Fast Regular Circuits for Network-based Parallel Data Processing

Valery SKLYAROV, Iouliia SKLIAROVA
University of Aveiro/IEETA, 3810-193, Portugal
skl@ua.pt, iouliia@ua.pt

***Abstract*—This paper is dedicated to the design, implementation, and evaluation of fast circuits executing operations that are frequently required in data processing which are: 1) discovering the maximum and minimum values in a given set of data; and 2) sorting data items. We found that minimizing the number of circuit components does not guarantee minimal hardware resources. This is because interconnections also influence the complexity significantly. Network-based circuits are often considered to be combinational. However, this does not mean that they are faster than sequential circuits solving the same problem because propagation delays can be considerable. We revised the existing network-based solutions and proposed regular circuits which provide a good compromise between hardware resources and performance.**

***Index Terms*—data processing, field-programmable gate arrays, parallel processing, reconfigurable architectures, sorting.**

I. INTRODUCTION

Parallel data processing frequently uses sorting networks to enable multiple operations to be applied simultaneously. A review of recent results in this area can be found in [1-3]. Such a comparison-based technique is especially beneficial for field-programmable gate arrays (FPGA) and graphics processing units (GPU) that execute operations over streams and apply a single instruction multiple data (SIMD) strategy. The research efforts are mainly concentrated on networks with minimal depth/number of comparators [1,2] and on co-design, rationally splitting the problem between software and hardware [2]. To our knowledge, the regularity of the designed circuits and interconnections are almost never taken into account. The only report appeared in [4] where a generator for networks with reusable components was proposed. The networks in [4] were discussed just in terms of circuit sizes and performance was not shown. We would like to present research results which permit the following conclusions to be drawn:

- Although measuring the complexity of circuits is often based on the number of components used, it is not always correct because complexity of interconnections might involve significant resources exceeding resources of the components.
- In many practical applications combinational operations over data executing in one clock cycle might be slower than sequential multi-cycle operations due to difference in propagation delays.
- We found that the best designs rely on regular circuits, rationally combining parallel and sequential operations and allowing propagation delay and hardware resources to be minimized.

II. REGULAR AND EASILY SCALABLE NETWORKS

An analysis of different networks permits to conclude that even-odd transition [3] and the described in the paper max-min networks are among the most regular and easily scalable. However, they are often characterized as considerably slower and more resource consuming [3] comparing with potential alternatives such as even-odd merge and bitonic merge (which are among the fastest known [1,2]). We would like to demonstrate that such conclusion is not always correct and besides, for circuits that can potentially be implemented in FPGAs, even-odd transition and max-min networks are not slower and they are significantly less resource consuming. Let us first compare even-odd transition (EOT), even-odd merge (EOM) and bitonic merge (BM) networks on a simple example. It is known [2,5,6] that for N data items the number of comparators $C(N=2^p)$ for EOT, EOM and BM is equal accordingly: $C(N)=N\times(N-1)/2$, $C(N=2^p)=(p^2-p+4)\times2^{p-2}-1$, $C(N=2^p) = (p^2+p)\times2^{p-2}$. Yet for small values of N (let us say N=128) these networks cannot be implemented even in advanced FPGAs due to the lack of hardware resources (see the results of experiments in [2]). However, regularity of the EOT network (and also the max-min network) permits to find elegant solutions, which cannot be applied to the EOM and BM networks for which the existing methods [4] undoubtedly require numerous multiplexers and complex interconnections that increase propagation delays and decrease throughput.

Let us look at an example in Fig. 1 where the EOT network is shown for N=8 data items and it can easily be scaled for any number N. N=8 input data (27, 31, 14, 99, 62, 7, 9, and 31) are converted to output by the network of comparators, which can be described in VHDL, as shown on the top-left corner of Fig. 1a. The first network (Fig. 1a) sorts the input data and the second network (Fig. 1b) finds the items with maximum and minimum values.

The circuits in Fig. 1 can be implemented either non-sequentially or sequentially. Non-sequential (combinational) implementations have many limitations. For example, the results of [2] show that even in the relatively advanced and expensive FPGA XC5VFX130T from the Xilinx Virtex-5 family, the maximum number of input data items (of size M=32 bit) is 64. In addition, signal propagation through

This research was supported by FEDER through the Operational Program Competitiveness Factors - COMPETE and by National Funds through FCT - Foundation for Science and Technology in the context of the projects FCOMP-01-0124-FEDER-022682 (FCT reference PEst-C/EEI/UI0127/2011) and Incentivo/EEI/UI0127/2013.

Digital Object Identifier 10.4316/AECE.2013.04008

many vertical levels involves excessive delay. We suggest an alternative solution to the circuits in Fig. 1a/1b, which is outlined in Fig. 2a/2b.

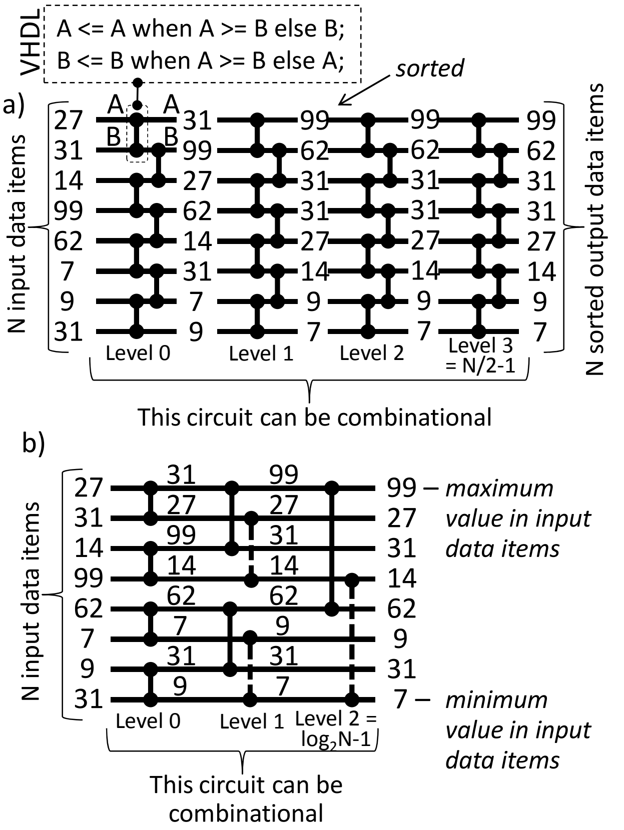

Figure 1. Networks of comparators: Network for sorting N data items (a); Network for discovering minimum and maximum values (b)

The idea is to use a feedback *Register R* and to activate different levels sequentially, still using many parallel operations at each level. Initially N data items are copied in parallel to the *Register R*. Thus, there are N multiplexers at the register inputs taking data from outside (before processing) and from the comparators (during processing). The circuits in Fig. 2 provide the following advantages:

- Hardware resources are obviously decreased. Indeed, the circuits in Fig. 1a and in Fig. 1b require N×(N-1)/2 and $N+\sum_{n=1}^{(\log_2 N)-2} 2^n$ comparators, respectively, whereas the circuits in Fig. 2a and Fig. 2b require N-1 and N/2 comparators, respectively.
- The implementation of the circuits in Fig. 2 is very regular, easily scalable for any N, and does not require complex interconnections.
- The number of paths through vertical levels of comparators is decreased. Indeed, the result of sorting in Fig. 1a is produced at level 1, but since the network is hardwired, the remaining levels (2 and 3) are involved, causing two unnecessary paths to be followed and additional propagation delay. The circuit in Fig. 2a does not involve additional iterations. As soon as the enable signal that is produced at each level is 0, sorting is finished. Thus, only the two iterations that are actually needed are executed.

Since the depth of comparators is just 2 in Fig. 2a and just 1 in Fig. 2b, the propagation delay is reduced.

The circuit in Fig. 2a sorts N input data items in T_s clock cycles and $T_s \leq N/2$ [3]. Indeed, there are N/2 levels in Fig. 1a [3] and the number of cycles in Fig. 2a is less or equal to N/2 because the result can be produced before passing through all the levels of Fig. 1a. The circuit in Fig. 2b finds the minimum and maximum values in T_f clock cycles and $T_f = (\log_2 N)-1$. Indeed, at the iteration $(\log_2 N)-1$ the results are already on the outputs of the combinational comparators.

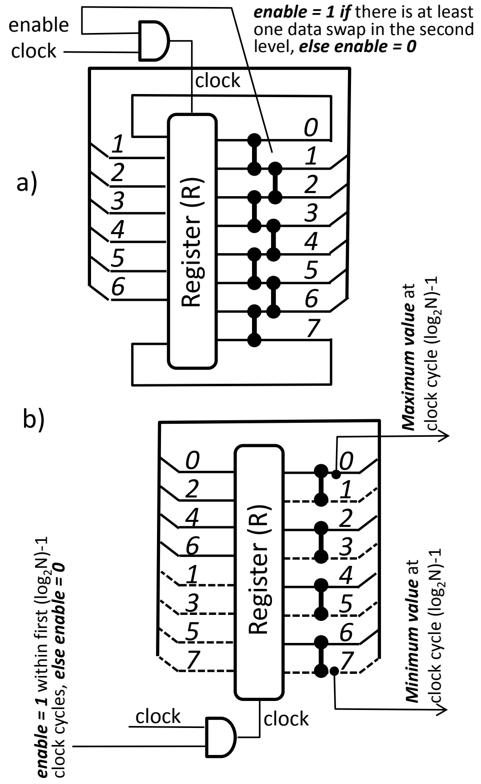
enable
clock
enable = 1 if there is at least one data swap in the second level, else enable = 0
clock
a)
Register (R)
1 2 3 4 5 6
0 1 2 3 4 5 6 7
b)
Register (R)
0 2 4 6 1 3 5 7
0 1 2 3 4 5 6 7
Maximum value at clock cycle (log2N)-1
Minimum value at clock cycle (log2N)-1
enable = 1 within first (log2N)-1 clock cycles, else enable = 0
clock
clock

Figure 2. One-level sequential circuits vs. multi-level circuits in Fig. 1: Circuit for sorting N data items (a); Circuit for discovering the minimum and maximum values (b)

The charts in Fig. 3 present the results of analysis and comparison of the circuits in Fig. 1 and Fig. 2 in hardware. All the experiments were done in the Atlys prototyping board containing one FPGA XC6SLX45 of Xilinx Spartan-6 family. Synthesis and implementation of the circuits was carried out in the Xilinx ISE 14.4 environment. The optimization goal for ISE 14.4 was set to speed and the optimization effort was set to normal.

Hardware resources for the circuits in Fig. 2 are decreased (see Fig. 3a) compared to the circuits in Fig. 1. This is obvious and does not require additional comments. What is important is that the results of experiments show that the sequential circuits in Fig. 2b have the same performance as the combinational circuits in Fig. 1b. This is because $T_f \times C_{min} \approx D$, where C_{min} is the minimum clock period of the circuits in Fig. 2b, and D is the propagation delay of the circuits in Fig. 1b. Thus, the same performance has been achieved with significantly less hardware resources and more complicated circuits have been built on the same microchip. The sequential circuits in Fig. 2a also have practically the same performance as the circuits in Fig. 1a. However, we were able to build a sequential circuit for N=256, while the available resources only allowed combinational circuits to be constructed for N≤24. Thus, the difference in problem size that can be accommodated is a factor of 10.7. The results of [2] demonstrate that even for the more advanced FPGA XC5VFX130T of Virtex-5 family, N≤64. Therefore, combinational networks only permit very limited number of data items to be processed. We found that the hardware resources required for our sequential circuits can be further decreased if comparators are built from the embedded in the FPGA XC6SLX45 digital signal processing slices DSP48A1. Two 16-bit comparators (M=16) or one 48-bit comparator (M=48) were implemented in one DSP slice and tested. The results demonstrated that the number of occupied FPGA slices can be decreased by about 20%.

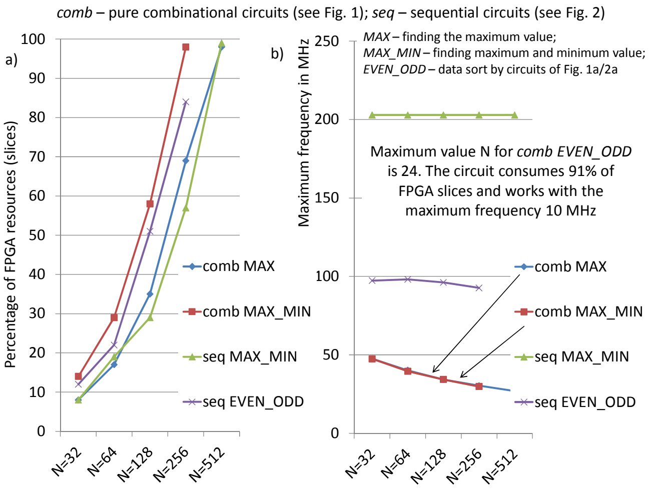

Figure 3. The results of experiments for M=32: Percentage of FPGA resources used for different values of N (a); Maximum attainable clock frequency for different values of N (b)

III. LARGE SCALE DATA SETS

Both circuits (Fig. 2a and Fig. 2 b) can be used for sorting in such a way that large data sets with Θ items are divided into sub-sets with up to N items that are sorted and then merged. For advanced FPGA, the value of N can be up to several thousands. For example, synthesis in ISE 14.4 for the Virtex-6 XC6VLX240t FPGA, without using the available 768 DSP slices 48E1, produces a circuit with the structure shown in Fig. 2a for N=1024, M=32, operating with the maximum frequency of 154 MHz and requiring 21772 (58 %) FPGA slices. Assuming that indices of the first and of the last outputs are $I_{first}=0$ and $I_{last}=N-1$, the circuit in Fig. 2b might be used as follows: 1) discovering the maximum and the minimum values; 2) incrementing I_{first} and decrementing I_{last} and repeating step 1) while $I_{first} < I_{last}$. The experiments have demonstrated that if we need to sort all data items before outputting the result, then the circuit in Fig. 2a is faster. However, if we need to output the sorted data as soon as possible, the second circuit is better and it is very fast. Indeed, once the input data are copied to the *Register R*, the first sorted item (maximum, minimum, or both) can be outputted practically immediately (the delay is less than 25 ns for N=1024, M=32 in the indicated above Virtex-6 FPGA for which 40% of slices are used with the maximum attainable clock frequency 454 MHz). All subsequent output items are produced with the same delay.

Now let us discuss the search problem (searching for the maximum/minimum value). From Fig. 2b we can see that at each clock cycle N/2 data items, which include the maximum/minimum value, will be copied to the top/bottom segment of the *Register R*. Thus, the remaining (either bottom or top) part of the *Register R* can be reused to load a new portion of data. This technique enables the maximum/minimum values in very large data sets to be found even in low-cost FPGAs (such as the considered above XC6SLX45). Fig. 4 gives necessary details.

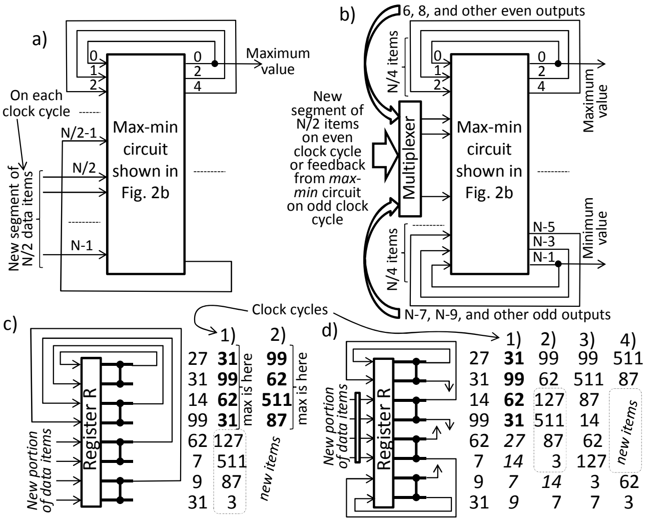

Figure 4. Using the circuit in Fig. 2b for large scale data sets: discovering the maximum value (a); discovering the maximum and the minimum values (b); example for Fig. 4a (c); example for Fig. 4b (d)

The circuit in Fig. 4a copies the even outputs of the network containing the maximum value (see Fig. 2b) to the upper N/2 M-bit words of the *Register R*. The bottom N/2 M-bit words of the *Register R* cannot contain the maximum value and may be reused to load a new portion of data items (such as 127, 511, 87, and 3 shown in the example in Fig. 4c). Since a new portion can be loaded at each clock cycle, the maximum value for data sets containing Θ items can be discovered in $\tau=2\times(\Theta-N)/N+\log_2 N$ clock cycles. For instance, if $\Theta = 2^{20} = 1\,048\,576$, N=512, then $\tau = 4103$.

The circuit in Fig. 4b discovers both the maximum and the minimum values in $\tau=4\times(\Theta-N)/N+\log_2 N$ clock cycles. At the beginning, two cycles are needed to produce (in the *Register R*) the upper N/4 M-bit words with the maximum value and the bottom N/4 M-bit words with the minimum value. After that the middle N/2 M-bit words (of the *Register R*) can be reused to load a new portion of N/2 data items and once again the maximum and the minimum values will be transferred to the upper and to the bottom quarters of the *Register R* in 2 clock cycles. Thus, $2\times(\Theta-N)/(N/2) = 4\times(\Theta-N)/N$ cycles are required to process (to upload) all data and $\log_2 N$ cycles to propagate the last portion through the max-min circuit. If $\Theta = 2^{20}$, N=512, then $\tau = 8197$.

Thus, the proposed technique enables processing large data sets in low-cost FPGAs with external memory supplying input data. Fast data exchange between the FPGA and a higher-level host system can be provided through high-performance on-chip interface such as that is available for Zynq all programmable system-on-chip [7].

IV. EXPERIMENTS

Synthesis in Xilinx ISE 14.4 for different FPGAs demonstrates that the proposed circuits (see Fig. 2) for up to thousands of items (M=32) can be implemented in one (more advanced) FPGA microchip and they operate with high clock frequency (see Sections II, III where the results of some projects are given).

From Fig. 3b we can see that the throughput of different circuits varies from 185 to 195 million items per second. Indeed, sorting 256 items (32-bit each) by the circuit in Fig. 2a can be done with the maximum clock frequency F_{max} = 92.6 MHz = 92 600 000 Hz. If the enable signal is not used (*i.e.* for the worst case) then 128 iterations are required to sort 256 items. Thus, we spend t_{item} = $\{[(10^9$ / 92 600 000 Hz) × 128 iterations] / 256 data items$\}$ ns per data item, and the throughput is 10^9 ns / t_{item} = 185 200 000 items per second. It is easy to show that the throughput for the circuit in Fig. 2a for any value of N is equal to [F_{max}(in Hz) × 2] items per second. From Fig. 3b we can see that the maximum/minimum value for N items can be found in $t_{max-min}$ = [(10^9 / 202 865 000 Hz) × $\log_2 N$ iterations] ns and, thus, for N=512, $t_{max-min}$ = 45 ns. For pure combinational max-min circuit from Fig. 1b and N=256, $t_{max-min}$ = 34 ns. Note that a combinational circuit for N=512 cannot be implemented in the FPGA XC6SLX45. If we compare the results above with [1,2,4,8-20] we can see that our circuits provide either similar or better throughput. For example, in [2] processing median operators in FPGA of Virtex-5 family for large data sets requires approximately 19 ns per data item. In our case this time is about 5.5 ns. Sorting of 16×2^{20} data items in GPU in [1] took between 500 and 1000 ms. So, the throughput is ≤ 34 million data items per second.

To demonstrate usefulness of the proposed circuits and their applicability to large data sets, a hybrid merge sort algorithm (similar to [8]) was described in C++ and used *without* (i.e. as a pure merge sort) and *with* the proposed circuits. For the latter case the number N in each preliminary sorted by the circuit in Fig. 2a subset was changed from 8 to 2048 and the size Θ of the initial (unsorted) set was taken as 1, 10, and 100 millions of data items. 1000 experiments with randomly generated data were done for each instance characterized by N/Θ and an average value was taken. The acceleration achieved varies from 1.4 (for smaller values of N) to 4.6 (for larger N). Thus, the more data are pre-sorted in FPGA in each sub-set the faster results are produced.

The analysis of Fig. 3 clearly demonstrates that resources for the proposed solutions are significantly smaller comparing to [2,4,8,9,10,12,13,14] allowing larger data sets to be handled in FPGA and permitting communication overhead to be reduced (*e.g.* burst mode can be applied more efficiently for transmitting larger sets of data). The comparison of performance and resources of the proposed networks with other known results leads to the following conclusions:

- Neither from the FPGA-based implementations [2,4,8,9,10,12,13,14] permits to construct circuits with so small hardware resources.
- The proposed solutions permit better compromise between resource consumption and performance to be achieved than in the other known designs.

V. CONCLUSION

Sequential circuits that implement functionality similar to combinational sorting networks are proposed and discussed. They consume significantly less resources and, thus, the same hardware can be used for processing substantially larger sets of data with similar performance. Furthermore, outputting sorted and maximum/minimum values can be done with a very small delay even in low-cost microchips.

ACKNOWLEDGMENT

The authors wish to thank Ivor Horton for very useful comments and suggestions.

REFERENCES

[1] G. Gapannini, F. Silvestri, and R. Baraglia, “Sorting on GPU for large scale datasets: A through comparison,” Information Processing and Management, 2012, vol. 48, no. 5, pp. 903–917.

[2] R. Mueller, J. Teubner, and G. Alonso, “Sorting Networks on FPGAs,” The International Journal on Very Large Data Bases, vol. 21, no. 1, 2012, pp. 1-23.

[3] *GPU Gems*, Improved GPU Sorting. [Online]. Available: http.developer.nvidia.com/GPUGems2/gpugems2_chapter46.html

[4] M. Zuluada, P. Milder, and M. Puschel, “Computer Generation of Streaming Sorting Networks,” in *Proc. 49th Design Automation Conf.*, San Francisco, June, 2012, pp. 1245-1253.

[5] D.E. Knuth, The Art of Computer Programming. Sorting and Searching, vol. III. Addison-Wesley, 1973.

[6] K.E. Batcher, “Sorting networks and their applications,” in *Proc. AFIPS Spring Joint Computer Conf.*, USA, 1968, pp. 307-314.

[7] Xilinx Inc., Zynq-7000, All Programmable SoC, 2013. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf

[8] R.D. Chamberlain and N. Ganesan, “Sorting on Architecturally Diverse Computer Systems,” in *Proc. 3rd Int. Workshop on High-Performance Reconfigurable Computing Technology and Applications – HPRCTA’09*, USA, 2009, pp. 39-46.

[9] J. Ortiz and D. Andrews, “A Configurable High-Throughput Linear Sorter System,” in *Proc. of IEEE Int. Symp. on Parallel & Distributed Processing*, April, 2010, pp. 1-8.

[10] D.J. Greaves and S. Singh, “Kiwi: Synthesis of FPGA circuits from parallel programs,” in *Proc. 16th IEEE Int. Symp. on Field-Programmable Custom Computing Machines - FCCM’08*, USA, 2008, pp. 3-12.

[11] S. Che, J. Li, J.W. Sheaffer, K. Skadron, and J. Lach, “Accelerating Compute-Intensive Applications with GPUs and FPGAs,” in *Proc. Symp. on Application Specific Processors – SASP’08*, USA, 2008, pp. 101-107.

[12] R. Mueller, Data Stream Processing on Embedded Devices. Ph.D. thesis, ETH, Zurich, 2010.

[13] D. Koch and J. Torresen, “FPGASort: a high performance sorting architecture exploiting run-time reconfiguration on FPGAs for large problem sorting,” in *Proc. 19th ACM/SIGDA Int. Symp. on Field Programmable Gate Arrays – FPGA’11*, USA, 2011, pp. 45-54.

[14] V. Sklyarov, I. Skliarova, D. Mihhailov, and A. Sudnitson, “Implementation in FPGA of Address-based Data Sorting,” in *Proc. 21st Int. Conf. on Field-Programmable Logic and Applications – FPL’11*, Greece, 2011, pp. 405-410.

[15] X. Ye, D. Fan, W. Lin, N. Yuan, and P. Ienne, “High Performance Comparison-Based Sorting Algorithm on Many-Core GPUs,” in *Proc. IEEE Int. Symp. on Parallel & Distributed Processing – IPDPS’10*, USA, 2010, pp. 1-10.

[16] N. Satish, M. Harris, and M. Garland, “Designing efficient sorting algorithms for manycore GPUs,” in *Proc. IEEE Int. Symp. on Parallel & Distributed Processing – IPDPS’09*, Italy, 2009, pp. 1-10.

[17] D. Cederman and P. Tsigas, “A practical quicksort algorithm for graphics processors,” in *Proc. 16th Annual European Symp. on Algorithms – ESA’08*, Germany, 2008, pp. 246–258.

[18] C. Grozea, Z. Bankovic, and P. Laskov, “FPGA vs. Multi-Core CPUs vs. GPUs,” in *Facing the multicore-challenge*, R. Keller, D. Kramer, and J.P. Weiss (Eds), Springer-Verlag Berlin, Heidelberg, 2010, pp. 105-117.

[19] M. Edahiro, “Parallelizing fundamental algorithms such as sorting on multi-core processors for EDA acceleration,” in *Proc. 18th Asia and South Pacific Design Automation Conf. - ASP-DAC’09*, Japan, 2009, pp. 230-233.

[20] H.S. Stone, "Parallel Processing with the Perfect Shuffle," IEEE Transactions on Computers, vol. C-20, (2), 1971.