Regressão Linear

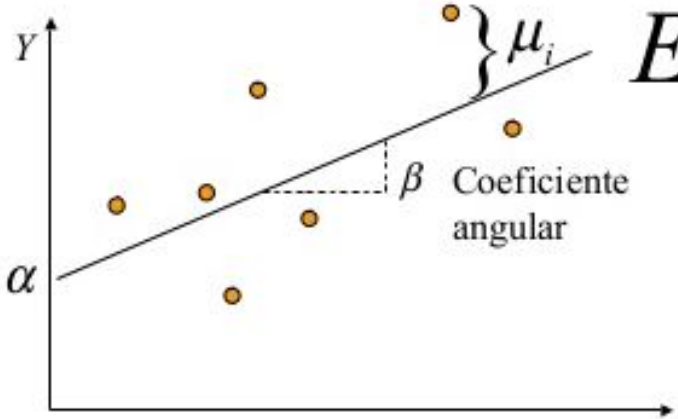
Ewerton Leandro de Sousa

Regressão Linear

É um modelo que define uma relação linear entre uma ou mais variáveis independentes com um variável dependente.

Em estatística, **regressão linear** é uma equação para se estimar a condicional (valor esperado) de uma variável *y*, dados os valores de algumas outras variáveis *x.*

Regressão Linear Simples



$$E(Y) = \alpha + \beta x_i$$

Inclinação populacional

Intercepto populacional

Erro Aleatório

$$Y_i = \alpha + \beta X_i + \mu_i$$

Variável Dependente

Variável Independente

Método dos mínimos quadrados

O método dos mínimos quadrados minimiza a soma dos quadrado dos resíduos, a ideia por trás dessa técnica é que, minimizando a soma do quadrado dos resíduos, encontraremos e que trarão a menor diferença entre a previsão de e o realmente observado.

$$S(a,b) = \sum_{i=1}^{n} (y_i - a - bx_i)^2$$

Fórmulas para encontrar os coeficientes

$$b = \frac{\sum_{i=1}^{n} (x_i \quad \bar{x})(y_i \quad \bar{y})}{\sum_{i=1}^{n} (x_i \quad \bar{x})^2}$$

$$a = \bar{y} - b\bar{x}$$

Implementação Serial

Gerar dados

```
// Gerar dataset
  for(int i=0; i<tamanho; i++){

     struct timespec seedtime1, seedtime2;

     clock_gettime(CLOCK_MONOTONIC, &seedtime1);
     unsigned int seedX = (unsigned int) seedtime1.tv_nsec;

     X[i] = (double)(rand_r(&seedX) % 10000) / 10;

     clock_gettime(CLOCK_MONOTONIC, &seedtime2);
     unsigned int seedErro = (unsigned int) seedtime2.tv_nsec;

     double erro = (double)(rand_r(&seedErro) % 500);

    ....
```

```
...

     if(erro > 0 && erro < 4){

       clock_gettime(CLOCK_MONOTONIC, &seedtime2);
       unsigned int seedErro = (unsigned int) seedtime2.tv_nsec;

       erro = (double)(rand_r(&seedErro) % 1000)/10;

       y[i] = a + (b * X[i]) + erro;
     }else if(erro > 4 && erro < 10){

       clock_gettime(CLOCK_MONOTONIC, &seedtime2);
       unsigned int seedErro = (unsigned int) seedtime2.tv_nsec;

       erro = (double)(rand_r(&seedErro) % 1000)/10;

       y[i] = a + (b * X[i]) - erro;
     }else{
       y[i] = a + (b * X[i]);
     }

  }
```

Média de X e Y

```
// Calcula media de X
 double media_X = 0;

 for(int i=0; i<tamanho; i++){

     media_X += X[i];

 }

 media_X /= tamanho;

 printf("Media de X: %f\n", media_X);
```

```
// Calcula media de y
double media_y = 0;

for(int i=0; i<tamanho; i++){

    media_y += y[i];

}

media_y /= tamanho;

printf("Media de y: %f\n", media_y);
```

Calculo de B

$$b = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}$$

```
// Calcular b
 double dividendo = 0;
 double divisor = 0;

 for(int i=0; i<tamanho; i++){

     dividendo = (X[i] - media_X) * (y[i] - media_y);

     divisor = (X[i] - media_X) * (X[i] - media_X);

 }

 double b_estimado = dividendo / divisor;

 printf("B real: %f\nB estimado: %f\n", b, b_estimado);
```

Calculo de A

$$a = \bar{y} - b\bar{x}$$

```
// Calcular a
 double a_estimado = media_y - (b_estimado * media_X);

 printf("A real: %f\nA estimado: %f\n", a, a_estimado);
```

Implementação Paralela

Gerar dados

```
// Gerar dataset
  #pragma omp parallel for shared(X, y, tamanho, a, b) \
   default(none) num_threads(num_threads) schedule(static, 1)

  for(int i=0; i<tamanho; i++){

     struct timespec seedtime1, seedtime2;

     clock_gettime(CLOCK_MONOTONIC, &seedtime1);
     unsigned int seedX = (unsigned int) seedtime1.tv_nsec;

     X[i] = (double)(rand_r(&seedX) % 10000) / 10;

     clock_gettime(CLOCK_MONOTONIC, &seedtime2);
     unsigned int seedErro = (unsigned int) seedtime2.tv_nsec;

     double erro = (double)(rand_r(&seedErro) % 40);

     ….
```

```
...

 if(erro > 0 && erro < 4){

       clock_gettime(CLOCK_MONOTONIC, &seedtime2);
       unsigned int seedErro = (unsigned int) seedtime2.tv_nsec;
       erro = (double)(rand_r(&seedErro) % 1000)/10;

       y[i] = a + (b * X[i]) + erro;
     }else if(erro > 4 && erro < 10){

       clock_gettime(CLOCK_MONOTONIC, &seedtime2);
       unsigned int seedErro = (unsigned int) seedtime2.tv_nsec;
       erro = (double)(rand_r(&seedErro) % 1000)/10;

       y[i] = a + (b * X[i]) - erro;
     }else{
       y[i] = a + (b * X[i]);
     }

 }
```

Média de X e Y

```
// Calcula media de X
  double media_X = 0;
  #pragma omp parallel num_threads(num_threads)
  {
    #pragma omp parallel for shared(X, i, tamanho)
default(none) reduction(+:media_X)
num_threads(num_threads)
    for(int i=0; i<tamanho; i++){
        media_X += X[i];
    }
    media_X /= tamanho;
    printf("Media de X: %f\n", media_X);
  }
```

```
  // Calcula media de y
  double media_y = 0;
  #pragma omp parallel num_threads(num_threads)
  {
    #pragma omp parallel for shared(y, i, tamanho) default(none)
reduction(+:media_y) num_threads(num_threads)
    for(int i=0; i<tamanho; i++){
        media_y += y[i];
    }
    media_y /= tamanho;
    printf("Media de y: %f\n", media_y);
  }
```

Calculo de A e B

$$b = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}$$

$$a = \bar{y} - b\bar{x}$$

```
// Calcular b
  double dividendo = 0;
  double divisor = 0;
  double b_estimado = 0;
  double a_estimado = 0;
  #pragma omp parallel num_threads(num_threads)
  {

    #pragma omp parallel for shared(X, media_X, y, media_y, i,
tamanho) default(none) reduction(+:dividendo) reduction(+:divisor)
num_threads(num_threads)
    for(int i=0; i<tamanho; i++){

        dividendo = (X[i] - media_X) * (y[i] - media_y);

        divisor = (X[i] - media_X) * (X[i] - media_X);
    }
    double b_estimado = dividendo / divisor;
    printf("B real: %f\nB estimado: %f\n", b, b_estimado);

    // Calcular a
    double a_estimado = media_y - (b_estimado * media_X);
    printf("A real: %f\nA estimado: %f\n", a, a_estimado);

  }
```

Verificação de corretude

```
[eldsousa@service0 Serial]$ ./RegressaoLinearSerial.exe 1000 1
Valores utilizados: y = a + b * x + E
A: 118.900000 <> B: 10.600000
y = 118.900000 + 10.600000 * x + E
Media de X: 497.238000
Media de y: 5389.275500
B real: 10.600000
B estimado: 10.603167
A real: 118.900000
A estimado: 116.977945

Tempo: 0.000174 seg.

[eldsousa@service0 Serial]$
```

Serial

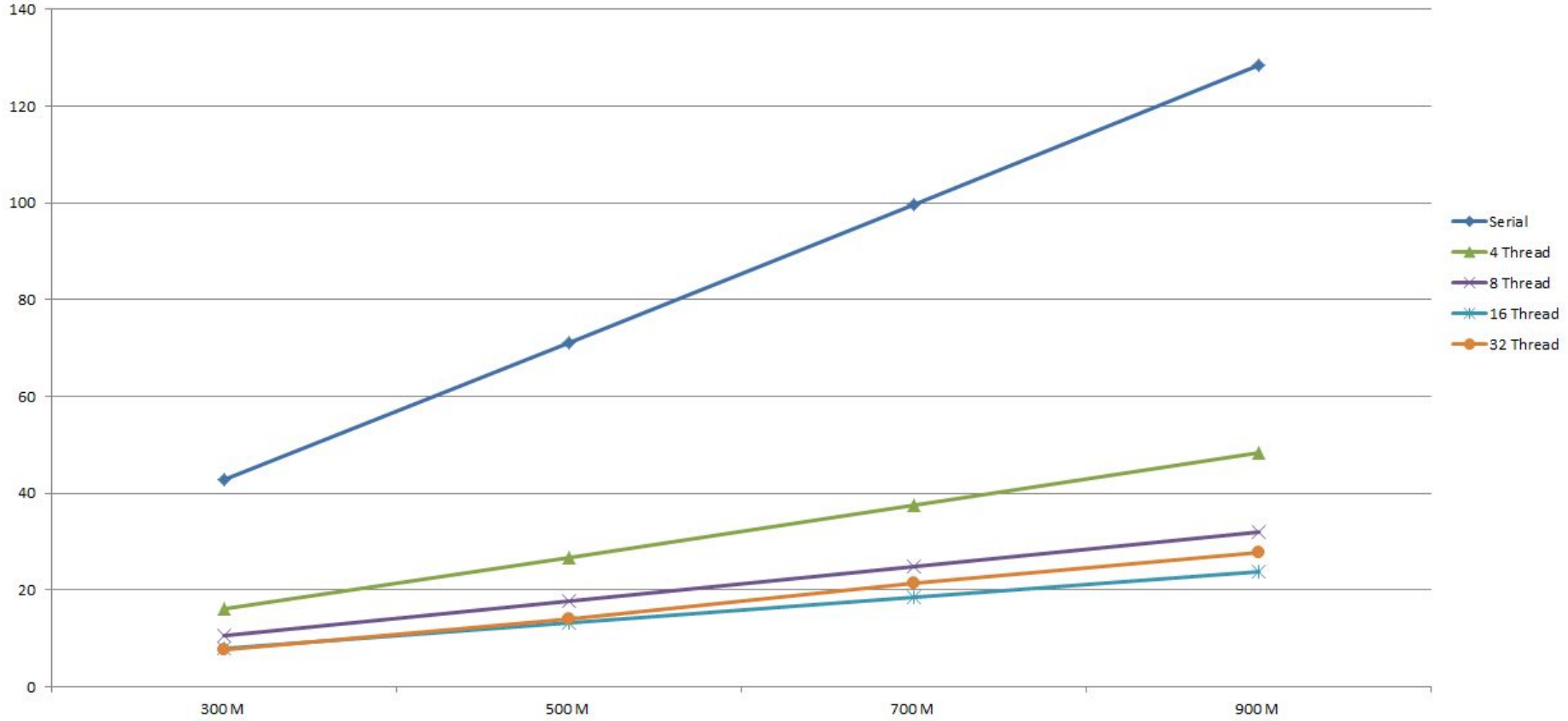
Paralelo

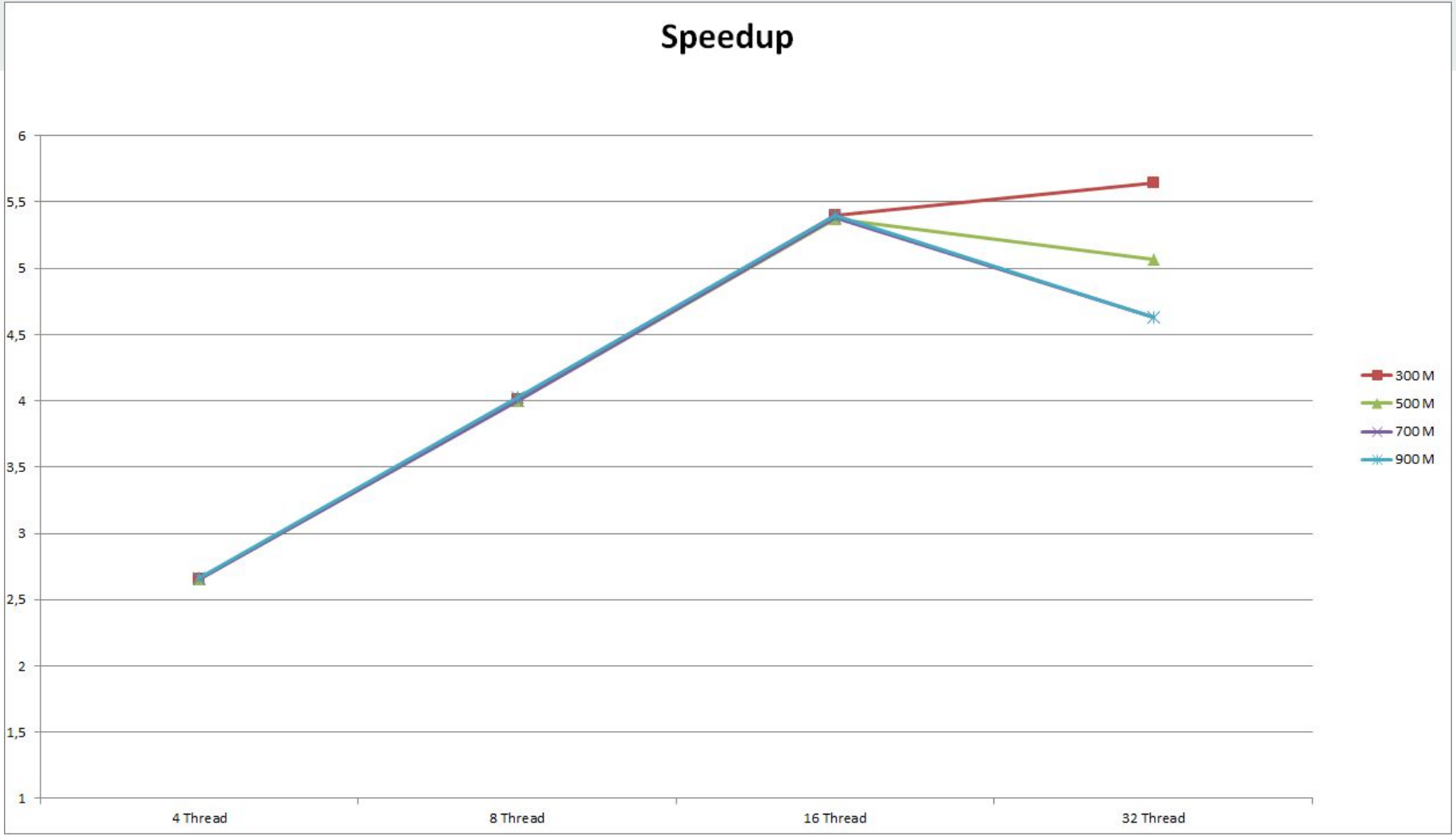
```
[eldsousa@service0 Paralelo]$ ./RegresaoLinearParalela.exe 1000 2
Valores utilizados: y = a + b * x + E
A: 90.000000 <> B: 30.500000
y = 90.000000 + 30.500000 * x + E
Media de X: 504.504800
Media de X: 504.504800
Media de y: 15475.034600
Media de y: 15475.034600
B real: 30.500000
B estimado: 30.495043
A real: 90.000000
A estimado: 90.138782
B real: 30.500000
B estimado: 30.495043
A real: 90.000000
A estimado: 90.138782

Tempo: 0.000398 seg.

[eldsousa@service0 Paralelo]$
```

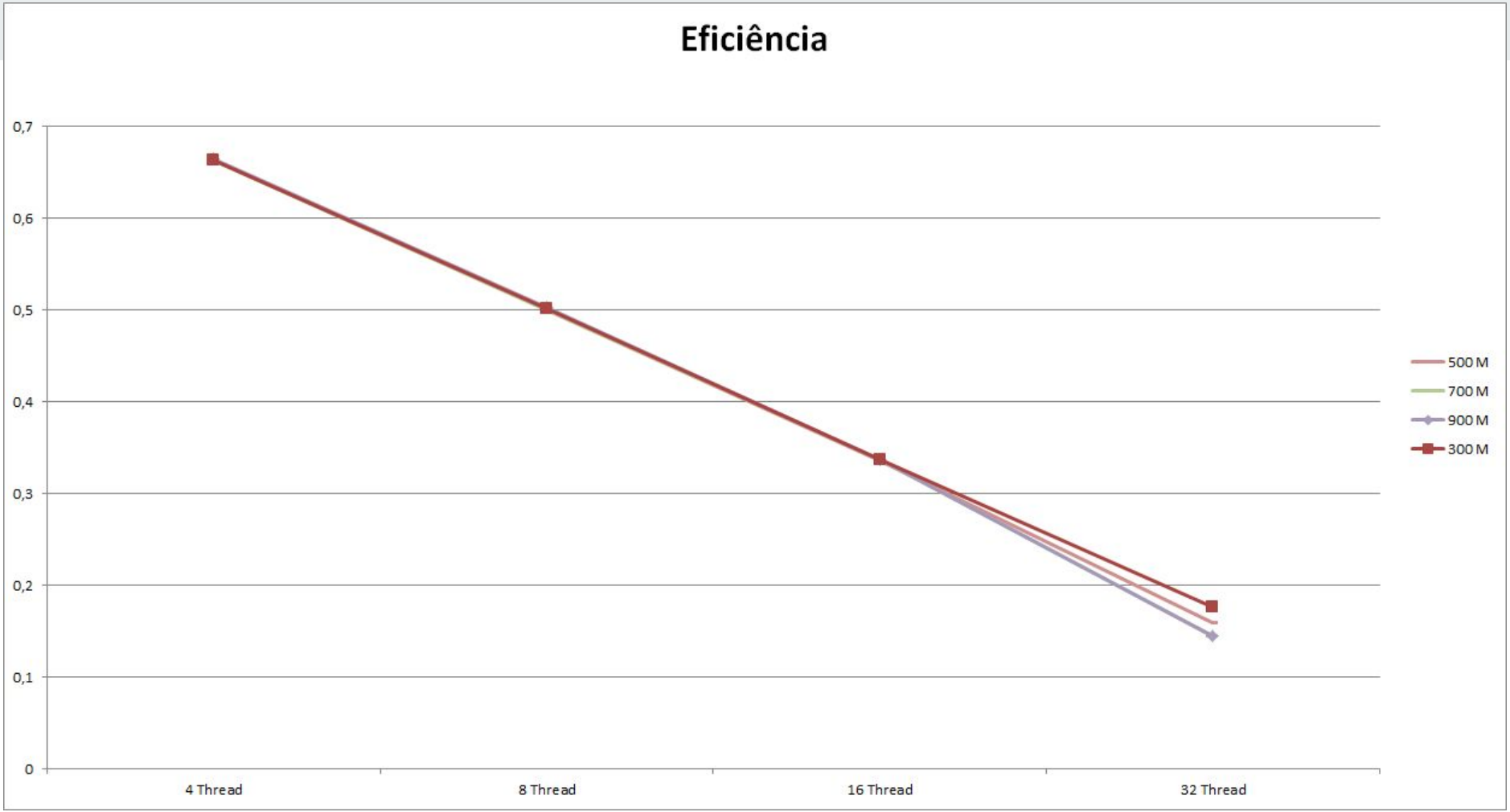

Comparação do resultados
30000
25000
20000
15000
10000
5000
0
0
200
400
600
800
1000
1200
Dados
y = 27,800x + 14.1
Excel
y = 27,795x + 13,23
Algoritmo
y = 27,81x + 5,24

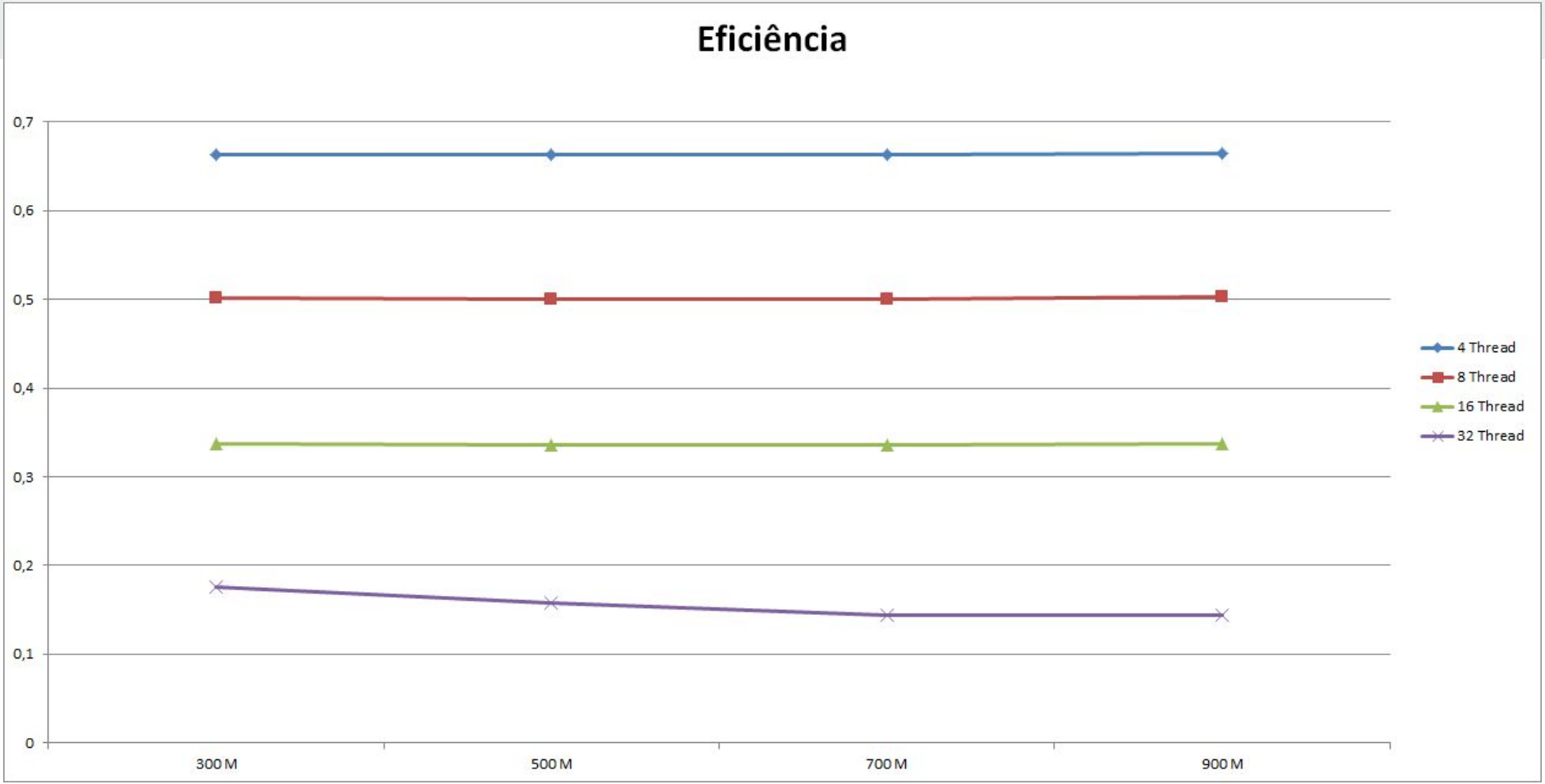
Analises

Tempo vs tamanho




Speedup
6
5,5
5
4,5
4
3,5
3
2,5
2
1,5
1
4 Thread
8 Thread
16 Thread
32 Thread
300 M
500 M
700 M
900 M


Eficiência
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
4 Thread
8 Thread
16 Thread
32 Thread
500 M
700 M
900 M
300 M


Eficiência
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0
300 M
500 M
700 M
900 M
4 Thread
8 Thread
16 Thread
32 Thread

Conclusão

A partir das análises de tempo de execução, speedup e eficiência, podemos observar um ganho de desempenho significativo e na análise da eficiência se observar um pequena perda na eficiência, então podemos concluir que o algoritmo é escalável.