Programação em CUDA)

Douglas Diniz Landim - 76681
Luiz Otávio Passos - 122.039

***Resumo*—Exercícios para a prática de programação com GPU - Linguagem CUDA.**

***Index Terms*—CUDA. Programação Concorrente.**

I. Introduction

Programação em CUDA
Compute Unified Device Architecture, o CUDA, é uma plataforma de computação concorrente da Nvidia para permitir que códigos serem executados através de placas de vídeo (GPUs). As linguagens compatíveis com CUDA são C, C++, Java, Fortran e Python. A plataforma CUDA então oferece uma camada de tradução entre a linguagem e os recursos da placa de vídeo através do uso de bibliotecas de linguagem.

Apesar do alto poder de processamento dos atuais processadores multi-cores, uma placa de vídeo suporte uma quantia de operações concorrentes muito maior do que um processador comum. Algumas aplicações, como processamento de vídeo, simulações físicas ou de fenômenos da natureza, previsão do tempo, possuem alto volume de dados, onde a quantia de núcleos de processamento domina assintoticamente o poder de processamento individual de cada núcleo, em termos de eficência de processamento.

Leia mais em:
`https://developer.nvidia.com/cuda-zone`

II. Definição do problema

Forma de entrega: Arquivo contendo breve relatório de desempenho e códigos-fonte. Não se esqueça de especificar a GPU utilizada no cluster.

1- Converta o programa serial para calc. conducao calor 1D (Arquivo: "fcts.c") para executar em GPU.

OBS: Preocupe-se em converter apenas o trecho responsável pelo cálculo das linhas 41 até 46:

```
for (i=1; i<n; i++) {
  u[i]=u_prev[i]+kappa*dt/(dx*dx)*
    (u_prev[i-1]-2*u_prev[i]+u_prev[i+1]);
  x += dx;
}

/* forca condicao de contorno */
u[0] = u[n] = 0.;

/* troca entre ponteiros */
tmp = u_prev; u_prev = u; u = tmp;
```

Escreva um breve relatório contendo os seguintes resultados: a) Desempenho com diferentes configurações de grade e blocos variando a quantidade de threads por bloco da seguinte forma: 512, 256, 128, 64, verificando em que situação o algoritmo melhorou o desempenho. b) Desempenho com o uso de shared memory para otimizar o acesso aos dados dentro de um bloco. Faça um comparativo de uma versão com e sem o uso de shared memory, para a configuração de melhor desempenho com relação a quantidade de threads por bloco.

III. Ambiente de desenvolvimento

Executado no cluster da Unifesp.

IV. EXPERIMENTOS

EXPERIMENTOS EM BLOCOS SEM SHARED MEMORY:

Thr/Bloco	Blocos	Cuda(ms)	Razão	S/ Cuda(ms)	Total(ms)
64	1.56	171.93	70.58	71.45	243.38
128	781	147.73	76.23	45.88	193.61
256	390	149.45	77.95	42.07	191.52
512	195	154.95	78.73	41.66	196.61

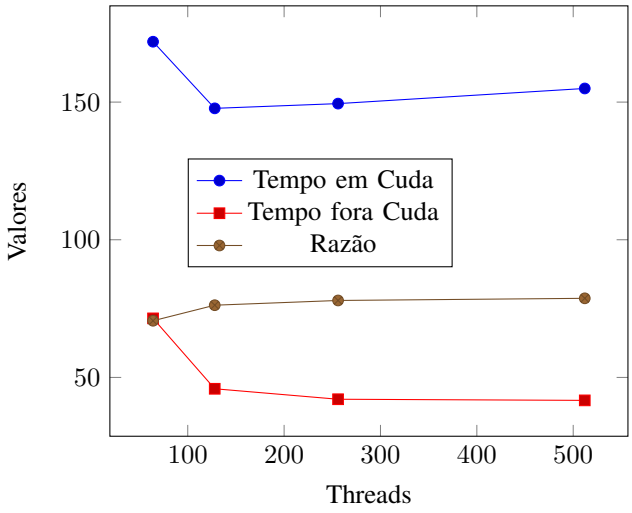
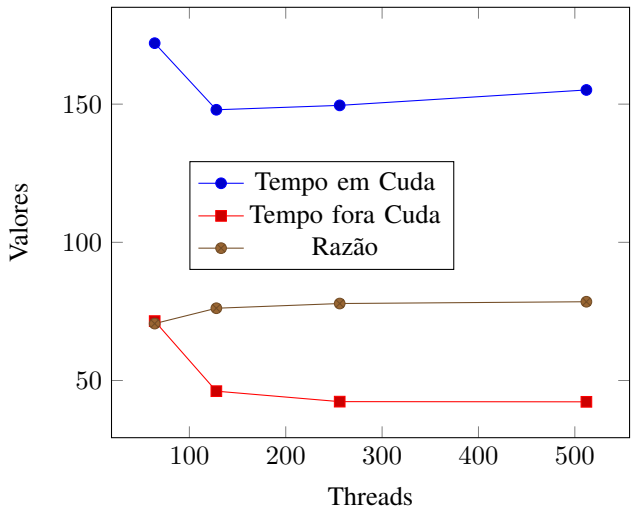


Figura 1. Experimentos Sem Barreira

EXPERIMENTOS EM BLOCOS COM SHARED MEMORY:

Thr/Bloco	Blocos	Cuda(ms)	Razão	S/ Cuda(ms)	Total(ms)
64	1.56	172.06	70.59	71.49	243.55
128	781	147.96	76.15	46.14	194.1
256	390	149.55	77.85	42.35	191.9
512	195	155.13	78.5	42.28	197.41



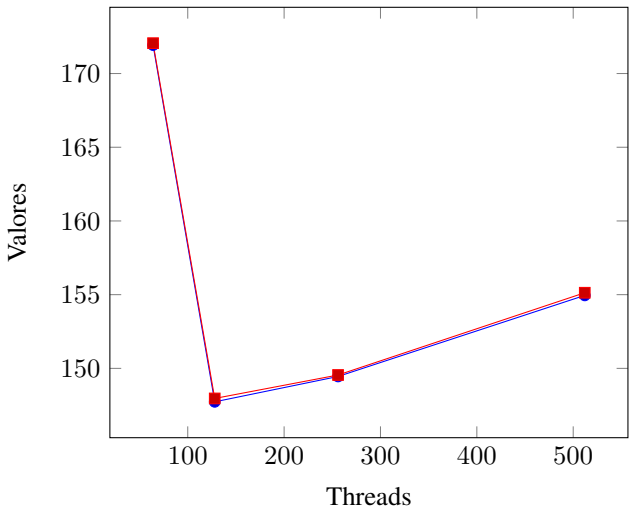


Figura 2. Experimentos Com Barreira

V. Análise dos resultados

Comparação do tempo em cuda com e sem barreira:

VI. Adendos

```c
#include <stdio.h>
#include <stdlib.h>

#define tam 1.0
#define dx 0.00001
#define dt 0.000001
#define T 0.01
#define kappa 0.000045

void main(void) {

  double *tmp, *u, *u_prev;
  double x, t;
  long int i, n, maxloc;

  /* Claculando quantidade de pontos */
  n = tam/dx;

  /* Alocando vetores */
  u = (double *) malloc((n+1)*sizeof(double));
  u_prev = (double *)
      malloc((n+1)*sizeof(double));

  printf("Inicio: qtde=%ld, dt=%g, dx=%g,
      dx=%g, kappa=%f, const=%f\n",
    (n+1), dt, dx, dx*dx, kappa,
        kappa*dt/(dx*dx));
  printf("Iteracoes previstas: %g\n", T/dt);

  x = 0;
  for (i=0; i<n+1; i++) {
    if (x<=0.5) u_prev[i] = 200*x;
    else      u_prev[i] = 200*(1.-x);
    x += dx;
  }

  printf("dx=%g, x=%g, x-dx=%g\n", dx, x,
      x-dx);

  printf("u_prev[0,1]=%g,
      %g\n",u_prev[0],u_prev[1]);
  printf("u_prev[n-1,n]=%g,
      %g\n",u_prev[n-1],u_prev[n]);

  t = 0.;
  while (t<T) {
    x = dx;
    for (i=1; i<n; i++) {
      u[i] = u_prev[i] +
          kappa*dt/(dx*dx)*(u_prev[i-1]-2*u_prev[i]+u_prev
      x += dx;
    }
    u[0] = u[n] = 0.; /* forca condicao de
        contorno */
```

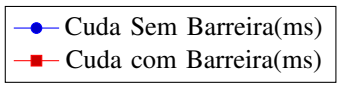


Figura 3. Comparação sem e com barreira

```
46      tmp = u_prev; u_prev = u; u = tmp; /*
            troca entre ponteiros */
47      t += dt;
48    }
49
50    /* Calculando o maior valor e sua
          localizacao */
51    maxloc = 0;
52    for (i=1; i<n+1; i++) {
53      if (u[i] > u[maxloc]) maxloc = i;
54    }
55    printf("Maior valor u[%ld] = %g\n", maxloc,
          u[maxloc]);
56
57  }
```