Abstract

Modern computers are far more sophisticated than simple sequential programs can lead one to believe; instructions are not executed sequentially and in constant time. In particular, the memory of a modern computer is structured in a hierarchy of increasingly slower, cheaper, and larger storage. Accessing words in the lower, faster levels of this hierarchy can be done virtually immediately, but accessing the upper levels may cause delays of millions of processor cycles.

Consequently, recent developments in algorithm design have had a focus on developing algorithms that sought to minimize accesses to the higher levels of the hierarchy. Much experimental work has been done showing that using these algorithms can lead to higher performing algorithms. However, these algorithms are designed and implemented with a very specific level in mind, making it infeasible to adapt them to multiple levels or use them efficiently on different architectures.

To alleviate this, the notion of cache-oblivious algorithms was developed. The goal of a cache-oblivious algorithm is to be optimal in the use of the memory hierarchy, but without using specific knowledge of its structure. This automatically makes the algorithm efficient on all levels of the hierarchy and on all implementations of such hierarchies. The experimental work done with these types of algorithms remain sparse, however.

In this thesis, we present a thorough theoretical and experimental analysis of known optimal cache-oblivious sorting algorithms. We develop our own simpler variants and present the first optimal sub-linear working space cache-oblivious sorting algorithm. These algorithms are implemented and highly optimized to yield high performing sorting algorithms. We then do a thorough performance investigation, comparing our algorithms with popular alternatives.

This thesis is among the first to provide evidence that using cache-oblivious algorithm designs can indeed yield superior performance. Indeed, our algorithms are able to outperform popular sorting algorithms using cache-oblivious sorting algorithms.

We conclude that cache-oblivious techniques can be applied to yield significant performance gains.