BulkSort: System Design and Parallel Hardware Implementation Considerations

Soukaina Ihirri[1], Ahmed Errami[2], Mohammed Khaldoun[3], Essaid Sabir[4]

NEST Research Group, LRI Lab., ENSEM, Hassan II University of Casablanca, 20000, Morocco[1,2,3,4]
LPRI, EMSI, Casablanca, Morocco[1].

***Abstract*—Algorithms are commonly perceived as difficult subjects. Many applications today require complex algorithms. However, the researchers look for ways to make them as simple as possible. In high time demanding fields, the process of sorting represents one of the foremost issues in the data structure for searching and optimization algorithms. In parallel processing, we divide program instructions among multiple processors by breaking problems into modules that can be executed in parallel, to reduce the execution time. In this paper, we proposed a novel parallel, re-configurable and adaptive sorting network of the BulkSort algorithm. Our architecture is based on simple and elementary operations such as comparison and binary shifting. The main strength of the proposed solution is the ability to sort in parallel without memory usage. Experimental results show that our proposed model is promising according to the required resources and its ability to perform a high-speed sorting process. In this study, we take into account the analysis result of the Simulink design to establish the required hardware resources of the proposed system.**

***Keywords*—Sorting; FPGA; bulk-sort; parallel processing**

I. Introduction

Sorting is taken as one of the most fundamental non-numerical algorithms needed in a multitude of applications. The operation of sorting data became an integral part of many large scale scientific and commercial applications such as data centers, database management or digital signal processing.These applications require parallel processing. Thus, the parallel version of sorting is one of the most required, for which the transition is sophisticate because it demands communication as well as computation. Many sorting algorithms have been developed over the decades. Most important are: Quick sort[1] [2]; Merge sort[3]; Parallel odd-even[4]; bubble sort[5]; Selection sort[6]. The quick sort is a divide-and-conquer algorithm[7] that sorts a sequence by recursively dividing it into smaller sub sequences. The limitation of its parallel version is that it performs the partitioning step serially. Its formulation makes it amenable for parallelization using task parallelism but it impacts the algorithm's scalability. The complexity of the quick sort is $O(n \log n)$ where n is the size of the array. Merge sort is a divide and conquer algorithm where data is divided into two halves and assigned to processors until individual numbers are obtained. After this, each two pair's numbers are merged into sorted list of 2 numbers. This sorted list is again merged to make 4 sorted numbers. This continues till the fully sorted one list is obtained. Merge sort is also easily applied to lists, not only arrays because its worst-case running time is (n log n). Parallel odd-even transposition is an extension of bubble sort, operates in two alternate phases. Even phase in which, values are exchanged between even processors, while the odd processors exchange their values in the odd phase. Its time complexity is$O(n^2)$. Selection sort is an in-place comparison sort. The algorithm finds the minimum value, swaps it with the value in the first position, and repeats these steps for the remainder of the list. It has a hypothetical complexity of $O(n^2)$. Although this algorithm is very slow for sorting larger amount of data, yet is simple.

The difference between these sorting algorithms can be seen in a view of measure of the amount of time and/or space required by an algorithm for an input of a given size (n)[8]. Now-a-days, the amount of information grow rapidly [9] by that a high speed computing to process this huge amount of data[10] is required. So, High performance computing involves parallel processing [11]. A number of research efforts explore how data bases can use the potential of modern hardware architecture. The majority of sorting architectures implemented in hardware use batcher even odd & Bitonic mergers because they are the fasted. Technical literature has called a model frequently used to study sorting algorithms, the sorting networks which have received much interest because of their widespread use in many computations. They represent an abstract machine which accesses the data only through compare-exchange operations by the use of compactors, which are wired together to implement the capability of general sorting. But a various sorting architectures have been presented. Bucket Sorter or FIFO as well as tree based merge sorter which is considered as a target designs for implementation. Bucket sorter follows divide and conquer strategy by distributing the elements of an array into a number of buckets. Each bucket is then sorted individually. FIFO-Based Merge Sorter[12] based on multiple FIFO merge sorters cascaded for sorting on a continuous stream. It shows excellent hardware resource utilization efficiency but requires high buffer memory usage. Merge sorter trees used to merge long sequences from external memory. Elements arranged in trees based structure for sorting sorted sub-sequences to one combined sequences that will be fully sorted. Sorting networks[13] which are models based on algorithms that sort a fixed sequence of numbers by using a fixed sequence of comparators and wires. They require greater number of I/O throughput. Insertion sorter Hardware[14] provides a shift register for storing the search keys.

In view of circuit types, sorting architectures are classified

according to 3 categories[13]: combinational sorting circuit, synchronous sorting circuit[15], and piplined circuit[16]. Combinational sorting circuit which is operated without clock signal; signal goes through the sorting stage without following a synchronization signal. The key characteristic of this type is the absence of registers. In view of metrics, the path delay is difficult to estimate, the max delay path depends on number of stages and comparators. Synchronous sorting circuit is a set of stages separated by registers. Signal in this category follows a synchronization signal as a clock. While the pipeline's implementation of sorting network require a new input set every clock cycle. Introducing only registers doesn't make a fully pipelined. Registers are required to buffer the value of wires between stages. The basic element of sorting architecture is the comparison element. It receives 2 numbers on its inputs and presents their Min or Max.

Up to now, there is no easy way to make hardware sorters run in parallel[17]. Research efforts now-a-days are concentrated on network with minimal depth or number of comparators. In parallel processing, program instructions are divided among multiple processors by breaking problems into modules that can be executed in parallel, with the goal of less time execution. Every processor comprehends its piece of general computational issue. But due to the limited number of I/O ports, the existing parallel hardware sorters can sort up to only hundreds of numbers. As sorting large datasets may impose undesired performance degradation too, acceleration units coupled to the embedded processor can be an interesting solution for speeding-up the computations.

Sorting can be implemented in several ways using different technologies. FPGA (Field-Programmable Gate Array)-based systems with re-programmability [18] have become popular for realizing sorting because of the ability to make a trade of between energy and performance. Sorting networks require greater I/O throughput as they consume more sort keys and produce a huge amount of data at the same time. FPGA can be used to almost sorting application. But, sorting a huge amount of data means that it cannot fit into FPGA memory, because of the lack of hardware resources. Despite the limitation in amount of chip space to accommodation functions' parallelism or to sort a huge amount of data, this problem can be managed and estimated. Therefore, FPGA can be added as additional process unit in standard CPU sockets[19] [20]. In some cases, an external memory is required to store intermediate values. This make FPGA good candidates for multicore systems. Latency, throughput and Memory have been used as the metrics for performance evaluation of sorting implementation.

In this context, we presented a novel Bucket sorter architecture designed for the BulkSort algorithm[21]. After the modeling of the BulkSort algorithm, by the use of mathematical equations and the validation by the use of the C++ simulation, we were able to present the system design of the proposed algorithm. We use the Xilinx ISE for FPGA product in order to synthesize our Simulink design to finally present the hardware performance of the pipeline system. Our approach can at best produce $O(nlog_2(n))$-time parallel sorting algorithms. Since a serial simulation that sorts by comparison requires at least $O(n)$ comparisons. The optimal speedup would be achieved when; by using n processors; n elements are sorted in $O(log_2(n))$ parallel comparisons.

This paper is organized as follows. Section II briefly introduces the general idea of the BulkSort. The approach of the BulkSort is presented in Section III, and Section IV provides details about the C++ simulation of the algorithm. System design of our proposed model and its sub-blocks are presented in Sections V and VI. Hardware implementation consideration are presented in Section VII. Finally, Section VIII presents some concluding remarks and perspectives.

II. General idea of the BulkSort

BulkSort, is a novel sorting algorithm that has been presented in [21]. It is an adaptive and parallel algorithm ; based on divide and conquer technique; designed to be implemented in a parallel and re configurable machine for sorting numbers. The concept is easy to understand, we examine the first bit (Must Significant Bit (MSB)) of the concerned unsorted set of elements. After each comparison, each sub set is divided into 2 subsets, winners and losers, each of which undergo the same roles. Fig. 1 illustrate the concept of BulkSort. A set of 6 numbers (5, 2, 10, 4, 6, 8, 10) are firstly converted into binary sequence.Based on a comparison,the whole set is divided into 2 subset representing respectively winners and losers until we have one element elected as the winner of the group.

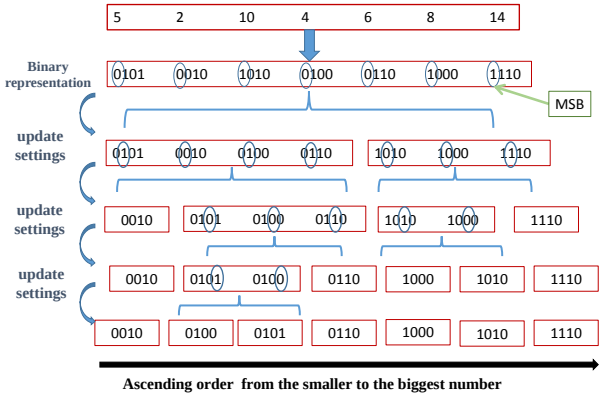

Fig. 1. The Main idea of Bulk Sort [21]

In a binary representation, we have two cases, either 0 or 1.Element with 1 represent a winner, while 0 represent the loser of a subset. The proposed process is based on an iterative pipeline system. Sequence of iterations are executed until a single element is elected as the group winner. Each element in the group is characterized by certain parameters which specify respectively its rank, position, the concerned bit in the current iteration and finally its state (if it is going to be compared or not with the set). We mention that the BulkSort is an iterative algorithm where the iteration is a computation instruction process which loops until stopped condition[21].

III. The Approach of the Bulk Sort

The approach of our algorithm is as follows: all the elements will be compared in a pipe stage system (see Fig. 2).

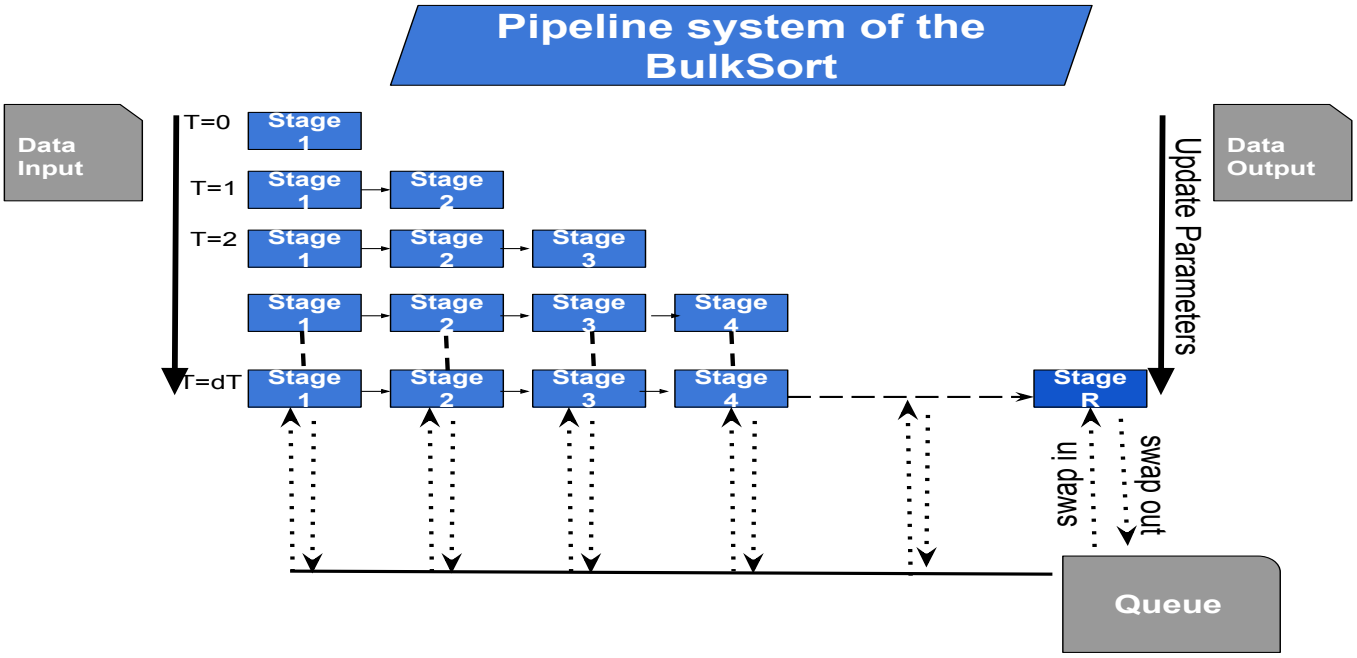

Fig. 2. The approach of the Bulk Sort [21]

We assume that each stage of our system represent a compactor that can compare from 2 to N elements at the same time. At the beginning the set of MSBs of unsorted set will be compared in the first stage. Once we have at least one loser, the whole will be divided into two stages.Then we move to compare the next bit of the 2 sub groups in each stage. And so on until all stages are filled. In this case we are in a scenario where the set of elements are going to be either in a processing or pending state. Elements in the pending state will wait (in the queue) until the concerned stage is emptied. We highlight that elements transit from the pipe to the queue according to the availability of the stages and this process is automatically managed away by a system of mathematical equations.

At each iteration, the parameters of all unsorted elements are updated; according to a specific equations as mentioned below; depending on their state, either processing or pending and if the element is winner or loser. The principal notation used in the following equations are summarized:

- i refers to the current iteration of the sorting process,
- j represents the index of an element in the array, starting in one.
- P_j refers to an element j of the set of the unsorted elements,
- Z_i^j represents the stage index in which the element P_j is going to be processed,
- X_i^j represents the bit index of an element,
- Y_i^j indicates the iteration the element with index j will be processed,
- R_i^j refers to the rank of the element P_j,
- SR_i^k indicates the result of a comparison between elements of the same stage. The value of this parameters is either 0 or 1, depending whether there is a loser or not in the concerned stage k,
- R represents the total number of stages.

A For the processing elements:
Each element in the processing state updates its parameters automatically following the equation presented below:

1 For the rank (R):

- The rank of the winner is computed as follow:

$$R_i^j = R_{i-1}^j - \sum_{k=0}^{Z_{i-1}^j} SR_i^k \quad (1)$$

- The rank of loser is:

$$R_i^j = R_{i-1}^j - \sum_{k=0}^{Z_{i-1}^{j-1}} SR_i^k \quad (2)$$

2 The stage (Z):

- Z_{i+1}^j For winners:

$$Z_{i+1}^j = modulo(Z_i^j + \sum_{k=0}^{Z_i^{j-1}} SR_i^k, R) \quad (3)$$

- Z_{i+1}^j For losers:

$$Z_{i+1}^j = modulo(Z_i^j + \sum_{k=0}^{Z_i^j} SR_i^k, R) \quad (4)$$

3 The iteration index (Y):
The parameter Y, which refers to the iteration where the element will be processed depend on the the result of the parameter Z as follow:

- If $Z_{i+1}^j = 0$

$$Y_{i+1}^j = Y_i^j + \frac{\sum_{k=0}^{Z_i^{j-1}} SR_i^k + Z_i^j}{R} + 1 \quad (5)$$

- If $Z_{i+1}^j >= 1$

$$Y_{i+1}^j = Y_i^j + \frac{\sum_{k=0}^{Z_i^{j-1}} SR_i^k + Z_i^j - 1}{R} + 1 \quad (6)$$

4 Bit Index (X):
This parameter is incremented automatically by one, except the case where element is pending

$$X_{i+1}^j = X_i^j + 1 \quad (7)$$

B For the pending elements:
The difference between calculation in the case of pending elements and the processing ones lies in the sum of SR_i^k(indicates the result of a comparison between elements of the same stage. The value of this parameters is either 0 or 1, depending whether there is a loser or not in the concerned stage k) which go from 0 to the total number of stages(R). And also in pending state, we don't talk about loser and winner. Both have the same mathematical equations.

1 The rank(R):

$$R_i^j = R_{i-1}^j - \sum_{k=0}^{R} SR_i^k \quad (8)$$

2 The pipe stage (Z):

$$Z_{i+1}^j = modulo(Z_i^j + \sum_{k=0}^{R} SR_i^k, R) \quad (9)$$

3 The iteration index (Y):
For the reason of dependency, we have to check the value of Z each time we compute the iteration index

- If $Z_{i+1}^j = 0$

$$Y_{i+1}^j = Y_i^j + \frac{\sum_{k=0}^{R} SR_i^k + Z_i^j}{S} + 1 \quad (10)$$

- If $Z_{i+1}^j >= 1$

$$Y_{i+1}^j = Y_i^j + \frac{\sum_{k=0}^{R} SR_i^k + Z_i^j - 1}{S} + 1 \quad (11)$$

4 Bit Index (X):
This parameter remains in standby until the element goes to the processing state.

$$X_{i+1}^j = X_i^j \quad (12)$$

A numerous test has been done (A C++ simulation) to validate the proposed sorting algorithm and examine the proposed idea [21].

We should highlight that the BulkSort is a reconfigured algorithm in view of number of stages. We have proved that by increasing the number of stages, the system become faster. It was also found that we recorded a minimum iteration for very important data numbers.

IV. C++ Simulation

To validate and examine performance of the proposed sorting algorithm; a program was developed in C++, and test results are demonstrated using Intel(R) Core(TM) i5-2430M CPU @ 2.40GHz/2394 MHz. Many experiments with different size of data, different number of pipe stages(communication links) were carried out. We should highlight that the proposed algorithm is iterative and each iteration refers to a clock cycle; in which several operation are executed in parallel.

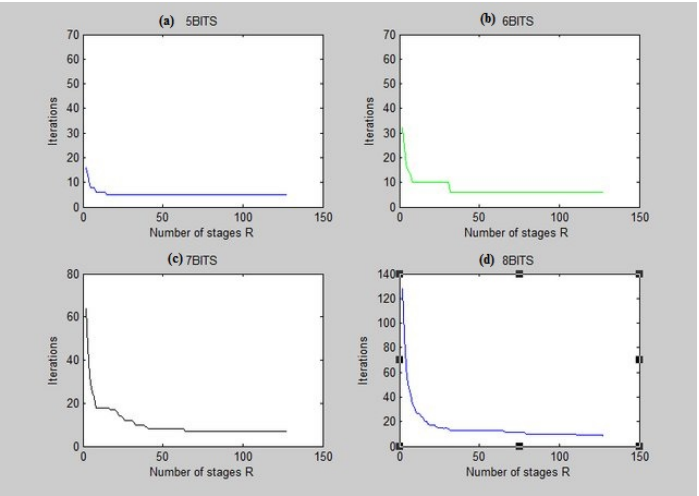

Fig. 3. Illustrate the-number of iteration according to the number of pipe stage.(a),(b),(c) and (d) represent how number of iterations change according to number of stages in case of 5,6,7 and 8 bits [21].

The number of pipe stages is taken as input parameters. By means of this point, our algorithm can be re-configurable. It is found that the number of iterations becomes a constant as the number of pipe stages increases after a while.As shown in Fig. 3, increasing the number of pipe stages until we achieved iteration equal to N (N refers to the total number of bits)with $\frac{2^N}{2}$ pipe stages. It is also found that even with a minimum number of pipe stages (1 pipe stage), the number of iteration is equal to the number of unsorted elements. Our results verify that our algorithm is re-configurable in view of number of pipe stages. More stages we have, the more the system gets faster. A careful analysis reveals also that the number of iterations can be reduced.

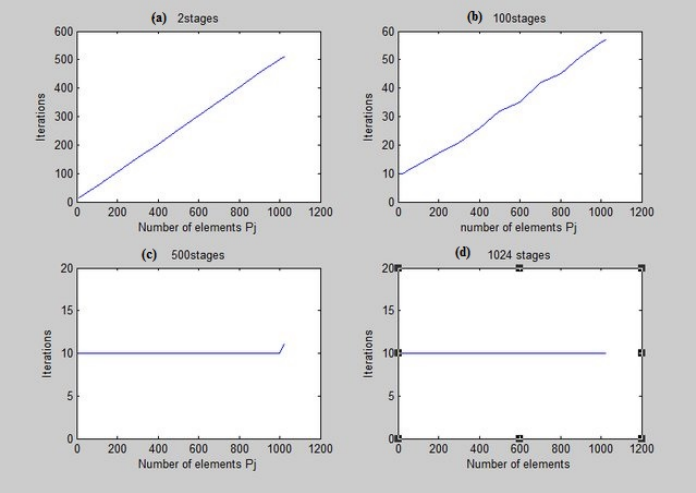

Fig. 4. Illustrate the-number of iteration according to the number of elements.Figures (a),(b),(c) and (d) show the behaviour of number of iterations for different number of stages, according to number of elements. [21].

Based on our experiments (see Fig. 4), we reveal that our algorithm recorded minimum sorting iteration for very large data numbers. For example for N=10 bit, the max number of elements is 1024=2^N. With 1 pipe stage, the number of iteration is 2^N , while if we use for example 50 pipe stages, we have just 28 iterations. while with 500 pipe stages, we can reach $10 = log(1024)$ Iterations for 1024 elements. According to more careful analysis, the complexity in view of number of iteration to sort n elements is O(n logn) in the best case and O (n) in its worst case. The main iteration based on a series of computation of the element's rank and parameters used in the next iteration of the sorting process.

V. The BulkSort System design

Now it is the time to learn the philosophies of the proposed sorting architecture and the corresponding SIMULINK model. We should mention that certain parameters in this section will be presented otherwise. The clock cycle represents the iteration and the bus is equivalent to the stage of pipeline system.

The BulkSort process relies on synchronous architecture; based on parallel processing distributed among several identical units. Our system performs the process of sorting ; based on a decentralized comparison between these bits; starting from the most significant bits and ending after several clock cycles by the least significant bits. We should emphasize that in hardware, the choice of the data do not affect the sorting network, it affects only the implementation of compactors.

The main strength of our system is the ability to compare a set of elements simultaneously instead of going through them one by one. In order to make the proposed model easy to understand, we use Fig. 5 to illustrate the general model. The BulkSort architecture is composed of n cells PEi with $1 \leq i < n+1$. These cells are interconnected via a communication system distributed between 3 families of lines. Each of these cells is made up of seven units (see Fig. 6).

The process of the bulk Sort system involves several data elements classified into two families: A first family of data

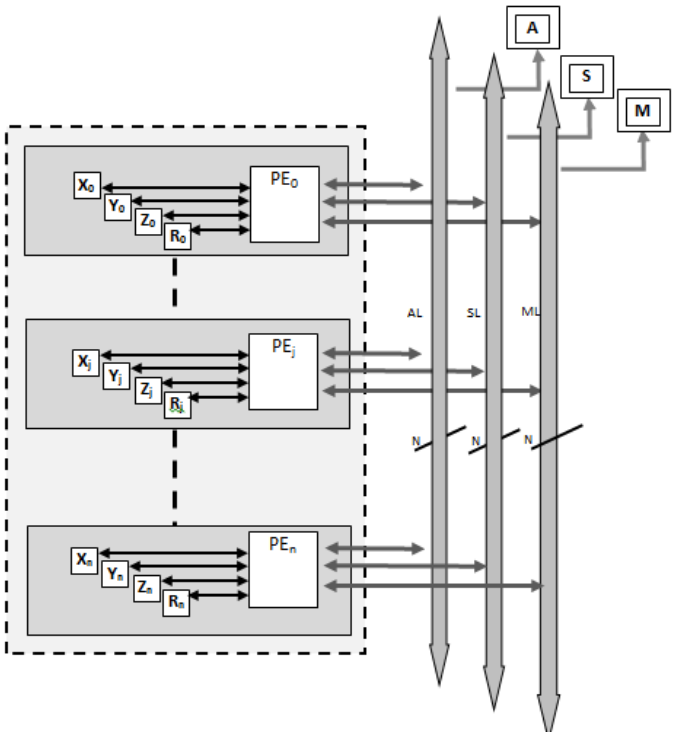

Fig. 5. The external interface of the BulkSort Model.Each Process Element(PE) represent an element to be sorted.It's characterized by the set of parameters.A,S,M represent the communications lines(Buses)

which are distributed among all the processing units(PE_i) (X_i, Y_i, Z_i and R_i of Fig. 5). The second family of data, whose elements are shared between all PE_i of our system (S, M, A): Arbitrary(A), Status(S) and Masque lines(M); used in the comparison process, to indicate the status of lines and the computation of the basics parameters of the system

Line (A) of Fig. 5 represent a line where data input/output is transmitted, (S) represent the status line used in the equation (3), (4), (5), (6), (8) and (9) of the system. While (M) refers to a set of lines used as a masque in the process.

Before describing the BulkSort blocks, we should define the use of the data elements in our system.We highlight that data is read from memory and sent to the input of each PE_i at every clock cycle. As a distributed data, we found in each PE_i a binary sequence to be compared. These numbers refers to the ID of the PE_i. The information of Clock cycle (Y_i) is one of the basis distributed data; used in order to indicate when the element PE_i will be transmitted to the comparison process. (Z_i) indicate the concerned shared line AL. At each Clock cycle,PE_i has a specific rank in a view of the other PE_s R_i . During each clock cycle the values of the elements X_i, Y_i,Z_i and R_i are calculated within the processing and calculation unit PE_i by using competing functions which has been described before.

A line is set to the logic state '1' if there is at least one cell PE_i transmitting on this line a binary element = '1', otherwise this line is set to the logic state '0'. Thus, the value of the element and the state of the lines AL ('0' or '1') will make it possible to identify the status of cell (losing cell or winning

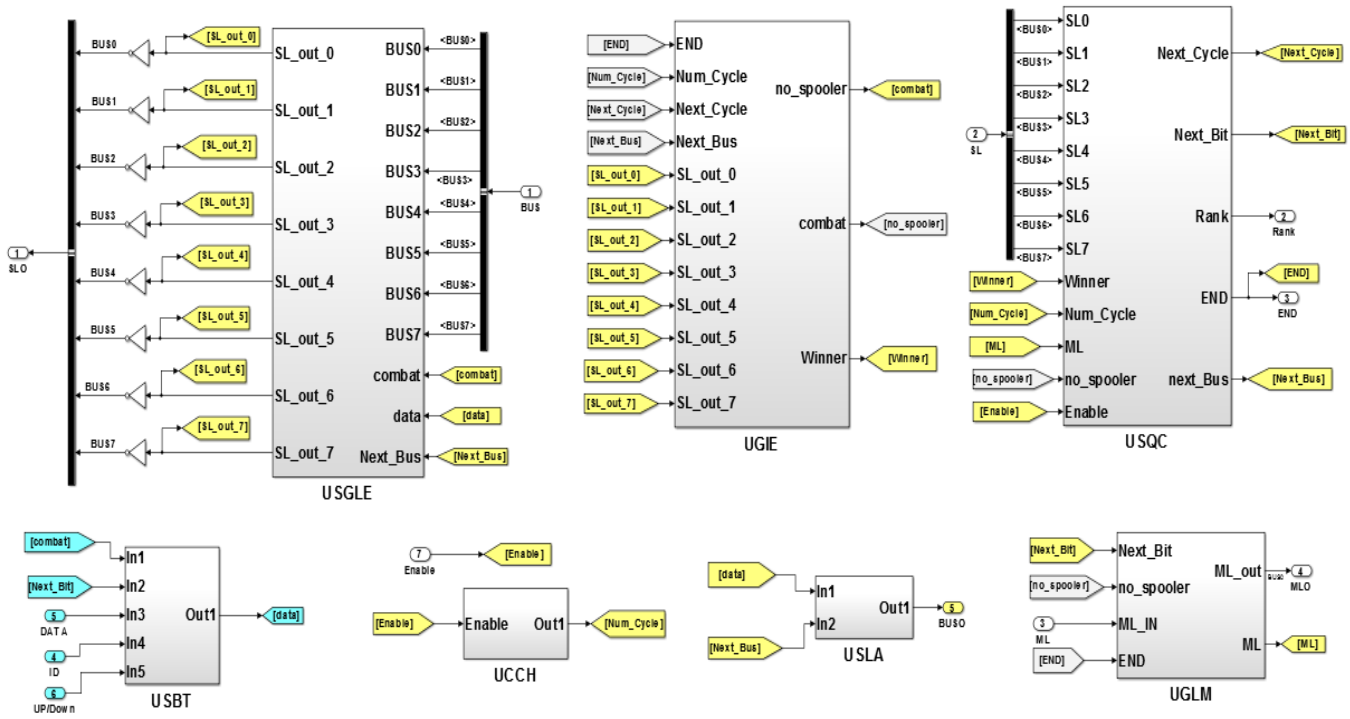

Fig. 6. PE'Sub-Blocks.

cell).

Instead of an iterative treatment, the steps performed by the processors are determined by the clock cycle, which includes reading, interpreting and executing the processes. Indeed, in the hardware implementation, the concept of pending does not exist. The queue refers to shifting in a view of clock cycle.

VI. Description of the BulkSort Sub-Blocks

The BulkSort model is built using Matlab/Simulink in order to be used in the Hardware implementation on FPGA. For the sake of simplicity, the proposed Simulink design will be limited to 7 cells. These cells are interconnected via a communication system. Within each PE_i, there is a system of cooperating units to perform the sorting; each of which has specific tasks (Fig. 6 and 7).

1) USBT (Bit Selection Unit): It generates the respective bit for each PE_i during the concerned clock cycle. The bit generation takes into consideration the result of the combat, either pending or processing state.If the PE_i is in processing state, the USBT unit generates the concerned bit according to the Next Bit set point, otherwise no generation is performed.

2) USQC (Calculation, Selection and counting Unit): Is the pivotal unit of the system, at which the parameters X_i, Y_i, $Z_i$and R_i; are generated. USQC is the unit that communicates with all other units.

Within the USQC unit there are 6 sub units, each of which is dedicated to make a main task in the system (see Fig. 8):

- Sigma SL: The crucial sub-unit for the PE parameters computation. Depends on the state of a PE if it is a loser or a winner. The unit receives the signal (the information) from the BLOCKS Sum_i and next bus, and chooses the sum concerned according to the next bus.
- Bloc Cycle Number:A sub-unit which generates the Next cycleY_i and the current Cycle based on a mathematical equation (Equation 5,6,10 and 11)
- Bloc Process Bus Number:A sub-unit generating the Next Bus and the current Bus according to a mathematical equation (Equation 3,4 and 9)
- Bloc Process Rank Number: Allows to define the rank of each PE at each iteration (Equation 1, 2 and 8)
- Process Bit Number:sub-unit which generates the next bit and current bit as well as the indicator END (Equation 7 and 12)

3) USGLE (Selection & Status Line management Unit): Each PE is connected to the set of state lines. The generation of the result provided by the USGLE unit is based on the following indicators: Next Bit, Data, Bus and Combat, in order to feed the Status Lines (S).

4) UGIE (State Indicator Management Unit): The main reason of this unit is to generate the state indicators of a PE. The different state that a PE can take are: WINNER, Combat and loser. To define the state, each UGIE unit is based on the following indicators to generate the State: num cycle, Next Cycle, End and Masque Lines(ML).

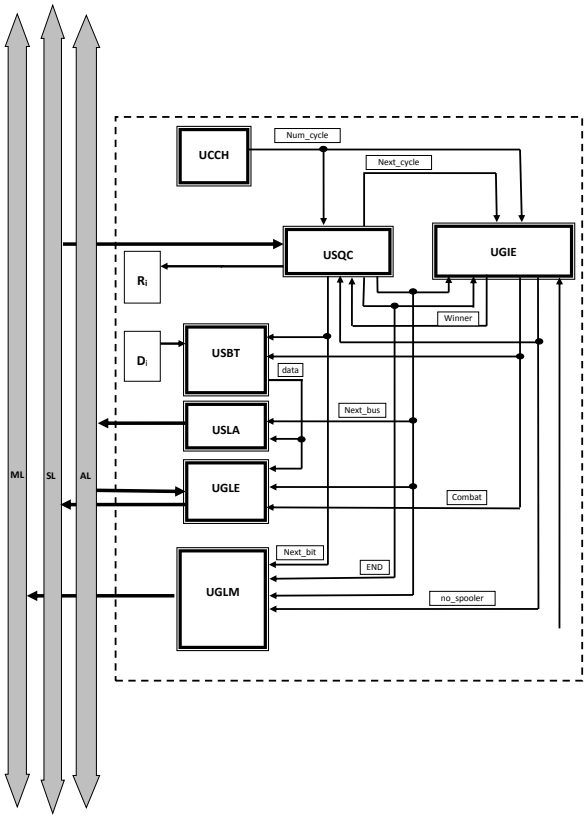

Fig. 7. Interconnection blocks.

5) USLA (Arbitration line selection unit): Responsible for sending the data to the concerned arbitration. It takes into consideration the data: next bit and data to define the concerned arbitration line .

6) UCCH (clock cycle Competing unit): Unit that counts the cycles of sorting processes, and allows us to define the number of iteration at each execution of the system.

7) UGLM (Masque lines Management Unit): It indicates whether there is combat on the bus or not. To carry out this operation, we use Next bit, Next bus and the masks lines.

The interconnection between the different Sub-unit of a PE is shown in Fig. 7. Each sub-unit generate and use as an input several parameters in order to provide at least the corresponding rank of the concerned PE.

To switch from the MATLAB / SIMULINK model to the ISE design model, we proceed as follow: The Simulink model was first checked for compatibility with HDL code generation. We then generated the HDL code in order to synthesize and analyze the timing of our design via the integration with ISE. We finally make an estimation of the resources using ISE.

For the sack of compatibility, before we generate a design report, we specify the characteristics of our FPGA(FPGA Spartan 6 XC6SLX150 (184304 Slice register)) after validating all stages of the HDL advisor. Then an ISE report; providing the required hardware resources; will be generated. The generated report gives as details the device utilization and timing summery of the BulkSort Model.

VII. Results and Discussion

A. Matlab / Simulink Simulation

In this section, we will provide the hardware simulation of our algorithm to demonstrate the effectiveness of the theoretical results in this brief. The C++ Simulation was presented to validate the proposed concept, while the Simulink modeling is in order to have an idea about the required resources.

There has been a question to find an optimal sorting architecture in view of size and depth. The implementation of the majority of sorting algorithm is limited because of the insufficient number of resources.

A number of research efforts are interested on sorting with minimal number of comparators. When we talk about comparators, we ought to think first about the basic devices of an FPGA such as: Slice Registers Look up Tables (LUTs) and others. In this paper, the results will be discussed in view of these parameters.

To simulate, synthesize, and implement HDL code generated from the model, Xilinx ISE Design Suite Version 14.5 is used in this work.The ISE provides an environment to go from design to an implementation of the proposed model by a specification of the design needs.

A various number of models has been synthesized in view of number of Process elements (PE). We should mention that the PEs refer to the set of elements we want to sort. We want to elaborate the variation of slice registers compared to the number of PE for different bus values. The bus refers to the communication link between process elements. Fig. 9 represent the required resources of the BulkSort model in case of 8 and 16 Buses, for 8, 16, 32 and 64 PEs. We found that the percentage do not exceed 21 percent of the available slice LUTs of the used FPGA for the different cases, and remains constant for the slice registers.

It is observed that the variation of LUTs is linear with respect to the number of data to be sorted. We note that using half of Buses we end up with a number of LUTs which is equal to half the number of the beginning. The amount we found is the same when we sort half of the data. The use of LUTs can go up to 1 per cent of the available resources on the platform. These results prove that our model is optimal in view of resources.

For the second parameter which is the Slices registers, we note that a variation of the number of buses for a given data size do not have a very great effect on the slices registers. On the other hand, when acting on the amount of data we are left with the half-slice registers as shown on all tests we performed. The utilization of slice registers did not exceed 1 per cent of the available amount on the platform FPGA Spartan 6.

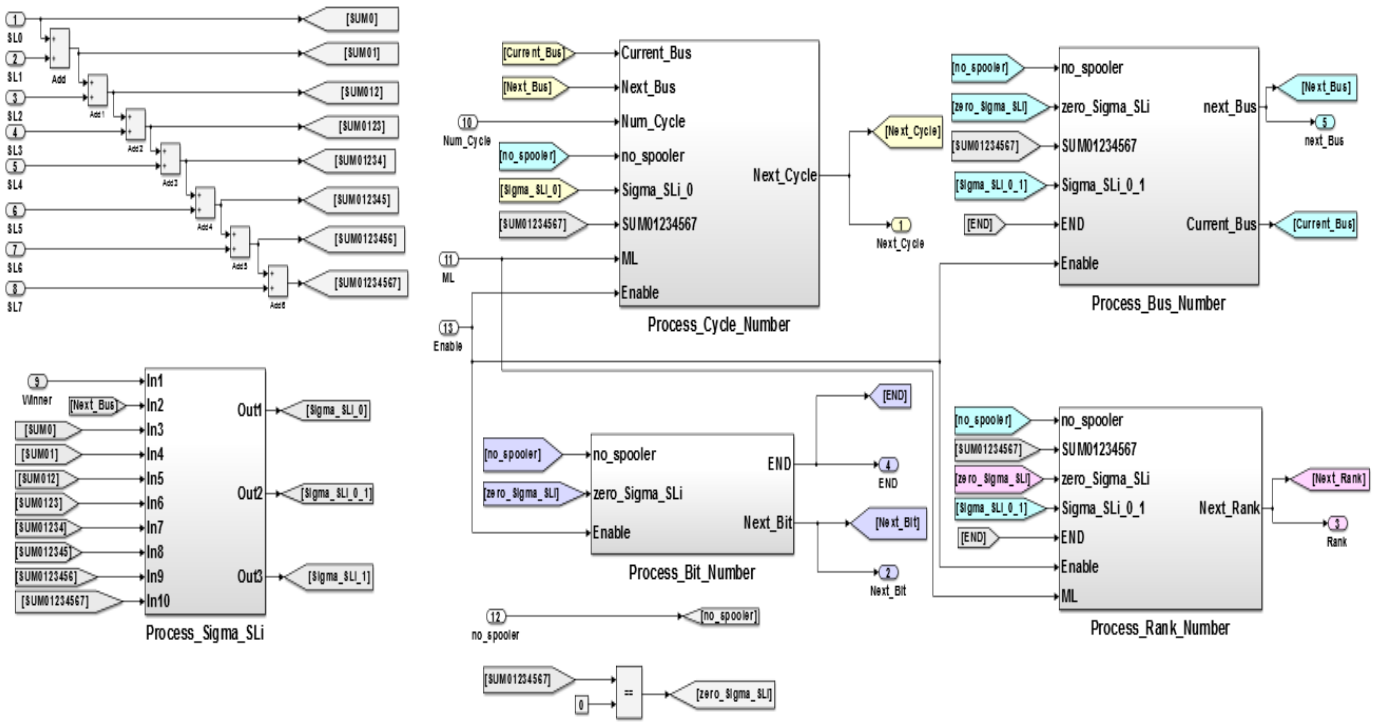

Fig. 8. The USQC sub_blocks: $Process_cycle_Number$, $Process_Bus_Number$, $Process_Rank_number$, $Process_Bit_Number$ and $Process_sigma_SLi$

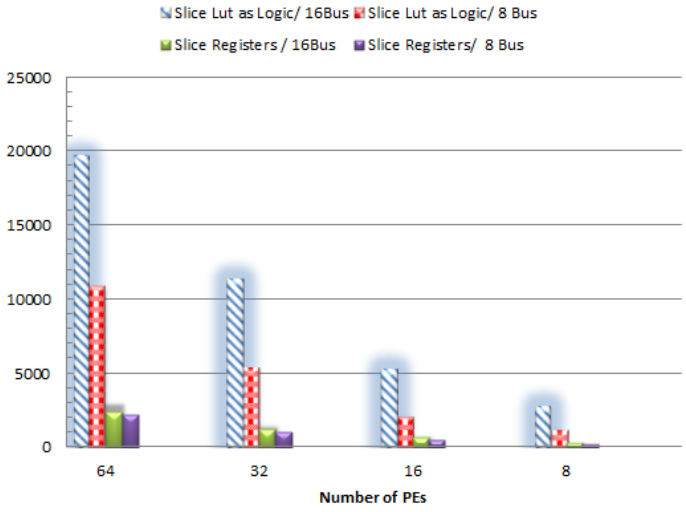

Fig. 9. Required resources of the BulkSort Model in term of slice registers, slice Luck up Tables and clock cycles

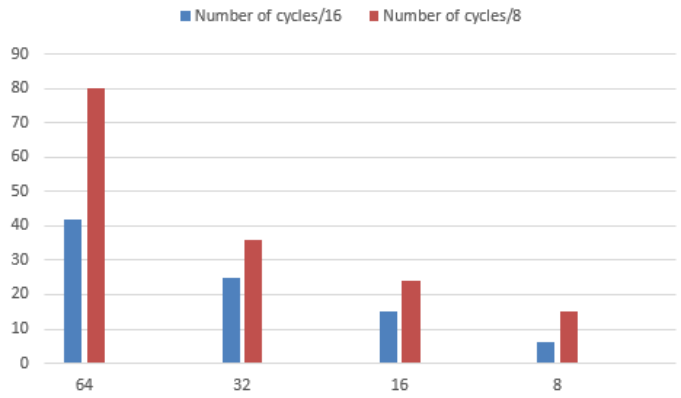

Fig. 10. Number of clock cycles in case of 8 and 16 Buses, for 8, 16, 32 and 64 PEs

In view of clock cycles (see Fig. 10), we found that even if we use half of the Buses to sort the same number of elements, a small increase in clock cycles occurs. So, we can use this point to say that by the use of this model we can sort a big data on a specific clock cycles. We can act also on the number of buses for a model optimization.

The numerical simulation makes it possible to calculate on the computer the solutions of models and to simulate the physical reality in order to have an idea about the hardware implementation. While designing the model of the parallel hardware BulkSort, we focus especially on making an efficient and optimal system. Our central concern is in dividing the required work up into pieces to be processed by several blocks. We proceeded by a partial parallelism (instead of sorting the elements one by one, a single passage allows us to cross the set of elements at each clock stroke).

We should highlight that this model is designed to be implemented in FPGA SPARTAN 6 XC6SLX150 (184304 Slice register). Our system provide both the sorting and ranking list as well as the number of clock cycles of the process.

VIII. Conclusion

In this article, we present a new divide-to-rule-like algorithm, called "BulkSort". We study and exhibit its parallel hardware implementation feasibility. Yet, we implemented it on Matlab-Simulink, and synthesized it using ISE suite design tool. We highlight that our scheme is parallel and reconfigurable according to the number of buses and number of processor elements. We also show that the proposed Simulink design was checked for compatibility with the hardware. Several tests have been conducted to show the applicability and illustrate the performance of our algorithm. Next, we evaluate the BulkSort behaviour in terms of slice registers, slice LUTs and Clock cycles, while varying the number of processor elements. Our proposal exhibits nice performance both in terms of resource utilization as well as sorting time. In view of perspectives, a high-performance, parallel architecture for an FPGA-based accelerator implementing the Bulk-Sort algorithm will be presented. The IP will be modeled using Vivado HLS and an end-to-end system (ZynQ ZC706 board) will be developed in order to assess the performance and resource usage.

References

[1] M. Aumüller and M. Dietzfelbinger, "Optimal partitioning for dual-pivot quicksort," *ACM Transactions on Algorithms (TALG)*, vol. 12, no. 2, p. 18, 2016.

[2] S. Wild, "Quicksort is optimal for many equal keys," in *2018 Proceedings of the Fifteenth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*. SIAM, 2018, pp. 8–22.

[3] D. P. Singh, I. Joshi, and J. Choudhary, "Survey of gpu based sorting algorithms," *International Journal of Parallel Programming*, vol. 46, no. 6, pp. 1017–1034, 2018.

[4] H. Peng, L. Huang, and J. Chen, "An efficient fpga implementation for odd-even sort based knn algorithm using opencl," in *2016 International SoC Design Conference (ISOCC)*. IEEE, 2016, pp. 207–208.

[5] A. Kazim, "A comparative study of well known sorting algorithms," *International Journal of Advanced Research in Computer Science*, vol. 8, no. 1, 2017.

[6] D. R. Musser, "Introspective sorting and selection algorithms," *Software: Practice and Experience*, vol. 27, no. 8, pp. 983–993, 1997.

[7] S. Mishra, S. Saha, S. Mondal, and C. A. C. Coello, "A divide-and-conquer based efficient non-dominated sorting approach," *Swarm and evolutionary computation*, vol. 44, pp. 748–773, 2019.

[8] R. M. Karp, "A survey of parallel algorithms for shared-memory machines," 1988.

[9] J. M. Liberti and M. A. Petersen, "Information: Hard and soft," *Review of Corporate Finance Studies*, vol. 8, no. 1, pp. 1–41, 2018.

[10] S. G. Akl, *Parallel computation: models and methods*. Prentice Hall Upper Saddle River, 1997, vol. 4.

[11] Y. Zhang, T. Cao, S. Li, X. Tian, L. Yuan, H. Jia, and A. V. Vasilakos, "Parallel processing systems for big data: a survey," *Proceedings of the IEEE*, vol. 104, no. 11, pp. 2114–2136, 2016.

[12] D. Koch and J. Torresen, "Fpgasort: A high performance sorting architecture exploiting run-time reconfiguration on fpgas for large problem sorting," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2011, pp. 45–54.

[13] R. Mueller, J. Teubner, and G. Alonso, "Sorting networks on fpgas," *The VLDB Journal—The International Journal on Very Large Data Bases*, vol. 21, no. 1, pp. 1–23, 2012.

[14] R. Perez-Andrade, R. Cumplido, C. Feregrino-Uribe, and F. M. Del Campo, "A versatile linear insertion sorter based on an fifo scheme," *Microelectronics Journal*, vol. 40, no. 12, pp. 1705–1713, 2009.

[15] H.-T. Hu, J.-R. Chang, and S.-J. Lin, "Synchronous blind audio watermarking via shape configuration of sorted lwt coefficient magnitudes," *Signal Processing*, vol. 147, pp. 190–202, 2018.

[16] M. H. Najafi, D. J. Lilja, M. D. Riedel, and K. Bazargan, "Low-cost sorting network circuits using unary processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 8, pp. 1471–1480, 2018.

[17] W. Song, D. Koch, M. Luján, and J. Garside, "Parallel hardware merge sorter," in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2016, pp. 95–102.

[18] S. Hauck, "The roles of fpgas in reprogrammable systems," *Proceedings of the IEEE*, vol. 86, no. 4, pp. 615–638, 1998.

[19] D. Andrews, D. Niehaus, R. Jidin, M. Finley, W. Peck, M. Frisbie, J. Ortiz, E. Komp, and P. Ashenden, "Programming models for hybrid fpga-cpu computational components: a missing link," *IEEE micro*, vol. 24, no. 4, pp. 42–53, 2004.

[20] N. Stekas and D. van den Heuvel, "Face recognition using local binary patterns histograms (lbph) on an fpga-based system on chip (soc)," in *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2016, pp. 300–304.

[21] S. Ihirri, A. Errami, and M. Khaldoun, "Bulk-sort: A novel adaptive and parallel sorting algorithm," in *Third International Congress on Information and Communication Technology*. Springer, 2019, pp. 725–736.