HW 4-1: Blocked All-Pairs Shortest Path

Deadline: 21 Dec 2020, 23:59

Contents

Goal

This assignment helps you get familiar with CUDA by implementing a blocked all-pairs shortest path algorithm. Besides, to measure the performance and scalability of your program, experiments are required. Finally, we encourage you to optimize your program by exploring different optimizing strategies for optimization points.

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 $\{0, 1, 2, \ldots, k-1\}$.

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 $B = 2$.

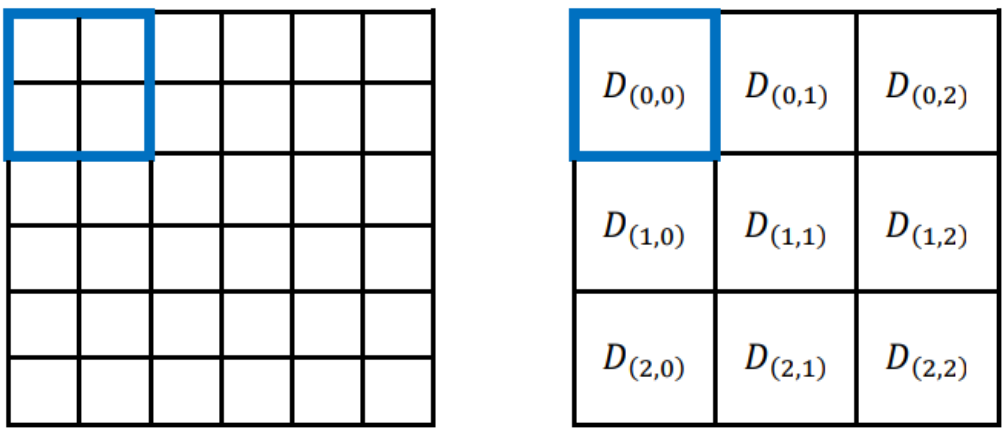

Figure 1: Divide a matrix by B = 2

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 round, 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 round, $D^{(2)}_{(0,0)}$ 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 round, it computes all $D^{(k \cdot B)}_{(h,k-1)}$ and $D^{(k \cdot B)}_{(k-1,h)}$ where $h \neq k-1$.

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

  For instance, in the 1st round, 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 round, it computes all $D^{(k \cdot B)}_{(h_1,h_2)}$ where $h_1, h_2 \neq k-1$.

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

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

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

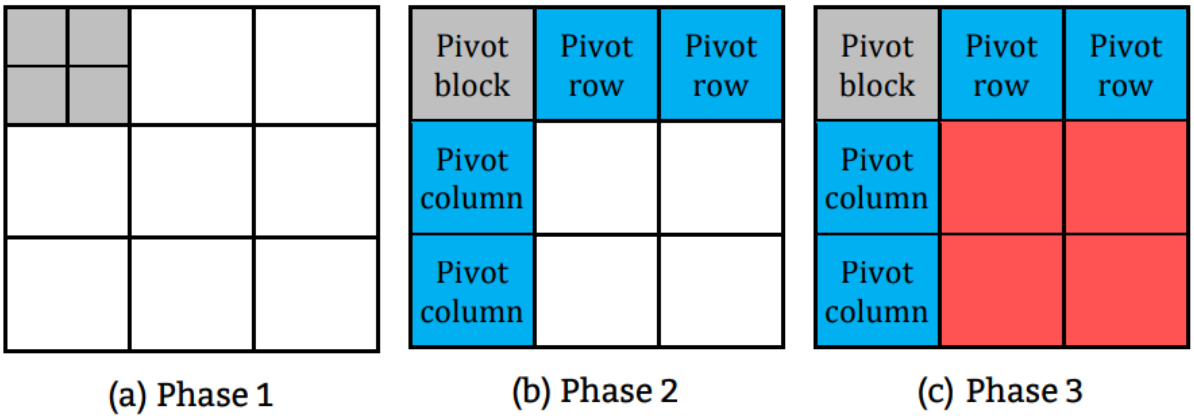

Figure 2: The 3 phases of blocked FW algorithm in the first round.

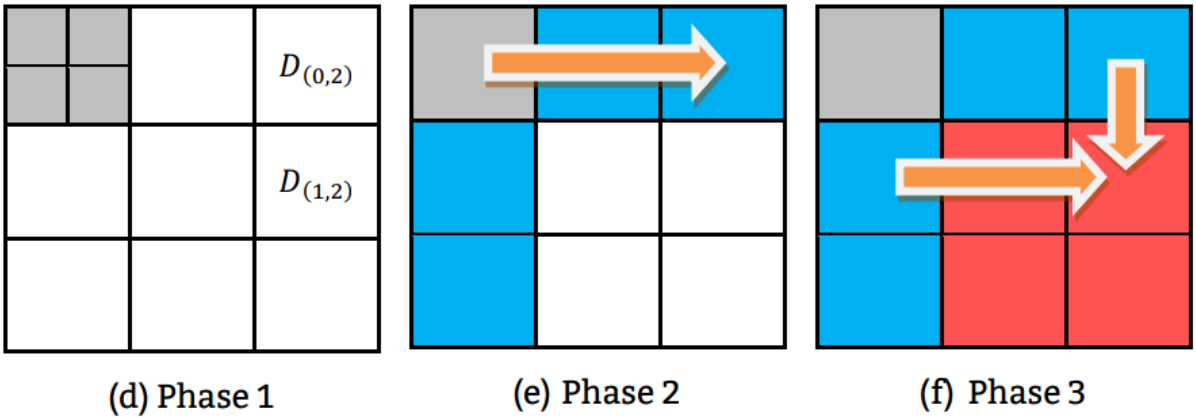

Figure 3: The computations of $D^{(2)}_{(0,2)}$, $D^{(2)}_{(1,2)}$ and their dependeicies in the first round.

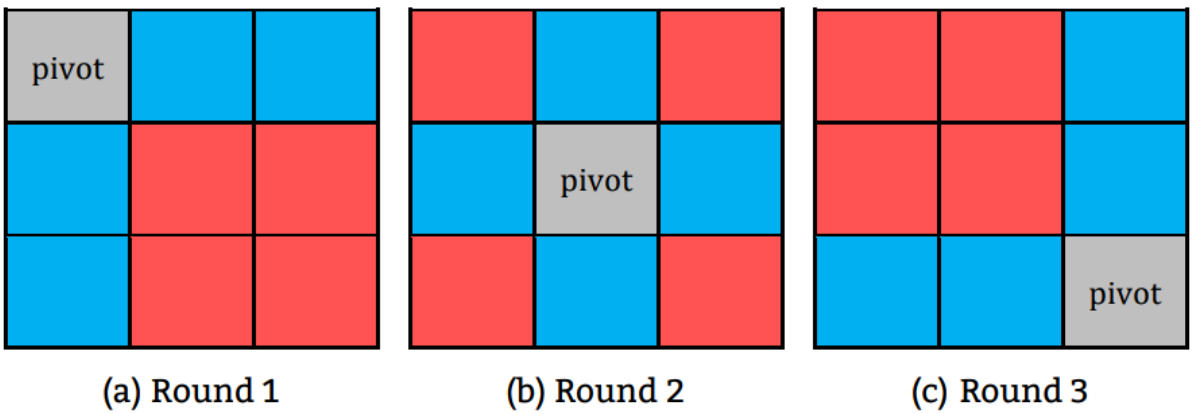

Figure 4: In this particular example where $V = 6$ and $B = 2$, we will require $\lceil V/B \rceil = 3$ rounds.

Input / Output Format

Command line specification

```
srun -N1 -n1 --gres=gpu:1 ./hw4-1 INPUTFILE OUTPUTFILE
```

- `INPUTFILE`: The pathname of the input file. Your program should read the input graph from this file.

- `OUTPUTFILE`: The pathname of the output file. Your program should output the shortest path distances to this file.

Input specification

- The input is a directed graph with non-negative edge distances.
- The input file is a binary file containing 32-bit integers. You can use the `int` type in C/C++.
- The first two integers are *the number of vertices (V)* and *the number of edges (E)*.
- Then, there are E edges. Each edge consists of 3 integers:
  1. *source vertex id* (src_i)
  2. *destination vertex id* (dst_i)
  3. *edge weight* (w_i)
- The values of vertex indexes & edge indexes start at 0.
- The ranges for the input are:
  - $2 \leq V \leq 40000$
  - $0 \leq E \leq V \times (V-1)$
  - $0 \leq \text{src}_i, \text{dst}_i < V$
  - $\text{src}_i \neq \text{dst}_i$
  - if $\text{src}_i = \text{src}_j$ then $\text{dst}_i \neq \text{dst}_j$ (there will not be repeated edges)
  - $0 \leq w_i \leq 1000$
  - No need to consider the condition of signed integer overflow

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	min dist(0, 0)
0004	32-bit integer	?	min dist(0, 1)

offset	type	decimal value	description
0008	32-bit integer	?	min 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)

Report

Answer the questions below. You are recommended to use the same section numbering as they are listed.

1. Implementation

    a. How do you divide your data?
    b. What's your configuration? And why? (e.g. blocking factor, #blocks, #threads)
    c. Briefly describe your implementation in diagrams, figures and sentences.

2. Profiling Results

    Provide the profiling results of following metrics on the biggest kernel of your program using NVIDIA profiling tools. NVIDIA Profier Guide.

    - occupancy
    - sm efficiency
    - shared memory load/store throughput
    - global load/store throughput

2. Experiment & Analysis

    We encourage you to show your results by figures, charts, and description.

    a. System Spec

    If you didn't use our `hades` server for the experiments, please show the CPU, GPU, RAM, disk of the system.

    b. Time Distribution

    Analyze the time spent in:

    - computing
    - memory copy (H2D, D2H)
    - I/O of your program w.r.t. input size.

    > Note:
    > You should explain how you measure these time in your programs and compare the time distribution under different configurations.

    c. Blocking Factor

    Observe what happened with different blocking factors, and plot the trend in terms of Integer GOPS and global/shared memory bandwidth. (You can get the information from profiling tools or manual) (You might want to check nvprof and Metrics Reference)

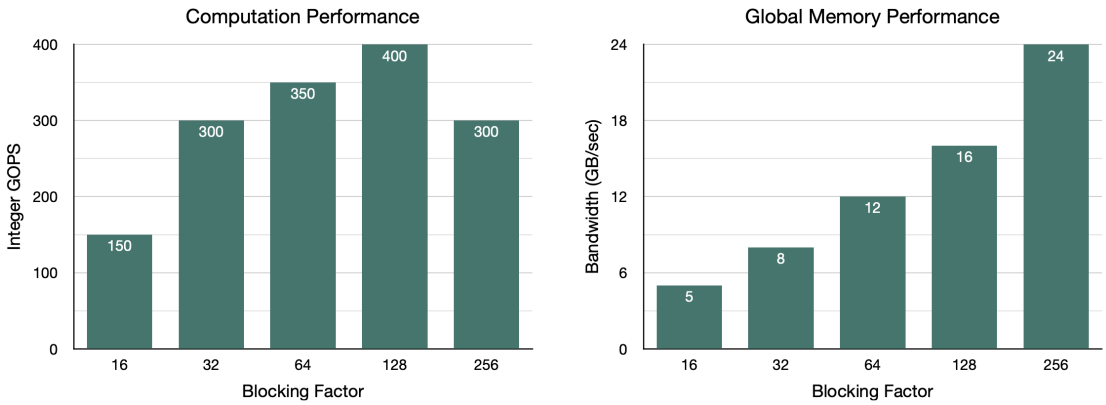

Figure 5: Example chart of performance and global memory bandwidth trend w.r.t. blocking factor

> **Note:**
>
> Note: To run `nvprof` on hades with flags like `--metrics`, please run on the slurm partition `prof`. e.g. `srun -p prof -N1 -n1 --gres=gpu:1 nvprof --metrics gld_throughput ./hw4-1 /home/pp20/share/hw4-1/cases/c01.1 c01.1.out`

d. Optimization

Any optimizations after you port the algorithm on GPU, descript them with sentences and charts. Here are some techniques you can implement:

- Coalesced memory access
- Shared memory
- Handle bank conflict
- CUDA 2D alignment
- Occupancy optimization
- Large blocking factor
- Reduce communication
- Streaming

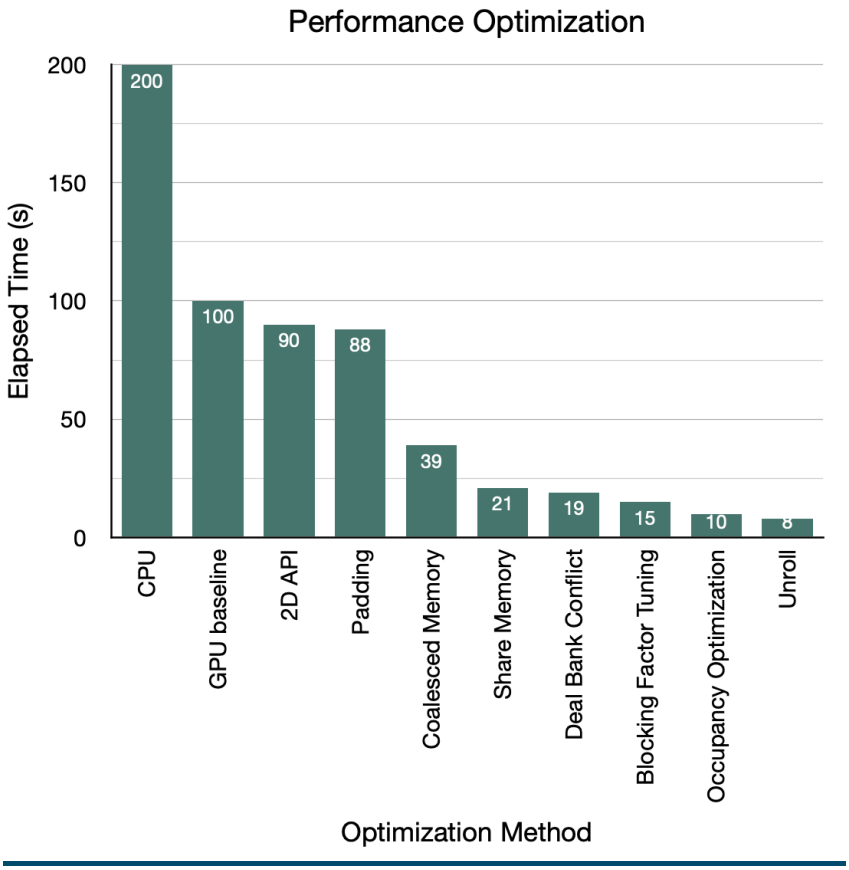

Figure 6: Example chart of performance optimization

e. Others

Additional charts with explanation and studies. The more, the better.

> Note:
>
> Note: The numbers in the charts above may not come from real data. Do NOT compare them with your own results!

3. Experience & conclusion

a. What have you learned from this homework?
b. Feedback (optional)

Grading

1. Correctness (30%)

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

2. Performance (30%)

- We have 40 performance testcases named *pXXk1*. *XX* = 11 ~ 50
- Each testcase has 30s time limit.
- Basically, larger *XX* test case require longer time.
- You will get *XX* point if you pass test cases *p11k1* ~ *pXXk1*. Otherwise zero.
- For example, if you pass test cases *p11k1* ~ *p23k1*, *p25k1* and failed other testcases. You will get 23 point.
- If *XX* > 30, than extra point will still count. (but the max point of this homework is still 100)

> Note:

> Note: Since the writing file will cost a lot of time when testcases become large. We will write the output to ramdisk when judging. If you want to measure the time of running a performance testcase, you can set the output path to `/dev/shm/xxx.out` to reduce writing time.
>
> e.g. `srun -N1 -n1 --gres=gpu:1 hw4-1 /home/pp20/share/hw4-1/cases/c01.1 /dev/shm/c01.1.out`

3. Report (25%)

   Grading is based on your evaluation results, discussion and writing. If you want to get more points, design as more experiments as you can.

   Must be a PDF document.

4. Demo (15%)

   A demo session will be held in the Lab. You'll be asked questions about the homework.

Submission

Upload these files to ILMS. Do not compress them.

- hw4-1.cu
- Makefile (optional)
- hw4-1_{student_ID}.pdf

Resources

- Use the `hw4-cat` command to view the binary test cases in text format.
- Resources are provided under `hades:/home/pp20/share/hw4-1/`:
  - `seq.cu` - Sequential Blocked FW source code
  - `Makefile` - example Makefile
  - Correctness Testcases: `hades:/home/pp20/share/hw4-1/cases/c*`
  - Performance Testcases: `hades:/home/pp20/share/hw4-1/cases/p*`
- Name your code `hw4-1.cu`. And under same folder, type `hw4-1-judge` to run the test cases.
- Scoreboard: https://apollo.cs.nthu.edu.tw/pp20/scoreboard/hw4-1/

Final Notes

- Contact TA via pp@lsalab.cs.nthu.edu.tw or iLMS 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 get **0 points** if we found you cheating.