

# EECS201000

## Introduction to Programming Laboratory

### Homework 4: Blocked All-Pairs Shortest Path

**Due: Aug 3, 2017, 8AM**

## 1 GOAL

---

This assignment helps you get familiar with CUDA on multi-GPU environment by implementing a blocked all-pairs shortest path algorithm. We encourage you to optimize your program by exploring different optimizing strategies for optimization points.

## 2 PROBLEM DESCRIPTION

---

In this assignment, you are asked to modify sequential Floyd-Warshall algorithm to a parallelized CUDA version which take advantages of multiple GPUs.

Given an  $N \times N$  matrix  $W = [w(i, j)]$  where  $w(i, j) \geq 0$  represents the distance (weight of the edge) from a vertex  $i$  to a vertex  $j$  in a **simple directed graph** with  $N$  vertices. We define an  $N \times N$  matrix  $D = [d(i, j)]$  where  $d(i, j)$  denotes the shortest-path distance from a vertex  $i$  to a vertex  $j$ . Let  $D^{(k)} = [d^{(k)}(i, j)]$  be the result which all the intermediate vertices are in the set  $\{1, 2, \dots, k\}$ .

We define  $d^{(k)}(i, j)$  as follows:

$$d^{(k)}(i, j) = \begin{cases} w(i, j) & \text{if } k = 0; \\ \min \left( d^{(k-1)}(i, j), d^{(k-1)}(i, k) + d^{(k-1)}(k, j) \right) & \text{if } k \geq 1. \end{cases}$$

The matrix  $D^{(N)} = d^{(N)}(i, j)$  gives the answer to the APSP problem.

In the blocked APSP algorithm, we partition  $D$  into  $[N/B] \times [N/B]$  blocks of  $B \times B$  submatrices. The number  $B$  is called **blocking factor**. For instance, we divide a  $6 \times 6$  matrix into  $3 \times 3$  submatrices (or blocks) by  $B = 2$ .

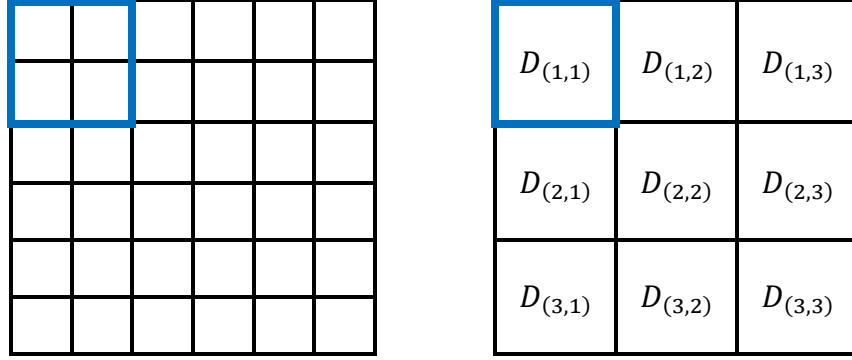


Figure 1: Divide a matrix by  $B = 2$

Blocked version of Floyd-Warshall algorithm will perform  $\lceil N/B \rceil$  rounds, and each round is divided into 3 phases. It performs  $B$  iterations in each phase.

Assumes a block is identified by its index  $(I, J)$ , where  $1 \leq I, J \leq \lceil N/B \rceil$ . The block with index  $(I, J)$  is denoted by  $D_{(I,J)}^{(k)}$ .

In the following explanation, we assume  $N = 6$  and  $B = 2$ . The execution flow is described step by step as follows:

- **Phase 1: Self-dependent blocks**

In the  $K$ -th iteration, the 1<sup>st</sup> phase is to compute  $B \times B$  pivot block  $D_{(K,K)}^{(K \times B)}$ .

For instance, in the 1<sup>st</sup> iteration,  $D_{1,1}^{(2)}$  is computed as follows:

$$\begin{aligned}
 d^{(1)}(1,1) &= \min \left( d^{(0)}(1,1), d^{(0)}(1,1) + d^{(0)}(1,1) \right) \\
 d^{(1)}(1,2) &= \min \left( d^{(0)}(1,2), d^{(0)}(1,1) + d^{(0)}(1,2) \right) \\
 d^{(1)}(2,1) &= \min \left( d^{(0)}(2,1), d^{(0)}(2,1) + d^{(0)}(1,1) \right) \\
 d^{(1)}(2,2) &= \min \left( d^{(0)}(2,2), d^{(0)}(2,1) + d^{(0)}(1,2) \right) \\
 d^{(2)}(1,1) &= \min \left( d^{(1)}(1,1), d^{(1)}(1,2) + d^{(1)}(2,1) \right) \\
 d^{(2)}(1,2) &= \min \left( d^{(1)}(1,2), d^{(1)}(1,2) + d^{(1)}(2,2) \right) \\
 d^{(2)}(2,1) &= \min \left( d^{(1)}(2,1), d^{(1)}(2,2) + d^{(1)}(2,1) \right) \\
 d^{(2)}(2,2) &= \min \left( d^{(1)}(2,2), d^{(1)}(2,2) + d^{(1)}(2,2) \right)
 \end{aligned}$$

Note that result of  $d^{(2)}$  depends on the result of  $d^{(1)}$  and therefore cannot be computed in parallel with the computation of  $d^{(1)}$ .

- **Phase 2: Pivot-row and pivot-column blocks**

In the  $K$ -th iteration, it computes all  $D_{(h,K)}^{(K \times B)}$  and  $D_{(K,h)}^{(K \times B)}$  where  $h \neq K$ .

The result of pivot-row/pivot-column blocks depend on the result in Phase 1 and itself

For instance, in the 1<sup>st</sup> iteration, the result of  $D_{(1,3)}^{(2)}$  depends on  $D_{(1,1)}^{(2)}$  and  $D_{(1,3)}^{(0)}$ :

$$d^{(1)}(1,5) = \min \left( d^{(0)}(1,5), d^{(2)}(1,1) + d^{(0)}(1,5) \right)$$

$$d^{(1)}(1,6) = \min \left( d^{(0)}(1,6), d^{(2)}(1,1) + d^{(0)}(1,6) \right)$$

$$d^{(1)}(2,5) = \min \left( d^{(0)}(2,5), d^{(2)}(2,1) + d^{(0)}(1,5) \right)$$

$$d^{(1)}(2,6) = \min \left( d^{(0)}(2,6), d^{(2)}(2,1) + d^{(0)}(1,6) \right)$$

$$d^{(2)}(1,5) = \min \left( d^{(1)}(1,5), d^{(2)}(1,2) + d^{(1)}(2,5) \right)$$

$$d^{(2)}(1,6) = \min \left( d^{(1)}(1,6), d^{(2)}(1,2) + d^{(1)}(2,6) \right)$$

$$d^{(2)}(2,5) = \min \left( d^{(1)}(2,5), d^{(2)}(2,2) + d^{(1)}(2,5) \right)$$

$$d^{(2)}(2,6) = \min \left( d^{(1)}(2,6), d^{(2)}(2,2) + d^{(1)}(2,6) \right)$$

- **Phase 3: Other blocks**

In the  $K$ -th iteration, it computes all  $D_{(h_1,h_2)}^{(K \times B)}$  where  $h_1, h_2 \neq K$ .

The result of these blocks depend on the result in Phase 2 and itself.

For instance, in the 1<sup>st</sup> iteration, the result of  $D_{(2,3)}^{(2)}$  depends on  $D_{(2,1)}^{(2)}$  and  $D_{(1,3)}^{(2)}$ :

$$d^{(1)}(3,5) = \min \left( d^{(0)}(3,5), d^{(2)}(3,1) + d^{(2)}(1,5) \right)$$

$$d^{(1)}(3,6) = \min \left( d^{(0)}(3,6), d^{(2)}(3,1) + d^{(2)}(1,6) \right)$$

$$d^{(1)}(4,5) = \min \left( d^{(0)}(4,5), d^{(2)}(4,1) + d^{(2)}(1,5) \right)$$

$$d^{(1)}(4,6) = \min \left( d^{(0)}(4,6), d^{(2)}(4,1) + d^{(2)}(1,6) \right)$$

$$d^{(2)}(3,5) = \min \left( d^{(1)}(3,5), d^{(2)}(3,2) + d^{(2)}(2,5) \right)$$

$$d^{(2)}(3,6) = \min \left( d^{(1)}(3,6), d^{(2)}(3,2) + d^{(2)}(2,6) \right)$$

$$d^{(2)}(4,5) = \min \left( d^{(1)}(4,5), d^{(2)}(4,2) + d^{(2)}(2,5) \right)$$

$$d^{(2)}(4,6) = \min \left( d^{(1)}(4,6), d^{(2)}(4,2) + d^{(2)}(2,6) \right)$$

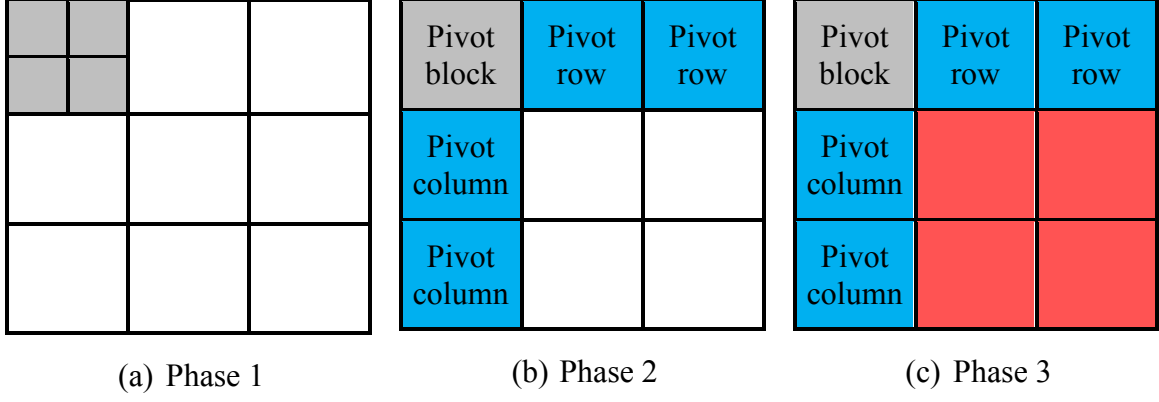


Figure 2: The 3 phases of blocked FW algorithm in the 1<sup>st</sup> iteration

The computations of  $D_{(1,3)}^{(2)}$ ,  $D_{(2,3)}^{(2)}$  and its dependencies are illustrated in Figure 3.

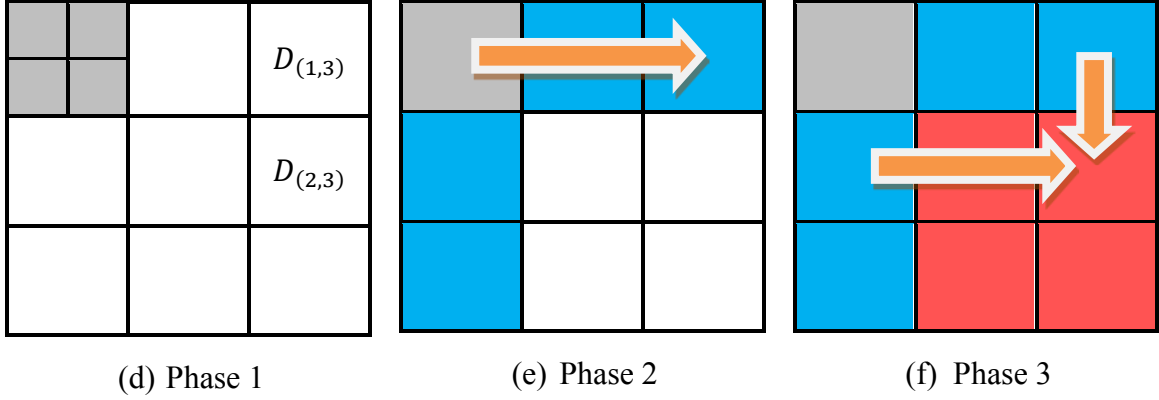


Figure 3: Dependencies of  $D_{(1,3)}^{(2)}$ ,  $D_{(2,3)}^{(2)}$  in the 1<sup>st</sup> iteration

In this particular example where  $N = 6$  and  $B = 2$ , we will require  $\lceil N/B \rceil = 3$  rounds.

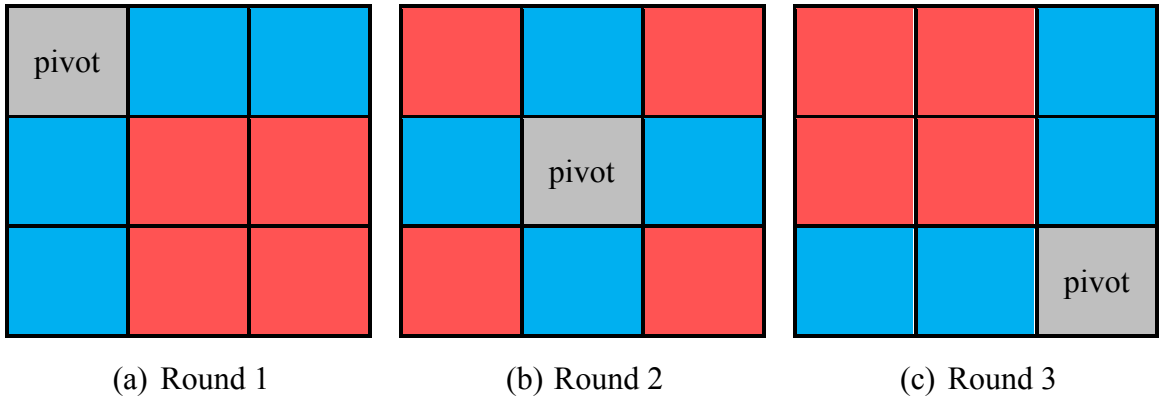


Figure 4: Blocked FW algorithm in each iteration

### 3 INPUT / OUTPUT FORMAT

---

1. Your program is required to read an input file, and generate output in another file.
2. Your program accepts 2 input parameters. They are:

- i 、 (String)      the input file name
- ii 、 (String)      the output file name
- iii 、 (Integer)    the blocking factor

Make sure users can assign test cases through command line. For instance:

```
$ ./executable in_file out_file 32
```

TAs will judge your program as follows:

```
$ diff -b out_file answer
```

3. The 1<sup>st</sup> line of an input test case consists of 2 integers  $N$  ( $1 \leq N \leq 10000$ ) and  $M$  ( $0 \leq M \leq 10^9$ ) separated by a single space, which represents number of vertices and number of edge weight assignments respectively.

Each of the following  $M$  lines consists of 3 integers  $i, j$  and  $W$  ( $i \neq j$ ), separated by a single space between any two numbers.

- $i$  represents the index of the source vertex ( $1 \leq i \leq N$ )
- $j$  represents the index of the destination vertex ( $1 \leq j \leq N$ )
- $W$  represents the distance (weight of edge) from vertex  $i$  to vertex  $j$  ( $0 \leq W \leq 100$ )

Edges which are not listed in the input file do not exist in the graph. That is, for all  $i \neq j$ , if edge( $i, j$ ) does not show up in the input at all, vertex  $i$  does not have an edge to vertex  $j$ . But since we are dealing with a **directed** graph, this does NOT imply that edge( $j, i$ ) is also non-existent.

Besides, if there are re-assignments of an edge, please follow the latest one.

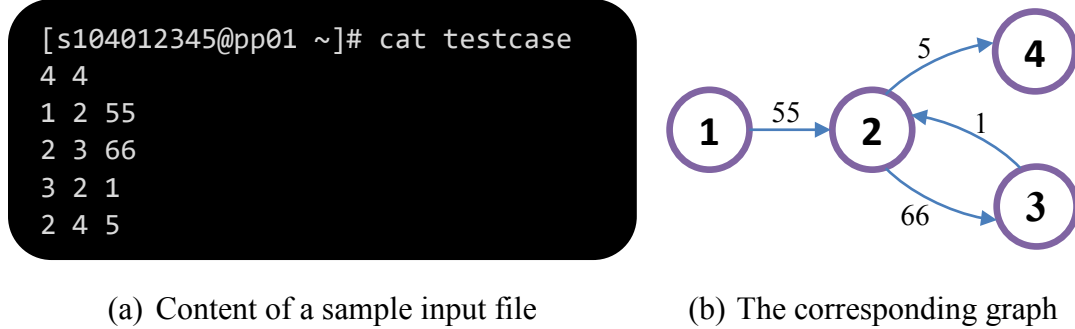


Figure 5: Sample Input

4. For output file, list the shortest-path distance of all vertex pairs.

Assume  $N$  represents total number of vertices, the output file should consists of  $N$  lines, each line consists of  $N$  numbers and separate them by a single space.

The number at the  $i^{\text{th}}$  line and the  $j^{\text{th}}$  column is the shortest-path distance from the  $i^{\text{th}}$  vertex to the  $j^{\text{th}}$  vertex if there is a path; otherwise, the corresponding output should be **INF**.

```
[s104012345@pp01 ~]# cat output
0 55 121 60
INF 0 66 5
INF 1 0 6
INF INF INF 0
```

Figure 6: Sample output

The sample test cases are provided in **/home/ipl2017/shared/hw4** on pp31.

## 4 WORKING ITEMS

---

You are required to implement **2** versions of blocked Floyd-Warshall algorithm under the given restrictions.

### 1. Single GPU

- Implement blocked APSP algorithm as described in Section 2.
- The main algorithm should be implemented in CUDA C/C++ kernel functions.
- Achieve better performance than sequential Floyd-Warshall implementation.

## 2. Multi GPUs implementation with OpenMP

- The restrictions of single-GPU version still hold.
- Able to utilize multiple GPUs available on single node.
- Achieve better performance than single GPU version.

## 3. Makefile

Please refer to the example in [/home/ipl2017/shared/hw4](#) on **apolloGPU**.

Don't modify execution file name(HW4\_cuda.exe, HW4\_openmp.exe) in sample Makefile.

## 4. README

You should specify your best block factor in README, TAs will use this configuration to test your performance.

Please refer to the example in [/home/ipl2017/shared/hw4](#) on **apolloGPU**.

# 5 OPTIMIZATION HINTS

---

- Shared memory
- Streaming
- Resolve bank conflicts
- Dynamic load-balancing (for openMP and MPI)

# 6 GRADING

---

## 1. Correctness (70%)

i 、 [50%] Single-GPU

ii 、 [20%] Multi-GPU implementation with OpenMP

## 2. Performance (20%)

- Performance is measured by the execution time of your program using 'time' Linux command.
- Points are giving according to the performance ranking of your program among all the students.

## 3. Demo (10%)

- Each student is given 10 minutes to explain your implementation followed by some questions from TA.
- No debugging or code modification is allowed during the demo.

- Points are given according to your understanding and explanation of your code, and your answers of the TA questions.

## 7 REMINDER

---

1. Please upload the following files to **HW\_submission/HW4** directory on **apolloGPU** under your home directory before **8/3 8:00AM** (**The folder will be locked after deadline**)

i 、 **HW4\_{student-ID}\_cuda.cu**

ii 、 **HW4\_{student-ID}\_openmp.cu**

iii 、 **Makefile**

iv 、 **README**

Make sure your compile script can execute correctly and your code has no compile error before you upload your homework.

2. We provide sample code `seq_FW.cpp` and `block_FW.cpp` in **/home/ipl2017/shared/hw4** on **apolloGPU**.
3. Since we have limited resources for you guys to use, please start your work ASAP. Do not leave it until the last day!
4. **0 will be given to cheater** (even copying code from the Internet), but discussion on code is encouraged.
5. Asking questions through iLMS or by email are welcomed!