CS542200 Parallel Programming
HW4: Blocked All-Pairs Shortest Paths

Due: January 11, 2015

1 Goal

This assignment helps you to get familiar with CUDA on multi-GPUs by implementing blocked APSP algorithm. Besides, in order to measure the performance and scalability of your program, experiments are required. Finally, we encourage you to optimize your program by exploring different strategies for bonus points.

2 Problem Description

In this assignment, you will modify sequential version of blocked Floyed-Warshall algorithm to CUDA version.

Given an $N \times N$ matrix $D = [d(i,j)]$ where $d(i,j)$ represents 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 $\{1, 2, \ldots, k\}$.

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

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

In the blocked algorithm, we partition $D^{(0)}$ into $N/B \times N/B$ blocks of $B \times B$ submatrices. The number B is called ***blocking factor***. For instance, we divide a 6×6 matrix into 3×3 blocks by $B = 2$.

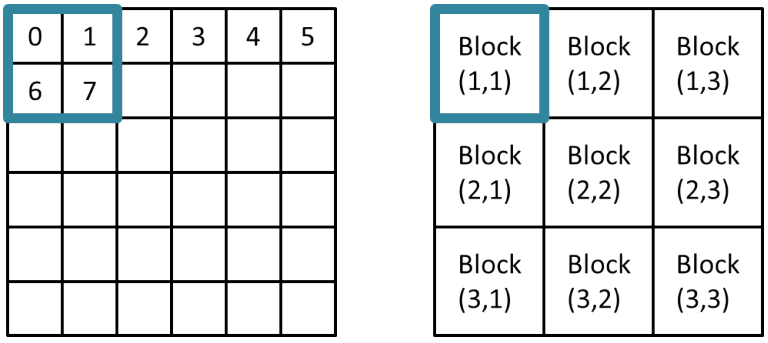


Figure 1: Divide a matrix by B = 2

For simplicity, we assume that B divides N. Blocked version of FW algorithm will perform N/B rounds, and each round is divided into 3 phases. It performs B iterations in each phase. The execution flow of a round is described step by step as below (N = 6, B = 2):

Assume we identify a block with index (I, J), where $1 \leq I, J \leq n/b$.

- **Phase 1** : Self-dependent blocks
  In K-th iteration, the first phase is to compute the $B \times B$ pivot block $D^{(K \cdot B)}_{(K,K)}$.
  For instance, In 1-st iteration, it computes $D^{(2)}_{(1,1)}$:

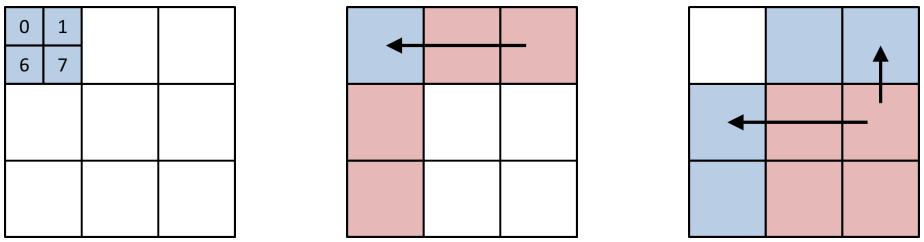
$$
\begin{aligned}
d^{(1)}(1,1) &= min(d^{(0)}(1,1), d^{(0)}(1,1) + d^{(0)}(1,1)) \\
d^{(1)}(1,2) &= min(d^{(0)}(1,2), d^{(0)}(1,1) + d^{(0)}(1,2)) \\
&\vdots \\
d^{(2)}(1,2) &= min(d^{(1)}(1,2), d^{(1)}(1,2) + d^{(1)}(2,2)) \\
&\vdots \\
d^{(2)}(2,2) &= min(d^{(1)}(2,2), d^{(1)}(2,2) + d^{(1)}(2,2))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

- **Phase 2** : Pivot-row and pivot-column blocks
  The result of pivot-row (pivot-column) blocks depends on pivot block in phase 1 and itself.
  For instance, the result of $D^{(2)}_{(1,3)}$ depends on $D^{(2)}_{(1,1)}$ and $D^{(0)}_{(1,3)}$:

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

- **Phase 3** : Other blocks
  The third phase computes remaining blocks $D_{(I,J)}$.
  For instance, the result of $D^{(2)}_{(2,3)}$ depends on $D^{(2)}_{(2,1)}$ and $D^{(2)}_{(1,3)}$:

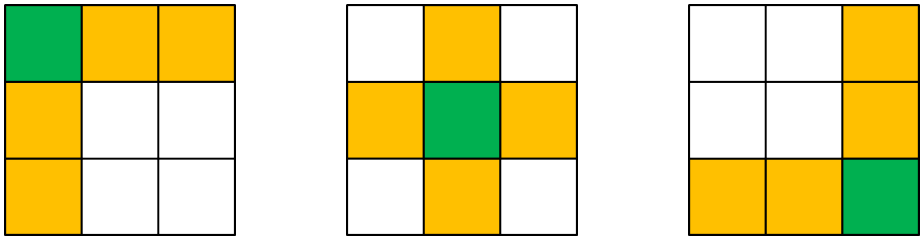
$$
\begin{aligned}
d^{(1)}(3,5) &= min(d^{(0)}(3,5), d^{(2)}(3,1) + d^{(2)}(1,5)) \\
d^{(1)}(4,5) &= min(d^{(0)}(4,5), d^{(2)}(4,1) + d^{(2)}(1,5)) \\
&\vdots \\
d^{(2)}(3,6) &= min(d^{(1)}(3,6), d^{(2)}(3,2) + d^{(2)}(2,6)) \\
&\vdots \\
d^{(2)}(4,6) &= min(d^{(1)}(4,6), d^{(2)}(4,2) + d^{(2)}(2,6))
\end{aligned}
$$



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

Figure 2: Blocked algorithm in 1^{st} iteration

In this example, it will run $6/2 = 3$ iterations, see Figure 3.



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

Figure 3: Blocked algorithm

For detailed information please refer to ***shared/HW4/block_APAP.cpp***.

3 Input/Output Format

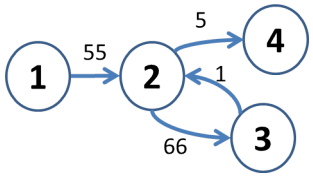
1. Your program have to **read/write** data from file. We will judge your program as below:

```
[kjs1095@lsalab ~]$ ./HW4_103065566_cuda.exe tiny_testcase output
[kjs1095@lsalab ~]$ diff -b output answer
```

2. The test case begins with a line contains two integer $\boldsymbol{N}$ ($1 \leq N \leq 6000$) and $\boldsymbol{M}$, representing the number of vertices and the number of edges. Each of the following $\boldsymbol{M}$ lines contains three integers ***Source_ID***, ***Target_ID*** and ***Value_of_Edge*** ($Source\_ID \neq Target\_ID$). All edges will be non-negative and less than 100. If there are re-assignments of weight for an edge, please follow the latest one.

```
[kjs1095@lsalab ~]$ cat tiny_testcase
4 4
1 2 55
2 3 66
3 2 1
2 4 5
```

(a) Sample content of input



(b) Graph

Figure 4: Sample Input

3. For output file, show shortest from i^{th} vertex to j^{th} vertex, separate them by single space. If there is no path from i^{th} vertex to j^{th} vertex, then the output should be ***INF***.

```
[kjs1095@lsalab ~]$ cat output
0 55 121 57
INF 0 66 5
INF 1 0 71
INF INF INF 0
```

Figure 5: Sample Output

4 Working items

You must implement four versions of blocked FW algorithm on GPU with the given restriction.

1. Single GPU
   a) Modify ***shared/HW4/block_APAP.cpp*** to CUDA version.
2. MPI version
   a) You should write CUDA program using multiple GPUs on multi nodes by MPI.
3. OpenMP version
   a) You should write CUDA program using multiple GPUs on single node by OpenMP.
4. Makefile
   Please refer some examples in ***shared/HW4/***.
5. Report
   a) **Title, name, student ID**
   b) **Implementation**
   How you divide the data? How to distribute? What's your configuration (e.g. block factor, #blocks, #threads) ?
   Briefly describe your implementation in diagrams, figures or sentences.
   c) **Profiling results**
   Provide the profiling results using profiling tools. (e.g. nvprof, nvvp)
   It's better to use print-screen and paste them as results on report.
   d) **Experiment & analysis**
   What and why you do this experiment? the result of your experiments, and **try to explain them**. (put some figures and charts) You can use the time measured by profiling tools.
      i. **System Spec**
      If you run experiment on your own machines, please attaches GPU, CPU, RAM, disk and network (ethernet/infiniband) information of your system.
      ii. **Weak scalability & Time distribution**
      Observing weak scalability of four implementations. Moreover, analyze the time for **computing**, **memory copy** and **communication** of your program. You should explain how you measure these time intervals in your program, and compare the time distribution under different distribution.
      iii. **Blocking Factor**
      Observing what happened when you tune the blocking factor, and plot the trend. You can refer Figure (6) and Figure (7) as examples.
      iv. **Compare four implementation**
      Compare the performance of your three implementations.
      v. **Others**
      Additional performance analysis plots and studies. The more, the better.
   e) **Experience/Conclusion**

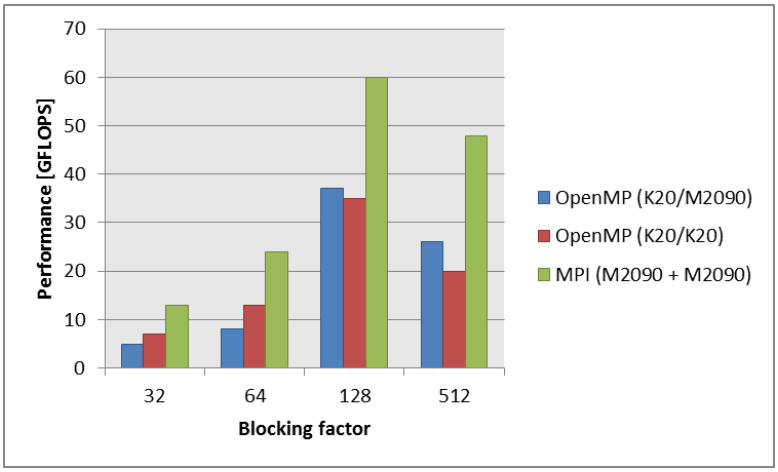


Figure 6: Example performance trend of blocking factor

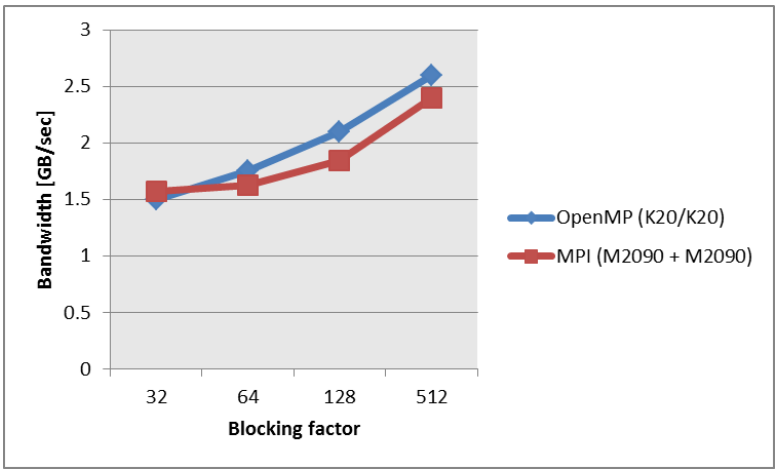


Figure 7: Example bandwidth trend of blocking factor

5 Grading

1. **Correctness** (48%)

   a) **[16%]** Single-GPU

   b) **[16%]** Multi-GPU with MPI

   c) **[16%]** Multi-GPU with OpenMP

   It will be judged by released testcases in ***shared/HW4*** directory.

2. **Report** (30%)
   It is based on your evaluation results, discussion and writing.
   If you want to get more points, design as more experiments as you can.

3. **Demo** (20%)

4. **Bonus** (5%)

   1^{st} optimization technique will get 3 points.
   2^{nd} optimization technique will get 2 points.

   These optimization techniques you used must be implemented in all version of CUDA code, and should achieve better performance.

   i) Shared memory

   ii) Streaming

   iii) Pinned memory

   iv) Dynamic Load-Balancing (This can be only implemented in (b) & (c))

   v) Others (TA will judge on demo, techniques like #unroll are unacceptable.)

   ***Total score* = *min*(Lab2+Correctness+Report+Demo+Bonus , 100)**

6 Reminder

1. Please compress your codes and report in the file named **HW4_{Student-ID}.zip** which contains:

   a) HW4_{Student-ID}_cuda.cu

   b) HW4_{Student-ID}_mpi.cu

   c) HW4_{Student-ID}_openmp.cu

   d) HW4_{Student-ID}_report.pdf

   e) Makefile

   And upload it to iLMS before 1/11(sun) 23:59:59

2. Because we have limited machines for you guys for tuning. Please **start your work ASAP** and do not leave it until the last day!

3. Late submission penalty policy please refer to syllabus.

4. Asking questions on iLMS or through e-mail is also welcome!