Homework 2: OpenMP Programming

Introduction to Big Data Systems course

Due: March 12, 2021 23:59 China time. Late submission results in lower (or even no) scores.

For questions or concerns, contact TA (Jiping Yu) by WeChat. Or send an email to yjp19@mails.tsinghua.edu.cn if you could not use WeChat.

Overview

In this assignment, you will implement a parallel graph algorithm called **Page Rank** on a multi-core CPU. You need use **OpenMP** for multi-core parallelization.

Environment

You will need to run code on the server machine finally for this assignment.

You can refer to `server.pdf` about how to connect to the cluster.

For this homework, you need to program in **C++** to use OpenMP directly. The compiler is GNU C++ compiler (`g++`) version 9.3.0.

Files

We have placed the starter code directory at `/data/hw2_src` on the server. You should copy it to your home (by `cp -r /data/hw2_src ~` for example). Alternatively, you can use `hw2_src.zip` that we sent to you in your local computer. The contents are the same.

We placed some data for this homework at `/data/hw2_data` . Inside it, `com-orkut_117m.graph` is the dataset used for final evaluation. We also supplied smaller graphs

at `/data/hw2_data/small_graphs`. **Avoid copying large dataset to your home directory, and do not exceed the 1 GB limit.**

Description

Background: Representing Graphs Using Arrays

The starter code operates on directed graphs (stored in binary format), whose implementation you can find in `common/graph.h` and `common/graph_internal.h`. A graph is represented by an array of edges (both `outgoing_edges` and `incoming_edges`), where each edge is represented by an integer describing the id of the destination vertex. Edges are stored in the graph sorted by their source vertex, so the source vertex is implicit in the representation. This makes for a compact representation of the graph, and also allows it to be stored contiguously in memory.

For example, to iterate over the outgoing edges for all nodes in the graph, you can use the following code which makes use of convenient helper functions defined in `graph.h` (and implemented in `graph_internal.h`):

```
for (int i=0; i<num_nodes(g); i++) {
  // Vertex is typedef'ed to an int. Vertex* points into g.outgoing_edges[]
    const Vertex* start = outgoing_begin(g, i);
    const Vertex* end = outgoing_end(g, i);
    for (const Vertex* v=start; v!=end; v++)
      printf("Edge %u %u\n", i, *v);
}
```

In the `tools/` directory of the starter code you will find a useful program called `graphTools` that prints statistics about graphs. You should execute `make` in `tools` directory to generate the execuatable file `graphTools`.

Students sometimes want to make their own graphs for debugging. You can write down a graph definition in a text file and use the graphTools app to convert it to a binary file that can be used with the assignment. See the command help for: `./graphTools text2bin graphtextfilename graphbinfilename`.

Implement Parallel Page Rank

We would like you to begin by implementing a parallel version of the well-known page rank algorithm using OpenMP. We will provide you a starter code in the attachment, which contains some C++ program files. Please first take a look at the simple serial implement of Page Rank in the function `pageRank()`, in the file `page_rank.cpp`. After understanding the serial implement, you should implement a parallel version with OpenMP.

Notes: the only file you should modify when you submit the code is `page_rank.cpp`. Of course, you can modify other files to help you debug or understand the starter code. So, make sure that all code is compilable and runnable without other files modi!ed.

How to run the code

The code is implemented in C++. We provide a `Makefile` in the code directory for compiling. You can use `make` to compile code. If there is no error, you will get an executable named `pr`. You can run your code, checking correctness and performance against the serial version we provided using:

```
./pr <graph_file> <threads_num>
```

We located the graph data at `/data/hw2_data`.

To run the program on a compute node (which is required when you want to measure the performance), use this command:

```
srun -n 1 ./pr <graph_file> <threads_num>
```

You must use `-n 1` for this lab, because the program only works on one node.

You must run your program at the dataset called `com-orkut_117m.graph`, and report the performance at your writeup.

For example, you can use `srun -n 1 ./pr /data/hw2_data/com-orkut_117m.graph 4`, to test correctness and performance your algorithm at the dataset called `com-orkut_117m.graph` when using 4 threads. If your implementation is correct, there will be a message `Your Page Rank is Correct` and your running time will be reported.

Hand-in

Files

You should submit a single ZIP file, strictly following the format.

For example, if your cluster username is `2020123456`, you should submit a ZIP file exactly named `hw2_2020123456.zip`. Inside it, there should not be any subdirectories, but it should **only contain these two files**:

- `hw2_2020123456_report.pdf`. Your report in PDF format.
- `hw2_2020123456_page_rank.cpp`. Your modified version of Page Rank.

You do not submit other source code, since `page_rank.cpp` is the only file you may modify (for hand-in). You should make sure we can compile your code, by simply replace the `page_rank.cpp` inside the original `hw2_src` directory and type `make`. The compilation should not issue any errors (warnings are fine), and you should make sure your code works correctly with `com-orkut_117m.graph` dataset.

Submitting

If you have access, submit the ZIP file to the web learning homeworks. Otherwise, email the ZIP file to TA. You should submit before **March 12, 2021** 23:59 China time. Late submission results in lower (or even no) scores.

Scoring

Correctness (20%)

You will get full marks for correctness, if you successfully parallelize the Page Rank algorithm with OpenMP, and your results are correct for `com-orkut_117m.graph`.

Performance (40%)

The faster your implementation is, the more scores you can get (if your results are correct).

Report (40%)

Your report should at least contains:

- Brief description of your implementation
- Time result of 1, 2, 3, and 4 threads
- Time result of different OpenMP schedule strategies (if applicable to your code)
- Explain possible reasons, in brief, why your speedup for 4 threads is more/less than 4 (e.g. what causes performance loss?)

Bonus (optional, up to +10%)

The node has hyperthreading. There are 4 cores, 8 threads on each node. Report your result of 8 threads and compare it with 4 threads. Is it faster or slower? According to your implementation, how does hyperthreading affect the performance? Explain why.

Some materials

These are only for reference, and you are not required to read them.

LLNL openmp tutorial: https://computing.llnl.gov/tutorials/openMP/

Pagerank: https://en.wikipedia.org/wiki/PageRank