Universidade Federal de São Carlos

Programa de Pós-Graduação em Ciência da Computação

CCO-129-7 Introdução à Computação de Alto Desempenho (2021/2)

Exercício Programa - EP 5

OpenHPC UFSCar

Alcides Mignoso e Silva 760479

Submission Date : 13/04/2022

1 Atividade

O objetivo desta atividade é paralelizar a execução do algoritmo de Warshall para que seja executado em GPUs através de CUDA. O código base utilizado foi o existente em `https://github.com/HPCSys-Lab/HPC-101/blob/main/examples/warshall/appWarshall.c` e este relatório visa discutir valores de tempo de execução, speedup e escalabilidade das execuções.

As execuções foram realizadas utilizando matrizes com o número de linhas e colunas igual a 1024, 2048, 2071 e 4096.

Em relação a paralelização, houve uma tentativa de dividir as tarefas de forma a utilizar um grid tri-dimensional, mas por conta do tamanho das matrizes foi necessário trabalhar com duas dimensões e iterar em relação a terceira dentro do código do kernel a ser executado na GPU. O melhor cenário apareceu utilizando 8x8 threads para N/8 x N/8 blocos a serem executados na GPU - sendo N o número de linhas e colunas da matriz.

```
__global__ void warshall_gpu(int *m) {
  int k = blockIdx.x * blockDim.x + threadIdx.x;
  int i = blockIdx.y * blockDim.y + threadIdx.y;

  if (k >= NUM_ELEMENTS || i >= NUM_ELEMENTS)
    return;
  for (int j = 0; j < NUM_ELEMENTS; j++) {
    if (getAt(m, k, j) == 1 && getAt(m, i, k) == 1) {
      setAt(m, i, j, 1);
    }
  }
}
```

2 Resultados

2.1 Tabelas

A tabela abaixo trás os tempos de execução do programa (em segundos) na versão sequencial e na paralelizada.

Dimensões da matriz	T(sequencial)	T(cuda)	Speedup
1024x1024	4,2127s	0,0247s	170,66614
2048x2048	26,2407s	0,1462s	179,542671
3072x3072	87,6586s	0,5060s	173,2430581
4096x4096	218,1629s	1,1702s	186,4375653

2.2 Discussão

Como pode-se perceber através do gráfico de speedup e da tabela de tempos de execução, o tempo de execução do problema na GPU é duas ordens de grandeza menor do que quando executado na CPU. Por conta da simplicidade do problema

e falta de dependência entre dados, o problema mostra ser fácil de ser paralelizado e por isso a grande quantidade de núcleos de uma GPU realizando as mesmas operações faz com que seja vantajoso utilizá-la.

Em resumo, a versão final faz com que as threads na GPU tenham valores de duas dimensões fixados e itera entre a terceira dimensão - equivalente ao terceiro laço de repetição no código sequencial. Como não existem muitas estruturas condicionais dentro do kernel, nem dependência entre dados, e as operações são as mesmas para qualquer posição da matriz, a paralelização utilizando cuda se mostra muito eficiente.