UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO
TÓPICOS ESPECIAIS EM INTERNET DAS COISAS “B”
COMPUTAÇÃO PARALELA - (IMD0291)
PROF. KAYO GONCALVES E SILVA
2020.6

Relatório

DAWERTON EDUARDO CARLOS VAZ

Aplicação de computação paralela utilizando Pthreads

SUMÁRIO

Introdução

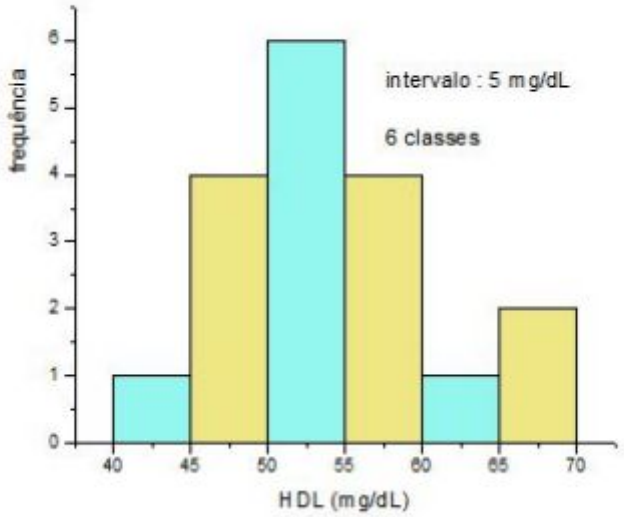
Neste relatório aplicamos o método Pthread, Para multiplicar duas matrizes quadradas

Histograma

O que é um Histograma

O histograma é a representação gráfica em colunas (ou em barras) de um conjunto de dados previamente tabulado. A base (eixo x) de cada retângulo representa uma classe de dados. Este conjunto pode ser um dado específico, um número, intervalos de números ou qualquer informação que se deseja classificar. A altura (eixo y) de cada retângulo representa a quantidade ou a frequência absoluta com que o valor da classe ocorre no conjunto daqueles dados

A Figura a seguir mostra um exemplo de um histograma do nível de HDL em mg/dL e a frequência que ocorre em um dado conjunto de dados. O gráfico é composto por 6 intervalos (a citar, 40-45, 45-50, 50-55, 55-60, 60-65 e 65-70) e suas frequências de ocorrência estão dispostas de tal forma que é possível perceber que existe uma maior frequência no entorno de 50-55 mg/dL. I



Implementação do Código

Serial

A seguir implementamos o código para gerar ponto usando uma distribuição normal e depois verificando onde cada ponto deve ficar, e logo depois escrevendo em um arquivo o vetor de pontos, como explicado nos comentários feitos nele

```
// normal_distribution
#include <iostream>
#include <string>
#include <random>
#include <fstream>
#include <sys/time.h>
#include <time.h>

long double nrolls;
long double vnumber;
int pnumber;
long double divnumber;

int main(int argc, char const *argv[])
{
 struct timeval start, stop; //variaveis para guardar o temp  o
 gettimeofday(&start, 0);    //Começa a marcar o tempo
 nrolls = atoi(argv[1]);     //recebe numero de pontos
 vnumber = 10000000;         //atoi(argv[2]);//tamanho total das
unidades
 std::default_random_engine generator;
 std::normal_distribution<long double> distribution(0.0, (vnumber /
6));

 pnumber = 10;                  //tamanho do vetor
 divnumber = vnumber / pnumber; //dividindo total das unidades pelo
vetor
 long double p[pnumber] = {};   //criando vetor

 for (int j = 0; j < nrolls; j++) //começando laço para gerar os pontos
 {

   double number = distribution(generator); //gerando número
```

```
   for (int i = 0; i < pnumber; i++) //laço para saber o lugar a qual o
ponto pertence
   {
     if (number >= divnumber * (i) && number < divnumber * (i + 1))
//verificando onde o ponto deve ficar
     {
       ++p[i];
     }
   }
 }

 std::ofstream out("Histograma.txt"); /*IMPRIMINDO RESULTADO PARA
PLOTAR GRAFICO*/
 out << "# Histograma:" << std::endl;
 for (int i = 0; i < pnumber; ++i)
 {
   out << "[" << i << "-" << (i + 1) << "]:\t";
   out << p[i] << std::endl;
 }
 gettimeofday(&stop, 0); //finaliza a contagem do tempo

 return 0;
}
```

Em Seguida temos o código para adicionar em um arquivo o tempo de execução.

O código guarda o início da marcação de tempo na variável start e depois de rodar a função marca o final da execução na variável stop, depois criar uma arquivo e escreve a diferença do start e stop, marcando assim o tempo de de execução da função.

```
 std::ofstream out2;
 out2.open("tempo_de_mm.txt", std::ios::app); /*IMPRIMINDO TEMPO*/
 out2 << "\t" << (double)(stop.tv_usec - start.tv_usec) / 1000000 +
(double)(stop.tv_sec - start.tv_sec);
   fprintf(fp, "\t%f ", (double)(stop.tv_usec - start.tv_usec) /
1000000 + (double)(stop.tv_sec - start.tv_sec));//imprimindo tempo no
arquivo tempo_de_mm.txt
   fclose(fp);

   return 0;
}close(fp);
```

```
    return 0;
}
```

Paralelo

A seguir implementamos a função para gerar o histograma em forma paralela na linguagem c++.A função distribuir é chamada em forma paralela usando Pthread passando a quantidade de threads determinada, logo depois é dada para cada thread qual a quantidade de pontos que eles devem gerar, que em seguida distribuem cada ponto ao determinado lugar

```
// normal_distribution
#include <iostream>
#include <string>
#include <random>
#include <fstream>
#include <sys/time.h>
#include <time.h>
#include <pthread.h>

long double nrolls;
long double vnumber;
int pnumber = 10;
long double divnumber;
int thread_count;
long double p[10] = {};//criando vetor

void *distribuir(void *rank);
int main(int argc, char const *argv[])
{
 struct timeval start, stop; //variaveis para guardar o temp  o
 gettimeofday(&start, 0);//Começa a marcar o tempo
 nrolls = atoi(argv[2]);//recebe numero de pontos
```

```
 vnumber = 1000000; //atoi(argv[2]);//tamanho total das unidades

 pnumber = 10;//tamanho do vetor
 divnumber = vnumber / pnumber;//dividindo total das unidades pelo
vetor

 thread_count = strtol(argv[1], NULL, 10); //variavel que recebe o
numero de threads

 long thread;
 pthread_t *thread_handles = new pthread_t[thread_count];

 //thread_handles = malloc(thread_count * sizeof(pthread_t));

 for (thread = 0; thread < thread_count; thread++) //iniciando a função
distribuir para cada thread
 {
   pthread_create(&thread_handles[thread], NULL, distribuir, (void
*)thread);
 }

 for (thread = 0; thread < thread_count; thread++) //esperando cada
thread finalizar o processamento
 {
   pthread_join(thread_handles[thread], NULL);
 }

 gettimeofday(&stop, 0); //finaliza a contagem do tempo

 free(thread_handles);

 std::ofstream out("Histograma.txt");/*IMPRIMINDO RESULTADO PARA PLOTAR
GRAFICO*/
 out << "# Histograma:" << std::endl;
 for (int i = 0; i < pnumber; ++i)
 {
   out << "[" << i << "-" << (i + 1) << "]:\t";
   out << p[i] << std::endl;
 }
 gettimeofday(&stop, 0); //finaliza a contagem do tempo
 std::ofstream out2;
 out2.open("tempo_de_mm.txt", std::ios::app);/*IMPRIMINDO TEMPO*/
```

```
 out2 << "\t" << (double)(stop.tv_usec - start.tv_usec) / 1000000 +
(double)(stop.tv_sec - start.tv_sec);
 return 0;
}
void *distribuir(void *rank)//Função executada pelas threads
{
 std::default_random_engine generator;
 std::normal_distribution<long double> distribution(0.0, (vnumber /
6));
 long my_rank = (long)rank;//variavel para indetificação de cada thread
 int local_rolls = nrolls / thread_count;//numero de pontos que cada
thread deve criar
 int my_first = my_rank * local_rolls;//primeiro numero da cada thread
 int my_last = (my_rank + 1) * local_rolls - 1;//ultimo numero da cada
thread
 for (int j=my_first; j <= my_last; ++j)//iniciando laça para gerar
todos os pontos
 {
   double number = distribution(generator);//gerando os pontos

   for (int i = 0; i < pnumber; i++)//laço para verificar onde ficara
cada ponto
   {
     if (number >= divnumber * (i) && number < divnumber * (i +
1))//adicionando os pontos ao vetor
     {
       ++p[i];
     }
   }
 }
 return NULL;
}
```

Execução dos códigos

Código serial

Para os testes do código serial foi utilizado o seguinte código em Shell Script. usado no supercomputador da ufrn

```
#!/bin/bash

#SBATCH --partition=full
#SBATCH --job-name=decvaz
#SBATCH --output=decvazOutput.out
#SBATCH --error=decvazError.err
#SBATCH --time=0-00:40
#SBATCH --hint=compute_bound

#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=32

#No Supercomputador, 1 nó = 32 Cores (ou CPUs)
#Lembrar que: TASK = PROCESSO

#A configuração acima reserva 1 nó inteiro (32 cpus no mesmo
processador), vai rodar apenas 1 processo por vez,
# mas este processo terá acesso a 32 cores

#Loop principal de execuções. São 10 tentativas

   tentativas=10 #Quantas vezes o código será executado

   for cores in 1 #números de cores utilizados
   do
               for size in 500000000 #tamanho do problema
          do
               echo -e "\n$cores\t$size\t\t\c" >> "tempo_de_mm.txt"

               for tentativa in $(seq $tentativas) #Cria uma vetor de 1
a "tentativas"
               do
                   ./teste $size
               done
                   #gnuplot -e "filename='graphics/image-$cores-$size'"
"gnuplot"
                   mv Histograma.txt
histogram/"histograma-$cores-$size.txt"
```

```
        done

done

exit
```

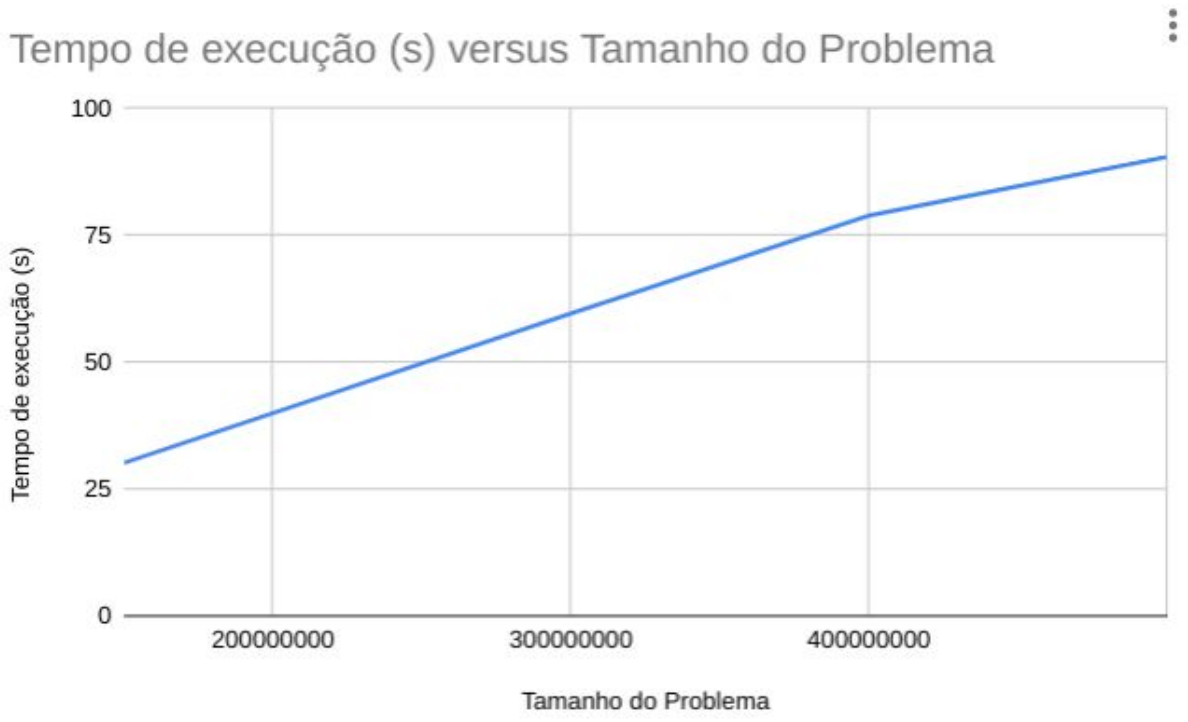
O Shell Script executa o código serial 10 vezes, com 4 tamanhos do problema diferentes, e ao final de cada execução escreve o tamanho do problema que junto com o próprio código serial que escreve o tempo que levou para executar logo depois.

Testes

Ao executar o códigos 10 vezes em cada tamanho do problema fiz uma média como mostra a tabela a seguir

Tamanho do Problema	Tempo de execução (s)
150000000	30,16461
300000000	59,59265
400000000	78,89746
500000000	90,46122

No gráfico a seguir observa-se que a cada vez que o tamanho do problema aumentar, o tempo para execução ficar maior.



Código Paralelo

Para os testes do código paralelo foi utilizado o seguinte código em Shell Script.usado no supercomputador da ufrn

```
#!/bin/bash

#SBATCH --partition=full
#SBATCH --job-name=decvaz
#SBATCH --output=decvazOutput.out
#SBATCH --error=decvazError.err
#SBATCH --time=0-02:00
#SBATCH --hint=compute_bound
module load compilers/gnu/8.3
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --mem=64000
#SBATCH --cpus-per-task=32

#No Supercomputador, 1 nó = 32 Cores (ou CPUs)
#Lembrar que: TASK = PROCESSO
```

```
#A configuração acima reserva 1 nó inteiro (32 cpus no mesmo
processador), vai rodar apenas 1 processo por vez,
# mas este processo terá acesso a 32 cores

#Loop principal de execuções. São 10 tentativas

   tentativas=10 #Quantas vezes o código será executado

   for cores in 16 32 #números de cores utilizados
   do
               for size in 150000000 300000000 400000000 500000000
#tamanho do problema
           do
               echo -e "\n$cores\t$size\t\t\c" >> "tempo_de_mm.txt"

               for tentativa in $(seq $tentativas) #Cria uma vetor de 1
a "tentativas"
               do
                   ./teste $cores $size
               done
                   #gnuplot -e "filename='graphics/image-$cores-$size'"
"gnuplot"
                   mv Histograma.txt
histogram/"histograma-$cores-$size.txt"
           done

   done

   exit
```

O Shell Script executa o código paralelo 10 vezes, com 4 tamanhos do problema diferentes, e ao final de cada execução escreve o tamanho do problema que junto com o próprio código paralelo que escreve o tempo que levou para executar logo depois.

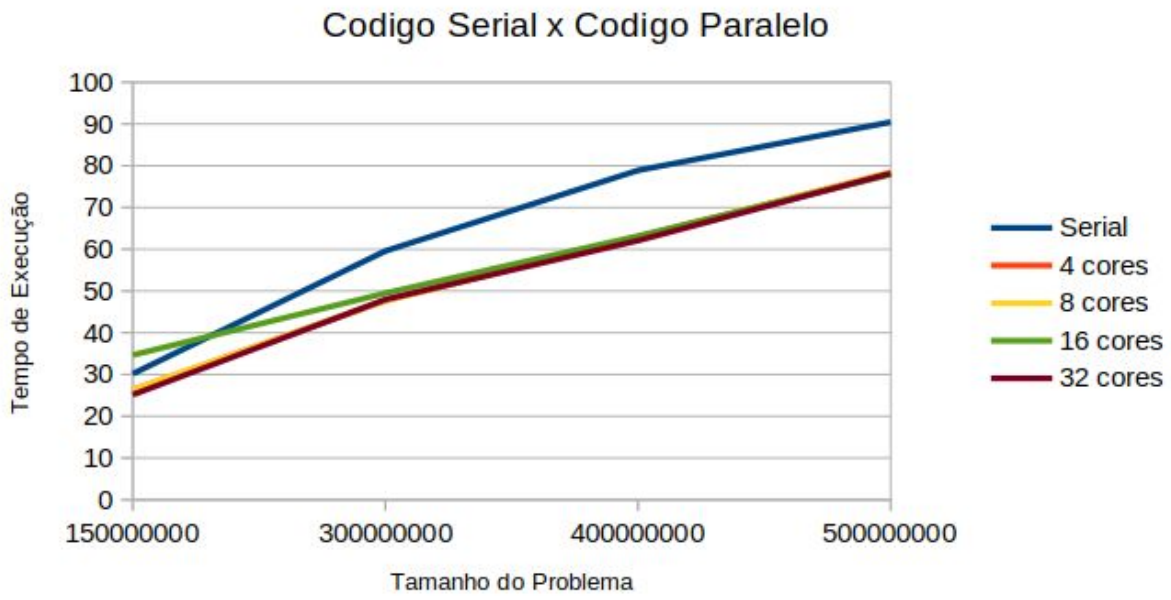
Testes

Ao executar o códigos 10 vezes em cada tamanho do problema fiz uma média como mostra a tabela a seguir

Cores	Tamanho do Problema	Tempo de execução (s)
Serial	150000000	30,16461
Serial	300000000	59,59265
Serial	400000000	78,89746
Serial	500000000	90,46122
4	150000000	26,28475
4	300000000	47,70739
4	400000000	62,89554
4	500000000	78,36108
8	150000000	26,37265
8	300000000	47,66563
8	400000000	62,83475
8	500000000	78,43676
16	150000000	34,65619
16	300000000	49,46786
16	400000000	63,16153
16	500000000	77,90053
32	150000000	25,14757
32	300000000	47,96119
32	400000000	62,06914
32	500000000	78,19235

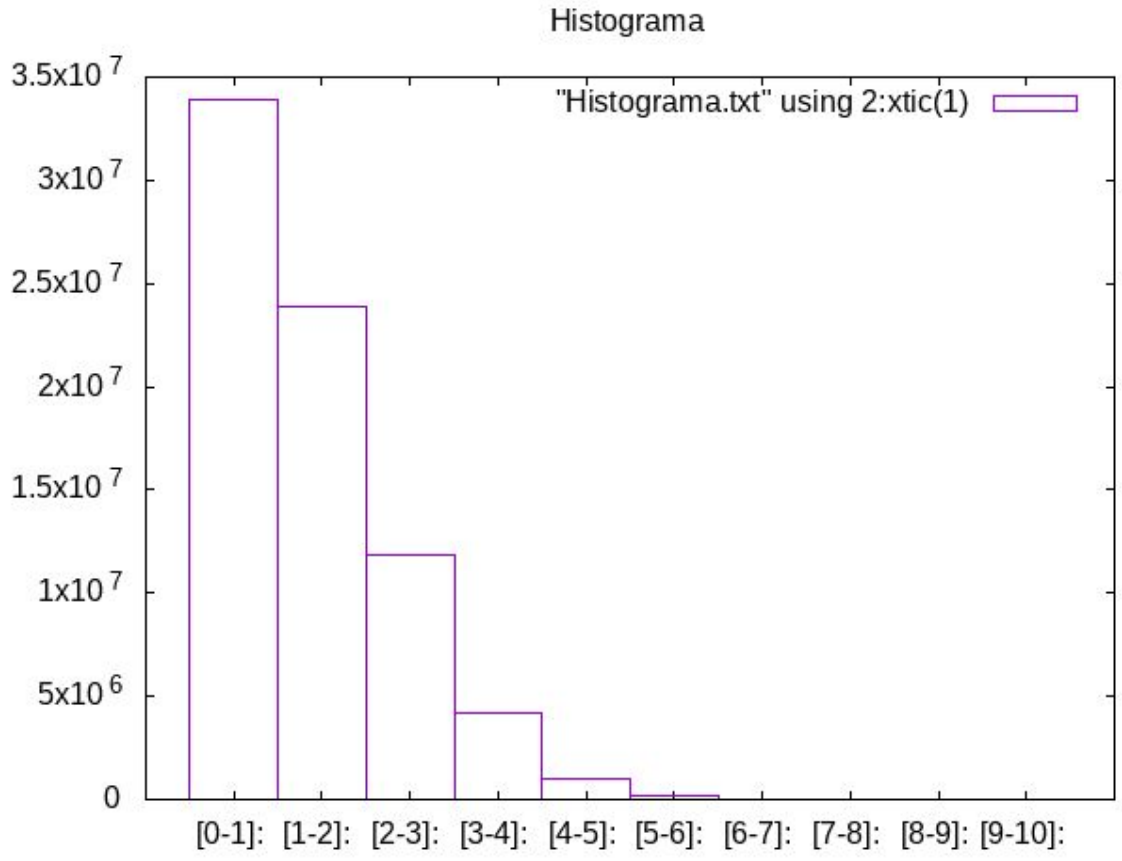
No gráfico a seguir está a comparação do código serial com o paralelo, da pra observar que usando o codigo paralelo teve um bom desempenho mas não teve muita diferença de usar 4 cores ou 32 cores, mas pelo menos o desempenho aumentou consideravelmente usando

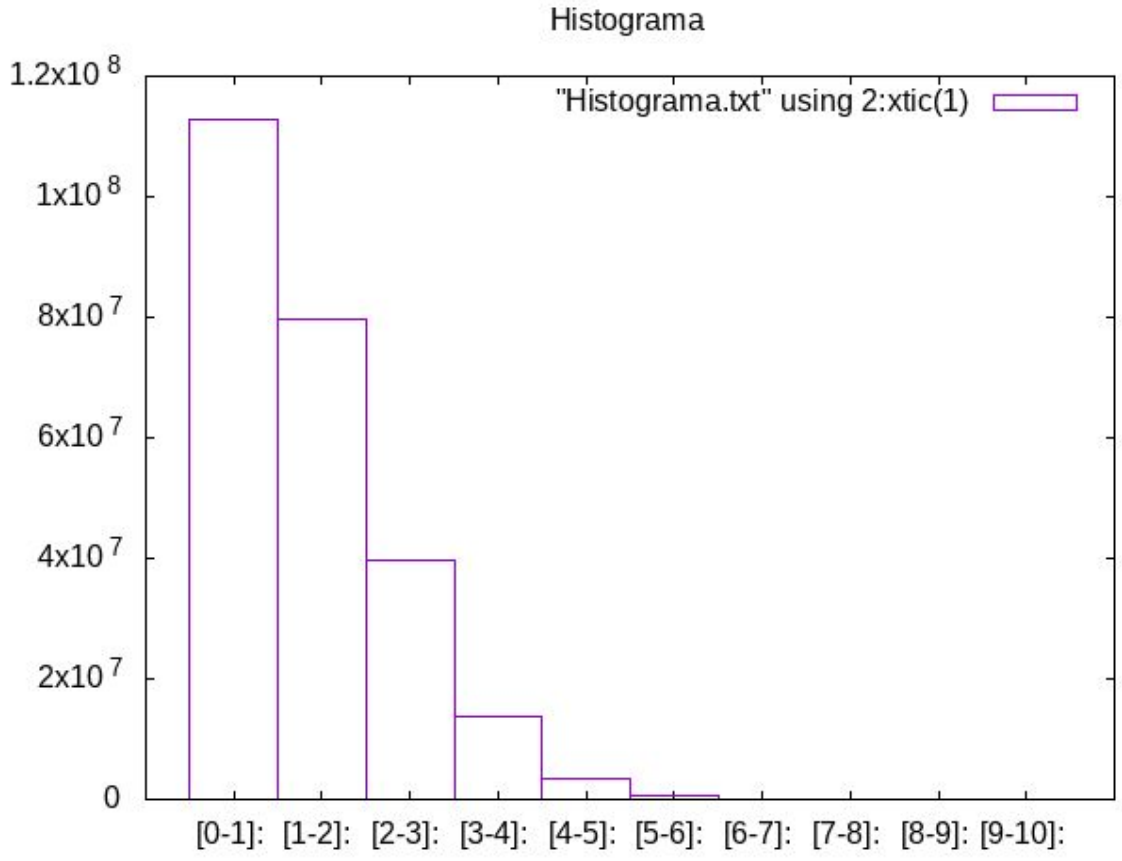
o código paralelo.



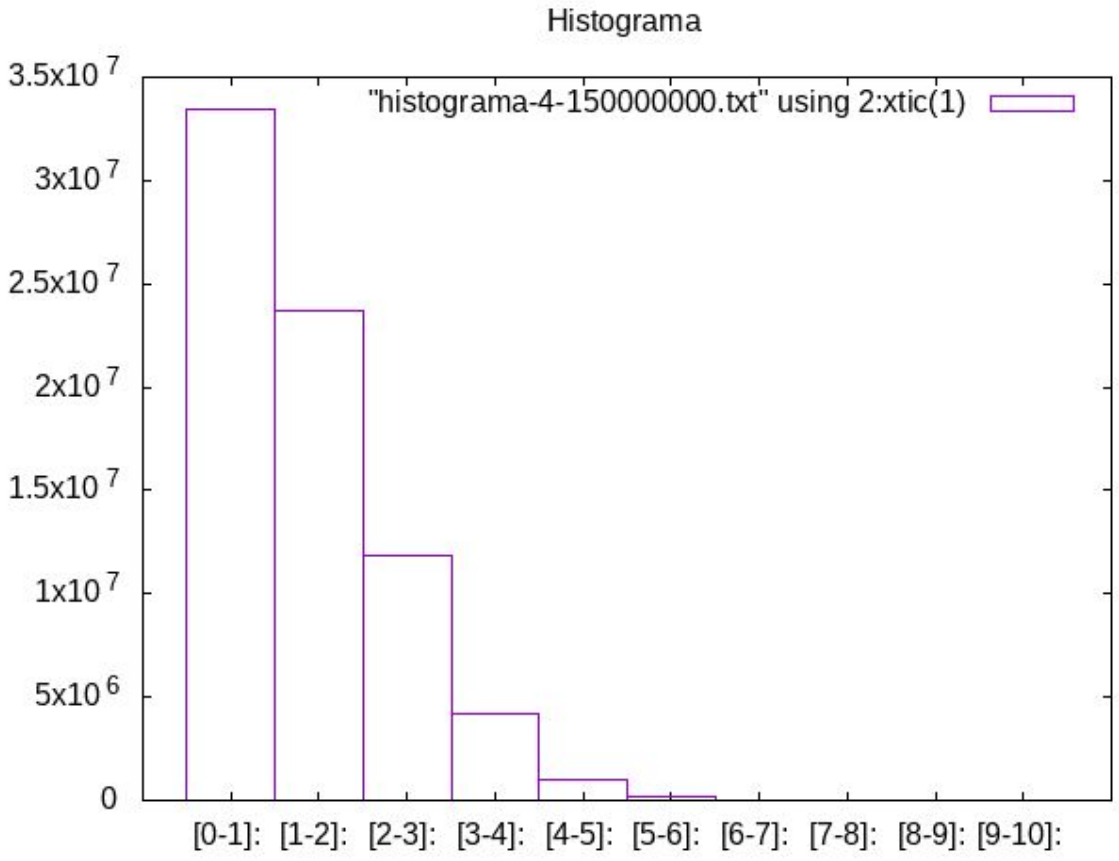
A seguir os gráficos gerados do maior e menor tamanho do problema em forma serial, com 4 cores,8 cores,16 cores,32 cores respectivamente

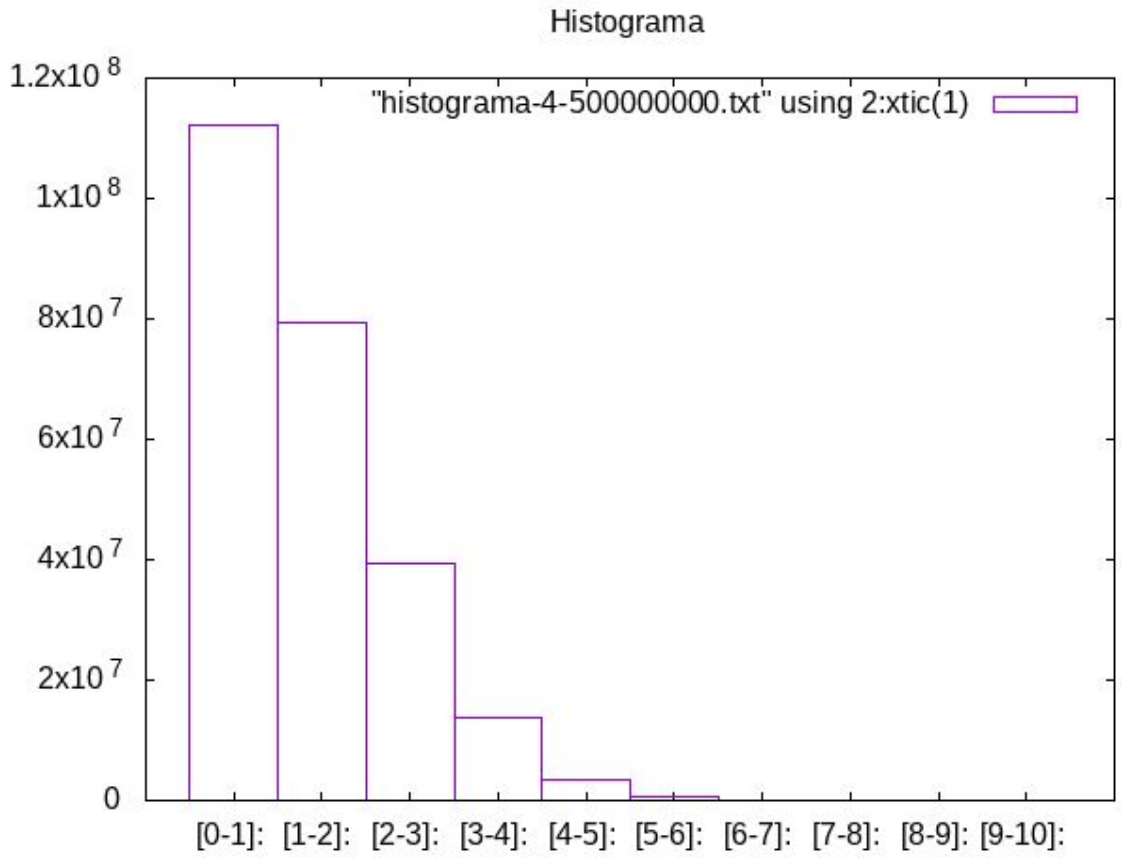
Serial


Histograma
"Histograma.txt" using 2:xtic(1)
3.5x10 7
3x10 7
2.5x10 7
2x10 7
1.5x10 7
1x10 7
5x10 6
0
[0-1]:
[1-2]:
[2-3]:
[3-4]:
[4-5]:
[5-6]:
[6-7]:
[7-8]:
[8-9]:
[9-10]:

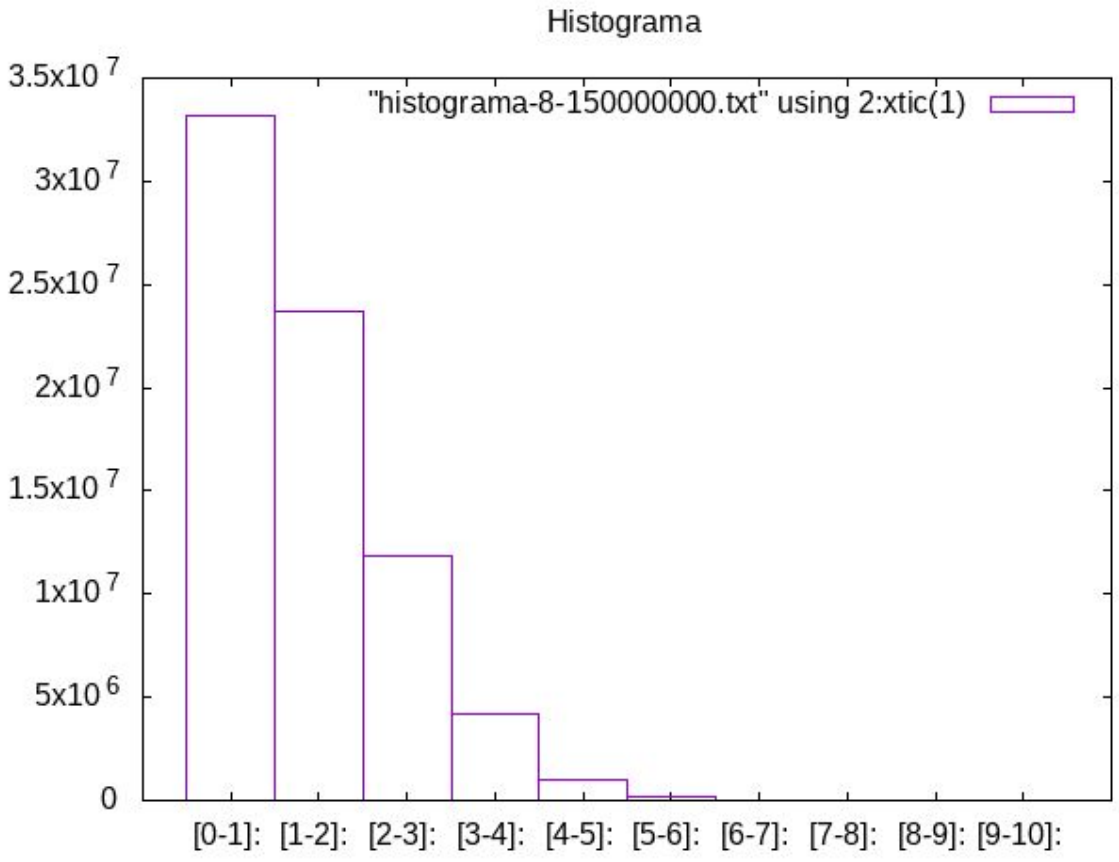

Histograma
"Histograma.txt" using 2:xtic(1)
1.2x10 8
1x10 8
8x10 7
6x10 7
4x10 7
2x10 7
0
[0-1]:
[1-2]:
[2-3]:
[3-4]:
[4-5]:
[5-6]:
[6-7]:
[7-8]:
[8-9]:
[9-10]:

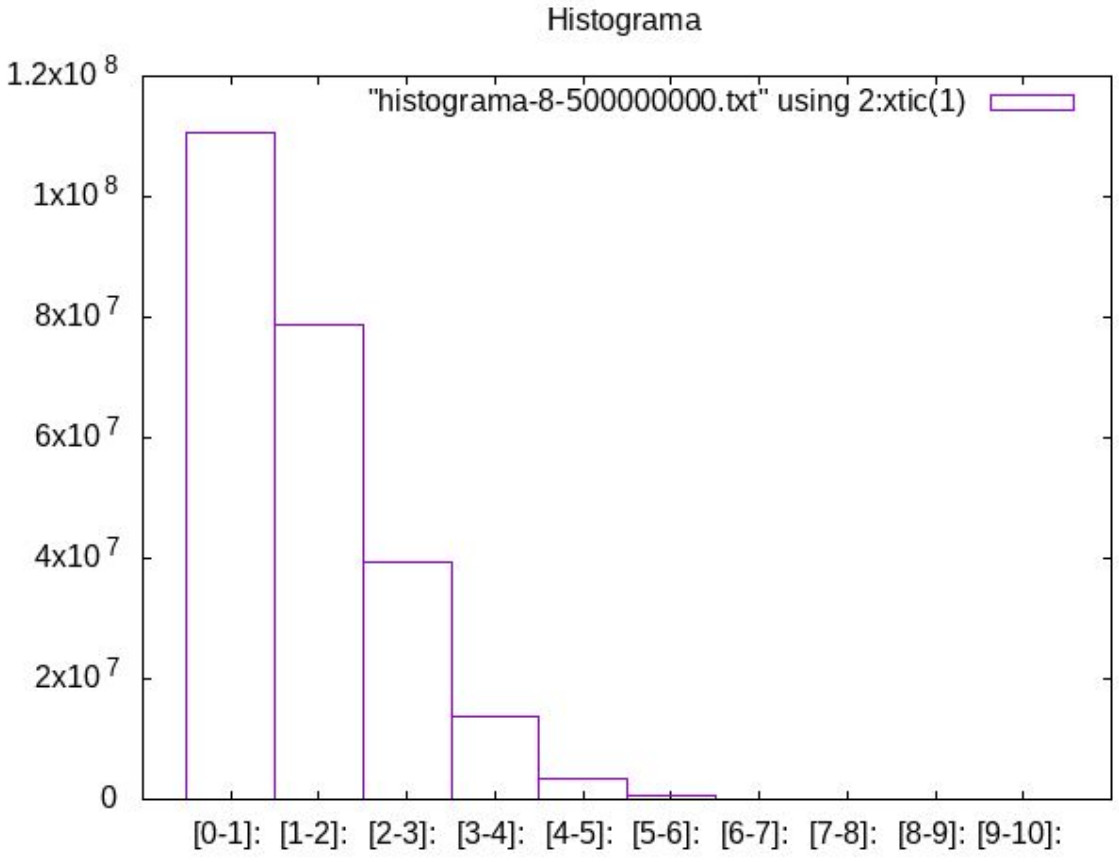
4 Cores


Histograma
"histograma-4-150000000.txt" using 2:xtic(1)
3.5x10 7
3x10 7
2.5x10 7
2x10 7
1.5x10 7
1x10 7
5x10 6
0
[0-1]:
[1-2]:
[2-3]:
[3-4]:
[4-5]:
[5-6]:
[6-7]:
[7-8]:
[8-9]:
[9-10]:

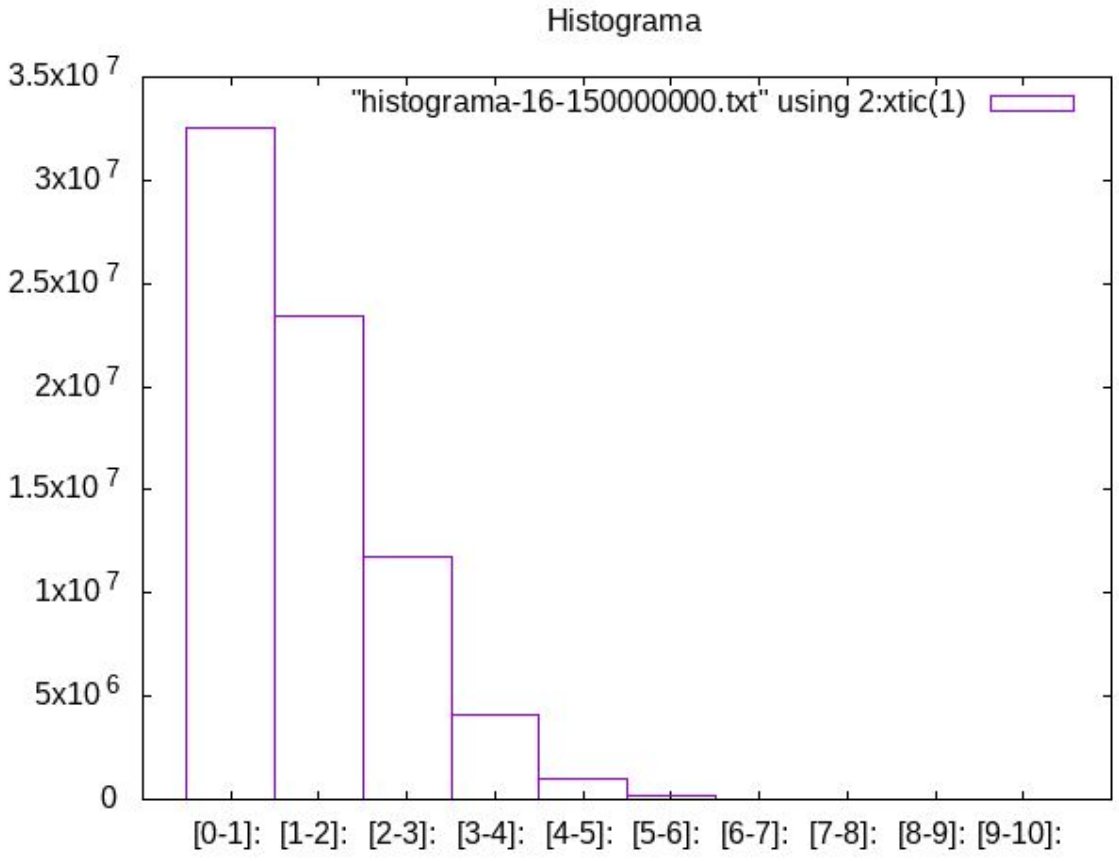

Histograma
"histograma-4-500000000.txt" using 2:xtic(1)
1.2x10 8
1x10 8
8x10 7
6x10 7
4x10 7
2x10 7
0
[0-1]:
[1-2]:
[2-3]:
[3-4]:
[4-5]:
[5-6]:
[6-7]:
[7-8]:
[8-9]:
[9-10]:

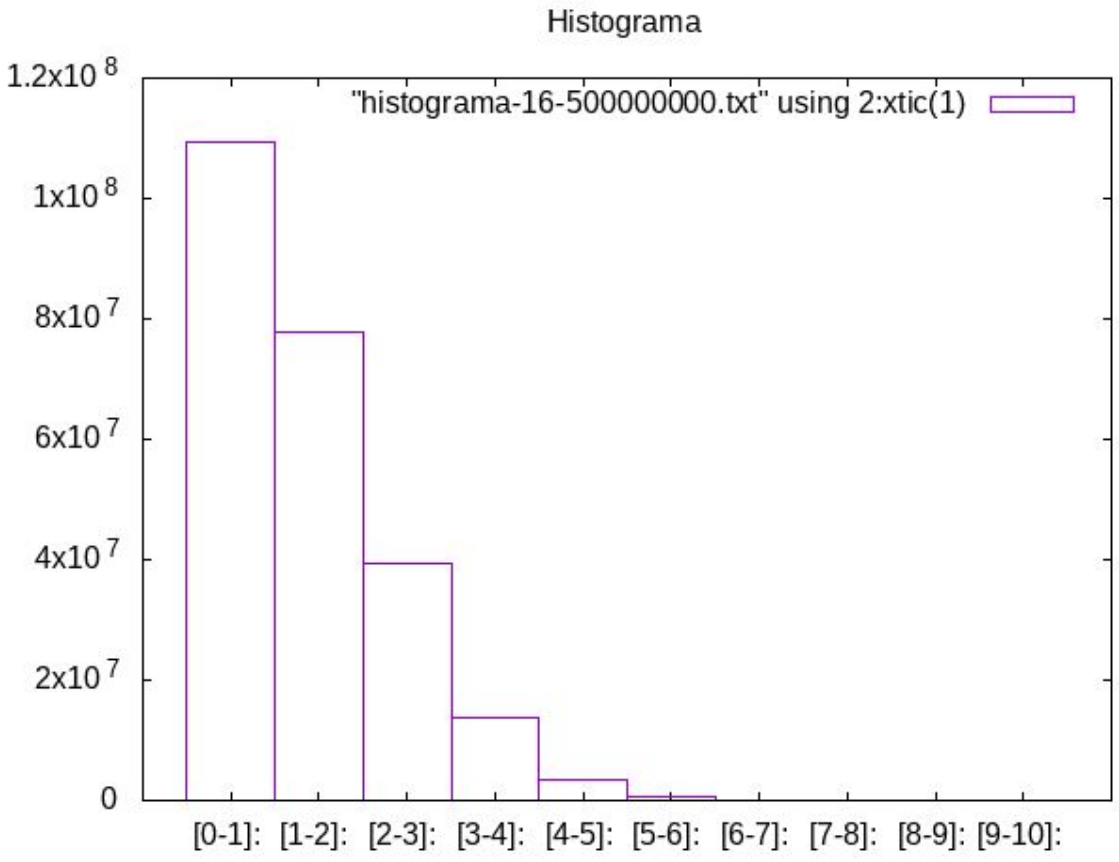
8 Cores


Histograma
"histograma-8-150000000.txt" using 2:xtic(1)
3.5x10 7
3x10 7
2.5x10 7
2x10 7
1.5x10 7
1x10 7
5x10 6
0
[0-1]:
[1-2]:
[2-3]:
[3-4]:
[4-5]:
[5-6]:
[6-7]:
[7-8]:
[8-9]:
[9-10]:

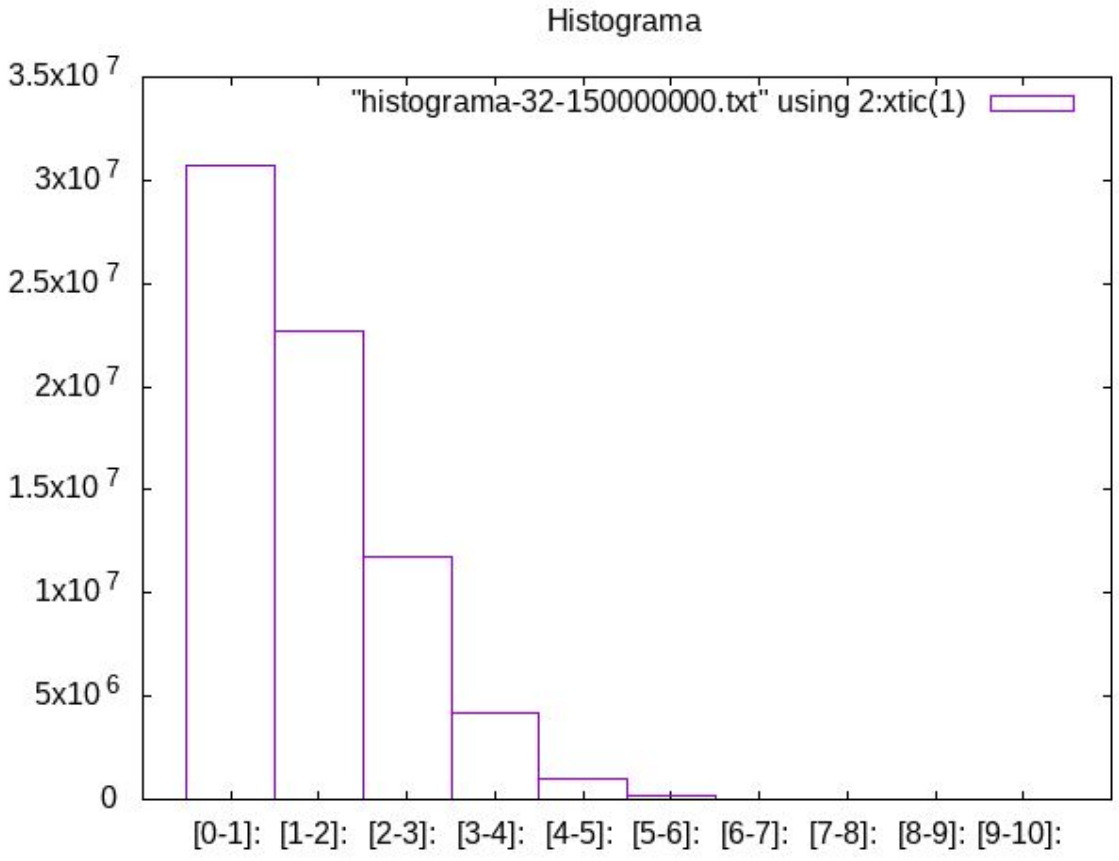

Histograma
"histograma-8-500000000.txt" using 2:xtic(1)
1.2x10 8
1x10 8
8x10 7
6x10 7
4x10 7
2x10 7
0
[0-1]:
[1-2]:
[2-3]:
[3-4]:
[4-5]:
[5-6]:
[6-7]:
[7-8]:
[8-9]:
[9-10]:

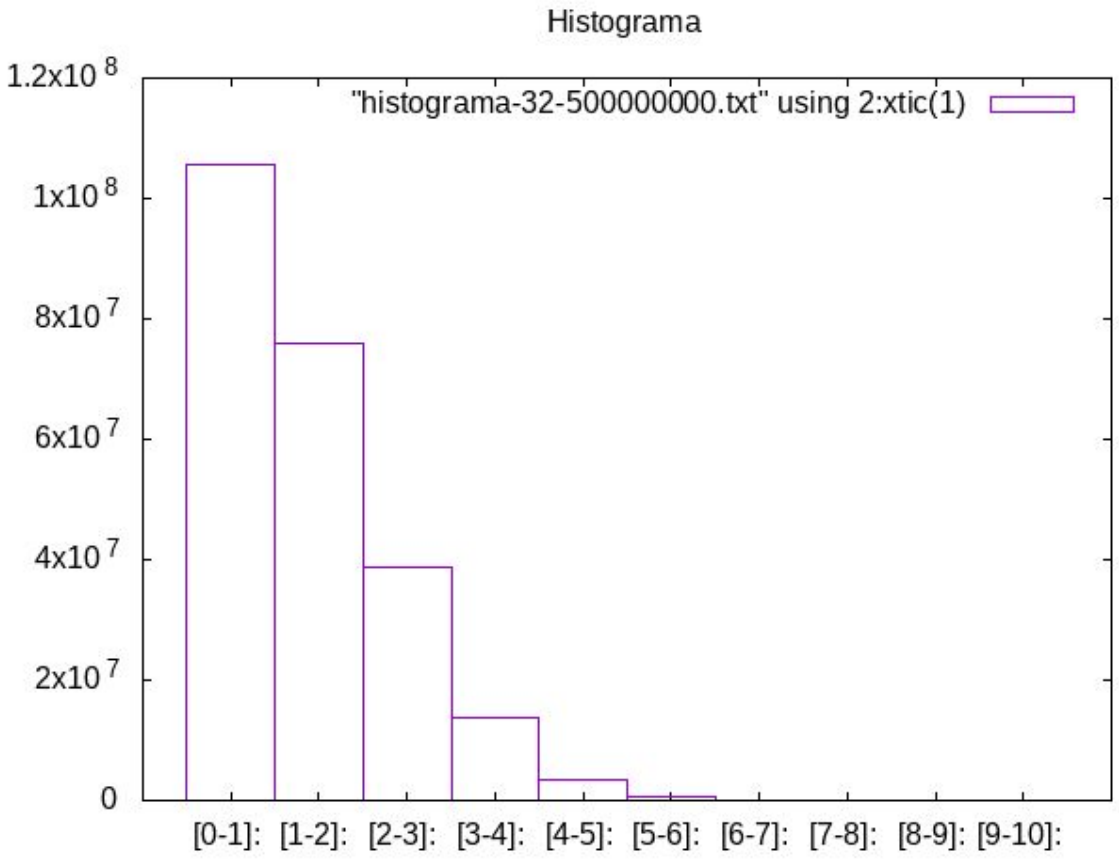
16 Cores


Histograma
"histograma-16-150000000.txt" using 2:xtic(1)
3.5x10 7
3x10 7
2.5x10 7
2x10 7
1.5x10 7
1x10 7
5x10 6
0
[0-1]:
[1-2]:
[2-3]:
[3-4]:
[4-5]:
[5-6]:
[6-7]:
[7-8]:
[8-9]:
[9-10]:


Histograma
"histograma-16-500000000.txt" using 2:xtic(1)
1.2x10 8
1x10 8
8x10 7
6x10 7
4x10 7
2x10 7
0
[0-1]:
[1-2]:
[2-3]:
[3-4]:
[4-5]:
[5-6]:
[6-7]:
[7-8]:
[8-9]:
[9-10]:

32 Core


Histograma
"histograma-32-150000000.txt" using 2:xtic(1)
3.5x10 7
3x10 7
2.5x10 7
2x10 7
1.5x10 7
1x10 7
5x10 6
0
[0-1]:
[1-2]:
[2-3]:
[3-4]:
[4-5]:
[5-6]:
[6-7]:
[7-8]:
[8-9]:
[9-10]:


Histograma
"histograma-32-500000000.txt" using 2:xtic(1)
1.2x10 8
1x10 8
8x10 7
6x10 7
4x10 7
2x10 7
0
[0-1]:
[1-2]:
[2-3]:
[3-4]:
[4-5]:
[5-6]:
[6-7]:
[7-8]:
[8-9]:
[9-10]:

Análise de Speedup, Eficiência e Escalabilidade

SpeedUp

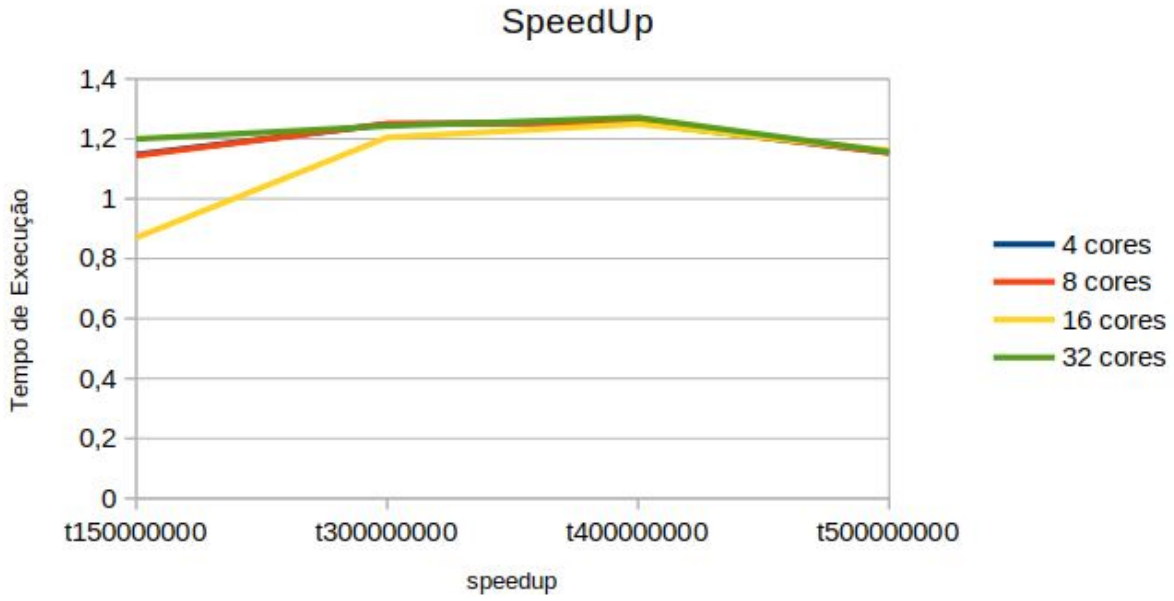
O speedup paralelo S definido como a relação entre o tempo de processamento serial Ts de um algoritmo e seu tempo de processamento paralelo Tp, tal que

$$S = \frac{T_s}{T_p}.$$

O speedup expressa quantas vezes o algoritmo paralelo é mais rápido do que o sequencial.
A seguir a tabela referente ao SpeedUp do código paralelo comparado com o serial.

Speedup				
Tamanho	150000000	300000000	400000000	500000000
4 cores	1,147608785	1,24912828	1,254420584	1,154415177
8 cores	1,143783806	1,250222645	1,255634183	1,153301335
16 cores	0,8703960245	1,204674106	1,249137885	1,16124011
32 cores	1,199503968	1,242518169	1,271122171	1,15690627

Em seguida temos o gráfico de SpeedUp onde podemos observar que o gráfico é constante em todos os cores.



Eficiência e Escalabilidade

Considere também a eficiência paralela Ef como a razão entre o speedup e o número de núcleos de processamento m, o que indica o quão bem os núcleos de processamento estão sendo utilizados na computação, na forma

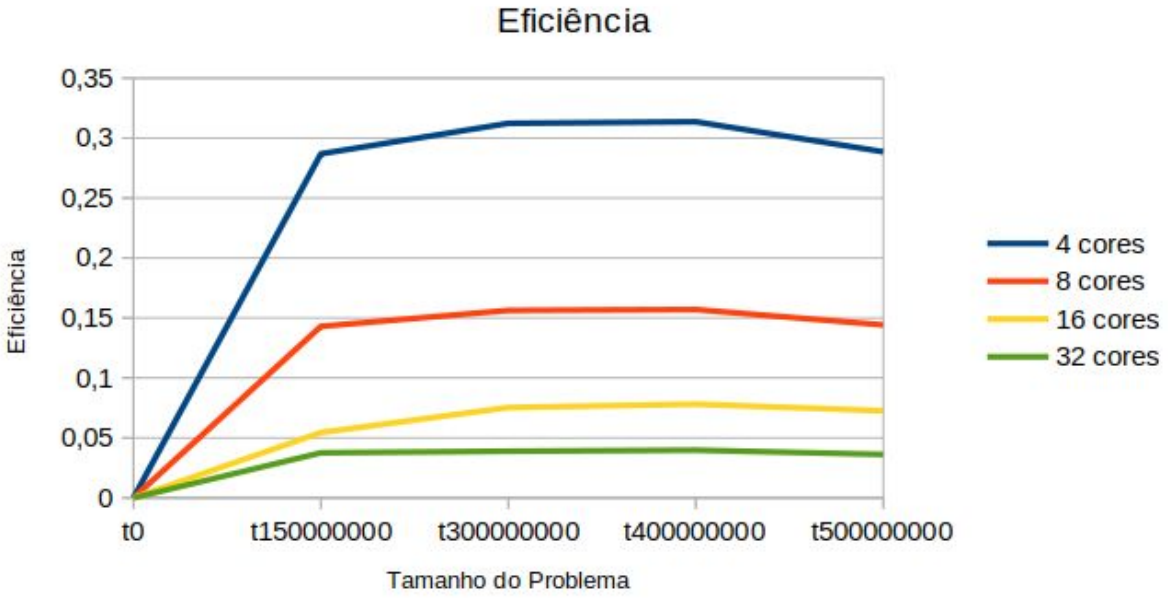
$$E_f = \frac{S}{m}.$$

A seguir a tabela referente a eficiência do código paralelo.

Eficiência				
Tamanho	4 cores	8 cores	16 cores	32 cores
t150000000	0,2869021961	0,1429729758	0,05439975153	0,037484499
t300000000	0,3122820699	0,1562778306	0,0752921316	0,03882869279
t400000000	0,3136051459	0,1569542729	0,07807111781	0,03972256785
t500000000	0,2886037941	0,1441626668	0,07257750685	0,03615332095

Comentário o gráfico a seguir na conclusão.

Eficiência



Conclusão

Após vários testes dá pra perceber que o código não foi muito eficiente nesse caso,pelo menos usando mais cores, a eficiência se manteve constante mesmo aumentando os números de cores, mas temos um aumento muito bom comparando os codigo serial,com o paralelo usando 4 cores, logo em seguindo não teve melhoras no desempenho,sendo assim podemos dizer que o código é fracamente escalável.