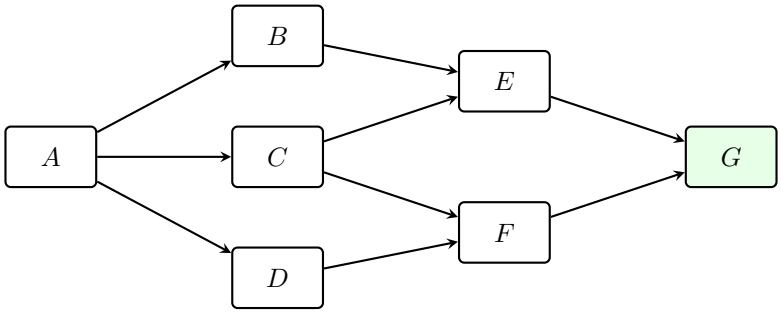
Note: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Parallel Algorithms Warmup

Consider the following computation DAG. Each node takes 1 time unit of work. An edge means the destination node cannot begin until the source node finishes.



(a) Compute the total work T_1, the time to finish everything if we only have one processor.

(b) Compute the span T_∞, the time to finish everything if we have an unlimited number of processors.

(c) Suppose we have $p = 3$ processors. Use Brent's rule

$$T_p \leq \frac{T_1}{p} + T_\infty$$

to upper bound the runtime T_3. Then use Brent's rule to estimate the optimal number of processors for this DAG.

(d) Give a schedule using the optimal number of processors.

2 Parallel Balanced Binary Tree

Consider an algorithm that processes an array of size n (where n is a power of 2) by structuring the computation as a perfectly balanced binary tree. The leaves of the tree represent the input elements. Each internal node requires $O(1)$ time to compute a result based on its two children.

(a) Formulate the dependency DAG for this algorithm and determine the exact asymptotic bounds for T_1 and T_∞.

(b) Using Brent's rule, determine the theoretical minimum number of processors p required to achieve an asymptotic runtime of $O(T_\infty)$. Show your work.

3 Work-Efficient Parallel Merging

Standard mergesort splits an array in half, sorts each half, and merges them. In a parallel setting, making the recursive calls in parallel is easy, but the merge step often becomes the bottleneck if done sequentially.

Let A and B be two sorted arrays of length n. We want to merge them into a single sorted array C of length $2n$.

Recall that a parallel algorithm is *work-efficient* if its total work matches the asymptotic runtime of the best sequential algorithm for the same problem.

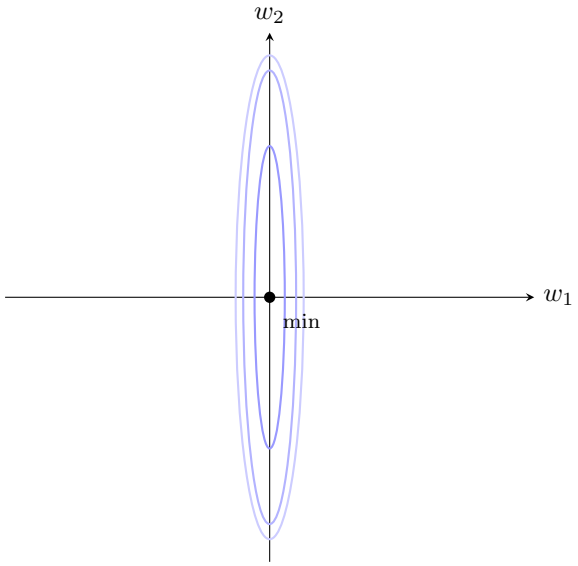
(a) Naive parallel merge: suppose we assign one processor to each element in A and B. Each processor uses binary search on the other array to find exactly how many elements are smaller than its assigned element, thereby calculating its exact final index in C. What are the work T_1 and span T_∞ of this naive approach? Is it work-efficient?

(b) Propose a work-efficient parallel algorithm for merging A and B. Then write down the recurrence relations for both the work $T_1(n)$ and the span $T_\infty(n)$ of your algorithm, and solve them to give the final asymptotic bounds.

4 Adaptive Gradient Descent

Consider a 2D quadratic cost function representing a deep, narrow valley:

$$f(w_1, w_2) = 50\, w_1^2 + \tfrac{1}{2}\, w_2^2.$$



(a) Write out the standard gradient descent update rules for w_1 and w_2 using a single, shared scalar learning rate η.

(b) What is the absolute maximum value η can take before the w_1 parameter strictly diverges?

(c) Suppose you use a "safe" learning rate just below the divergence limit, say $\eta = 0.01$. If both w_1 and w_2 start at 1.0, approximately how many iterations does it take for each to shrink by half?

(d) You are trapped: a learning rate large enough to optimize w_2 quickly will cause w_1 to explode. Propose a mathematical modification to the standard gradient descent update rule

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla f(\mathbf{w}^{(t)})$$

that solves this problem, allowing fast convergence in both directions simultaneously.

