

CS542200 Parallel Programming

Homework 3: All Pair Shortest Path

Due: November 26, 2017

1 GOAL

This assignment helps you get familiar with **Pthread** and know the differences between **synchronous** and **asynchronous** computing using a **fully distributed vertex centric graph computation** model. In this assignment, you need to implement all pair shortest path in following versions:

1. Pthread
2. Fully distributed synchronous vertex-centric MPI
3. Fully distributed asynchronous vertex-centric MPI
4. Fully distributed asynchronous vertex-centric MPI + Pthread (**bonus**)

2 PROBLEM DESCRIPTION

2.1 PTHREAD VERSION

→ In this version, you are asked to implement a solution using Pthread. You may choose to implement **any algorithm without any limitation or constraints** as long as its result is **correct**, and follows the input/output file format.

2.2 FULLY DISTRIBUTED SYNCHRONOUS VERTEX-CENTRIC MPI

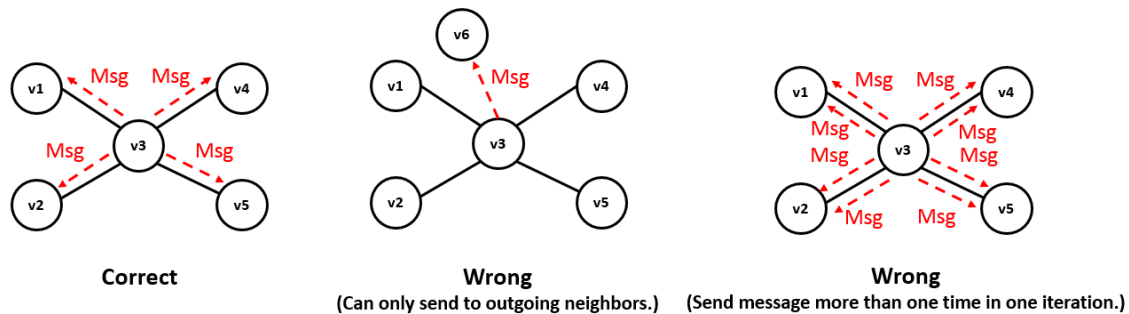
→ In this version, you are asked to implement a fully distributed **synchronous** solution using MPI.

→ **Vertex centric** programming model means that during the computation, **each MPI process controls a vertex** and does local computation (ex: calculate the new shortest distance of its own vertex) and local communication (ex: only send messages to its outgoing neighbors).

→ **Synchronous** computation means that each MPI process (vertex) can only **communicate with each of its neighbors once per iteration!**

→ **Termination condition:** After each iteration, a **blocking collective call** (e.g., `MPI_Allreduce ()`) **MUST** be called to determine a termination condition is reached. The termination condition is reached when no vertex expects to send message for

the next iteration. This is also the required synchronous point of this implementation.



[Example of message propagation in one iteration.]

2.3 FULLY DISTRIBUTED ASYNCHRONOUS VERTEX-CENTRIC MPI

- ➔ In this version, you are asked to implement a fully distributed **asynchronous** solution using MPI.
- ➔ The difference between asynchronous and synchronous version is that each MPI process (vertex) **acts independently**, and allows to communicate with its neighbors at any time.
- ➔ **You need to design a termination detection mechanism to make sure the correctness of your implementation.**

2.4 FULLY DISTRIBUTED ASYNCHRONOUS MPI WITH PTHREAD

- ➔ In this version, you are asked to extend your fully distributed asynchronous solution using both MPI and Pthread.
- ➔ Here, we partition a graph into subgraphs and assign each sub-graph to a process. A process creates a set of threads, and let each thread handles the computation of a vertex. In other words, the MPI processes are only responsible for communication, and threads are doing the actual computation work.
- ➔ It is an **optional** working item for **bonus point**. So you can choose not to implement it. But noted, this version can greatly reduce the overhead of process creation and accelerate execution. So not only you can receive the bonus points, you will receive **higher performance score** as well.

3 INPUT/OUTPUT

3.1 INPUT PARAMETERS:

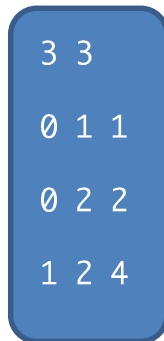
```
$ ./executable ${1} ${2} ${3}
```

- $\${1}$: the input file name [**string**]
- $\${2}$: the output file name [**string**]
- $\${3}$: number of threads or MPI processes [**integer**]

※ For the hybrid parallel (4th) version, it is the number of MPI processes. The number of threads per process can be decided by the number of vertices in a sub-graph.

3.2 INPUT FILE FORMAT:

The first line of an input test case consists of 2 integers V and E separated by a single space, which represent number of vertices and number of edge of this graph. Each of the following E lines consists of 3 integers i, j ($i \neq j$) and W , separated by a single space between any two numbers. It means that the distance of vertex i and vertex j is W . Figure (1). is a sample input format.

A blue rounded rectangle containing the sample input format text:

```
3 3
0 1 1
0 2 2
1 2 4
```

Figure (1). Sample input format

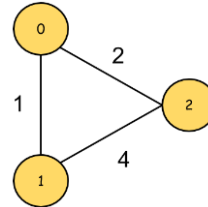


Figure (2). Graph of sample input

Notice that:

- ✧ $1 \leq V \leq 2000$
- ✧ $V - 1 \leq E \leq 4000000$
- ✧ $0 \leq i, j < V$
- ✧ $0 \leq W \leq 100$

- ☞ All graphs are connected.

3.3 OUTPUT FORMAT

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

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

The number at the i^{th} line and j^{th} column is the shortest-path distance from the i^{th} vertex to the j^{th} vertex.



```
0 1 2
1 0 3
2 3 0
```

Figure (3). Sample output

Notice that:

- ☞ The output records must be **sorted by the vertex id**.
- ☞ $\text{Distance}(i, j) = 0$, where $i = j$

4 GRADING

This homework will be graded by the correctness, report, demonstration, and performance as described below:

4.1 CORRECTNESS (55%)

During the demo time, TA will use different graph to test your program, and check whether your **output** can pass our judge script.

The detail score distribution of each part is:

- ☞ Pthread version (20%)
- ☞ Synchronous vertex-centric MPI version (15%)
- ☞ Asynchronous vertex-centric MPI version (20%)

4.2 REPORT (15%)

A. Design

Explain the **details** of your implementations of three versions in diagrams, figures, sentences. You also need to answer the questions below:

- (a) What algorithm you choose to implement Pthread version? Why?
- (b) What are the pros and cons of synchronous and asynchronous version?
- (c) Other efforts you've made in your program

B. Performance analysis

You can use your own server to run the experiments, but you need to list the hardware of your own server in the report. (Ex: CPU, memory, network...)

However, make sure your code can run on our platform since we will use our platform to test your correctness and performance!

Plots for all three (or four) versions of implementation:

PLEASE EXPLAIN YOUR GRAPH. DO NOT JUST PUT THE GRAPHS WITHOUT ANY EXPLANATIONS!

(a) Strong scalability chart

Scalability to number of cores (Problem size is fixed)

- i. The size of the problem is determined by the number of vertices
- ii. Select **ANY 4 input** test cases for this plotting
- iii. For MPI, scaling means to increase the number of **CORES** not processes. (The number of processes must be the same as the number of vertices.)

(b) Performance profiling:

- i. Time distribution for Pthread:
computation, synchronization (lock wait time), I/O, others you prefer to show.
- ii. Time distribution for MPI:
communication, computation, synchronization (barrier wait time), I/O

(c) Any other experiments you think meaningful to compare and analyze the differences between implementation versions.

C. Experience and conclusion

- (a) What have you learned and observed from this assignment?
- (b) What difficulty did you encounter when implementing this assignment?
- (c) If you have any feedback, please write it here.

4.3 DEMO (10%)

- ☞ Test the correctness of your codes.
- ☞ Go through your codes. Make sure your implementation follows the requirements.
- ☞ We will ask some questions about your codes and report.

4.4 PERFORMANCE (20%)

- ☞ Pthread version (10%)
- ☞ MPI version (10%) (We will use the faster one from the sync, async, hybrid versions for scoring.)

4.5 BONUS (15%)

- ☞ Implement a hybrid parallelism version with correct result.

5 REMINDER

1. We provide sample test case under /home/pp17/ta/hw3 for your reference.
2. We provide a judge script under /home/pp17/ta/hw3 for your reference.
3. Please package your codes and report in a file named **HW3_{student-ID}.zip** and name your codes/report as follows:

- (a) APSP_Pthread.cc
- (b) APSP_MPI_sync.cc
- (c) APSP_MPI_async.cc
- (d) APSP_Hybrid.cc (optional)
- (e) Report: HW3_\${student_id}_report.pdf
- (f) Please pack your compile script or Makefile with your source code and report.

Make sure your compile script can execute correctly and your code has no compile error before you upload your homework, and copy to **homework/HW3 directory under your home directory in our server** before **11/26(Sun) 23:59**

(You also need to upload to iLMS, but we will use the **file timestamp** under your homework/HW3 to see if you submit your homework late.)

4. Since we have limited resources for you guys to use, please start your work **ASAP**. Do not leave it until the last day!
5. Late submission penalty policy please refer to the course syllabus.
6. **Do NOT try to abuse the computing nodes by ssh to them directly.** If we ever find you doing that, you will get 0 point for the homework!
7. Asking questions through iLMS or email are welcome!