UNIVERSIDADE DO VALE DE ITAJAÍ
ESCOLA POLITÉCNICA

ALEXANDRE DEBORTOLI DE SOUZA
BERNAR FREITAS DUARTE
BERNARDO LUCKMANN

THREADS E PARALELISMO

AVALIAÇÃO 01

GRADUAÇÃO EM ENGENHARIA E CIÊNCIA DA COMPUTAÇÃO
DISCIPLINA DE SISTEMAS OPERACIONAIS
PROFESSOR FELIPE VIEL

ITAJAÍ, SETEMBRO DE 2023

REPOSITÓRIO

https://github.com/bernarDuarte/repoSoTrheads

PROJETO 1

Enunciado

Realize uma implementação em **sua linguagem de preferência** de uma multiplicação entre matrizes utilizando o sistema single thread e multithread (pelo menos duas threads), no qual o último deve ser feito usando as bibliotecas **thread suportada na linguagem escolhida**. Realize uma análise comparativa no quesito tempo de processamento utilizando bibliotecas como **time.h (como o exemplo fornecido no material ou biblioteca equivalente na linguagem escolhida)**. A operação de multiplicação deve usar duas abordagens, a multiplicação matricial e a posicional, e deve ser entre, nomínimo, matrizes quadráticas de 3X3, como no exemplo apresentado e os números deve estar em float (ponto flutuante):
Matrizes a serem multiplicadas (exemplo):

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \quad e \quad B = \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

Multiplicação matricial:

$$AB = \begin{bmatrix} 1 \times 9 & 2 \times 6 & 3 \times 3 \\ 4 \times 8 & 5 \times 5 & 6 \times 2 \\ 7 \times 7 & 8 \times 4 & 9 \times 1 \end{bmatrix}$$

Multiplicação posicional:

$$A@B = \begin{bmatrix} 1 \times 9 & 2 \times 8 & 3 \times 7 \\ 4 \times 6 & 5 \times 5 & 6 \times 4 \\ 7 \times 3 & 8 \times 2 & 9 \times 1 \end{bmatrix}$$

Responda: Você conseguiu notar a diferença de processamento? O processamento (multiplicação) foi mais rápido com a implementação single thread ou multithread? Explique os resultados obtidos.
Você é livre para implementar estratégias diferentes para conseguir processar bem como usar recursos de aceleração em hardware das bibliotecas.

Solução Proposta

Criamos um programa em C que realiza a multiplicação de matrizes de duas maneiras diferentes: multiplicação matricial e multiplicação posicional. Também implementamos uma versão single-thread e uma versão multi-thread usando a biblioteca pthreads. Além disso, geramos valores aleatórios para as matrizes e medimos o tempo de processamento com a biblioteca time.h. Em seguida, comparamos os tempos de processamento para determinar qual abordagem é mais rápida.

Implementação

Função para multiplicação matricial

```
void MultiplicarMatrizMatricial(int start, int end) {
  for (int i = start; i < end; i++) {
            for (int j = 0; j < TamanhoMatriz; j++) {
            float temp = 0.0f;
            for (int k = 0; k < TamanhoMatriz; k++) {
            temp += matrizA[i][k] * matrizB[k][j];
            }
            resultado[i][j] = temp;
            }
  }
}
```

Função para multiplicação posicional

```
void MultiplicarMatrizPosicional(int start, int end) {
  for (int i = start; i < end; i++) {
            for (int j = 0; j < TamanhoMatriz; j++) {
            resultado[i][j] = matrizA[i][j] * matrizB[i][j];
            }
  }
}
```

Função para multiplicação posicional – multi-threads

```
void *ThreadMultiplication(void *args) {
  ThreadArgs *threadArgs = (ThreadArgs *)args;
  MultiplicarMatrizPosicional(threadArgs->start, threadArgs->end);
  return NULL;
}
```

Função para multiplicação matricial – multi-threads

```
void *ThreadMultiplicationMatricial(void *args) {
  ThreadArgs *threadArgs = (ThreadArgs *)args;
  MultiplicarMatrizMatricial(threadArgs->start, threadArgs->end);
  return NULL;
}
```

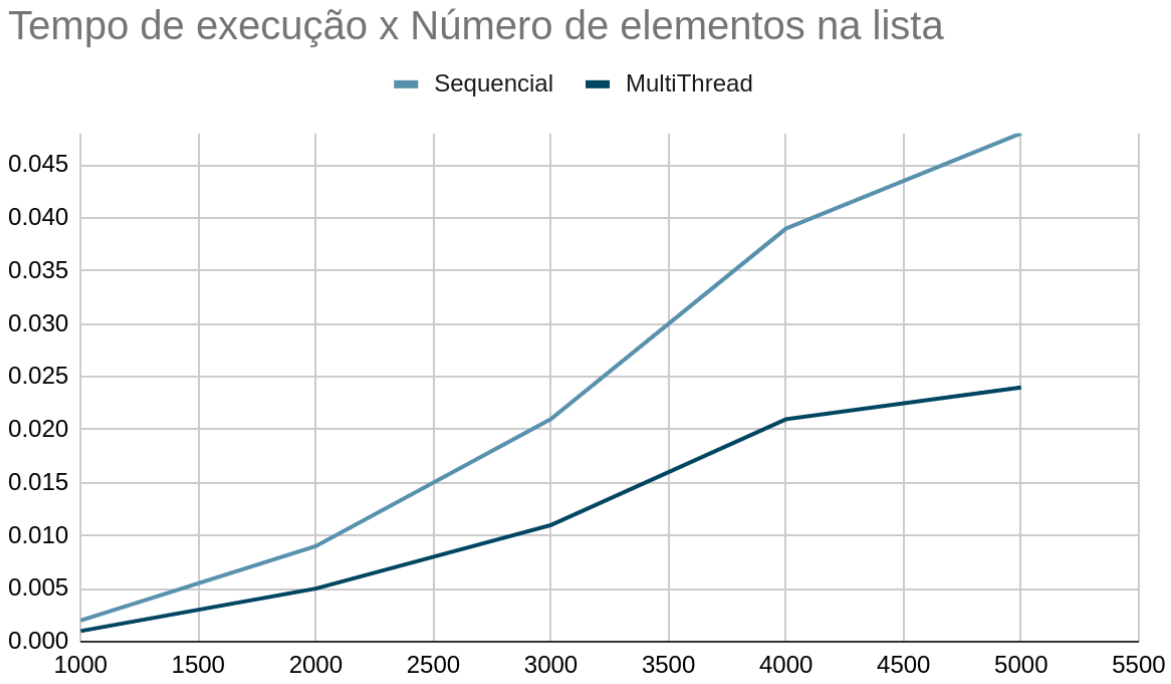
Resultados

Os testes foram feitos em uma máquina com 16 GB de RAM DDR3 e um AMD FX-8320E, foram feitas 5 execuções de cada método para as seguintes quantidades de elementos: 1000, 2000, 3000, 4000 e 5000 e as médias dos tempos de execução de cada teste formaram o seguinte gráfico:

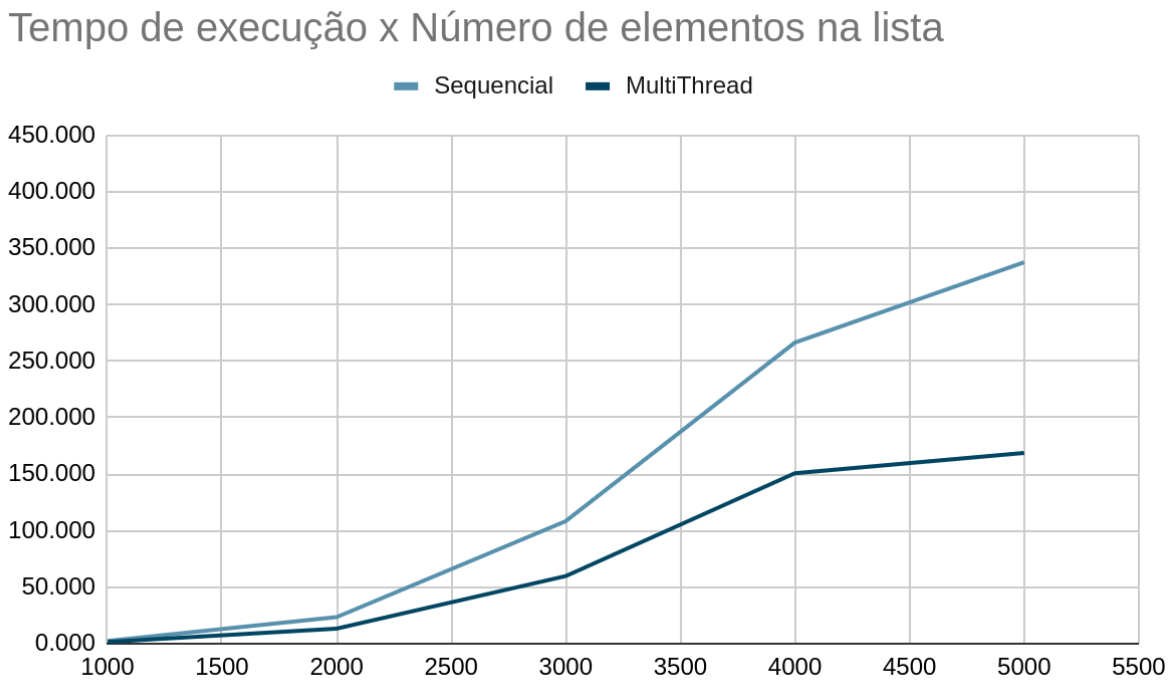
Na horizontal a quantidade de elementos na lista, e na vertical o tempo em segundos.

Tabelas:

Posicional	Sequencial	MultiThread
1000	0.002	0.001
2000	0.009	0.005
3000	0.021	0.011
4000	0.039	0.021
5000	0.048	0.024



Matricial	Sequencial	MultiThread
1000	2.456	1.392
2000	23.420	13.236
3000	108.358	59.714
4000	266.475	150.801
5000	337.500	168.700



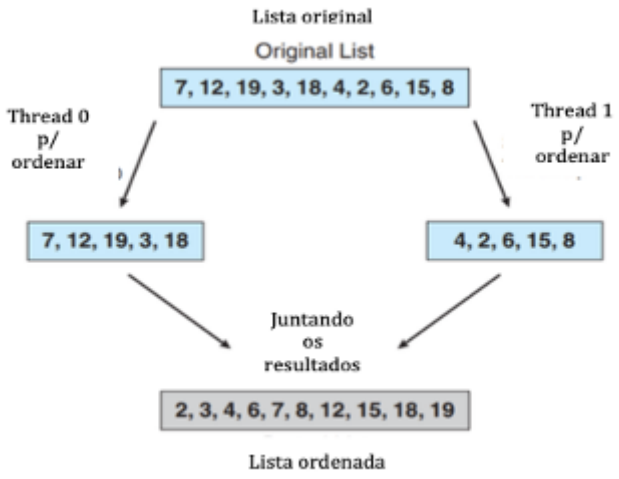
Conclusão

Com a implementação feita com 2 threads em ambas as matrizes o tempo médio é 50% a menos do que a implementação sequencial.

PROJETO 2

Enunciado

Realize uma implementação em **sua linguagem de preferência** de algoritmo de ordenação de vetores *Bubble sort*. O vetor deverá ser de pelo menos 200 posições e deverá ser comparado um sistema singlethread com um sistema multithread (com pelo menos 2 threads). Além disso, a ordenação deverá ser na ordem crescente (do maior para o menor) e o vetor de valores deve ser iniciado do maior para o menor (descrente) não importando o valor das posições, desde que respeita essa regra. Isso gerará o pior caso de uso do *bubble sort*. Exemplo do bubble sort e outros algoritmos: exemplos.



Responda: Você conseguiu notar a diferença de processamento? O processamento foi mais rápido com a implementação single thread ou multithread? Explique os resultados obtidos.
Você é livre para implementar estratégias diferentes para conseguir processar bem como usar recursos de aceleração em hardware das bibliotecas.

Solução Proposta

Esse projeto foi desenvolvido em C#, usando um projeto de console padrão do Visual Studio com .NET 6. Foram criados dois métodos, um com uma implementação estrutural do Bubble Sort e outro método que separa uma lista de inteiros em duas listas, uma contendo os menores valores da lista original e outra com os maiores valores, ainda desordenados. Para definir quais seriam os menores elementos, o maior inteiro da lista original é dividido por 2 (pois vamos usar duas Threads) e o resultado dessa divisão será usado como o critério de separação da lista original, uma contendo valores abaixo do resultado dessa divisão e outra contendo os maiores. Em seguida são criadas duas Threads, uma executa o método de Bubble Sort estrutural na lista com os menores inteiros e a outra Thread faz o mesmo, porém na lista com os maiores inteiros e no final as 2 listas são unidas, resultando na lista original, porém ordenada de maneira crescente.

Os inputs no console são: O tamanho da lista que o usuário deseja gerar, ao digitar 30000 por exemplo, será gerada uma lista em que o primeiro elemento é 30000 e vai decrescendo até 1.

Depois o usuário será perguntado se deseja realizar o Bubble Sort com o método estrutural ou se deseja executar o método que usa 2 Threads.

Ao final será exibido a lista ordenada e um contador mostrando quanto tempo levou a execução dos métodos.

A biblioteca usada foi a System.Threads para usar as Threads e System.Diagnostics para fazer o contador.

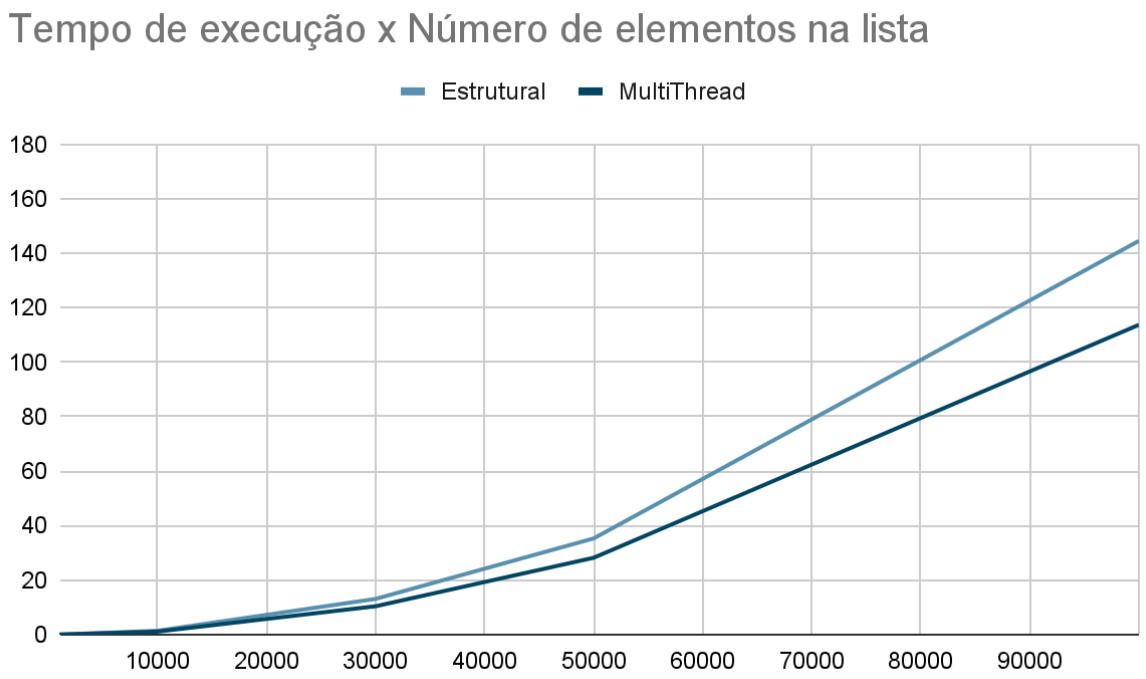
Implementação

Classe Bubble Sort com List padrão e multi-thread

```
class BubbleSort
    {
        static void Main( string[] args )
        {

            List<int> BubbleSortPadrao( List<int> list )
            {
                int temp;

                //Loop do sort
                for ( int j = 0; j <= list.Count - 2; j++ )
                {
                    for ( int i = 0; i <= list.Count - 2; i++ )
                    {
                        if ( list[i] > list[i + 1] )
                        {
                            temp = list[i + 1];
                            list[i + 1] = list[i];
                            list[i] = temp;
                        }
                    }
                }

                return list;
            }

            List<int> BubbleSortMultiThread( List<int> list )
            {
                //Cria uma lista com os menores valores
                List<int> listMenores = new List<int>();
                foreach ( int i in list )
                {
                    if( i <= list.Max() / 2 )
                        listMenores.Add( i );
                }

                //Remove os menores valores da lista principal
                list = list.Except(listMenores).ToList();


                //Uma Thread para cada lista
                Thread t1 = new Thread( () => BubbleSortPadrao( list ) );
                Thread t2 = new Thread( () => BubbleSortPadrao( listMenores ) );

                //Inicia a execução das Threads
                t1.Start();
                t2.Start();

                //Espera que a execução seja finalizada
                t1.Join();
                t2.Join();

                //Une as 2 listas
                return listMenores.Union(list).ToList();
       }
}
```

Resultados

Os testes foram feitos em uma máquina com 16 GB de RAM DDR3 e um AMD FX-8320E, foram feitas 5 execuções de cada método para as seguintes quantidades de elementos: 1000, 10000, 30000, 50000 e 100000 e as médias dos tempos de execução de cada teste formaram o seguinte gráfico:



Na horizontal a quantidade de elementos na lista, e na vertical o tempo em segundos.

Tabela:

Elementos	Estrutural	MultiThread
1000	0.01	0.03
10000	1.4	1.1
30000	13.1	10.4
50000	35.3	28.2
100000	144.5	113.7

Conclusão

Com a implementação feita, em uma lista com poucos elementos, o método estrutural executa o Sort mais rápido, porém conforme a lista de elementos fica maior há em média uma ganho de 20% no tempo de execução.