Universidade Federal do Rio Grande do Sul
Instituto de Informática
INF01008 - Programação Distribuída e Paralela

Trabalho Final

Comparação Paralelismo: C# vs Java

Inatan Lopes Hertzog 219420

1.Introdução

Nesse trabalho será comparado paralelismo e sincronização em duas linguagens distintas, Java e C#. Serão escolhidos dois algoritmos diferentes, para essa comparação com diversos numeros de cores e entradas. Em Java será utilizado Java.Threads e em C# usaremos a Biblioteca de System.Threading para comparação.

Os algoritmos escolhidos são Mergesort (recursivo) e Matrix Multiplication(iterativo).

2.Linguagens

2.1.Java

Java é uma linguagem de programação interpretada orientada a objetos. Diferente das linguagens de programação convencionais, que são compiladas para código nativo, a linguagem Java é compilada para um *bytecode* que é executado por uma máquina virtual. O desenvolvimento dos algoritmos foi utilizado o NetBeans com Java 8.

2.2.C#

C# é uma linguagem elegante e de tipos protegidos, orientada a objeto e que permite aos desenvolvedores construírem uma variedade de aplicações seguras e robustas, compatíveis com o .NET Framework. É possível usar C# para criar muito aplicativos de cliente do Windows, serviços Web XML, componentes distribuídos, aplicativos de cliente-servidor, aplicativos de banco de dados, etc. O desenvolvimento para os testes dos algoritmos foi utilizado o Visual Studio 2015, c# por ser uma linguagem da microsoft possui otimizações no Windows.

3.Algoritmos

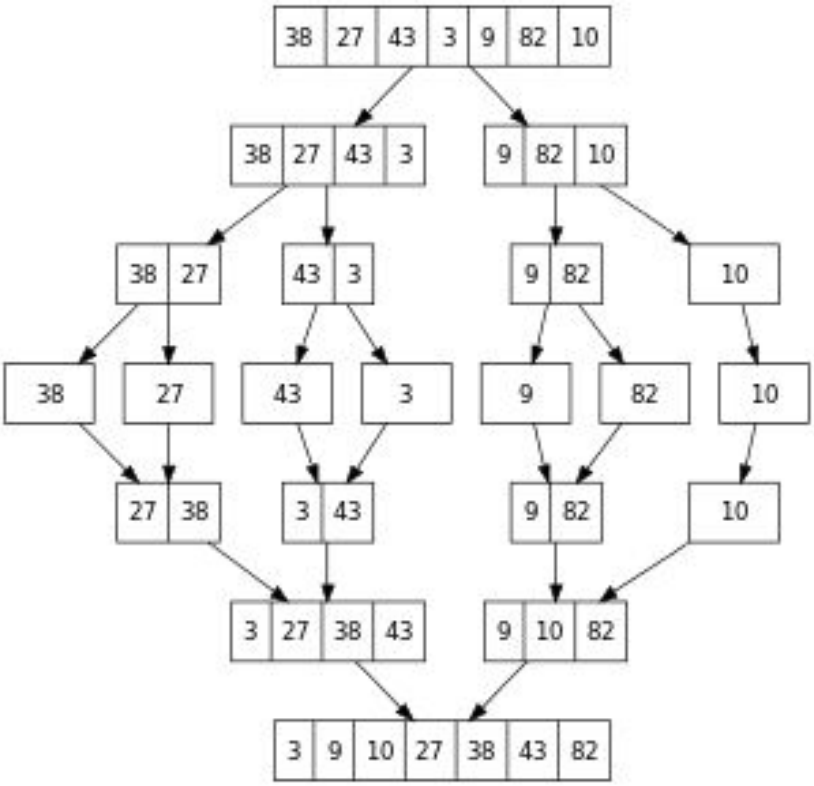
3.1.Mergesort

O mergesort, ou ordenação por mistura, sua ideia básica consiste em Dividir (o problema em vários sub-problemas e resolver esses sub-problemas através da recursividade) e Conquistar (após todos os sub-problemas terem sido resolvidos ocorre a conquista que é a união das resoluções dos sub-problemas). Como o algoritmo Mergesort utiliza a recursividade, há um alto consumo de memória e tempo de execução, tornando esta técnica não muito eficiente em alguns problemas A é complexidade O(n logn).

Uma forma de paralelizar o algoritmo, seria paralelizar as partições divididas podem realizar a ordenação em paralelo, no momento de realizar o merge das partições devem dar um join para o fluxo de processamento proceguir corretamente, o problema que essa espera pode não garantir muita eficiencia.

Os testes foram realizados com arrays de ordem invertida divididos em 4 casos:

- 100000 Elementos
- 1000000 Elementos
- 1000000 Elementos
- 10000000 Elementos



Exemplo Mergesort

3.2.Multiplicação de Matrizes

A multiplicação de matrizes é um dos algoritmos mais comuns para serem usados para paralelismo, um grande exemplo para isso são as GPUs que tem como objetivo multiplicar as matrizes para realizar processamentos de imagem ou gráfico. O método consiste que cada elemento é calculado com soma da multiplicação de cada elemento da linha da matriz A com a os elementos da coluna da matriz B.

O algoritmo de multiplicação de matrizes, contém complexidade $O(n^3)$. São 3 fors que realizam a operação para toda a matriz. A estratégia escolhida foi paralelizar um conjunto de linhas para ser processados em paralelo para buscar um melhor desempenho.

$$\begin{vmatrix} a & c \\ b & d \end{vmatrix} x \begin{vmatrix} e & g \\ f & h \end{vmatrix} = \begin{vmatrix} a*e+c*f & a*g+c*h \\ b*e+d*f & b*g+d*h \end{vmatrix}$$

Exemplo do algoritmo de multiplicação de matrizes

Os teste definem patições de linhas para ser divididas entre as threads, utilizando:

- 500 elementos
- 750 elementos
- 1000 elementos
- 1250 elementos
- 1500 elementos
- 1750 elementos
- 2000 elementos

4.Implementações

4.1.Java Mergesort

No uso de java threads uma thread executa uma das partições e cria uma outra thread para outra partição, cada partição divide por 2 o numero de threads disponiveis para essa partição. Quando o tamanho de threads for 1 é executado o algoritmo sequêncial. A classe Sorter é uma classe que executa o Mergesort quando é executada.

```
public static void parallelMergeSort(int[] a, int threadCount) {
        if (threadCount <= 1) {
                mergeSort(a);
        } else if (a.length >= 2) {
                int[] left  = Arrays.copyOfRange(a, 0, a.length / 2);
                int[] right = Arrays.copyOfRange(a, a.length / 2, a.length);
                Thread rThread = new Thread(new Sorter(right, threadCount / 2));
                parallelMergeSort(left,threadCount / 2);
                rThread.start();
                try {
                        rThread.join();
                } catch (InterruptedException ie) {}
                merge(left, right, a);
        }
}
```

Método Mergesort Paralelo

4.2.Java Multiplicação de Matrizes

A multiplicação de matrizes contém uma classe WorketTh que executa a Multiplicação de matrizes por um trecho da matriz definido. A multiplicação para cada thread é pega uma partição da matriz e depois é executado até as threads terminarem e seguirem a execução principal.

```
for(int ThreadIndex = 0 ; ThreadIndex < NUM_OF_THREADS; ThreadIndex++)
{
   thrd[ThreadIndex] = new Thread(new WorkerTh(int_num*ThreadIndex,int_num*(ThreadIndex+1), A, B, C,Msize));
   thrd[ThreadIndex].start();
}
try
{
    for (int ThreadIndex = 0 ; ThreadIndex < NUM_OF_THREADS; ThreadIndex++)
    {
        thrd[ThreadIndex].join();
    }

} catch (InterruptedException e) {}
```

Chamada da multiplicação de Matrizes

4.3.C# Mergesort

O algoritmo de mergesort nesse caso define anteriormente o numero de partições que serão divididas entre cada uma das threads. Para que todas as threads sejam sincronizadas no final, é utilizado o uso de barreiras.

```
int totalWorkers = Program.numberOfProcesses;//  Environment.ProcessorCount; // must be power of two

//worker tasks, -1 because the main thread will be used as a worker too
Task[] workers = new Task[totalWorkers - 1];

// number of iterations is determined by Log(workers), this is why th workers has to be power of 2
int iterations = (int)Math.Log(totalWorkers, 2);

// Number of elements for each array, if the elements number is not divisible by the workers, the remainde
// worker (the main thread)
int partitionSize = array.Length / totalWorkers;
int remainder = array.Length % totalWorkers;
//Barrier used to synchronize the threads after each phase
```

Definição do espaço do tamanho das partições para cada thread

A biblioteca de threading do C# utiliza uma ação que define a execução da tarefa, nesse caso a execução do Mergesort, após todas as threads terem passado a barreira, todas as partições realizam um merge.

```
Action<object> workAction = (obj) =>
{
    int index = (int)obj;

    //calculate the partition boundary
    int low = index * partitionSize;
    if (index > 0)
        low += remainder;
    int high = (index + 1) * partitionSize - 1 + remainder;

    MergeSort(array, auxArray, low, high, comparer);
    barrier.SignalAndWait();
    for (int j = 0; j < iterations; j++)
    {
        //we always remove the odd workers
        if (index % 2 == 1)
        {
            barrier.RemoveParticipant();
            break;
        }

        int newHigh = high + partitionSize / 2;
        index >>= 1; //update the index after removing the zombie workers
        Merge(array, auxArray, low, high, high + 1, newHigh, comparer);
        high = newHigh;
        barrier.SignalAndWait();
    }
```

Estrutura dentro do Action Object

Cada tarefa após de ser definida é executada, com o valor correspondente a sua id. Dentro do conjunto de tarefas. Logo após a execução de todas as threads o fluxo continua na thread main.

```
for (int i = 0; i < workers.Length; i++)
{
    workers[i] = Task.Factory.StartNew(obj => workAction(obj), i + 1);
}

workAction(0);
```

Execução das Tasks

4.4.C# Multiplicação de Matrizes

Em c# uma forma interessante de se utilizar da biblioteca de Threading é o uso de Parallel.For, uma forma mais agradável a usuário de criar threads utilizando laços. No caso de multiplicação de matrizes é utilizado a Parallel.For definindo um valor de inicio, outro de fim, o máximo de threads em paralelo e a variavel de interação.

```
static void ParalleledMatrixMultiplication(int[,] a, int[,] b, int[,] c)
{
    int s = a.GetLength(0);
    Parallel.For(0, s, new ParallelOptions { MaxDegreeOfParallelism = numberOfProcesses }, delegate (int i)
    {
        for (int j = 0; j < s; j++)
        {
            int v = 0;

            for (int k = 0; k < s; k++)
            {
                v += a[i, k] * b[k, j];
            }

            c[i, j] = v;
        }
    });
    //printMatrix(c);
}
```

Multiplicação de Matrizes utilizando Parallel.For

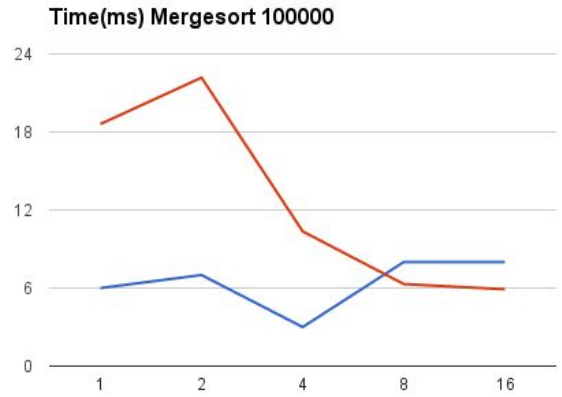
5.Máquina de testes

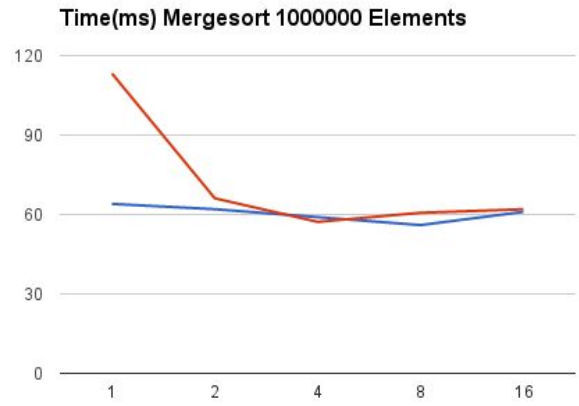
- Processador :Intel® Core™ i7-4510U
- Número de Cores Físicos: 4
- Número de Cores Lógicos: 8 (CPU com Hyper-threading)
- Memória RAM: 8 GB
- Sistema Operacional: Windows 10 64-bits

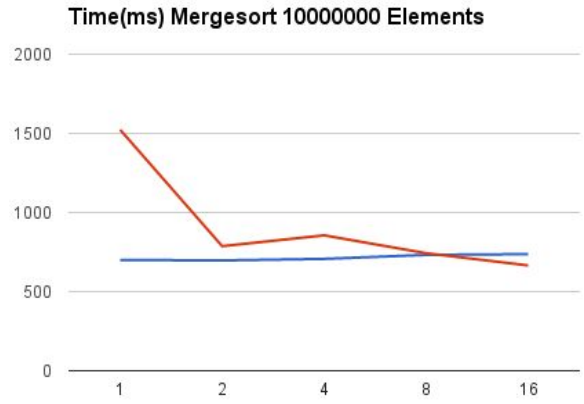
6.Benchmarks

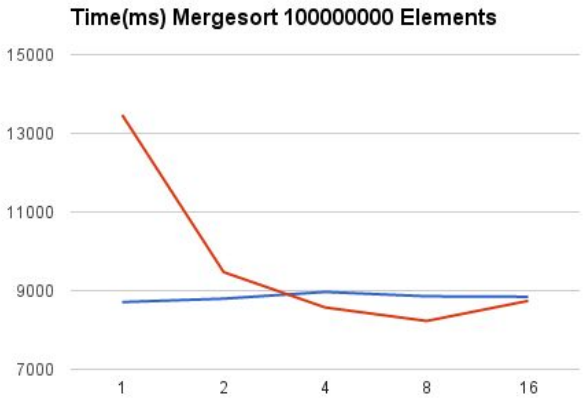
Todos os testes fora testados com 1,2,4,8 e 16 threads.Observação: Java em azul e C# em vermelho. Todos os tempos são expressos em milisegundos.

6.1.Testes de Tempos Mergesort

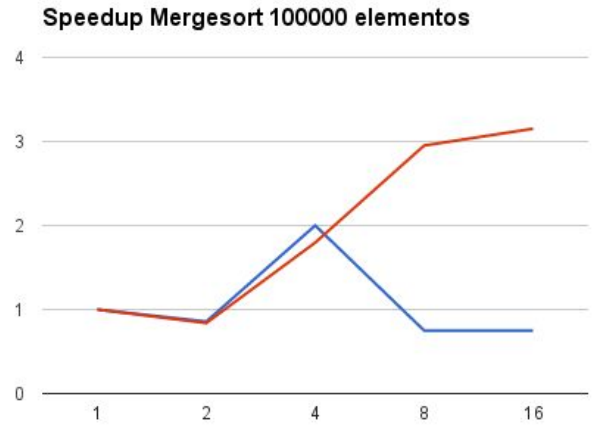


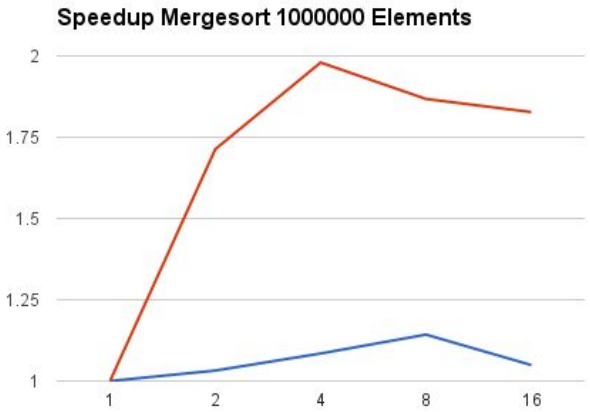


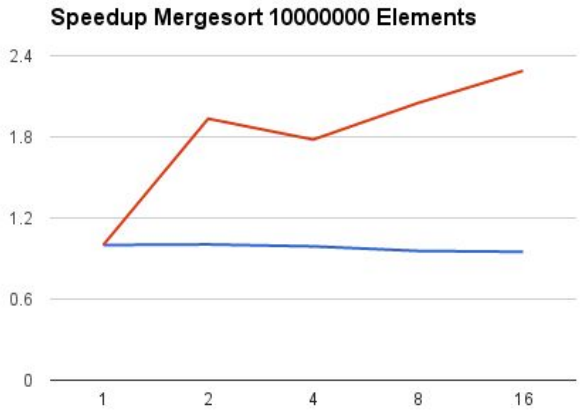


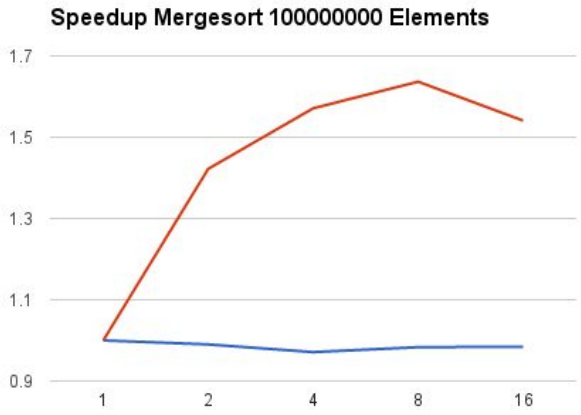


6.2.Speedup Megersort

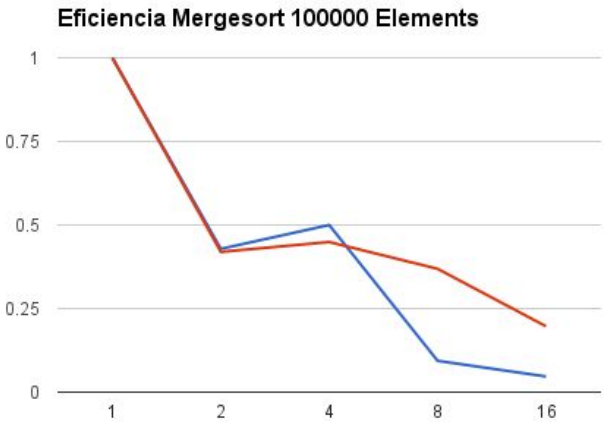


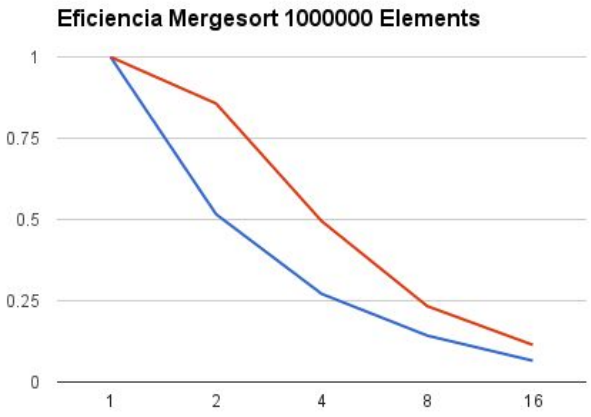


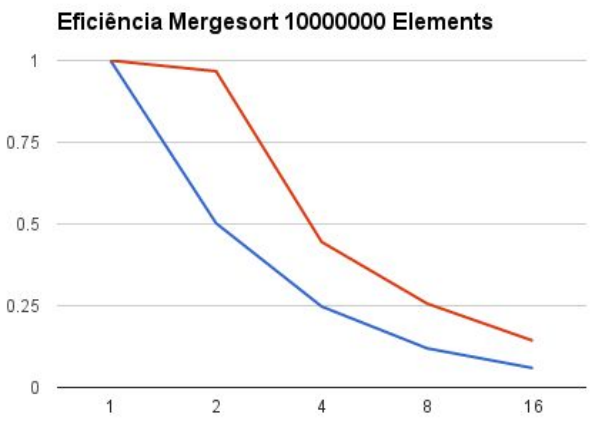


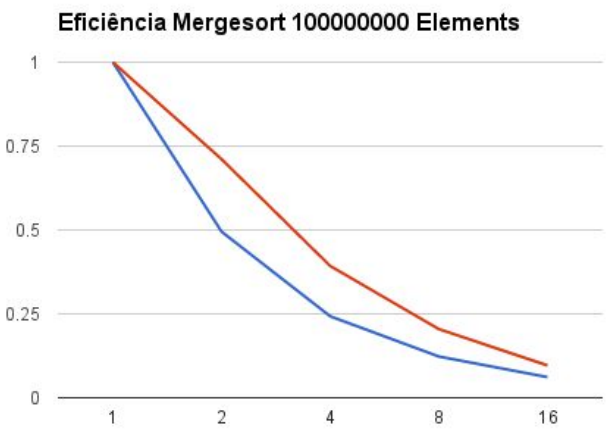


6.3.Eficiência Mergesort

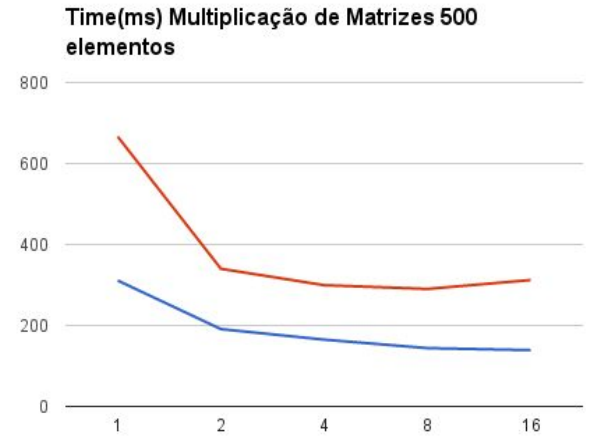


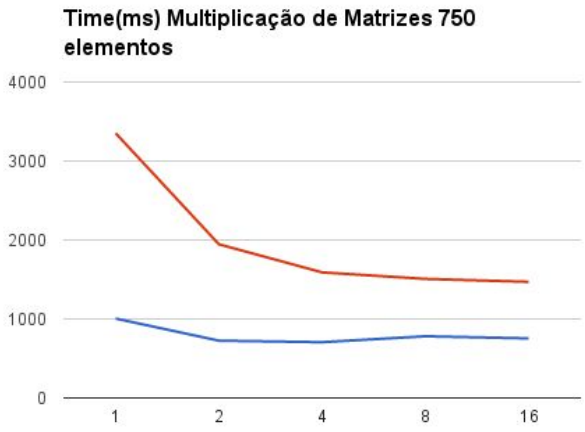


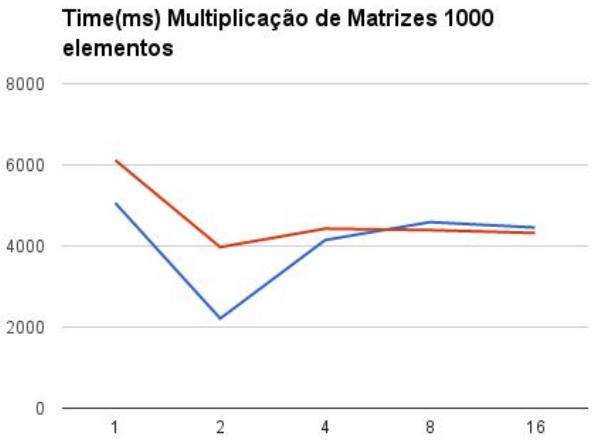


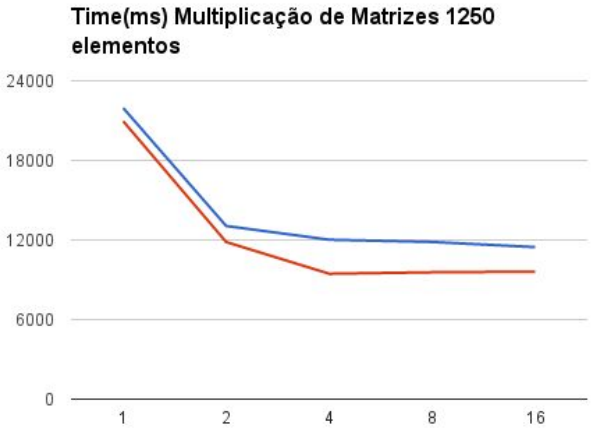


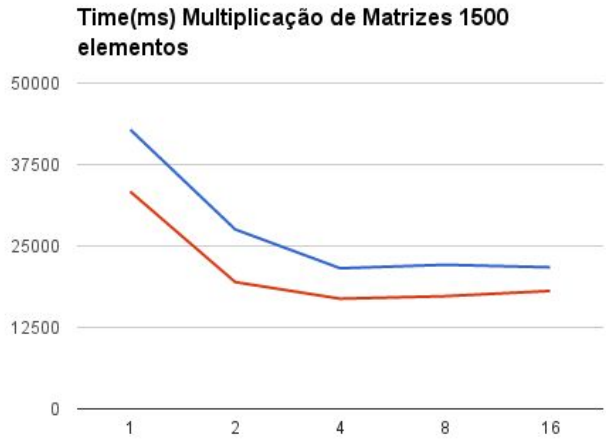
6.4.Tempo Multiplicação de Matrizes

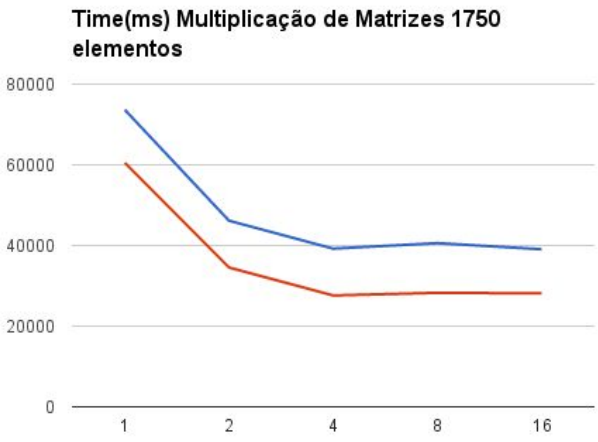


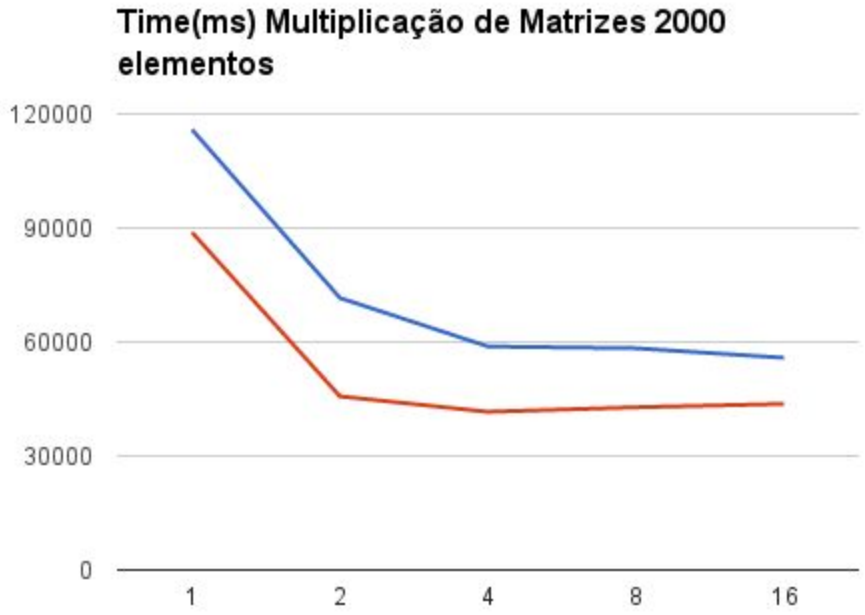




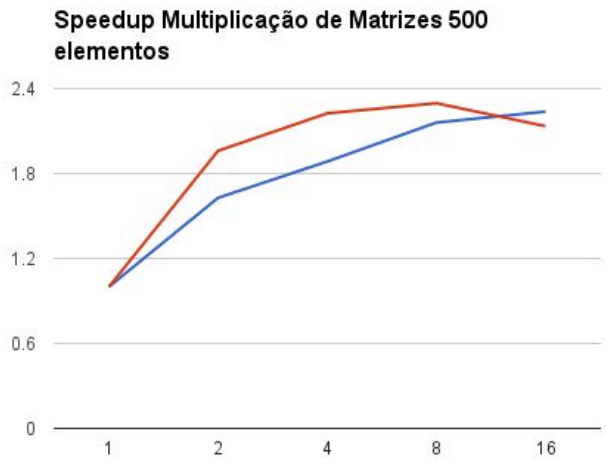


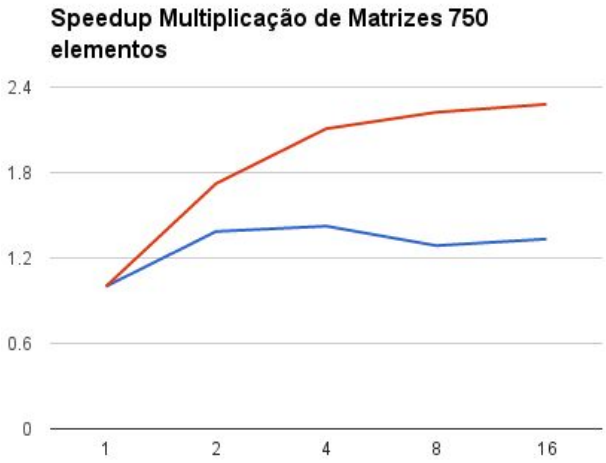

Time(ms) Multiplicação de Matrizes 1500 elementos
50000
37500
25000
12500
0
1
2
4
8
16

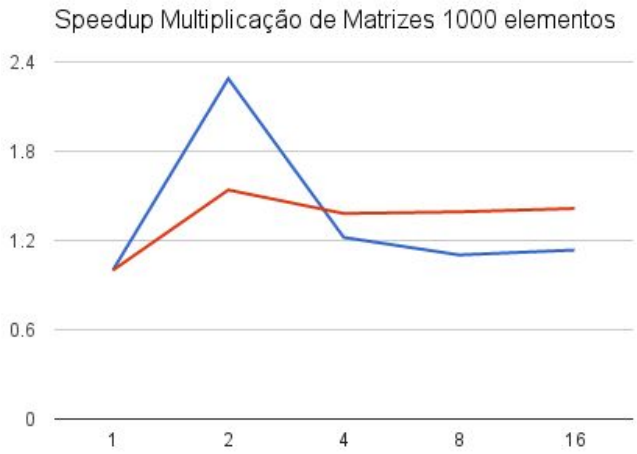

Time(ms) Multiplicação de Matrizes 1750 elementos
80000
60000
40000
20000
0
1
2
4
8
16

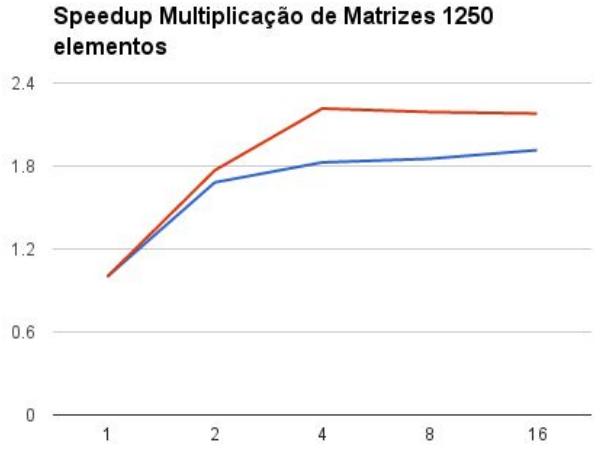

Time(ms) Multiplicação de Matrizes 2000 elementos
120000
90000
60000
30000
0
1
2
4
8
16

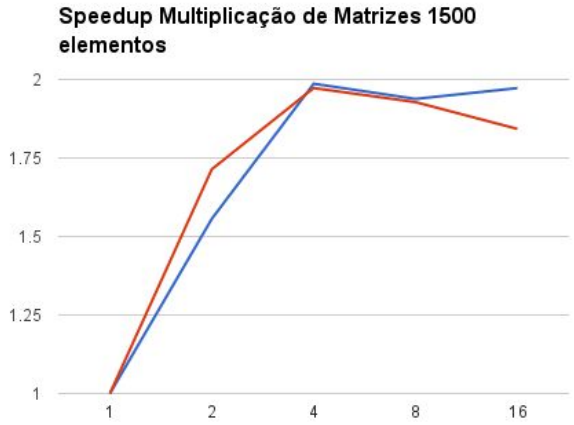
6.5.Speedup Multiplicação de Matrizes

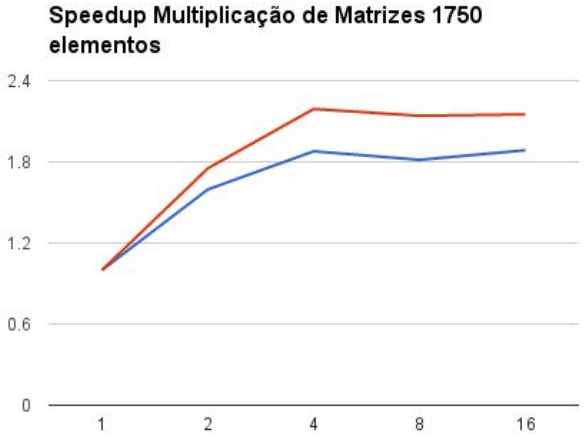


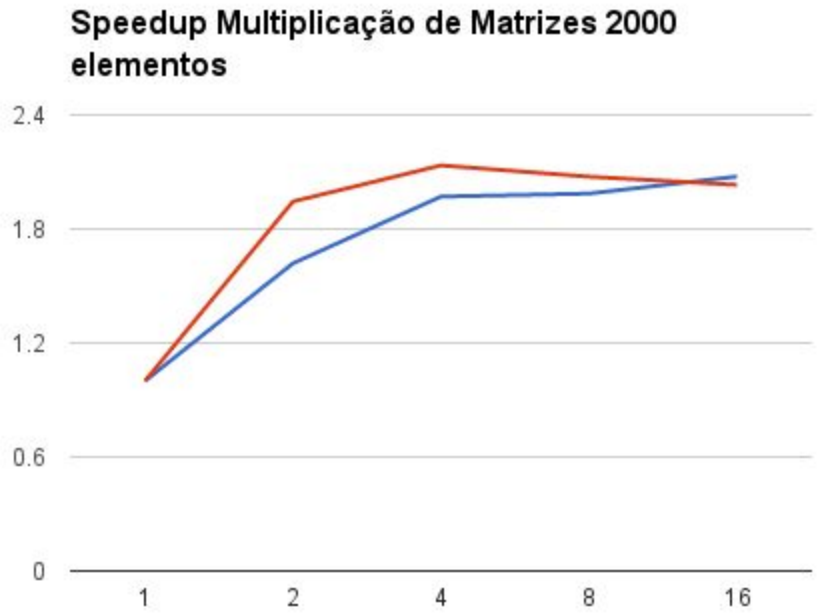




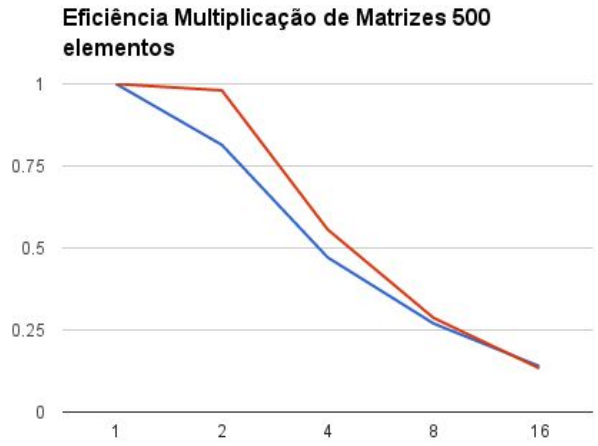


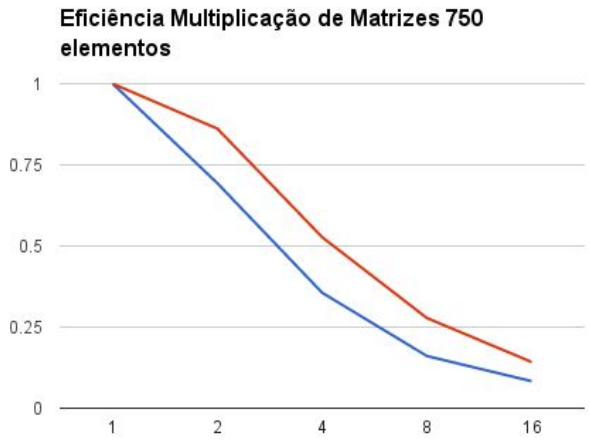


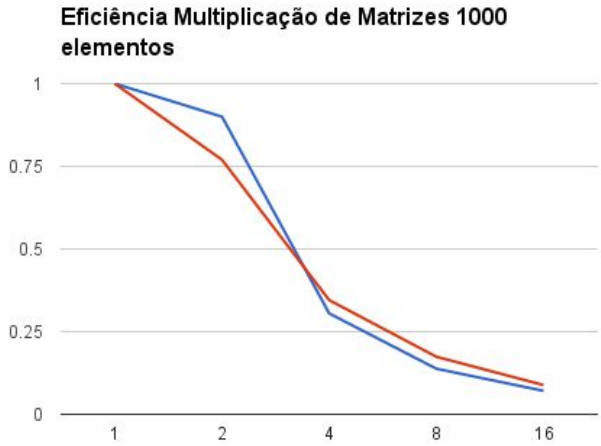


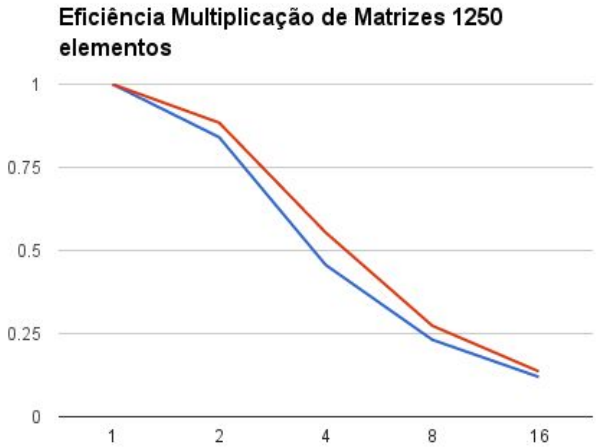


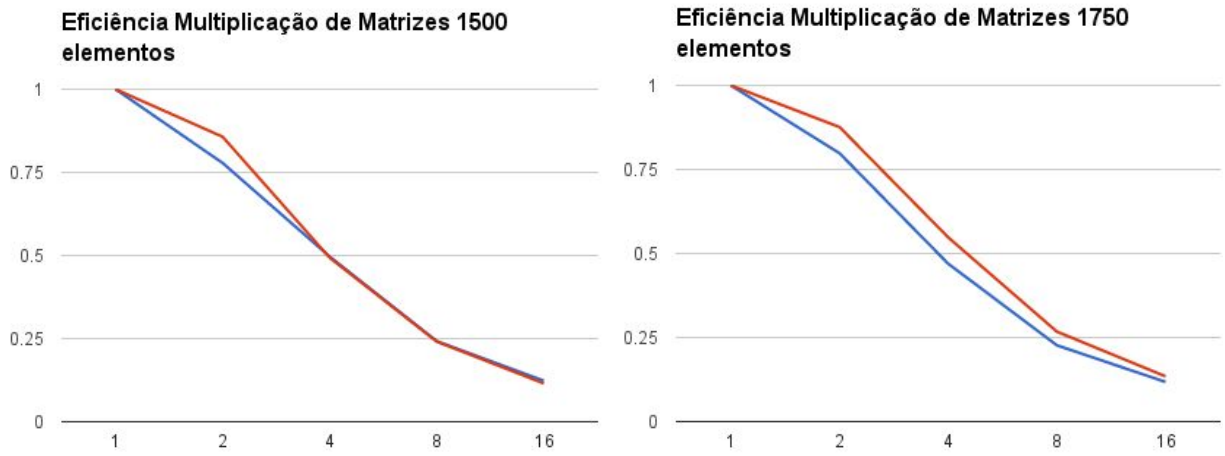
6.6.Eficiência Multiplicação de Matrizes

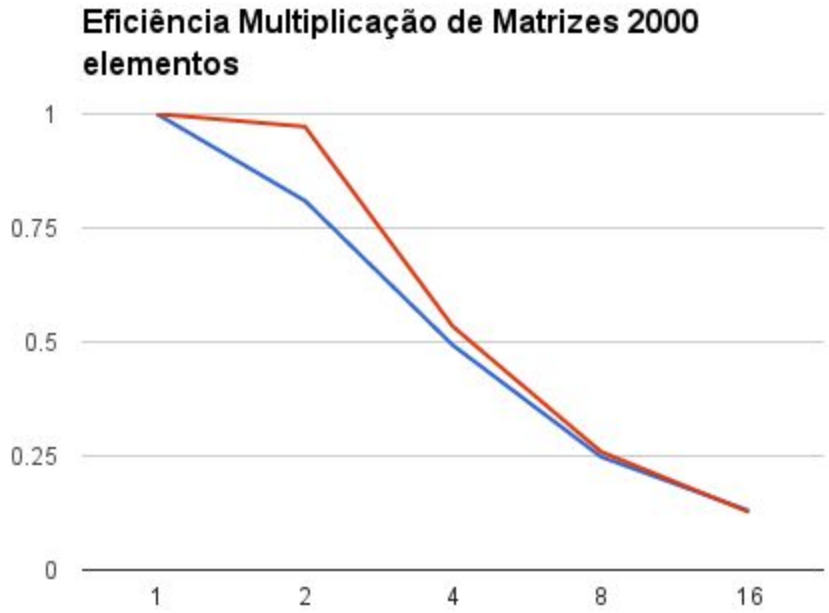












7.Conclusões

A partir das comparações realizadas, pode-se notar que o mergesort em java não teve muita eficiencia, um desses fatos pode-se inferer que a dependencia de espera entre as threads pode ter causado um overhead o qual trazia grande espera afetando o tempo de processamento. Já em C# isso não aconteceu, pois foi executado em ambiente windows o qual a linguagem possuí otimizações e a comunicação entre threads é fácilitada com o sistema operacional nesse caso.

Em casos de tamanhos menores de entradas C# não teve grande desempenho comparado a Java, isso deve-se ao fato que as criações de threads podem ser mais custosas (exemplo Parallel.For), uma vez que elas seja mais legiveis para o desenvolvedor. Mas quando

trata-se de entradas grandes C# acaba superando java, devido as otimizações para windows em gerenciamento de Threads.

O Speedup fica bem fácil de notar que c# acabou levando uma vantagem com relação a java, e sua eficiencia também, embora as duas linguagens não são teoricamente eficientes, analizando os graficos, pois uma excelente eficiencia seria uma constante em 1 e no experimento pode-se notar uma queda brusca.

Minha escolha entre as linguagens para utilizar no windows, seria c#, pois além de ter melhor desempenho, possui recursos mais interessantes para o programador como o parallel.for, por exemplo.

8.Referencias

Java

http://freesource-codes.blogspot.com.br/2011/11/matrix-multiplication-using-threading.html

https://courses.cs.washington.edu/courses/cse373/13wi/lectures/03-13/MergeSort.java

C#

http://blogs.msdn.com/b/pfxteam/archive/2011/06/07/10171827.aspx