MAC 0219/5742 – Introdução à Computação Concorrente, Paralela e Distribuída

Prof. Dr. Alfredo Goldman

2º Exercício Programa `versão 1.0` - CUDA

Monitores: Giuliano Belinassi e Marcos Amaris

1 Introdução

A redução é uma operação extremamente importante para a Computação Paralela e Distribuida, sendo um dos principais conceitos do `MapReduce`. Devido a tamanha importância, a Nvidia não pode deixar de implementar maneiras para acelerar este tipo de operação em suas GPUs [1] [2] [3].

A redução é uma estratégia de divisão e conquista, e consiste em definir um operador sobre todos os elementos de uma coleção de forma a particionar o problema, resolver as partes, e combiná-las para obter a solução final. Neste exercício programa, você deverá implementar uma operação específica de redução em `CUDA`, conforme definida na Subseção 2.2.

2 O Operador Redução

2.1 Na Teoria

Seja R um conjunto de elementos. Uma operação binária $\oplus$ sobre R é uma operação de redução se e somente se ele respeita as seguintes três condições:

1. *Fechado*: Para todo $r_1, r_2 \in R$, tem-se que $r_1 \oplus r_2 \in R$.
2. *Comutativo*: Para todo $r_1, r_2 \in R$, tem-se que $r_1 \oplus r_2 = r_2 \oplus r_1$.
3. *Associativo*: Para todo $r_1, r_2, r_3 \in R$, tem-se que $r_1 \oplus (r_2 \oplus r_3) = (r_1 \oplus r_2) \oplus r_3$.

Seja $S \subseteq R$ uma coleção finita de elementos de R, podendo haver elementos repetidos. Reduzir S com respeito à $\oplus$ é operar:

$$\bigoplus_{s \in S} s = s_1 \oplus s_2 \oplus \cdots \oplus s_{|S|}$$

Como $\oplus$ é associativo e comutativo, podemos agrupar os elementos de S em qualquer ordem e aplicar tal operação, e portanto podemos distribuir a carga de trabalho de forma eficiente entre as diversas máquinas de uma rede, por exemplo.

2.2 Neste EP

Seja $\mathbb{Z}^{3\times3}$ o conjunto das matrizes 3×3 com entradas inteiras. Sejam $A, B \in \mathbb{Z}^{3\times3}$ e denote a_{ij} como o elemento correspondente a i-ésima linha e j-ésima coluna de A, e analogamente b_{ij} para B. Definiremos o operador $\oplus$ sobre $\mathbb{Z}^{3\times3}$ da seguinte forma:

$$A \oplus B = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \oplus \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} \min\{a_{11}, b_{11}\} & \min\{a_{12}, b_{12}\} & \min\{a_{13}, b_{13}\} \\ \min\{a_{21}, b_{21}\} & \min\{a_{22}, b_{22}\} & \min\{a_{23}, b_{23}\} \\ \min\{a_{31}, b_{31}\} & \min\{a_{32}, b_{32}\} & \min\{a_{33}, b_{33}\} \end{bmatrix}$$

Note que $\oplus$ é um operador de redução (você pode inclusive provar isto ☺). Aqui trabalharemos com $S \subset \mathbb{Z}^{3\times3}$, ou seja, uma coleção finita de elementos de $\mathbb{Z}^{3\times3}$, e você deverá escrever pelo menos um kernel na GPU que reduz S com respeito à $\oplus$.

3 Restrições

1. Este EP deve ser desenvolvido em CUDA C/C++, e a redução de S com respeito à $\oplus$ deve ser implementada em pelo menos um kernel na GPU.
2. O código deve ser compatível com o `nvcc` usando como compilador de Host o `g++-4.9`, `g++-5`, ou o `clang-3.8`.
3. O código desenvolvido deve ser compatível com as placas de compute capability 3.0 para acima.
4. Os elementos das matrizes devem ser inteiros de pelo menos 32-bits. Não testaremos com valores que cause overflow em inteiros de 32-bits.
5. Toda a coleção S de matrizes cabe na memória principal do Device (placa de vídeo).
6. O Host contém mais memória que o Device.

4 Entrada/Saída

Seu programa deverá receber um parâmetro como argumento: o caminho para um arquivo contendo todas os elementos de $S \subset \mathbb{Z}^{3\times3}$, ou seja

```
./main <caminho_lista_matrizes>
```

O formato do arquivo deverá seguir:

```
quantidade_de_matrizes
***
a11 a12 a13
a21 a22 a23
a31 a32 a33
***
b11 b12 b13
b21 b22 b23
b31 b32 b33
***
... //muitas matrizes aqui
***
a11 a12 a13
a21 a22 a23
a31 a32 a33
***
```

Exemplo:

```
2
***
1 1 2
3 5 8
13 21 34
***
2 3 5
7 11 13
17 19 23
***
```

Como saída, o seu EP deverá imprimir na tela o resultado de reduzir todas as matrizes lidas com respeito à $\oplus$. No exemplo acima:

```
1 1 2
3 5 8
13 19 23
```

Não é necessário imprimir informações a respeito do tempo de execução.

5 Dicas

- Trabalhe com um número de threads múltiplo de 32 no CUDA. A GPU sempre escalona conjuntos de 32 threads (warps) e isto evita possíveis complicações se optarem por usar as instruções de `shfl`.
- Não é necessário usar as instruções de `shfl`, até mesmo porque sua placa pode não ter suporte a *compute capability* necessária.
- Lembre-se que a flag `-Xptxas --opt-level=3` habilita a otimização de código na GPU, e que `-arch sm_30` habilita os recursos da compute capability 3.0.
- O `cuda-gdb` pode ser uma ferramenta de grande ajuda. Use a flag `-g -G` no `nvcc` para habilitar o modo de debugging.
- Pense em como calcular $\min\{a, b\}$ sem causar divergência de branch.
- Escreva um teste automatizado que compara o resultado obtido na GPU com uma implementação sequencial do mesmo algoritmo.

6 Entrega

Deverá ser entregue um pacote no sistema PACA com uma pasta com o nome e o sobrenome do estudante que o submeteu no seguinte formato: `nome.sobrenome.zip`. Se o EP for feito em dupla, o formato deve ser `nome1.sobrenome1.nome2.sobrenome2.zip`. Somente um estudante da dupla submeterá a tarefa. Essa pasta deve ser comprimida em formato `ZIP` e deve conter dois itens:

- Os códigos fonte do programa, em conjunto com um `Makefile` que o compila, gerando um executável `main`.
- Um relatório em `.txt` ou `.pdf` explicando a solução implementada. Além disto, ele deve conter um breve discurso sobre a viabilidade de reduzir S com respeito à $\oplus$ na GPU utilizada.

Em caso de dúvidas, use o fórum de discussão do Paca. A data de entrega deste Exercício Programa é até às **23:55 horas da Segunda-Feira, 11 de Junho**.