Chapter 7

Conclusion

In this thesis, we set out to produce a high performing cache-oblivious sorting algorithm in part to clarify the feasibility of cache-oblivious algorithms in the context of sorting.

This feasibility is not a given. Cache-oblivious algorithms are proven optimal in the memory hierarchy using assumptions that appear unrealistic. Not before an implementation of the algorithms has been created and thoroughly analyzed experimentally, will it become clear whether these assumptions are indeed unrealistic, or cover for aspects of the realities of modern day hardware that are important for achieving high performance. Furthermore, optimal cache-oblivious sorting algorithms are more complex and require more instructions to be executed per element sorted. Thus, it is unclear whether this increased complexity will cause the algorithm to perform badly despite any improvement in cache utilization. Again, only an experimental analysis of the algorithms will provide a clear answer.

Realizing that space consumption is of great importance when working with large datasets, we have developed a novel low-order working space cache-oblivious sorting algorithm, LOWSCOSA. It has optimal complexity and uses sub-linear working space to sort elements, keeping them in the array in which they are stored.

We have provided an implementation of the cache-oblivious sorting algorithm funnelsort and an implementation of the LOWSCOSA.

Using a detailed knowledge of the inner workings of both compilers, modern processors and modern memory systems, we build an understanding of what ingredients are needed in a high performance cache-oblivious sorting algorithm. We have used a process of thorough experimentation to determine exactly which ingredients improve performance in practice. Through this process, many good ideas were tried out in practice. However, only very few proved able to yield improvements in performance. Though it may seem unfortunate that we were unable to improve performance, using approaches that are more sophisticated, it is indeed not. By showing that performance does not improve significantly when using complex solutions, we have also shown that the simplest implementation of these algorithms will likely be as fast as or even faster than solutions that are more complex. That fast implementations of algorithms can be created using only simple techniques, is important for the spread and acceptance of the algorithm.

In particular, we have shown both theoretically and in practice that perhaps the most complex aspect of the algorithms, namely the controlled layout, is of minimal importance for performance.

This thesis breaks new ground by proving that implementations of cache-oblivious sorting algorithms can have performance comparable or even superior to popular alternatives. We have provided evidence that the assumptions, such as full associativity, made to argue optimal utilization of cache, is no hindrance to achieving high cache utilization. Indeed, we have shown that cache-oblivious algorithms can exploit the caches even better than algorithms tuned for the cache and in turn achieve higher performance.

We have also shown that these results are consistent through several different types of input and on several radically different hardware architectures.

However, our cache-oblivious algorithms are not able to outperform algorithms designed and implemented specifically to be efficient in handling disk accesses.

7.1 Further Improvements of the Implementation

We believe the performance of our implementations can be further improved. Our implementation is build as a set of clear-cut abstractions. This was done to ensure modularity, enabling us to try out different pieces of code in the same contexts. However, the compiler may not be able to see through all of these abstractions, making it hard for it to generate optimal code.

Now that we have determined what pieces of code yield high performance, this structure of abstractions is no longer needed. We believe, implementing the algorithms from scratch using these pieces of code but without the abstractions, will yield a higher performing implementation.

We believe that to achieve the performance levels of algorithms designed and implemented for efficient handling of disk accesses, either the I/O of the operating system need to be reworked, or cache-oblivious algorithms need to begin utilizing the same techniques. These techniques include double buffering and prefetching. Using such techniques would likely also result in a faster implementation.

7.2 Further Work

When using constant sized samples for finding medians for use in partitions, we risk making uneven partitions. For performance reasons, we cannot afford to find the exact median in the LOWSCOSA. This in turn means that in practice, we run the risk of quadratic complexity. To make the LOWSCOSA more compelling, we need to find a way to guarantee worst-case optimality in practice.

Depending on the choice of parameters used in the algorithm, the LOWSCOSA still consumes a significant amount of additional memory. We would like to see this amount reduced to a logarithmic term or perhaps constant.

The LOWSCOSA does not have competitive performance in practice for large datasets. It incurs $6N/B$ memory transfers, where only $4N/B$ is needed. We would like to see a variant of the LOWSCOSA that also had a competitive performance on large datasets.

Still only very little work has been done in the experimental study of cache-oblivious algorithms. A lot more ground needs to be covered. It is the hope of the author that this thesis will help pave the way for more work to be done in this area.