Problem 1. (15 points) This problem will explore the effect on speedup when the number of processors available to us can vary over time. Let’s envision a parallel architecture with a large (effectively infinite) number of processors and a very large shared memory; every processor will be capable of communicating with any part of the shared memory. Instead of considering the communication time separately, for the sake of simplicity, assume that communication time is included with the serial portion of the algorithm.

We will consider a task that can be recursively broken down, using a divide-and-conquer algorithm. Solving the task on one processor takes time $T = 1$, as usual. However, the problem can be solved on $n$ processors by splitting the parallelizable portion of the problem in half and having more processors take the work. So, with one processor,

$$T = s + (1-s)$$

On two processors we have the serial work, a split, then another serial section to re-assemble the answer to the problem:

$$T = \frac{s + (1-s)}{2} + s$$

We can half the problem with each iteration, doubling the number of processors at our disposal. Each time we do this, we incur an extra step of re-assembly that must be performed in serial.

(a) Is there a maximum or a limit to the achievable speedup? Derive the formula.

(b) Now assume that we can fragment the task by more than two with each step. Will this affect the maximum achievable speedup?

(c) Comment on the efficiency of this approach.

Problem 2. (15 points) Below is a section of code that subtracts two $N \times N$ matrices, and sets any negative values in the results to zero. The pseudocode for this is as follows:

```plaintext
for (i = 1 to n) do
    for all C[i] < 0 set C[i] = 0
end for
```

Using the array-processor instructions from Lecture 3 to set appropriate mask bits, implement the pseudocode above. You may assume that there is a row of size $n$ containing all –1’s loaded into location $P$ in memory, and a row of all 0’s into row $Q$ in memory.

Problem 3. (25 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 \( \mu \)s and a data-transfer bandwidth of 20 MB/s. What is the total latency of the remote operation?