1. (a) Consider the problem of *Inter-Model Emulation*, that is, the creation of general purpose mappings from the structure of one message-passing parallel architecture into another. Explain in general terms why such mappings may be useful, and the properties we would like them to have. [*6 marks*]

   (b) In the Gray code based 2-D Mesh to Hypercube mapping discussed in the course, processor 6 of a 16 processor row-major ordered square 2-D Mesh maps to processor 7 of a 16 processor Hypercube.

      i. Explain carefully why this is the case. [*5 marks*]

      ii. Explain the corresponding mapping for square 2-D Mesh processor 12. [*4 marks*]

   (c) Consider the Hypercube to 2-D Mesh emulation problem (i.e. the reverse of the problem considered above).

      i. Explain why this is a challenging problem as the number of processors grows (NB you are not being asked to solve the problem). [*3 marks*]

      ii. In the course, we examined a 2-D Mesh mapping of the Hypercube-based Bitonic Mergesort algorithm. Explain the sense in which the resulting algorithm can be said to be "optimal", and describe the additional information which was exploited in achieving this successful mapping (NB you are not being asked to define the mapping itself). [*7 marks*]

Question 1

(a)

Finding a general purpose mapping could help us so that we can use networks which are easier to reason about, and help problem understanding, and then map the developed algorithm onto a different network, which may be more readily implemented.

We may also want to exploit algorithms that are optimized for frequency of link usage, and then apply in them in any type of network.

(b)

Gray code mapping for 16 processors:

Gray code	Row major index	Mesh (x,y) pos	Mesh to Gray	Hypercube index
0 0 | 0	0	(0,0)	0 0 0 0	0
0 1 | 1	1	(0,1)	0 0 0 1	1
1 1 | 2	2	(0,2)	0 0 1 1	3
1 0 | 3	3	(0,3)	0 0 1 0	2
	4	(1,0)	0 1 0 0	4
	5	(1,1)	0 1 0 1	5
	6	(1,2)	0 1 1 1	7
	7	(1,3)	0 1 1 0	6
	8	(2,0)	1 1 0 0	12
	9	(2,1)	1 1 0 1	13

	10 11 12 13 14 15	(2,2) (2,3) (3,0) (3,1) (3,2) (3,3)	1 1 1 1 1 1 1 0 1 0 0 0 1 0 0 1 1 0 1 1 1 0 1 0	15 14 8 9 11 10

(i)

Because Gray codes convert the integer to a binary number which is shuffled due to the mirroring that happens when creating them, 6 is mapped to 7.

(ii)

12 is mapped to 8.

(c)

(i)

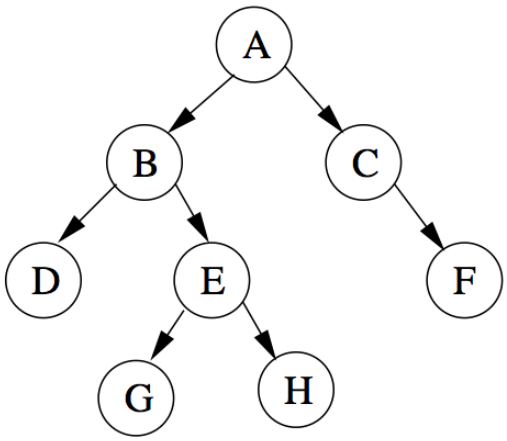
This is challenging because connections, yo! Hypercube has log(n) connections per processor, whereas 2D mesh always has four, so we run into problems for n > 16. Missing connections therefore have to be compensated for by routing the original hypercube connection through several mesh connections, resulting in different runtimes.

(ii)

The resulting mapping was said to be optimal, because the communication overhead from mapping non-existing connections to routes was minimized: more frequently used connections were laid closer together. This was accomplished by observing the recursive structure of the bitonic mergesort algorithm, where each of the steps that *generates* the bitonic sequence is composed of a similar operation found in the previous step, using the same connections for the compare-swap/-split operation.

2. (a) With the help of a small example, describe the standard pointer-jumping CREW PRAM algorithm for list ranking. Your answer should describe the conditions under which the algorithm is applicable and state its run-time and cost. [*10 marks*]

(b) Suppose that PRAM memory contains an array of n objects. As well as various data fields, the objects each have two pointer fields, `left` and `right`, whose values have been set to describe an arrangement of the objects into a binary tree (i.e. `left` points to an object's left child, and `right` points to its right child). An additional shared pointer `root` points to the tree's root object. The tree cannot be assumed to balanced. We would like to compute, for each object, its depth within the tree, where the root has depth `0`, its children have depth `1`, and so on. For example, for the tree below (in which objects are identified by letters), the depths are A:0, B:1, C:1, D:2, E:2, F:2, G:3, H:3.



Describe an n processor, pointer-jumping CREW PRAM algorithm for this problem, and explain its run-time. [*10 marks*]

(c) Briefly discuss the challenges which would arise in adapting pointer-jumping algorithms to work with

i. far fewer processors than objects, but still shared memory; [*3 marks*]

ii. far fewer processors than objects and non-shared memory. [*2 marks*]

(a)

Pointer jumping for list ranking works by following the forward pointers of connected nodes to increase the jump distance exponentially at each step. The ranks stored at each node are then used to increase the rank of the current node.

Processor:	1	2	3	4	5	6	7	8	9	10
rank	1	1	1	1	1	1	1	1	1	0
jump:	2	3	4	5	6	7	8	9	10	-
new rank:	2	2	2	2	2	2	2	2	1	0
jump:	3	4	5	6	7	8	9	10	-	-
new rank:	4	4	4	4	4	4	3	2	1	0
jump:	5	6	7	8	9	10	-	-	-	-
new rank:	8	8	7	6	5	4	3	2	1	0
jump:	9	10	-	-	-	-	-	-	-	-
new rank:	9	8	7	6	5	4	3	2	1	0

It's applicable when we have at least a single linked list and each processor has the necessary information about which element it belongs to before the algorithm starts, and n = p.

Runtime is log(n), cost is n*log(n)

(b)

Algorithm:

- Reverse the direction of the pointer
    - Assume that every node has a parent pointer available
    - Every node sets the parent pointer of it's children to itself - run time of O(1)
- Use pointer jumping from each processor to move to parent until root is reached
    - assume that each node holds it's rank in a shared array or in some means that is accessable from the others
    - this will have run time of O(log d) where d is the largest depth of the tree.
    - the run time depends on the structure of the tree.

Root can set its depth immediately. It then follows the pointers to its children and sets their depth values as its value plus one. After every such step each processor checks whether its depth value has been set, and then sets the values of its children. Since this is a binary tree, setting the value of children is at most cost "2", so the overall runtime is the same asymptotically

== depth of tree. If the tree is just a long “list”, this algorithm would have runtime O(n). If the tree is balanced it would have runtime O(log(n)). This isn’t really pointer jumping tho, because we’re not increasing the jump distance at each step.

c)

(i)

With less processors, the run time may not increase proportionally (i.e not n/p) since for this to happen the processors would need to process the nodes in the order that they are present in the list, but this is obviously unknown. So the worst case run time becomes O $(\frac{n}{p})^2$ if all the nodes happen to be processed sequentially. The optimal run time would be (n/p) and in this case each processor would have a part of the link list which itself is all linked and the processor happens to process them in the inverse of their rank (the one that happened to have the smallest rank first)

(ii)

With Message Passing, we either incur a high overhead to transmit the results after each computation to update the values that any other processor has, or else we risk linear processing through the local part of the list that the processor has. It will be a balance between increasing computation time or increasing communication time.

3. (a) i. Define *cost-optimality* and explain why it is a desirable property for PRAM algorithms. [*3 marks*]

ii. You are in charge of a shared-memory parallel programming project. You have employed a PRAM algorithms expert as part of the team. For some new problem, the expert offers you an asymptotically cost-optimal algorithm for the CRCW-associative PRAM, and an asymptotically non cost-optimal algorithm for the EREW PRAM. Explain why the EREW algorithm might nevertheless be the better choice for your project. [*5 marks*]

(b) An array in PRAM shared memory stores a sequence of n characters. You are to design a CREW PRAM algorithm which checks whether the sequence is correctly *parenthesised*, meaning that occurrences of the characters "(" and ")" are correctly matched and nested. For example, with "X" standing for any characters other than "(" or ")", the sequences

"XXX(XX(X)XX(XXX))XX", "XX()X((X))" and "(XX)X()"

are correctly parenthesised, whereas

"(XX(X)X", "XX)(X)(" and "XX(X)X)(X"

are not. More precisely, a sequence is correctly parenthesised if, for every "(", there is a matching ")" further along the sequence, and the sequence between these two occurrences is also correctly parenthesised. A sequence with no parentheses at all is defined to be correctly parenthesised.

Give a clear description of your algorithm and a clear explanation of its run-time. You may find it helpful to use small examples, but you should be careful to ensure that your description covers all situations. You may re-use any algorithm presented in the course without the need for further detailed explanation. You should aim for an algorithm which runs in $\Theta(\log\ n)$ time on n processors. [*12 marks*]

(c) Discuss the cost-optimality and scalability of your algorithm. [*5 marks*]

(a)

(i)

A cost optimal solution is one which has the same cost as the best known sequential algorithm. It is a desirable feature since cost optimal algorithms remain faster than sequential algorithms even when scaled down.

(ii)

CRCW models cannot be implemented on real systems with the same complexity that they originally had, since the hardware available would not support this. However EREW are easier to implement, and may be implemented with the same complexity that the algorithm had for the PRAM model.

(b)

Use prefix sum, output 1 for “(“ and -1 for “)”. If sum at any point < 0 we found a mismatch (and sum needs to be 0 on final processor). See solution to question 1.c) in 2012 paper.

(c)

Can be scaled down cost optimally (with p = O(n/log(n))), just output the overall diff/count of parentheses when each processor has n/p parts of the string.

In the n processor form it’s not cost optimal, because cost = O(nlog(n)), whereas sequential case, cost = O(n)