Recitation 5: Analyzing Parallel Programs

In this recitation you will gain more experience analyzing the work and span of parallel programs.

1 Getting started

We recommend that you work on the course machines.

```
$ ssh <username>@cloud.csail.mit.edu
```

To get a local copy of the repository for your work, you need to use Git to clone it.

```
$ git clone /afs/csail.mit.edu/proj/courses/6.172/student-repos/fa14/recitations/\> recitation/<username>.git recitation5
```

2 Benchmarking parallel programs using Lanka

It is important to use a job queueing system when benchmarking parallel programs. When two students run serial programs on a common cloud machine at the same time, their programs will generally run on different cores. When two students run parallel programs on a common cloud machine, however, each program will run on all cores, causing both students to get inaccurate performance results.

When you execute a program on Lanka, the program’s execution is added to a job queueing system, which ultimately runs your program on a machine isolated from other users. As you’ve seen before, you can run the your implementations for this recitation as follows:

```
$ lexec ./main
```

3 Matrix-vector multiplication

In the recitation directory, there is a simple, serial program which computes $A \cdot v$ for random matrix $A$ and vector $v$. Compile and run this program on Lanka, and take a note of its running time.

We would like you to implement and analyze three different versions of this program.

1. Switch the for loop in `matrix_vector_product` to a `cilk_for` loop.
2. Parallelize `dot_product` using a Cilk opadd reducer.
3. Parallelize both `matrix_vector_product` and `dot_product`.

Here is an example of how you can use the Cilk opadd reducer:
```c
#include <cilk/reducer_opadd.h>

void sum_A(const double A[], int n) {
    CILK_C_REDISCRIP_OPPADD(r, double, 0);
    CILK_C_REGISTER_REDISCRIP(r);
    cilk_for(int i = 0; i < n; ++i) {
        REDUCER_VIEW(r) += A[i];
    }
    printf("The sum of the elements of A is %f\n", REDUCER_VIEW(r));
    CILK_C_UNREGISTRER_REDISCRIP(r);
}
```

For each of the three parallel versions of your program, please perform the following exercises.

- **Exercise:** Analyze the theoretical work, span, and parallelism of the parallel program. Based on this analysis, and given that this program operates on a $40000 \times 40000$ matrix and a $40000$-element vector, what do you estimate the measured parallelism of your program to be?

  *Hint:* When a Cilk program uses a single “simple” reducer, such as an opadd reducer, whose associative binary operator is a $\Theta(1)$-time operation, then the Cilk runtime system incurs no additional asymptotic overhead to maintain the reducer’s views. You can therefore analyze the asymptotic work and span of such a Cilk program by analyzing its source code alone, just as you have done already.

- **Exercise:** Use cilkview to measure the parallelism of your program. How does this measurement correspond to your theoretical analysis? What could account for any discrepancies between your measurements and your analysis?

- **Exercise:** Report on the running time of your program. Try to explain any unexpected results. How can you improve its empirical performance? For example, can you reduce the empirical overhead of the cilk_for loops and reducers using coarsening?