

A FPGA-Based Custom Computing System for Solving the Assignment Problem

Donald L. Hung

School of Electrical Engineering and Computer Science
Washington State University, Tri-Cities
Richland, WA 99352, USA

email: dhung@beta.tricity.wsu.edu, Phone: (509) 372-7234

Jun Wang

Department of Mechanical and Automation Engineering
The Chinese University of Hong Kong
Shatin, NT, Hong Kong

Abstract – The assignment problem is a classical combinatorial optimization problem arising in numerous design and planning context. Solving an assignment problem of large scale is computationally intensive and time consuming. This paper discusses the development of an FPGA-based custom computing system that can accelerate the computation by exploiting the intrinsic parallelism of a recently proposed recurrent neural network for solving the assignment problem. The theoretical background of this work has been discussed in other papers. The digital realization of the system, including architecture, design, FPGA implementation and verification are discussed in this paper.

1. Introduction

The assignment problem consists of finding, via minimizing a cost function, the optimal solution for assigning a given number of entities to an identical number of cells. The traditional methods [1,2] for solving the assignment problem are not efficient due to their sequential nature. To address this problem, Wang proposed an analog recurrent neural network (RNN) [3] but its realization is not feasible due to the massive interconnections required and the difficulties in programming the RNN's weights and parameters. For a digital realization of Wang's RNN, a discrete-time algorithm has been developed in [4]. Based on this algorithm, this paper describes the architecture, design, FPGA implementation and verification of the custom computing system for solving the assignment problem.

2. The Algorithm

Based on the algorithm developed in [4], solving the assignment problem of size n requires the following computation:

```

let  $u_i(1) = 0; x_i(1) = 0$  for  $i = 1, 2, \dots, n^2$ 
while  $k < \text{given number of iterations}$ 
begin
  while  $i \leq n^2$ 
  begin
     $u_i(k+1) = -k \left[ \sum_{j=1}^{n^2} w_{ij} x_j(k) - 2 \right] + u_i(k) + k_{2i} \cdot k - k_{3i}$  (1)
     $x_i(k+1) = g[u_i(k+1)]$  (2)
  end /*while (i)*/
end /*while (k)*/

```

In the above algorithm, x_i is a decision variable; u_i is a net input to the i^{th} neuron of Wang's RNN; k_1, k_{2i}, k_{3i} are parameters with predetermined values; $g[\cdot]$ is a ramp activation function with negative limiter and unit gain; w_{ij} is a weight which can have the

value of 0, 1 and 2. The solution of the assignment problem is obtained as the index k in (1) and (2) increases.

3. Architecture

The dominant computational task of the algorithm shown in (1) and (2) is to calculate the term $\sum_{j=1}^{n^2} w_{ij} x_j(k)$ for all decision variables. In order to accelerate the execution of this task, we used a 1D systolic array architecture as shown in Fig. 1.

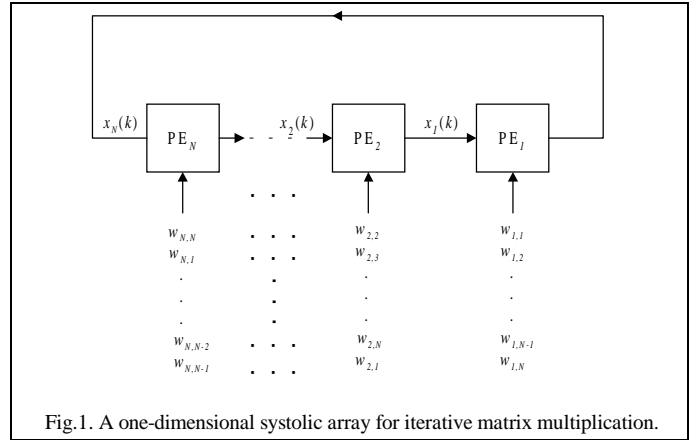


Fig. 1. A one-dimensional systolic array for iterative matrix multiplication.

In Fig. 1, each PE is dedicated to updating the value of one decision variable. The array size N is equivalent to n^2 . Note that the systolic array is connected in a ring topology and the data (the weights) used by an arbitrary PE p is stored with the data structure shown below:

$$d_{p,i} = w_{p, (p+i-2) \bmod N + 1} \text{ for } i = 1, 2, \dots, N. \quad (3)$$

Executions of all PEs in the array are completely identical, i.e., the pipeline is filled starting from the first cycle; furthermore, no extra clock cycles are needed between iterations. Once the decision variables circle through the ring, their updated values are obtained simultaneously at the corresponding PEs.

Based on the overall system framework shown in Fig. 1, an individual PE's internal structure can be determined as a direct mapping from (1) and (2). The resultant PE's dataflow graph is given in Fig.2, where the $h1$ and $h2$ blocks are correspondent to the terms $-k \left[\sum_{j=1}^{n^2} w_{ij} x_j(k) - 2 \right]$ and $[u_i(k) + k_{2i} \cdot k - k_{3i}]$ in (1) respectively; the block labeled g Function is

correspondent to (2); the memory block labeled *Weight Memory* stores the data described by (3).

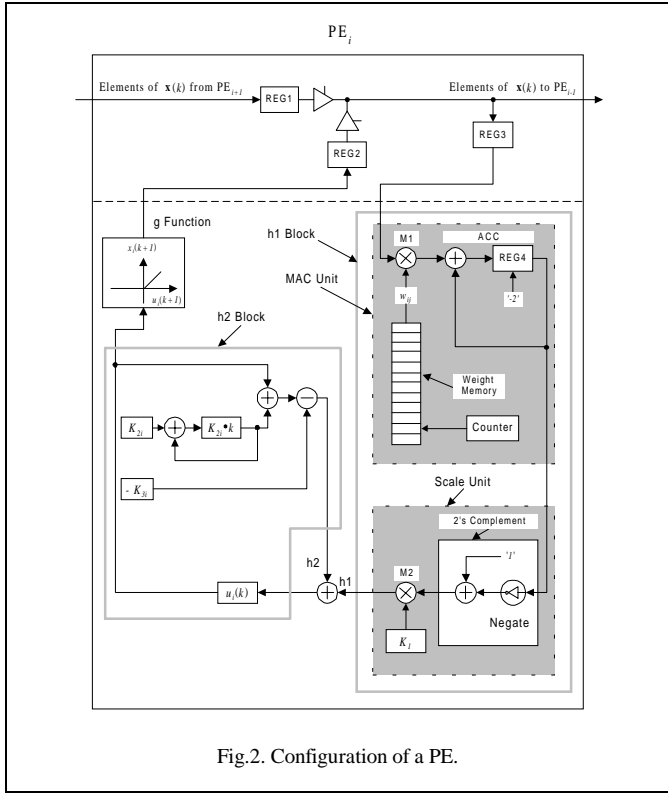


Fig.2. Configuration of a PE.

In order to make the array scalable, FIFOs must be used and the data structure shown in (3) needs to be modified accordingly.

4. Design and FPGA Implementation

Fig. 3 shows the RTL data path of the PE's lower part. The system is designed to operate on signed integers and negative numbers are represented in their 2's complements. Due to the nature of the

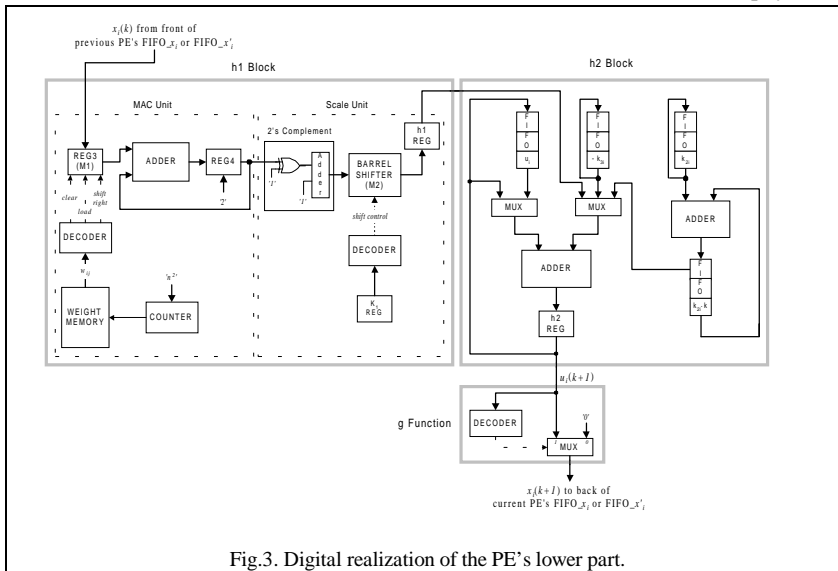


Fig.3. Digital realization of the PE's lower part.

weights, multiplications performed by M1 are reduced to the 'clear', 'load' and 'shift to the left by 1-bit' operations accordingly. Normally, the *h1* block's *Scale* unit requires a multiplier (or divisor) M2, but as extensive simulation shows that the parameter k_1 can be restricted to powers of two, in the design the function of M2 was realized by a barrel shifter. The two blocks *h1* and *h1* execute concurrently under the control of two hardwired controllers. In this design, for an assignment problem of size n , if the array size is N , then the total computing time required to find the solution \mathbf{x}^* is:

$$T_{total} = T_c(2n^2 + 3)qk, \quad (4)$$

where T_c is the clock period, k is the number of total iterations required, and $q = n^2/N$. As a comparison, to execute the same algorithm, a single processor will need at least the amount of time as shown below:

$$T_{\mu p} = T_c(mn^4)k, \quad (5)$$

where m is the number of clock cycles a specific processor needed for a MAC operation.

For verification purpose, we implemented a 16-bit version of the system using Xilinx FPGAs. Each PE was placed into a 4020E device, then five PEs were connected in a ring to form the array. For convenience, memories were implemented using on-chip resources. The FIFOs were register based with depth of five. Without special efforts in optimizing the physical layout in the FPGAs, the PEs were able to run at speeds beyond 20 MHz. With this 5-PE prototype, we have been able to verify the system by running small-scaled assignment problems, including the example given in Wang's paper [3].

5. Conclusions

Although the current prototype is a scaled-down version, the system can be extended to full size without sacrificing its performance. The system's performance can be doubled by applying a 2-stage pipeline to its MAC unit. Performance can be also improved by increasing the clock rate, via optimizing the physical layout in the FPGAs.

References

- [1] M.S. Bazaraa, J. J. Jarvis, and H.D. Sherali, *Linear Programming and Network Flows* (2nd Ed.), John Wiley & Sons, New York, NY, 1990.
- [2] D.G. Luenberger, *Linear and Nonlinear Programming* (2rd Ed.), Addison-Wesley, Reading, MA, 1984.
- [3] J.Wang, "An Analog Neural Network for Solving the Assignment Problem," *Electronic Letters*, vol. 28, no. 11, pp. 1047-1050, 1992.
- [4] D.L. Hung, J. Wang and Z. Zhou, "Digital realization of a recurrent neural network for solving the assignment problem," submitted to *IEEE Transactions on Neural Networks*, under review.