

## CSE 512/CS 554 Midterm Exam — Solutions

1. Amdahl's Law was originally formulated for a fixed problem size, but Amdahl's result can be used to characterize the behavior of the serial fraction  $s$  as a function of the number of processors  $p$  as the problem size grows with  $p$ . According to Amdahl's Law, how would  $s$  have to vary in order to maintain each of the following conditions? Your answers should express  $s$  as a function of  $p$ .

(a) Speedup of  $p/2$ .

*Answer:* According to Amdahl's Law, if  $s$  is the serial fraction, then the parallel time  $T_p$  is given by

$$T_p = sT_1 + (1 - s)T_1/p,$$

and hence the speedup  $S_p$  is given by

$$S_p = T_1/T_p = p/(sp + (1 - s)).$$

If we set Amdahl's formula for  $S_p$  equal to  $p/2$  and solve for  $s$ , we obtain

$$s = 1/(p - 1).$$

(b) Speedup of  $p - 1$ .

*Answer:* If we set Amdahl's formula for  $S_p$  equal to  $p - 1$  and solve for  $s$ , we obtain

$$s = 1/(p - 1)^2.$$

(c) Constant efficiency.

*Answer:* According to Amdahl's Law, the efficiency is given by

$$E_p = T_1/(pT_p) = 1/(sp + (1 - s)).$$

Setting the right hand side to a constant, say  $E$ , and solving for  $s$  gives

$$s = ((1/E) - 1)/(p - 1).$$

Thus, as  $p$  grows,  $s$  must diminish proportionally to  $1/(p - 1)$ .

2. In partitioning a problem for parallel solution, for best scalability should the *size* of each task or the *number* of tasks grow as the problem size increases? Why?

*Answer:* For best scalability, the *number* of tasks should grow as the problem size increases, rather than the size of tasks. One reason is that more tasks are needed in order to utilize more processors. If the size of tasks were growing, on the other hand, then the overall execution time would necessarily increase regardless of how many processors were used, and the memory capacity of individual processors would eventually be exceeded.

3. For a given parallel algorithm for solving a given problem on a given parallel architecture, what would typically be the effect (*increase* or *decrease*) on *parallel efficiency* of each of the following changes, assuming all other independent parameters are held fixed.

(a) Increase number of processors

*Answer:* decrease

(b) Increase size of problem

*Answer:* increase

(c) Increase communication bandwidth

*Answer:* increase

(d) Increase communication latency

*Answer:* decrease

(e) Increase computing speed of processors

*Answer:* decrease

(f) Increase number of communication ports on which each processor can communicate simultaneously

*Answer:* increase

4. (a) In MPI, what is the specific fundamental distinction between blocking and nonblocking communication functions?

*Answer:* In MPI, resources specified in the call to a blocking communication function, such as the message buffer, can safely be reused immediately upon return. For example, a blocking receive returns only after the receive buffer contains the requested message. With a nonblocking communication function, on the other hand, the resources specified in the call cannot safely be reused until completion of the requested operation has subsequently been confirmed explicitly using a separate function call to test or wait for completion. A non-blocking communication function initiates a communication operation, but does not complete it.

(b) What are the advantages and disadvantages of each of these two types of communication functions compared with the other?

*Answer:* Blocking communication functions have the advantage of automatically providing synchronization, whereas synchronization must be explicitly managed by the programmer with nonblocking communication. Blocking communication functions require no additional copying or buffer space, since messages can be transferred directly to and from memory locations in the user's code, whereas nonblocking communication functions require copying messages

to and from either system- or user-supplied buffer space. Nonblocking communication functions enable much more aggressive overlapping of communication with computation than blocking communication functions permit.

5. To compute the matrix-matrix product  $\mathbf{C} = \mathbf{AB}$ , where  $\mathbf{A}$ ,  $\mathbf{B}$ , and  $\mathbf{C}$  are  $n \times n$  matrices, using a 2-D parallel algorithm on a  $\sqrt{p} \times \sqrt{p}$  processor grid, what is the *total* storage (in words) per processor required for each of the following algorithms? You should include storage for the input matrices, message buffers, and result matrix. You may assume that  $p$  is a perfect square and that  $\sqrt{p}$  divides  $n$ .

(a) Straightforward algorithm using broadcasts along both rows and columns of processors

*Answer:* With any algorithm, each processor must store its own portions of  $\mathbf{A}$ ,  $\mathbf{B}$ , and  $\mathbf{C}$ , each of which is of size  $n^2/p$ . In addition, for this algorithm each processor requires enough buffer space to store the portions of  $\mathbf{A}$  belonging to the other  $\sqrt{p} - 1$  processors in its row and the portions of  $\mathbf{B}$  belonging to the other  $\sqrt{p} - 1$  processors in its column. Summing all these, the total storage required per processor is  $(3 + 2(\sqrt{p} - 1))n^2/p = (2\sqrt{p} + 1)n^2/p$ .

(b) Fox's algorithm (broadcasts along one dimension and ring circulation along the other)

*Answer:* Circulating submatrices in ring fashion along one dimension of the processor grid requires a single message buffer of size  $n^2/p$  in each processor. Thus, the total storage required per processor is  $(4 + (\sqrt{p} - 1))n^2/p = (\sqrt{p} + 3)n^2/p$ .

(c) Cannon's algorithm (ring circulation along both dimensions)

*Answer:* Circulating submatrices in ring fashion along both dimensions of the processor grid requires two message buffers, each of size  $n^2/p$  in each processor. Thus, the total storage required per processor is  $5n^2/p$ .

6. Consider 1-D partitioning versus 2-D partitioning of the 2-D fine-grain task array in implementing parallel LU factorization by Gaussian elimination without pivoting.

(a) Which of these two approaches requires more messages?

*Answer:* 2-D partitioning requires twice as many messages as 1-D partitioning.

(b) Which of the two requires more total volume of communication?

*Answer:* 1-D partitioning requires more total volume of communication than 2-D partitioning.

(c) How do the orders of magnitude of the resulting isoefficiency functions compare for the two approaches, implemented on a 1-D or 2-D mesh of processors, respectively? (You can cite this from memory or derive it, if necessary.)

*Answer:* 2-D partitioning has an isoefficiency function of  $\mathcal{O}(p^{3/2})$ , whereas 1-D partitioning has an isoefficiency function of  $\mathcal{O}(p^3)$ .