EECS 587 Homework due September 10, 2024

This homework must be submitted by the start of class. Submit it to the Assignments folder on Canvas. On the due date we will go over the assignment in class, examining the solutions. Late submissions will not be accepted. As the course overview states, the homework is to be typewritten, though you can draw figures by hand (and can take a photo of the figure and include it in the pdf).

For this homework you can work in pairs, or just by yourself if you prefer. If you work in a pair then just turn in one copy of the assignment, with both names on it. Once you start working with a person you cannot change and start working with another, nor can you stop working together. However, you can talk with prospective partners about things like when/where you would get together to discuss the problems. If you want a partner but can't find one we can help.

You cannot search on the web to see if the solution has been posted there, you cannot use any AI entity to help solve the problem, nor can you ask people taking the course now or that have taken this course previously. If previous attendees help you they run the risk of having the college retroactively reducing their grade. In addition, you are not allowed to post this to the web. If you do then you are at risk of having your grade lowered, retroactively if necessary.

For this project the processors are constructed of standard microprocessors, able to do standard operations such as multiply, compare, square root, find the j^{th} bit of a number, etc., in a constant amount of time. There are p processors, with ids $0 \ldots p-1$. Each processor starts with the values of p, its id, and an integer value v. All processors must run the same program (i.e., SPMD, Single Program Multiple Data), though the program may have conditional statements which use p, id, etc. The program can use as many scalar variables as you want. However, the total amount of memory a processor has must be a fixed amount which does not depend on p.

For communication, there are 2 operations:

- send(destination, value): send value to processor destination.
- receive(origin, variable): receive the value sent by processor origin, putting it into variable.

The receive operation is *blocking*, meaning that the processor will wait until the value has arrived. If the value is sent before the receive is issued, then it is stored in a buffer and immediately available when the receive command starts. The send operation is *nonblocking* in that the processor sends the value and immediately goes on to the next statement.

If a processor tries to send to or receive from a processor that does not exist or it is not connected to, then the program fails to run and your homework fails to solve the problem.

For quicksort we needed to compute scan(A,i,j,sum), that is, when we did the scan when working on $A(i:j)$ we wanted processsor k, $i \le k \le j$ to end up with the value nextdest $= \sum_{j=i}^{k} v(j)$, where $v(j)$ denotes the v value in processor j. The values were either 1 or 0, depending on where the key was before or after the pivot. The initial scan we did over all the values was scan(A,0,p-1,sum). For this homework problem we will compute scan(A,0,p-1,min), that is, to have processor i end up with the value $\min_{j=0}^{i} v(j)$, where all values are integers. We won't worry about computing scans over only part of A. To be able to compute scan(A,0,p-1,min) efficiently in parallel, you need to figure out how to decompose the problem into parts that can be done in parallel. For example, suppose $p = 8$ and the values are 4, 2, 1, 5, 7, 0, -8, 4. Then scan(min) is 4, 2, 1, 1, 1, 0, -8, -8. If you find the scan(A,0,p-1,min) on the first half, and simultaneously on the second half, you get 4, 2, 1, 1 and 7, 0, -8, -8. Call these values $w(0), w(1), \ldots, w(7)$. The first half is the correct values of scan(min), but for every value in the second half, i.e., for all $4 \le k \le 7$, the correct value of scan(min) is $\min\{w(3), w(k)\}$. Also note that the values for the halves may have been determined iteratively, simultaneously solving four problems of size 2.

For the two abstract computers listed below

1. Write an efficient program, in pseudo-code, to compute scan(A,0,p-1,min). You must make it clear that your program is correct. By an efficient program I mean one that minimizes worst-case running time, as measured in O-notation.

   The program must be written in SPMD style, i.e., the program explicitly shows what the processor does depending on its *id*.

2. Produce an O-notational analysis of the running time of your algorithm as a function of p. Assume that all processors start at the same time and run at the same speed.

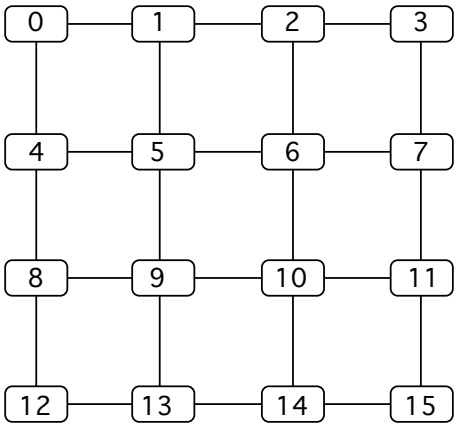
Do this for

1. A *completely connected computer* (also called *fully connected*), where every processor is connected to every other one. If you need to you may assume that p is an integral power of 2, but you will lose a small amount of credit. If you make this assumption make it clear that you have done so.

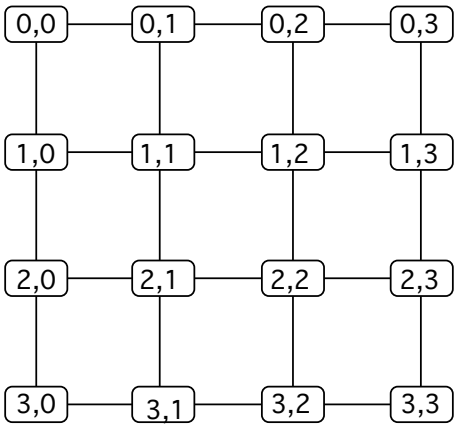
2. A *2-dimensional mesh*. For this computer p is a perfect square. For processor i, let $\mathrm{row}_i = \lfloor i/\sqrt{p} \rfloor$ and $\mathrm{col}_i = i - \mathrm{row}_i\sqrt{p}$. Processors i and j are connected if and only if either $\mathrm{col}_i = \mathrm{col}_j$ and $|\,\mathrm{row}_i - \mathrm{row}_j\,| = 1$, or $\mathrm{row}_i = \mathrm{row}_j$ and $|\,\mathrm{col}_i - \mathrm{col}_j\,| = 1$, i.e., each computer is connected to its up, down, left and right neighbors. Note that some processors do not have 4 neighbors.

   Hint: first solve it for a 1-dimensional mesh, i.e., processors in a line. Note that this is very similar to what we did in class, but there wasn't communication between a processor and the ones immediately below it or above it. That would have confused the class even more.

To better understand the numbering in the 2-dimensional mesh, assume $p = 16$. The connnections between the processors in terms of their ids look like



This is more obvious once each processor has computed its row and column coordinates from its id.



You can reverse the process, converting (row, column) coordinates into the processor id, by $id = \text{row} \cdot \sqrt{p} + \text{col}$.