Juggling Networks

Nicholas Pippenger*
e-mail: nicholas@cs.ubc.ca

Department of Computer Science
The University of British Columbia
Vancouver, British Columbia V6T 1Z4
CANADA

Abstract: Switching networks of various kinds have come to occupy a prominent position in computer science as well as communication engineering. The classical switching network technology has been space-division-multiplex switching, in which each switching function is performed by a spatially separate switching component (such as a crossbar switch). A recent trend in switching network technology has been the advent of time-division-multiplex switching, wherein a single switching component performs the function of many switches at successive moments of time according to a periodic schedule. This technology has the advantage that nearly all of the cost of the network is in inertial memory (such as delay lines), with the cost of switching elements growing much more slowly as a function of the capacity of the network.

In order for a classical space-division-multiplex network to be adaptable to time-division-multiplex technology, its interconnection pattern must satisfy stringent requirements. For example, networks based on randomized interconnections (an important tool in determining the asymptotic complexity of optimal networks) are not suitable for time-division-multiplex implementation. Indeed, time-division-multiplex implementations have been presented for only a few of the simplest classical space-division-multiplex constructions, such as rearrangeable connection networks.

This paper shows how interconnection patterns based on explicit constructions for expanding graphs can be implemented in time-division-multiplex networks. This provides time-division-multiplex implementations for switching networks that are within constant factors of optimal in memory cost, and that have asymptotically more slowly growing switching costs. These constructions are based on a metaphor involving teams of jugglers whose throwing, catching and passing patterns result in intricate permutations of the balls. This metaphor affords a convenient visualization of time-division-multiplex activities that should be of value in devising networks for a variety of switching tasks.

* This research was partially supported by an NSERC Operating Grant.

1. Introduction

In this paper we will present a metaphor for describing the construction and operation of time-division-multiplex networks, and use it to present a new time-division-multiplex implementation of an explicit construction for expanding graphs, which are an essential component in many constructions for switching networks. Both the new metaphor and the main techniques for construction of time-division-multiplex networks will be illustrated in Section 2 by a well known construction for rearrangeable connection networks. This construction was described in the context of space-division-multiplex networks by Beneš [6] in 1964. The time-division-multiplex implementation was first described by Marcus [13] in 1970, and has recently been rediscovered by Ramanan, Jordan and Sauer [23]. The resulting implementation is "time-slot interchanger" in the sense of Inose [9]. In Section 3 we indicate how these methods can be adapted to other types of switching networks. The main obstacle for such applications is the requirement for "expanding graphs" (and related objects) presented by many constructions for switching networks. In Section 4 we present a time-division-multiplex implementation of a well known construction for expanding graphs (and, more generally, for graphs with a prescribed "eigenvalue separation ratio"). This construction was first proposed by Margulis [14] in 1974. Quantitative estimates essential for its application were provided by Gabber and Galil [8] in 1981, and improvements to these estimates have been given by Jimbo and Maruoka [10], whose version of the space-division-multiplex construction we follow. In Section 5 we present some open problems prompted by this work.

2. Connectors

Imagine a *juggler* who can with complete reliability throw balls to a fixed height, so that they always return a fixed amount of time after they are thrown. All amounts of time considered in this paper will be multiples of some fixed unit of time that will be called the *pulse*. Suppose that our juggler can take a ball at each pulse from an external agent, the juggler's *source*, and can give a ball at each pulse to another external agent, the juggler's *sink*. Suppose further that at each pulse the juggler can execute either of two *moves*, which will be called the *straight* and *crossed* moves. In the straight move, the juggler rethrows the ball that returns from the air (if any such ball returns), and gives the ball taken from the source to the sink (if any such ball is taken). In the crossed move, the juggler throws the ball taken from the source (if any is taken), and gives the ball that returns from the air to the sink (if any returns).

Now imagine a *chain* of jugglers; that is, a finite sequence of jugglers $J_1, \ldots, J_\mu$ in which J_λ is the source of $J_{\lambda+1}$, and $J_{\lambda+1}$ is the sink of J_λ, for $1 \le \lambda < \mu$. (The source of J_1 and the sink of J_μ are external to the chain. They will be called the *source* and *sink* of the chain.) We assume that the jugglers may have different "spans" (where the *span* of a juggler is the amount of time between the throw of a ball and its return), but that all of these are multiples of a common pulse. Depending on the spans of the various jugglers, and on the sequence of straight and crossed moves executed by each juggler, the sequence of balls passed by the source of the chain (and empty pulses during which no ball is passed) will be rearranged in some way before being passed to the sink of the chain.

In what follows, we shall regard the span of each juggler as a fixed and unchanging attribute of the juggler, while we regard the sequence of moves as being variable. How does each juggler decide what sequence of moves to execute? Our assumption will be that each juggler has a partner, called the juggler's *cox*, who calls out the name, "straight" or "crossed", of the move to execute at each pulse. How does the cox decide what sequence of moves to call? Our assumption will be that each cox is also a juggler who juggles a fixed sequence of balls. A cox has no source or sink, and always executes straight moves, rethrowing each ball as it returns from the air. We shall assume that a ball returns at each pulse (there are no empty pulses), so that the number of balls being juggled by the cox is equal to the cox's span (which may be different from the cox's partner's span). Finally, we shall assume that each ball juggled by the cox has one of two *colors*, say *red* for "straight" and *blue* for "crossed", and that the cox calls out the move corresponding to the color of each ball as it is rethrown. Thus each cox calls for a periodic sequence of moves, corresponding to the cyclic sequence of colors of balls in the cox's pattern, with a period that is equal to the cox's span.

We can now give a simple example showing how a coxed chain of jugglers can serve as a model for a time-division-multiplex rearrangeable connection network. Let $n = 2^\nu$ be an integral power of 2. Consider a chain of $2\nu - 1$ jugglers $J_1, \ldots, J_{2\nu-1}$. Suppose that jugglers $J_1, \ldots, J_\nu$ have spans $2^0 = 1, \ldots, 2^{\nu-1} = n/2$, respectively, and that jugglers $J_{\nu+1}, \ldots, J_{2\nu-1}$ have spans $2^{\nu-2} = n/4, \ldots, 2^0 = 1$, respectively. Suppose further that all $2\nu - 1$ coxes have span $2^\nu = n$.

Suppose that the source of the chain just described passes it a sequence of balls at successive pulses. Let us divide the pulses into a sequence of *frames*, with each frame comprising n successive pulses. The sequence of balls passed by the source to the chain may be broken into frames, with each frame of balls comprising the balls passed to the chain during a frame of pulses. The sequence of balls passed by the chain to its sink may be broken into frames in a similar way. Furthermore, we may establish a correspondence between source frames and sink frames in the following way. Imagine that each juggler in the chain executes only crossed moves, so that the stream of balls from the source is passed on to the sink after a fixed delay, equal to the sum of the spans of the jugglers in the chain (which is in this case $3n/2 - 2$). Thus each source frame corresponds to a sink frame that is the series of n pulses during which the balls of the source frame emerge from the chain in this situation. The positions of the n balls within their frame will be called *slots*. We shall index the slots of each frame from $1, \ldots, n$ (slot 1 is the earliest, and slot n the latest, slot of its frame).

Theorem 0: For every permutation $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$, there exist patterns for each cox that cause each ball that is passed by the source to the chain in slot i of a frame to be passed by the chain to its sink in slot $\pi(i)$ of the corresponding frame.

The proof of this theorem, which is implicit in the work of Marcus [13] in 1970, is based on the construction of Beneš for rearrangeable connection networks. This space-division-multiplex construction employs $(2\nu - 1)2^{\nu-1}$ switching elements (2×2 crossbar switches), arranged in $2\nu - 1$ *stages*, with each stage comprising $2^{\nu-1}$ crossbars. In the time-division-multiplex implementation of this construction, each of the $2\nu - 1$ jugglers in the chain will simulate the $2^{\nu-1}$ crossbars of the corresponding stage.

The space-division-multiplex construction is usually described recursively. In the drawing resulting from this description, crossbars are depicted as “boxes” and the wires interconnecting them are depicted as “lines” that follow “perfect shuffle” interconnection patterns. It is possible to redraw the this picture, however, so that the wires that carry the signals from the inputs to the outputs remain parallel to each other, with the crossbars of each stage conditionally exchanging the signals on wires separated by a fixed distance (depending upon the stage). This can in fact be done so that the distance in each stage is just the span that we have assigned to the corresponding juggler.

When this redrawing has been done, we see that the task of a juggler for each pair of slots (separated by the span of the juggler) is either to leave them unaffected, or to exchange the balls in these two slots. In the latter case, we need to “delay” the contents of the earlier slot by a number of pulses equal to the span, and to “advance” the contents of the later slot by the same amount. Since we cannot implement negative delays, we add a constant delay, equal to the span, to all slots of the frame. With this adjustment, each juggler’s task is either to delay both slots by the span (which can be accomplished by three crossed moves at appropriate pulses), or to delay the earlier slot by twice the span and the later slot not at all (which can be accomplished by a crossed move, followed by a straight move, followed by another crossed move). Thus in any case the juggler can be instructed to perform the appropriate sequence of moves by a suitable pattern for the cox.

We may summarize the import of Theorem 0 by saying that a time-division-multiplex rearrangeable connection network with $n = 2^\nu$ slots can be implemented by a *juggling network* with $2\nu - 1 = O(\log n)$ jugglers, overall delay $3n/2 - 2 = O(n)$, and total memory $(3n/2 - 2) + (2\nu - 1)2^\nu = O(n \log n)$. (In the expression for the total memory, the term $(3n/2 - 2)$ represents the memory for the principal jugglers in the chain, while the term $(2\nu - 1)2^\nu$ represents the memory of the coxes.) This yields an extremely attractive time-division-multiplex implementation, since the only aspect of the cost that grows as fast as the size of the corresponding space-division-multiplex network (as $O(n \log n)$) is the total memory, which can be furnished by relatively inexpensive technology (inertial delay lines), whereas the number of high-speed switching elements (represented by the jugglers) grows much more slowly (as $O(\log n)$).

In our description of juggling networks, we have assumed that jugglers execute their moves instantaneously, so that a ball received by a juggler executing a straight move is passed on at the same pulse. In practice there would be a fixed overhead time for a juggler, which might be a large fixed multiple of the pulse. In the chain of jugglers we have described, and more generally in any juggling network in which all balls are processed by the same number of jugglers, this overhead delay can be ignored in the analysis of the network, and it results merely in the addition of a constant delay per juggler being added to the overall delay. Even in more complicated juggling networks, with different numbers of jugglers on various paths between the source and the sink (as is necessary, for example, for the efficient construction of superconcentrators), this overhead delay can be taken into account by setting up "time zones" for the various jugglers, and introducing extra delays to compensate for differences in time zones. Thus we shall maintain the convenient fiction that jugglers act instantaneously, as it will have no effect on our conclusions and will simplify our analysis.

3. Applications

The great economy and elegance of the construction given in Section 2 leads us to seek other applications for these ideas. The natural starting point is the class of switching networks with interconnection patterns similar to that of the Beneš network. Some prominent members of this class are (1) the spider-web interconnection networks (see Pippenger [19,20]), (2) the Cantor non-blocking network [7], and (3) the Batcher bitonic sorting network [5]. The first two of these are externally controlled interconnection networks analogous to the Beneš network, and require no further comment. The Batcher bitonic sorting network, however, is based on comparators, and we should say something about how these devices can be realized by jugglers.

As described by Batcher [5], a comparator is a finite automaton that sorts two records received at its inputs, producing the same two records in sorted order at its outputs. To do this, it receives the records one bit at a time, with the bits of the keys by which the records are to be sorted preceding any other data in the records, and with the bits of the keys being received in order of decreasing significance. As long as the bits of the two input records remain identical, these identical streams of bits are reproduced at the outputs. Once the bits of the input keys differ, the correct sorted order is established, and the remainders of the records are reproduced at the outputs in this order. Viewed as a finite automaton, a comparator requires two bits of state information to keep track of whether or not the sorted order has been established and, if so, what that order is.

A time-division-multiplex implementation of a comparator entails three jugglers: a principal juggler who juggles balls representing the successive bits of the records, an assistant who juggles balls representing the state of the comparator (these balls will be of three distinct colors, representing the three possible states of the automaton), and a cox who instructs the other two jugglers as to which of the larger and smaller records should appear in the earlier and later output slots. In this way one can easily construct a time-division-multiplex implementation of Batcher's bitonic sorting network [5] with $O\big((\log n)^2\big)$ jugglers, overall delay $O(n)$ and total memory $O\big(n(\log n)^2\big)$.

To go beyond these simple applications, however, it is necessary to employ one of the essential tools of the theory of switching networks: expanding graphs (or, more generally, graphs with favorable eigenvalue separation ratios). Armed with an efficient time-division-multiplex implementation of this tool, we can explore the possible time-division-multiplex analogs of the following kinds of networks: (1) concentrators and superconcentrators, as introduced by Pinsker [16] and Valiant [24] (see also Pippenger [21]), (2) non-blocking connection networks, following Bassalygo and Pinsker [4] (see also Pippenger [17]), (3) sorting networks, following Ajtai, Komlós and Szemerédi [1,2] (see also Pippenger [18]), and (4) self-routing networks, as introduced by Arora, Leighton and Maggs [3] and Pippenger [22]. We shall not delve further into any of these applications here, but will describe in Section 4 a time-division-multiplex implementation for expanding graphs that should be of use in attacking all of them.

4. Expanders

This section is devoted to the time-division-multiplex implementation of expanding graphs. Our implementation will be based upon a particular explicit construction for expanding graphs, originated by Margulis [14], with improvements due to Gabber and Galil [8] and Jimbo and Maruoka [10].

We shall construct a basic expanding graph, which is a regular bipartite multigraph $G = (A, B, E)$, in which every vertex (in $A \cup B$) has degree 8 (meets 8 edges in E), and in which A and B each contain n vertices, where $n = m^2$ is a perfect square, and $m = 2^\mu$ is a perfect power of 2 (so that $n = 4^\mu$ is a perfect power of 4).

We shall do this by describing 8 perfect matchings $E_1, \ldots, E_8 \subseteq A \times B$, the union $E_1 \cup \cdots \cup E_8$ of which is E. To describe these matchings, we let $\mathbf{Z}_m$ denote the ring of integers modulo m, and identify both A and B with the direct product $\mathbf{Z}_m \times \mathbf{Z}_m$, which we shall regard as having for its elements the 2-element columns of elements from $\mathbf{Z}_m$. Each of the matchings E_i will then have the form

$$E_i = \{(z, \pi_i(z)) : z \in \mathbf{Z}_m \times \mathbf{Z}_m\},$$

where π_i is a permutation of $\mathbf{Z}_m \times \mathbf{Z}_m$ defined by an affine mapping of the form

$$\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} u \\ v \end{pmatrix}.$$

Thus it will suffice to specify, for each $i \in \{1, \ldots, 8\}$, the matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ and the column $\begin{pmatrix} u \\ v \end{pmatrix}$.

For one particular construction given by Jimbo and Maruoka [10], the matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ is one of the matrices $\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$, $\begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix}$ or their inverses $\begin{pmatrix} 1 & -2 \\ 0 & 1 \end{pmatrix}$, $\begin{pmatrix} 1 & 0 \\ -2 & 1 \end{pmatrix}$, and the column $\begin{pmatrix} u \\ v \end{pmatrix}$ is one of the columns $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ or their negatives $\begin{pmatrix} -1 \\ 0 \end{pmatrix}$, $\begin{pmatrix} 0 \\ -1 \end{pmatrix}$. Thus it will suffice to show how the permutations corresponding to each of these matrices and columns can be implemented by juggling networks, since then the permutations corresponding to the affine transformations can be implemented by connecting two such juggling networks in series, while the basic expanding graph can be implemented by connecting 8 such series combinations in parallel.

One approach to the problem of implementing these permutations would be to observe that, like all permutations, they can be carried out by the network described in Section 2, provided the coxes juggle appropriate patterns. The total number of balls juggled by coxes in Section 2 is $O(n \log n)$, but it might be possible to reduce this to $O(n)$ by careful analysis of the structure of the permutations. This sort of analysis has been done by Lenfant [11] for the space-division-multiplex implementation of Beneš's rearrangeable connection network. We shall not undertake such an analysis here, but rather will directly implement the required permutations with juggling networks.

First let us consider the map

$$\varrho : \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y+1 \end{pmatrix}.$$

We may arrange the elements of $\mathbf{Z}_m \times \mathbf{Z}_m$ in an $m \times m$ array, with $\begin{pmatrix} x \\ y \end{pmatrix}$ in the x-th row (numbered from the top) and the y-th column (numbered from the left). The map ϱ then corresponds to the operation of cyclically rotating each row one position to the right. Let the m^2 entries in this array correspond to the m^2 slots in a frame in "row-major order"; that is, let the first m slots in the frame correspond to the entries in the top row of the array (from left to right), and so forth. The successive rows of the array correspond to successive intervals of m slots (which we shall call "lines"), and to implement the map ϱ, we need to cyclically rotate each line of the frame, so that the last slot of the line is moved to the first slot of that line, and each other slot of the line is moved to the immediately following slot. Since the same operation is to be performed on each line, we may ignore the overarching organization of lines into frames, and consider simply the operation of cyclically rotating a line by one position.

We seek to delay slots 0 through $m-2$ of each line by 1 pulse and to "delay" slot $m-1$ by $-(m-1)$ pulses (that is, to advance it by $m-1$ pulses). We eliminate the negative delay by adding a delay of $m-1$ pulses to every slot of the line: thus we seek to delay slots 0 through $m-2$ by m pulses, and to delay slot $m-1$ by 0 pulses. This pattern of delays can be achieved by a single juggler who passes balls either immediately or after a single toss with a delay of m pulses. The corresponding pattern of straight and crosses moves has a period of m pulses, and thus can be coxed by a juggler with m balls. To summarize, the permutation ϱ can be implemented by a juggling network with $O(1)$ jugglers, $O(m)$ memory, and overall delay $O(m)$.

We can easily generalize the foregoing argument to the map

$$\varrho^k : \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ k \end{pmatrix} = \begin{pmatrix} x \\ y+k \end{pmatrix},$$

where $1 \leq k \leq m-1$. In this case, we seek to delay the first $m-k$ slots of each line by k pulses, and to "delay" the last k slots by $-(m-k)$ pulses. Adding a constant delay to eliminate negative delays, we find that the resulting pattern of delays can be achieved by the same juggler and cox as before; only the cox's pattern and the overall delay are changed, and the overall delay is *reduced* from its maximum of $m-1$. To summarize, the permutation ϱ^k can be implemented by a juggling network with $O(1)$ jugglers, $O(m)$ memory, and overall delay $O(m)$, where all constants are *independent* of k.

Next let us consider the map

$$\sigma : \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ x+y \end{pmatrix}.$$

Using the same organization of frames into lines as was used above, to implement the map σ we need to cyclically rotate the 0-th line not at all, rotate the 1-st line 1 position to the right, and in general rotate the x-th line x positions to the right.

To obtain an efficient implementation of this permutation, we shall assume that $m = 2^\mu$, for some natural number μ, so that each element of $\mathbf{Z}_m$ can be regarded as a μ-bit word (with the usual binary interpretation). Then, instead of subjecting each line to one of m different cyclic rotations, we will subject each line to a different subset of μ different rotations, with amounts of $2^0 = 1$ through $2^{\mu-1} = m/2$. In general, we will subject the x-th line to the rotation 2^λ positions to the right (for $0 \leq \lambda \leq \mu - 1$) if the $(\lambda+1)$-st bit in the binary representation of x is 1 (where the 1-st bit is the least, and the μ-th bit is the most, significant).

The permutation σ can thus be implemented by a chain of μ jugglers, each of whom passes each ball to the next juggler in the chain, either directly or after a single toss with a span of m pulses. Since each juggler contributes at most $O(m)$ to the overall delay, the chain contributes at most $O(\mu m) = O(m^2)$ to the overall delay. Each of these jugglers has a cox whose pattern has a period that depends on the position of the juggler in the chain. The cox for the juggler with rotation amount 1 has a period of 2 lines, the cox for the juggler with rotation amount 2 has a period of 4 lines, and in general the cox for the juggler with rotation amount 2^λ has a period of $2^{\lambda+1}$ lines. Summing these periods, we see that the total memory required by the coxes is $O(m)$ lines, or $O(m^2)$ pulses. To summarize, the permutation σ can be implemented by a juggling network with $O(\log m)$ jugglers, $O(m^2)$ memory, and overall delay $O(m^2)$.

We can easily generalize the foregoing argument to the map

$$\sigma^k : \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} 1 & 0 \\ k & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ kx + y \end{pmatrix},$$

where $1 \leq k \leq m-1$. We need only alter the behavior of each juggler to replace a cyclic rotation of 2^λ pulses by one of $k2^\lambda$ pulses (modulo m), for $1 \leq \lambda \leq \mu - 1$. This affects the patterns of the coxes, but not their periods or the spans of the jugglers. To summarize, the permutation σ^k can be implemented by a juggling network with $O(\mu)$ jugglers, $O(m^2)$ memory, and overall delay $O(m^2)$, where all constants are *independent* of k.

At this point we have seen how to implement 4 of the 8 permutations of our expanding graph, each with a juggling network of $O(\mu)$ jugglers, total memory $O(m^2)$ and overall delay $O(m^2)$. If we were to use the same strategy for the remaining 4 permutations, we would encounter the following problem: in order to cyclically rotate a column (rather than a row) we need a juggler with a span of $O(m^2)$ (rather than $O(m)$), and thus a chain of μ such jugglers would require a total memory of $O(\mu m^2)$, which exceeds our goal of $O(m^2)$. We shall therefore use a different strategy for these 4 remaining permutations. We shall consider the map

$$\tau : \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} y \\ x \end{pmatrix}.$$

We shall implement the corresponding permutation using a chain of $O(\mu)$ jugglers, with $O(m^2)$ memory and overall delay $O(m^2)$. We can then implement the permutation corresponding to the map

$$\varrho'^k : \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} k \\ 0 \end{pmatrix} = \begin{pmatrix} x + k \\ y \end{pmatrix}$$

using the identity $\varrho'^k = \tau \circ \varrho^k \circ \tau$, and the permutation corresponding to the map

$$\sigma'^k : \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} 1 & k \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x + ky \\ y \end{pmatrix}$$

using the identity $\sigma'^k = \tau \circ \sigma^k \circ \tau$.

The map τ corresponds to the permutation that transposes the array of elements of $\mathbf{Z}_m \times \mathbf{Z}_m$. Our implementation of this permutation will be based on the following identity, in which A, B, C and D denote $(m/2) \times (m/2)$ subarrays of an $m \times m$ array, and a superscript T denotes "transpose":

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^T = \begin{pmatrix} A^T & C^T \\ B^T & D^T \end{pmatrix}.$$

This identity suggests a strategy that begins by exchanging the subarrays B and C (without transposing them), then proceeds recursively to transpose all four subarrays.

The operation of exchanging B and C is straightforward, since it reduces to exchanging a sequence of pairs of slots at a fixed distance in each frame. Specifically, we want to delay the last $m/2$ slots of the first $m/2$ lines (the elements of B) by $m(m-1)/2$ pulses ($m/2$ lines minus $m/2$ pulses), delay the first $m/2$ slots of the last $m/2$ lines (the elements of C) by $-m(m-1)/2$ pulses, and delay all other slots by 0 pulses. Adding an overall delay of $m(m-1)/2$ pulses to eliminate the negative delays, we see that the required exchange can be accomplished by a juggler with a span of $m(m-1)/2$ pulses, who passes each ball after 0, 1 or 2 tosses, for a delay of 0, $m(m-1)/2$ or $m(m-1)$ pulses. The juggler is coxed by a partner with a pattern of period 1 frame, or m^2 pulses.

After the exchanges performed by the juggler just described, it remains to transpose each of the subarrays A, B, C and D. To do this we proceed recursively, partitioning each of these subarrays into four $(m/4) \times (m/4)$ subsubarrays, exchanging the two off-diagonal subsubarrays of each subarrays, and so forth. Each level of the recursion will contribute one juggler to a chain of μ jugglers, of which the first (described above) is responsible for exchanging two subarrays, the second is responsible for exchanging four pairs of subsubarrays (one pair in each subarray), and so forth. The λ-th juggler will have a span of $m(m-1)/2^\lambda$ pulses (and will pass each ball after 0, 1 or 2 tosses), and will be coxed by a partner with a period of $m^2/2^{\lambda-1}$ pulses. Adding the contributions of the jugglers in this chain, we see that the permutation corresponding to the map τ is implemented by a juggling network with $O(\mu)$ jugglers, total memory $O(m^2)$ and overall delay $O(m^2)$, as claimed above.

This completes the implementation of our basic expanding graph, since the 8 permutations required for this graph can be fabricated by composing a bounded number of permutations, each of the form ϱ^k, σ^k, or τ. Furthermore, a graph with any desired fixed ratio of eigenvalue separation can be obtained by raising our basic expanding graph to a fixed power (see for example Pippenger [22]). Thus each of the bounded number of permutations required for this desired graph can be fabricated by composing a bounded number of permutations from the basic expanding graph, and we obtain the following theorem.

Theorem 1: For any desired eigenvalue separation ratio (that is, ratio between largest two absolute values of eigenvalues) R, there exists a natural number $d = 2^\delta$ such that, for every natural number $n = 4^\mu$, there exist d permutations $\pi_1, \ldots, \pi_d$ of n objects such that (1) the sum of the matrices of the permutations $\pi_1, \ldots, \pi_d$ has eigenvalue separation ratio at least R, and (2) each of the permutations $\pi_1, \ldots, \pi_d$ can be implemented by a juggling network with $O(\log n)$ jugglers, total memory $O(n)$, and overall delay $O(n)$.

5. Conclusion

We have shown in this paper how to construct time-division-multiplex analogues of expanding graphs, which are an essential component in many asymptotically optimal constructions for switching networks. We have described this construction in terms of a juggling metaphor that is useful in its own right as an aid to visualizing the operation of switching networks. Aside from more or less routine applications of this construction to various problems concerning switching networks, some more conceptual problems remain to be addressed.

At this time there are no lower bounds for time-division-multiplex networks except for those inherited in an obvious way from the theory of space-division-multiplex networks. Consider for example the construction of connectors given in Section 2. The memory requirement $O(n \log n)$ is clearly best possible, since that much memory, $\Omega(\log(n!)) = O(n \log n)$, is needed to remember the identity of 1 out of $n!$ possible permutations. Similarly, the overall delay of $O(n)$ is best possible, since routing the last slot of an input frame to the first slot of an output frame clearly requires that the frame be delayed by $\Omega(n)$ pulses. Finally, the bound of $O(\log n)$ switches is best possible, provided we assume an overall delay of $O(n)$, since the number of switches in a time-division-multiplex network, times the overall delay of that network, must be at least as large as the number of switches in a space-division-multiplex network performing the same task. We do not know, however, how to prove a lower bound to the number of switches when the constraint on the overall delay is relaxed (say to $O(n \log n)$), or how to prove a lower bound to the number of switches required to implement specific permutations such as those treated in Section 4.

While the construction for expanding graphs used in Section 4 suffices to provide any desired eigenvalue separation ratio (given that the degree is no object), there are other constructions that are both more economical from a practical point of view and essential for certain theoretical purposes. The most prominent of these are the Ramanujan graphs introduced by Lubotzky, Phillips and Sarnak [12] and by Margulis [15]. Whether there are efficient time-division-multiplex implementations of these graphs remains an open question.

6. References

[1] M. Ajtai, J. Komlós and E. Szemerédi, "Sorting in $c \log n$ Parallel Steps", *Combinatorica*, 3 (1983) 1–19.

[2] M. Ajtai, J. Komlós and E. Szemerédi, "An $O(n \log n)$ Sorting Network", *Proc. ACM Sym. on Theory of Computing*, 15 (1983) 1–9.

[3] S. Arora, T. Leighton and B. Maggs, "On-Line Algorithms for Path Selection in a Nonblocking Network", *Proc. ACM Sym. on Theory of Computing*, 22 (1990) 149–158.

[4] L. A. Bassalygo and M. S. Pinsker, "Complexity of an Optimal Nonblocking Switching Network without Reconnections", *Problems of Inform. Transm.*, 9 (1974) 64–66.

[5] K. E. Batcher, "Sorting Networks and Their Applications", *Proc. AFIPS Spring Joint Computer Conf.*, 32 (1968) 307–314.

[6] V. E. Beneš, "Optimal Rearrangeable Multistage Connecting Networks", *Bell Sys. Tech. J.*, 43 (1964) 1641–1656.

[7] D. G. Cantor, "On Non-Blocking Switching Networks", *Networks*, 1 (1971) 367–377.

[8] O. Gabber and Z. Galil, "Explicit Constructions of Linear-Sized Superconcentrators", *J. Comp. and System Sciences*, 22 (1981) 407–420.

[9] H. Inose, "Blocking Probability in 3-Stage Time Division Switching Network", *J. IECEJ*, 44 (1961) 935–941.

[10] S. Jimbo and A. Maruoka, "Expanders Obtained from Affine Transformations", *Combinatorica*, 7 (1987) 343–355.

[11] J. Lenfant, "Parallel Permutations of Data: A Beneš Network Control Algorithm for Frequently Used Permutations", *IEEE Trans. on Computers*, 27 (1978) 637–647.

[12] A. Lubotzky, R. Phillips and P. Sarnak, "Ramanujan Graphs", *Combinatorica*, 8 (1988) 261–277.

[13] M. J. Marcus, "Designs for Time Slot Interchangers", *Proc. National Electronics Conf.*, 26 (1970) 812–817.

[14] G. A. Margulis, "Explicit Construction of Concentrators", *Problems of Inform. Transm.*, 9 (1974) 71–80.

[15] G. A. Margulis, “Explicit Group-Theoretical Constructions of Combinatorial Schemes and Their Application to the Design of Expanders and Concentrators”, *Problems of Inform. Transm.*, 24 (1988) 39–46.

[16] M. S. Pinsker, “On the Complexity of a Concentrator”, *Proc. Internat. Teletraffic Congr.*, 7 (1973) 318/1–4.

[17] N. Pippenger, “Telephone Switching Networks”, *Proc. AMS Symp. Appl. Math.*, 26 (1982) 101–133.

[18] N. Pippenger, “Communication Networks”, in J.van Leeuwen (Ed.), *Handbook of Theoretical Computer Science—Volume A: Algorithms and Complexity*, Elsevier, Amsterdam, 1990.

[19] N. Pippenger, “The Blocking Probability of Spider-Web Networks”, *Random Structures & Algorithms*, 2 (1991) 121–149.

[20] N. Pippenger, “The Asymptotic Optimality of Spider-Web Networks”, *Discr. Appl. Math.*, 37/38 (1992) 437–450.

[21] N. Pippenger, “Rearrangeable Circuit-Switching Networks”, *Proc. Internat. Conf. on Graph Theory, Combinatorics, Algorithms and Applications*, 7 (1992) (to appear).

[22] N. Pippenger, “Self-Routing Superconcentrators”, *Proc. ACM Sym. on Theory of Computing*, 25 (1993) 355–361.

[23] S. V. Ramanan, H. F. Jordan and J. R. Sauer, “A New Time-Domain, Multistage Permutation Algorithm”, *IEEE Trans. Info. Theory*, 36 (1990) 171–173.

[24] L. G. Valiant, “Graph-Theoretic Properties in Computational Complexity”, *J. Comp. and System Sciences*, 13 (1976) 278–285.