15-418/618 Fall 2021
Assignment 4
Parallel VLSI Wire Routing via MPI

Assigned: Wednesday, October 13th
Due: Wednesday, October 27th, 11:59PM
Last day to handin: Friday, October 29th, 11:59PM

This assignment aims to introduce you to parallel programming using MPI. You will solve the same problem as in Assignment 3, but using the message passing model rather than a shared memory model.

1 Policy and logistics

You will work in groups of two for the problems in this assignment. Turn in a single writeup per group, indicating all group members. Any clarifications and revisions to the assignment will be posted on the class "Assignments" web page, and also announced via our class Piazza web site.

You may clone the Assignment 4 starter code from the course Github:

```
git clone https://github.com/cmu15418f21/Assignment-4.git
```

2 Programming task: Parallel VLSI Wire Routing

As in Assignment 3, the programming task for this assignment is inspired by VLSI wire routing. Please refer to the Assignment 3 handout for details of the problem. Input and output files are expected to be in the same format as in Assignment 3.

Your goal is to optimize the performance of the simulated annealing algorithm described in Assignment 3. As a reminder, the algorithm is as follows:

```
for (iteration = 0; iteration < N_iters; iteration++) {
  foreach wire {
    with probability P {
      select random wire route
    } otherwise {
      consider all paths which first travel horizontally
      consider all paths which first travel vertically
      select best wire route
    }
  }
}
```

Note that best indicates the lowest cost path, where paths are restricted to having 0, 1, or 2 bends. Default values for N_{iters} (5) and P (0.1) are specified in the starter code and do not need to change.

3 Logistics

You should parallelize this application using the OpenMPI implementation of the MPI programming model. To get started with MPI, here are some websites you may find useful:

- http://mpitutorial.com/ contains a brief introduction to MPI.
- https://www.open-mpi.org/doc/v4.1/ contains the documentation for the MPI API.

In addition to the input files that were provided in Assignment 3, we have made some new input files available to you in the `code/inputs` directory. These input files have a smaller area and contain more wires than the input files from Assignment 3. You will need to report your performance on the new `code/inputs` files as part of your assignment submission.

We provide a script called `code/validate.py` to validate the consistency of output wire routes and the cost array. You can run `python validate.py -h` for instructions on how to use the script.

We also provide a visualization tool, located in `code/WireGrapher.java`, to help you visualize the wire routes in a graphical format. Unlike Assignment 3, you are not required to show the routes and cost array in a graphical format within your submission. You can compile and run WireGrapher using the following commands (make sure to SSH with the `-Y` flag):

```
$ javac WireGrapher.java
$ java WireGrapher [input]
```

4 The Parallel Machines

Feel free to use the machines in the Gates cluster for your initial development and testing. Host names for these machines are `ghcX.ghc.andrew.cmu.edu`, where X is between 47 and 86.

However, you will collect your final performance numbers on the PSC Bridges-2 RM (Regular Memory) machines. For more information on these machines, please see the `tutorials/machines.pdf` document. You may also find it helpful to refer to the Bridges-2 User Guide.

To use OpenMPI on the PSC machines, you must first run `module load openmpi` to load the OpenMPI module. To run a program with MPI, use the `mpirun` command. An example invocation of this command is as follows:

```
mpirun -np <numprocs> ./wireroute -f inputs/easy_1024.txt
```

The handout also provides an example MPI program `sqrt3` that approximates sqrt(3).

5 Performance Metrics

Metric 1: Computation time

To measure the performance, the starter code measures and prints the total computation time for each MPI process. Each MPI process will print its computation time, and the program's running time is determined based on the maximum across each processes' computation time.

Metric 2: Max cost

We are also interested in the quality of the solution. The primary quality metric is the max cost in the cost array. In VLSI, this determines the number of layers needed to fabricate the chip, which has a large impact on cost.

Metric 3: Sum of squares cost metric

As a secondary quality metric, we are interested in the cost summed for each wire path, as the simulated annealing algorithm attempts to minimize this quantity. This metric is calculated as follows:

```
cost = 0
foreach wire {
  cost += cost along wire route
}
```

Note that this is **not** the same as adding all the elements in the cost array. Rather, this is equivalent to summing the square (i.e., x * x) of each element in the cost array, hence the name sum of squares cost metric.

6 Performance Analysis

As there are many real-world factors that affect the performance of your code, it is important to be able to explain how changes to your code affect performance. For example, if a version of your program exhibits disappointing performance, we would like you to explain what is causing poor performance for that code. We would also like you to explain what you did to fix the problem and improve performance. **We are especially interested in hearing about the thought process that went into designing your program, and how it evolved over time based on your experiments**.

Your report should include the following items:

1. A detailed discussion of the design and rationale behind your approach to parallelizing the algorithm. Specifically try to address the following questions:
   - What approaches have you taken to parallelize the algorithm? What efforts did you make to profile your code and identify bottlenecks, and how did this data affect your designs?
   - What information is communicated between MPI processes, and how does this affect performance? How does communication scale with the number of processes?
   - If your implementation operates on stale (or partially stale) data, how does the degree of staleness affect the quality of the solution?
   - At high process counts, do you observe a drop-off in performance and/or solution quality? If so, (and you may not) why do you think this might be the case?
   - Why do you think your code is unable to achieve perfect speedup? (Is it workload imbalance? communication? synchronization? data movement? etc?)
2. A plot of the computation speedup (metric 1) vs. the number of processors sampled at 1, 4, 16, 64, and 128 processors. Speedup is calculated as $\frac{T_1}{T_P}$, where T_1 is the time for one processor, and T_P is the time for P processors.
3. A plot of the max cost (metric 2) vs. the number of processors. Please explain your results and how it's affected by your design.
4. A plot of the sum of squares cost metric (metric 3) vs. the number of processors. Please explain your results and how it's affected by your design.

7 Grading

Your grade will be based on the report and the performance and correctness of your code. We will use the performance metrics as described in Section 5. We will run with the three inputs in the `code/inputs` directory of the handout, and we reserve the right to run additional inputs. For your reference, here are the performance metric numbers we obtained with our reference solution, measured on PSC:

`inputs/easy_1024.txt`

Processors	Computation time (s)	Max Cost	Sum of Squares Cost
1	8.42	19	98746608
4	2.34	18	98774356
16	0.84	18	98758806
64	0.37	19	99296828
128	0.31	28	102226530

`inputs/medium_1024.txt`

Processors	Computation time (s)	Max Cost	Sum of Squares Cost
1	42.26	35	488644808
4	27.35	36	488570754
16	11.91	35	488768862
64	3.70	40	489636714
128	2.25	42	494256544

`inputs/hard_1024.txt`

Processors	Computation time (s)	Max Cost	Sum of Squares Cost
1	50.89	41	625016209
4	32.59	40	625272337
16	14.93	43	625874503
64	4.34	42	626381975
128	2.72	48	629894107

We expect your solution to not be much worse than this baseline solution. Since there is a tradeoff between the quality of the solution and the computation time, we don't expect your results to match up perfectly, and we are more interested in your thought process on parallelizing the algorithm and the tradeoffs you make in designing your solution.

8 Hand In

If you are working with a partner, please form a group on Autolab before submitting your assignment. One submission per group is sufficient.

Your submission should be a `.tar` file with a single directory titled `andrewid1_andrewid2` consisting of your source code for `wireroute.cpp` and `wireroute.h` as well as your wire outputs for each of the files in `code/inputs` with `mpirun -np 128 <COMMAND>`. Please follow the directory structure below.

```
andrewid1_andrewid2
├── code
│   ├── wireroute.cpp
│   ├── wireroute.h
│   ├── main.cpp
│   ├── Makefile
│   └── Extra source files, if applicable.
└── outputs
    ├── output_easy_1024_128.txt
    ├── output_hard_1024_128.txt
    └── output_medium_1024_128.txt
```

Also remember to do a good job on *commenting your code*, which is very important.

Please upload your writeup in `.pdf` format to Gradescope (one submission per team) and select the appropriate pages for each part of the assignment. After submitting, you will be able to add your teammate using the add group members button on the top right of your submission. Your report should include the items listed in Section 6.