Programming Assignment #1

It is recommended that students work in groups of two for this assignment.

In this assignment, you will implement a shared memory parallel version of the *LocusRoute* application using the ANL macros. *LocusRoute* is a standard cell router which routes wires between endpoints so as to minimize the total area of the layout. It is described in the paper "SPLASH: Stanford Parallel Applications for Shared Memory" that is available off my web page.

Problem Description

You will implement a simplified version of *LocusRoute*:

- Wires will have only two end-points (each specified as two non-negative integers).
- All routes will be at most two-bend routes, i.e., routes which have at most two corners or three straight segments. All routes should also fall within the bounding box defined by the two end-points of the wire.
- Only vertical height, i.e. height of the routing channels, will be used as a measure of area. The height of a routing channel is defined as the maximum number of wires running horizontally through any point of the channel.

Each wire should be routed from its start point to its end point, using at most two bends in the wire. You will need to keep a cost array which keeps track of the height of each channel at every routing pad, i.e. the coordinates in the "x" dimension. Your application should examine all permutations of the possible routes for a wire and pick the one which minimize the cost function given by:

$$\text{cost of wire route} = \Sigma \text{ value of all Cost array cells traversed by the route.}$$

While this is a local heuristic, we expect that it will work well in minimizing the global cost, which is the overall height of the layout. Figure 1 shows some possible routes for three different wires, including the entries which should be updated in the cost array.

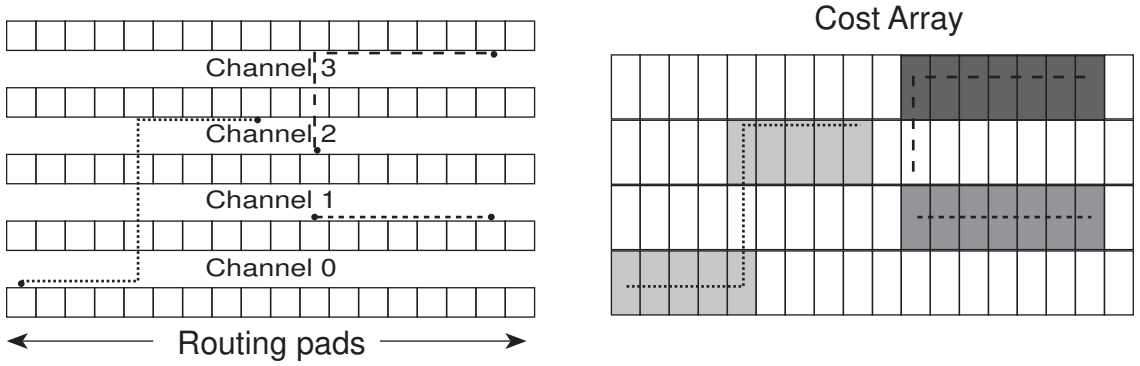

Figure 1: Wire routes and Cost array

Because the order of placement of the wires affects the cost, your program should perform more than one iteration of placement and routing. On every iteration after the first, you will need to ''rip out'' each

wire, recompute the costs for the different routes, and relay the wire. The number of iterations to perform will be specified in the input file. At the end of each of these iterations, you will print out the current total height of the circuit.

Conceptually, your loop should look like:

```
for i = 1 to number_of_iterations {
   for j = 1 to number_of_wires {
      if (i != 1) {
         rip_out_wire(j);
      }
      find_best_route_for(j);
      place_wire(j);
   }
   estimate_global_cost(G);
   print_global_cost(G);
}
```

Data Structures

The most important data structures are the *Cost* array and the *Wires* array.

- *Cost*: The cost array keeps track of the number of wires running through each routing cell of the circuit. The vertical dimension of the array is the number of routing channels in the circuit, and the horizontal dimension is the width of the circuit in routing cells. Each array element consists of one integer - the number of wires running horizontally across the channel at that routing cell.
- *Wires*: The wires array keeps track of the endpoints of the wire and the bend points of a routed wire. At initialization the endpoints will be read in from the data file. As a route for each wire is computed, the points where the wire turns should also be stored in this array. These are needed for subsequent iterations when the wire is "ripped up" and re-laid so that you can decrement the appropriate entries in the cost array.

You will also need other data structures to describe things such as the current route being examined.

Input

Your program should take two arguments - the number of processors and the input file describing the wires. The input file describes the circuit size, the number of iterations to perform, and a list of wire endpoints. The input files will be in the form:

```
<num_of_channels> <width_of_routing_channel>
<num_of_iterations>
<Wire1_x1> <Wire1_y1> <Wire1_x2> <Wire1_y2>
<Wire2_x1> <Wire2_y1> <Wire2_x2> <Wire2_y2>
...
```

You will have to run your program with a variety of input files. These will test how well your program handles locality and load balancing. Input files are provided in ~jps/cs598/pa1/inputs on toofan.princeton.edu.

Problem Specifics

You are to parallelize the program using three different methods of task assignment:

- **Simple global task queue** - All of the wires that need routing should be inserted into a single structure. Whenever a processor is freed, it should pick one or more wires from this queue and route them.
- **Distributed task queue** - To avoid the bottleneck of a single task queue, use a distributed task queue, with one queue per processor. Do an initial assignment of wires to queues so as to exploit memory / cache locality. For load balance, if one processor finishes working on its task queue, it should "steal" one or more wires from one of the other processors' task queue.
- **Best static partition** - For this method, once you have read in the input file of wires, you are to statically assign them to processors (based on some heuristic) so as to maximize load balance and cache / memory locality. (Note processors can no longer steal other wires.) For this case, you should report your results for two different implementations. In the first implementation, the cost array elements are updated without any locks. The second implementation should use proper synchronization to assure proper updating of the cost array.

What to Turn In

In addition to calculating the overall and computational speedup of your algorithms, you should print the total height (the sum of the maximum heights of each routing channel) of your routed circuit after each iteration. This will give you some idea of the rate of convergence of your algorithms. In addition, your write-up to the assignment should include:

- A description of your algorithms, noting any special features. Give justifications for the various design decisions that you made, and where possible relate observed performance to these design decisions.
- A table of the total height after each iteration for each of the input files . Also include the total runtime of your program on each input file.
- A graph for each input file of overall and computational speedup for 1, 2, 4, 8 and 12 processors.
- A printout of your code.