SMA5509: Theory of Parallel Systems
Project Proposal
Cache-oblivious sorting for Burrows-Wheeler Transform

Advait D. Karande   Sriram Saroop

October 2003

1 Background and Motivation

Burrows and Wheeler [1] presented a lossless text compression method based on Burrows-Wheeler Transform (BWT). A computationally expensive activity during this compression is the sorting of all rotations of the block of data to be compressed. The original implementation incurs the worst case complexity of $O(N^2log(N))$, if there is a high degree of repetitiveness in the input file. The number of characters, N, is typically quite large and such worst case behavior is unacceptable.

In [2], Sadakane has provided a sorting algorithm with a better worst case bound of $O(Nlog(N)^2)$. Seward [3], however, has shown that even though Sadakane's algorithm has better asymptotic behavior, it incurs high overheads and a tuned direct-comparison implementation of rotation sorting performs much better in practice. The tuned implementation presented in [3] mainly improves the performance of the original algorithm by minimizing the memory references, especially the cache misses for most inputs. The author concludes that these memory effects are critical for the performance of the algorithm on the modern architectures.

We believe that such a heavy dependence of the performance of the algorithm on the memory effects can be minimized by developing a *cache-oblivious* implementation of the algorithm. Such an implementation is especially useful for the compression applications such as **bzip2** based on the algorithm, which are required to perform well on a wide range of platforms and their underlying architectures. We proceed to give an overview of Cache Obliviousness and how it serves the purpose of making BWT sorting more efficient.

Cache Oblivious Algorithms

An algorithm is cache oblivious if no program variables, dependent on hardware configuration parameters such as cache size and cache-line length, need to be tuned to minimize the number of cache misses. It has been shown in the past [4] that several cache oblivious algorithms use cache as effectively as *cache-aware* algorithms. The cache obliviousness concept has been employed to efficiently perform tasks like rectangular matrix multiplication, matrix transposition, FFT, Funnel Sort, Distribution Sort etc.

Where does Cache Obliviousness fit in?

It is to be noted that cache oblivious algorithms would be of maximal benefit when the developed application needs to be run on several different platforms, of which we do not have any prior knowledge. In this case, it is not possible to tune variables according to hardware parameters since they vary across different platforms. Hence, we have chosen an application bzip that is based on the Burrows Wheeler Transform(BWT), which shall be adapted to suit the cache oblivious paradigm.

Block Sorting Lossless Data Compression algorithm

Michael Burrows and David Wheeler in their paper [1] discuss a complete set of algorithms for compression and decompression. The essence of the paper consists of the disclosure of the BWT algorithm. The algorithm takes a block of data and rearranges it using a sorting algorithm. This transformation is reversible, meaning that the original ordering of the data elements can be restored with no loss of fidelity. The authors have shown that the transformed string results in much better compression than the original string.

Why do we target Bzip2?

Bzip2 [5] is an open source data compression application which achieves a good compression ratio. It compresses files using the BWT algorithm, and Huffman coding. However, it can be quite slow at compression, as compared to the fast decompression. It is an application that can be used on any platform. Hence, we cannot possibly tune the variables in the application according to hardware parameters. Therefore, cache obliviousness is an apt paradigm to be applied to this scenario.

2 Project Plan

We realize that the sorting algorithm in BWT is the most expensive operation as it possibly incurs a lot of cache misses. Making the sorting algorithm cache oblivious would provide a performance gain in the average case across different platforms.

Hence in this project, we propose to explore ways of making the sorting algorithm cache oblivious. The open source implementation of bzip2 will serve as a concrete reference for our work. We wish to investigate whether the cache oblivious implementation actually results in a better or at least comparable performance with respect to the reference implementation. Even if the results are not satisfactory, we hope to explain why one implementation works better than the other.

We also wish to explore ways of parallelizing the BWT sorting algorithm using Cilk. This research direction will serve as a fallback mechanism, in case the cache oblivious implementation does not provide satisfactory results. We will investigate into the possible speedup that can be obtained by parallelizing the algorithm and compare it with the empirical results.

The key segments of this project would be:

- Designing a cache oblivious sorting algorithm for BWT.(3 weeks)
- Performance Comparisons with available benchmarks.(2 weeks)
- Evaluating the effectiveness of employing cache-obliviousness.(1 week)
- Exploring ways to parallelize the BWT sorting algorithm, if time permits. (2 weeks)

References

[1] M. Burrows and D. J. Wheeler, “A block-sorting lossless data compression algorithm.,” Tech. Rep. 124, Digital Systems Research Center, Palo Alto, 1994.

[2] K. Sadakane, “A Fast Algorithm for Making Suffix Arrays and for Burrows-Wheeler Transformation,” in *Proceedings of IEEE Data Compression Conference (DCC’98)*, pp. 129–138, 1998.

[3] J. Seward, “On the performance of bwt sorting algorithms,” in *Proceedings of IEEE Data Compression Conference*, pp. 173–182, 2000.

[4] H.Prokop, “Cache oblivious algorithms,” Master’s thesis, Massachusetts Institute of Technology, 1999.

[5] J.Seward, *The bzip2 and libzip2 homepage*, 2002.