Relatório do Trabalho I de Programação Paralela

Diego Luiz N. Gonçalves[1],Thainara Orneles Matos[2], Ygor Takashi Nishi[3]

[1]Universidade Federal de Mato Grosso do Sul- Campus de Ponta Porã (UFMS-CPPP)
Res. Julia de Oliveira Cardinal – Ponta Porã– Mato Grosso do Sul – MS – Brazil

diegoreke@hotmail.com, orneles.thainara23@gmail.com,ygortn14@gmail.com

Resumo. *O trabalho realizado tem como objetivo analisar a diferença do tempo de execução entre um código sequencial e um código paralelo, utilizando a API de desenvolvimento CUDA, e a linguagem de programação C, na resolução do problema onde deve-se multiplicar todos os elementos de um vetor de tamanho 2^n, onde n começa em 10 e termina em 19 (pela limitação apresentada no código sequencial); pelo número π com precisão de 4 casas decimais.*
No código em paralelo, na função responsável por chamar a GPU decidimos por fixar 1024 blocos, e o número de threads é max/nb (onde max=tamanho do vetor,nb =tamanho do bloco).
Para a realização dos códigos, foram realizados encontros com todos os membros, e os mesmos foram desenvolvidos no Google Colab. Como ferramenta auxiliar de comunicação e organização do projeto foi utilizado o Trello.

1. Código Sequencial

```c
/*PROGRAMA SEQUENCIAL*/

#include <stdio.h>
#include <time.h> //biblioteca para calcular o tempo
#include <stdlib.h>

#define max 32768 //tamanho do vetor
#define pi 3.1415 // numero pi com 4 casas decimais

int main(){
    /*Variável que contabilizará o tempo gasto*/
    double time=0.0;
    /*Vetor onde serão realizado os cálculos*/
    double vet[max];

    /*Variável do tipo clock_t, que inicia a contagem do tempo*/
    clock_t begin=clock();

    /*Inicio do processamento*/
    for(long int i=0;i<max;i++){
        vet[i]= i*pi;
    }
    /*Fim do processamento*/
```

```
    /*Variável do tipo clock_t, que termina a contagem do tempo*/
    clock_t end=clock();

    /*Calculando o tempo em segundos*/
    time+= (double)(end - begin) / CLOCKS_PER_SEC;

    /*Mostrando o tempo final*/
    printf("Tempo gasto: %f segundos", time);
    return 0;
}
```

2. Código em paralelo

```
/*PROGRAMA PARALELO*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define max 1024 //Tamanho do Vetor
#define pi 3.1415 //numero PI com 4 casas decimais

/*Função que será executada pelos blocos da GPU e suas threads*/
__global__ void mpi(double *a, double *b){
      // id = id do bloco * dimensão do bloco * thread do block
      int id= blockIdx.x * blockDim.x + threadIdx.x;

      /*
      Verificação necessária para evitar acessos indevidos a memória
      */
      if(id< max)
        b[id]= a[id]*pi; //realizando a operação de multiplicação
}

int main(){
    /*Variável para armazenar o tempo final gasto no processamento*/
    double time=0.0;

    /*
      --> Como as variáveis estão sendo utilizadas:
      - h_a : ponteiro do tipo double que armazena o endereço do
      vetor na CPU e que será preenchido com seu índice de
      posição + 1 posição
      - h_b : ponteiro do tipo double que armazena o endereço do
      vetor na CPU onde será guardado o resultado da operação de
      multiplicação do vetor h_a
```

```
  - d_a : ponteiro do tipo double que armazena o endereço do
  vetor na GPU e que receberá o vetor h_a pela função
  cudaMemcpy()
  - d_b : ponteiro do tipo double que armazena o endereço do
  vetor na GPU e que receberá o resultado da operação da
  multiplicação do vetor d_a
*/

/*
    Variáveis do tipo double e ponteiro, que são usados pela CPU e
    que armazenam o endereço dos vetores que serão alocados
*/

double *h_a, *h_b;

/*
    Alocando dinamicamente um vetor na memória de tamanho
    max * tamanho de double
*/
h_a= (double*) malloc(max*sizeof(double));

/*
    Alocando dinamicamente um vetor na memória de tamanho
    max * tamanho de double
*/
h_b= (double*) malloc(max*sizeof(double));

/*
    Laco que inicia cada posição da memória com seu
    respectivo índice + 1 unidade
*/
for(int i=0;i<max;i++){
    h_a[i]= i+1;
}
/*
    Variáveis do tipo double e ponteiro que guardarão os
    endereços dos vetores na GPU
*/
double *d_a, *d_b;

/*
    Alocando um vetor na memoria da GPU de tamanho
    max * tamanho de double
*/
cudaMalloc(&d_a,max*sizeof(double));
/*
```

```
    Alocando um vetor na memoria da GPU de tamanho
    max * tamanho de double
*/
cudaMalloc(&d_b,max*sizeof(double));

/*
  Função que copia o conteúdo do vetor h_a (CPU) para o vetor
  d_a (CPU)
  Parâmetros: (destino, origem, tamanho do elemento a ser
  copiado, tipo de cópia)
*/
cudaMemcpy(d_a,h_a,max*sizeof(double),cudaMemcpyHostToDevice);

//---------------------------------------------------------------
/*Variável do tipo clock_t, que inicia a contagem do tempo*/
clock_t begin=clock();

/*
  Invocando método responsável por realizar a operação
  Parâmetros: nome_função<<<numero de blocos, qtd
  threads>>>(argumento 1, argumento 2);
*/
mpi<<<1024,1>>>(d_a,d_b);


/*Variável do tipo clock_t, que termina a contagem do tempo*/
clock_t end=clock();
//---------------------------------------------------------------

/*
  Função que copia o conteúdo do vetor d_b (GPU) para o vetor h_b
  (CPU)
  Parâmetros: (destino, origem, tamanho do elemento a ser
  copiado, tipo de copia)
*/
cudaMemcpy(h_b, d_b, max * sizeof(double), cudaMemcpyDeviceToHost);

/*Calculando o tempo em segundos*/
time+= (double)(end - begin) / CLOCKS_PER_SEC;

/*Exibindo o resultado final*/
printf("Tempo gasto: %f segundos", time);

/*Liberando a memória da GPU*/
cudaFree(d_a);
cudaFree(d_b);
```

```
    /*Liberando a memória da CPU*/
    free(h_a);
    free(h_b);


    return 0;
}
```

3. Comparativo entre sequencial e paralelo

Entrada	Tempo Sequencial ($segundos$)	Tempo Paralelo ($segundos$)	Speed up
2^{10}	0,0000066	0,0000707	0,093352192
2^{11}	0,0000084	0,0000875	0.096
2^{12}	0,0000254	0,000016	1,5875
2^{13}	0,0000485	0,0000202	2,400990099
2^{14}	0,0001018	0,0000211	4,82464455
2^{15}	0,0002044	0,0000202	10,11881188
2^{16}	0,0003999	0,0000207	19,31884058
2^{17}	0,0008107	0,0000239	33,92050209
2^{18}	0,0016843	0,0000255	66,05098039
2^{19}	0,0033249	0,0000281	118,3238434
2^{20}	-	0,0000239	-

Tabela 1. Tabela comparativa entre o tempo sequencial vs. tempo paralelo e o speed up