EE/CS 451 Midterm 1

Open book, Open notes

Fall 2020
Instructor: Viktor Prasanna
3:30 – 5:30pm
Friday, 9/25/2020

Problem #	Topic	Points	Score
1	Memory System Performance	15	
2	Shared Memory Programming	20	
3	Shared Memory Programming	20	
4	Message Passing	20	
5	Interconnection network (4 sub)	25	
Total		100	

Student Name:
Student USC-ID:

The exam duration is 120 mins (3:30pm-5:30pm). You have 5 mins to download the exam in advance (Piazza->Resources->Exams), and 10 mins to upload the completed exam at the end (Blackboard->Assignments->Midterm 1).

- **Option 1**: Download the exam electronically, use a writable tablet for writing your answers, stop writing at 5:30pm, export the exam to PDF and upload it before 5:40pm.
- **Option 2**: Download the exam, write clearly on paper with problem numbers labeled at the top of each page, stop writing at 5:30pm, scan the paper and upload before 5:40.

Your writing must be **recognizable with human eyes**. Any un-clear answer may not receive full credit.

Please turn on your camera on Zoom.

If you have any questions regarding the exam, use the chat on Zoom.

1 Memory System Performance (15 Points)

Consider a memory system which has 200-cycle latency DRAM and is connected to a processor operating at 2GHz. The processor-memory bus can support one word per cycle streaming bandwidth. Assume cache has been disabled. The processor has two floating point multiply-add units. Each multiply-add unit can execute one multiplication and one addition per processor cycle.

a. What is the peak floating-point performance of the processor? (2 pts)

b. Consider the execution of the following code on such a platform:

```
$result = 0;$
$For\ (i = 0;\ i < 2000000000;\ i++)$
```

For all the following subproblems, assume each element of A, B and C is one word initially stored in DRAM, and result can be stored in the local register of the processor.

b.(1) Assume the memory access is streaming. What is the sustained performance in the best case? State any assumption(s) you make. (3 pts)

b.(2) Now assume the processor-memory bus can support **four** words per cycle streaming bandwidth, and the memory access is streaming. What is the sustained performance in the best case? State any assumption(s) you make. (3 pts)

b.(3) Now assume data access is not streaming. What is the worst-case achievable performance? (2 pts)

c. Consider a uni-processor that has a direct-mapped cache with 8 cache lines. The cache line size is 4 words.

c.(1) Consider the execution of the following program on the processor described in part c. The size of each element is 1 word. The matrix is stored in row-major order.

```
$For\ (i = 0; i < 128; i++)$
    $For\ (j = 0; j < 128; j++)$
```

compute the cache hit ratio for read operations. Explain. (2 points)

c.(2) Consider the execution of the following program on the same processor. The size of each element is 1 word. The matrix is stored in row-major order.

```
$For\ (j = 0; j < 128; j++)$
    $For\ (i = 0; i < 128; i++)$
```

Compute the cache hit ratio for read operations. Explain. (2 points)

c.(3) Repeat c.(1) and c.(2) assuming cache line size is 16 words instead. (1 point)

2 Shared Memory Programming (20 points)

A Minimum Spanning Tree (MST) is a subset of the edges of a connected, edge-weighted undirected graph (G) that connects all the vertices together, without any cycles and with the minimum possible total edge weight. The serial version of Boruvka's algorithm to find a MST is shown below. It is an iterative algorithm - in each iteration, it greedily selects the minimum-weight edge emanating from each component and merges the connected components. Note that a connected component is a connected subgraph such that any two vertices of the component are connected by a path in G. In the algorithm, ***E*[]** is the array of all edges in the graph, where each edge has a source vertex, destination vertex and weight attributes (e.src, e.dst, e.weight).

Data structures:

1. **components[]** is an array with size $|\boldsymbol{V}|$ to track the root vertex and all other vertices of each component, it is indexed by vertex numbers and valued by component-root-vertices. For example, if a graph has 6 vertices, where vertices 1,3 are in component 1, vertices 2,5 are in component 2, and vertices 4,6 are in component 3, components = {1,2,1,3,2,3}
2. **min_edge[]** is an array with size $|\boldsymbol{V}|$ where min_edge[i] is the pointer to minimum-weighted edge emanating from vertex i if vertex i is root of a component (-1 otherwise).

Functions:

1. **find_root(components[], vertex i)** returns the root vertex of the component to which vertex i belongs – it simply returns components[i].
2. **union(vertex r1, vertex r2)** merges the two components (represented by their root vertices r1 and r2) into a new component, and automatically assigns the larger index between r1 and r2 as the new root vertex and updates components[]. The union pseudo code for Boruvka's is shown below.

Question: We parallelize each iteration of this algorithm (starting at line 3) using p threads, assuming each thread is responsible for $|\boldsymbol{V}'| = \frac{n}{p}$ vertices, and the arrays **components[], min_edge[], *E*[]** and ***V*[]** are shared. You can use locks and/or barriers to avoid race conditions, but you should explain the need and justify the usage of such operations and avoid any unnecessary use of them. You should avoid deadlock. Your implementation should ensure that no error is thrown - You may add any conditional statements in the pseudo code where appropriate to ensure correctness, including in the union function.

Serial pseudo codes:

```
Boruvka's algorithm (G(V,E))
Input: undirected weighted graph G
Output: the set mst containing all edges in MST
--------------------------------------------------
Initialize: mst=empty set;
While total number of components>1:
  Initialize min_edge[] to -1 for all vertices
  /*1. identify minimum-weighted edge going out of each component*/
  For all edges e in E:
    C1=find_root(components, e.src)
    C2=find_root(components, e.dst)
    If (C1!=C2):
      If(min_edge[C1]==-1 or e.weight<E[min_edge[C1]].weight):
        min_edge[C1]=e
      If(min_edge[C2]==-1 or e.weight<E[min_edge[C2]].weight):
        min_edge[C2]=e
  Endfor
```

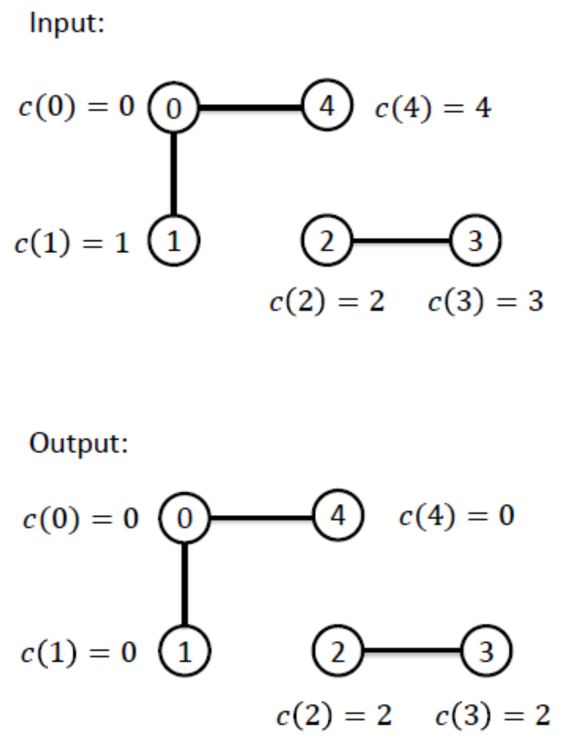
```
union(C1, C2)
--------------------------------------------------
    For all i where components[i]=min(C1,C2):
      Components[i]=max(C1,C2)
      if (no such i): throw_error() & end program
    Endfor
```

3 Shared Memory Programming (20 points)

Given an undirected graph $G(V, E), V = \{0, \dots n-1\}$, a connected component is defined as a sub-graph such that any two vertices of the component are connected by a path in G. The root of a connected component is defined as the smallest vertex in the connected component. The label of a connected component is its root vertex.

The algorithm to find the connected component (the label of the root vertex) to which each vertex belongs to is illustrated in the Figure below. For each vertex i $(0 \le i < n)$, we use $c(i)$ to keep track of the label of the connected component that the vertex belongs to. The algorithm is iterative; in each iteration, all the edges are traversed to update $c(0), \dots, c(n-1)$. The algorithm terminates when there is no update for any $c(i)$ in an iteration. Then, $c(i)$ becomes the label of the connected component that vertex i belongs to. In Figure 1, we also show an example input graph and its output. Suppose we want to parallelize the algorithm using p $(p > 1)$ threads, with each thread executing the computation for $\frac{|E|}{p}$ edges ($|E|$=total # of edges in G) in each iteration. We define an iteration for a thread as the work done to traverse its own edges once.

```
Connected components
Let edge$(i, j)$ denote the edge connecting vertex $i$ and vertex $j$
Let $c(i)$ denote the smallest vertex such that there exists a path
from vertex $i$ to vertex $c(i)$
Connected_components $(G(V, E))$
For each vertex $i$ in $V$                    // Initialization //
    $c(i) = i$;
    At_least_one_vertex_has_update = true;
End for
Number_of_iterations = 0;
While At_least_one_vertex_has_update = true do
    Number_of_iterations ++;
    At_least_one_vertex_has_update = false;
    For each edge$(i, j) \in E$ do
        $m$ = min$(c(i), c(j))$;
        If $c(i) > m$ then
            $c(i) = m$;
            At_least_one_vertex_has_update = true;
        End if
        If $c(j) > m$ then
            $c(j) = m$;
            At_least_one_vertex_has_update = true;
        End if
    End for
End while
```

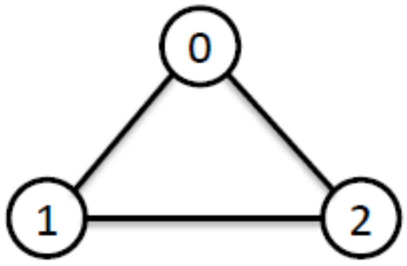


a. What are the shared variables for each thread? (3 points)

b. Write the pseudo code of the function executed by $Thread_w$ $(0 \leq w < p)$. Note that your code must ensure that at the end of k-th iteration, all vertices in a connected component at a distance less than or equal to k from the root should have the correct label of that component. (7 points)

c. If your code in part b does not use any locks, will the execution terminate? If yes, will it be able to produce the correct result? Explain. If no, explain why the execution will not terminate. (3 points)

d. Given the input graph shown below, if your code in Part b does not use any lock, for p = 3, what is the total number of iterations that the algorithm executes in the best case? What is the total number of iterations that the algorithm executes in the worst case? Explain. (4 points)



e. For every barrier inserted in part b, explain its necessity using a concrete example showing that without the inserted barrier, the program may exit incorrectly. (3 points)

4 Message Passing (20 points)

a. The figure below shows two processes, Process A and B, running at 1 GHz. At time 0 ns, Process A issues a Send command to send a 4-byte integer to B; and at time 10 ns, Process B issues a Receive command to receive the integer from A. Assume there are no other processes. The transmission bandwidth between Process A and Process B is 4 bytes/ns. If using buffered Send/Receive, the bandwidth for writing or reading local buffers is 8 bytes/ns. If "handshake" is required, the request or response takes 3 ns.

Process A	Process B
Line 1: int a[2]={1,10}	Line 1: Receive(a, 2, A)
Line 2: Send(a, 2, B)	Line 2: Printf("%d, %d",a[0],a[1])
Line 3: a[1]+=1	

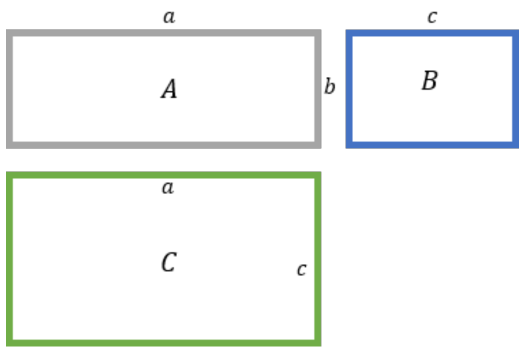
a.(1) If Process A and B use blocking buffered Send/Receive data transfer, when will Line 2 of Process B start to be executed? Why? [3 points]

a.(2) If Process A and B use non-blocking buffered Send/Receive data transfer, when is it safe to execute Line 3 of Process A? Why? [3 points]

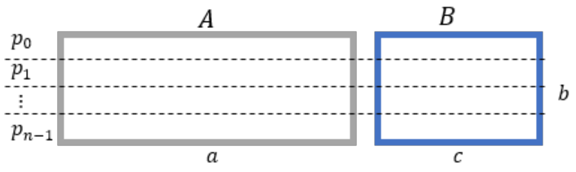
a.(3) If Process A and B use blocking non-buffered send/receive data transfer, when will Line 3 of Process A start to be executed? When will a reach process B? Why? [4 points]

a.(4) If Process A and B use non-blocking non-buffered send/receive data transfer, when is it safe to execute Line 3 of Process A? Why? [3 points]

b. Consider the matrix multiplication $A(a \times b) * B(b \times c) = C(a \times c)$ shown in Figure below.



b.(1) Suppose we parallelize this problem using $\boldsymbol{n}$ **processors** in a **1-D mesh with wraparound connection**. Both matrices are partitioned row-wise such that each processor stores $\frac{b}{n}$ rows of A and B, respectively. Assume each processor has enough local memory to store the entire output matrix. Write a message passing program to be executed on processor k $(0 \leq k < n)$ to perform the matrix multiplication. At the end of the execution every processor should store the entire output matrix C.(4 points).



b.(2)Assuming blocking Send/Receive, derive expressions (in order notation) for the **total** parallel execution time - specify the computation and communication times and explain your answer. (3 points).

5 Interconnection Network (25 points)

a. Definition 5.1: A p-input and p-output CLOS network can be defined as a 3-stage network where stage 0 and stage 2 consist of two $\frac{p}{2} \times \frac{p}{2}$ switches and Stage 1 consists of $\frac{p}{2}$ 2×2 switches.

a.(1) Draw such a network for $p = 8$ (2 points)

a.(2) Apply definition 5.1 of CLOS network to recursively decompose all the switches in Stage 0 and Stage 2 until the network only has 2×2 switches. Draw such a network for $p = 8$ (3 points)

a.(3) In general, derive an expression for the total number of switches and the total delay from an input to an output for an n input and n output CLOS network which is obtained by recursively applying Definition 5.1 to the switches in Stage 0 and Stage 2 as done in part a.(2). Note that the final network consists of only 2×2 switches. (Assume the delay of each 2×2 switch is equal to 1 unit). (5 points)

b. For the 2-stage permutation network in Figure 1, how many permutations in total can be realized? Show a permutation that cannot be realized. (3 points)

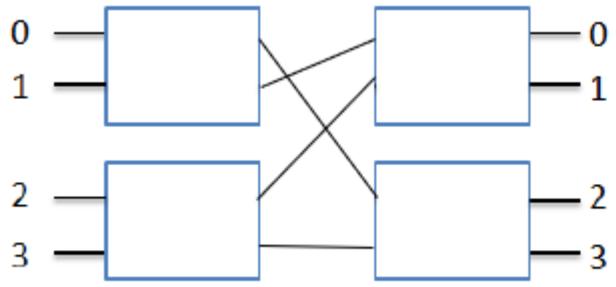
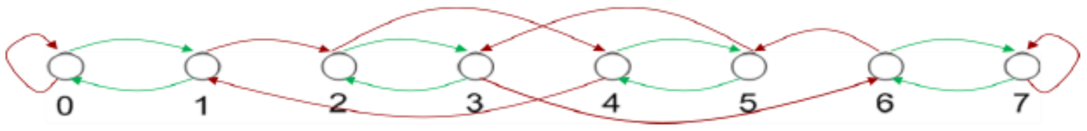


Figure 1: 2-stage permutation network

c. For $n = 8$ Shuffle Exchange network, show the complete routing path from processor 5 to processor 0 using the routing algorithm discussed in the class. Additionally, show a path with distinct source and destination vertices that conflicts with the above path. Mark your paths on the shuffle exchange network shown below. (3 points)

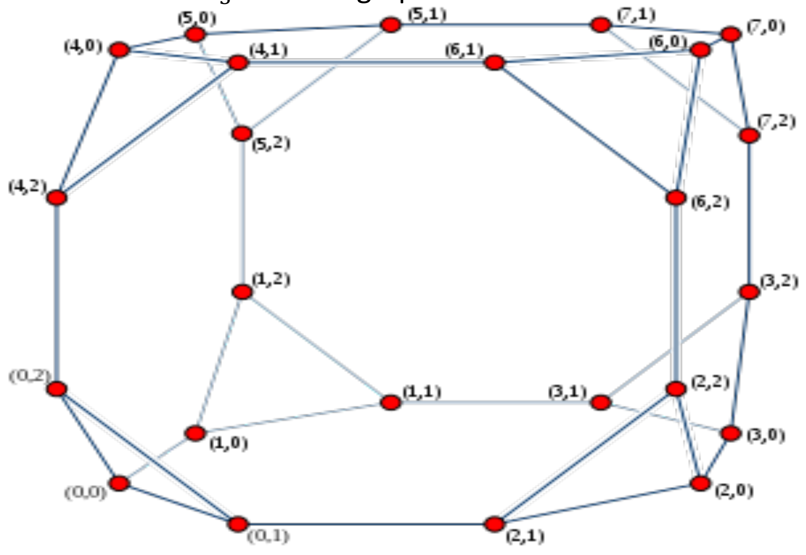


d. Definition 5.2: A Cube-Connected Cycle (CCC) is a network formed by replacing each vertex of a hypercube network by a cycle.
CCC of order n (denoted CCC_n) can be defined as a graph formed from a set of $n2^n$ nodes, indexed by pairs of numbers (x, y) where $0 \leq x < 2^n$ and $0 \leq y < n$.
There is a link between two processors with IDs (i, j) and (k, m) if and only if

a) $j = m$ and i differs from k in the j^{th} least significant bit or

b) $i = k$ and $j = m + 1 \; mod \; n$

Figure below shows a CCC_3 network graph.



d.(1) What is the diameter and bi-section width of CCC_2 ? Explain your answer. (4 points)

d.(2) We denote the links connecting different cycles as cross-links and the links within the same cycle as cycle-links. Consider the following routing algorithm for sending a message from node $A(x_A, y_A)$ to node $B(x_B, y_B)$ in CCC_n:

```
Start address $S(x_S, y_S) = A(x_A, y_A)$
While $x_S! = x_B$ or $y_S! = y_B$ repeat:
    $i = y_S$
    Denote $i^{th}$ bit of $x_S$ as $C_{iS}$, and $i^{th}$ bit of $x_B$ as $C_{iB}$
    If $(C_{iS}! = C_{iB})$
        Let $C_{iS} = C_{iB}$ ;   //Use $i^{th}$ dimension cross-link to relay the message to a neighboring cycle
    Else if $(C_{iS} == C_{iB}$ and $x_S! = x_B)$
        Let $y_S = y_S + 1 \, mod \, n$   //Use cycle-link to send to the next node within the same cycle
    Else if $x_s == x_B)$
        //Use cycle-links to hop to the destination node within the same cycle from the shortest direction
        $cc_+$ = number of hops required to route $y_S$ to $y_B$ in the index-increasing direction
        $cc_-$ = number of hops required to route $y_S$ to $y_B$ in the index-decreasing direction
        Take the routing path with min $(cc_+, cc_-)$
```

What is the maximum path length to route a message in a CCC_n based on the **above routing algorithm,** as an expression of total number of nodes, p. Explain your answer. (5 points)

Scratch Space