Universidade Federal do Rio Grande do Norte
Instituto Metrópole Digital

Comparação entre desempenho de implementação serial e paralela utilizando OpenMP

Aluno: Ewerton Leandro de Sousa

Natal-RN, Brasil
Dezembro de 2020

1. Introdução

Para a análise de speedup, eficiência e escalabilidade, foi proposto o problema do cálculo de uma regressão linear utilizando o método dos mínimos quadrados.

1.1. 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.*

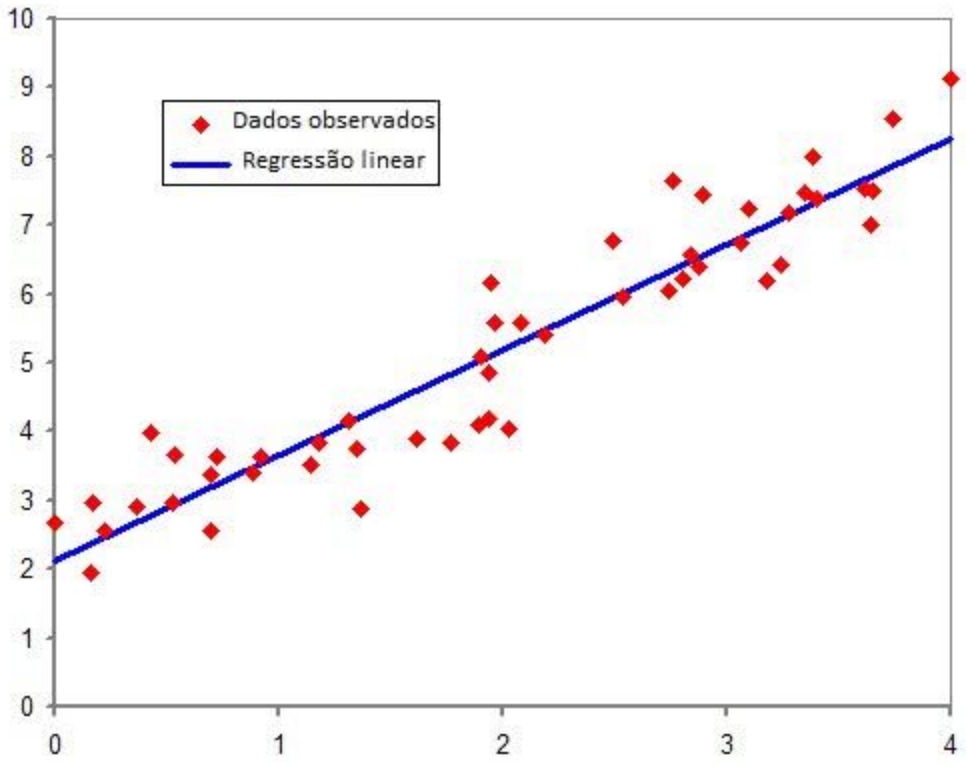


Gráfico 1 - Exemplo de regressão linear

A equação de uma regressão linear simples pode ser definida conforme imagem abaixo:

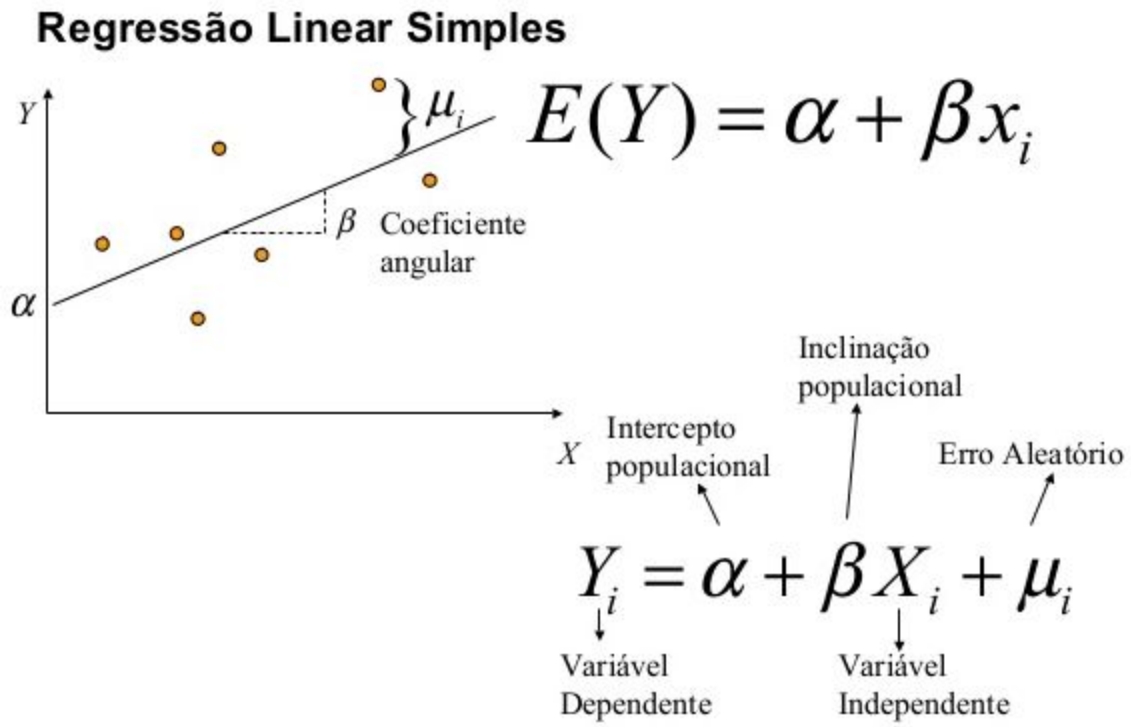


Figura 1 - Regressão Linear e suas variáveis e coeficientes

1.2. Método dos mínimos quadrados

Existem diferentes métodos para encontrar os coeficientes estimadores de uma regressão linear, alguns métodos são determinísticos como o do mínimos quadrados outros métodos são estocásticos como por exemplo o Gradiente Descendente (GD), cada método com seu ponto positivo e negativo.

O método dos mínimos dos quadrados é a forma de estimação mais amplamente utilizada na econometria. Consiste em um estimador que minimiza a soma dos quadrados dos resíduos da regressão, de forma a maximizar o grau de ajuste do modelo aos dados observados.

Ele o método que obtém a melhor resposta para os coeficientes da regressão linear entretanto acaba sendo inviável para problemas muito grandes.

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 α e β que trarão a menor diferença entre a previsão de γ e o γ realmente observado.

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

A minimização se dá ao derivar $S(a,\ b)$ em relação a a e b utilizando a regra da cadeia e então igualar a zero:

$$\frac{\partial S}{\partial a} = \frac{\partial S}{\partial x} * \frac{\partial x}{\partial a}$$
$$\frac{\partial S}{\partial x} = 2\sum_{i=1}^{n}(y_i - a - bx_i)$$
$$\frac{\partial x}{\partial a} = \quad 1$$
$$\frac{\partial S}{\partial a} = -2\sum_{i=1}^{n}(y_i - a - bx_i) = 0$$
$$\frac{\partial S}{\partial b} = -2\sum_{i=1}^{n} x_i\,(y_i - a - bx_i) = 0$$

Distribuindo e dividindo a primeira expressão por $2n$ temos:

$$\frac{-2\sum_{i=1}^{n} y_i}{2n} + \frac{2\sum_{i=1}^{n} a}{2n} + \frac{2\sum_{i=1}^{n} bx_i}{2n} = \frac{0}{2n}$$
$$\frac{-\sum_{i=1}^{n} y_i}{n} + \frac{\sum_{i=1}^{n} a}{n} + \frac{b\sum_{i=1}^{n} x_i}{n} = 0$$
$$-\bar{y} + a + b\bar{x} = 0$$
$$a = \bar{y} - b\bar{x}$$

onde $\bar{y}$ é a média amostral de y e $\bar{x}$ é a média amostral de x.

Substituindo esse resultado na segunda expressão temos:

$$-2\sum_{i=1}^{n} x_i\,(y_i - \bar{y} + b\bar{x} - bx_i) = 0$$
$$\sum_{i=1}^{n}\left[x_i\,(y_i - \bar{y}) + x_i b\,(\bar{x} - x_i)\right] = 0$$
$$\sum_{i=1}^{n} x_i\,(y_i \quad \bar{y}) + b\sum_{i=1}^{n} x_i\,(\bar{x} \quad x_i) = 0$$
$$b = \frac{\sum_{i=1}^{n} x_i\,(y_i - \bar{y})}{\sum_{i=1}^{n} x_i\,(x_i - \bar{x})}$$

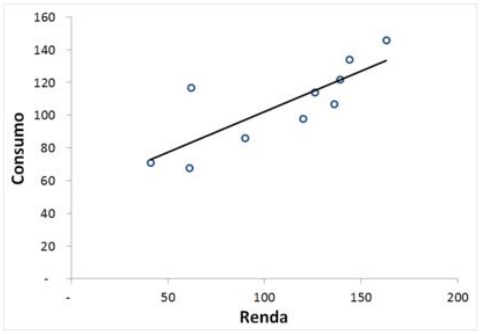
Alguns livros também usam uma fórmula diferente que gera o mesmo resultado:

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

Exemplo de regressão simples

Considere a seguinte base de dados:

i	y Consumo	x Renda
1	122	139
2	114	126
3	86	90
4	134	144
5	146	163
6	107	136
7	68	61
8	117	62
9	71	41
10	98	120



Aplicando as fórmulas acima, chega-se em:

$$b = \frac{7.764,40}{15.671,60} = 0,4954$$

$$a = 106,30 - 0,4954 \times 108,20 = 52,69$$

portanto,

$$Consumo = 52,69 + 0,4954 \times Renda + e$$

Interpretação: Tirando a parte do Consumo que não é influenciada pela Renda, o incremento de $ 1 na Renda causa um incremento esperado de $ 0,4954 no Consumo.

2. Desenvolvimento

Nesta seção serão descritos as implementações dos algoritmos para a resolução do problema proposto, explicado a implementação serial e a paralela utilizando a biblioteca OpenMP.

2.1. Configuração do computador

Tanto as implementações seriais como a paralela foram executadas no supercomputador do IMD, alocando um nó do supercomputador que é composto por 32 cores.

2.2. Algoritmo serial

O código para a geração e cálculo de uma regressão linear usando o método dos mínimos quadrados serial foi implementado utilizando a linguagem de programação C++, conforme código fonte abaixo.

```
/* Para compiular: g++ -g RegresaoLinear.cpp -o RegresaoLinear.exe -lrt -Wall */

#include <stdio.h>
#include <time.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

void imprimir_vetor(double* my_vetor, long tamanho){

  printf("[ ");

        for (long i = 0; i < tamanho; ++i) {
                printf("%f ", my_vetor[i]);
        }

  printf("]\n");
}

void gravar_dataset(FILE* fpDataset, long tamanho, double X[], double y[]){

        fprintf(fpDataset, "seq; x; y\n");

        for (long i = 0; i < tamanho; ++i) {
          fprintf(fpDataset, "%ld; %f; %f\n", i, X[i], y[i]);
        }

}


int main(int argc, char* argv[])
{
  bool debug = true;
        struct timespec beginDataset, endDataset;
  struct timespec seedV;
  double elapsed;

        if (argc != 3) {
                printf("Quantidade argumentos invalido.");
                printf("\nFormato: RegressaoLinear.exe t e");
                printf("\nt: tamanho do dataset.");
                printf("\ne: numero threads.\n");

                return 0;
        }

  /* Variaveis a e b */
  clock_gettime(CLOCK_MONOTONIC, &seedV);
  unsigned int seedB = (unsigned int) seedV.tv_nsec;
  double b = ((double)(rand_r(&seedB) % 400) + 100) / 10;

  clock_gettime(CLOCK_MONOTONIC, &seedV);
  unsigned int seedA = (unsigned int) seedV.tv_nsec;
  double a = ((double)(rand_r(&seedA) % 1500) - 200) / 10.0;

  FILE* fpTempo;
        char filenameTempo[80];
  sprintf(filenameTempo, "%s_TEMPO_%s_THREAD_%s_A_%f_B_%f.csv", argv[0], argv[1], argv[2], a, b);
  fpTempo = fopen(filenameTempo, "a");
        if (fpTempo == NULL) {
                fprintf(stderr, "Can't open output file %s!\n", filenameTempo);
                return 1;
```

```
        }

FILE* fpDataset;
        char filenameDataset[80];
sprintf(filenameDataset, "%s_DATASET_%s_THREAD_%s_A_%f_B_%f.csv", argv[0], argv[1], argv[2], a, b);
fpDataset = fopen(filenameDataset, "a");
        if (fpDataset == NULL) {
                fprintf(stderr, "Can't open output file %s!\n", filenameDataset);
                return 1;
        }

FILE* fpVariaveis;
        char filenameVariaveis[80];
sprintf(filenameVariaveis, "%s_Variaveis_%s_THREAD_%s_A_%f_B_%F.csv", argv[0], argv[1], argv[2], a, b);
fpVariaveis = fopen(filenameVariaveis, "a");
        if (fpDataset == NULL) {
                fprintf(stderr, "Can't open output file %s!\n", filenameVariaveis);
                return 1;
        }

/* Tamanho do dataset */
        long tamanho = atol(argv[1]);

if(tamanho > 50)
  debug = false;

/* Variavel independente */
double* X = new double[tamanho];
        for (long i = 0; i < tamanho; ++i) {
                X[i] = 0.0;
        }

/* Variavel dependente */
double* y = new double[tamanho];
        for (long i = 0; i < tamanho; ++i) {
                y[i] = 0.0;
        }


printf("Valores utilizados: y = a + b * x + E\n");
printf("A: %f <> B: %f\n", a, b);
printf("y = %f + %f * x + E\n", a, b);

/*****************************************/
/*              Inicio                   */
/*****************************************/
clock_gettime(CLOCK_MONOTONIC, &beginDataset);

// 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]);
  }

}

// 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);


// 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);


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


/*****************************************/
/*              FIM                      */
/*****************************************/
clock_gettime(CLOCK_MONOTONIC, &endDataset);
elapsed = endDataset.tv_sec - beginDataset.tv_sec;
elapsed += (endDataset.tv_nsec - beginDataset.tv_nsec) / 1000000000.0;

if(debug){
  printf("X: ");
  imprimir_vetor(X, tamanho);
  printf("y: ");
  imprimir_vetor(y, tamanho);
}

//gravar_dataset(fpDataset, tamanho, X, y);
fclose(fpDataset);
```

```
  printf("\nTempo: %f seg.\n\n", elapsed);
  fprintf(fpTempo, "%f ", elapsed);
  fclose(fpTempo);


  fprintf(fpVariaveis, "Valores utilizado: y = a + b * x + E\n");
  fprintf(fpVariaveis, "A: %f <> B: %f\n", a, b);
  fprintf(fpVariaveis, "Valores obtidos: y = a + b * x + E\n");
  fprintf(fpVariaveis, "A: %f <> B: %f\n", a_estimado, b_estimado);
  fprintf(fpVariaveis, "y = %f + %f * x\n", a_estimado, b_estimado);
  fclose(fpVariaveis);

  return 0;

}
```

Para facilitar a implementação e deixa a implementação serial mais próxima da implementação paralela foram criadas apenas três funções auxiliares mas todo o processo do cálculo do método do mínimo dos quadrado está dentro da função main.

Iniciamos gerado o dataset de forma aleatória a onde inicialmente geramos os coeficientes da regressão linear de forma aleatória e depois aplicamos a variável independente ao modelo o número de vezes passado por argumento para aplicação mais um erro também aleatório.

```
// 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]);
     }

  }
```

Assim, geramos uma distribuição de dados com uma relação linear entre duas variáveis.

O primeiro passo para o cálculo do mínimo dos quadrados é encontrar a média da variável independente, nesse caso X, e a média da variável dependente, nesse caso representado por 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);
```

Fragmento de código responsável por cálcula a média de Y.

```
 // 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);
```

Após obter a média de X e Y, partimos para encontrar o primeiro coeficiente, conforme fórmula abaixo:

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

Abaixo segue o fragmento de código responsável pelo calculo de b:

```
// 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);
```

Com os valores da média de X, Y e o valor do coeficiente b o coeficiente a é obtido a partir da aplicação da fórmula abaixo:

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

A onde o coeficiente a é igual a diferença da média de y menos a multiplicação do coeficiente b vezes a média de x.

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

Abaixo segue um exemplo de execução do programa e a verificação de corretude.

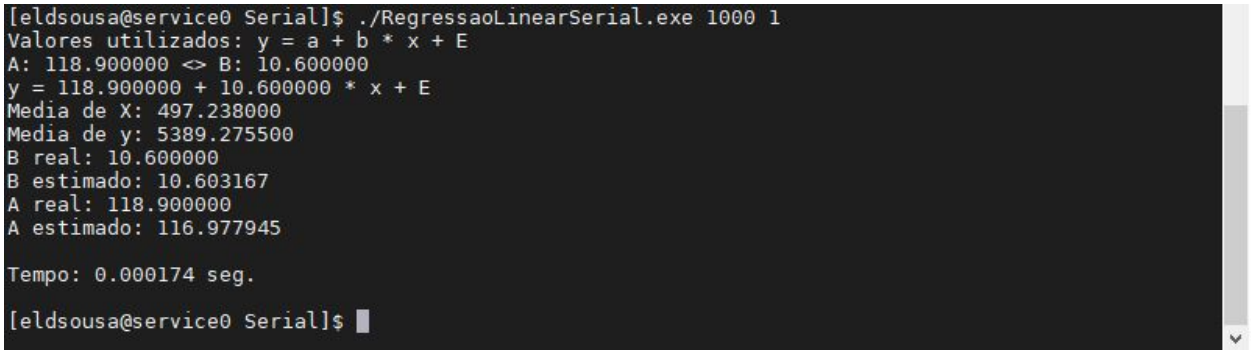


Figura 2 - Verificação de corretude do algoritmo serial

Abaixo segue a comparação dos resultados obtidos com o excel e com algoritmo desenvolvido no trabalho, nesse exemplo foi geradas 200 observações e a relação linear é igual y=27,8x+14,1.

Inicialmente tentei plotar o gráfico com o tamanho utilizado nas análise mas como os valores utilizados nas análises eram muito grande o gráfico acabava ficando muito poluído então foi gerado um dataset com apenas 200 observações e comparado os resultados com a fórmula original, a regressão linear gerada pelo Excel e a regressão gerada pelo algoritmo, conforme podemos observar na imagem acima os resultados obtidos ficaram muito próximo, levando em consideração que na geração dos dados foi incluído um erro aleatório.

2.3. Algoritmo paralelo

A implementação paralela com OpenMP, não apresentou grandes problemas principalmente devido a função “reduction”, que simplificou o tratamento da região crítica e as demais modificações basicamente foram a inclusão região de paralelização dos FOR e tratamento do escopo das variáveis compartilhadas.

```
/* Para compiular: g++ -g RegresaoLinear.cpp -o RegresaoLinear.exe -lrt -fopenmp -Wall */

#include <stdio.h>
#include <time.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <omp.h>

void imprimir_vetor(double* my_vetor, long tamanho){

  printf("[ ");
```

```
		for (long i = 0; i < tamanho; ++i) {
			printf("%f ", my_vetor[i]);
		}

  printf("]\n");
}

void gravar_dataset(FILE* fpDataset, long tamanho, double X[], double y[]){

		fprintf(fpDataset, "seq; x; y\n");

		for (long i = 0; i < tamanho; ++i) {
		  fprintf(fpDataset, "%ld; %f; %f\n", i, X[i], y[i]);
		}

}


int main(int argc, char* argv[])
{
  bool debug = true;
		struct timespec beginDataset, endDataset;
  struct timespec seedV;
  double elapsed;

		if (argc != 3) {
			printf("Quantidade argumentos invalido.");
			printf("\nFormato: RegressaoLinear.exe t e");
			printf("\nt: tamanho do dataset.");
			printf("\ne: numero threads.\n");

			return 0;
		}

  /* Variaveis a e b */
  clock_gettime(CLOCK_MONOTONIC, &seedV);
  unsigned int seedB = (unsigned int) seedV.tv_nsec;
  double b = ((double)(rand_r(&seedB) % 400) + 100) / 10;

  clock_gettime(CLOCK_MONOTONIC, &seedV);
  unsigned int seedA = (unsigned int) seedV.tv_nsec;
  double a = ((double)(rand_r(&seedA) % 1500) - 200) / 10.0;

  FILE* fpTempo;
  char filenameTempo[80];
  sprintf(filenameTempo, "%s_TEMPO_%s_THREAD_%s_A_%f_B_%f.csv", argv[0], argv[1], argv[2], a, b);
  fpTempo = fopen(filenameTempo, "a");
  if (fpTempo == NULL) {
		fprintf(stderr, "Can't open output file %s!\n", filenameTempo);
		return 1;
  }

  FILE* fpDataset;
  char filenameDataset[80];
  sprintf(filenameDataset, "%s_DATASET_%s_THREAD_%s_A_%f_B_%f.csv", argv[0], argv[1], argv[2], a, b);
  fpDataset = fopen(filenameDataset, "a");
  if (fpDataset == NULL) {
		fprintf(stderr, "Can't open output file %s!\n", filenameDataset);
		return 1;
  }

  FILE* fpVariaveis;
  char filenameVariaveis[80];
  sprintf(filenameVariaveis, "%s_Variaveis_%s_THREAD_%s_A_%f_B_%F.csv", argv[0], argv[1], argv[2], a, b);
  fpVariaveis = fopen(filenameVariaveis, "a");
  if (fpDataset == NULL) {
		fprintf(stderr, "Can't open output file %s!\n", filenameVariaveis);
		return 1;
   }
```

```
 /* Tamanho do dataset */
 long tamanho = atol(argv[1]);

 /* Numero de Threads */
int num_threads = atoi(argv[2]);


 if(tamanho > 50)
    debug = false;

 /* Variavel independente */
 double* X = new double[tamanho];
 for (long i = 0; i < tamanho; ++i) {
            X[i] = 0.0;
 }

 /* Variavel dependente */
 double* y = new double[tamanho];
 for (long i = 0; i < tamanho; ++i) {
            y[i] = 0.0;
 }


 printf("Valores utilizados: y = a + b * x + E\n");
 printf("A: %f <> B: %f\n", a, b);
 printf("y = %f + %f * x + E\n", a, b);

 /*****************************************/
 /*              Inicio                   */
 /*****************************************/
 clock_gettime(CLOCK_MONOTONIC, &beginDataset);

 // 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]);
   }

 }
```

```
  // 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);
  }

  // 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);

        //#pragma omp parallel shared(X, media_X, i) default(none) reduction(+:divisor) num_threads(num_threads)
        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);

  }
  /*****************************************/
  /*              FIM                      */
  /*****************************************/
  clock_gettime(CLOCK_MONOTONIC, &endDataset);
  elapsed = endDataset.tv_sec - beginDataset.tv_sec;
  elapsed += (endDataset.tv_nsec - beginDataset.tv_nsec) / 1000000000.0;

  if(debug){
    printf("X: ");
    imprimir_vetor(X, tamanho);
    printf("y: ");
    imprimir_vetor(y, tamanho);
  }

  //gravar_dataset(fpDataset, tamanho, X, y);
  fclose(fpDataset);

  printf("\nTempo: %f seg.\n\n", elapsed);
```

```
  fprintf(fpTempo, "%f ", elapsed);
  fclose(fpTempo);


  fprintf(fpVariaveis, "Valores utilizado: y = a + b * x + E\n");
  fprintf(fpVariaveis, "A: %f <> B: %f\n", a, b);
  fprintf(fpVariaveis, "Valores obtidos: y = a + b * x + E\n");
  fprintf(fpVariaveis, "A: %f <> B: %f\n", a_estimado, b_estimado);
  fprintf(fpVariaveis, "y = %f + %f * x\n", a_estimado, b_estimado);
              fclose(fpVariaveis);

  return 0;

}
```

A estratégia utilizada para a paralelização do algoritmo foi a divisão do problema pelo número de repetições do loop, utilizando um FOR com chunksize de tamanho 1.

```
  // 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]);
     }

  }
```

O trecho de código onde temos a região crítica é o cálculo da média de X e Y, porém com a utilização da função reduction da biblioteca OpenMP o tratamento da região crítica foi muito simples deixando toda a complexidade do controle de acesso a variável compartilhada entre as threads para a biblioteca.

```
// 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);
  }
```

No cálculo dos coeficientes a e b, também foi utilizado a função reduction para o tratamento da região.

```
// 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);

  }
```

Abaixo segue exemplo de execução do programa e a verificação de corretude.

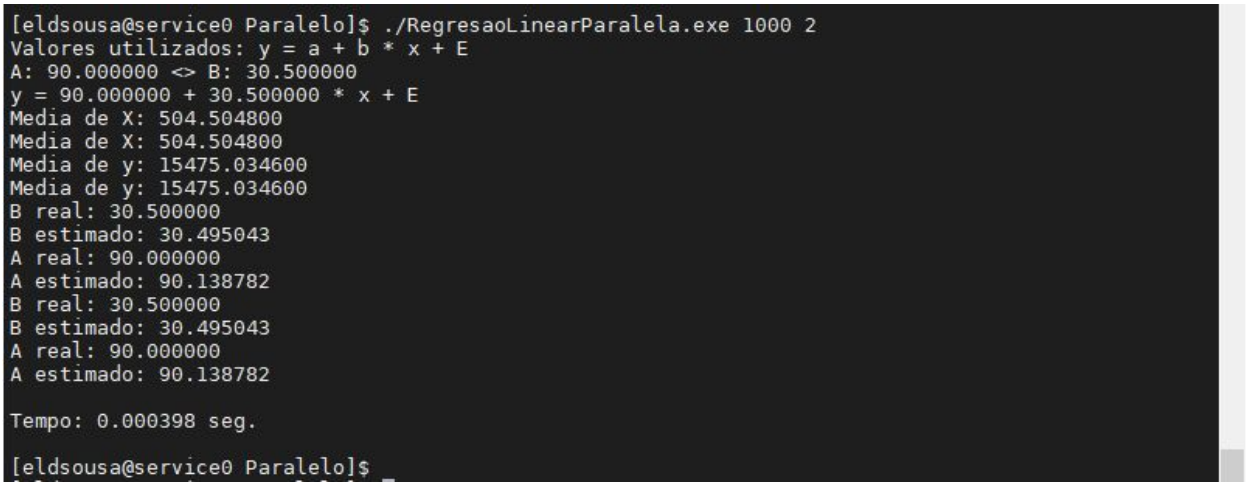


Figura 3 - Verificação de corretude do algoritmo paralelo

2.5. Resultados

Nesta seção iremos analisar os resultados obtidos da execução da implementação serial e a paralela sendo executada com 4, 8, 16 e 32 thread, com a geração de observações com tamanho de 300.000.000, 500.000.000, 700.000.000 e 900.000.000.

Abaixo temos as tabelas com os tempos de execução para as diferentes implementações.

Algoritmo de regressão linear - Serial				
Execução	300 M	500 M	700 M	900 M
1	42,812	71,354	99,918	128,450
2	42,813	71,359	96,294	128,450
3	42,815	71,345	99,913	128,444
4	41,381	71,356	99,905	128,456
5	42,812	71,349	99,900	128,447
6	42,906	68,839	99,916	128,448
7	42,815	71,354	99,910	128,466
8	42,819	71,362	99,882	128,441

9	42,814	71,406	100,036	128,436
10	42,952	71,354	99,899	128,451
Média	42,694	71,108	99,557	128,449

Tabela 1 - Tempo de execução do algoritmo Serial

	Algoritmo de regressão linear - Paralelo				
4 Thread	Execução	300 M	500 M	700 M	900 M
	1	16,047	26,725	37,402	48,177
	2	16,050	26,868	37,625	48,526
	3	16,342	26,851	37,727	48,133
	4	16,059	26,718	37,637	48,169
	5	16,118	26,882	37,453	48,148
	6	16,139	26,889	37,658	48,448
	7	16,065	26,722	37,547	48,396
	8	16,048	26,888	37,390	48,313
	9	16,041	26,900	37,389	48,340
	10	16,043	26,751	37,419	48,177
	Média	16,095	26,819	37,525	48,283
	SpeedUp	2,652560537	2,651356932	2,653105191	2,660355539
	Eficiência	0,663140134	0,662839233	0,663276298	0,665088885
8 Thread	Execução	300 M	500 M	700 M	900 M
	1	10,636	17,673	25,094	31,994
	2	10,646	17,782	24,758	31,928
	3	10,645	17,863	24,826	31,743

	4	10,665	17,719	24,947	31,947
	5	10,629	17,659	24,860	31,979
	6	10,622	17,773	24,943	31,779
	7	10,653	17,880	24,876	32,027
	8	10,618	17,889	24,961	31,961
	9	10,687	17,858	24,968	32,086
	10	10,681	17,665	24,858	31,799
	Média	10,648	17,776	24,909	31,924
	SpeedUp	4,009471417	4,000177851	3,996847244	4,023542439
	Eficiência	0,501183927	0,500022231	0,499605905	0,502942805
16 Thread	Execução	300 M	500 M	700 M	900 M
	1	7,901	13,306	18,459	23,754
	2	7,950	13,144	18,507	23,967
	3	7,905	13,217	18,490	23,764
	4	7,891	13,216	18,439	23,844
	5	7,916	13,272	18,489	23,685
	6	7,905	13,437	18,497	23,819
	7	7,893	13,239	18,689	23,744
	8	7,942	13,200	18,414	23,972
	9	7,937	13,183	18,555	23,799
	10	7,892	13,218	18,427	23,721
	Média	7,913	13,243	18,497	23,807
	SpeedUp	5,395227849	5,369333364	5,382464014	5,395456606
	Eficiência	0,337201741	0,335583335	0,336404001	0,337216038

32 Thread	Execução	300 M	500 M	700 M	900 M
	1	7,086	15,646	21,956	28,190
	2	7,293	12,114	22,113	22,273
	3	7,317	12,589	22,193	29,061
	4	8,197	12,148	22,468	28,166
	5	7,268	12,300	21,995	28,496
	6	7,678	15,848	16,909	27,547
	7	7,339	16,213	21,933	28,773
	8	7,970	15,573	22,425	27,928
	9	7,524	15,775	21,976	28,478
	10	7,970	12,170	21,214	28,699
	Média	7,564	14,037	21,518	27,761
	SpeedUp	5,644281638	5,06560139	4,626666917	4,626936997
	Eficiência	0,176383801	0,158300043	0,144583341	0,144591781

Tabela 2 - Tempo de execução do algoritmo paralelo com diferente números de thread

Abaixo temos o gráfico comparativo com os valores das tabelas acima.

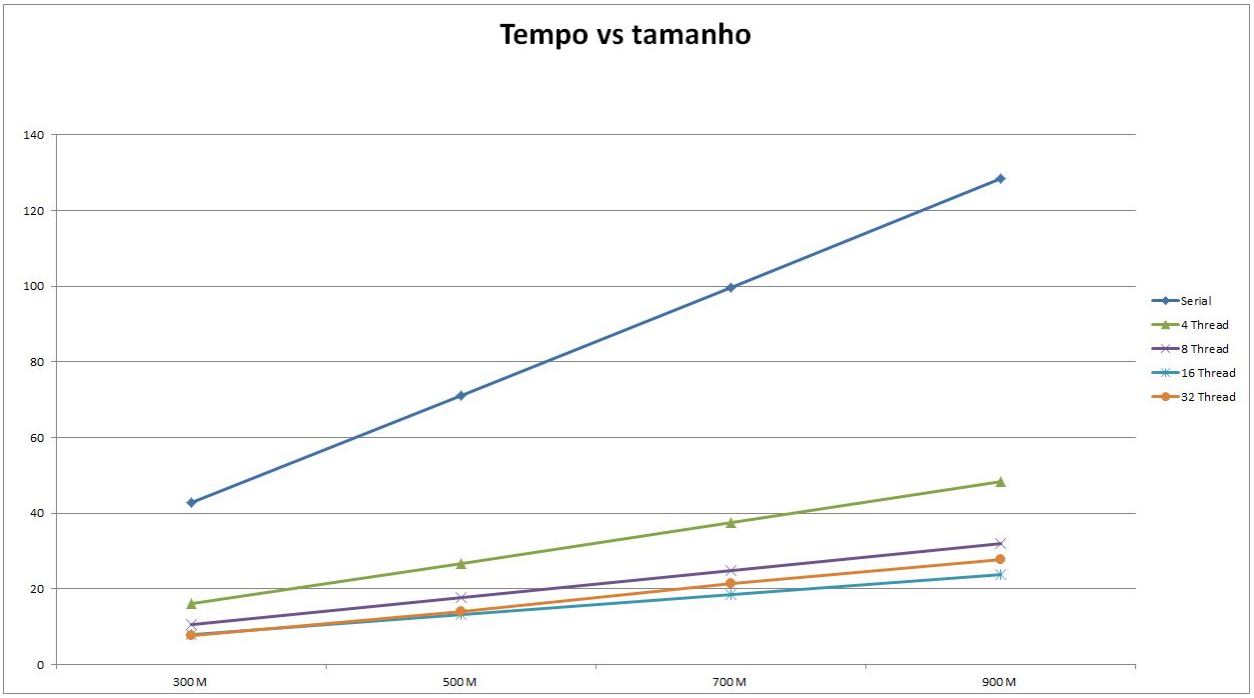


Gráfico 2 - Comparação entre os tempos de execução entre as diferentes implementações

Conforme pode ser analisado no gráfico 2, a implementação serial teve um aumento do tempo de execução muito expressivo à medida que o tamanho do problema aumenta.

A implementação com OpenMP também teve um aumento no tempo execução, como se era esperado mas bem suave em comparação a implementação serial.

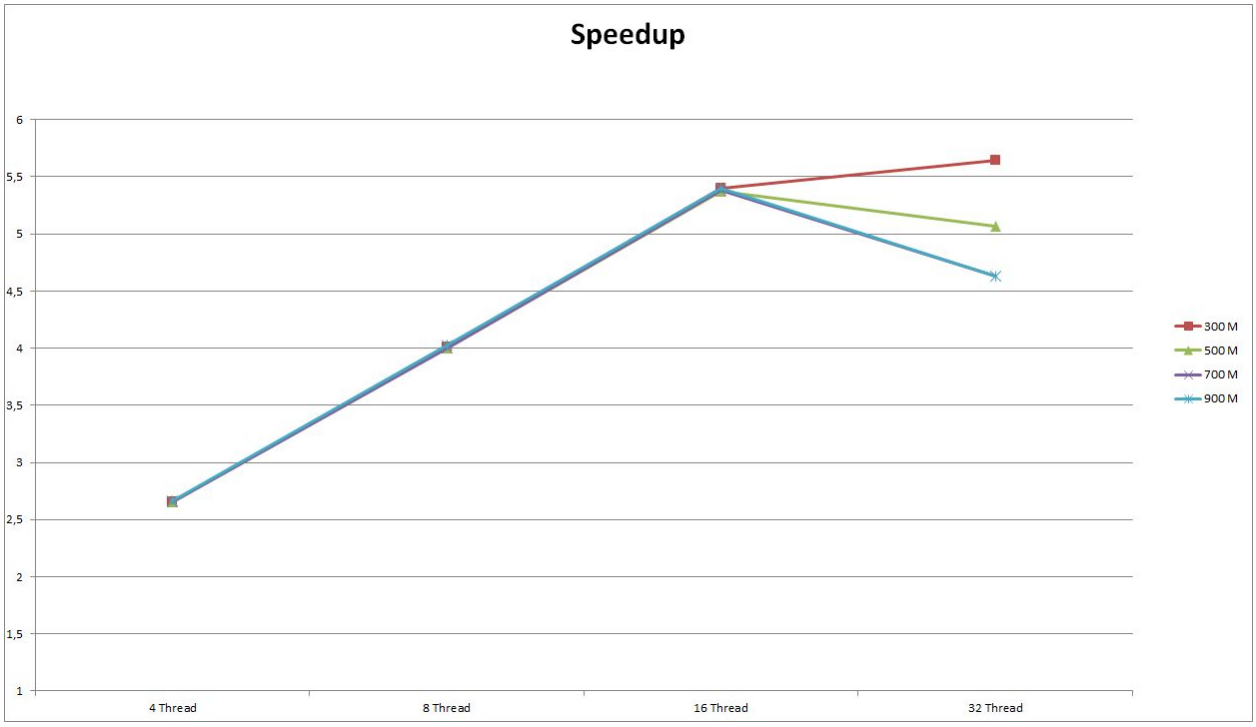


Gráfico 2 - Speedup

Conforme o gráfico de speedup a implementação paralela apresenta um ganho progressivo praticamente quase constante à medida que aumenta o número de threads dentro do intervalo de estudo definido para o trabalho, apresentando apenas uma queda no speedup com 32 threads.

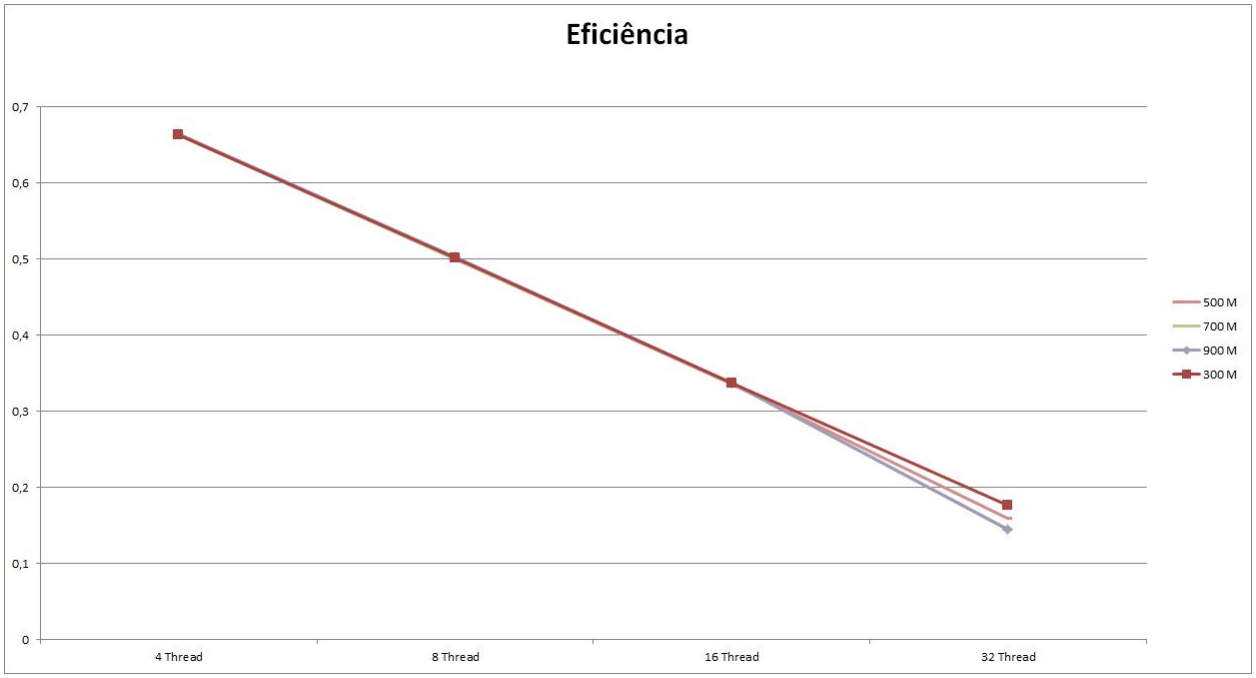


Gráfico 3 - Eficiência vs números de thread

Conforme podemos analisar o gráfico de eficiência acima, a implementação paralela para o algoritmo de regressão linear tem sua eficiência diminuída à medida que aumenta o número de thread.

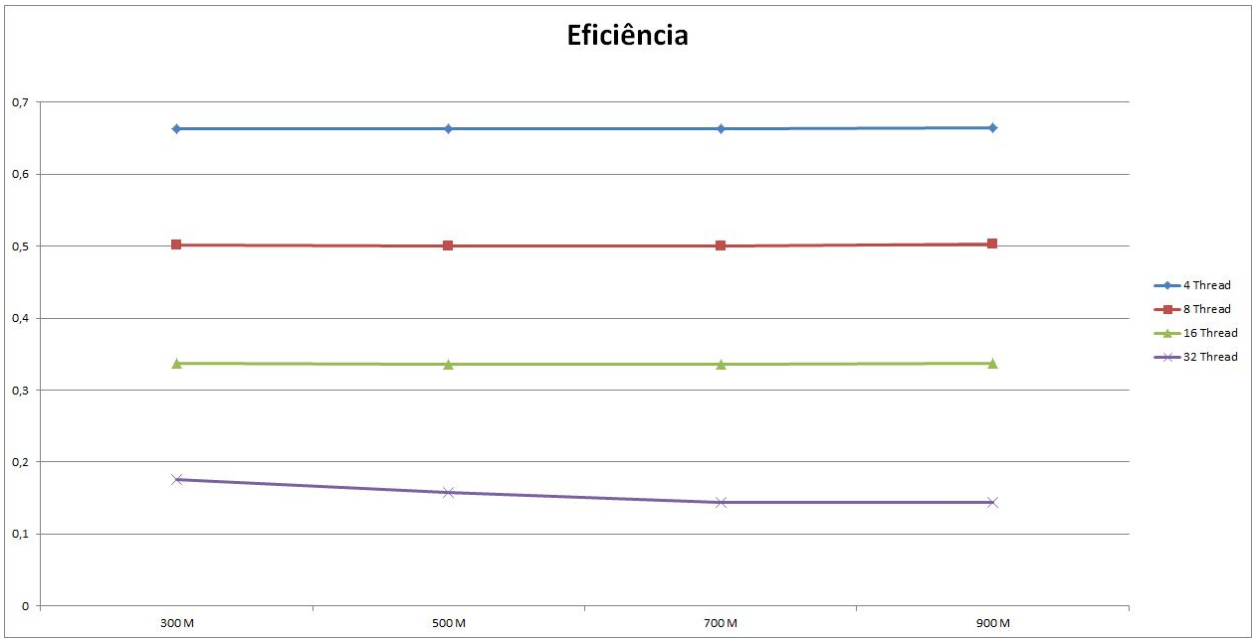


Gráfico 4 - Eficiência vs Tamanho do problema

No gráfico acima podemos observar uma eficiência constante em praticamente todas as implementações exceto na execução com 32 thread como já era esperado devido ao comportamento anormal nos gráficos anteriores.

Com base na análise desses gráficos podemos concluir que a implementação paralela é escalável já que à medida que aumentamos o tamanho do problema a eficiência se mantém constante em praticamente todos os testes e mesmo nas execuções com 32 thread a queda é suave de apenas 0,02.

3. Vídeo da apresentação

O vídeo está publicado no youtubo, link de acesso: https://youtu.be/GMMT4U32nhw

4. Considerações Finais

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.