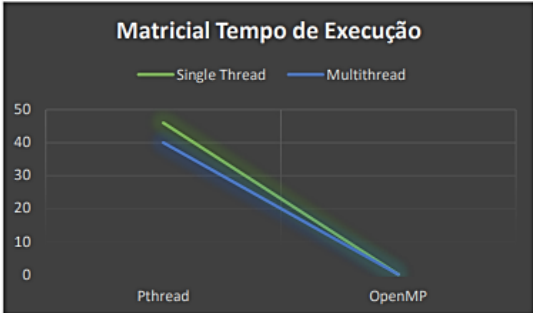
Figura 3. Pthread_join é umm chamador que tem a garantia de que o thread de destino foi encerrado.

Figura 4. Pragma omp instrui explicitamente o compilador a paralelizar o bloco de código escolhido.

Avaliação Comparativa entre Matrizes Matricial e Posicional, e Arrays.

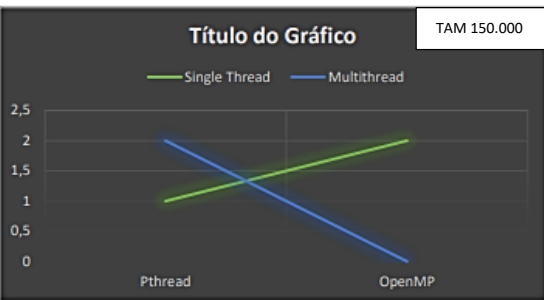
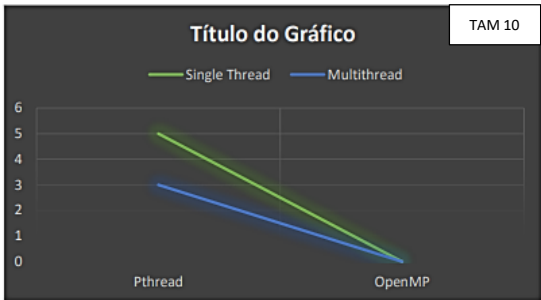
Avaliação Comparativa de Matriz Matricial		
	Pthread	OpenMP
Single Thread	46	0
Multithread	40	0

Avaliação Comparativa de Matriz Posicional		
	Pthread	OpenMP
Single Thread	50	0
Multithread	5	0



Avaliação Comparativa de Array		
	Pthread	OpenMP
Single Thread	5	0
Multithread	3	0

Avaliação Comparativa de Array		
	Pthread	OpenMP
Single Thread	1	2
Multithread	2	0



A execução single thread é muito lenta, pois toda a carga de trabalho está vinculada a uma única thread, ou seja, desta forma o programa deve executá-la por completo de uma única vez, ocasionando assim, uma demora maior do que se fosse dividida suas tarefas.

No entanto, à medida que a carga de trabalho aumenta (as matrizes ficam maiores), as opções de multithreading ficam cada vez melhores. Isto é, obviamente, porque a cada vez, mais e mais trabalho pode ser realizado em paralelo e a sobrecarga é muito pequena em comparação com o tempo de cálculo.

Conclusão

Conclui-se que na parte das threads consumiram menos recursos computacionais do que os processos. Pois a forma que é executada dentro do processador, as threads não subdividem o programa, os processos ficam estáticos esperando que outro termine fazendo com que o uso seja menor de recursos do sistema. O uso de threads é interessante quando quer executar pelo menos duas coisas ao mesmo tempo em um programa para tirar vantagem das múltiplas CPUs ou ainda para evitar que o programa inteiro fique travado ao executar uma operação demorada.

Os testes também foram essenciais para o melhor entendimento de cada aplicação, através da prática entende-se com mais intensidade o que é apresentado na disciplina de Sistemas Operacionais. Por fim pode-se compreender o funcionamento de um sistema multithreading e as ferramentas utilizadas para o gerenciamento de tal, podendo-se aplicar na prática o conhecimento teórico adquirido na presente disciplina.