Matrix Multiplication

This assignment is due at class time Thursday, 1 November 2001.

This assignment is to make sure you know how to use more sophisticated communication and communicator mechanisms.

You are to write a program to do matrix multiplication in parallel. Read Chapter 7 in Pacheco carefully, and feel free to use code segments from that chapter as needed. Note that this assignment is essentially the same as programming assignment 1 on page 133.

Your goal should be to multiply two $1024 \times 1024$ matrices together on 16 processors.

Your program should be able to run on either 4 processors as a $2 \times 2$ grid or on 16 processors in a $4 \times 4$ grid.

Start with the 4-processor version. Use your social security number to separate yourself from other people in the class with regard to machine use. That is, if the last two digits of your SSN are 0 mod 4, use processors n01 through n04. That is, if the last two digits of your SSN are 1 mod 4, use processors n05 through n08. And so forth.

Some helpful hints follow.

First, don’t try to do the big problem first. Start, for example, with a $64 \times 64$ matrix on 4 processors (if you feel you must hard code the matrix size, then use a defined constant so you can readily change the size; don’t hard code the 64 in $n + 1$ places in the code). Then go to $256 \times 256$, and then scale up again.

Second, don’t try to do the matrix multiplication and the communication together. Get the communication working first, and then work on the multi-
plication. I would suggest something like the following. Have each processor generate its part of the matrix with unique identifiers, and then pass the data from processor to processor as would be needed. You could simply number the $64^2$ cells of a $64 \times 64$ matrix, for example, from 0 to 4095, and be able to recognize which numbers were supposed to be stored in which locations.

Finally, do the multiplication in the following way. Generate two identity matrices $A_0$ and $B_0$ of the appropriate size. Use the random number generator ($\texttt{man}$ $\texttt{rand}$ to find this) to generate random $i$ and $j$ coordinates and a random $x$ between 0 and 1 that will give you an elementary $E_{ij}$ matrix with 1’s down the diagonal, $x$ in the $(i, j)$ location, and zeros elsewhere. Multiplying $A_0 \cdot E_{ij}$ and $(E_{ij})^{-1} \cdot B_0$ to get $A_1$ and $B_1$ should be something that can be done using the matrix multiplication code, and then $A_1 \times B_1$ should be recognizable as the identity (within machine epsilon). Then repeat 100 or 200 or 1000 times, say, to get $A_2, B_2, A_3, B_3$ and so forth.

Checking that the $A_i$ and $B_i$ matrices are correctly multiplied should be easy, at least for the first few steps, since these are after all elementary matrices.

The bottom line is that this should allow you to generate a “random” matrix $A_{100}$ or so, together with its inverse $B_{100}$, so that multiplication of the two matrices can be verified to be correct because the result is the identity. You might also want to write a sequential version of matrix multiply to check your answers. Note that the random number generators might be different on the sequential version’s machine and on daniel. To deal with this, write a noddy that writes 12,000 random numbers to a file. Then, instead of generating the random numbers, read them from the file, and you have a
reproducible situation you can test and debug against.