CS 240A Assignment 4:
Cilkified Inner Products

Assigned April 26, 2010

Due by 11:59 pm Monday, May 3

The purpose of this assignment is to gain familiarity with Cilk++ constructs and tools, as well as to think about different ways of parallelizing an algorithm using Cilk++. You will write a parallel routine that computes the inner product (dot product) of two arrays in three different ways. For grading purposes, both correctness and performance will count.

1. Background

The inner product of two vectors is the sum of their elementwise multiplications. In pure C, the inner product of two integer arrays, each of size *n*, can be implemented as follows:

```
int innerprod = 0;
for(int i=0; i< n; ++i)
{
        innerprod += a[i] * b[i];
}
```

In C++, the standard library has a dedicated function for computing the inner product. It is made available by including the <numeric> header. For the purposes of this assignment (and this class), you don't have to know and use C++, but you're allowed to do so because the Cilk++ compiler is an extension of C++ and will accept all C++ constructs.

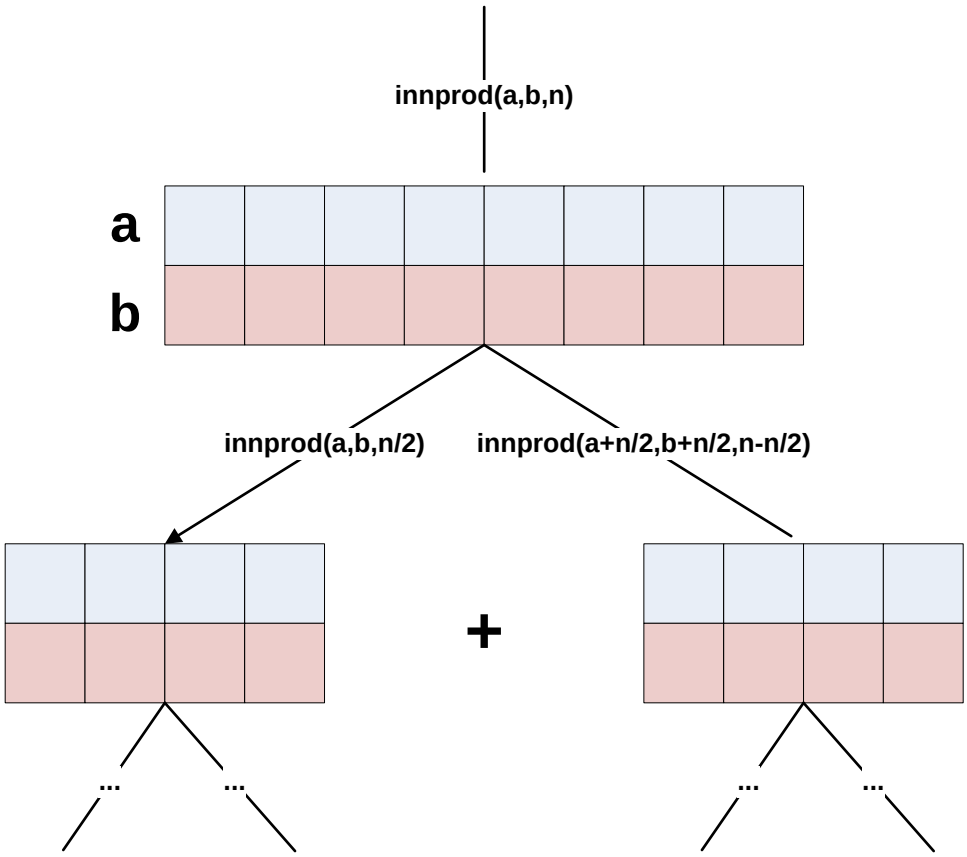
```
int innerprod = std::inner_product(a, a+n, b, 0);
```

2. What to implement

You will write three different functions to evaluate the inner product in parallel.

1. *rec_cilkified(int * a, int * b, int n)*

   This will evaluate the inner product recursively, by splitting each array into two at each stage of the recursion and adding the partial sums at the end. You should switch to a sequential execution when the subarray sizes become smaller than a threshold called *coarseness*. Here is a picture of the calculation:



2. loop_cilkified(int * a, int * b, int n)

   This function contains two nested loops: the outer loop executes *n / coarseness* times, and the inner loop executes *coarseness* times. You should parallelize the outer loop with *cilk_for*, and let each inner loop execution proceed sequentially. Finally, you'll combine the results of the inner loop executions by adding them all together sequentially.

3. hyperobject_cilkified(int * a, int * b, int n)

   For this function, you'll use a hyperobject. Specifically, you'll use a reducer similar to one shown in page 54 of the Cilk++ programmer's guide, and let the Cilk++ reducer concept take care of data races and combining the results.

There is a driver / harness code called "innerproduct.cilk" under hw4 on the course web site. You can use this file as a template and implement the required functions, which are left blank for you to fill in. The same directory contains a sample makefile and a header with the timing function.

Your program will be executed as:

>> **./innerproduct [sizeofarray]**

where the parameter is optional and default is 1 million.

If the harness reports "incorrect" on your results, use **cilkscreen** to identify any data races that might exist in your code. (Don't forget that you must build the debug version of your code for **cilkscreen** to report line numbers where data races occur)

3. What to report/answer

1. Run your code with different input sizes (**sizeofarray** = 10^4,10^5,10^6,10^7,10^8) on a fixed number of cores (**CILK_NPROC** = 4). Plot a graph that has input size on its x-axis (in log scale) and speedup on the y-axis. Put all three lines (one for each parallel routine you wrote) on the same graph.

2. Run your code on different numbers of cores (**CILK_NPROC** = 1,2,3,4,5,6,7,8) with a fixed input size (**sizeofarray** = 10^6). Plot a graph that has the number of cores on its x-axis (in normal scale) and speedup on the y-axis. Plot all three lines (one for each parallel routine you wrote) on the same graph.

3. (Extra Credit) Experiment with different *coarseness* values and report on the sensitivity of the performance to different values.

4. (Extra Credit) The harness reports the speedup of your implementations compared to the sequential inner product. It is highly likely that your routines won't enjoy linear speedup. What might be the reason? Do you think that the algorithm does not have sufficient parallelism, or do you think there is another bottleneck other than parallelism?

Hint for (4): Using **cilkview**, you can empirically find out the parallelism of your code. However, you should remember that the tool estimates parallelism for the entire program, meaning that your possibly sequential input parsing will make your program look less parallel than it actually is. Refer to page 105 of the cilk++ programming guide on how to estimate work/span of a specific portion of your code.

You may do this assignment solo or in a group of two. Be sure your turnin contains the names and perm numbers of both group members.