CS 740, Fall 1997
Assignment 6:
Part I: Programming in Cilk

Assigned: Fri., Nov. 21
Due: Fri., Dec. 5

1. Acknowledgements

This programming assignment was designed by the Cilk group at M.I.T., and is used with their permission.

2. Policy

You may work in a group of up to 3 people in solving the problems for this assignment.

3. Logistics

Any clarifications and revisions to the assignment will be posted on the class bboard and Web page. For this problem you will turn in both written answers and Cilk code. Please turn in your written answers in class, and turn in your working code electronically by mailing it to `bmm@cs.cmu.edu`. In both cases, please be sure to identify all of the members in your group.

4. Matrix multiplication

In this problem you will implement Strassen's algorithm for matrix multiplication in Cilk. The goal of this problem is threefold. First, we want you to learn Cilk. Second, we want you to get a feeling of the tradeoffs among work, space and depth. Finally, the fastest program will win a Cilk T-shirt.

4.1. Warm up

In this section you will write some short and easy Cilk programs, so that you get accustomed with Cilk.

(a) To begin you will implement a divide-and-conquer algorithm for matrix multiplication. The inputs to the algorithm are two dense $n \times n$ matrices A and B. The algorithm is based on the following identity,

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix} .$$

where each submatrix A_{ij} or B_{kl} is an $n/2 \times n/2$ matrix.

Make sure your algorithm has a depth $\Theta(\lg^2 n)$; you need to parallelize the additions in order to achieve this.

(b) Implement a matrix multiplication algorithm with $\Theta(n^3)$ work that does not use extra space.

(c) Implement a matrix multiplication algorithm with $\Theta(n^3)$ work and $\Theta(\lg n)$ depth.

4.2. Strassen's algorithm

In this section we present Strassen's algorithm for multiplying two $n \times n$ matrices.[1] The remarkable property of the algorithm is that it is asymptotically faster than the naïve $O(n^3)$ algorithm.

Let A and B be two $n \times n$ matrices. For the rest of the assignment, assume that n is a power of 2. Recall that the product of A and B is defined to be $C = AB$, where

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj} \ .$$

The definition leads to a straightforward $O(n^3)$ algorithm to compute the product. However, there is a remarkable identity that leads to a faster divide-and-conquer algorithm. Partition A and B as follows:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad \text{and} \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \ .$$

Then we have:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} \ ,$$

where

$$\begin{aligned}
C_{11} &= A_{11}B_{11} + A_{12}B_{21} \ , \\
C_{12} &= w + (A_{21} + A_{22})(B_{12} - B_{11}) + (A_{11} + A_{12} - A_{21} - A_{22})B_{22} \ , \\
C_{21} &= w + (A_{11} - A_{21})(B_{22} - B_{12}) - A_{22}(B_{11} - B_{21} - B_{12} + B_{22}) \ , \\
C_{22} &= w + (A_{11} - A_{21})(B_{22} - B_{12}) + (A_{21} + A_{22})(B_{12} - B_{11}) \ , \\
w &= A_{11}B_{11} - (A_{11} - A_{21} - A_{22})(B_{11} - B_{12} + B_{22}) \ .
\end{aligned}$$

If all intermediate results are appropriately saved, a multiplication of size n can be reduced to 7 multiplications of size $n/2$, plus 15 matrix additions. The relevant steps are outlined below (the S_i's, M_i's and R_i's are temporary matrices):

$$\begin{aligned}
S_1 &\leftarrow A_{21} + A_{22} \\
S_2 &\leftarrow S_1 - A_{11} \\
S_3 &\leftarrow A_{11} - A_{21} \\
S_4 &\leftarrow A_{12} - S_2 \\
S_5 &\leftarrow B_{12} - B_{11} \\
S_6 &\leftarrow B_{22} - S_5 \\
S_7 &\leftarrow B_{22} - B_{12} \\
S_8 &\leftarrow S_6 - B_{21} \\
M_1 &\leftarrow S_2 S_6 \\
M_2 &\leftarrow A_{11} B_{11} \\
M_3 &\leftarrow A_{12} B_{21} \\
M_4 &\leftarrow S_3 S_7
\end{aligned}$$

[1] This particular version of the algorithm is actually due to S. Winograd.

$$
\begin{aligned}
M_5 &\leftarrow S_1 S_5 \\
M_6 &\leftarrow S_4 B_{22} \\
M_7 &\leftarrow A_{22} S_8 \\
R_1 &\leftarrow M_1 + M_2 \\
R_2 &\leftarrow R_1 + M_4 \\
C_{11} &\leftarrow M_2 + M_3 \\
C_{12} &\leftarrow R_1 + M_5 + M_6 \\
C_{21} &\leftarrow R_2 - M_7 \\
C_{22} &\leftarrow R_2 + M_5
\end{aligned}
$$

Clearly, the 7 intermediate multiplications can be computed recursively by the same algorithm.

The work obeys the recurrence

$$T_1(n) = 7T_1(n/2) + \Theta(n^2) \ ,$$

whose solution is $T_1(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$. More details on Strassen's algorithm can be found in Chapter 31 of *Introduction to Algorithms* by Cormen, Leiserson, and Rivest.

(d) Describe an implementation of Strassen's algorithm with small depth. In other words: identify which parts can be done in parallel, and analyze the critical path of the resulting parallel algorithm.

(e) Analyze the space required by the implementation you described in part (a).

4.3. Implementation of Strassen's algorithm

You are now ready for the main part of the assignment.

(f) Implement a simple Cilk version of the algorithm from part (a). This should really be simple, just to get you going. Do all the recursive multiplications in parallel, and try to do some of the additions in parallel, too.

(g) In this part, we ask you to optimize your program. Experiment with *at least* one of the following ideas.

1. Experiment with tradeoffs between parallelism and space. The idea is that if you force certain operations to happen after other operations (with a `sync`), you may be able to reuse memory, at the expense of a longer critical path. Reusing memory may increase your speed, because of better cache behavior.

   If your changes alter the (asymptotic) critical path and space requirements of your program, analyze them again.

   You can use the Cilk `SYNCHED` predicate to find out at runtime whether previous computations have already terminated. In this case, you may be able reuse some space.
2. Instead of recursing down to $n = 1$, switch to the $O(n^3)$ algorithm for small n. Find the best switch point. Determine experimentally the constants involved in the $O(n^3)$ expression and in your expression for the total work, and analyze what the optimal switch point should be. Do your predictions agree with the experiments?

3. Present-day computers are faster when they can access memory sequentially. The matrix product AB is bad from this point of view, since if both A and B are stored in row-major order, then the inner loop will need to access the columns of B, which are not stored sequentially in memory. Modify your program to compute AB^T instead.
4. Alternatively, suggest other improvements and experiment with them.

Fast and/or original programs will win a prize!

5. Resources

The directory `/afs/cs/academic/class/15740-f97/public/asst/asst6` contains a file `strassen.cilk`, that provides a testing framework for the program you write, as well as a `Makefile` that you should use.

The test program takes a couple of options. The option `-n <n>` determines the size of the matrix to be tested. By default, the matrix is 256×256. The test program creates a random matrix and runs your multiplication routine once, telling you the run time, the work and the critical path. The other option is `-c`, that compares the output of your multiplication routine with the correct result, and tells you if they are the same or not.

In addition, every Cilk program accepts a predefined set of Cilk options. These are explained in the Cilk manual. You should at least be aware of the options `-nproc <n>`, that sets the number of processors your program is running on, and `-heap <n>`, that determines how much memory your program is allowed to use (in megabytes). The default is 1 processor and 64 megabytes. For example:

```
% ./strassen -nproc 8 -n 1024
```

Remember that Cilk options must precede program-specific options in the command line.

5.1. Conventions used in the test program

You have to put your multiplication routine into the `strassen.cilk` program. As provided, the program calls a sequential routine. Your goal is to write the function

```
cilk void strassen(int n, REAL *A, int an, REAL *B, int bn,
                   REAL *C, int cn)
```

Your function must compute $C = AB$, where A, B and C are $n \times n$ real matrices, and n is a power of two. The type `REAL` is predefined to `double`. The matrices are stored in the following way. Every matrix (say, `A`) has an associated integer (`an`). `A` points to the first element of the matrix, and `an` is the distance between two elements in the same column. In other words, the element A_{ij} is stored in location `A[i * an + j]`. The macro `ELEM(A, an, i, j)` is predefined to be `A[i * an + j]`, for convenience.

In the test program there are a bunch of utility functions that allocate and destroy matrices, print, compare two matrices, etc. Their use and implementation are straightforward, and documented in the code. We have provided a sequential function `matrixmul`, that computes the product of two matrices. That function is likely to be slow, but it is useful for checking the result of your program. We also provided a faster function `matrixmul2`, that works only for even n. You may use either in your program, if you wish.

6. The Competition (Optional)

We are planning to hold a competition by running the Cilk programs on `cuff.link.cs.cmu.edu`, which is a multiprocessor from Sun. I am working on installing Cilk on this machine, and setting up accounts so that you can evaluate the performance of your programs. Programs will be judged on the basis of speed alone.

Each member of the winning team will receive a Cilk T-shirt. The decision of the judges is final! Members of the immediate families of the instructors of the course, their heirs, and their creditors are not eligible.

More details corcerning the competition will be provided in a future handout.