PARALLEL SORTING ALGORITHMS IN FPGA

András SZÉLL
Advisor: Béla FEHÉR

I. Introduction

Sorting algorithms have been investigated since the beginning of computing. There's a lot of well-known and thoroughly analyzed, optimized algorithms for different problem sets, with different average runtimes and memory needs. For specific input data and specific computer architectures, there isn't a general "best of all" algorithm, sorting algorithms like quick sort that tend to be fast on average, may be less efficient in special cases.

In this paper I describe some major sorting algorithms and architectures taken into consideration and present a method to create fast, hardware-efficient and deterministic sorting solution for a specific data set in molecule data clustering on a Virtex-II FPGA. As data clustering is an often applied method in bioinformatics, efficient sorting is a vital intermediate step.

II. Sorting architectures

Most sorting algorithms consist of three main steps: read, compare and store – sorting is a memory intensive task. For finding a good sorting solution, memory operations has to be performed fast and the algorithm must be smart enough to reduce the number of necessary operations. As general purpose processors have extremely fast L1 and L2 caches and clock frequencies several times higher than in any processor implemented in FPGA, their sorting performance can be achieved only by higher level of parallelism in embedded applications.

For parallel operations, there are several different possibilities [1]. Sorting networks (see Figure 1) have a great performance but only for limited data count, as their hardware requirements increase exponentially. Sorting with mesh-connected processor arrays is an interesting method, but only for limited data sizes. Another way for parallel sorting is using several processors and a more sophisticated data distribution mechanism. These processors can be either MicroBlaze or PicoBlaze processors (according to the complexity of the sorting algorithm), a Virtex-II 6000 may implement 8-12 MicroBlaze processors and much more PicoBlazes.

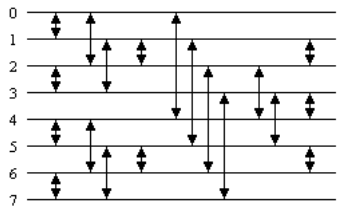

Figure 1: Sorting network

All these solutions are limited by the speed of the input/output memory and the hardware resources of the FPGA, so for the optimal performance, these attributes and the type of the input data has to be considered.

III. Input data and hardware limitations

The input data is a 256 megabyte set of 64 bit elements, as on Figure 2. Data has to be sorted in A, R, B order; storage also follows this order. This way either 32/64 bit integer comparisons or bit-wise comparisons can be carried out. The algorithm which generates the data performs a bucket sort on index A with a bucket size of 128 indexes, and in our case sorting by B index is unnecessary when using stable sorting algorithms ([2], e.g. merge sort).

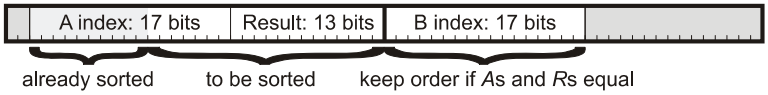

Figure 2: Input data format

Input data is stored in a 100 MHz, 64 bit wide external SDRAM memory, while the parallel sorting process can use the internal block RAMs and distributed RAMs. In Virtex-II 6000 there are 144 block RAMs with 18Kbit in each block. They are best utilized as 512x64 bit dual port modules by coupling them, this way a 100 MHz clock speed can be easily achieved with double speed for read operations. Another external memory is present for storage of partial results.

Input data has been thoroughly analyzed by simulations on data sets extracted from real molecule databases. The deviation of the number of pairs at a specific A index is much higher than the deviation in buckets of 128 A indexes, resulting in sub-optimal hardware utilization, though due to the greater element count, sorting will take much more time in the latter case. Merge block sizes are set according to the results of the analysis.

IV. Parallel merge sorting

The main concept of the recursive merge sort is the following: divide the unsorted elements into two equal size sublists, sort the sublists and merge the two sorted lists into one.

Merge sort is an *O(n log(n))* linear algorithm [2]. (Real parallel algorithms may have an asymptotic complexity of *O(n)*, but they need extreme hardware resources, so they are not feasible for our data set, where data is read in from a linear memory.) In general it is slower than quicksort, but there is smaller difference in its best and worst case run lengths unlike in quicksort, making hardware timing optimizations easier, and its simplicity results in a smaller hardware complexity. Merge sort needs *2n* memory for sorting *n* elements – a big disadvantage that has to be addressed as memory limitation is a bottleneck of the sorting procedure.

The proposed algorithm is a two-round merge sort with a previous 8-element sorting network to reduce recursive complexity. Unsorted data is written into the 32 merge cells via this net; then in $log_2(512/8)=6$ merge cycles, data is sorted by these units in parallel (each cell sorts its 512 elements in 2x512 places). Storing the results of the cells is done via a binary tree which sorts the 32*512 elements on the fly; while storing the results, the next set of data is read into the empty half of the merge cells. After sorting each 16384-element block of a bucket, these blocks are merged from the temporary memory and the sorted bucket is written over the unsorted input.

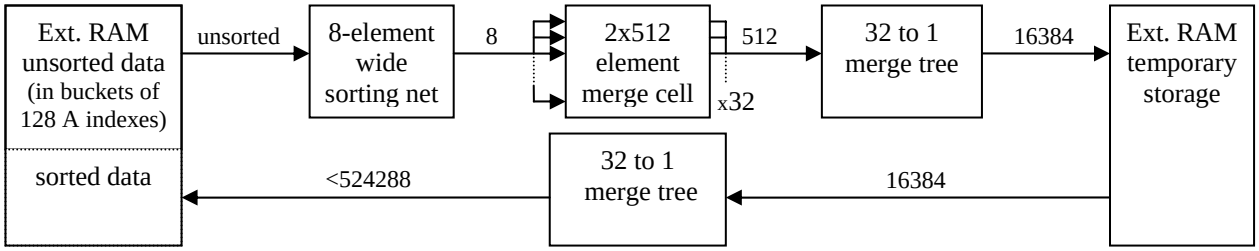

Figure 3: Sorting steps – numbers between sorting steps show the size of sorted blocks

V. Conclusion

The proposed architecture sorts the input data in 1 memory read cycle (33M steps), 1 memory write cycle (33M) and 6 merge cycles per 16K-blocks (>6.3M steps) time, with deterministic run times. The architecture depends on input data properties and hardware features, but the algorithm itself is much more general. The objective to get the sorting speed of Pentium 4 class processors on Virtex-II FPGAs was reached.

References

[1] H.W. Lang, *Sequential and parallel sorting algorithms*, Fachhochschule Flensburg, 2000, URL: http://www.iti.fh-flensburg.de/lang/algorithmen/sortieren/algoen.htm

[2] *Sorting algorithm*, Wikipedia, 2005, URL: http://en.wikipedia.org/wiki/Sorting_algorithm