A Comparison-Free Sorting Algorithm

Saleh Abdel-hafeez[1] and Ann Gordon-Ross[2]

[1]Jordan University of Science and Technology
IRBID 22110, Jordan, sabdel@just.edu.jo
[2]University of Florida, FL. 36211, USA, ann@ece.ufl.edu
Also with the NSF Center for High-Performance Reconfigurable Computing (CHREC) at the University of Florida

Abstract

We propose a novel sorting algorithm that sorts data elements without data comparison operations—a comparison-free sort. Our hardware-based sorting algorithm leverages Hamming memory, which is an SRAM-based memory structure that stores the data elements based on the elements' Hamming maximum order representations. The data elements are also stored in a serial shift buffer in binary representation, and a simple matrix multiplication between this buffer and the Hamming memory produces the outputted sorted elements in 2*N* clock cycles for *N* data elements.

Keywords- Hardware-based sorting, comparison-free sorting, SOC design, Hamming memory, Hamming maximum order representation, 8T-CELL memory.

I. Introduction and Motivation

Prior research in sorting algorithms must consider the complexity of efficiently sorting data elements while maximizing the capabilities of the available computing resources, thus making efficient hardware realization challenging [1][2]. Sorting algorithms iteratively move data between comparison units and memory, requiring wide, high-speed data buses, complex control logic, and numerous shift, swap, comparison, etc. operations [3][4], thus requiring special design considerations for scalability to big data and specialization for certain data-type particulars.

We propose a new sorting algorithm that leverages the data elements' binary and Hamming weight representations to sort the data elements without comparison operations. A simple matrix multiplication (ANDING) operation outputs the sorted data elements, and the associated hardware structure alleviates the iterative movement of data elements between the memory and processing units. Our sorting algorithm's complexity is on the order of O(*N*), which makes our sorting method suitable for a wide range of sorting applications, and is competitive with state of the art sorting methods.

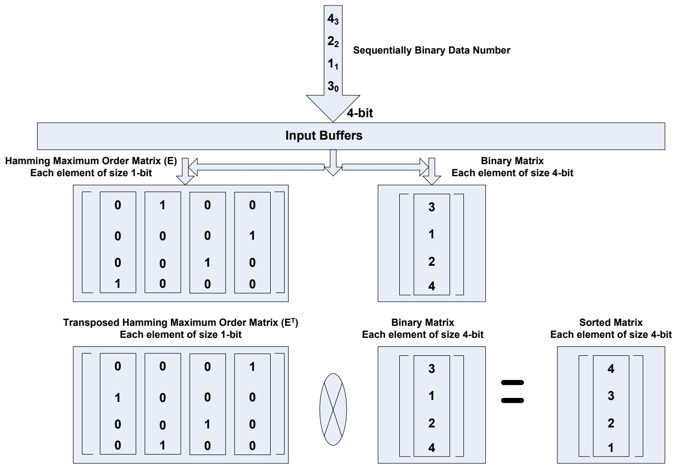
4_3
2_2
1_1
3_0
Sequentially Binary Data Number
4-bit
Input Buffers
Hamming Maximum Order Matrix (E)
Each element of size 1-bit
Binary Matrix
Each element of size 4-bit
Transposed Hamming Maximum Order Matrix (E^T)
Each element of size 1-bit
Binary Matrix
Each element of size 4-bit
Sorted Matrix
Each element of size 4-bit

Fig. 1 Sorting example using matrix multiplication operations considering a 4-bit data input bus.

II. Comparison-free Sorting Algorithm

The sorting algorithm's input is an *m*-bit bus carrying the data element's binary representation, which enables sorting $N=2^m$ data elements where each element has a Hamming representation of size $K=N$ for a lossless representation. For example, 5 has a binary representation of "101", and could have several Hamming representations, such as "10101011", "11100011", "00111110", etc. (i.e., covering all possible maximum order representations). However, our binary-to-Hamming converter deterministically converts 5 to "00011111", with a Hamming maximum order representation of "00010000". This Hamming maximum order representation ensures that different elements are orthogonal with respect to each other when projected to a R^n linear space.

Our sorting algorithm operates in two sequential phases: the write phase and the read phase. During the write phase, the data elements are sequentially inputted, converted to the element's Hamming representation, and stored into an SRAM-based memory [5] with a counter-based decoder address [6] in Hamming maximum order representation. We refer to this memory as a Hamming memory due to our Hamming representation storage methodology. The data elements are interpreted as a two-dimensional (2D) Hamming matrix *E* of size *N*x*K* where every element of the Hamming memory/matrix is of size 1-bit. In parallel, the element's binary representation is also sequentially stored in a serial shift buffer of registers, creating a one-demensional (1D) binary matrix *B* of size *N*x*1,* where each register is of size *m*-bit. Since there are *N* data elements, the write phase requires *N* clock cycles.

The read phase effectively sorts and outputs the data elements using a matrix multiplication (ANDING) operation, rather than comparison operations, as in prior work. The matrix multiplication multiplies the transposed 2D Hamming representation matrix E^T (i.e., the transpose of *E*) with the 1D binary matrix *B*. This multiplication essentially enables a read from the associated binary matrix *B*'s register that is aligned with a '1' in the read column of the Hamming matrix *E*. The result is the sorted matrix $S=S^T$xB, where *S* is of size *K*x1-bit

```
Input: integer Element[0 : n – 1]
Output: integer Sorted[0 : n – 1]
Hamming memory: Boolean H[0 : n – 1][ 0 : n – 1] initialize to zero
while i < n-1 do
    H[i][Element[i]-1] ← 1
endwhile
k ← 0
while j >= 0 do
    while i < n-1 do
        then if H[i][j] = 1
            then Sorted[k] ← Element[0: n-1]
            k ← k+1
        endif
    endwhile
endwhile
```

Fig. 2 Pseudo code for our sorting algorithm assuming a uniprocessor system with no threading.

and contains the sorted data elements that are shifted into a sorted shift buffer, and outputted after the read phase completes.

Duplicated data elements are represented using the same vector space, such that the corresponding Hamming matrix column has multiple '1' values. These multiple '1's enable multiple registers in the binary matrix and these registers store duplicate data elements. Therefore, our sorting algorithm counts the number of '1's in the Hamming representation matrix column using simple control logic, and sends the repeated register value to the sorted shift buffer.

Fig. 1 illustrates a sorting example for four 4-bit data elements {3,1,2,4}, which generates the sorted matrix (sorted shift buffer) S = {1,2,3,4}. Fig. 2 shows the pseudo code for our sorting algorithm, assuming a single-threaded uniprocessor system (future work will extend this to multi-threaded multiprocessor systems).

III. Comparison-free Sorting Hardware Data Path and Functional Details

Fig. 3 depicts a block diagram for our sorting algorithm's data path assuming a sample $m=10$-bit input bus, which sorts $N=2^m=1024$ distinct data elements. The binary-to-Hamming converter generates the Hamming maximum order representation by converting the m-bit binary representation to the N-bit Hamming representation using a simple one-hot decoding unit, which directly connects to the SRAM-based 8T-Cell [5][6] Hamming memory's input bus. During the write phase, the data elements are stored serially into the serial shift buffer, and in parallel, the data elements are converted to their Hamming representation and stored in the Hamming memory in row order. After all elements are written to the Hamming memory and the serial shift buffer, the read phase processes the elements from the Hamming memory in column order. This transpose during the read phase facilitates the matrix multiplication (ANDING) operation E^TxB, where E^T is of size KxN and B is of size Nx1, and produces the sorted output.

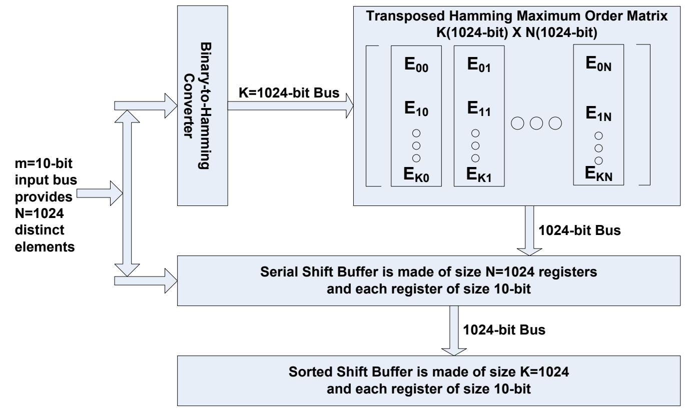
Binary-to-Hamming Converter
K=1024-bit Bus
Transposed Hamming Maximum Order Matrix K(1024-bit) X N(1024-bit)
E00
E01
E0N
E10
E11
E1N
EK0
EK1
EKN
m=10-bit input bus provides N=1024 distinct elements
1024-bit Bus
Serial Shift Buffer is made of size N=1024 registers and each register of size 10-bit
1024-bit Bus
Sorted Shift Buffer is made of size K=1024 and each register of size 10-bit

Fig. 3 Block diagram of our sorting algorithm's hardware data path using Hamming memory.

IV. Conclusions

We presented a novel comparison-free sorting algorithm and associated hardware implementation. To the best of our knowledge, our design is the first to simultaneously leverage the data elements' Hamming weight and binary representations to sort the data elements without any comparison operations, and using only a simple matrix multiplication (ANDING) operation. Since comparison operations are eliminated, a key contribution of our sorting method is the elimination of the associated comparison units' high power dissipation. Additionally, our sorting algorithm is data-type/ordering/duplication independent, and can sort N data elements in $2N$ clock cycles—on the order O(N)— using simple control logic. We implemented and evaluated our sorting algorithm for a sample sorting of N=1024 data elements using 90 nm TSMC technology at 1V and an 8T-cell memory. Results verified that the sorting requires $2N$=2048 clock cycles at an operating frequency of 0.25 GHz.

V. Acknowledgments

This work was supported in part by the National Science Foundation (CNS-0953447). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

VI. References

[1] Enzo Mumolo, Gabriele Capello, and Massimiliano Nolich, VHDL Design of a Scalable VLSI Sorting Device Based on Pipelined Computation, Journal of Computing and Information Technology, Vol. 12, pp. 1-14, 2004.

[2] A. A. Colavita, A Cicuttin, F. Fratnik, and G. Capello, SORTCHIP: A VLSI Implementation of a Hardware Algorithm for Continuous data Sorting, IEEE Journal of Solid-State Circuits, Vol. 38, No. , pp. 1076-1079, June 2003.

[3] Li Xiao, Xiaodong Zhang, Stefan A. Kubricht, Improving Memory Performance of Sorting Algorithms, ACM Journal on Experimental Algorithmics, Vol. 5, 1-21, 2000.

[4] L. M. Busse, M. H. Chehreghani, J. M. Buhmann, The Information Content in Sorting Algorithms, IEEE International Symposium on Information Theory Proceedings (ISIT), pp. 2746-2750, 2012.

[5] Saleh Abdel-Hafeez and Anas Matalkah, "CMOS Eight-Transistor Memory Cell for Low-Dynamic-Power High-speed Embedded SRAMS," Journal of Circuits, Systems and Computers, Vol. 17, No. 5, pp. 845-863,Oct. 2008.

[6] Saleh Abdel-Hafeez and Ann Gordon-Ross, "A Digital CMOS Parallel Counter Architecture Based on State Look-Ahead logic", Journal of IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Vol. 19, Issue 6, pp. 1023-1034, May 23, 2011.