1. O PROBLEMA

Implementação do algoritmo de seleção descrito em
http://www.facom.ufms.br/~marco/paralelos2008/sel.pdf na linguagem C

2. ALGORITMOS

2.1. ALGORITMO SEQUENCIAL

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


int tamanho = 100000;


void pushZerosToEnd(int *arr, int n) {
   int temp[n];

   int j = 0;

   for (int i = 0; i < n; i++) {
       if (arr[i] != 0)
           temp[j++] = arr[i];
   }

   while (j < n)
       temp[j++] = 0;


   for (int i = 0; i < n; i++)
       arr[i] = temp[i];
}


int comp (const void * elem1, const void * elem2)
{
```

```
   int f = *((int*)elem1);
   int s = *((int*)elem2);
   if (f > s) return  1;
   if (f < s) return -1;
   return 0;
}

double round(double v) {
   return ((int)(v * 100 + .5)) / 100.;
}


int main()
{

srand(time(NULL));

int valores[tamanho];

int valores_sort[tamanho];

int metade1[tamanho/2];

int metade2[tamanho/2];

int mediana1;

int mediana2;

int mediana_ponderada;

int l1 = 0;
int e1 = 0;
int g1 = 0;

int l2 = 0;
int e2 = 0;
int g2 = 0;

int kth;

kth = rand() % (tamanho - 1) + 1;

printf("kth: %d\n", kth);

clock_t begin = clock();
```

```
for(int a = 0;a < tamanho;a++)
{
   valores[a] =  (rand() % 99) + 1;
}

for(int a = 0;a<tamanho;a++)
{
   valores_sort[a] = valores[a];
}

qsort (valores_sort, sizeof(valores_sort)/sizeof(*valores_sort),
sizeof(*valores_sort), comp);

for(int a = 0;a < tamanho;a++)
{
   printf("ValoresSort%d\n",valores_sort[a]);
}

memcpy(metade1, valores, tamanho/2 * sizeof(int));
memcpy(metade2, valores + tamanho/2, tamanho/2 * sizeof(int));

qsort (metade1, sizeof(metade1)/sizeof(*metade1), sizeof(*metade1),
comp);
qsort (metade2, sizeof(metade2)/sizeof(*metade2), sizeof(*metade2),
comp);
size_t tamanho_metade = (sizeof(valores) / sizeof(*valores)) / 2;

if(tamanho_metade%2 == 0)
{

   mediana1 = (metade1[tamanho_metade/2 - 1] + metade1[tamanho_metade/2])
/ 2;
   mediana2 = (metade2[tamanho_metade/2 - 1] + metade2[tamanho_metade/2])
/ 2;
}
else
{
   mediana1 = metade1[tamanho_metade/2];
   mediana2 = metade2[tamanho_metade/2];
}

int M[2] = {mediana1,mediana2};

int W[2] = {tamanho_metade,tamanho_metade};
```

```
int soma_pesos = tamanho_metade*2;

int i = 0;

int sum = soma_pesos - W[0];

while (sum > soma_pesos/2)
{
   ++i;
   sum -=W[i];
}

mediana_ponderada = M[i];

printf("mediana_ponderada:%d\n\n",mediana_ponderada);

for(int a = 0;a < tamanho_metade;a++)
{
   if(metade1[a] < mediana_ponderada)
   l1++;
   else if(metade1[a] == mediana_ponderada)
   e1++;
   else if(metade1[a] > mediana_ponderada)
   g1++;
}

for(int a = 0;a < tamanho_metade;a++)
{
   if(metade2[a] < mediana_ponderada)
   l2++;
   else if(metade2[a] == mediana_ponderada)
   e2++;
   else if(metade2[a] > mediana_ponderada)
   g2++;
}

int L = l1 + l2;
int E = e1 + e2;
int G = g1 + g2;

printf("L:%d E:%d G:%d\n",L,E,G);

if(L < kth && kth <= L + E)
{
   printf("A solucaoL é: %d\n",mediana_ponderada);
```

```
}
else if(kth <= L)
{
  int index = 0;
  int novos_valores[tamanho];

  for(int a = 0; a<tamanho;a++)
  {
   novos_valores[a] = 0;
  }

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

   if(valores[a] < mediana_ponderada)
   {
       novos_valores[index] = valores[a];
       printf("valoresN:%d\n",novos_valores[index]);
       index++;
   }
  }

   qsort (novos_valores, sizeof(novos_valores)/sizeof(*novos_valores),
sizeof(*novos_valores), comp);
   pushZerosToEnd(novos_valores,tamanho);
   printf("A solucaoE é:%d\n",novos_valores[kth - 1]);
}

else if(kth > L + E)
{
  int index = 0;
  int index_medina_ponderada = 0;
  int novos_valores[tamanho];

  for(int a = 0; a<tamanho;a++)
  {
   novos_valores[a] = 0;
  }


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

   if(valores[a] > mediana_ponderada)
   {
       novos_valores[index] = valores[a];
```

```
        printf("valoresN:%d\n",novos_valores[index]);
        index++;
    }
  }
   qsort (novos_valores, sizeof(novos_valores)/sizeof(*novos_valores),
sizeof(*novos_valores), comp);
   pushZerosToEnd(novos_valores,tamanho);
   for(int a = 0; a < tamanho;a++)
   {
        if(valores_sort[a] == mediana_ponderada){
        index_medina_ponderada = a;
        }
   }

   printf("A solucaoG é: %d\n",novos_valores[kth - index_medina_ponderada
- 2]);
}

clock_t end = clock();
double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
printf("\nTempo de Execucao:%d\n",time_spent);

}
```

2.2. ALGORITMO PARALELO COM OPENMP

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

int tamanho = 1000;

struct lessEqualGreater
{
   int l;
```

```
    int e;
    int g;
};

void pushZerosToEnd(int *arr, int n) {
    int temp[n];


    int j = 0;


    for (int i = 0; i < n; i++) {
        if (arr[i] != 0)
            temp[j++] = arr[i];
    }


    while (j < n)
        temp[j++] = 0;


    for (int i = 0; i < n; i++)
        arr[i] = temp[i];
}


int comp (const void * elem1, const void * elem2)
{
    int f = *((int*)elem1);
    int s = *((int*)elem2);
    if (f > s) return  1;
    if (f < s) return -1;
    return 0;
}

double round(double v) {
    return ((int)(v * 100 + .5)) / 100.;
}

int tarefa1(int array[])
{
int mediana = 0;

if((tamanho/2)%2 == 0)
{
```

```
   mediana = (array[tamanho/4 - 1] + array[tamanho/4]) / 2;

}
else
{
   mediana = array[tamanho/4];

}
   return mediana;
}

struct lessEqualGreater tarefa2(int m,int array[])
{

int l = 0;
int e = 0;
int g = 0;

for(int a = 0;a < tamanho/2;a++)
{
   if(array[a] < m)
   l++;
   else if(array[a] == m)
   e++;
   else if(array[a] > m)
   g++;
}

struct lessEqualGreater x = {l,e,g};

return x;
}

void tarefa3(int array[],int l,int e,int g,int kth,int m,int **ptr,int
novos_valores[])
{

*ptr = novos_valores;

if(kth <= l)
{
  int index = 0;

  for(int a = 0; a<tamanho/2;a++)
  {
   novos_valores[a] = 0;
```

```
  }

  for(int a = 0;a < tamanho/2;a++)
  {

   if(array[a] < m)
   {
       novos_valores[index] = array[a];
       printf("valoresN:%d\n",novos_valores[index]);
       index++;
   }
  }

   qsort (novos_valores, tamanho/2, sizeof(*novos_valores), comp);
   pushZerosToEnd(novos_valores,tamanho/2);
}

else if(kth > l + e)
{
  int index = 0;
  int index_medina_ponderada = 0;

  for(int a = 0; a<tamanho/2;a++)
  {
   novos_valores[a] = 0;
  }

  for(int a = 0;a < tamanho/2;a++)
  {

   if(array[a] > m)
   {
       novos_valores[index] = array[a];
       printf("valoresN:%d\n",novos_valores[index]);
       index++;
   }
  }
   qsort (novos_valores, tamanho/2, sizeof(*novos_valores), comp);
   pushZerosToEnd(novos_valores,tamanho/2);

}
}

int main()
{
```

```
srand(time(NULL));

int valores[tamanho];

int valores_sort[tamanho];

int metade1[tamanho/2];

int metade2[tamanho/2];

int mediana1;

int mediana2;

int mediana_ponderada;

int kth;

struct lessEqualGreater struct1;

struct lessEqualGreater struct2;

kth = rand() % (tamanho - 1) + 1;

clock_t begin = clock();

printf("kth: %d\n", kth);

for(int a = 0;a < tamanho;a++)
{
   valores[a] =  (rand() % 99) + 1;
}

for(int a = 0;a<tamanho;a++)
{
   valores_sort[a] = valores[a];
}

qsort (valores_sort, sizeof(valores_sort)/sizeof(*valores_sort),
sizeof(*valores_sort), comp);

for(int a = 0;a < tamanho;a++)
{
   printf("ValoresSort:%d\n",valores_sort[a]);
}
```

```
for(int a = 0;a < tamanho;a++)
{
   printf("Valores:%d\n",valores[a]);
}

memcpy(metade1, valores, tamanho/2 * sizeof(int));
memcpy(metade2, valores + tamanho/2, tamanho/2 * sizeof(int));

qsort (metade1, sizeof(metade1)/sizeof(*metade1), sizeof(*metade1),
comp);
qsort (metade2, sizeof(metade2)/sizeof(*metade2), sizeof(*metade2),
comp);
size_t tamanho_metade = (sizeof(valores) / sizeof(*valores)) / 2;

#pragma omp parallel
{
   #pragma omp sections
   {
        #pragma omp section
        {
        mediana1 = tarefa1(metade1);
        }
        #pragma omp section
        {
        mediana2 = tarefa1(metade2);
        }
   }
}

int M[2] = {mediana1,mediana2};

int W[2] = {tamanho_metade,tamanho_metade};

int soma_pesos = tamanho_metade*2;

int i = 0;

int sum = soma_pesos - W[0];

while (sum > soma_pesos/2)
{
   ++i;
   sum -=W[i];
}

mediana_ponderada = M[i];
```

```
printf("mediana_ponderada:%d\n\n",mediana_ponderada);

#pragma omp parallel
{
   #pragma omp sections
   {
        #pragma omp section
        {
        struct1 = tarefa2(mediana_ponderada,metade1);
        }
        #pragma omp section
        {
        struct2 = tarefa2(mediana_ponderada, metade2);
        }
   }
}
   int L = struct1.l + struct2.l;
   int E = struct1.e + struct2.e;
   int G = struct1.g + struct2.g;

   printf("L:%d E:%d G:%d\n",L,E,G);

   int* ptr1;
   int* ptr2;
   int valores1[tamanho/2];
   int valores2[tamanho/2];
   int novos_valores[tamanho];

   for(int b = 0;b <tamanho;b++)
   {
        novos_valores[b] = 0;
   }

   #pragma omp parallel
{
   #pragma omp sections
   {
        #pragma omp section
        {
        tarefa3(metade1,L,E,G,kth,mediana_ponderada,&ptr1,valores1);
        }
        #pragma omp section
        {
        tarefa3(metade2,L,E,G,kth,mediana_ponderada,&ptr2,valores2);
        }
```

```
    }
}


    int size = 0;
    for(int a = 0;ptr1[a]!=0;a++)
    {
        novos_valores[a] = ptr1[a];
        size++;
    }

    for(int b = 0; ptr2[b] != 0;b++)
    {
        novos_valores[size + b] = ptr2[b];
    }

    qsort (novos_valores, sizeof(novos_valores)/sizeof(*novos_valores),
sizeof(*novos_valores), comp);
    pushZerosToEnd(novos_valores,tamanho);

if(L < kth && kth <= L + E)
{
    printf("A solucaoL é: %d\n",mediana_ponderada);
    }
else if(kth <= L)
{

    printf("A solucaoE é:%d\n",novos_valores[kth - 1]);
}

else if(kth > L + E)
{
  int index_medina_ponderada = 0;

    for(int a = 0; a < tamanho;a++)
    {
        if(valores_sort[a] == mediana_ponderada){
        index_medina_ponderada = a;
        }
    }

    printf("A solucaoG é: %d\n",novos_valores[kth - index_medina_ponderada
- 2]);
}

clock_t end = clock();
```

```
double time_spent = (double)(end - begin) / CLOCKS_PER_SEC;
printf("\nTempo de Execucao:%d\n",time_spent);

}
```

2.3. ALGORITMO PARALELO COM OPENMPI

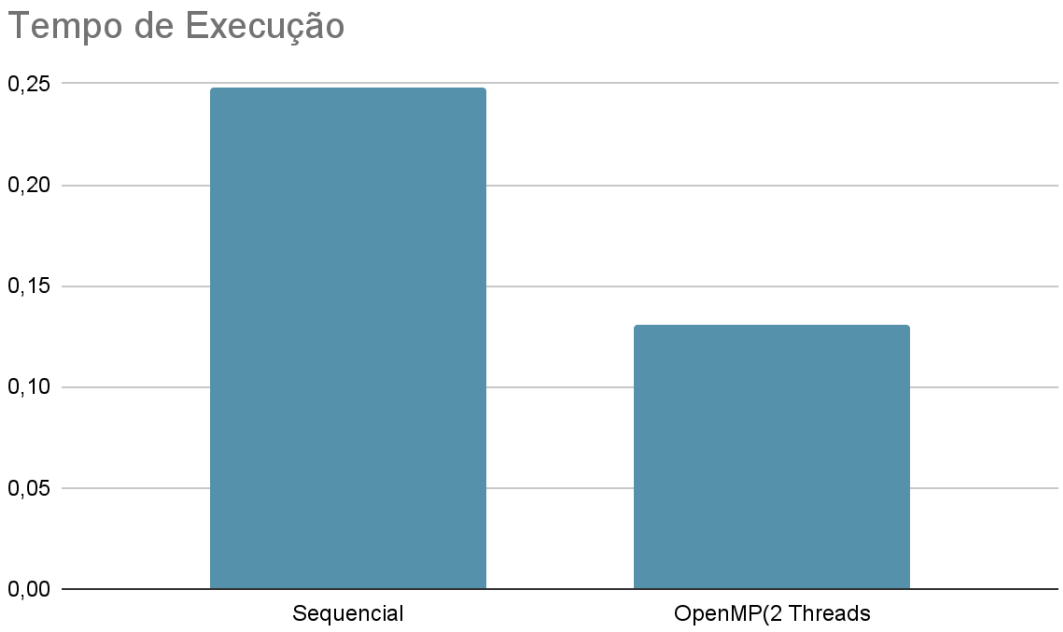
```
```

3. ANÁLISE DE DESEMPENHO

3.1. RESULTADOS

Algoritmo	Quantidade de elementos	Tempo (segundos)	Speedup	Comentários
Sequencial	100.000	0.248	1.0x	
OpenMP (2 threads)	100.000	0.131	1.9x	

3.2. GRÁFICOS


Tempo de Execução
0,25
0,20
0,15
0,10
0,05
0,00
Sequencial
OpenMP(2 Threads

4. HARDWARE E SOFTWARE

HARDWARE/SOFTWARE	MODELO
CPU	13th Gen Intel(R) Core(TM) i5-13450HX 2.40 GHz
RAM	2Gb
SISTEMA OPERACIONAL	Debian 64 Bit
LINGUAGEM DE PROGRAMAÇÃO	C
COMPILADOR	gcc 11.2 https://bigsearcher.com/mirrors/gcc/releases/gcc-11.2.0/
MPI	

5. IMPLEMENTAÇÃO

ViniDinz/ProjetoConcorrencia (github.com)

6. DIFICULDADES ENCONTRADAS

Configurar o compilador do MPI
Instalar o Slurm Cluster
Colocar o tamanho como maior que 100.000 dava segmentation fault

7. REFERÊNCIAS