Problem 1. (15 points) [Culler, Singh, & Gupta 1.3] Generally, in evaluating performance tradeoffs, we evaluate the improvement in performance, or speedup, due to some enhancement. Formally,
\[
\text{Speedup due to enhancement } E = \frac{\text{Time without } E}{\text{Time with } E} = \frac{\text{Performance with } E}{\text{Performance without } E}.
\]

In particular, we will often refer to the speedup as a function of the machine parallelism (e.g., the number of processors).

Suppose you are given a program that does a fixed amount of work, and some fraction \( s \) of that work must be done sequentially. The remaining portion of the work is perfectly parallelizable on \( P \) processors. Assuming \( T_1 \) is the time taken on one processor, derive a formula for \( T_P \), the time taken on \( P \) processors. Use this to get a formula giving an upper bound on the potential speedup on \( P \) processors. (This is a variant of what is often called Amdahl’s Law.) Explain why it is an upper bound.

Problem 2. (25 points) (a) Given an \( n \times n \) matrix \( A = (a_{ij}) \), we want to find the \( n \) column sums:
\[
S_j = \sum_{i=0}^{n-1} a_{ij} \quad \text{for } j = 0, 1, \ldots, n-1
\]

using an SIMD machine with \( n \) PEs. The matrix is stored in a skewed format, as shown below. Note that each row is shifted right circularly by one element from the previous row. The \( j \)th column sum \( S_j \) is stored in location \( \beta \) in memory \( j \) at the end of the computation. Use the instruction set defined in Lecture 3 to write an assembly-language program to compute these column sums.

Assume the array computer has an ASHIFT instruction which does either a one-place left circular shift (ASHIFT L) of the values in the accumulators; that is, the value in ACC[1] moves to ACC[0], ACC[2] moves to ACC[1], \ldots, and ACC[0] moves to ACC[\( N-1 \)], or a one-place right circular shift (ASHIFT R). Assume also that each \( k \)th accumulator has a private index register LINEX[k], and that there is an instruction LOADI that loads the index registers from memory, just like LOAD loads the accumulators from memory. Also, ISHIFT does the same thing as ASHIFT, except that it shifts the index registers.

Your program should access all the memories concurrently.
(b) Rewrite the program to calculate row sums:

\[ S_i = \sum_{j=0}^{n-1} a_{ij} \text{ for } i = 0, 1, \ldots, n-1 \]

**Problem 3.** *(15 points) [Culler, Singh, & Gupta 1.8]* Consider a simple 2D finite-difference scheme where, at each step, every point in the matrix is updated by a weighted average of its four neighbors,


All the values are 64-bit floating-point numbers. Assuming one element per processor and 1024 \( \times \) 1024 elements, how much data must be communicated per step? Explain how this computation could be mapped onto 64 processors so as to minimize the data traffic. Compute how much data must be communicated per step.

Note that, in order to implement this, each processor must send data to the processor that is logically “below” it, to the processor that is “above” it, to the processor on the “left” and the processor on the “right,” in four separate steps. Whether or not the processors are actually arranged in a 2D mesh is not important, since in modern multiprocessors it takes essentially constant time to send data anywhere within the processor array.

**Problem 4.** *(25 points)* Consider how a four-PE array processor can be used to multiply two 3 \( \times \) 3 matrices. The interconnection structure is shown in the diagram at the right. There are “wraparound” connections in both rows and both columns of the array. This permits one data element to be shifted, for example, from the top processor of some row to the bottom processor, at the same time that another element is moving from the bottom processor to the top.

All three multiplications needed to compute each output element \( c_{ij}, (i = 1, 2, 3; j = 1, 2, 3) \) must be performed in the same PE to allow the products to be summed without extra transfers.

(a) Show the initial mapping of the \( A \) and \( B \) matrix elements to the processors. Initially, the rows of \( A \) and the columns of \( B \) should be distributed among the processors in such a way that all elements of a row of \( A \) reside on the same processor, and all elements of a column of \( B \) reside on the same processor. Since there are a total of six rows of \( A \) and columns of \( B \), obviously some processors will contain more elements than others.

(b) What multiplications are carried out in each processor (perhaps more than one multiplication per processor) before any array elements are shifted?

(c) Shifts are performed in the horizontal and vertical directions to move the matrix elements into place for the remaining multiplications. Show the mapping of the elements of \( A \) and \( B \) to the processors before the second group of multiplications is carried out.
(d) What multiplications are now carried out in each processor? You should be able to complete the matrix multiplication without any further shifts.

(e) What is the total time needed to multiply the matrices, in terms of multiplications, adds, and shifts? Since each of the four processors is sequential, it can multiply, shift, or add a single element in one time unit. However, since the “wraparound” links are bidirectional, it can send one element to an adjacent processor at the same time that it is receiving an element from the other processor.

**Problem 5.** (10 point) [Culler, Singh, & Gupta 1.10] Show that Equation 1.4 (p. 60) follows from Equation 1.3.

**Problem 6.** (10 point) [Culler, Singh, & Gupta 1.13] Suppose a 32-byte cache line is transferred to another processor, and the communication architecture imposes a start-up cost of 2 μs and a data-transfer bandwidth of 20 MB/s. What is the total latency of the remote operation?