Improving Scheduling Efficiency for High-Speed Routers with Optical Switch Fabrics

Bin Wu, Kwan L. Yeung and Xin Wang
Dept. of Electrical and Electronic Engineering
The University of Hong Kong, Pokfulam, Hong Kong
E-mail: {binwu, kyeung, xinwang}@eee.hku.hk

***Abstract*—Aiming at providing 100% throughput with bounded packet delay, we consider traffic scheduling in high-speed routers with optical switch fabrics. Because of the switch reconfiguration overhead, a speedup in the switch fabric is essential. For a given packet delay bound, our objective is to minimize the overall speedup $S=S_{\text{reconfigure}}\times S_{\text{schedule}}$ so as to lower the implementation cost. Leveraging on the existing ADAPTIVE and DOUBLE algorithms, we show the speedup can be reduced by improving scheduling efficiency. Specifically, following the traffic matrix decomposition in ADAPTIVE and DOUBLE, we shift some packets from the residue matrix R to the quotient matrix Q, while keeping the number of configurations required to cover each matrix the same. We reduce the number of time slots required to send the diminished residue matrix. In case of DOUBLE, this translates into a 12.5% cut in S_{schedule} (from 2 to 1.75). We call the resulting algorithm Scheduling Residue First (SRF).**

Keywords-Performance guaranteed switching; reconfiguration overhead; scheduling Residue First (SRF); speedup.

I. INTRODUCTION

Recent progress on optical switching technologies [1-2] has enabled the implementation of high-speed routers with optical switch fabrics as shown in Fig. 1. The optical switch fabric provides huge switching capacity as demanded by the backbone routers in the Internet. Since the input/output modules are connected with the central switch fabric by optical fibers, they can be distributed into several racks. As a result, power consumption can be reduced for each rack. This makes the resulting switch architecture more scalable.

On the other hand, optical switch fabric usually needs a non-negligible amount of *reconfiguration overhead* time to change its switch configuration from one to another, as well as to synchronize the optical signals arriving at the input ports [3]. During this period, no packet can be transmitted across the switch fabric. To achieve *performance guaranteed switching* (i.e. 100% throughput with bounded packet delay) [4-6], the switch fabric must run faster to compensate for both the reconfiguration overhead and the scheduling inefficiency. The required *speedup S* is defined as the ratio of the internal packet transmission rate to the external line-rate ($S\geq 1$).

Assume time is slotted and each time slot can accommodate one fixed-size packet. Based on the $N\times N$ unicast switch in Fig. 1, several scheduling algorithms are proposed to achieve performance guaranteed switching [4-6]. They all adopt the same four-stage scheduling procedure as shown in Fig. 2. Stage 1 is for traffic accumulation. A traffic matrix $\boldsymbol{C(T)}=\{c_{ij}\}$ is obtained at the input buffers every T time slots. Each entry c_{ij} denotes the number of packets arrived at input i and destined to output j. As a common assumption in [4-6], the entries in each

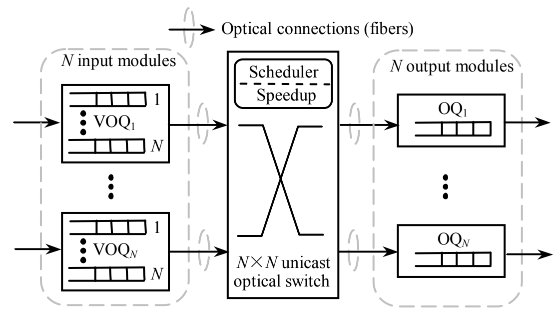

Fig. 1. A scalable optical switch fabric for high-speed routers.

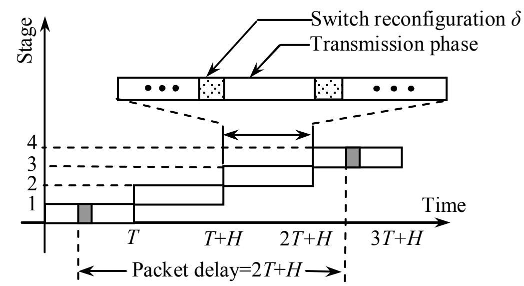

Fig. 2. Timing diagram for packet scheduling.

line (i.e. row or column) of $\boldsymbol{C(T)}$ sum to at most T. In Stage 2, a scheduling algorithm generates a schedule consisting of (at most) N_S switch configurations in H time slots. Each configuration is denoted by a permutation matrix $\boldsymbol{P_n}=\{p^{(n)}_{ij}\}$ ($N_S\geq n\geq 1$), where $p^{(n)}_{ij}=1$ means that input port i is connected to output port j (In this case, we also say that $\boldsymbol{P_n}$ covers entry (i, j)). A weight ϕ_n is assigned to each $\boldsymbol{P_n}$, indicating the number of slots that $\boldsymbol{P_n}$ should be kept for packet transmission. The set of N_S configurations generated must *cover* $\boldsymbol{C(T)}$, i.e. $\sum^{N_S}_{n=1}\phi_n p^{(n)}_{ij}\geq c_{ij}$ for any $i, j\in\{1, \ldots, N\}$. Then $\sum^{N_S}_{n=1}\phi_n$ is the number of slots required to transmit all the packets in $\boldsymbol{C(T)}$. Let each reconfiguration take an overhead of δ time slots. Accordingly, sending $\boldsymbol{C(T)}$ requires $\delta N_S+\sum^{N_S}_{n=1}\phi_n$ time slots. This is generally larger than the traffic accumulation time T. Without speedup, 100% throughput is not possible. Stage 3 is for actual packet transmission, where the switch fabric is reconfigured according to the N_S configurations. At a speedup of S, the slot size for a single packet transmission in Stage 3 is shortened by S times. Then 100% throughput is ensured by having

$$\delta N_S+\frac{1}{S}\sum_{n=1}^{N_S}\phi_n=T\ . \qquad (1)$$

The values of N_S and $\sum^{N_S}_{n=1}\phi_n$ in (1) are determined by the scheduling algorithm. Note that the total reconfiguration

This work is supported by Hong Kong Research Grant Council Earmarked Grant HKU 7032/01E.

overhead time δN_S cannot be reduced by speedup and thus $T>\delta N_S$. Finally, Stage 4 takes another T time slots to send the packets onto the output lines from the output buffers.

Rearranging (1), we have

$$S = \frac{1}{T-\delta N_S}\sum_{n=1}^{N_S}\phi_n = S_{\text{reconfigure}} \times S_{\text{schedule}}, \qquad (2)$$

where $S_{\text{reconfigure}}$ and S_{schedule} are defined as

$$S_{\text{reconfigure}} = \frac{T}{T-\delta N_S} \qquad (3)$$

$$S_{\text{schedule}} = \frac{1}{T}\sum_{n=1}^{N_S}\phi_n \qquad (4)$$

$S_{\text{reconfigure}}$ is the speedup factor to compensate for the idle time caused by reconfigurations, whereas S_{schedule} is the speedup factor to compensate for the scheduling inefficiency.

In Fig. 2, packet delay is bounded by $2T+H$ slots and $T>\delta N_S$. With a smaller N_S, T and the packet delay bound can be reduced. But N_S must be no less than N. Otherwise, the N_S configurations are not sufficient to cover every entry in $\boldsymbol{C(T)}$ [4, 6]. Besides, an algorithm with a smaller N_S generally has poorer scheduling efficiency, and requires a larger S_{schedule}.

Among the existing scheduling algorithms, MIN [4] and QLEF [6] use the minimum number of $N_S=N$ configurations for scheduling, but the corresponding S_{schedule} is very large. On the other hand, EXACT [4] uses $N_S=N^2-2N+2$ configurations to get $S_{\text{schedule}}=1$, but the packet delay is very large due to the large value of N_S. DOUBLE [4] makes an efficient tradeoff by using $N_S=2N$ to achieve $S_{\text{schedule}}=2$. Note that those algorithms can only produce schedules with one of the three N_S values, N, N^2-2N+2 or $2N$. Recently, ADAPTIVE [5] is proposed to allow any integer value of N_S in (N, N^2-2N+2). It is also shown in [5] that DOUBLE is a special case of ADAPTIVE at $N_S=2N$.

In this paper, we show that the schedules returned by ADAPTIVE can be further optimized. A new scheduling algorithm SRF (Scheduling Residue First) is proposed to improve the scheduling efficiency (i.e. to reduce S_{schedule}). In Section II, we review the two closely related algorithms DOUBLE [4] and ADAPTIVE [5], based on which our SRF is designed in Section III. Performance analysis and discussions are given in Section IV. We conclude the paper in Section V.

II. DOUBLE AND ADAPTIVE ALGORITHMS

We divide $\boldsymbol{C(T)}$ by $T/(N_S-N)$ to get a quotient matrix $\boldsymbol{Q}=\{q_{ij}\}$ and a residue matrix $\boldsymbol{R}=\{r_{ij}\}$ as follows. If $T/(N_S-N)$ is not an integer, use $\lceil T/(N_S-N)\rceil$ instead [5].

$$\boldsymbol{C(T)} = \frac{T}{N_S-N}\times\boldsymbol{Q}+\boldsymbol{R} \qquad (5)$$

Since the entries in each line of $\boldsymbol{C(T)}$ sum to at most T, the maximum line sum of $\boldsymbol{Q}$ is N_S-N. Therefore, we can apply edge-coloring [7] to the bipartite multigraph of $\boldsymbol{Q}$ and get N_S-N colors/configurations to cover $\boldsymbol{Q}$ [5]. On the other hand, each entry in $\boldsymbol{R}$ is smaller than $T/(N_S-N)$. So, $\boldsymbol{R}$ can be covered by *any* N non-overlapping configurations (i.e. any two of them do not cover the same entry), with each weighted by $T/(N_S-N)$. As

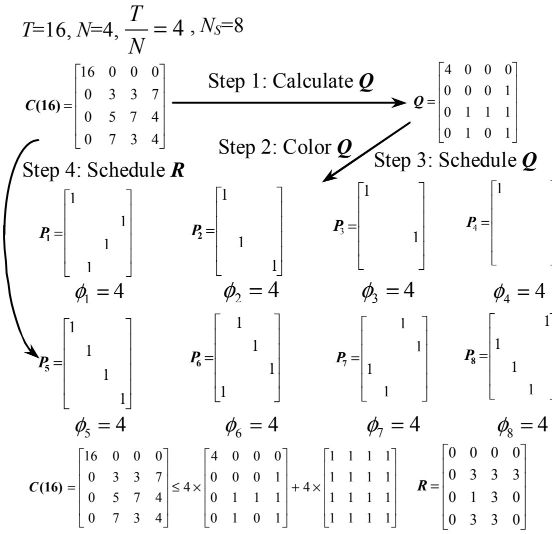

Fig. 3. An example of DOUBLE execution. The all-1 matrix used to cover $\boldsymbol{R}$ equals to the sum of the N non-overlapping configurations ($\boldsymbol{P}_5$-$\boldsymbol{P}_8$).

a result, $\boldsymbol{C(T)}$ can be covered by $(N_S-N)+N=N_S$ configurations, each equally weighted by $\phi_n=T/(N_S-N)$. From (4), we have

$$S_{\text{schedule}} = \frac{1}{T}\sum_{n=1}^{N_S}\phi_n = \frac{1}{T}\times\frac{T}{N_S-N}\times N_S = 1+\frac{N}{N_S-N}. \qquad (6)$$

Replacing N_S by $2N$ in (6), we get $S_{\text{schedule}}=2$ for DOUBLE [4]. Each of the matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ in DOUBLE is covered by N configurations, with an equal weight $\phi_n=T/N$ for each configuration. Fig. 3 gives an example of DOUBLE execution.

Unlike DOUBLE, ADAPTIVE substitutes (6) into (2), and finds a proper N_S to minimize the overall speedup S by solving

$$\frac{\partial S}{\partial N_S} = 0. \qquad (7)$$

III. SRF ALGORITHM

Since DOUBLE is a special case of ADAPTIVE at $N_S=2N$, for simplicity, we first design SRF based on DOUBLE.

A. Observation and Motivation

From (5), we get $\boldsymbol{C(T)}=[T/N]\times\boldsymbol{Q}+\boldsymbol{R}$ with DOUBLE. For any $r_{ij}\in\boldsymbol{R}$, we have $r_{ij}<T/N$. If $r_{ij}>T/(2N)$, we call it an *LER* (large entry in $\boldsymbol{R}$). Otherwise it is an *SER* (small entry in $\boldsymbol{R}$). We have the following Lemma 1 (proved in the Appendix).

Lemma 1: In DOUBLE, if a line (row i or column j) in $\boldsymbol{R}$ contains k LERs, then in $\boldsymbol{Q}$ we have

$$\sum_{j=1}^{N} q_{ij} \le N-\left\lceil\frac{k}{2}\right\rceil \text{ for row } i, \text{ or } \sum_{i=1}^{N} q_{ij} \le N-\left\lceil\frac{k}{2}\right\rceil \text{ for column } j.$$

Based on Lemma 1, we can *move some packets* from $\boldsymbol{R}$ to $\boldsymbol{Q}$, while still keeping the maximum line sum of $\boldsymbol{Q}$ not more than N. Note that all the weights in DOUBLE are equal and $\phi_n=T/N$. So, if a line in $\boldsymbol{R}$ contains k LERs, we can move half (i.e. $\lceil k/2\rceil$) of them to $\boldsymbol{Q}$ by setting them to 0 in $\boldsymbol{R}$, and at the same time increasing the corresponding entries in $\boldsymbol{Q}$ by one. Fig. 4 gives an example based on the $\boldsymbol{Q}$ and $\boldsymbol{R}$ in Fig. 3. We use $\boldsymbol{Q'}$ and $\boldsymbol{R'}$ to denote the updated $\boldsymbol{Q}$ and $\boldsymbol{R}$. Because the maximum

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.

$$\boldsymbol{C}(16)=4\times\underbrace{\begin{bmatrix}4&0&0&0\\0&0&0&1\\0&1&1&1\\0&1&0&1\end{bmatrix}}_{\boldsymbol{Q}}+\underbrace{\begin{bmatrix}0&0&0&0\\0&3&3&3\\0&1&3&0\\0&3&3&0\end{bmatrix}}_{\boldsymbol{R}}\longrightarrow \boldsymbol{C}(16)\le 4\times\underbrace{\begin{bmatrix}4&0&0&0\\0&1&0&2\\0&1&2&1\\0&1&1&1\end{bmatrix}}_{\boldsymbol{Q}'}+\underbrace{\begin{bmatrix}0&0&0&0\\0&0&3&0\\0&1&0&0\\0&3&0&0\end{bmatrix}}_{\boldsymbol{R}'}$$

Fig. 4. Move the circled LERs from $\boldsymbol{R}$ to $\boldsymbol{Q}$.

line sum of $\boldsymbol{Q}'$ is at most N, $\boldsymbol{Q}'$ can still be covered by N configurations, each with the same original weight $\phi_n=T/N$. It means that we can schedule/send more packets in the N configurations for the quotient matrix (than DOUBLE).

Note that each line of $\boldsymbol{R}$ contains at most N LERs. If half of them are moved to $\boldsymbol{Q}$ (without increasing its maximum line sum), then each line of $\boldsymbol{R}'$ contains at most $N/2$ LERs. (Assume N is even.) Although we still need N non-overlapping configurations to cover $\boldsymbol{R}'$, it is possible to reduce some of their weights. Specifically, we may find $N/2$ configurations, each weighted by $\phi_n=T/N$, to cover all the remaining LERs in $\boldsymbol{R}'$. At the same time, we may find another $N/2$ configurations, each with a *reduced* weight of $\phi_n=T/(2N)$, to cover the remaining SERs. Then, S_{schedule} of DOUBLE can be reduced to

$$S_{\text{schedule}}=\frac{1}{T}\sum_{n=1}^{2N}\phi_n=\frac{1}{T}\times\left(\frac{T}{N}\times N+\frac{T}{N}\times\frac{N}{2}+\frac{T}{2N}\times\frac{N}{2}\right)=1.75 \qquad (8)$$

The above observation motivates us to explore a more efficient scheduling algorithm than DOUBLE. The key issue is to find a proper residue matrix $\boldsymbol{R}'$, such that $\boldsymbol{R}'$ contains at most $N/2$ LERs in *each* line, and *each* line sum of $\boldsymbol{Q}'$ is not greater than N. Generally, this is not easy, as illustrated in Fig. 5a, where we assume all the 3*s* are LERs. The number next to each line of $\boldsymbol{R}$ indicates the number of LERs (i.e. quota) that can be moved to $\boldsymbol{Q}$. Its value is equal to half of the number of LERs in each line of $\boldsymbol{R}$, or $\lceil k/2\rceil$. Without violating the quota, we may move the four circled LERs to $\boldsymbol{Q}$. If so, the last row of $\boldsymbol{R}'$ will contain 3 LERs (Note that 3>N/2=2). This makes it impossible to cover the remaining LERs by $N/2$ configurations.

B. Residue Matrix Scheduling

We now focus on determining a proper residue matrix $\boldsymbol{R}'$ and covering it by N non-overlapping configurations. We define a *reference matrix* $\boldsymbol{F}=\{f_{ij}\}$, where $f_{ij}=1$ if r_{ij} is an LER in the original $\boldsymbol{R}$ and $f_{ij}=0$ otherwise, as illustrated in Fig. 5b. $\boldsymbol{F}$ is partitioned into four $(N/2)\times(N/2)$ sub-matrices $\boldsymbol{Z}_1\sim\boldsymbol{Z}_4$ by two orthogonal lines as shown in Fig. 5c (where N=8 is assumed). A set of $N/2$ *predefined partial configurations* (PPCs) are defined in $\boldsymbol{Z}_1$ in a cyclic manner according to (9) below.

$$p=\left[\frac{N}{2}-(i-j)\right]\bmod\left(\frac{N}{2}\right)+1,\quad N/2>i\ge 0,\quad N/2>j\ge 0. \qquad (9)$$

Particularly, entry (i, j) is covered by PPC$_p$. In Fig. 5c, the number at each entry in $\boldsymbol{Z}_1$ gives the p value for that entry.[1] For example, the four circled entries are covered by PPC$_3$. Note that the PPCs are mutually non-overlapping, and each PPC covers $N/2$ entries in $\boldsymbol{Z}_1$. For an arbitrary entry f_{ij} in $\boldsymbol{Z}_1$, we define its *line images* and *diagonal image* as shown in Fig. 5d. Particularly, $f_{i(N-j-1)}$ and $f_{(N-i-1)j}$ are line images of f_{ij} and they are

[1] For simplicity, we reuse Fig. 5c as an example to show the entries covered by PPC$_p$. Generally, f_{ij} in $\boldsymbol{Z}_1$ takes the value of 1 or 0 (instead of p as in Fig. 5c) to indicate whether the corresponding r_{ij} is an LER or an SER.

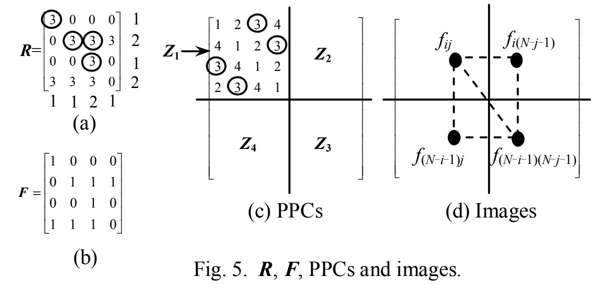

Fig. 5. $\boldsymbol{R}$, $\boldsymbol{F}$, PPCs and images.

symmetrical to f_{ij} with respect to the two matrix partitioning lines; $f_{(N-i-1)(N-j-1)}$ is the diagonal image and it is symmetrical to f_{ij} with respect to the cross-point of the two partitioning lines.

Without loss of generality, we consider PPC$_p$. For *each* entry f_{ij} covered by PPC$_p$, we find its line and diagonal images. The 4-tuple $\{f_{ij}, f_{i(N-j-1)}, f_{(N-i-1)j}, f_{(N-i-1)(N-j-1)}\}$ may have 16 possible values as shown in Fig. 6. The tuples in Figs. 6a～6l are said to be *diagonal dominant* (DD), and the two circled entries are defined as *dominant entries*. Each of the two dominant entries in a DD tuple always contains dominant (or equal) number of 1*s* than *both* of its line images. On the contrary, the non-DD tuples in Figs. 6m～6p do not have such a property. Each non-DD tuple in Figs. 6m～6n is called a *column isomorphic tuple* because the two columns are exactly the same. Similarly, the non-DD tuples in Figs. 6o～6p are *row isomorphic tuples*.

To cover the residue matrix in N configurations $\boldsymbol{P}_1\sim\boldsymbol{P}_N$, we first initialize them to all-0*s*. Based on reference matrix $\boldsymbol{F}$, each PPC$_p$ is sequentially checked to construct two configurations $\boldsymbol{P}_p$ and $\boldsymbol{P}_{p+N/2}$. Particularly, for each f_{ij} covered by PPC$_p$, we find the 4-tuple $\{f_{ij}, f_{i(N-j-1)}, f_{(N-i-1)j}, f_{(N-i-1)(N-j-1)}\}$. Then, the entries in $\boldsymbol{P}_p$ and $\boldsymbol{P}_{p+N/2}$ are set according to different cases discussed below.

Case 1: If the 4-tuple is a DD tuple, we set the two dominant entries to "1" in $\boldsymbol{P}_p$, and at the same time set the other two non-dominant entries to "1" in $\boldsymbol{P}_{p+N/2}$.

Case 2: If the 4-tuple is a row isomorphic tuple, we check whether another row isomorphic tuple *most recently occurred in the same row pair*. If no, we can set either pair of the diagonal entries (say, the solid-circled entries in Figs. 6o & 6p) to "1" in $\boldsymbol{P}_p$. The other two entries (dash-circled) are set to "1" in $\boldsymbol{P}_{p+N/2}$. On the other hand, if such a row isomorphic tuple occurred in examining a previous PPC$_t$ (where $p>t$), we set the corresponding entries in $\boldsymbol{P}_p$ and $\boldsymbol{P}_{p+N/2}$ according to a *butterfly mechanism*, such that the 1*s* in the two rows of $\boldsymbol{F}$ can be covered by $\boldsymbol{P}_p$ and $\boldsymbol{P}_t$ in an alternative manner. Specifically, assume the two row isomorphic tuples occur in row pair $\{i, N-i-1\}$. If $\boldsymbol{P}_t$ was set to cover a "1" in row i and a "0" in row $N-i-1$ (in $\boldsymbol{F}$), then $\boldsymbol{P}_p$ is set to cover a "1" in row $N-i-1$ and a "0" in row i. Accordingly, the other two entries are set to "1" in $\boldsymbol{P}_{p+N/2}$.

Case 3: If the 4-tuple is a column isomorphic tuple, the same mechanism as in Case 2 is used but for the column pair.

Fig. 7. gives an example. For simplicity, we only discuss the construction of $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$. In examining PPC$_1$, $\{f_{00}, f_{33}\}$ are first identified as dominant entries in the 4-tuple $\{f_{00}, f_{03}, f_{30}, f_{33}\}$, and thus entries (0, 0), (3, 3) are set to "1" in $\boldsymbol{P}_1$. Then, we can see that $\{f_{11}, f_{12}, f_{21}, f_{22}\}$ is a row isomorphic tuple. Since no other isomorphic tuples precede it, we simply set the two solid-circled entries (1, 1), (2, 2) to "1" in $\boldsymbol{P}_1$. In examining PPC$_2$,

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.

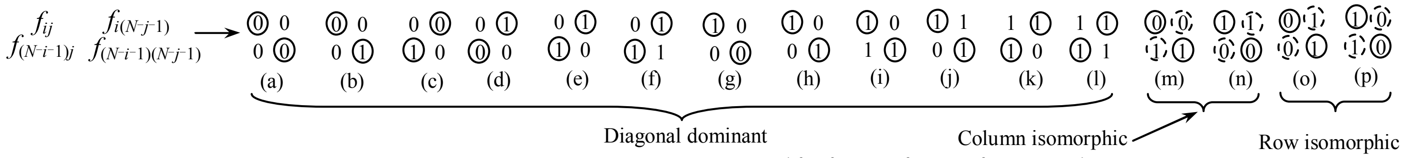

Fig. 6. Sixteen possible combinations of $\{f_{ij}, f_{i(N-j-1)}, f_{(N-i-1)j}, f_{(N-i-1)(N-j-1)}\}$.

$\{f_{01}, f_{02}, f_{31}, f_{32}\}$ is a row isomorphic tuple but it resides in a different row pair as $\{f_{11}, f_{12}, f_{21}, f_{22}\}$ (which occurred in examining PPC$_1$). So, we can use either diagonal to set $\boldsymbol{P_2}$. In Fig. 7, $\{f_{01}, f_{32}\}$ are used and entries (0, 1), (3, 2) are set to "1" in $\boldsymbol{P_2}$. After that, the row isomorphic tuple $\{f_{10}, f_{13}, f_{20}, f_{23}\}$ is checked. Because another row isomorphic tuple $\{f_{11}, f_{12}, f_{21}, f_{22}\}$ precedes it in the same row pair, we need to set $\boldsymbol{P_2}$ according to the butterfly mechanism. Specifically, the two dashed-circled entries (1, 0) and (2, 3) are set to "1" in $\boldsymbol{P_2}$.

Obviously, the above process generates N non-overlapping configurations $\boldsymbol{P_1} \sim \boldsymbol{P_N}$ to cover every entry of the residue matrix. This is because the PPCs do not overlap each other. On the other hand, $\boldsymbol{P_1} \sim \boldsymbol{P_{N/2}}$ usually cover more than half of the 1*s* for *each and all* of the lines in $\boldsymbol{F}$. This is because the dominant entries in each DD tuple are always covered in $\boldsymbol{P_1} \sim \boldsymbol{P_{N/2}}$, whereas the number of 1*s* covered by $\boldsymbol{P_1} \sim \boldsymbol{P_{N/2}}$ and $\boldsymbol{P_{N/2+1}} \sim \boldsymbol{P_N}$ in each line of $\boldsymbol{F}$ are well balanced for the non-DD tuples. The only exception is that, for a particular line, the number of 1*s* covered by $\boldsymbol{P_1} \sim \boldsymbol{P_{N/2}}$ (denoted by x) may be *one less* than that covered by $\boldsymbol{P_{N/2+1}} \sim \boldsymbol{P_N}$ (denoted by y). This is due to the butterfly mechanism. Assume row pair $\{i, N-i-1\}$ is considered and there are z row isomorphic tuples in this row pair, and we only consider them in counting x and y. If z is even, then x and y can be perfectly balanced according to the butterfly mechanism, and thus $x=y$. If z is odd, then $|x-y|=1$. In any case, we always have either $x \geq y$ or $x+1=y$ for each and all lines in $\boldsymbol{F}$.

Consequently, if some 1*s* in $\boldsymbol{F}$ are covered by $\boldsymbol{P_{N/2+1}} \sim \boldsymbol{P_N}$, we can move the corresponding LERs from $\boldsymbol{R}$ to $\boldsymbol{Q}$. This gives us $\boldsymbol{R'}$ and $\boldsymbol{Q'}$. From Lemma 1, the maximum line sum of $\boldsymbol{Q'}$ will not exceed N. This is also true if $x+1=y$ due to the roof function in Lemma 1 (Note that k in Lemma 1 equals to $x+y$). As a result, all the remaining LERs in $\boldsymbol{R'}$ can be covered by $\boldsymbol{P_1} \sim \boldsymbol{P_{N/2}}$ with a weight $\phi_n=T/N$, and other entries in $\boldsymbol{R'}$ can be covered by $\boldsymbol{P_{N/2+1}} \sim \boldsymbol{P_N}$ with a reduced weight of $\phi_n=T/(2N)$.

In short, with the residue matrix scheduling mechanism, S_{schedule} of DOUBLE can be reduced from 2 to 1.75 as in (8).

C. SRF Algorithm

Note that DOUBLE is a special case at $N_S=2N$. Generally, we can use $T/(N_S-N)$ (instead of T/N) to divide $\boldsymbol{C(T)}$ as in (5), and define any $r_{ij} \in \boldsymbol{R}$ as an LER if $r_{ij} > T/[2(N_S-N)]$ (instead of $r_{ij} > T/(2N)$). With the same mechanism, we can construct $\boldsymbol{R'}$ and $\boldsymbol{Q'}$ while keeping the maximum line sum of $\boldsymbol{Q'}$ not more than N_S-N. Then, we find N non-overlapping configurations to cover $\boldsymbol{R'}$, with a weight $T/(N_S-N)$ for half of them and a reduced weight $T/[2(N_S-N)]$ for the other half. At the same time, $\boldsymbol{Q'}$ is covered by N_S-N configurations with a weight $T/(N_S-N)$ for each. Consequently, S_{schedule} in (6) can be reduced to

$$S_{\text{schedule}} = \frac{1}{T}\sum_{n=1}^{N_S}\phi_n = 1 + \frac{3N}{4(N_S-N)}. \tag{10}$$

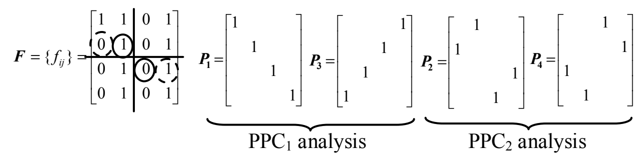

Fig. 7. Residue matrix scheduling based on $\boldsymbol{F}$.

SRF ALGORITHM

Input:

δ and an $N\times N$ matrix $\boldsymbol{C(T)}$ with maximum line sum not more than T.

Output:

N configurations $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_{N_S}$ and weights $\phi_1, \ldots, \phi_{N_S}$.

Step 1: Divide $\boldsymbol{C(T)}$:

Calculate N_S by (11). Use $\lceil T/(N_S-N)\rceil$ to divide $\boldsymbol{C(T)}$ and separate it into $\boldsymbol{Q}=\{q_{ij}\}$ and $\boldsymbol{R}=\{r_{ij}\}$ as in (5).

Step 2: Schedule the residue matrix:

a) Define a reference matrix $\boldsymbol{F}=\{f_{ij}\}$ based on $\boldsymbol{R}$, where $f_{ij}=1$ if r_{ij} is an LER ($>\lceil T/\lceil 2(N_S-N)\rceil\rceil$) and $f_{ij}=0$ otherwise. Define $N/2$ PPCs based on (9). Initialize $\boldsymbol{P_1} \sim \boldsymbol{P_N}$ to all-0 matrices. Set $1 \rightarrow p$.

b) Pick up an entry f_{ij} covered by PPC$_p$, find its images to form a 4-tuple $\{f_{ij}, f_{i(N-j-1)}, f_{(N-i-1)j}, f_{(N-i-1)(N-j-1)}\}$. If it is a DD tuple, set the two dominant entries to "1" in $\boldsymbol{P_p}$, and the other two non-dominant entries to "1" in $\boldsymbol{P_{p+N/2}}$. If it is an isomorphic tuple, check whether another isomorphic tuple most recently occurred in the same line pair. If no, set any pair of the diagonal entries of the tuple to "1" in $\boldsymbol{P_p}$ and the other two diagonal entries to "1" in $\boldsymbol{P_{p+N/2}}$. Otherwise, if such a preceding isomorphic tuple occurred in examining PPC$_t$ ($p>t$), then invoke the butterfly mechanism to set the entries in $\boldsymbol{P_p}$, such that the number of 1*s* in this line pair of $\boldsymbol{F}$ can be covered by $\boldsymbol{P_p}$ and $\boldsymbol{P_t}$ alternatively. If a pair of diagonal entries are set to "1" in $\boldsymbol{P_p}$, then set the other pair of diagonal entries to "1" in $\boldsymbol{P_{p+N/2}}$. Repeat *Step 2b)* for each f_{ij} covered by PPC$_p$, until the two configurations $\boldsymbol{P_p}$ and $\boldsymbol{P_{p+N/2}}$ are obtained.

c) Set $p+1 \rightarrow p$. Repeat *Step 2b)* until $p=N/2+1$.

d) Set $\phi_1 \sim \phi_{N/2}$ to $\lceil T/(N_S-N)\rceil$ and $\phi_{N/2+1} \sim \phi_N$ to $\left\lceil T/\lceil 2\times(N_S-N)\rceil \right\rceil$. If some 1*s* in $\boldsymbol{F}$ are covered by $\boldsymbol{P_{N/2+1}} \sim \boldsymbol{P_N}$, increase the corresponding entries in $\boldsymbol{Q}$ by one. Denote the updated $\boldsymbol{Q}$ by $\boldsymbol{Q'}$.

Step 3: Schedule the quotient matrix:

a) Construct a bipartite multigraph $\boldsymbol{G}$ from $\boldsymbol{Q'}$. Find a minimal edge-coloring of $\boldsymbol{G}$ to get at most N_S-N colors. Set $N+1 \rightarrow n$.

b) For a specific color in the edge-coloring of $\boldsymbol{G}$, construct a configuration $\boldsymbol{P_n}$ from the edges assigned to that color. Set $\phi_n = \lceil T/(N_S-N)\rceil$ and $n+1 \rightarrow n$. Repeat *step 3b)* for each of the colors in $\boldsymbol{G}$.

Fig. 8. SRF algorithm (for even switch size N).

In SRF (see Fig. 8), we replace S_{schedule} in (2) by (10), and solve (7) to get the proper N_S value, as given in (11).

$$\begin{cases} N_S^{real} = \dfrac{1}{4}\left(N + \sqrt{\dfrac{12NT}{\delta} - 3N^2}\right) \\ N_S = \lfloor N_S^{real} \rfloor \quad if \quad N^2 - 2N + 2 > N_S^{real} \geq N+1\,. \\ N_S = N+1 \quad if \quad N+1 > N_S^{real} > N \end{cases} \tag{11}$$

Compared to ADAPTIVE [5] which has a time complexity of $O(N^{1.5}\log N)$, SRF needs an extra $O(N^2)$ computation for residue matrix scheduling. So the time complexity of SRF is $O(N^{1.5}\log N + N^2)$, or $O(N^2)$.

IV. Performance Analysis and Discussion

S_{schedule} in (6) for ADAPTIVE and in (10) for SRF are compared in Fig. 9 for N=64. We did not directly compare the overall speedup S because it involves other parameters such as T and δ. From Fig. 9, again we can see that DOUBLE [4] is a special case of ADATIVE [5]. DOUBLE generates S_{schedule}=2 at N_S=128, but we can get S_{schedule}=1.75 with SRF. This gives a cut of 12.5% on S_{schedule}. On the other hand, with S_{schedule}=2, SRF only requires N_S=112 configurations to schedule $\boldsymbol{C(T)}$.

A simple example is given in Fig. 10 to compare the three algorithms. Although DOUBLE produces the smallest S_{schedule} (but with the largest N_S), it has the largest overall speedup of S=14, whereas ADAPTIVE has S=8.4 and SRF has S=7.

Note that SRF does not allow N_S=N. But this case can be efficiently handled by the minimum delay scheduling algorithm QLEF [6]. Also note that predefining PPCs in a cyclic-manner using (9) is not necessary. We did so only to facilitate our presentation. In fact, any set of N/2 non-overlapping permutation sub-matrices in $\boldsymbol{Z_1}$ can be used as PPCs.

V. Conclusion

We proposed a new algorithm SRF (Scheduling Residue First) to improve the scheduling efficiency for high-speed routers with optical switch fabrics. SRF significantly reduces the speedup factor S_{schedule}. Compared with DOUBLE, a 12.5% cut on S_{schedule} is achieved. Consequently, SRF outperforms the existing DOUBLE and ADAPTIVE algorithms by requiring a lower overall speedup for performance guaranteed switching.

Appendix

Lemma 1: In DOUBLE, if a line (row i or column j) in $\boldsymbol{R}$ contains k LERs, then in $\boldsymbol{Q}$ we have

$$\sum_{j=1}^{N} q_{ij} \le N - \left\lceil \frac{k}{2} \right\rceil \text{ for row } i, \text{ or } \sum_{i=1}^{N} q_{ij} \le N - \left\lceil \frac{k}{2} \right\rceil \text{ for column } j.$$

Proof: After $\boldsymbol{C(T)}$ is divided by T/N, we have

$$\boldsymbol{C(T)} = \frac{T}{N}\boldsymbol{Q} + \boldsymbol{R} \quad or \quad c_{ij} = \frac{T}{N} q_{ij} + r_{ij} \ ,$$

Assume row i of $\boldsymbol{R}$ contains k LERs. Because

$$\sum_{j=1}^{N} c_{ij} = \frac{T}{N}\sum_{j=1}^{N} q_{ij} + \sum_{j=1}^{N} r_{ij} \le T \ ,$$

we have

$$\sum_{j=1}^{N} q_{ij} \le \frac{T - \sum_{j=1}^{N} r_{ij}}{\frac{T}{N}} < \frac{T - \frac{T}{2N}\times k}{\frac{T}{N}} = N - \frac{k}{2} \ .$$

Since $\sum_{j=1}^{N} q_{ij}$ is an integer, we then have

$$\sum_{j=1}^{N} q_{ij} \le N - \left\lceil \frac{k}{2} \right\rceil .$$

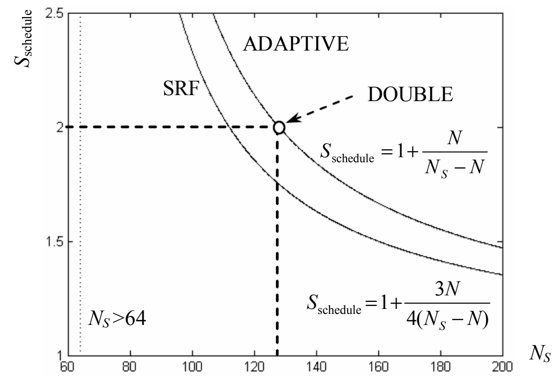

Fig. 9. S_{schedule} for N=64.

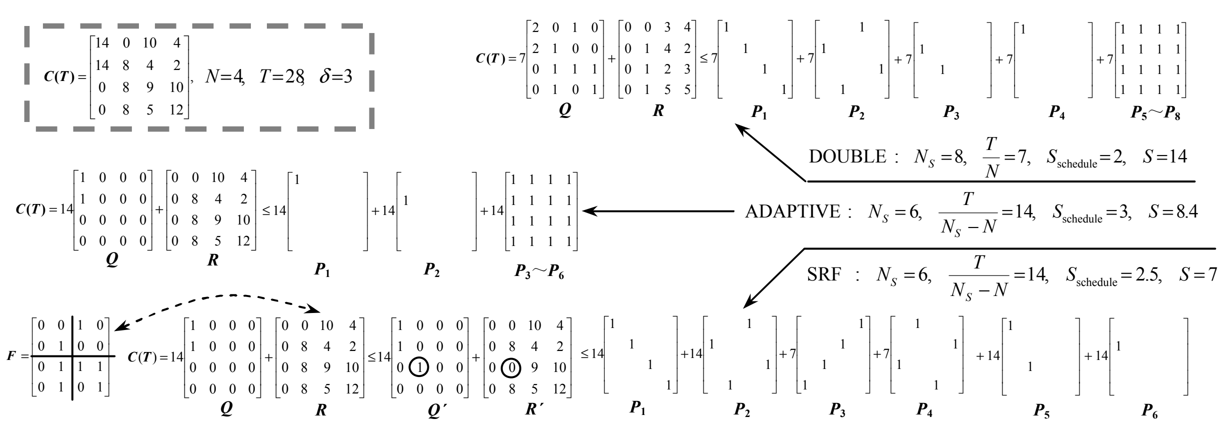

Fig. 10. An example of DOUBLE, ADAPTIVE and SRF.

References

[1] O. B. Spahn, C. Sullivan, J. Burkhart, C. Tigges, and E. Garcia, “GaAs-based microelectromechanical waveguide switch”, *Proc. 2000 IEEE/LEOS Intl. Conf. on Optical MEMS*, pp. 41-42, Aug. 2000.

[2] A. J. Agranat, “Electroholographic wavelength selective crossconnect”, *1999 Digest of the LEOS Summer Topical Meetings*, pp. 61-62, Jul. 1999.

[3] K. Kar, D. Stiliadis, T. V. Lakshman and L. Tassiulas, “Scheduling algorithms for optical packet fabrics”, *IEEE Journal on Selected Areas in Communications*, vol. 21, issue 7, pp. 1143-1155, Sept. 2003.

[4] B. Towles and W. J. Dally, “Guaranteed scheduling for switches with configuration overhead”, *IEEE/ACM Trans. Networking*, vol. 11, no. 5, pp. 835-847, Oct. 2003.

[5] Bin Wu and Kwan L. Yeung, “Minimizing internal speedup for performance guaranteed optical packet switches”, *IEEE GLOBECOM '04*, vol. 3, pp. 1742-1746, Dec. 2004.

[6] Bin Wu and Kwan L. Yeung, “Traffic scheduling in non-blocking optical packet switches with minimum delay”, *IEEE GLOBECOM '05*, vol. 4, pp. 2041-2045, Dec. 2005.

[7] R. Cole and J. Hopcroft, “On edge coloring bipartite graphs”, *SIAM Journal on Computing*, vol. 11, pp. 540-546, Aug. 1982.