Assignment 5–Simple OpenMP Programming

This assignment is due at class time Wednesday, 1 December 2004 and is to provide you with experience doing a simple problem in OpenMP.

You are given four matrices $A_1, A_2, A_3, A_4$. Your job is to compute the product

$$A_7 = A_1 \cdot A_2 \cdot A_3 \cdot A_4$$

by computing $A_5 = A_1 \cdot A_2$, then $A_6 = A_3 \cdot A_4$, and then $A_7 = A_5 \cdot A_6$.

You are to write code to do this in two different ways. The first way is to parallelize the code so that $A_5$ and $A_6$ are computed in parallel (this is functional parallelism). The second way is to parallelize the matrix multiplication routine itself so that different outer loop iterations are assigned to different threads.

Sample matrices, of size $10 \times 10$, and sample output will be place in `/home/csce590/public/Dassignment05`.

Remarks:

- You will be given two sets of matrices, one of four $10 \times 10$ matrices and one of four $100 \times 100$ matrices.

  You need not dimension your arrays any larger than this, and you may use global arrays. Perhaps the simplest way to avoid worrying about multidimensional array handling in C is to globally dimension seven arrays and use subscripts as array numbers. This is ok.
• You should probably write the binary tree breakdown of the multiplication as a function, and you should probably write the matrix multiplication triple loop as a separate function. If you do this, you can use one set of pragmas in the matrix mult routine to parallelize loop iterations and another set of pragmas for the functional parallelism, enclose the pragmas in \texttt{ifdefs} (perhaps \texttt{#ifdef FUNC} and \texttt{#ifdef FOR}) and have a makefile that will compile both versions depending on which preprocessor symbols are provided.

• You need not worry about more than eight threads being available, since we have only eight processors in the machine.

\textbf{Extra Credit:} Dig through the OpenMP documentation. Find out how to parallelize the functional operation of the product as a binary tree and then use more threads to sub-parallelize the loops in the matrix multiplication itself.