Design and Implementation of an Acyclic Stable Matching Scheduler

Enyue Lu[†]
[†]Dept. of Computer Science
University of Texas at Dallas
Richardson, TX 75083-0688, USA
enyue@utdallas.edu

Mei Yang[*]
[*]Dept. of Computer Science
Columbus State University
Columbus, GA 31907, USA
yang_mei@colstate.edu

Yi Zhang[‡] and S. Q. Zheng[†]
[‡]Dept. of Electrical Engineering
University of Texas at Dallas
Richardson, TX 75083-0688, USA
{yxz023000, sizheng}@utdallas.edu

***Abstract*— Applications of stable matching in switch scheduling have been proposed. However, the classical GS stable matching algorithm is infeasible for high-speed implementation due to its high complexity. Instead, acyclic stable matching algorithms have been shown useful in implementing scheduling for high-speed switches/routers. In this paper, we model the acyclic stable matching problem as the dominating set problem for a rooted dependency graph, and propose a parallel algorithm for finding the dominating set in $O(n \log n)$ time. We design and implement a scheduler based on the proposed algorithm in hardware. Simulation results show that the number of 2-input NAND gates and the timing of our design are proportional to n^2 and n respectively, making it feasible to be implemented at high speed with current CMOS technologies.**

***Index Terms*— Stable matching, acyclic graph, dependency graph, dominating set, switch scheduling.**

I. Introduction

The stable marriage problem (or stable matching problem) was first introduced by Gale and Shapley (GS) in 1962 [1]. Given n men, n women, and $2n$ ranking lists in which each person ranks all members of the opposite sex in the order of preference, a *matching* is a set of n pairs of man and woman with each man/woman in exactly one pair. A matching is *stable* if there does not exist one man and one woman who are not matched to each other, but each of whom strictly prefers the other to his/her current partner in the matching; otherwise, the matching is *unstable*. Gale and Shapley showed that every instance of the stable matching problem admits at least one stable matching, which can be computed in $O(n^2)$ iterations. The paper [1] sparked much interest in many aspects and variants of the classical stable matching problem [2].

The solutions to the stable matching problem have been applied to switch scheduling for packet switches. Many GS based stable matching scheduling algorithms have been proposed for both input queued (IQ) switches and combined input and output queued (CIOQ) switches [3]-[10]. In these algorithms, the man set and the woman set consist of all input ports and all output ports respectively, and the ranking list for each input/output is defined differently according to different performance requirements. For example, McKeown proposed two scheduling algorithms, GS longest queue first (GS-LQF) and GS oldest cell first (GS-OCF), with ranking lists based on the occupancy of the input queues and the waiting time of the cells at the head of input queues respectively in [4]. GS-LQF and GS-OCF algorithms were shown to achieve asymptotically 100% throughput under both uniform and non-uniform traffic for IQ switches.

The scheduling algorithms based on general stable matchings, however, are too complex for high-speed implementation. It turns out that for stable matching instances with acyclic dependency graphs, finding stable matchings takes less time. Researchers have proposed several scheduling algorithms for CIOQ switches based on acyclic stable matchings. In [5], Prabhakar and McKeown proposed the most urgent cell first algorithm (MUCFA) for a CIOQ switch with a speedup of 4 to emulate an output queued (OQ) switch performance. Chuang and Stoica improved the result to a speedup of 2 by the critical cell first (CCF) algorithm [6] and the joined preferred matching (JPM) algorithm [7] independently. In [8], Nong *et al*. proved that with some speedup, an acyclic stable matching scheduling algorithm can provide QoS guarantees for both unicast and multicast traffic with fixed-length and variable-length packets.

The advantage of acyclic stable matching scheduling algorithms is its feasibility for high-speed implementation. However, there is no hardware design and implementation of acyclic stable matching scheduling algorithms in the literature. In this paper, we propose a parallel algorithm for the acyclic stable matching problem, and present its hardware implementation. We first model the acyclic stable matching problem as the dominating set problem for rooted dependency graphs. We show that the root set and the dominating set of a rooted dependency graph are identical. We then propose a parallel algorithm, FIND_ROOTS, to find the root set of a rooted dependency graph in $O(n \log n)$ time with n^2 simple processing elements (PEs). We further present hardware design and implementation of the proposed algorithm. Simulation results show that the number of 2-input NAND gates and the timing of our design are proportional to n^2 and n respectively. The proposed design can be used to implement schedulers based on acyclic stable matching algorithms, such as those in [5]-[8].

The rest of the paper is organized as follows. In Section II, we propose our parallel algorithm FIND_ROOTS. In Section III, we focus on the design and implementation of FIND_ROOTS in hardware. Section IV concludes the paper.

II. A Parallel Stable Matching Algorithm for Rooted Dependency Graph

A. Preliminaries

Let $M = \{m_1, m_2, \cdots, m_n\}$ and $W = \{w_1, w_2, \cdots, w_n\}$ be the sets of n men and n women respectively. Let $mR_i = \{wr_{i,1}, wr_{i,2}, \cdots, wr_{i,n}\}$ and $wR_j = \{mr_{j,1}, mr_{j,2}, \cdots, mr_{j,n}\}$ be the *ranking lists* for man m_i and woman w_j respectively, where $wr_{i,j}$ (resp. $mr_{j,i}$) is the rank of woman w_j (resp. man m_i), $1 \leq i, j \leq n$. That is, if $wr_{i,j} = k$ (resp. $mr_{j,i} = k$), then woman w_j (resp. man m_i) is the kth choice of man m_i (resp. woman w_j).

Let A be a *ranking matrix* of size $n \times n$, where each entry of $a_{i,j}$ of A is a pair of $(wr_{i,j}, mr_{j,i})$ in which $wr_{i,j}$ is the rank of woman w_j in man ranking list mR_i and $mr_{j,i}$ is the rank of man m_i in woman ranking list wR_j. We call $wr_{i,j}$ (resp. $mr_{j,i}$) the *horizontal value* (resp. *vertical value*) of $a_{i,j}$, and denote it by $a^h_{i,j}$ (resp. $a^v_{i,j}$). Example 1 shows a 4×4 ranking matrix obtained from two given ranking lists.

Example 1: An instance of stable matching problem.

Man ranking lists:	Woman ranking lists:
$mR_1 : \{3, 4, 1, 2\}$;	$wR_1 : \{3, 2, 1, 4\}$;
$mR_2 : \{1, 2, 3, 4\}$;	$wR_2 : \{1, 4, 3, 2\}$;
$mR_3 : \{1, 2, 4, 3\}$;	$wR_3 : \{1, 2, 3, 4\}$;
$mR_4 : \{2, 3, 1, 4\}$.	$wR_4 : \{3, 2, 1, 4\}$.

Ranking matrix A:

3,3	4,1	1,1	2,3
1,2	2,4	3,2	4,2
1,1	2,3	4,3	3,1
2,4	3,2	1,4	4,4

Definition 1: Given an $n \times n$ ranking matrix A, a set of man-woman pairs is a *matching* $\mathcal{M}$ if any two pairs (m_{i_1}, w_{j_1}) and (m_{i_2}, w_{j_2}) in $\mathcal{M}$ are corresponding to two entries a_{i_1,j_1} and a_{i_2,j_2} in different rows/columns of A; $\mathcal{M}$ is a *stable matching* if there does not exist a pair $(m_i, w_j) \notin \mathcal{M}$ such that $a^h_{i,j} < a^h_{i,k}$ and $a^v_{i,j} < a^v_{l,j}$, where $(m_i, w_k), (m_l, w_j) \in \mathcal{M}$.

In Example 1, by Definition 1, we know the stable matching is the set of pairs $(1, 3)$, $(2, 4)$, $(3, 1)$ and $(4, 2)$, whose corresponding entries in the ranking matrix are marked by underlines.

B. Dominating Set for Dependency Graph

Given a ranking matrix A, we define the *dependency graph* of A as a directed graph $\vec{G}$ constructed as follows: every $a_{i,j}$ of A is represented by a vertex $v_{i,j}$ of $\vec{G}$; there is an edge from $v_{i,j}$ to $v_{i,k}$ if and only if $a^h_{i,j} < a^h_{i,k}$; there is an edge from $v_{i,j}$ to $v_{l,j}$ if and only if $a^v_{i,j} < a^v_{l,j}$. A stable matching instance is *acyclic* if its corresponding dependency graph does not contain any cycle. A *dominating set* of dependency graph $\vec{G}$ is a set of vertices, denoted by V_d, such that the following two conditions are satisfied: (1) for any two vertices in V_d, they are corresponding to two entries in different rows/columns of the ranking matrix; (2) for any vertex $v \in V(\vec{G}) - V_d$, there is a directed edge from a vertex in V_d to v.

Since each vertex $v_{i,j}$ in $\vec{G}$ is corresponding to a pair of man and woman (m_i, w_j), by the definitions of stable matching and dominating set, we have the following fact.

Fact 1: Let $\vec{G}$ be a dependency graph. V_d is the vertex subset corresponding to a stable matching if and only if V_d is a dominating set of $\vec{G}$.

By Fact 1, the problem of finding a stable matching is reduced to the problem of finding a dominating set. In general, the dominating set for a dependency graph may not be unique, and finding one is time consuming. However, we find that the problem of finding dominating sets for a special class of dependency graphs, named *rooted dependency graphs*, is much easier. A rooted dependency graph is defined recursively as follows: an empty graph is a rooted dependency graph; a non-empty dependency graph $\vec{G}$ is a rooted dependency graph if (1) it contains one or more roots, each being a vertex without any incoming edge; (2) the *reduced subgraph*, which is obtained from $\vec{G}$ by removing all vertices in the same rows/columns as the roots and all outgoing edges from these removed vertices, is also a rooted dependency graph. The *root set* of a rooted dependency graph $\vec{G}$ is a set that consists of all roots of $\vec{G}$ and its reduced subgraphs recursively generated from $\vec{G}$.

Fact 2: Let $\vec{G}$ be the dependency graph of a ranking matrix A where each entry $a_{i,j} = (wr_{i,j}, mr_{j,i})$. For any vertex $v_{i,j}$, the number of incoming edges coming from the vertices in row i is equal to $wr_{i,j} - 1$ and the number of incoming edges coming from the vertices in column j is equal to $mr_{j,i} - 1$.

By Fact 2, we know that a vertex with corresponding entry $(1, 1)$ is a root since it has no incoming edge. By Facts 1 and 2, we have the following theorem.

Theorem 1: For a rooted dependency graph $\vec{G}$, the root set is the same as the dominating set, which is unique for $\vec{G}$.

Example 2: Figure 1 (a) shows the dependency graph $\vec{G}$ for the ranking matrix in Example 1. The horizontal value and vertical value of each entry in the ranking matrix are shown in each corresponding vertex. From the figure, clearly, neither of two vertices $v_{1,3}$ and $v_{3,1}$, which are marked as dark circles in $\vec{G}$, has incoming edge since each of them corresponds to an entry $(1, 1)$ in the ranking matrix. Hence, $\vec{G}$ has two roots, $v_{1,3}$ and $v_{3,1}$. After removing all vertices in rows 1, 3 and columns 1, 3 and their outgoing edges in $\vec{G}$, we get the reduced subgraph $\vec{G'}$, which has root $v_{4,2}$ marked as dark circle in $\vec{G'}$. After removing all vertices in row 4 and column 2 and their outgoing edges in $\vec{G'}$, we get the reduced subgraph $\vec{G''}$, which contains only one vertex $v_{2,4}$ that is also a root of $\vec{G''}$. By the definition, we know $\vec{G}$ is a rooted dependency graph. It is easy to verify that the root set, $\{v_{1,3}, v_{3,1}, v_{4,2}, v_{2,4}\}$, is the dominating set of $\vec{G}$. By Theorem 1, the dominating set corresponds to the stable matching of Example 1, which is $\{(1, 3), (3, 1), (4, 2), (2, 4)\}$.

A rooted dependency graph may not be acyclic (i.e. the graph may have a directed cycle). In Example 2, $\vec{G}$ contains a cycle ($v_{1,1}$,$v_{4,1}$,$v_{4,2}$,$v_{3,2}$,$v_{2,2}$,$v_{2,3}$,$v_{2,4}$,$v_{1,4}$) (see Figure 1 (a), in which edges in the cycle are marked as dark edges). However, an acyclic graph always has at least one root, and its reduced subgraph is also acyclic. Thus, we have the following fact.

Fact 3: An acyclic dependency graph is a rooted dependency graph, but a rooted dependency graph may not be an acyclic dependency graph.

In the following, we propose a parallel algorithm for finding

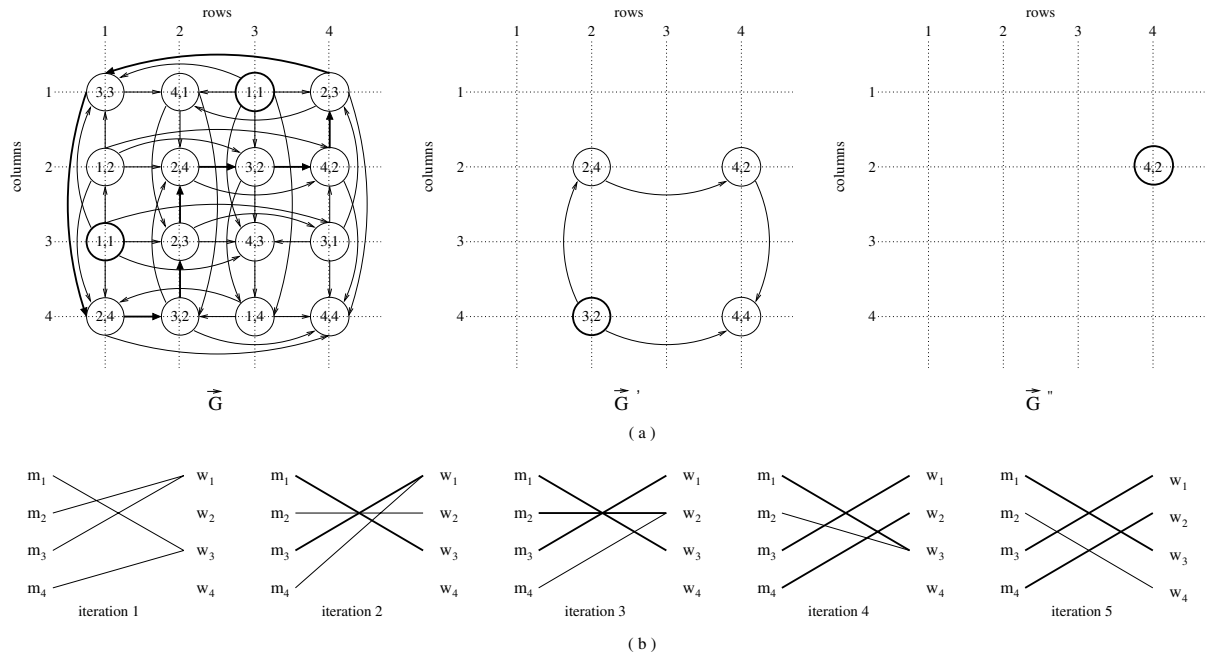

Fig. 1. (a) A rooted dependency graph $\vec{G}$ and its reduced subgraph $\vec{G'}$ and $\vec{G''}$. The roots of $\vec{G}$ are $v_{1,3}$, $v_{3,1}$. The roots of $\vec{G'}$ and $\vec{G''}$ are $v_{4,2}$ and $v_{2,4}$ respectively. The root set is $\{v_{1,3},v_{3,1},v_{4,2},v_{2,4}\}$, and each root is marked as a dark circle. (b) The stable matching is found by GS algorithm in 5 iterations. In each iteration, new proposals are marked as light lines and the kept proposals are marked as dark lines.

the root set (i.e. the stable matching) in a rooted dependency graph.

C. The Algorithm

Given a rooted dependency graph $\vec{G}$ constructed from an $n\times n$ ranking matrix A, we first find the roots of $\vec{G}$. If the reduced subgraph $\vec{G'}$ of $\vec{G}$ is not empty, we continue to find remaining vertices in the root set of $\vec{G}$ recursively until the total number of found roots equals to n. The algorithm for finding the root set of a rooted dependency graph, FIND_ROOTS, is described in the following.

Algorithm FIND_ROOTS
begin
$G := \vec{G}$ /* $\vec{G}$ is the dependency graph */
$V_r := \emptyset$ /* V_r is the root set */
while there exists a root in G **do**
Step 1: find the set of roots V_r' of G and let $V_r := V_r \cup V_r'$
Step 2: find the reduced subgraph G' of G and let $G := G'$
end

Based on Theorem 1 and Fact 1, the set of roots obtained from FIND_ROOT is corresponding to the set of man-woman pairs in the stable matching. We analyze the time complexity of FIND_ROOTS using n^2 PEs as follows. The n^2 PEs are placed as an $n\times n$ array, and the n PEs in the same row/column are fully connected.

Each $\text{PE}_{i,j}$ is corresponding to a vertex $v_{i,j}$ of $\vec{G}$ and has a pair of horizontal (h for short) and vertical (v for short) values set as $(wr_{i,j}, mr_{j,i})$ initially. Since the total number of roots in root set of $\vec{G}$ is equal to n, FIND_ROOTS runs in at most n iterations. Each iteration of FIND_ROOTS consists of two steps. Based on Fact 2, we know step 1 can be done in $O(1)$ time by each $\text{PE}_{i,j}$ checking if its $(h, v) = (1, 1)$. Conceptually, step 2 contains 2 substeps. In substep 1, each root vertex $v_{i,j}$ found in step 1 sets its $(h, v) = (0, 0)$ and marks all vertices in row i and column j as the vertices to be deleted. Since all PEs in the same row/column are fully connected, this substep takes $O(1)$ time. In substep 2, each undeleted vertex $v_{i,j}$ decreases its h (resp. v) value by k if its h (resp. v) value is greater than that of k deleted vertices in row i (resp. column j). Since there are at most n deleted vertices in each row/column, this substep can be done in $O(\log n)$ time. Therefore, based on the above discussion, we have the following theorem.

Theorem 2: Given any instance of stable matching problem, if its corresponding dependency graph is a rooted dependency graph (including acyclic dependency graph), we can find the stable matching in $O(n \log n)$ time on n^2 PEs.

D. Comparison with GS Algorithm

Gale and Shapley proposed an algorithm for solving the stable matching problem in [1]. The GS algorithm works in the following way. Each man first proposes to his most favorite woman; each woman will keep the proposal proposed by the man who has the highest rank in her ranking list among those who have proposed to her, and reject all the rest proposals. Each rejected man then proposes to his next favorite woman on his ranking list. The GS algorithm will continue this process until all women get proposals. When GS algorithm stops, each woman and man whose proposal the woman keeps become a pair of partners. All pairs of these partners form a stable matching. GS showed that a stable matching always exists and can be found in $O(n^2)$ iterations. Due to the dependency in GS algorithm, the number of iterations can not be easily reduced by parallelism regardless of the number of PEs used. The running time of parallel GS algorithm is $O(n^2 \log n)$ time

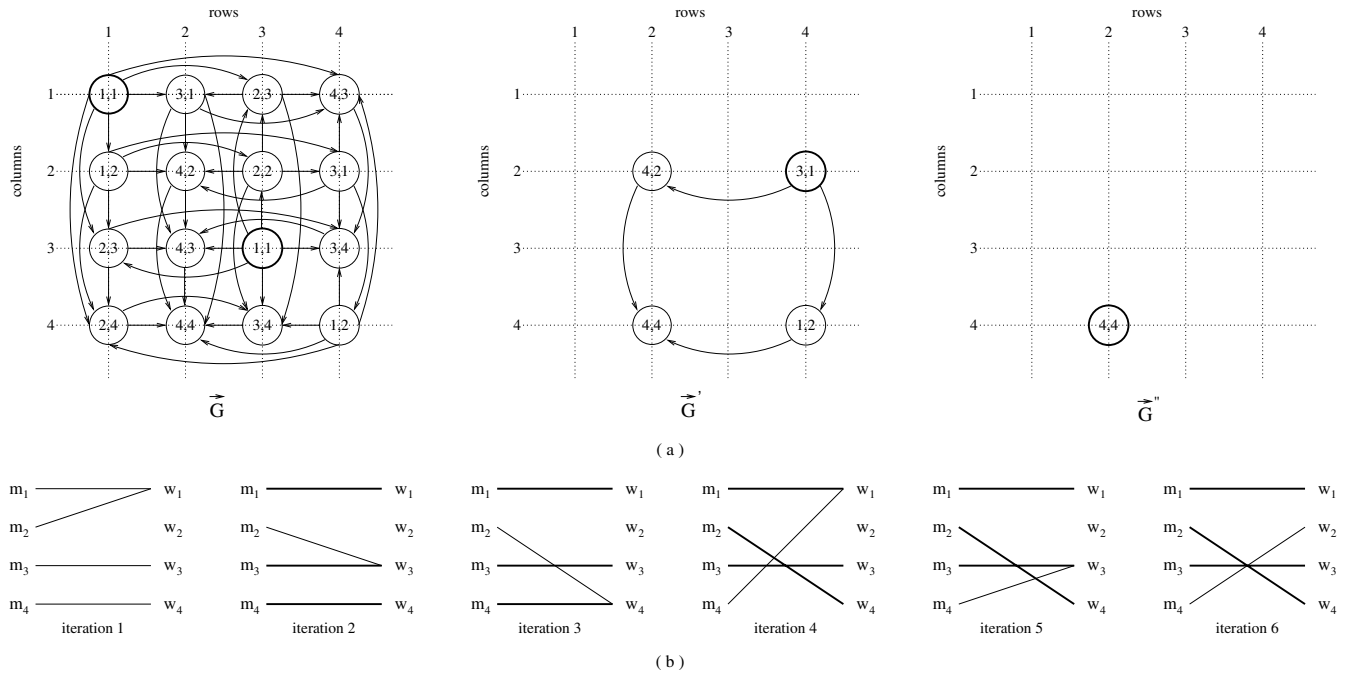

Fig. 2. (a) An acyclic dependency graph $\vec{G}$ and its reduced subgraph $\vec{G'}$ and $\vec{G''}$. The roots of $\vec{G}$ are $v_{1,1}$, $v_{3,3}$. The roots of $\vec{G'}$ and $\vec{G''}$ are $v_{2,4}$ and $v_{4,2}$ respectively. The root set is $\{v_{1,1},v_{3,3},v_{2,4},v_{4,2}\}$, and each root is marked as a dark circle. (b) GS algorithm finds the stable matching in 6 iterations. In each iteration, the new proposed proposals are marked as light lines and the kept proposals are marked as dark lines.

on n PEs since each iteration takes $O(\log n)$ time to find the minimum from at most n distinct numbers.

For stable matching problems with rooted dependency graphs, GS algorithm does not work as fast as FIND_ROOTS. As shown in Figure 1, to find the stable matching for Example 1, GS algorithm needs 5 iterations while FIND_ROOTS only needs 3 iterations. This means that $O(n)$ iterations are not sufficient for GS algorithm to find the stable matching for rooted dependency graphs. Furthermore, $O(n)$ iterations are not sufficient for GS algorithm to find the stable matching for acyclic dependency graphs. Figure 2 shows an example of an acyclic dependency graph. To find the stable matching of this example, GS algorithm needs 6 iterations and FIND_ROOTS needs 3 iterations.

Based on the above discussion, we know that the parallel GS algorithm finds the stable matching for a rooted dependency graph and an acyclic dependency graph in $O(n^2 \log n)$ time. However, FIND_ROOTS finds the stable matching for a rooted dependency graph and an acyclic dependency graph in $O(n \log n)$ time. Thus, the speedup for worst time complexity of FIND_ROOTS to GS algorithm is $O(n)$. Both FIND_ROOTS and GS algorithms take n man/woman ranking lists as inputs and every list contains n numbers, each with length of $\log n$ bits[1]. Thus, the needed spaces for both algorithms are the same. Table I compares the parallel GS algorithm and the parallel FIND_ROOTS algorithm for finding the stable matching in any rooted dependency graph or acyclic dependency graph with respect to time, the number of PEs and memory space (in bits).

[1]In this paper, all logarithms are in base 2.

Algorithm	Time	PEs	Space
GS	$O(n^2 \log n)$	n	$O(n^2 \log n)$
FIND_ROOTS	$O(n \log n)$	n^2	$O(n^2 \log n)$

TABLE I
COMPARISON OF ALGORITHMS FOR FINDING A STABLE MATCHING

III. Implementing the Scheduler

One of the objectives of our work is to design a scheduler that is feasible to implement. In this section, we present the hardware design and implementation of the scheduler based on the FIND_ROOTS algorithm. An $n \times n$ scheduler has n^2 pairs of inputs as $(wr_{1,1}, mr_{1,1}), \cdots, (mr_{n,n}, wr_{n,n})$, and n pairs of outputs as the indices of n roots, $s_1, s_2, \cdots, s_n$. The circuit consists of n^2 nodes arranged as an $n \times n$ array. Each node corresponds to an entry in a ranking matrix A and a vertex of A's dependency graph. We use $2n$ buses to interconnect n^2 nodes such that node $n_{i,j}$, where $1 \leq i, j \leq n$, is connected to the ith row bus, r_i, and the jth column bus, c_j. Each bus is $\log n$-bit wide. The first bit line of all n row buses are connected to a controller, which is used to select one out of possibly multiple bus requests (in the case of multiple root nodes exist in a graph). Each node $n_{i,j}$ has 2 inputs for reading its (h, v) pair, and one output to send out its index. Figure 3 shows the scheduler block diagram, circuit structure, and node block diagram of a 4×4 scheduler.

The operation of an $n \times n$ scheduler has n iterations. Initially, each node $n_{i,j}$ sets its $(h, v) = (wr_{i,j}, mr_{j,i})$. Each iteration operates as follows. For each node $n_{i,j}$, if it finds its $(h, v) = (1, 1)$ (i.e. it is a root node), it will send a 'request_signal' on its row bus. If the controller detects that there

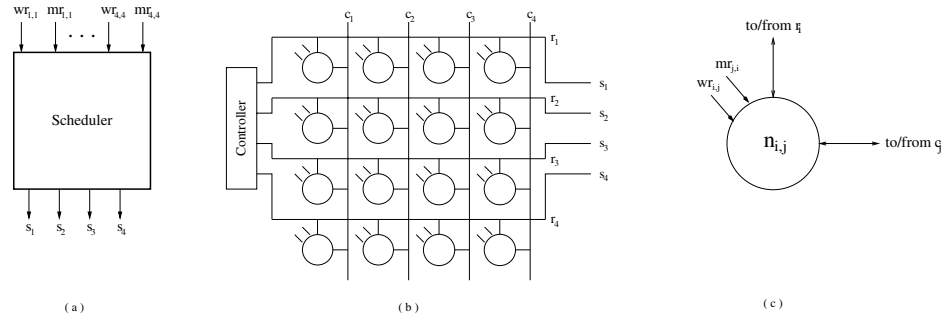

Fig. 3. A 4×4 scheduler design. (a) Scheduler block diagram (b) Circuit structure (c) Node block diagram.

Size	N=2	N=4	N=6	N=8	N=10	N=12
Timing	62.24	137.28	239.52	315.84	399.6	479.52
Area	1166	5410	12263	29283	41002	59342

TABLE II

TIMING AND AREA RESULTS OF THE SCHEDULER DESIGN.

are more than one buses requesting, it will confirm the bus with the minimum row index and send back a 'grant_signal' to the bus. Once a root node $n_{i,j}$ gets the 'grant_signal' from its row bus, it will send a 'mask_signal' on row bus r_i and column bus c_j to eliminate all nodes on row i and column j; meanwhile, it will update its $(h, v) = (0, 0)$ and send out its index. Once a node on row i (resp. column j) receives a 'mask_signal', it will send out its v (resp. h) value on its column (resp. row) bus. If a node with its h (resp. v) value is greater than the h (resp. v) value received from its row bus (resp. column bus), it will subtract its h (resp. v) value by 1.

The major advantage of this design is its simplicity. We only use $2n \log n$-bit buses to broadcast signals to nodes of the same row or the same column, and one $\log n$-bit priority encoder functioning as a controller for bus arbitration. Although n^2 nodes are used, the logic of each node is simple, which mainly includes 2 $\log n$-bit registers used to store its h and v values, one $\log n$-bit comparator, and one $\log n$-bit adder. We conducted simulations of the scheduler design on Synopsys's design tools. We wrote the VHDL [11] code, compiled and synthesized it on Synopsys's *design_analyzer* [12] using its library *lsi_10k*. The *design_analyzer* was directed to minimize the area cost of the design. Table II depicts the timing results (in terms of ns) and the area results (in terms of the number of 2-input NAND gates) of the scheduler design for $n = 2, 4, 6, 8, 10, 12$. The timing and the number of 2-input NAND gates are proportional to n and n^2 respectively, making the design feasible to be implemented with current CMOS technologies.

Another advantage of the design is its compatibility. Our scheduler design works well for real applications, including the case that ranks in some ranking lists are not distinct (e.g. cells with the same priority), the case that the lengths of some ranking lists are not equal to n (e.g. in some input queue, there is no cell destined for some output port), and the case that the sizes of man set and woman set are not equal (e.g. the number of input queues is not equal to the number of output queues).

IV. CONCLUSION

In this paper, we addressed the acyclic stable matching problem and proposed a parallel algorithm to solve the stable matching problem for rooted dependency graphs, which contains all acyclic dependency graphs as special cases. We designed a hardware scheduler based on the proposed algorithm. Simulation results show that the proposed scheduler design is feasible with current CMOS technologies. To the best of our knowledge, the scheduler design is the first hardware design for acyclic stable matching algorithms. It is very useful for switch controls of high-speed switches/routers. Future work includes hardware design optimization to achieve different application requirements.

REFERENCES

[1] D. Gale and L.S. Shapley, "College admissions and the stability of marriage", *American Mathematical Monthly*, vol. 69, pp. 9-15, 1962.

[2] D. Gusfield and R.W. Irving, *The stable marriage problem structure and algorithms*, MIT Press, 1989.

[3] G. Nong and M. Hamdi, "On the provision of quality-of-service guarantees for input queued switches", *IEEE Communications Magazine*, vol. 38, no. 12, pp. 62-69, Dec. 2000.

[4] N. McKeown, "Scheduling algorithms for input-buffered cell switches", Ph.D. Thesis, University California at Berkeley, 1995.

[5] B. Prabhakar and N. McKeown, "On the speedup required for combined input and output-queued switching", *Automatica*, vol. 35, no. 12, pp. 1909-1920, Dec. 1999.

[6] S. T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output queuing with a combined input/output-queued switch", *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1030-1039, Jun. 1999.

[7] I. Stoica and H. Zhang, "Exact emulation of an output queueing switch by a combined input output queueing switch", in *Proceedings of the 6th IEEE/IFIP IWQoS'98*, Napa Valley, CA, pp. 218-224, May 1998.

[8] G. Nong and M. Hamdi, "On the provision of integrated QoS guarantees of unicast and multicast traffic in input-queued switches", in *Proceedings of IEEE Globecom 1999*, vol. 3, pp. 1742-1746, 1999.

[9] A. C. Kam, K. Y. Siu, R. A. Barry, and E. C. Swanson, "A cell switch WDM broadcast LAN with bandwidth guarantee and fair access", *IEEE Journal of lightwave technology*, vol. 16, no. 12, pp. 2265-2280, Dec. 1998.

[10] A. Kam and K.-Y. Siu, "Linear complexity algorithms for QoS support in input-queued switches with no speedup", *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1040-1056, June 1999.

[11] IEEE Standards Board, *IEEE Standard VHDL Language Reference Manual*, 2002.

[12] Synopsys Design Analyzer Datasheet, available at http://www.synopsys.com/products/logic/deanalyzer_ds.html, 1997.