This homework is **ungraded/optional** and meant purely to help you be better prepared for the exams. Feel free to collaborate with your fellow students in the respective *Ed Discussions* post.

Question 1 (10 points)

Parallel Efficiency You have an algorithm, FOO, that you want to run on a distributed memory system. The algorithm takes $2n^{3/2}\tau$ time on a sequential processor, where τ is the time for a single computation. The distributed version is written for p processors and is listed below:

Algorithm: Distributed-FOO

```
1  for i in 1 ... √p:
2    Sequential–Foo(...)
3    MPI_Communicate(...)
```

The communication step takes time $\alpha + \frac{n}{p}\beta$ where α is the latency and β is the inverse bandwidth of the system. Data is evenly distributed across processors.

(a) (2 points) Is this algorithm work optimal? Explain.

(b) (2 points) What is the speedup of this algorithm on p processors?

(c) (3 points) What is the parallel efficiency of this algorithm?

(d) (3 points) What is the isoefficiency function for this algorithm? (Remember: the isoefficiency function is a lower bound on how fast n must grow as a function of p to keep constant efficiency). Explain your answer.

Question 2 (10 points)

Bumper cars. Researchers from the Sociology department are looking for some help doing a simulation. They want to simulate a large city with millions of people. All the people in the city are to ignore one another unless they bump into someone. If they bump into someone, they will then be polite and exchange pleasantries for a minute before resuming. If someone bumps into a conversation then it ends and begins with the two people and the third continues on in a random direction. The researchers want to model this and see how the crowd reacts over time. The modeled city is similar to New York with millions of people spread throughout the city to start.

(a) (4 points) You are tasked with implementing this on the cluster. What sort of software frameworks (OpenMP, MPI or CUDA) from this class would you use for this project? Be specific about why you want to use this in your design as opposed to something else. You might want to think about **Part B** before answering this.

(b) (6 points) Write pseudo code that models how you would implement this simulation if this was a real life task. Many things are going on here that should be considered. Think about how the data and workload will be distributed. This may then influence how you design hit detection to work. Keep in mind that one node may have more people to simulate than another, so how will time slices be handled?

Question 3 (10 points)

Polynomial Evaluation With Horner's Rule

Evaluating a degree n polynomial is the task of solving

$$P(x) = a_{n-1} * x^{n-1} + \cdots + a_2 * x^2 + a_1 * x + a_0$$

for a given list of n coefficients: $A = [a_{n-1}, a_{n-2}, \cdots, a_1, a_0]$ and some specific input value of x.

The iterative method for evaluating is a serial for loop over all the coefficients that runs in $O(n)$ time.

```
# serial polynomial eval in python
def poly_eval(A : List[], x : int):
    x_to_power_degree = 1
    value = 0
    for coeff in A:
        value += x_to_power_degree * coeff
        x_to_power_degree *= x
    return value
```

As is evident, this method runs in $\Theta(n)$ time and is not intended to scale to multiple processors.

Horner's rule however, re-interprets the evaluating a polynomial is as follows:

$$P(x) = a_0 + x(a_1 + x(a_2 + \cdots x(a_{n-1})))$$

This way of "rephrasing" polynomial evaluation looking at the computation differently allows us to parallelize the evaluation to an arbitrary number of processors.

Construct a distributed memory parallel algorithm for P number of processors (that is, all communication between any two ranks must be explicit). Assume that number of coefficients $N >> P$ and that each processor is ranked starting at zero through $P - 1$. Further assume that each rank contains its block of coefficients at the start of the algorithm, i.e. rank 0 contains coefficients $A[0 : N/P]$, rank 1 contains coefficients $A[N/P : 2N/P]$ and that N is divisible by P.

Analyze the asymptotic complexity of your parallel algorithm. Please express your solution as pseudocode or short phrases for each step, do not write actual code for your algorithm.

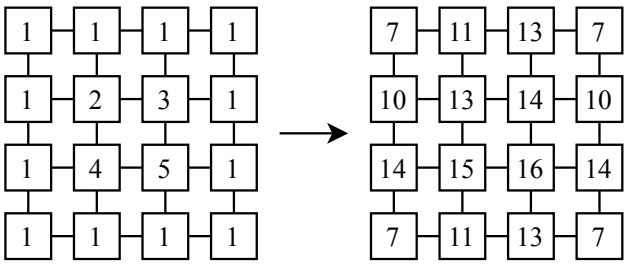
Question 4 (10 points)

Cartesian Reduce

You are tasked with implementing a new kind of MPI reduce, called an *MPI_CartesianReduce*. This reduce is only defined for processors arranged in a d-dimensional grid, and is defined component-wise as follows:

$$\begin{aligned} B[i_1, i_2, ..., i_d] \leftarrow & A[i_1, i_2, ..., i_d] \\ & + \sum_{k=1, k \neq i_1}^{n} A[k, i_2, ..., i_d] \\ & + \sum_{k=1, k \neq i_2}^{n} A[i_1, k, ..., i_d] \\ & + ... \\ & + \sum_{k=1, k \neq i_d}^{n} A[i_1, i_2, ..., k] \end{aligned}$$

In words, this means that each processor ends up with the sum of the elements that vary in exactly one or zero indices. Let's take a look at an example in two dimensions:



We see that each processor ends up with the sum of all of the elements in its row, and all of the elements in its column.

(a) (4 points) Design a distributed memory algorithm to compute the *MPI_CartesianReduce* for a d-dimensional torus, where each dimension has length p, and each processor needs to send n words. Pseudo-code is not required, but you provide enough detail so that your answer to part (b) is clear.

(b) (4 points) What is the time spent on communication, $T(n, p, d)$? Explain.

(c) (2 points) Your friend has designed a new architecture that will allow each node to receive d messages and send d messages at once. How can you use their architecture to write a faster version of this algorithm? How does the communication time change?

Question 5 (10 points)

Uh oh...something is wrong. One of the nice features of MPI is that it includes a lot of primitives that we can use. But what happens if you find out they aren't working correctly? In particular, suppose you're trying to implement an algorithm and find out that *MPI_Barrier()* no longer works.

For this problem, assuming the following pieces of information.

1. The number of nodes is P.
2. The communicator is $comm$.
3. The rank of a processor is $rank$.

(a) (6 points) Give detailed pseudocode for your own *MPI_Barrier()* using point-to-point operations. Your pseudocode should be reasonably detailed, that is, close to being "compileable."

(b) (2 points) What do you think the asymptotic lower bound is?

(c) (2 points) Why does your algorithm work and reach that bound?

Question 6 (10 points)

Dense Matrix Algorithms

(a) (5 points) Consider the matrix-vector multiplication operation, $y = Ax$, where A is an $n \times n$ matrix and x and y are $n \times 1$ vectors. Each output y vector element may be computed as $y[i] = \sum_{j=0}^{n} A[i, j] \cdot x[j]$

Suppose that the matrix is partitioned with 1D *column-wise* partitioning, such that each processor stores $\frac{n}{p}$ complete columns of the matrix A and has matching $\frac{n}{p}$ elements of the vectors x and y, i.e., if a processor holds column j (i.e., $A[*, j]$), then it also owns $x[j]$ and $y[j]$. Design an efficient distributed memory parallel algorithm to compute matrix-vector multiplication in parallel. Analyze the run-time of this algorithm and specify the computation and communication time.

(b) (5 points) Let A be an $n \times n$ matrix. The transpose of A, denoted A^T is defined as $A^T[i, j]$ = $A[j, i]$. Design a parallel algorithm to compute A^T from A using an $n \times n$ virtual torus topology. During the execution of the algorithm, each processor can hold at most a constant number of matrix elements at a time (can be greater than one). Specify the total runtime for your algorithm.

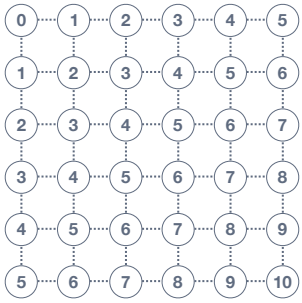


Figure 1: Results of a breadth-first search on a 2-D grid graph. The source is the upper-leftmost vertex. Each vertex is labeled by its level as computed by the BFS.

Question 7 (10 points)

BFS cache analysis. Suppose you run a sequential breadth-first search (BFS) on a two-dimensional (2-D) *grid graph* with n vertices. An example appears in fig. 1, where the $n = 36$ vertices are laid out on a 2-D grid with regular spacing, and each vertex is connected by edges to its north, south, east, and west neighbors. (Though the example is square, do **not** assume a square grid graph in this problem.) If you start from the upper-leftmost vertex and label each unseen vertex by its level during the search, you will get the labels shown in fig. 1.

In the following exercises, you will analyze the cache behavior of BFS on grid graphs for a machine with a two-level memory hierarchy. Let Z be the size of the cache and L be the line size. Furthermore, assume that the vertices in a row are laid out consecutively in memory, and *ignore* the storage cost of edges. That is, suppose we don't need to store edges at all and that you can determine location in memory of any neighbor of a vertex in $\mathcal{O}(1)$ time.

(a) (5 points) For what sizes n would you expect good cache behavior? "Good" means no thrashing of cache lines due to insufficient capacity. Do **not** assume anything about the order in which the algorithm visits the neighbors of a vertex. Explain your reasoning.

(b) (5 points) Suppose the graph is, instead, a **three-dimensional** (3-D) grid graph, again with n vertices. For what value of n would you expect to see cache thrashing due to a lack of capacity?

Question 8 (10 points)

Distributed 3D fast Fourier transforms (FFTs). Consider a supercomputer that can execute up to $R_{\max}$ operations per unit time. (That is, an embarrassingly parallel program that performs no communication and moves no data through the memory hierarchy will execute at this rate.) Further suppose that the architecture of this machine is a distributed memory network of P nodes connected by a 3D torus. Let the time to send a message of size m words between any two nodes, in the absence of congestion, be m/β_{link}, where β_{link} is the bandwidth (words per unit time) of a network link. (That is, we simplify our usual (α, β) model by *ignoring* any latency term, α.) Within each node, suppose there is a two-level memory hierarchy with a fast memory of size Z and main memory bandwidth of β_{mem} words per unit time. Lastly, assume that the node itself can execute up to R_0 operations per unit time, so that its balance (within the node) is $B_0 \equiv \frac{R_0}{\beta_{\text{mem}}}$ ops per word.

Suppose you wish to compute a 3D FFT. Here are its main performance characteristics.

- Let n^3 denote the size of the input. The output will be of the same size.
- The total work (e.g., arithmetic operations) is $\mathcal{O}\big(n^3 \log n\big)$.
- Let the parallel execution time be $T \geq \max\{T_{\text{comp}}, T_{\text{mem}}, T_{\text{net}}\}$, where T_{comp} is the time to perform just the arithmetic operations; $T_{\text{mem}} = T_{\text{mem}}(n; Z, \beta_{\text{mem}})$ is the time spent on local (intranode) communication between slow and fast memory; and $T_{\text{net}} = T_{\text{net}}(n; \beta_{\text{link}}, P)$ is the time spent on network communication. This equation says the running time is, in the best case, the largest of these three components, assuming arithmetic, local data movement, and network communication can all be completely overlapped.
- It is possible to show that $T_{\text{mem}} = \mathcal{O}\big(\frac{n^3 \log n}{P \beta_{\text{mem}} \log Z}\big)$ and $T_{\text{net}} = \mathcal{O}\big(\frac{n^3}{P^{2/3} \beta_{\text{link}}}\big)$.

With this background, answer the following questions.

(a) (1 point) Assume the arithmetic work is perfectly parallelizable. Calculate T_{comp}, in a big-O sense.

(b) (2 points) Next, show that $T_{\text{mem}} = \mathcal{O}\big(\frac{B_0}{\log Z} \cdot T_{\text{comp}}\big)$.

(c) (2 points) Define "network node balance" for a 3D FFT as $B_{\text{net}} \equiv \frac{R_0^{2/3}}{\beta_{\text{link}}}$. Ignoring the $2/3$ exponent for the moment, briefly explain what this quantity tells you about a system.

(d) (2 points) Derive an expression for T_{net} in terms of B_{net}.

(e) (3 points) Assume that communication dominates, i.e., $T_{\text{comp}} \ll T_{\text{mem}}$ and $T_{\text{comp}} \ll T_{\text{net}}$ for most supercomputers you can build from today's parts. Now suppose you have a choice of two supercomputers with the same $R_{\max}$. One uses high-end server-grade processors and the other uses processors more commonly found in mobile phones. Which system is likely to be faster, and why?

Question 9 (10 points)

Six Degrees of HPC. Last semester, you worked closely with the sociology department to model some behavior in New York City. People were randomly placed in the city and then set in motion. If they happened to collide with someone then the two people would stop and talk to each other and then move on after a few seconds. Since it is summer, the sociology department wants to do some low key additions to this project. When the two people bump and talk, they also get to know each other thus making a connection. Your colleagues in the sociology department are interested to see how this network grows over time. Specifically, they want to see how many people lie within the 6 degrees of separation from one another. It should be noted that this is all simulated using several nodes on the cluster communicating using MPI.

(a) (6 points) Write **pseudocode** that determines how many people in the simulation lie within six degrees of separation from each other.

(b) (2 points) What is the **work** and **span** for your proposed solution?

(c) (2 points) What other aspects of the design might influence performance?

Question 10 (10 points)

Red vs Blue. Consider n balls distributed evenly across p processor nodes, each colored either ***RED*** or ***BLUE***. We want to ensure that each continuous sequence of r balls has at least $\underline{s}$ ***BLUE*** balls (assume $n > r > s$).

(a) (8 points) Design a parallel **algorithm** to check if the condition is *true* for every continuous sequence of length r.

(b) (2 points) **Analyze** your algorithm separately in terms of both **work** and **span** complexities.

Question 11 (10 points)

VeryOP. Let's start by defining a new reduction operation primitive and calling it op.

Now imagine we have a p processor system where each processor node has an array of size p stored local to itself. If we perform our reduction operation primitive, op, on our p processor system, the processor node at the i^{th} rank will have the reduction of all the i^{th} elements of all the processor nodes' arrays.

Design an efficient parallel algorithm for this reduction operation primitive and then compute its computation cost/runtime complexity.

(a) (8 points) Provide your parallel algorithm design.

(b) (2 points) Find the **work** and **span** of the algorithm.

Question 12 (10 points)

RANKrank. Let's start with our parallel **SampleSort** algorithm.

We intend to feed our algorithm n elements such that all of these elements are distinct from one another and they are *globally* in decreasing order with respect to the order of our processor nodes. Our algorithm is going to utilize p processor nodes and each processor will initially have $\frac{n}{p}$ elements locally stored in them (assume n is divisible by p).

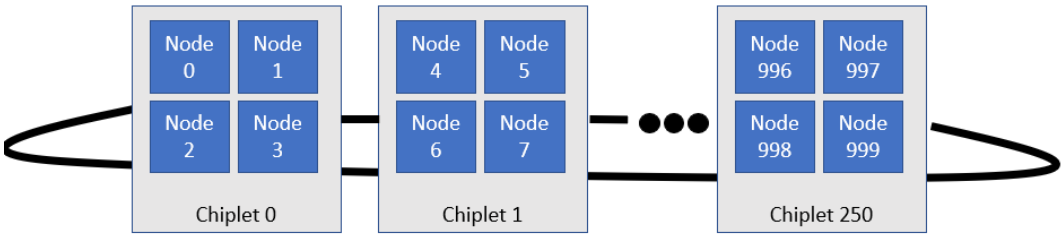
Our **SampleSort** is run to sort our input elements in *ascending* (increasing) order. Now Compute the number of input elements in each processor, after running our algorithm, as a function of the processor rank (i.e. i).

(a) (10 points) As mentioned, **compute** the number of input elements per processor (as a function of/based-on the processor rank/number) and provide **detailed** steps for doing so.

Question 13 (10 points)

Communication Madness. We studied several distributed memory topologies such as the ring, 2-D mesh, hypercube, and fully connected. Of course in real world supercomputers (like our PACE cluster), the actual topology is hybrid, complicated, and messy.

Consider a machine with P=1000 nodes, where each a node is a processor on a chiplet of four CPUs. Within any one chiplet, the four processors can communicate with extremely low latency and high bandwidth, call it α_{chip} and β_{chip}. However, all the four node chiplets are connected in a ring network (250 total chiplets) and those inter-chiplet connections have regular $\alpha + \beta n$ cost.



Your task: Outline, in high-level pseudocode, a personalized all-to-all messaging algorithm that is efficient for this topology.

(a) (6 points) **Describe your algorithm** using high-level **pseudocode**. That means, no variable declarations or C syntax; just provide sufficient detail to understand your approach.

(b) (4 points) **Analyze** the efficiency of your algorithm in terms of alpha and beta model, but using α_{chip} and β_{chip} to note intra-chiplet communications. Remember that:
$\beta_{chip} << \alpha_{chip} << \beta << \alpha$
Hint: you can just write AlphaChip, BetaChip, Alpha etc. in Canvas as text.

Question 14 (10 points)

Projection. You have learned about the 2D distributed Matrix Multiplication algorithm in this class. In practice, instead of computing and forming new matrices explicitly, the matrix as an operator could be applied to data in multiple steps.

Consider the operation AA^Tx, where matrix A is of size $n \times k$, and x a vector of size n, and A^T denotes A's transpose.

In the real world $n >> k$ so we will assume k is a constant, and the problem size is n.

(a) (1 point) What is the **optimal** amount of floating point operations required to complete this computation in serial? Give the precise number in terms of n and k, **do not use big-O notation**.

(b) (6 points) Now we have p distributed processors and p is a perfect square. **Describe a parallel algorithm** to compute AA^Tx.

You may arrange p processors in an arbitrary grid/topology if you would like, and partition/store A in the processes in any way you want. Do not sweat about optimality, but clearly state the partition of matrix, process network layout, and use communication primitives like *send, recv, broadcast*, etc. you learned from MPI.

Analyze runtime of your algorithm $T(n, p)$ with respect to problem size n and available workers p.

(c) (3 points) Now suppose we have an algorithm that runs in $T(n, p) = \Theta(\frac{nk}{p} + k \log p)$, what is the largest value of p that maintains the optimal $\Theta(1)$ efficiency?

Question 15 (10 points)

SortAndSolve. This question has two parts. The first is meant to serve as motivation and help for the second.

(a) (4 points) **Show the steps** for bitonic sorting the following 8-elements in non-increasing order. You can for example use lines to denote comparison operations- but there are no hard requirements as long as the correct comparisons are mentioned. You can also for example label the comparison operations as $\uparrow$ or $\downarrow$ where the direction of your arrow indicates the destination of the smaller element (or bigger element but make sure to mention if not following the default assignment). Illustrate how the following input is sorted using a text diagram:

$$(10, 14, 9, 6, 17, 3, 8, 4)$$

NOTE: Here's an example of what to write and what to mention (position of the comparisons in this example are totally random) given the constraints of *Canvas*.

Step 0: $5, 6, 9, 23, 1, 2$

Step 1: $5, 23, 9, 6, 2, 1$ (23 compared with 6, 2 with 1) (or you can use index starting with 0)

Step 2: $23, 5, 9, 6, 2, 1$ (23 compared with 5)

(b) (4 points) Now **provide an efficient parallel algorithm** to merge two sorted sequences of lengths m and n, respectively. You may assume that the input is an array of length $m + n$ with one sequence followed by the other, distributed across processors such that each processor has a sub-list of size $\frac{m+n}{p}$.

(c) (2 points) What is the **work** and **span** of your algorithm? (*You may assume the use of any permutation style communication, not necessarily hyper-cubic for example.*)

Question 16 (10 points)

Parallel Distributed Scan. As we have seen, the general running time $T_P(n)$ for a parallel scan is $O(\frac{n}{P} + \log^2 P)$ if we have fewer than $O(n)$ processors. This algorithm exhibits strong scalability - whenever $\frac{n}{P} \geq \log^2 P$, the speedup is linear in P. However, there are practical limitations to the number of processors we can fit on a single node. If we want to get a 500x or 1000x speedup, we have to expand to more than one node to do so. How does parallel distributed scan perform on multiple nodes?

(a) (4 points) **Design a parallel add-scan algorithm** to be run on a system with P_d *distributed* nodes and P_l processors *local* to each node. The system is connected via a 2D grid interconnect. **Do not provide pseudocode**.

(b) (3 points) What are the **computation** and **communication** times of your algorithm? Provide your answers in terms of n, P_d, P_l, and, where appropriate, α and β.

(c) (2 points) **What is the isoefficiency function** of your algorithm? How does it compare to **scalability** of the PRAM add-scan algorithm mentioned above? For what relationship(s) between P_d and P_l does it approach the PRAM add-scan? *Helpful hint: when computing efficiency from speedup, note that your system has $P_d \cdot P_l$ total processors*.

(d) (1 point) What if your algorithm needed to be run on a linear network? How would it change?

Question 17 (10 points)

BeastMode Returns! You're working for a big social networking company, *Squawker*. It has an enormous dataset spread across a thousand processor nodes consisting of *unordered* messages of the form: keyword, squawk content. Example:

"#OMSCS","HPC is the best class!"
"#MOVIES","I wish Indiana Jones stopped at three"
"#ELON","I hope he doesn't buy Squawker"
"#OMSCS","I really hated CN"

In this example, "#OMSCS" is the most common keyword.

You have been told to efficiently find the most common keywords(s) distributed across the huge dataset over P nodes. There is only one keyword per message, but there are at least 10 million different keywords in use. There may turn out to be more than one "most common value".

(a) (6 points) Using **pseudocode** and referencing any MPI functions you wish, describe an **efficient** solution. The output should be a list of one or more most common keywords.

(b) (4 points) Assuming a fully-connected network, analyze the communications complexity of your approach in terms of α and β.

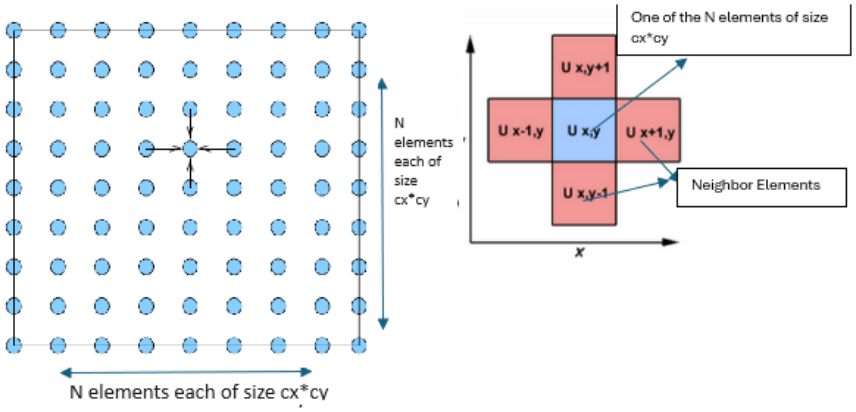


Figure 2: Heat Diffusion Calculation Setup - $U(x, y)$ represents temperature of element x, y

Question 18 (10 points)

Analyzing a distributed memory problem. Let us consider the problem of simulating heat diffusion on a 2-D surface of N elements in each dimension. Lets say cx (constant) is the size of each element in x-dimension and cy (constant) is the size of the element in y-dimension. The heat equation calculates the change in temperature over a period of time. The initial temperature and boundaries are given. Lets assume that $t1$ & $t2$ are two consecutive time steps having temperatures $U1$ & $U2$. The resulting heat equation is given as:

$$U2_{x,y} = U1_{x,y} + cx * (U1_{x+1,y} + U1_{x-1,y} - 2 * U1_{x,y}) + cy * (U1_{x,y+1} + U1_{x,y-1} - 2 * U1_{x,y})$$

The calculation of temperature of an element is dependent upon the neighbor element temperature values at the prior time step.

Let us consider analyzing a distributed memory algorithm to solve this problem.

(a) (2 points) Lets start with partitioning the data (elements of the 2-D surface) across the processors of the distributed memory setup. Assuming that the size of the element is 1 (cx, cy are both 1) and there are N elements in each dimension (x & y) and there are P processors to do the heat diffusion simulation, can you **provide the number of elements in each processor** in a simple 1-D and 2-D partitioning of data?

(b) (5 points) Assuming the same setup as **part (a)**, Can you come up with an **estimate of the computation and communication times for one time step** of temperature calculation of all $N * N$ elements for both the partitioning schemes? Explain the size of the communication block that you are using for each partitioning scheme.
Hint: You need to consider the number of elements that have to communicated per process to other processes given the partitioning scheme and the heat diffusion equation

(c) (3 points) What will be the **isoefficiency** of the 1-D partitioning scheme and what can you conclude from it?
Hint: In the serial solution of the heat diffusion equation stated above, we will just have two outer loops - one across x axis and one across y - axis