15-418/618 Fall 2018
Assignment 4: Parallel VLSI Wire Routing via MPI

Assigned: Friday, October 12th
Due: Wednesday, October 24th, 11:59pm

The purpose of this assignment is 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 people in solving the problems for this assignment. Turn in a single writeup per group, indicating all group members. Any clarifications and revisions to the assignment will be announced via our class Piazza web site.

To get started, download assignment4-handout.tar from Autolab (https://autolab.andrew.cmu.edu) to a directory accessible only to your team. Unpack the handout with:

```
> tar xvf assignment4-handout.tar
```

You will collect your numbers on the Latedays cluster. For this assignment, you will be using Xeon CPUs rather than Xeon Phi's, and your performance will be evaluated on at most 2 nodes (machines) each running up to 16 processes (threads). The `README` file explains how to build, test, and handin code. Please make sure you read this document and the `README` file before you start working on the assignment.

2 Programming task: Parallel VLSI wire routing

You will solve the same problem assignment from assignment 3. Please refer to assignment 3's 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 {
      select best wire route
    }
  }
}
```

Note that best indicates the lowest cost path, just like in assignment 3, 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 MPI

We are 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/v1.6/ contains the documentation for the MPI API.
- https://www.citutor.org/ contains online courses for MPI and various other topics. Specifically, the `Introduction to MPI` is relevant. Registration is free.

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

4 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, and 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. This is because the simulated annealing algorithm attempts to minimize this quantity. The metric is calculated as follows:

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

Note that this is **not** the same as the secondary quality metric from assignment 3. 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.

5 Performance analysis

You will be required to also submit a writeup of your findings. As there are many real-world factors that affect the performance of your code, it is important to be able to explain how your code changes 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 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?
   - How does the problem size (in terms of grid size, number of wires, and length of wires) affect performance?
   - At high process counts, do you observe a drop-off in performance? 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 time (metric 1) vs. the number of processors. Please generate separate graphs for 1 node and 2 nodes. Note that each node can run at most 24 processes.
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.
5. Division of effort: Given that you worked in a group, please identify how you divided the work for this project and how the credit should be distributed among the group members (e.g., 50%-50%, 60%-40%, etc.).

Please include your writeup as writeup.pdf in the root directory of the assignment handout.

6 Grading

Your grade will be based on the report and the performance and correctness of your code. The official performance and quality targets have not been released yet; we will try to post them on Piazza as soon as possible. There are currently three input files in the 'inputs/' directory. Stay tuned on Piazza for more inputs to test on.

7 Handin instructions

Please run `make handin` from the handout directory to generate the handin.tar file. It will require you have added your writeup.pdf to the directory. The Makefile will run `make clean` as part of `make handin`, but please remove any other large files not removed by `make clean`. Only your code and writeup.pdf are required. Your code should be readable and well-documented. Submit your handin.tar file to Autolab (https://autolab.andrew.cmu.edu).