

# An FPGA-Based Acceleration Platform for Auction Algorithm

Pengfei Zhu\*, Chun Zhang\*, Hua Li<sup>†</sup>, Ray C.C. Cheung<sup>‡</sup> and Bryan Hu\*

\*Department of Electrical and Computer Engineering,  
University of Alberta, Canada

<sup>†</sup>Department of Mathematics and Computer Science,  
University of Lethbridge, Canada

<sup>‡</sup>Department of Electronic Engineering, City University of Hong Kong, China

**Abstract**—Auction algorithms have been applied in various linear network problems, such as assignment, transportation, max-flow and shortest path problem. The inherent parallel characteristics of these algorithms are well suited for FPGA hardware implementation. In this paper, we focus on the acceleration of auction algorithm to solve assignment problem. The main contribution is to set up a flexible platform to generate efficient and extendable application-based hardware acceleration. It aims at solving both symmetric and asymmetric assignment problem. Experimental results show that 10X speedup can be achieved using 128 Processing Elements for the problem size of 500.

## I. INTRODUCTION

The assignment problem (AP) is one of the fundamental combinatorial optimization problems, which tries to find the maximum weight matching in a weighted bipartite graph. Naturally, it is useful to solve such problems as assigning employees to tasks or machines to production jobs. Furthermore, it is also a key building block in several applications of network modeling in real world. For example, to allocate and route the traffic in a large city or coordinating a large fleet of service people to respond to requests. Thus it is essential to develop an efficient implementation of assignment problem to deal with these ambitious tasks.

Due to its importance, the assignment problem has been studied extensively in the literature [1][2]. Among various approaches, auction [3] and Hungarian [1] algorithm are two efficient methods to solve the problem optimally. Compared to Hungarian algorithm, the auction algorithm exhibits a natural parallel computing characteristic, which is well suited for hardware acceleration. Although they have various forms, different auction-based algorithms share similar working flow as [4][3]. Therefore, although we focus on [3] in this paper, it can be extended to other auction algorithms with little efforts.

In this paper, we propose to build a common platform for accelerating auction algorithm using FPGAs. The assignment problem is used as a concrete example to illustrate the effectiveness of the proposed method. Two implementation architectures, Jacobi and Gauss-Seidel are discussed, and the latter one is proved to be more efficient for hardware acceleration. In addition, a CAD flow is implemented to automatically map the problem of different sizes into hardware. To the best of our knowledge, this is the first paper to set up auction-based architecture based on FPGAs.

In existing work of FPGA implementation to solve assignment problem [5][6], Hung and Wang proposed a computing system based on recurrent neural network. It takes a few seconds to compute  $100 \times 100$  assignment problem, which is not efficient enough to meet the practical needs. In contrast, our proposed architecture can deal with a  $500 \times 500$  problem in milliseconds. Other work related with hardware implementations of auction algorithm can be referred to [7]. Recently, Vasconcelos and Rosenhahn [7] used GPU to accelerate auction algorithm and achieved 7X with problems from 400 to 4000. Our results show that we can achieve more than 10X when problem size is 500.

The organization of this paper is as follows. In Section II, we introduce the preliminaries. In Section III, two different kinds of hardware implementation architectures are discussed and compared. In Section IV, we discuss the implementation details of the proposed architecture. The experimental results are shown in Section V. Section VI concludes the paper and proposes the future work.

## II. PRELIMINARIES

### A. The Assignment Problem

In graph theoretic terms, the assignment problem aims at finding a maximum weight matching of a weighted bipartite graph. Given  $m$  agents and  $n$  objects ( $m \leq n$ ), and a benefit  $a_{ij}$  associated with the assignment of object  $j$  to agent  $i$ , the assignment problem is to find an assignment of each agent to exactly one object that maximizes the total benefit. Furthermore, objects are allowed to remain. It can be described as a linear programming problem as:

$$\text{Maximize } \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} a_{ij} \cdot x_{ij} \quad (1)$$

$$\text{Subject to } \sum_{i=0}^{m-1} x_{ij} \leq 1, \forall j = 0, 1, \dots, n-1 \quad (2)$$

$$\sum_{j=0}^{n-1} x_{ij} = 1, \forall i = 0, 1, \dots, m-1 \quad (3)$$

$$x_{ij} \in \{0, 1\}, \forall i = 0, 1, \dots, m-1, j = 0, 1, \dots, n-1 \quad (4)$$

where  $x_{ij} = 1$  if and only if object  $j$  is assigned to agent  $i$ .

## B. Auction Algorithm

Auction algorithm is one of the classical methods to optimally solve the assignment problem [3][4]. Intuitively, it borrows the idea of market equilibrium in economy. Consider the assignment problem as an auction where agents bid for objects. Suppose that agent  $i$  has potential benefit  $a_{ij}$  in bidding for object  $j$  at the price of  $p_j$ , and then the net benefit will be  $a_{ij} - p_j$ . To its best interest, each agent will always want to get the object with largest net benefit. However, due to the competition from other agents, the assignment progresses in an auction manner.

Several variations of the auction algorithm for solving assignment problem exist in the literature [3][4][8]. In this paper, we focus on the forward auction algorithm [3], which is widely known to have good performance in CPU implementation. However, the same discussions and implementation framework can be adapted to other variations as well.

TABLE I  
PSEUDO-CODE OF FORWARD AUCTION ALGORITHM

|         |  |
|---------|--|
| Input:  | agents $m$ , objects $n$ ; benefit matrix $A$  |
| Output: | assignment $\Phi$  |
| 1       | BEGIN  |
| 2       | INIT $p_j=0$ , $\Phi_j=-1$ for all objects $j$ ;<br>put all unassigned agents in a list;<br>set the number of assigned agents, $nAssi=0$ |
| 3       | WHILE $nAssi \neq m$   |
|         | /* Bidding Phase */  |
| 4       | Get an unassigned agent $i$ from the list  |
| 5       | FOR all the objects $j$  |
| 6       | Compute the net benefit, $a_{ij} - p_j$  |
| 7       | GET $V_{ij_i}$ , $W_{ij_i}$ and $j_i$  |
|         | /* Assignment Phase */   |
| 8       | COMPUTE the bid of agent $i$ for object $j_i$  |
| 9       | IF $\Phi_{j_i} \neq -1$  |
| 10      | ADD $\Phi_{j_i}$ to the list   |
| 11      | ELSE   |
| 12      | $nAssi++$  |
| 13      | $\Phi_{j_i}=i$ ;   |
| 14      | $p_{j_i}=bid$  |
| 15      | RETURN $\Phi$  |
| 16      | END  |

Basically, the forward auction algorithm is composed of two phases in each iteration: the bidding and assignment phase.

**Bidding Phase:** each agent  $i \in I$ , where  $I$  is the set of unassigned agents, finds an object  $j_i$  with the best net benefit, and gives its reasonable bid, that is,

$$b_{ij_i} = p_{j_i} + V_{ij_i} - W_{ij_i} + \epsilon \quad (5)$$

$$\text{where } V_{ij_i} = \max_{\forall j} \{a_{ij} - p_j\}$$

$$W_{ij_i} = \max_{\forall j, j \neq j_i} \{a_{ij} - p_j\}$$

$$\epsilon : \text{a constant}$$

**Assignment Phase:** each object  $j$  that is selected as best bid by the agents in  $I$ , determines the highest bidder  $i_j$ , raises its price to  $b_{ij_j}$ , and gets assigned to the highest bidder  $i_j$ ; The agent that is assigned to  $j$  at the beginning of iteration (if any) get unassigned.

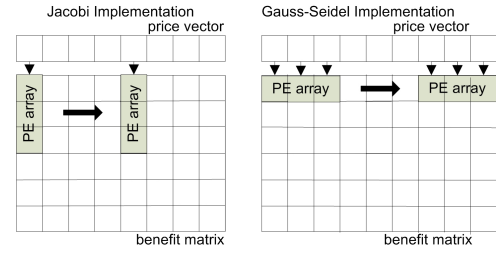


Fig. 1. Jacobi implementation (left) and Gauss-Seidel implementation (right) of auction algorithm

## III. FPGA-BASED ARCHITECTURE EXPLORATION

The key point of using FPGA for acceleration lies in extracting the parallelism in a target problem. Considering the forward auction algorithm, there are two potential hardware architectures, namely, Jacobi and Gauss-Seidel, to expose the parallel computing flow. In this section, we make a comparison of these two architectures and argue the latter one is better for acceleration.

### A. Jacobi Parallelization Architecture

Jacobi parallelization regards each agent as an individual processing element and allows them to bid for objects simultaneously. The problem can be understood in an easier way, shown in the left part of Figure 1. For the  $m$ -to- $n$  assignment, a benefit matrix  $A$  that each row represents an agent and each column represents an object is introduced. The element of the benefit matrix  $a_{ij}$  represents the benefit that agent  $i$  bids for object  $j$ . The prices of all the objects are stored in a separate vector. At the beginning of each iteration, the processing elements array (1-D array of PEs) computes the net benefits for the first column. The corresponding price is provided for all the PEs. Then it goes through all the columns to compute the bids of the unassigned rows.

### B. Gauss-Seidel Parallelization Architecture

Opposite to Jacobi method, the Gauss-Seidel method regards each object as an individual PE, shown in the right part of Figure 1. Each PE is responsible for computing the net value of one object. At first, the PE array computes the net benefit for the first several columns at the same time. After finishing the computation of the first block, it moves forward a block to calculate the next set of net values. By several steps of computation, it will go through all the columns, to find the best object of the computed agent. Different from Jacobi method, it computes one bid per iteration.

### C. Jacobi method vs. Gauss-Seidel method

The major reason for the difference is due to the fact that during the auction process, the number of assigned agents increases to the problem size quickly. The Jacobi architecture, whose processing granularity is agent, leads to a large number of idle PEs as there are few unassigned agents left. Different from the Jacobi architecture, the Gauss-Seidel architecture handles one agent at a time, and the bids for different objects

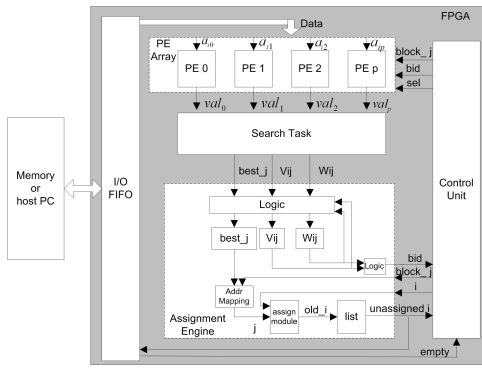


Fig. 2. Gauss-Seidel Architecture with  $p$  PEs for parallel computation.

are computed in parallel. As each agent will always need to compare all objects, the hardware usage does not drop. Experiments show that Gauss-Seidel architecture achieves a 6X speedup on average compared to Jacobi architecture.

## IV. PLATFORM IMPLEMENTATION

In this section, we discuss in detail the implementation of Gauss-Seidel architecture. We develop a flexible hardware platform which can automatically generate the architecture of user-specified size to solve assignment problems of arbitrary size efficiently. It ensures that no PE will be idle during bidding phase throughout the whole computation progress. We leverage the Block RAMs (BRAMs) in modern FPGAs to keep the instances data so as to save the energy dealing with the interface to the external memory or host machine. Figure 2 illustrates the detailed architecture implemented in FPGA.

### A. Gauss-Seidel Architecture

The architecture is composed of five modules, I/O FIFOs, PE Array, Search Task, Assignment Engine, and Control Unit. Among them, PE Array and Search Task module are two data-intensive computing units. The basic units to compose these two modules have regularity in structure, which brings the benefit of highly extendable architecture.

1) *PE Array*: PE array is composed of a number of identical PEs in one dimension. Each PE aims at computing a net benefit in the bidding phase (corresponding to line 6 in Table I). Inside a PE, one distributed RAM is used as a cache to store a group of prices in a specific order when the problem size is larger than the number of PEs, which often is the case. Since the number of prices grows linearly with problem size, it is costless to keep the prices in a FPGA. All the PEs are identical in data-processing behavior so that they share the same control signal.

To update a specified price, we use a decoder to select the target PE and provide the address of the local RAM at the same time, so that we can access any price in the PE array.

2) *Search Task*: After an array of net benefits are calculated, search task computes the best net value, second best net value and the best value's location in the PE array in this block. The search task module consists of many regular

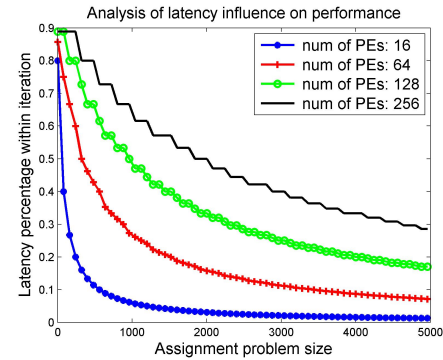


Fig. 3. The percentage of cycles consumed by the latency within iteration.

two-input comparators, bringing the benefit of extensibility. And it is pipelined to prevent clock frequency from dropping dramatically when architecture size increases.

3) *Assignment Engine*: The goal of Assignment Engine is to derive all the required values to compute the bid and to complete the functions in assignment phase (corresponding to line 7-14 in Table I). We map those lines into three steps and each step takes one clock cycle. In the first step, the best net value  $V_{ij_i}$ , its location in the PE array  $j_i$ , and the second best bid  $W_{ij_i}$ , are computed (line 7). Based on the derived data, the bid  $b_{ij}$  can be computed at this time (line 8). Meanwhile, the assignment status of object  $j_i$  is checked (line 9). Finally, it completes the three functions in assignment phase (line 10, 13, 14 in Table I).

### B. Influence of search task on performance

There is no doubt that better performance can be achieved by a parallel architecture with more processing elements. The latency introduced by the search task has the negative impact on the overall performance. In each assignment phase the price of the bid object requires updating, it has to wait until search task finishes the latest computation.

Since full pipeline of the search task module introduces high latency, which heavily affects the performance, we reduce the pipeline by half. We make sure that search task module does not become the longest path in the whole design. The latency introduced in search task is 4, 6 and 8 cycles when the number of PEs is 16, 64 and 128 respectively. The influence on performance based on different architecture size is depicted in Figure 3. From the figure, we can see that given a certain architecture size, the percentage of latency decreases and more cycles will be spent on effective computation when the problem size increases. Moreover, for the problems of small size, it is wise to implement a relatively small architecture, to avoid the influence of high latency.

## V. EXPERIMENTAL RESULT

The proposed Gauss-Seidel architecture for dense assignment problem is implemented on Xilinx Virtex-5 LX110T FPGA with a speed grade of -1. A CAD flow is employed to generate the architecture of user-specified size in verilog

TABLE II  
RESOURCE UTILIZATION BASED THE ON ARCHITECTURE OF TWO  
SELECTED SIZE AND THEIR CLOCK FREQUENCY

| Resources    | Size |             |       |             |
|--------------|------|-------------|-------|-------------|
|              | 16   |             | 128   |             |
|              | Used | Utilization | Used  | Utilization |
| Registers    | 1647 | 2%          | 10834 | 15%         |
| LUT as logic | 2265 | 4%          | 13501 | 19%         |
| LUT as mem.  | 540  | 3%          | 4127  | 23%         |
| BRAMs        | 6    | 4%          | 34    | 23%         |
| Freq. (MHz)  | 220  |             | 139   |             |

HDL. We use ModelSim 6.2g for simulation, and Xilinx ISE 12.2 to synthesize and place and route our design.

The architectures of several sizes have been mapped into hardware. For a single Virtex-5 LX110T FPGA, we can implement more than 400 PEs for parallel computation. Table 2 shows the resource utilization for two selected architectures. From the table we can see that LUTs are heavily consumed when the architecture size is 128. This is because the search task module consumes a large amount of resources, nearly 95%. The timing report shows that the clock frequency decreases with the increasing architecture size. There is no surprise that the large usage of FPGA routing resource results in the difficulty in routing task, which increases the time consumed by routes. However, the clock frequency can be further improved by manual floor planning.

To test the performance of the proposed architecture, we have used the most common classes of instances used in the literature [9] to test AP algorithms, which is called uniform random class. This class includes that the entries of the benefit matrix are integers uniformly randomly generated in  $[0, K]$ , with  $K = 1000$ . As symmetric problem often needs more time to compute, we use it as the worst case to test the performance.

For comparison, we implement a C program based on the public FORTRAN codes originally written by the author of [4]. The program was run on a PC with a Pentium Dual Core CPU running at 2.7GHz with 4.00GB RAM. Symmetric assignment problems of size from 200 to 500 are tested due to the limitation of BRAM resources. We use 5 instances of uniform random class per problem size to test. All the assignment results are optimal verified by public codes of Hungarian algorithm. The speedup is computed as follows:

$$Speedup = \frac{Computation\ time\ by\ CPU}{Computation\ time\ by\ FPGA}$$

The computation time is core calculation time to execute the algorithm. The speedup of 16-PEs and 128-PEs architectures on four sets of assignment problems is shown in Figure 4. There is an increasing trend of speedup when the problem size grows, because the latency influence degrades and more cycles are spent on effective computation, so the proposed architecture can achieve a higher speedup ratio for large problems. Currently, the speedup is more than 10X when the problem size reaches 500.

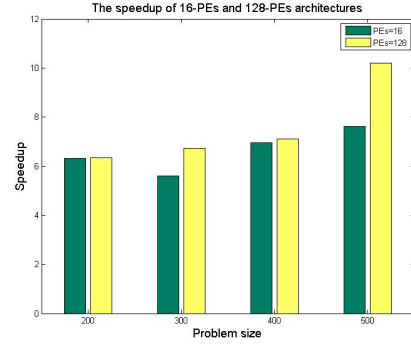


Fig. 4. The speedup is averaged based on 5 random generated instances. The architecture contains 16 PEs and 128 PEs for parallel computing respectively. The problem size covers from 200 to 500.

## VI. CONCLUSION

In this paper we explored two classes of auction-based parallel architecture for solving assignment problem and figured out that Gauss-Seidel method is more suitable for parallel implementation. Then we set up a general platform which generates extendable hardware architecture using CAD flow. It can efficiently deal with assignment problem of any size. The proposed architecture can achieve more than 10X speedup with 128 PEs on problems at size 500.

Currently, the scale problem is unknown due to the resource limitation of on-chip memory. We are considering high-speed interfaces to solve memory problem. In the future, we plan to extend our hardware architecture to other problems in linear network field as well.

## ACKNOWLEDGMENT

This research work has been supported by NSERC (Natural Sciences and Engineering Research Council of Canada).

## REFERENCES

- [1] H. Kuhn, "The Hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83-97, 1955.
- [2] J. Wang, "Analogue neural network for solving the assignment problem," *Electronics Letters*, vol. 28, no. 11, pp. 1047-1050, 1992.
- [3] D. Bertsekas, "The auction algorithm: A distributed relaxation method for the assignment problem," *Annals of Operations Research*, vol. 14, no. 1, pp. 105-123, 1988.
- [4] —, "Auction algorithms for network flow problems: A tutorial introduction," *Computational Optimization and Applications*, vol. 1, no. 1, pp. 7-66, 1992.
- [5] D. Hung and J. Wang, "Digital hardware realization of a recurrent neural network for solving the assignment problem," *Neurocomputing*, vol. 51, pp. 447-461, 2003.
- [6] —, "A FPGA-based custom computing system for solving the assignment problem," in *FPGAs for Custom Computing Machines, 1998. Proceedings. IEEE Symposium on*. IEEE, 1998, pp. 298-299.
- [7] C. Vasconcelos and B. Rosenhahn, "Bipartite graph matching computation on GPU," in *Energy Minimization Methods in Computer Vision and Pattern Recognition*. Springer, 2009, pp. 42-55.
- [8] D. Bertsekas and D. Castañón, "A forward/reverse auction algorithm for asymmetric assignment problems," *Computational Optimization and Applications*, vol. 1, no. 3, pp. 277-297, 1992.
- [9] M. Dell'Amico and P. Toth, "Algorithms and codes for dense assignment problems: the state of the art," *Discrete Applied Mathematics*, vol. 100, no. 1-2, pp. 17-48, 2000.