Homework 3: All-Pairs Shortest Path

Deadline: Aug 1, 2019, 11:59 am

Contents

Goal

1. Understand the all-pairs shortest path problem.
2. Get familiar with CUDA by implementing the blocked Floyd-Warshall algorithm.

Requirements

In this assignment, you are asked to implement two versions of all-pairs shortest path.

hw3a. **Thread version**. You may use pthread or std::thread. The code should run on `apollo`. You can use any algorithm for shortest path.

hw3b. **CUDA version**. The code should run on `hades`. You should use the blocked Floyd-Warshall algorithm.

Both versions follow the same input & output format.

Command line specification

For `hw3a` on `apollo`:

```
srun -N1 -n1 -c12 ./hw3a inputfile outputfile
```

For `hw3b` on `hades`:

```
srun --gres=gpu:1 -pipl ./hw3b inputfile outputfile
```

Where `inputfile` and `outputfile` are the paths to the corresponding files. The format of the input/output files are specified below.

Input specification

- The input is a binary file containing a directed graph with non-negative edge distances.
- The first two integers mean *number of vertices (V)* and *number of edges (E)*.
- After that, there are the list of edges. Each edge contains three integers: *source vertex id* (src_i) , *destination vertex id* (dst_i), *edge weight* (w_i).
- All values are non-negative integers. The values of **vertex indexes start at 0**.
- The ranges for the input is:

- $2 \leq V \leq 10000$ for correctness
- $2 \leq V \leq 45000$ for performance testing
- $0 \leq E \leq V \times (V-1)$
- $0 \leq \text{src}_i, \text{dst}_i < V$
- $\text{src}_i \neq \text{dst}_i$
- if $\text{src}_i = \text{src}_j$ then $\text{dst}_i \neq \text{dst}_j$ (there will not be repeated edges)
- $0 \leq w_i < 1000$

Here's an example:

offset	type	decimal value	description
0000	32-bit integer	3	# vertices (V)
0004	32-bit integer	6	# edges (E)
0008	32-bit integer	0	src id for edge 0
0012	32-bit integer	1	dst id for edge 0
0016	32-bit integer	3	edge 0's distance
0020	32-bit integer		src id for edge 1
…	…	…	…
0076	32-bit integer		edge 5's distance

Output specification

- The output file is also in binary format.
- For an input file with V vertices, you should output an output file containing V^2 integers.
- The first V integers should be the shortest path distances for starting from edge 0: $\text{dist}(0,0), \text{dist}(0,1), \text{dist}(0,2), \ldots, \text{dist}(0,V-1)$; then the following V integers would be the shortest path distances starting from edge 1: $\text{dist}(1,0), \text{dist}(1,1), \text{dist}(1,2), \ldots, \text{dist}(1,V-1)$; and so on, totaling V^2 integers.
- $\text{dist}(i,j) = 0$ where $i = j$.
- If there is no valid path between i→j, please output with: $\text{dist}(i,j) = 2^{30} - 1 = 1073741823$.

Example output file:

offset	type	decimal value	description
0000	32-bit integer	0	dist(0, 0)
0004	32-bit integer	?	dist(0, 1)
0008	32-bit integer	?	dist(0, 2)
…	…	…	…
$4V^2$-8	32-bit integer	?	min dist(V-1, V-2)
$4V^2$-4	32-bit integer	0	min dist(V-1, V-1)

Grading

1. hw3a (30%)

   An unknown number of test cases will be used to test your implementation. You get 30 points for `hw3a` if you passed all the test cases, $\max(0, 30-2k)$ points if there are k failed test cases. There **will** be hidden testcases.

   For each test case, you pass it if:

- The answer is correct.
- The execution time of your implementation is *shorter* than $T_{\text{seq}} \div 6 + 10$ seconds, where T_{seq} is the execution time of TA's sequential code.

2. hw3b correctness (30%)

   An unknown number of test cases will be used to test your implementation. You get 30 points for `hw3a` if you passed all the test cases, $\max(0, 30 - 2k)$ points if there are k failed test cases. There **will** be hidden testcases.

   For each test case, you pass it if:

   - The answer is correct.
   - The execution time of your implementation is *shorter* than 30 seconds.

   Note that for the purpose for correctness testing, $V \leq 10000$.
3. hw3b performance (30%)

   Points are given according to the largest test case (largest V) you can complete within 30 seconds.

3. Demo (10%)

   - Each student is given 7 minutes to explain the implementation followed by some questions from TA.
   - Points are given according to your understanding and explanation of your code, and your answers of the TA questions.

Submission¶

Put these source code under at `~/homework/hw3` in `apollo31`, then run `hw3a-judge`:

- Your Makefile for hw3a – `~/homework/hw3/Makefile`
- Source code for thread version – `~/homework/hw3/hw3a.cc` or `~/homework/hw2/hw3a.c`

Put these source code under at `~/homework/hw3` in `hades01`, then run `hw3b-judge`:

- Your Makefile for hw3b – `~/homework/hw3/Makefile`
- Source code for the CUDA version – `~/homework/hw3/hw3b.cu`

`make hw3a` and `make hw3b` will be used to compile your code.

Use `hw3a-judge` and `hw3b-judge` to check and submit your code **before the deadline**. You can submit **as many times as you want**. You need to answer `y` when judging to update your submitted code.

Resources

- Sequential Blocked FW source code – `hades:/home/ipl19/y/hw3/seq.cc`
- Recommended compiler flags
  - hw3a: `g?? -O3 -pthread -std=c++11 -Wall`
  - hw3b: `nvcc -O3 -std=c++11 -Xptxas=-v -arch=sm_61`

  No example `Makefile` for you this time!
- Scoreboard location: http://ipl.cs.nthu.edu.tw/s/hw3
- Correctness Testcases: `/home/ipl19/y/hw3/c`
- Performance Testcases: `/home/ipl19/y/hw3/p`, only on `hades`
- Use the `hw3-cat` command to view the binary test cases in text format.

Blocked Floyd-Warshall algorithm

Given an $V \times V$ 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 directed graph with V vertices. We define an $V \times V$ 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 $\{1, 2, \ldots, k\}$.

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

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

The matrix $D^{(V)} = d^{(V)}(i,j)$ gives the answer to the all-pairs shortest path problem.

In the blocked all-pairs shortest path algorithm, we partition D into $\lceil V/B \rceil \times \lceil V/B \rceil$ blocks of $B \times B$ submatrices. The number B is called the *blocking factor*. For instance, in figure 1, we divide a 6×6 matrix into 3×3 submatrices (or blocks) by $B = 2$.

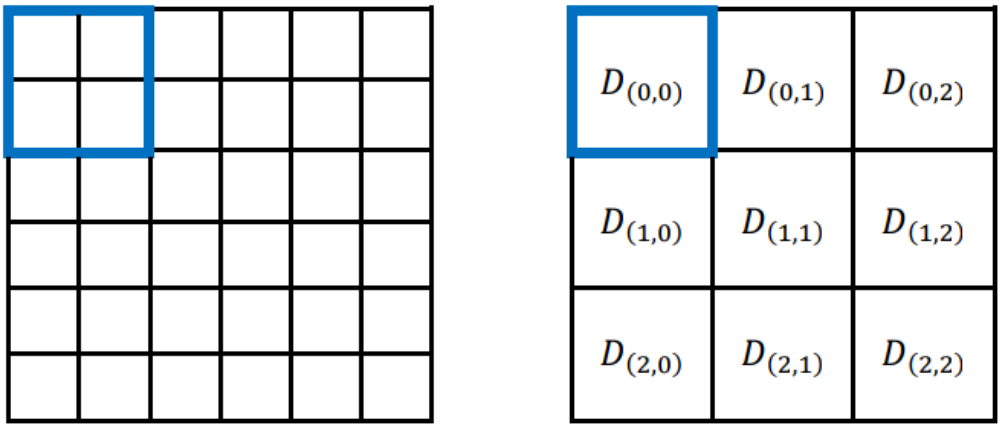

Figure 1: Divide a matrix by B = 2

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

Assuming a block is identified by its index (I, J), where $0 \leq I, J < \lceil V/B \rceil$. The block with index (I, J) is denoted by $D^{(k)}_{(I,J)}$.

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 first phase is to compute the $B \times B$ pivot block $D^{(k \cdot B)}_{(k-1,k-1)}$.

  For instance, in the 1st iteration, $D^{(2)}_{(0,0)}$ is computed as follows:

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

Note that the 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 $D^{(k \cdot B)}_{(h,k)}$ and $D^{(k \cdot B)}_{(k,h)}$ 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 1st iteration, the result of $D^{(2)}_{(0,2)}$ depends on $D^{(2)}_{(0,0)}$ and $D^{(0)}_{(0,2)}$:

$$
\begin{aligned}
d^{(1)}(0,4) &= \min(d^{(0)}(0,4), d^{(2)}(0,0) + d^{(0)}(0,4)) \\
d^{(1)}(0,5) &= \min(d^{(0)}(0,5), d^{(2)}(0,0) + d^{(0)}(0,5)) \\
d^{(1)}(1,4) &= \min(d^{(0)}(1,4), d^{(2)}(1,0) + d^{(0)}(0,4)) \\
d^{(1)}(1,5) &= \min(d^{(0)}(1,5), d^{(2)}(1,0) + d^{(0)}(0,5)) \\
d^{(2)}(0,4) &= \min(d^{(1)}(0,4), d^{(2)}(0,1) + d^{(1)}(1,4)) \\
d^{(2)}(0,5) &= \min(d^{(1)}(0,5), d^{(2)}(0,1) + d^{(1)}(1,5)) \\
d^{(2)}(1,4) &= \min(d^{(1)}(1,4), d^{(2)}(1,1) + d^{(1)}(1,4)) \\
d^{(2)}(1,5) &= \min(d^{(1)}(1,5), d^{(2)}(1,1) + d^{(1)}(1,5))
\end{aligned}
$$

- **Phase 3**: other blocks.

  In the k-th iteration, it computes all $D^{(k \cdot B)}_{(h_1,h_2)}$ where $h_1, h_2 \neq k$.

  The result of these blocks depends on the result from phase 2 and itself.

  For instance, in the 1st iteration, the result of $D^{(2)}_{(1,2)}$ depends on $D^{(2)}_{(1,0)}$ and $D^{(2)}_{(0,2)}$:

$$
\begin{aligned}
d^{(1)}(2,4) &= \min(d^{(0)}(2,4), d^{(2)}(2,0) + d^{(2)}(0,4)) \\
d^{(1)}(2,5) &= \min(d^{(0)}(2,5), d^{(2)}(2,0) + d^{(2)}(0,5)) \\
d^{(1)}(3,4) &= \min(d^{(0)}(3,4), d^{(2)}(3,0) + d^{(2)}(0,4)) \\
d^{(1)}(3,5) &= \min(d^{(0)}(3,5), d^{(2)}(3,0) + d^{(2)}(0,5)) \\
d^{(2)}(2,4) &= \min(d^{(1)}(2,4), d^{(2)}(2,1) + d^{(2)}(1,4)) \\
d^{(2)}(2,5) &= \min(d^{(1)}(2,5), d^{(2)}(2,1) + d^{(2)}(1,5)) \\
d^{(2)}(3,4) &= \min(d^{(1)}(3,4), d^{(2)}(3,1) + d^{(2)}(1,4)) \\
d^{(2)}(3,5) &= \min(d^{(1)}(3,5), d^{(2)}(3,1) + d^{(2)}(1,5))
\end{aligned}
$$

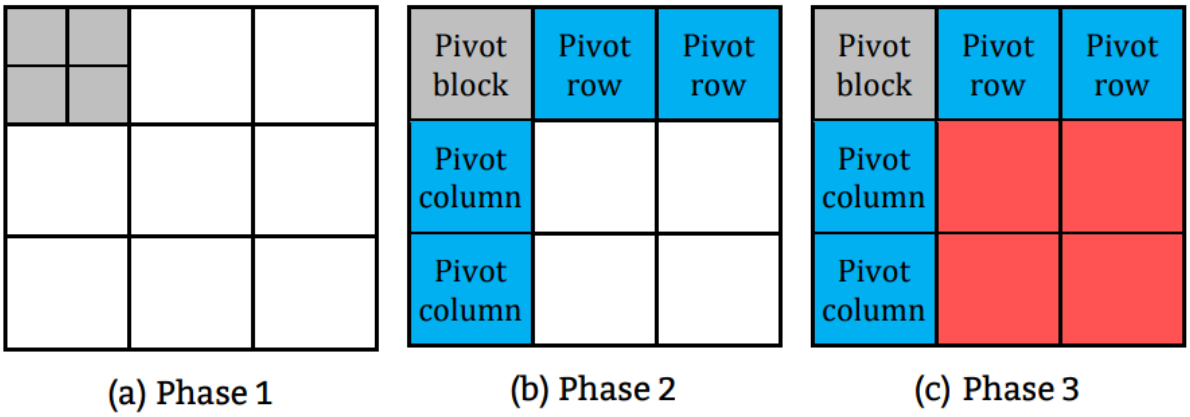

Figure 2: The 3 phases of blocked FW algorithm in the first 1st iteration.

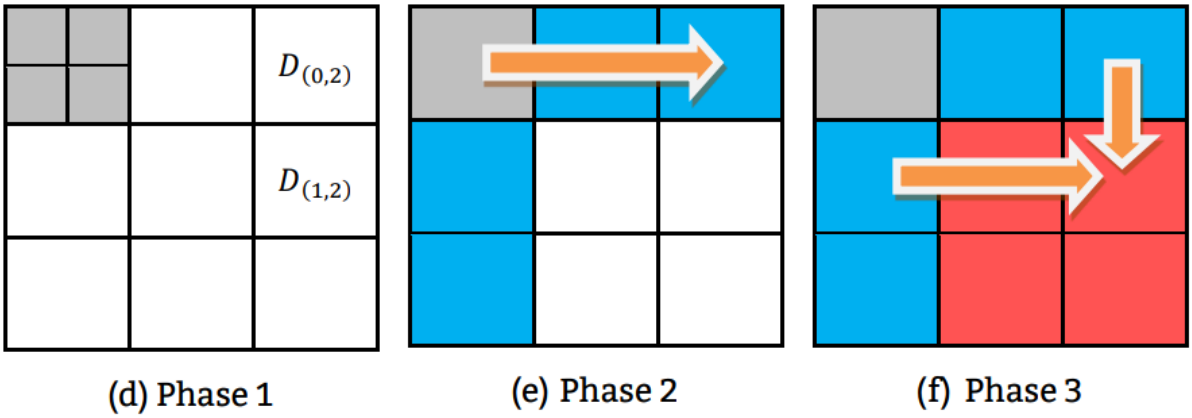

Figure 3: The computations of $D^{(2)}_{(0,2)}$, $D^{(2)}_{(1,2)}$ and their dependeicies in the first iteration.

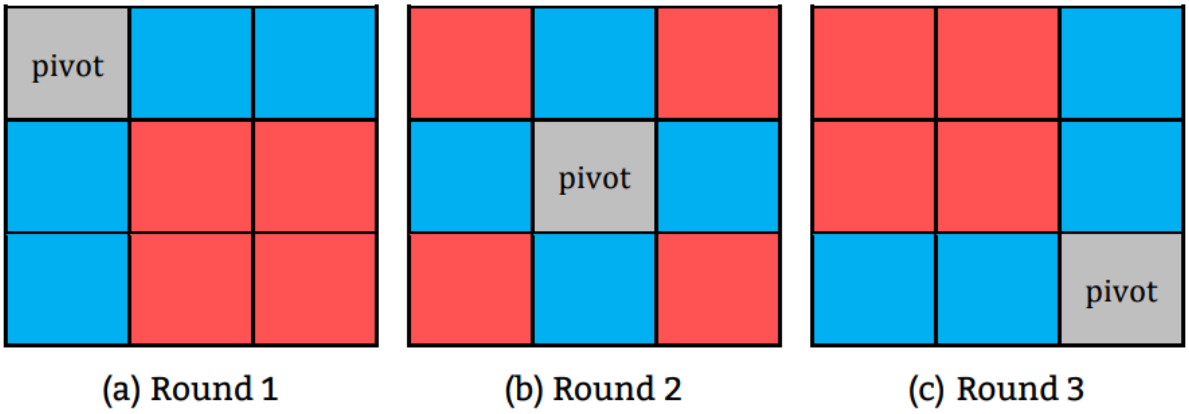

Figure 4: In this particular example where $V = 6$ and $B = 2$, we will require $\lceil V/B \rceil = 3$ rounds.

Final Notes

Contact TA via afg984@gmail.com or iLMS immediately if you find any problems with the homework specification, judge scripts, example source code or the test cases.

You are allowed to discuss and exchange ideas with others, but you are required to write the code on your own. You'll got 0 points if we found you cheating.