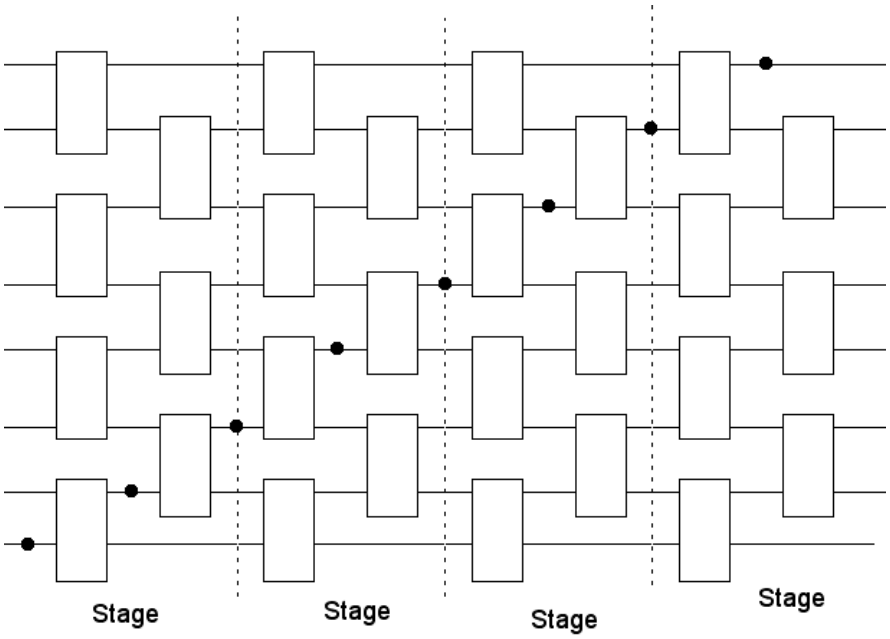
4DM4 Assignment #1 Hints,
Tues, Oct. 1, 2013

Hello Class

The course 4DM4 covers the issue of "Design", to meet our CEAB accreditation requirements. Assignment #1 is an exercise in Design. There are many different ways to proceed, and one needs to make assumptions, state these assumptions, and proceed along the design path. Iteration is likely required, since one's first attempt at a design might not work.

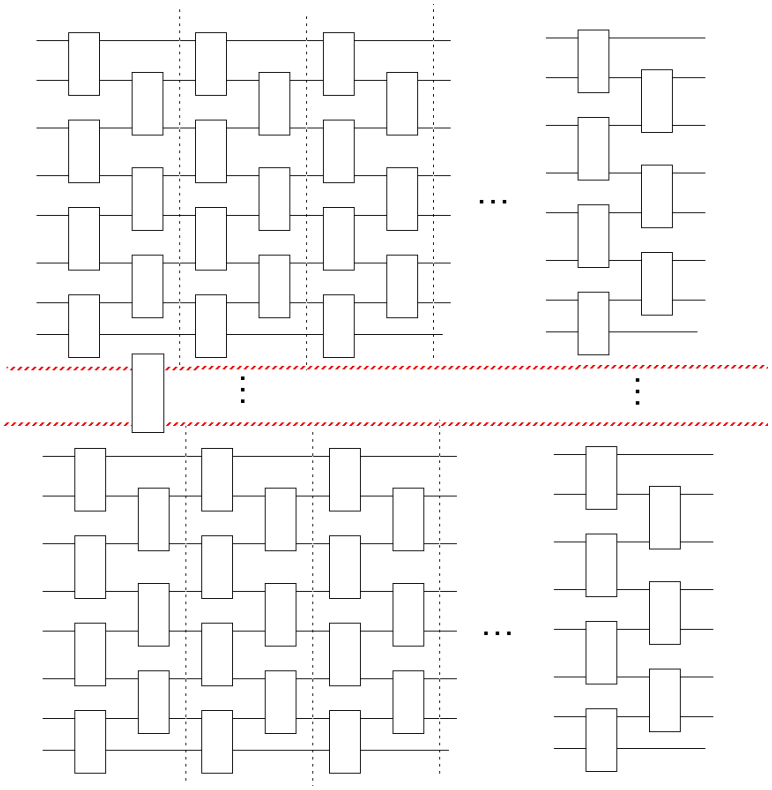
Here is my first attempt at a design. There are probably other ways to start the design too.

FIG 1: 4x4 Bubble Sort



- to sort N = 8 numbers, we need N/2 'stages', where each stage has (N-1) FLOPS
- lets round this off, so let each stage perform N flops
- overall, to sort N numbers there are (N/2)*(N) flops = (1/2)*N^2 flops

FIG 2: NxN Bubble Sort

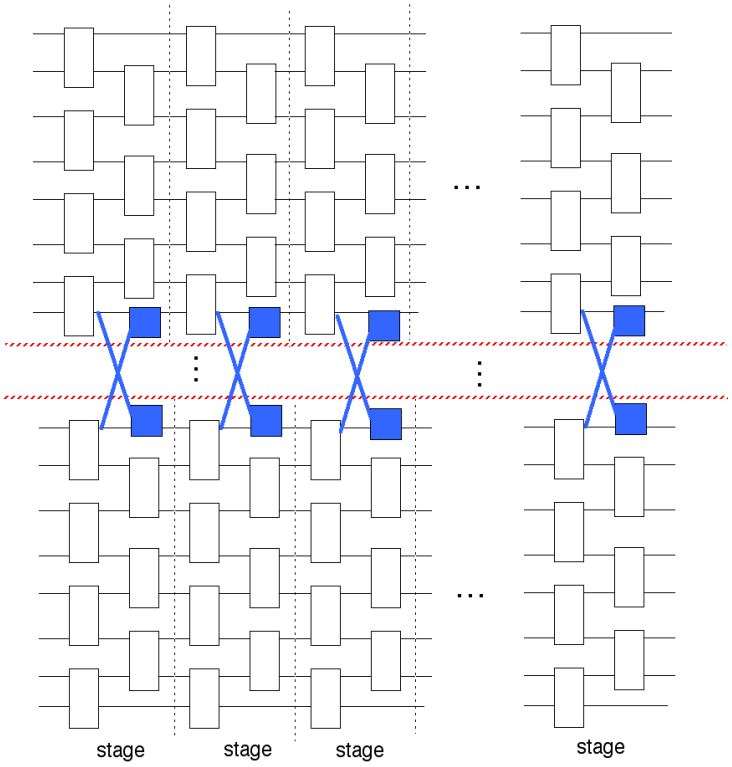


- consider a larger bubble sort;
- Lets sort N=1024 numbers = 2^10
- in total, we need N/2 stages, with (N) Flops/stage
- for N=1,204 we need (N/2)(N) flops = (512)(1,024) flops = 2^19 flops

- lets assume we have P = 8 processors available,
- lets divide the large bubble-sort data-flow graph into horizontal slices
- each processor performs one slice
- call the height of each slice N'
- in this example, the height of each slice = N' = (N/P) = (1,024) / (8) = 128 numbers

- there are some data communications at the boundaries of each slice
- slices in the middle have an upper and lower neighboring slice, and they must exchange data with those neighbors

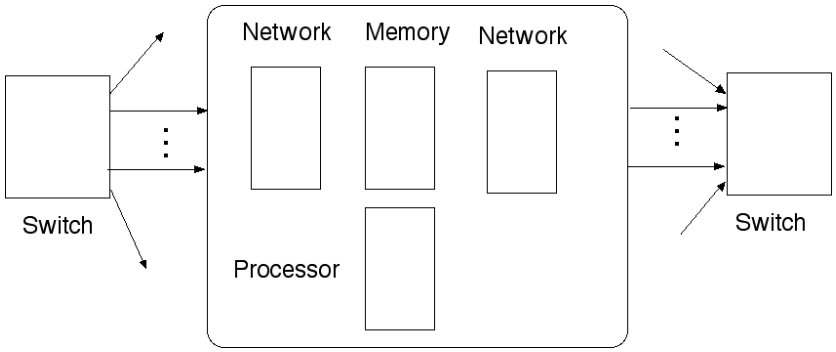
FIG 3: NxN Bubble Sort, with communication between processors



- let there be P processors, and N numbers to sort
- each slice has a height = (N/P) numbers
- in the above diagram, the 2 blue boxes represent a binary compare-exchange box sitting on the boundary between 2 slices; here, we duplicated it and put a copy in each slice
- each processor now performs (N/P) flops in each stage in its slice, then it send and receives 1 floating point number to each neighboring slice (assume this is 8 bytes)
- Asssume each processor has a performance of 100 Gflops/sec => (1/100) nanoseconds per flop

- each processor will work for (N/P)*(1/100) nanoseconds, and then it will send/receive 8 bytes to each neighboring slice
- assume the sending/receiving can be pipelined with another set of computations; that means we must have enough memory to store 2 sets of data (numbers) to be sorted
- for N=1,024 and P=8, each processor works for (1,024/8)*(1/100) nanoseconds, and then sends 8 bytes; If we ignore the packet header overhead, the bit rate of the communication link (one direction) is 64 bits every 1.28 nanoseconds = 50 Gigabits/sec (where 1 Gigabit = 10^9 bits)

FIG. 4 Design with one processor per board



Here is a board with 1 processor, which implements a slice
- it receives its data from the outside world, through a switch
- it performs its computation in a stage, and then sends/receives data to its neighboring slices
- we can now determine the placement and number of switches, and then compute the performance of the system, and the energy cost of the system

Other Options:
- we can try slicing up the data-flow graph other ways
- we cab try other sorting algorithms (but that gets harder)

- we can try putting multiple (<= 8) processors in a PCB (printed circuit board)
- this arrangement will allow processors on the same board to communicate with each other through a shared cache, without needing an Ethernet switch port or Ethernet switch
- the Ethernet switch bandwidth needed for communication with neighboring slices will decrease, as more communications is done between processors on the same board
- we need to ensure that each processor has enough cache IO bandwidth to support its instruction execution, and to support the exchange of data with neighboring processors
- we have to ensure that the shared cache has enough cache IO bandwidth to support its processors, and to support the communications with neighboring slices
- we could also arrange the cache in hierarchies, but we need to keep track of the cache IO bandwidth to make sure everything has enough IO bandwidth to avoid stalling

FIG 5: First Pass at a Design