26th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing

Scheduler Accelerator for TDMA Data Centers

Ioannis Patronas[1,2], Nikolaos Gkatzios[1]
Vasileios Kitsakis[1] and Dionysios Reisis[1,2]
[1]National and Kapodistrian University of Athens,
Electronics Lab, Physics Dpt,
GR-15784 Zografos Greece,
[2]Institute of Communications and Computers (ICCS)
National Technical Univ. of Athens
Email: {johnpat,ngkatz,bkits,dreisis}@phys.uoa.gr

Konstantinos Christodoulopoulos
and Emmanouel Varvarigos
National Technical University of Athens,
School of Electrical and Computer Engineering
Email: kchristodou@ceid.upatras.gr,
vmanos@central.ntua.gr

***Abstract*—Today's Data Centers networks depend on optical switching to overcome the scalability limitations of traditional architectures. All optical networks most often use slotted Time Division Multiple Access (TDMA) operation; their buffers are located at the optical network edges and their organization relies on effective scheduling of the TDMA frames to achieve efficient sharing of the network resources and a collision-free network operation. Scheduling decisions have to be taken in real time, a process that becomes computationally demanding as the network size increases. Accelerators provide a solution and the present paper proposes a scheduler accelerator to accommodate a data center network divided into points of delivery (pods) of racks and exploiting hybrid electro-optical top-of-rack (ToR) switches that access an all-optical inter-rack network. The scheduler accelerator is a parallel scalable architecture with application specific processing engines. Case studies of 2, 4, 8, 16 processors configuration are presented for the processing of all the transfer TDMA time slot requests for the cases of 512 and 1024 ToR network nodes. The architecture is realized on a Xilinx VC707 board to validate the results.**

***Keywords* —Data Center, Scheduler, Parallel Algorithms, FPGA accelarator**

I. INTRODUCTION

Currently, data centers include a large number of servers running Virtual Machines (VMs) and their performance relies on the provided computing capacity, the architecture, the features and the performance of the underlying interconnection network. Interconnection designs most often are based on Fat Tree or even folded Clos architectures [1], [2], which for large scale data centers require a very high number of switches, cables and transceivers. More efficient interconnection schemes are proposed that involve an optical circuit switching and an electrical packet switching network [3], [4], [5], [6], [7]. The data centers of [2], [8] distinguish the heavy and long-lived traffic to assign it to the circuit switched network and the remaining to the packet switched network. Among the most important tasks in the course of the design process of a data center is the scheduling [6], [9], [10], [11], [12], leading to a high utilization of the network capacity. To improve the scheduling performance the authors of [12] propose a scalable parallel technique for real time scheduling of an optical interconnection.

Aiming at the dynamic sharing of the optical resources and a collision-free network operation, the Nephele data center architecture [13] follows the approach of the Time Division Multiple Access (TDMA) operation in the optical data center interconnection using *slots* as fixed time segments; at each link connecting a transmitter and a receiver node, each slot is dedicated to the transmission of a TDMA frame between these two nodes. The scalable, high capacity Nephele network can utilize up to 1600 Top-of-Rack (ToR) switches, each with 20 links. Consequently, the scheduling of the frames in the Nephele, network, i.e. the assignment in real time of each slot, link and wavelength to a rack-to-rack communication becomes a computationally demanding task. The assignment of the slots is planned for a *scheduling period* lasting for a relatively small number of time slots [14] and it is based on the servers requests. During a scheduling period the network performs the planned communication, while it gathers the updated servers communication demands to compute the slot assignments of the following period. Another important fact is that the data center is a scalable architecture and thus, the performance of the scheduling algorithm has to meet the real time requirements of any future network expansion.

Focusing on providing a time wise efficient technique for the slot assignment in the Nephele the current paper presents a scheduling accelerator architecture. The proposed architecture is based on a parallel greedy technique, it gets as input a *two dimensional traffic matrix* with the communication requests and it produces the *permutation matrix* containing the slot assignments. The proposed scheduler accelerator is advantageous because: a) it is able to perform using simple processing elements, b) it is scalable with respect to the number of processing elements used, a fact which leads to a real time scheduler architecture meeting the data center requirements and future expansions, c) it utilizes data structures that can be divided and mapped to the processing elements in a way that minimizes dependencies and consequently, the communication among the processors. FPGA architectures implemented with 2, 4, 8 and 16 processing elements on a Xilinx VC707 validate the results.

The paper is organized as follows: Section II highlights the data center architecture. Section III gives the problem definition and the data structures that lead to efficient mapping of the computations on the processing elements of the accelerator. Section IV introduces the parallel technique. Section V shows the details of the accelerator architecture and the FPGA implementation, Section VI presents the results and finally, Section VII concludes the paper.

II. NEPHELE NETWORK ARCHITECTURE

The Nephele network is divided in points-of-delivery (pods) of racks and is built out of hybrid electrical/optical top-of-rack (ToR) switches and all-optical pod switches. In the Nephele interconnect there are I parallel optical planes, each of which consists of R unidirectional rings connecting P pods. Each one of the R fiber rings of the I planes carries wavelength division multiplexing (WDM) traffic. Figure 1 depicts the Nephele network. A pod consists of I pod switches and W ToR switches interconnected: each ToR switch has I ports facing the I pod switches and, in particular, each port is connected to a different pod switch. The ToR switch has S southbound ports (hosts, storage devices, or other systems).

Buffers are located only at the ToR switches and the Nephele network operates in a slotted TDMA manner to avoid collisions, resembling the operation of a single TDMA switch. Time slots are dynamically assigned for each plane to ToR-to-ToR communications by a central scheduler based on their respective traffic requirements. Scheduling decisions are taken in periods of T time slots to enable the aggregation and suppression of monitoring and control information.

In Nephele the communication begins with traffic originating from a ToR entering a pod switch a fast 1×2 space switch keeps it within the pod if the destination ToR belongs to the pod else it forwards it to the rings and to the next pod switch. The local intra-pod traffic is passively routed by a $1 \times W$ arrayed waveguide grating (AWG) based on the signal wavelength and the input port.

Inter-pod traffic is routed via the fast 1×2 switch towards a second $W \times R$ cyclic arrayed waveguide grating (CAWG) followed by couplers that combine multiple CAWG outputs into the fiber rings. So, the traffic enters the ring according to the CAWG function, propagates in the same ring through intermediate pod switches and is dropped at the destination pod. These routing decisions are applied by setting appropriately the related wavelength selective switches (WSS) in the pod switches. The WSSs can select whether traffic is passed or dropped on a per-fiber, per-wavelength and per-slot basis. Thus, each intermediate pod sets the related WSS to the pass state, while at the destination the related WSS is set to drop. The drop ports of the WSSs - corresponding to all the rings - are introduced into a $1 \times W$ AWG and are passively routed to the W ToRs in that pod. Through this AWG the traffic reaches the specific destination ToR.

ToR switches periodically report their bandwidth requests to the network controller, or applications report their requirements to the controller. Then, the controller constructs a $(W \times P) \times (W \times P)$ traffic matrix (TM) at the end of each reporting period: each $TM((w_1, p_1), (w_2, p_2))$ corresponds to the number of time slots requested for the communication between a source $TOR(w_1, p_1)$ with a destination $TOR(w_2, p_2)$, where p_1 and p_2 indicate the source and destination pods and w_1 and w_2 indicate the wavelength/position of the source and destination TORs inside the related pods, $1 \leq w_1, w_2 \leq W$ and $1 \leq p_1, p_2 \leq P$. Constraints are met to avoid collisions.

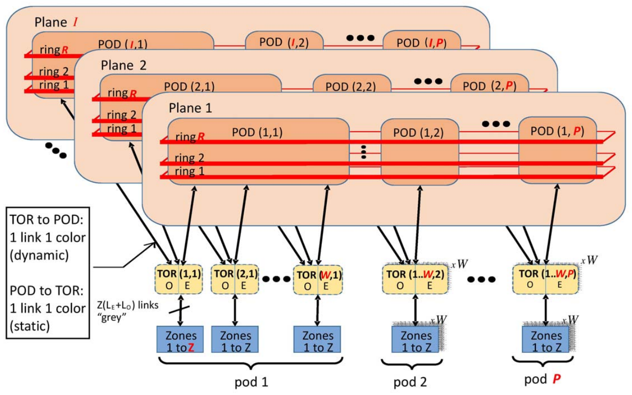


Fig. 1. NEPHELE Network

III. PROBLEM DEFINITION AND DATA STRUCTURES

The current section describes first, the scheduling problem and the dependencies limiting a parallel accelerator design; second, it presents the data structures given as input to the scheduler and those that are produced as output to the network controller.

A. Scheduling Problem Definition

Following the above description of the Nephele network and the TDMA operation, where the transmission time is divided into TDMA (time) slots and during each slot we can transmit one TDMA frame, the scheduling problem is as follows: divide the network operation time into scheduling periods with each period consisting of a few (T) TDMA slots; for each TDMA slot of the next period, the scheduler has to map a request for sending a TDMA frame from a ToR transmitter to a ToR receiver so that there are no conflicts with respect to the assignment of the other transmitters, receivers and the network links that are appointed to them. For example, during a slot transmitters p and q cannot both send to the receiver w and also, the requests for transmission from p to w and from q to y cannot be both met if their paths share any link and wavelength.

To accomplish the above, the scheduler gets as input a two dimensional array, namely the *Traffic Matrix* requesting for each transceiver pair $(i,\ j)$ the number of frames to be sent during the next period. In the course of the scheduling process, in order to avoid conflicts during each period the scheduler has to keep track of the assignments made so far for each slot. For this purpose, it creates two auxiliary arrays: the first is the *Receiver Constraint Matrix* storing for each receiver the slots already assigned as busy and the second array is the respective *Transmitter Constraint Matrix*. The output of the scheduling process is the two dimensional array called the *Permutation Matrix* showing for each transmitter to what receiver and which slot is going to transmit in the next period.

Given the size of the network and the real time constraints, the scheduling process becomes a complicated task, which a single processor cannot complete in a scheduling period. The proposed scheduling accelerator based on a parallel architecture is a suitable approach. The problem though, as described above is not inherently parallel, because a straightforward division of requests into multiple processors will let each processor to handle requests as responsible for a subset of either transmitters or receivers. In the former case there will be conflicts when the processors will try to access the same receiver array entries and in the latter case the same holds for the transmitter array entries. Moreover, the dividing strategy for the transmission requests and the parallel technique have to result in an interconnection among the accelerator processors and memory modules of relatively low latency and implementation cost. The proposed parallel technique is based on an effective strategy for dividing and handling the requests, it results in an accelerator architecture with simple processing elements and a barrel shifter for interconnection. Moreover, it can be scaled with respect to the number of processors and/or the network size.

B. Data Structures

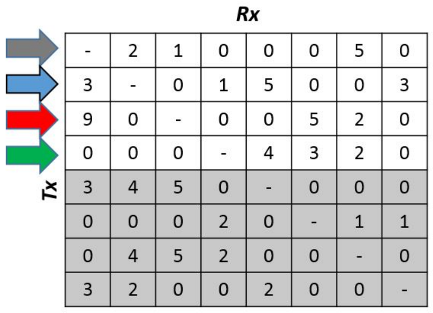
Traffic Matrix: The Traffic Matrix (TM) is a $N \times N$ array created by the central management (the SDN Controller) and it is the input of the scheduling algorithm. Note that N is the number of the ToRs and that also, $N = W \times P$ ($W = 80$ wavelengths and $P = 20$ planes); moreover, N/P are the time slots of each scheduling period. Entry (i, j) has the number of time slots requested for the communication between the transmitter i and the receiver j: each row i, where $i = w \times p$, of the Traffic Matrix represents the transmitter of the w^{th} ToR of the p^{th} pod; and each column j (as above $j = w \times p$) represents the receiver of the w^{th} ToR of the p^{th} pod. Figure 2 depicts the format of the Traffic Matrix for a network with eight (8) ToRs.

Permutation Matrix (PM): PM is the output of the scheduling algorithm. It is a $N \times N$ two dimensional array with each row i standing for the transmitter i and each column j the j^{th} *generic slot*, which is defined as the combination of a time slot and a specific plane, i.e. a time slot for 20 different planes is equal to 20 different generic slots. Hence, the number of generic slots in a scheduling period is the number of slots N/P (from paragraph III-B) multiplied by the number of planes P; thus, the total number of generic slots is N. Each entry (i, j) of the PM contains the id of the receiving ToR switch. Hence, the PM includes all the required information for coupling during any slot of a scheduling period: the transmitter i (specifies the ToR identification and the plane for transmission p) during the generic slot j (specifies the slot and the plane p to receive from) is coupled with the ToR receiver written in the entry (i, j). The format of the Permutation Matrix for a network with eight (8) ToRs is shown in Figure 3.

Constraints Matrices: the scheduling algorithm satisfies the traffic requests while conforming with the network's constraints to avoid collisions at any layer of the network. The Nephele's network constraints can be expressed in the form of the binary matrices presented in the following paragraphs.

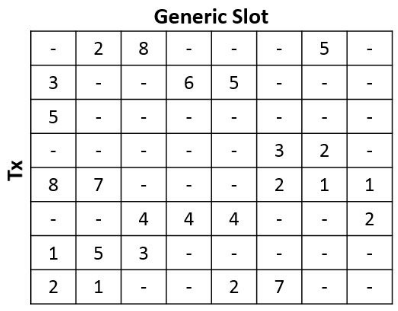
Transmitter's Constraint Matrix (TCM): TCM is a two dimensional $N \times N$ array, which indicates that a transmitter has been assigned for communication during a generic slot. The rows of the TCM represent the transmitters of the ToR switches and the columns the generic slots of the scheduling period. Each entry (i, j) is either 0 or 1 depending on the utilization of transmitter i during the generic slot j. An example of the TCM format is shown in Fig. 4.

Receiver's Constraint Matrix (RCM): it is a $N \times N$ array with each row i standing for the receiver i and each column j for the j^{th} generic slot and hence, $(i, j) = 1$ if the receiver is occupied during that generic slot, 0 otherwise. Figure 5 shows the format of the RCM for a network with eight (8) ToRs.



-	2	1	0	0	0	5	0
3	-	0	1	5	0	0	3
9	0	-	0	0	5	2	0
0	0	0	-	4	3	2	0
3	4	5	0	-	0	0	0
0	0	0	2	0	-	1	1
0	4	5	2	0	0	-	0
3	2	0	0	2	0	0	-

Fig. 2. Traffic Matrix


Generic Slot
Tx

-	2	8	-	-	-	5	-
3	-	-	6	5	-	-	-
5	-	-	-	-	-	-	-
-	-	-	-	-	3	2	-
8	7	-	-	-	2	1	1
-	-	4	4	4	-	-	2
1	5	3	-	-	-	-	-
2	1	-	-	2	7	-	-

Fig. 3. Permutation Matrix

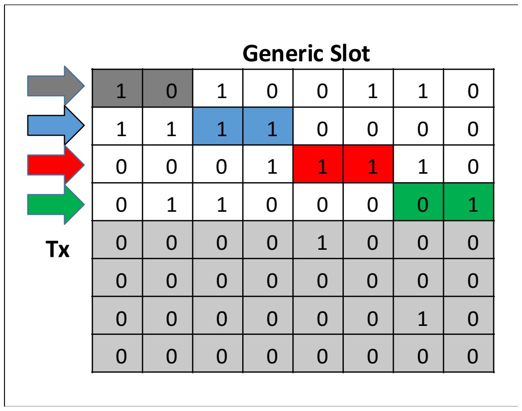


Fig. 4. Transmitter's Constraint Matrix (TCM)

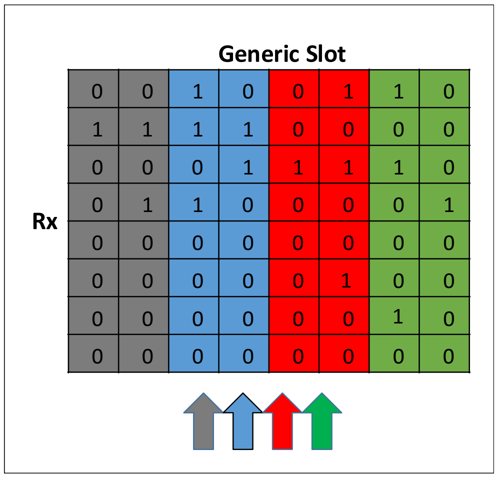


Fig. 5. Receiver's Constraint Matrix (RCM)

IV. Design of the Parallel Algorithm

The design of the scheduler accelerator follows the approach of parallelizing a sequential greedy scheduling technique. The parallel technique targets application specific processing engines in the Single Instruction Multiple Data (SIMD) model/class. The sequential greedy technique examines the entries of the Traffic Matrix and for each non zero entry processes the time slots: for each time slot it allocates an appropriate entry of the Permutation Matrix, if the scheduling constraints are met. The parallel technique includes similar operations, it divides though the data structures into blocks that can be processed in parallel to achieve minimization of the communication among the processing elements and avoid memory access conflicts or violation of the network's constraints. The current section presents first, how the data structures are divided. Second, it presents the functionality of the parallel technique, while the exact steps in an algorithmic fashion will be described in section V.

A. Data Structures Handling

The Traffic Matrix is divided into blocks of rows ($rowblocks$); each $rowblock$ contains E rows, where E is the number of processing elements. Each time a $rowblock$ is processed, a single processing element is assigned to process one row of that $rowblock$. An example of four (4) processing elements on a 8×8 TM and their currently assigned rows are shown in Figure 2: the colored arrows represent four (4) processing elements and the upper four rows the currently processed $rowblock$. We note here that, by applying rotation on the elements' assignments the technique allows all the processing elements to have access to all the rows of the TM.

The two constraint matrices TCM and RCM are divided into N/E $columnblocks$, where N is the number of generic slots (and that of the ToRs) and E is number of processing elements. Each of the processing elements (e.g. E_i) has access to only one of the $columnblocks$, (e.g. the $columnblock_i$ and hence, we avoid collisions as it will be shown in section V-A). The partition of TCM and RCM in $columnblocks$ is depicted in Figures 4 and 5 respectively: each processing element has access to the columns that have the same color with the element.

B. Parallel Algorithm

The parallel algorithm is executed in phases. During each phase the algorithm processes one $rowblock$ of the TM and therefore, the total number of phases equals to the number of rows of the TM divided by the number of processing elements E. Each phase consists of a variable number of iterations, the *phase iterations*. The number of *phase iterations* varies between 1 and the number of processing elements E.

At the start of each phase, at the first iteration, the processing elements are assigned to the rows of the TM, so that the processing element E_i will process the i^{th} row of the $rowblock$ and as mentioned in section IV-A has access to the $columnblock_i$ of the RCM. The processing elements execute the operations of the sequential greedy technique on their assigned row: for each time slot request in the TM, the element allocates an appropriate entry of the Permutation Matrix, if the scheduling constraints are met. In order to a) avoid conflicts by letting each E_i to assign different generic slots than each E_j and b) balance the computational load among the E_is: we allow each element, when it processes a row, to assign up to N/E (the number of generic slots divided by the number of elements) generic slots. The iteration is completed when all the processing elements have finished their task: each element E_i it a) has completed processing the row's requests, or b) run out of the generic slots that is allowed to use (reached the number N/E), or c) has filled all the entries in $columnblock_i$.

In the following iterations: during the phase iteration j the processing element E_i processes the $(i+j)$ mod E row of the currently processed $rowblock$ while it has still access only to the $columnblock_i$. We note here that, the *phase iterations* has been designed to overcome the fact that the processing element E_i has access to only a subset of all the slots ($columnblock_i$) and to guarantee that the requests of i^{th} row of the Traffic

Matrix can be satisfied in the entire set of slots (all the columnblocks) by letting all the processing elements access that row.

The algorithm proceeds to the next phase if all the entries of the $rowblock$ are equal to zero or if the number of iterations equals the number E. In the latter case there are still communication requests of the current $rowblock$ that are not met; these requests will be added to the Traffic Matrix of the next scheduling period.

It is noteworthy here that, the design of the algorithm's iterations leads to an accelerator architecture of low implementation cost, because it consists of E simple processing elements, E memory banks (each stores N/E rows of the TM) and an interconnecting switch, which has to realize only cyclic shifts during each iteration.

V. Accelerator Architecture and FPGA Implementation

The scheduler accelerator architecture is based on a parallel scheme organized in SIMD model with E simple processing elements. The architecture's main blocks also include: a memory storing the TM divided into E banks, which communicate through a Barrel Shifter with the processing elements, since only cyclic shifts are needed in each phase; the local banks of the processing elements storing the E parts of the RCM; and finally, the global controller of the architecture. Figure 6 depicts the main blocks of the scheduling accelerator architecture (example of an 8 processing elements configuration).

A. Accelerator Architecture

The controller creates and manages the phases and the iterations of the parallel technique (section IV-B). Depending on the current phase and iteration the controller dictates the configuration of the barrel shifter in order to realize the required connections between the processing elements and the rows of the TM, that are stored in the TM Block Rams. In addition, the controller manages the functionality and the synchronization of the processing elements: the controller awaits all of the processing elements to complete the processing of their currently assigned TM row and by realizing a cyclic shift through the barrel shifter, it assigns a new row to each processing element. Then it signals to the elements for the start of the next iteration and if the phase is over, the signal will mark also the start of another phase. As described in Section IV-B the phases of the algorithm consist of a variable number of iterations, that can be lesser than the maximum if the currently processed $rowblock$ contains zero requests. The processing elements notify the Rowblock Utilization Block regarding the status of the TM row they processed: $rowserved$ if the row has no more requests to be processed and $row\ not\ served$ if there are still requests pending in the row. The controller gathers the $row\ status$ signals from all the elements and proceeds to the next phase if all the rows of the $rowblock$ are served. The exact steps that the controller executes are shown in the Algorithm 1.

```
while current phase number ≤ total phases do
    while current iteration number ≤ total iterations do
        read processing elements status;
        if all rows served then
            proceed to $next\ phase$;
        else
            proceed to $next\ iteration$;
        end
    end
    proceed to $next\ phase$;
end
```

Algorithm 1: Controller Algorithm

```
1: Receive iteration number $j$ from the controller;
   status ← $processing$;
2: Process $(i+j)mod(E)$ row of the current $rowblock$;
3: while { not all cells checked
          and generic_slots for $element_i$ < full
          and $columnblock_i$ ≠ full } do
       process $next\ request$ ;
   end
4: if row requests = 0 then
       status ← $row\ served$;
   else
       status ← $row\ not\ served$;
   end
5: Wait for $next\ iteration$;
```

Algorithm 2: Processing Element E_i Algorithm

Figure 7 presents the architecture of the processing element. At the beginning of each iteration the TCM Vector Register is initialized with zeros and the RCM Vector Register receives the appropriate vector from the local RCM Block Ram. These vectors (utilization vectors) represent the occupation status of the transmitter and the receiver respectively during the generic slots that each processing element is allowed to process: each

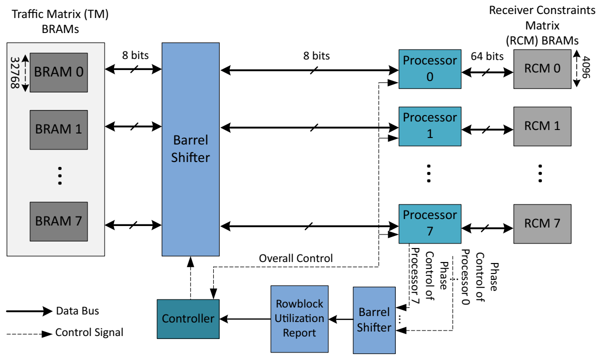


Fig. 6. Scheduler Accelerator Architecture's Main Blocks

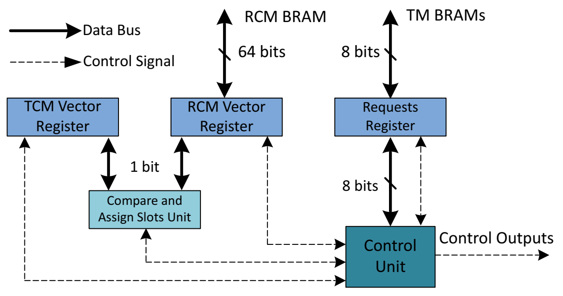


Fig. 7. Processing Element Architecture Overview

bit represents one generic slot. The Requests Register stores temporarily the traffic requests of the currently processed cell of the TM. For each request in the TM, the Compare and Assign Slots Unit compares the two utilization vectors and searches for an index where both vectors have a zero; that means that both the receiver and the transmitter are not utilized; hence, they are able to serve the traffic request during the examined slot (given by the index). Then, it repeats the search until it will locate all such indexes or it will complete the TM's requests.

The Traffic Matrix is stored in a set of Memory Banks. Each of the memory banks can be accessed by any of the processing elements via the barrel shifter: in the example of Figure 6 is 8×8. The entries of the Traffic Matrix are distributed in the memory banks so that all the rows of any $rowblock$ that are assigned to the processing elements during a given phase are located in a different bank to avoid collisions. Row i of the Traffic Matrix is stored in memory bank j where $j = (i) mod(E)$ and E is the number of processing elements. Since each processing element i has access to a subset of the generic slots, each $columnblock_i$ of the RCM can be stored in its local memory (RCM_i in Figure 6). We note here that, the parallel implementation does not use the entire TCM structure. Instead, for efficiency purpose, each element E_i creates a row of the TCM (all entries are 0), each time is using the corresponding row of the TM, and it sets "1" to a TCM entry whenever it assigns a generic slot to that transmitter.

The steps executed by each processing element, for all the elements in parallel, are described in Algorithm 2. In step 1 the element waits for the iteration number given by the controller and starts processing in Step 2. During step 3 it processes the requests in that row while it a) has not completed processing the row's requests, and b) still is allowed to use generic slots (has assigned less than N/E slots), and c) $columnblock_i$ has still zero entries. Step 4 informs the controller regarding the status of processing the row and finally, in step 5 if E_i has stopped, it waits the message from the controller that all the elements stopped and a new iteration will begin.

B. Accelerator Scalability

The above description leads to the fact that is a straightforward task to scale the accelerator architecture with respect to the number of processing elements, since the only module to be augmented is the barrel shifter switch. In the following we will show the results of implementing the architecture with 2, 4, 8 and 16 processing elements and in Section VI the performance that these configurations can achieve. The implementations include networks for 512 and 1024 ToR switches.

C. FPGA implementation

Our first experiment was the parallel accelerator implemented on a Virtex VC707 board. The architecture targeted the scheduling of a 512-ToR network featuring scheduling period of 512 generic slots, TM size 512×512 and number of planes $P = 16$. According to these specifications the scheduling period is $6.8ms$ and the time needed to transfer the TM or output the PM is less than $1ms$. We note here that: a) the network $load$ is defined as the ratio of the total ToR traffic in a reporting period over the total capacity of the ToR; b) the $density$ of the traffic matrix is defined as the ratio of the total number of transceiver pairs requesting communication over the total number of transceiver pairs. Fig. 8 shows that the 8 processing elements configuration can accomplish the scheduling task in real-time even in the case of load 50% and density 1.5% (Fig. 8 iii); such cases are considered as extreme for the operation of data centers, since the load seldom is greater than 20% Density though, varies from 1% to 25%. Regarding the FPGA resources utilization: a single processing element occupies 295 LUTs and 195 FFs and the 8-element parallel accelerator's processing elements 2660 LUTs and 1828 FFs, 216 LUTs are for the barrel shifter. The resources occupied by the 8-element configuration, apart the $8 \times (295\ LUTs\ and\ 195\ FFs)$ is required for their interconnection. The F_{max} is 202.02 MHz.

The scalability of the design is shown by the implementations of the architectures with 2, 4, 8 and 16 processing elements. Table I shows the resources occupied by the four configurations of 2, 4, 8 and 16 processing elements all for the 512-ToR network (they all require the same memory volume in the shared and local memories). F_{max} is almost the same for all configurations. The small variations are due to different placements on the FPGA.

An advantage of the design is that each time we double the processing elements, i.e. from 2 to 4, from 4 to 8 and from 8 to 16, the increase of the resources is linear with respect to the number of processing elements: the 4 processing element configuration uses almost 500 LUTs and 400 FFs more than the 2. The 8 use almost 1000 LUTs and 800 FFs more than the 4 and finally, the 16 use almost 2000 LUTs and 1600 FFs than the 8.

These results prove that the approach of including only cyclic shift permutations in the parallel technique constitutes an attractive solution, which leads to a scalable parallel accelerator architecture effectively supporting the scheduling requirements of any size TDMA data center.

TABLE I
FPGA XILINX VC707 RESOURCES UTILIZATION FOR THE PARALLEL ACCELERATOR WITH 2,4,8 AND 16 PROCESSING ELEMENTS.

Number of Processors	Architecture's LUTs	Architecture's Flips Flops	Barrel Shifter LUTs	Fmax (MHz)
2	2150	1432	18	197.24
4	2660	1828	72	198.81
8	3672	2632	216	202.02
16	5600	4240	576	196.46

VI. RESULTS

Figure 8 depicts the execution times required by the implemented accelerator configurations of 2, 4, 8 and 16 processing elements for a variety of loads and densities (defined in subsection V-C) and two network configurations: 512 and 1024 ToRs. The first nine curves show the experimental results for the 512-ToR network and the last three for the 1024-ToR network. For sake of comparison we have used the same operating frequency in all the experiments: 196 MHZ (5.1 nsec clock). We focus on densities and loads that are common to the data centers operation. These are loads that are up to 25% and we include the 50% load to show the architecture's performance in extreme cases. The densities range from 1% to 25%.

The first significant result comes from the observation that in all the experiments the speedup, using the 2 processing element configuration as the comparison basis, achieved by the architecture follows the N/log_2N Ahmdal's law, where N is the number of processing elements. For example, in the curve i) the execution times are 2.13, 1.01, 0.6 and 0.44 msec for the 2, 4, 8 and 16 processing element architectures respectively. All the other eleven curves report results that follow similar shape and thus, they show the same speedup for processing all the examined cases of loads and densities in both network configurations (512 and 1024 ToRs).

The second notable result is that the execution times increase linearly with respect to either the load or the density of the communication requests in the data center. This fact provides the advantage of a predictable performance of the scheduler accelerator in the majority of the cases. Finally, for the two different sized networks, the execution times follow the Traffic Matrix size: for the cases of densities 1.5% and 4% and load 10% that were examined for both networks, the execution times are about four times greater for the 1024-ToR network compared to those of the 512-ToR network.

VII. CONCLUSION

The current paper introduced a scheduler accelerator parallel architecture for TDMA data centers. The accelerator is based on a parallel greedy technique and its performance benefits from the fact that two dimensional arrays can be divided into blocks and be assigned to the processing elements in a way that minimizes the communication among the processors. The parallel technique leads to scalable parallel architecture achieving N/log_2N speedup, implementation cost proportional to the N processing elements and effectively supporting the real-time scheduling decision requests in the data center.

REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Aug. 2008. [Online]. Available: http://doi.acm.org/10.1145/1402946.1402967

[2] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 51–62. [Online]. Available: http://doi.acm.org/10.1145/1592568.1592576

[3] D. Alistarh, H. Ballani, P. Costa, A. Funnell, J. Benjamin, P. Watts, and B. Thomsen, "A high-radix, low-latency optical switch for data centers," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 367–368. [Online]. Available: http://doi.acm.org/10.1145/2785956.2790035

[4] H. H. Bazzaz, M. Tewari, G. Wang, G. Porter, T. S. E. Ng, D. G. Andersen, M. Kaminsky, M. A. Kozuch, and A. Vahdat, "Switching the optical divide: Fundamental challenges for hybrid electrical/optical datacenter networks," in *Proceedings of the 2Nd ACM Symposium on Cloud Computing*, ser. SOCC '11. New York, NY, USA: ACM, 2011, pp. 30:1–30:8. [Online]. Available: http://doi.acm.org/10.1145/2038916.2038946

[5] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: A hybrid electrical/optical switch architecture for modular data centers," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 339–350. [Online]. Available: http://doi.acm.org/10.1145/1851182.1851223

[6] G. Porter, R. Strong, N. Farrington, A. Forencich, P. Chen-Sun, T. Rosing, Y. Fainman, G. Papen, and A. Vahdat, "Integrating microsecond circuit switching into the data center," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 447–458. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486007

[7] K. Tokas, C. Spatharakis, I. Kanakis, N. Iliadis, P. Bakopoulos, H. Avramopoulos, I. Patronas, N. Liakopoulos, and D. Reisis, "A scalable optically-switched datacenter network with multicasting," in *2016 European Conference on Networks and Communications (EuCNC)*, June 2016, pp. 265–270.

[8] Q. Li, S. Rumley, M. Glick, J. Chan, H. Wang, K. Bergman, and R. Dutt, "Scaling star-coupler-based optical networks for avionics applications," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 5, no. 9, pp. 945–956, Sept 2013.

[9] J. E. Hopcroft and R. M. Karp, "A n5/2 algorithm for maximum matchings in bipartite," in *Proceedings of the 12th Annual Symposium on Switching and Automata Theory (Swat 1971)*, ser. SWAT '71. Washington, DC, USA: IEEE Computer Society, 1971, pp. 122–125. [Online]. Available: http://dx.doi.org/10.1109/SWAT.1971.1

[10] D. Shah and J. Shin, "Randomized scheduling algorithm for queueing networks," *CoRR*, vol. abs/0908.3670, 2009. [Online]. Available: http://arxiv.org/abs/0908.3670

[11] X. Lin and S. Rasool, "A distributed joint channel-assignment, scheduling and routing algorithm for multi-channel ad-hoc wireless networks," in *IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, May 2007, pp. 1118–1126.

[12] J. A. Corvera, S. M. G. Dumlao, R. S. J. Reyes, P. Castoldi, N. Andriolli, and I. Cerutti, "Hardware implementation of an iterative parallel scheduler for optical interconnection networks," in *Advanced Photonics 2016 (IPR, NOMA, Sensors, Networks, SPPCom, SOF)*. Optical Society of America, 2016, p. NeM3B.4. [Online]. Available: http://www.osapublishing.org/abstract.cfm?URI=Networks-2016-NeM3B.4

[13] www.nepheleproject.eu.

[14] K. Christodoulopoulos, K. Kontodimas, K. Yiannopoulos, and E. Varvarigos, "Bandwidth allocation in the nephele hybrid optical interconnect," in *2016 18th International Conference on Transparent Optical Networks (ICTON)*, July 2016, pp. 1–4.

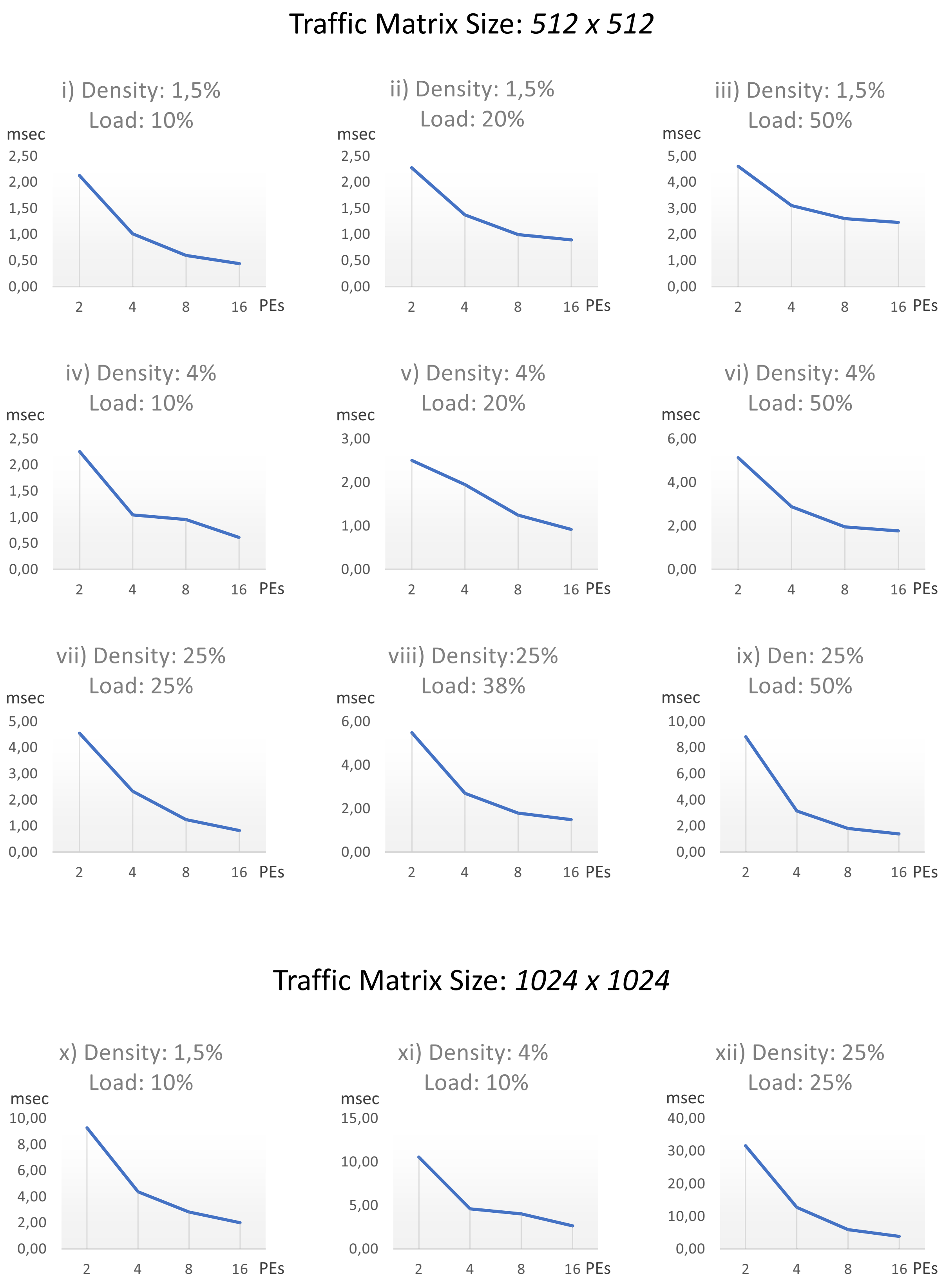


Fig. 8. Execution Time for the 2, 4, 8 and 16 processing element Parallel Architectures for a variety of densities and loads. On the Horizontal axis appear the number of processing elements and on the Vertical axis the execution times in msec.