Proposal: Parallel Edit Distance

Cary Yang (caryy), Kevin Zhang (kkz)

April 5, 2014

1 Summary

We are going to implement a parallel edit distance algorithm. Implementations on both CUDA and C++ (Gates machines) will be compared.

2 Background

The edit distance between two strings X and Y measures the minimum number of single-character modifications required to change X to Y. A popular dynamic programming (DP) algorithm can compute the edit distance in $O(|X||Y|)$ time and space. This involves creating a DP table of size $|X|$ by $|Y|$ and evaluating the entries in row-major order.

The edit distance provides a useful measure of similarity between strings, and can be used for many applications from DNA profiling to spell checking to approximate string matching.

3 The Challenge

In the most popular dynamic programming edit distance algorithm, the value of each DP entry depends only on the value above, to the left, and of the upper-left neighbor. Because of this, one potential parallel execution in evaluating the DP entries is along a "diagonal frontier." Since the length of the longest diagonal row changes as we traverse the table, the amount of achievable parallelism changes dynamically throughout a single problem instance. This poses an interesting challenge and involves a very dynamic but predictable allocation of resources.

Another potential avenue of parallelization is to evaluate the DP table in row-major order, but compute each row in parallel. However, as each entry additionally depends on the evaluated value of its left neighbor, this is non-trivial and involves a scan-esque procedure.

As seen, the primary challenge presents itself in the form of non-static parallelizability (diagonal method) or non-trivial parallel computation (row-major method). This is, of course, assuming our implementation is based on the standard DP algorithm. Alternative algorithms will also be considered should they provide potential for parallelization.

4 Resources

We will most likely start from code provided by Hongyi Xin, a reasearcher on this topic, but he has not yet gotten back to us on the subject. If he does not respond in time, then our first order of business will be to write some starter code as a foundation for our final product. In any case, we will investigate parallel implementations on both CUDA and the CPU on the Gates machines. Due to the simplicity of the problem statement, we will also likely experiment with implementations written from scratch. Useful tools for implementation include the Open MP and Cuda libraries provided with the 15418 homework assignments.

5 Goals/Deliverables

5.1 Plan to Achieve

Fast implementation of edit distance on the Gates machines. We wish to achieve reasonable speedup on either a 6-core processor or an Nvidia graphics card.

5.2 Hope to Achieve

Visual DNA matching demo which also displays performance metrics. Additionally, a useful application of approximate string matching.

5.3 Demo

At minimum, a speed-up analysis comparing the sequential and parallel execution times.

6 Platform Choice

The platform we have chosen is C++ running on the Gates machines. The main reason for this is that we're looking to investigate the performance of our implementation, which is most easily done in C++ due to the low abstraction distance from assembly. Additionally, because we are primarily concerned with achieving high speed-up on a shared memory space, machines with many cores will not be necessary.

7 Schedule

- Week of 04/7: Discuss potential algorithms with Hongyi. Begin conceptualizing a framework for the product.
- Week of 04/14: Implement useful starter code. Work on naive parallel approaches (pre optimization).
- Week of 04/21: Finish high-level implementations of final product. Begin collecting performance data for write-up.

- Week of 04/28: Experiment with final tweaks in low-level optimization. Here, we want to be meticulous with allocation of work and resource.
- Week of 05/05: Finalize write-up and polish demonstration.