HW 3: All-Pairs Shortest Path (CPU)

Deadline: 2 Dec 2020, 18:00

Contents

- Goal
- Requirements
  - Command line specification
  - Input specification
  - Output specification
- Report
- Grading
- Submission
- Final Notes

Goal

This homework helps you understand the all-pairs shortest path problem.

Requirements

In this assignment, you are asked to:

1. **Implement a program that solves the all-pairs shortest path problem**.
   - You are required to use **threading** to parallelize the computation in your program.
   - You can choose any threading library or framework you like (pthread, std::thread, OpenMP, Intel TBB, etc).
   - You can choose any algorithm to solve the problem.
   - You must implement the shortest path algorithm yourself. (Do not use libraries to solve the problem. Ask TA if unsure).
2. **Create your input test case**.

Command line specification

```
srun -N1 -n1 -cCPUS ./hw3 INPUTFILE OUTPUT
```

- `CPUS`: Number of CPUs, specified by TA.
- `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:
    - $2 \leq V \leq 6000$
    - $0 \leq E \leq V \times (V - 1)$
    - $0 \leq \text{src}_i, \text{dst}_i < V$
    - $\text{src}_i \neq \text{dst}_i$
    - if $\text{src}_i = \text{src}_j$ then $\text{dst}_i \neq \text{dst}_j$ (there will not be repeated edges)
    - $0 \leq w_i \leq 1000$

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	?	dist(V-1, V-2)
$4V^2$-4	32-bit integer	0	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. Which algorithm do you choose?

b. Describe your implementation.
c. What is the time complexity of your algorithm, in terms of number of vertices V, number of edges E, number of CPUs P?
d. How did you design & generate your test case?

2. Experiment & Analysis

a. System Spec

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

b. Strong scalability

Perform strong scalability experiments, plot your experiment results.

> Note:
>
> You have to make sure that your execution time is long enough so that the results are meaningful.

c. Time profile

How much time is spent in input / computation / output.

> Note:
>
> You should not use the `clock()` function to measure execution time, because it measures *CPU time*. You can use either:
>
> - `clock_gettime` with `CLOCK_MONOTONIC`
> - `std::chrono::steady_clock`
> - `omp_get_wtime`

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.

Time limit for each case: (960 seconds) / (number of CPU cores).

2. Performance (20%)

- We will use several different hidden test cases (denoted C) to run your code. Your performance score will be given by:
$$\sum S(C) \times \frac{T_{best}(C)}{T_{yours}(C)}$$
- C is a test case
- $S(C)$ is the score allocated to the test case.
- $T_{best}(C)$ is the shortest execution time of all students for the test case, excluding incorrect implementations.
- $T_{yours}(C)$ is your shortest execution time for the test case, excluding incorrect implementations.
- $\sum S(C) = 20$

3. Generated testcase (10%)

Your generated testcase should conform to the Input specification. Points are given according to the **total time all other students finish your test case**. More time results in higher testcase score.

If someone cannot finish your case corrrectly, the time used for calculation will be: time limit × 1.5.

Your code must pass your own testcase in order to get points on your generated testcase.

4. Demo (20%)

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

5. Report (20%)

   Grading based on your evaluation of the results, discussion and writing.

   Must be a PDF document.

Submission

Upload these files to ILMS. Do not compress them.

- hw3.cc
- Makefile (optional)
- hw3_{student_ID}.pdf

Put your generated test case in `~/homework/hw3/mycase.in` on apollo before the deadline. We will copy the file locate there afterward.

Final Notes

- Type `hw3-judge` to run the test cases.
- Scoreboard: https://apollo.cs.nthu.edu.tw/pp20/scoreboard/hw3/
- Use the `hw3-cat` command to view the binary test cases in text format.
- Resources are provided under `/home/pp20/share/hw3/`:
  - `Makefile` - example Makefile
  - `validator.cc` - sample source code to validate an input testcase against the input specification
  - `cases/` - sample testcases
- 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.