Indian Journal of Science and Technology Vol. 5 No. 9 (Sep. 2012) ISSN: 0974- 6846

Generalized approach to SOCD sorting on centralized diamond architecture

Masumeh Damrudi[1]* and Kamal Jadidy Aval[2]

[1,2]*Department of Computer Science, Firoozkooh Branch, Islamic Azad University, Firoozkooh, Iran*
m.damrudi@gmail.com*[1], k.jadidy@gmail.com[2]

Abstract

Quality of applications that includes robustness, real time response, and accurate performance has become a vital property for current applications. Among other existing solutions for such problems, parallel computing is a trending solution. Sorting is one of the main parts in almost every algorithm and various parallel sorting techniques have employed parallel architectures to do a qualified sorting. Gaining the best based on various different factors where speedup is the premier, is a topic of discussion. In this paper, we have issued the generalization of SOCD sort on the novel Centralized Diamond architecture which benefits from Single Instruction Multiple Data (SIMD) architecture with a time complexity of *O (logn)* on PRAM EREW(Parallel Random Access Machine Exclusive Read Exclusive Write). The results of conducted simulations of the algorithm prove the results of theoretical analysis of the algorithm. The findings of this research can be exploited in developing faster embedded systems. Using an appropriate interconnection network for achieving reasonable speedup in the execution of applications is important especially in embedded systems.

Keywords: Parallel sorting, Diamond architecture, SIMD, Generalized SOCD

Introduction

One of the most exciting research areas in computer science is parallel processing that has gained a lot of interest in the past decade. Nowadays, parallel processes are used for solving problems such as sort and search, which in result leads to a reasonable speed. Various problems of computer science era are benefiting from sorting algorithms. Database systems are the main applications among many other applications that are using sorting as a fundamental part. Some sorting algorithms are briefly described in (Damrudi *et al.*, 2008). Prasanta K. Jana proposed Multi-Mesh of Trees (MMT) architecture, which is a combination of multi mesh and mesh of trees and uses n^4 processors. One of the algorithms that is issued on this architecture is Esort which has a time complexity of *O(logn)* (Prasanta, 2004). A Multi-Sort algorithm, proposed by Rakesh (2009) on MMT, has the same time complexity with an internal improvement. Arefin *et al.* (2005) have issued an implementation of Batcher's Bitonic and Odd-Even merge sorts. Hayashi *et al.*, (1998) provided an EREW-PRAM and both the CREW-PRAM and the CRCW merge sorts are issued with a total cost of *O(log n)* and *O(log log n +log k)* respectively. They introduced *k*-merge algorithm which runs at *O (log n)* time with *O(n log k)* total cost on the EREW-PRAM.

A parallel computer is a collection of some processor elements that cooperate together to resolve massive problems in a faster manner. In this paper, we have generalized the new born SOCD sorting algorithm on Centralized Diamond architecture. The centralized diamond inherits its main specifications from Diamond architecture, which was issued in (Damrudi *et al.*, 2009). Diamond architecture has *(7/4) n* processor elements.

The architecture is heterogeneous and uses more number of cheaper simple PEs (Processor Elements) compared to the number of expensive complex ones. This specific design leads to a tradeoff between the number of each type of processor element and the related complexity and cost. This architecture has been employed by NOD and ENOD to sort input data elements (Damrudi *et al.*, 2009; Damrudi *et al.*, 2010; Jadidy Aval *et al.*, 2010). In comparison to the original Diamond architecture, Centralized Diamond has one more node in the center. This improved architecture has new techniques to interconnect processor elements. In the original SOCD, it is assumed that each processor element has one data element. In the real world, data elements are increasing rapidly and this makes it impossible to use the solution for general applications with unknown input data elements.

To achieve a general solution, this paper issues the generalized approach of SOCD in which the numbers of input data elements are not limited. Centralized Diamond is an SIMD architecture, which executes the same instructions on many data elements. Zhou and Ross (2002) reported that SIMD instructions can hasten many database operations by removing branch overhead, that makes SIMD based architectures more useful. Mainly, SIMD computers are divided into four subclasses EREW, CREW, ERCW and CRCW.

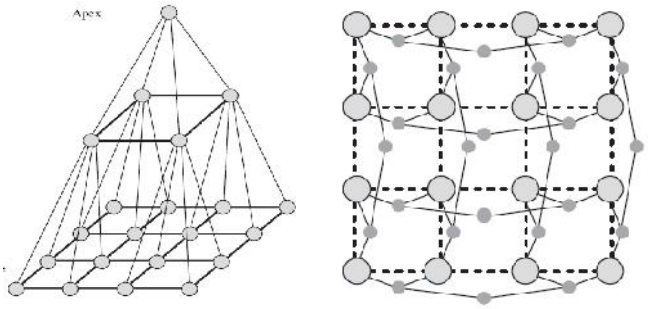
The rest of this paper is organized as follows: In the next section the Centralized Diamond architecture and its features are introduced. In the following section, generalized SOCD sort on centralized diamond architecture is presented. Afterwards, complexity and cost of the algorithm is analyzed. Finally, the conclusion is provided.

Indian Journal of Science and Technology Vol. 5 No. 9 (Sep. 2012) ISSN: 0974- 6846

Centralized diamond architecture

The first version of Centralized Diamond architecture was presented by Damrudi *et al.* (2009). It exploits exclusive techniques on interconnection relations among processor elements. The topology of centralized diamond is hybrid like some other existing architecture. This means that the topology is a combination of the other basic topologies like the ones that are illustrated in Fig.1

Fig. 1. Hybrid interconnection network topologies



a) Pyramid b) Mesh of Trees

While the mesh of trees and the pyramid are a combination of meshes and trees, the centralized diamond is a combination of trees and ring which means it inherits both the specifications of tree and ring. Processor elements' number starts from zero in the first level and the level numbers start from zero. The first level connects tree parts of architecture together using the centralized node. Let *n* be the number of data elements and *N* be the number of processor elements. Each centralized diamond architecture has four levels and each level has following features:

Level zero (L_0): There is one node in this level. There are 16 registers employed in this PE. We have called it centralized node and it has no number assigned to it. Sorting more data elements, processor elements must be added to the levels and more registers must be considered for central processor element.

Level one (L_1): Regardless of the central node, this level has *n/4* nodes and each PE is benefiting from eight registers. Each PE of this level has direct connectivity to the centralized node. The connections for this level are formulated as following:

$$\left.\begin{array}{l} x < n/4 \\ \prod_{n/4}(x) = 2x + n/4 \\ \prod_{n/4+1}(x) = 2x + n/4 + 1 \end{array}\right\} \textit{(Intralevel con.)} \quad (1)$$

According to the binary tree model, each PE in this level has two children. These children construct the second level eq.(1).

Level two (L_2): Ignoring the central node in PE count, this level has *n/2* PEs and each PE has four registers. There is a direct connection between every two neighboring PEs in this level that is formalized as following:

$$\left.\begin{array}{l} n/4 \le x < 3n/4 \\ \prod_{+1}(x) = x + 1 \quad where\ x \bmod 2 \neq 0 \\ \prod_{+n/4}(x) = 2x + n/4 \\ \prod_{+n/4+1}(x) = 2x + n/4 + 1 \end{array}\right\} \textit{(Intralevel con.)} \quad (2)$$

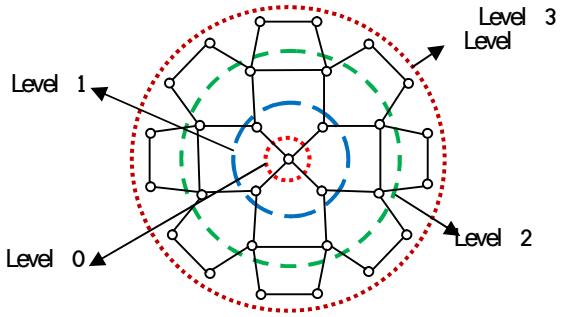
The processor elements of this level have two children. These children organize the third level eq.(2).

Level three (L_3): Without taking the central node into account, this level has *n* nodes. There are two registers in every node and each processor element is directly connected to the processor element next to it as formalized in the following:

$$\left.\begin{array}{l} x >= 3n/4 \\ \prod_{+1}(x) = x + 1 \quad where\ x \bmod 2 \neq 0 \end{array}\right\} \textit{(Intralevel con.)} \quad (3)$$

As it is explained by Damrudi *et al.,* (2011), the relation between data elements and PEs of this architecture is $N = 7/4\,n + 1$. Fig.2 presents Centralized Diamond architecture with *n=16* for data elements and

Fig. 2. Levels of centralized diamond (Damrudi et al., 2011)



N=29 for processor elements.

Four levels compose this architecture. The number of data elements in each PE in the third level is not limited in the generalized approach and it is based on the requirement of the application. Applying this solution, the number of PEs is as following:

$$N = 14k + 1 \quad \forall k \ge 2 \quad (4)$$

The number of processor elements for third, second, and first level is *8k*, *4k*, and *2k* respectively where *k≥2*. These values plus centralized PE compose the total number of PEs.

Generalization of SOCD Sort on Centralized Diamond architecture

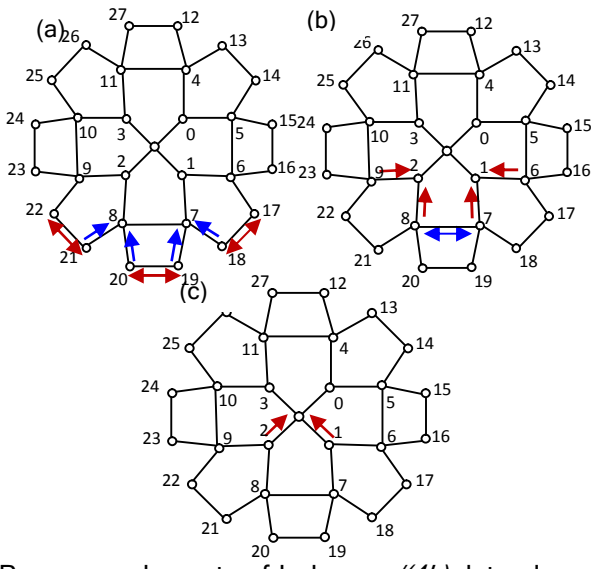
It is important to make sure that a parallel solution is not suffering from the problems such as comprehension complexity, great number of processor elements and impossibility of data element increment. Let *n* be the number of data elements and *N* be the number of processor elements. As mentioned above, it is assumed that the original SOCD (Sorting On Centralized Diamond) have one data element for each PE. In the real world applications, data elements are increasing rapidly

Indian Journal of Science and Technology Vol. 5 No. 9 (Sep. 2012) ISSN: 0974- 6846

and it is impossible to have one PE for each data elements. Therefore, the numbers of data elements are limited to the number of processor elements and original SOCD doesn't comply with the nature of applications.

To generalize the SOCD, in the initialization phase, *n* data elements are spread over PEs in the third level. Each PE gets *n/(8k)* data elements. Considering the example of architecture as Fig.2, each PE has *n/16* data elements to process. The PE must have adequate registers as memory to store data elements. At the first step, each PE such as *x* (which *x* is odd) in L_3, sends its data elements to the next PE in the same level indexed *x+1*all at the same time. PEs store received data elements into their empty registers, sort them and send the lower half of data elements to the PEs with the index of *x*. All PEs in L_3, send their data elements to their parents in L_2 which are presented with blue arrows in Fig. 3(a). Since the data elements in the generalized SOCD are not limited, this operation needs an algorithm to merge these sorted lists in L_3 and locate them in L_2. As mentioned in the introduction, many merge sort algorithms on *k* sorted lists have been issued. One of them is (Hayashi *et al.*, 1998), which needs *O(log n)* to sort *m* sorted list. In this part, we have *two* sorted lists that are merged employing the algorithm of (Hayashi *et al.*, 1998), to have one sorted list in each PE of L_2. Fig.3 (a) illustrates the order of these operations.

Fig. 3. Operations of generalized SOCD algorithm



Processor elements of L_2 have *n/(4k)* data elements. Considering the example of architecture illustrated in Fig.2, each PE has *n/8* data elements. In the next step, each PE such as *x* (which *x* is odd) sends its data elements to the next PE in the same level (L_2) indexed *x+1* all at the same time. Upon receiving data, each processor element stores the data in its registers and sorts all of its data elements and then sends lower half of data elements to the processor element which is indexed as *x*. Afterwards; processor elements of the L_2 send their data elements to their parents. Just like the processor elements of L_3, these processor elements are exploiting the merge sort algorithm of (Hayashi *et al.*, 1998) to have one sorted lists in every PE of L_1. Fig.3 (b) shows these operations using blue and red arrows. In this stage, the processor elements of the first level have *n/(2k)* data elements. Employing the algorithm of (Hayashi *et al.*, 1998), the sorted lists are now in L_0 which is illustrated in Fig.3(c).The algorithm of the above steps is provided in Table 1.

Complexity Analysis and Simulation Results

We have conducted a simulation for the proposed solution and showed the simulation on MATLAB SIMULINK R2008a, for 15 processor elements and 16 data elements (Table 1). The results of the simulation have proved the correctness of the above algorithm. In the generalized SOCD sorting algorithm, as it is obvious in the table, steps one, four, and seven have the complexity of *O(1).* The complexity of steps two and three is *O(log (n/(4k)))*, steps five and six is *O(log (n/(2k)))* and the last step is *O(log (n/k))*. Putting these orders together leads to the overall time complexity which is as follows:

$$t_n = O(1) + O(1) + O(1) + O(\log n/4k) + O(\log n/2k) + O(\log n/k) \cong O(\log n/k) \quad (5)$$

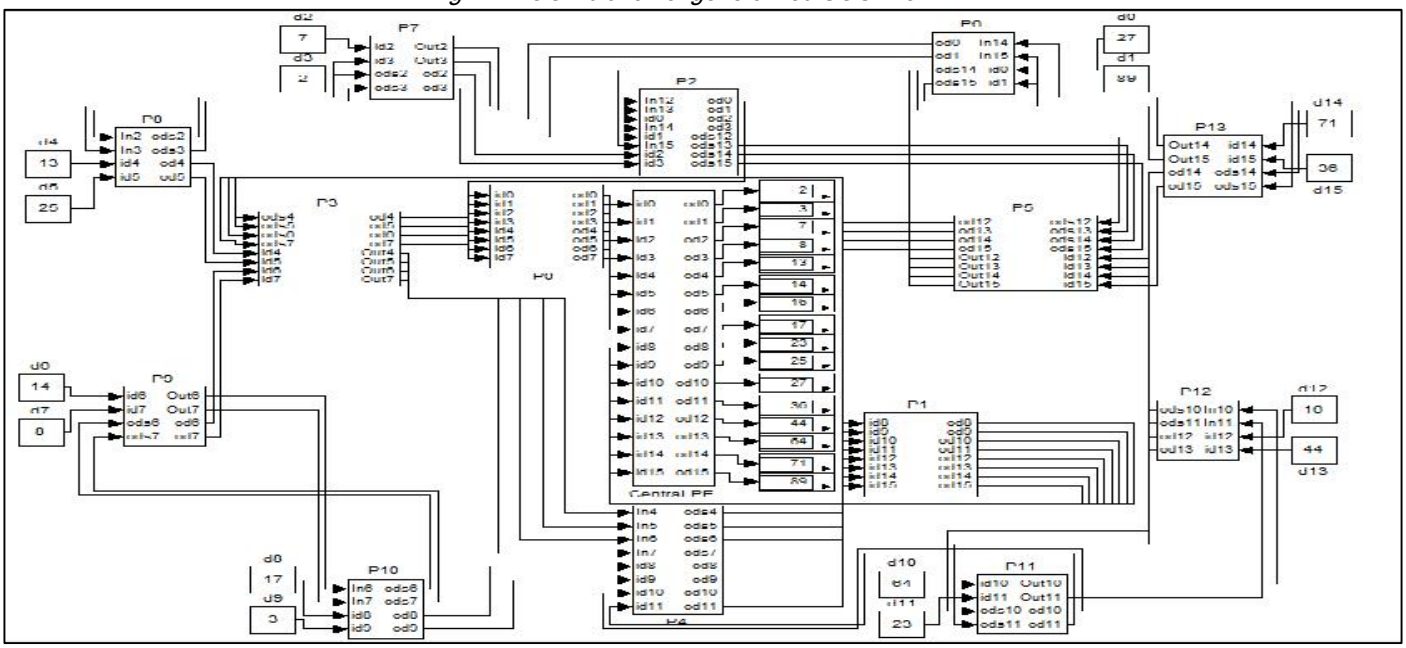
The SIMULINK of MATLAB has provided an environment which leads to simpler and faster simulation. In this simulation we have defined PEs, their connections, and assigned the required operations to evaluate the proposed algorithm and architecture. The simulation is performed exploiting the logic blocks. The sorted list in the central PE is shown using display block in the middle of the Fig.4.

Conclusion

The generalized SOCD sorting algorithm is not limited in terms of number of data elements regarding to original SOCD as well as the complexity time of this algorithm is the same as the original one. In the world of daily increasing data, it is impossible to have one PE for each data element which is time and area consuming and complex. Generalized SOCD employs less PEs which results in increased efficiency. Nowadays, there are many sorting algorithms such as Esort and Multi-Sort that have *O(log n)* time complexity. While their underlying architecture, Multi-Mesh of Trees, has n^4 processor elements, taking the number of PEs into account, their total cost is much more than the generalized SOCD. It should be considered that the architecture is heterogeneous, besides, there is no delay needed due to the fact that all PEs in each level are homogeneous and just differ from previous and next level.

Indian Journal of Science and Technology Vol. 5 No. 9 (Sep 2012) ISSN: 0974- 6846

Fig. 4. The simulation of generalized SOCD for k=2



References

1. Arefin Ahmed, Shamsul Mohammed and Abul Hasan (2005) An improvement of bitonic sorting for parallel computing. ICCOMP'05 Proc. of the 9th WSEAS Int. Conf. on Computers. Wisconsin, USA.
2. Damrudi M and Kamal Jadidy Aval (2009) Diamond architecture with NOD sorting. IEEE Youth Conf. Info. Comput. & Telecommun., Beijing, China. pp: 431-434.
3. Damrudi M and Kamal Jadidy Aval (2010) A new parallel sorting on Diamond architecture. Proc. 4th Conf. Europ. Comput. Conf., Univ. Politechnica of Bucharest, Bucharest, Romania. pp: 284-287.
4. Damrudi M and Kamal Jadidy Aval (2011) Sorting data elements by SOCD using centralized diamond architecture. *Comput. Technol. & Appl.* 2(5), 374-377.
5. Damrudi M, Mohammad R Salehnamadi and Kamal Jadidy Aval (2008) ROE sorting on ILLIAC array processor. *Intl. Appl. Comput. Conf.* (ACC'08), Istanbul, Turkey. pp: 326-330.
6. Hayashi Tatsuya, Koji Nakano and Stephan Olariu (1998) Work-time optimal k-merge algorithms on the PRAM. *IEEE Trans. Parallel & Distributed Sys.* 9(3). 275-282.
7. Jadidy Aval K., Masumeh Damrudi, 2010. ENOD sort on Diamond architecture. *9th WSEAS Intl. Conf. Software Engg, Parallel & Distributed Sys.,* (SEPADS '10), University of Cambridge, UK, pp: 205-208.
8. Prasanta K Jana (2004) Multi-mesh of trees with its parallel algorithms. *J. Sys. Archi.* 50(4), 193-206.
9. Rakesh Nitin (2009) Nitin multi-sort algorithm on multi-mesh of trees MMT. *Masaum J. Comput.* 1(1), 1-8.
10. Zhou J and Ross KA (2002) Implementing database operations using SIMD instructions. *Proc. ACM SIGMOD Int. Conf. Manage. Data*.