A divide and conquer multithreaded algorithm for matrix multiplication.

Suppose we have the following:

$$A=\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \qquad B=\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \qquad C=\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

Then we can write the matrix product as

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}=\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}=\begin{pmatrix} A_{11}B_{11} & A_{11}B_{12} \\ A_{21}B_{11} & A_{12}B_{12} \end{pmatrix}+\begin{pmatrix} A_{12}B_{21} & A_{12}B_{22} \\ A_{22}B_{21} & A_{22}B_{22} \end{pmatrix}$$

Thus, to multiply two n x n matrices, we perform eight multiplications of n/2 x n/2 matrices and one addition of two n x n matrices.

1) Verify the technique works by multiplying two 2 x 2 matrices.

The above definition leads to a divide and conquer strategy given on the next page. After some thought, you should conclude that the recursive calls can be done in parallel.

2) Now suppose A, B, and C are 4 x 4 matrices. Verify the given function will multiply two matrices correctly.

Now consider executing the recursive calls in Matrix-Mult(C, A, B) in parallel.

3) Are there any issues with all 8 calls executing in parallel. Why/why not?

4) Are there any issues with adding T to C when the 8 calls run in parallel?

5) If each parallel call is handled with a task (or another instance of the program running), how many tasks are there if the matrix is of size 2 x 2, 4 x 4, 8 x 8, … , n x n? What if n=1024? Hopefully you will conclude that all recursive calls cannot be converted to another instance of the running program. It will require too many resources if n gets large.

Utilizing a thought from class, let's implement this algorithm in 9 tasks.
Task 0: carve up the data and run the initial 8 recursive calls in parallel
            when the task 1-8 complete, task 0 does a final sum and reports the solution.
Task 1-8: perform either option A or option B
    Option A: perform the recursive function in non-parallel mode on the data received from task 0
    Option B: perform a normal matrix multiply on the data received from task 0
Use the same setup as you did in P1. Only open and process files in Task 0.

Collect and report stats on N x N matrices where N =512, 1024, 2048, 4096
comparing the 1st version (P1) with this new version of matrix mult.
Also, run all sizes of the matrix using 1 host, 4 hosts with slots=2 + aw01, 8 hosts + aw01 with slots=1.
NOTE: slots=1 should be the same as not using the slots option in the host file.

When you time the program, use the struck timeval as shown in class. Record two times for each setup. One time includes all operations including I/O and one time subtracts the time I/O is being performed. Be as accurate as possible to isolate the I/O. Put N values on the x axis, and the run environments on the y axis.

```
Matrix-Mult(C, A, B)
{
    // C = A * B  -  A, B, and C are square matrices.
    n = number of rows in A
    if n = 1
        $c_{11} = a_{11}\ b_{11}$
    else
        let T be a new N x N matrix
        partition A, B, C, and T into n/2 x n/2 submatrices.
            $A_{11}, A_{12}, A_{21}, A_{22}$
            $B_{11}, B_{12}, B_{21}, B_{22}$
            $C_{11}, C_{12}, C_{21}, C_{22}$
            $T_{11}, T_{12}, T_{21}, T_{22}$
       /* NOTE:  $A_{11}$, is the upper left n/2 x n/2 sub matrix of A.
        *        $A_{12}$, is the upper right n/2 x n/2 sub matrix of A.
        *        ...
        *        for all sub matrices.
        */
       Matrix-Mult($C_{11}, A_{11}, B_{11}$);
       Matrix-Mult($C_{12}, A_{11}, B_{12}$);
       Matrix-Mult($C_{21}, A_{21}, B_{11}$);
       Matrix-Mult($C_{22}, A_{21}, B_{12}$);
       Matrix-Mult($T_{11}, A_{12}, B_{21}$);
       Matrix-Mult($T_{12}, A_{12}, B_{22}$);
       Matrix-Mult($T_{21}, A_{22}, B_{21}$);
       Matrix-Mult($T_{22}, A_{22}, B_{22}$);

       // Finialize by adding T to C
       for i=1 to n
            for j=1 to n
                 $c_{ij} = c_{ij} + t_{ij}$

}
```