CSE531 - Programming Lab 3: CUDA

Due: April 11, 22:00.

In this assignment, we are going to implement a blocked all-pair shortest path (APSP) algorithm in CUDA.

Problem Description

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 undirected 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 in which all the intermediate vertices are in the set $\{1,2,\cdots,k\}$.

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

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

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

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

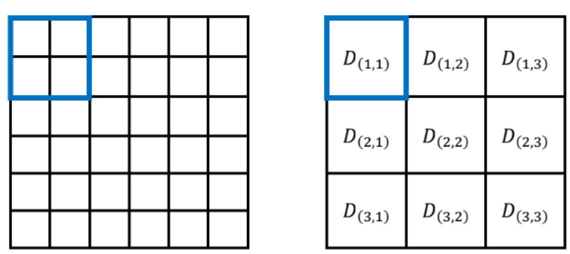


Figure 1: Divide a matrix by $B = 2$

The blocked 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. Assume 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 1st phase is to compute $B \times B$ pivot block $D_{(K,K)}^{(K \times B)}$. For instance, in the 1st iteration, $D_{(1,1)}^{(2)}$ is computed as follows:

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

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

  Note that the result of $d^{(2)}$ depends on the result of $d^{(1)}$ and thus 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 1st iteration, the result of $D_{(1,3)}^{(2)}$ depends on $D_{(1,1)}^{(2)}$ and $D_{(1,3)}^{(0)}$:

$$
\begin{aligned}
d^{(1)}(1,5) &= min(d^{(0)}(1,5), d^{(2)}(1,1) + d^{(0)}(1,5)) \\
d^{(1)}(1,6) &= min(d^{(0)}(1,6), d^{(2)}(1,1) + d^{(0)}(1,6)) \\
d^{(1)}(2,5) &= min(d^{(0)}(2,5), d^{(2)}(2,1) + d^{(0)}(1,5)) \\
d^{(1)}(2,6) &= min(d^{(0)}(2,6), d^{(2)}(2,1) + d^{(0)}(1,6))
\end{aligned}
$$

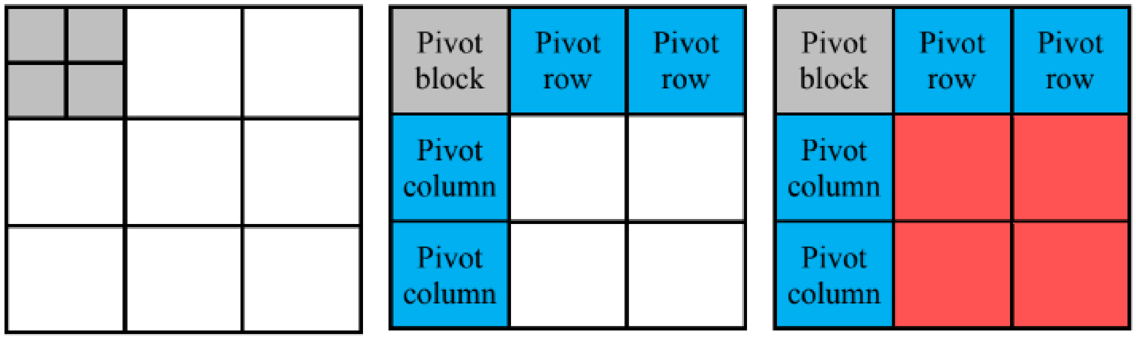
$$
\begin{aligned}
d^{(2)}(1,5) &= min\left(d^{(1)}(1,5), d^{(2)}(1,2) + d^{(1)}(2,5)\right) \\
d^{(2)}(1,6) &= min\left(d^{(1)}(1,6), d^{(2)}(1,2) + d^{(1)}(2,6)\right) \\
d^{(2)}(2,5) &= min\left(d^{(1)}(2,5), d^{(2)}(2,2) + d^{(1)}(2,5)\right) \\
d^{(2)}(2,5) &= min\left(d^{(1)}(2,6), d^{(2)}(2,2) + d^{(1)}(2,6)\right)
\end{aligned}
$$

- **Phase 3**: Other blocks

  In the K-th iteration, it computes all $D^{(K\times B)}_{(h_1,h_2)}$ 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 1st iteration, the result of $D^{(2)}_{(2,3)}$ depends on $D^{(2)}_{(2,1)}$ and $D^{(2)}_{(1,3)}$:

  $$d^{(1)}(3,5) = min(d^{(0)}(3,5), d^{(2)}(3,1) + d^{(0)}(1,5))$$
  $$d^{(1)}(3,6) = min(d^{(0)}(3,6), d^{(2)}(3,1) + d^{(0)}(1,6))$$
  $$d^{(1)}(4,5) = min(d^{(0)}(4,5), d^{(2)}(4,1) + d^{(0)}(1,5))$$
  $$d^{(1)}(4,6) = min(d^{(0)}(4,6), d^{(2)}(4,1) + d^{(0)}(1,6))$$

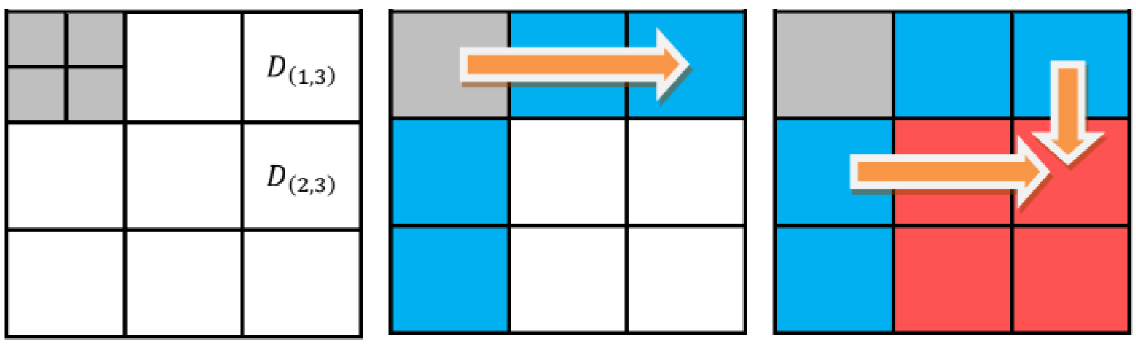
  $$d^{(2)}(3,5) = min\left(d^{(1)}(3,5), d^{(2)}(3,2) + d^{(1)}(2,5)\right)$$
  $$d^{(2)}(3,6) = min\left(d^{(1)}(3,6), d^{(2)}(3,2) + d^{(1)}(2,6)\right)$$
  $$d^{(2)}(4,5) = min\left(d^{(1)}(4,5), d^{(2)}(4,2) + d^{(1)}(2,5)\right)$$
  $$d^{(2)}(4,5) = min\left(d^{(1)}(4,6), d^{(2)}(4,2) + d^{(1)}(2,6)\right)$$



(a) Phase 1 (b) Phase 2 (c) Phase 3

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

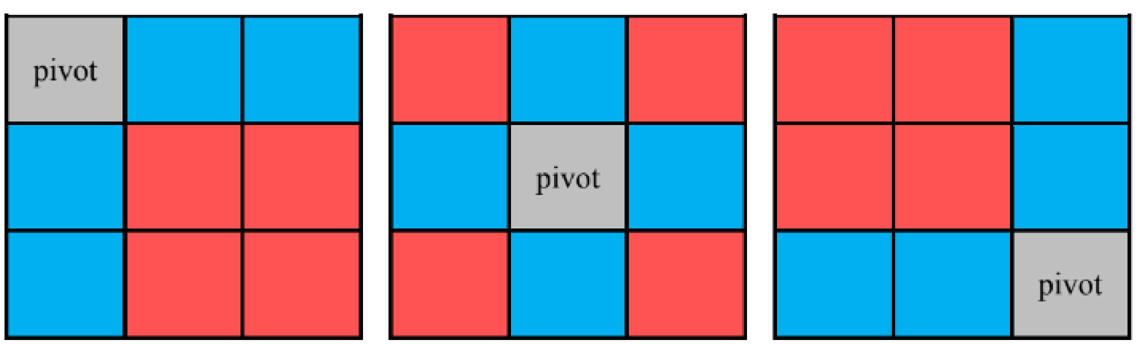
The computations of $D_{(1,3)}^{(2)}, D_{(2,3)}^{(2)}$ and its dependencies are illustrated in Figure 3.



(d) Phase 1 (e) Phase 2 (f) Phase 3

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

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



(a) Round 1 (b) Round 2 (c) Round 3

Figure 4: Blocked FW algorithm in each iteration

Input / Output Format

1. The following files are provided on Canvas: (we reuse the dataset in Lab2)

```
apsp.cu     # TODO: a template for your CUDA implementation
Makefile    # TODO: a template for your Makefile
```

2. The program accepts 3 parameters:

   ```
   ./executable $input_file $output_file $blocking_factor
   ```

```
$ ./apsp 100-4000.in 100-4000.my.out 32
$ diff 100-4000.my.out 100-4000.out
```

3. Input: Same as in Lab 2

4. Output: Same as in Lab 2

Report

The report must contain the following:

1. Title, name, PSU ID
2. Explain your implementations in the following aspects:
   - How do you choose the blocking factor?
   - Efforts you've made in your program. Performance optimization hints: (1) shared memory (2) streaming (3) dynamic load-balancing.
3. Experiment & Analysis
   - System & compiler spec (e.g., e5-cse-135-01 GCC 4.8.5)
   - Show the correctness of **all** datasets.
   - Any other discussions or analyses are encouraged. Make sure to explain how and why you do these experiments.

Rubrics

1. Correctness (50%)
    - 5 datasets (each 10%)
    - Your implementation should output the correct result
    - Your implementation should be faster than the sequential version; otherwise will get 0.
2. Performance (15%): Based on the fastest version using a **Lab135 machine under the 2000-1200000.in dataset** among all students.
3. Report (35%)

Submission

Upload these files to Canvas:
Please do not upload any dataset!
Any corrupted files will be regarded as a failure of submission.

```
Makefile
apsp.cu
Lab3_Report.pdf
```

Reminders

1. Since we have limited resources, please start your work ASAP. Do not leave it until the last day!
2. Copying any codes from the Internet is not allowed, but discussions are encouraged.
3. Office hour holding by Scott: Tuesday 15:00-16:00 @ Westgate Bldg W341