8.1 *Consider the two algorithms for all-to-all personalized communication. Which method would you use on a 64-node parallel computer with $\Theta(p)$ bisection width for transposing a 1024×1024 matrix with 1-D partitioning if $t_s = 100\mu s$ and $t_w = 1\mu s$? Why?*

Recall that there were two hypercube based all-to-all personalized communication algorithms. One proceeded in $\log_2 p$ steps and the other proceeded in $p - 1$ steps. The $\log_2 p$ step algorithm had $\log_2 p$ startup costs, but worse overall cost. Thus, for small messages, the $\log_2 p$ algorithm may be faster when the t_s associated costs dominate the overall execution time.

When transposing a matrix using 1-D partitioning we can use a all-to-all personalized broadcast algorithm. If the 1024×1024 matrix is partitioned to 16 processors, then each processor will contain a matrix of 16×1024. Each 16×16 block of this matrix will be a message in the all-to-all personalized communication. The size of this message will be 256 words. For this example, t_s is 100 times greater than t_w. With a message size of 256, t_s is not clearly dominate in the communication time of the optimal broadcast algorithm. However, the only way to find out which case is optimal is to compute: The hypercube store and forward algorithm execution time is

$$t_p = (t_s + t_w mp/2) \log_2 p = (100 + 1 * 256 * 64/2) * \log_2(64) = 49752\mu s, \quad (1)$$

while the ecube routing algorithm execution time is

$$t_p = (t_s + t_w m)(p - 1) = (100 + 1 * 256)(64 - 1) = 22428\mu s. \quad (2)$$

Clearly the all-to-all personalized using ecube routing is preferred for this case.

8.2 *Describe a parallel formulation of matrix-vector multiplication in which the matrix is 1-D block partition along the columns and the vector is equally partitioned among all the processes. Show that the parallel run time is the same as in the case of row-wise 1-D block partitioning.*

For column partitioning, we have data partitioned to processors such that a matrix-vector multiplication can proceed immediately without any communications step. The resulting vector distributed among processors must be accumulated to achieve the final result. Note, that this operation can be performed by using the dual of the all-to-all broadcast, the all-to-all reduction. The all-to-all reduction can be performed in time

$t_{comm} = t_s \log p + (t_w + t_{add})m(p-1)$

Note that, the message size is equal to the number of elements of the vector assigned to each processor, which is n/p. Thus the communication cost is similar (with the exception of the cost of adding). The computation cost, which occurs first in the computation, is the same as the row partitioning, n^2/p Thus the total time for the matrix-vector multiply with row-wise partitioning is

$t_p = n^2/p + (t_w + t_{add})(n/p)(p-1)$

Thus the time for row decomposition is the same as column decomposition except for the t_{add} term. However, if we accurately counted the number of adds in the local computation, we would find that there are $p-1$ adds that were over counted in the current formulation. Thus both algorithm take the exact same amount of time.

8.4 *The overhead function for multiplying an $n \times n$ 2-D partition matrix with an $n \times 1$ vector using p processes is $t_s plogp + t_w n\sqrt{p}\log_2 p$. Substituting this expression in equation 5.14 yields a quadratic equation in n. Using this equation, determine the precise isoefficiency function for the parallel algorithm and compare it with Equations 8.9 and 8.10. Does this comparison alter the conclusion that the therm associated with t_w is responsible for the overall isoefficiency function of this parallel algorithm?*

Substituting as directed, we get

$$W = n^2 = Kt_s p \log p + nKt_w\sqrt{p}\log p \tag{3}$$

This gives us the quadratic equation

$$Kt_s p \log p + nKt_w\sqrt{p}\log p - n^2 = 0 \tag{4}$$

Solving the quadratic equation (selecting the largest solution) we get (e.g. $a = -1, b = Kt_w\sqrt{p}\log p, c = Kt_s p \log p$ where $W = n^2 = 1/4(b + \sqrt{b^2 + 4c})^2 = 1/2b^2 + c + 1/4b^2\sqrt{1 + 4c/b^2}$.

$$W = n^2 = \frac{1}{2}K^2t_w^2 p\log^2 p + \frac{1}{4}K^2t_w^2 p\log^2 p\sqrt{1 + \frac{4t_s}{Kt_w\log p}} + Kt_s p\log p \tag{5}$$

In particular, if we look at the second term in this equation, we find a term containing $\sqrt{1 + \frac{4t_s}{Kt_w\log p}}$ which is a function that in the limit as p grows large approaches 1. Thus we can simplify the above expression when p is sufficiently large as

$$W = \frac{3}{4}K^2t_w^2 p\log^2 p + Kt_s p\log p \tag{6}$$

From this we can see confirmation of the isoefficiency analysis given in equations 8.9 and 8.10. Both terms appear. Similarly, the first term is will dominate. In addition, due to the squaring of K and t_w, it is likely to dominate for relatively small p.

8.5*Strassen's method for matrix multiplication is an algorithm based on the divide-and-conquer technique. The sequential complexity of multiplying two $n \times n$ matrices using Strassen's algorithm is $\Theta(n^{2.81})$. Consider the simple matrix multiplication algorithm for multiplying two $n \times n$ matrices using p processes. Assume that the $n/\sqrt{p} \times n/\sqrt{p}$ sub-matrices are multiplied using Strassen's algorithm at each process. Derive an expression for the parallel run time of this algorithm. Is the parallel algorithm cost-optimal?*

The communication time for the simple algorithm will not change. This is given as:

$$t_{comm} = \Theta(\log p) + \Theta(\frac{n^2}{\sqrt{p}}) \tag{7}$$

However, the computation time will be given by the time to perform $\sqrt{p}$ multiplications using Strassen's algorithm, thus the computation time is given by

$$t_{comp} = \sqrt{p} \times \Theta((\frac{n}{\sqrt{p}})^{2.81}) = \Theta(\frac{n^{2.81}}{p^{0.905}}) \tag{8}$$

The overall parallel time is

$$t_p = \Theta(\frac{n^{2.81}}{p^{0.905}} + \Theta(\log p) + \Theta(\frac{n^2}{\sqrt{p}}) \tag{9}$$

The parallel cost is thus

$$C_p = pt_p = \Theta(n^{2.81}p^{0.095}) + \Theta(p \log p)\Theta(n^2\sqrt{p}). \tag{10}$$

However, since serial cost defined by the Strassen's algorithm is $\Theta(n^{2.81})$ this parallel algorithm will always include an extra $p^{0.095}$ increase of cost as p is increased. Thus it is not possible to scale the system in such a way that serial cost remains in proportion to parallel cost and thus the algorithm cannot be cost optimal.