Solving the Corner-Turning Problem for Large Interferometers

Andrew Lutomirski
Center for Theoretical Physics, Massachusetts Institute of Technology, Cambridge, MA 02139

Max Tegmark, Nevada Sanchez, and Leo Stein
Dept. of Physics & MIT Kavli Institute, Massachusetts Institute of Technology, Cambridge, MA 02139

W. Lynn Urry
Dept. of Astronomy, University of California, Berkeley, CA 94720

Matias Zaldarriaga
Institute for Advanced Study, Einstein Drive, Princeton, NJ 08540, USA

(Dated: Submitted to *MNRAS* October 12 2009, accepted August 22 2010)

The so-called corner turning problem is a major bottleneck for radio telescopes with large numbers of antennas. The problem is essentially that of rapidly transposing a matrix that is too large to store on one single device; in radio interferometry, it occurs because data from each antenna needs to be routed to an array of processors that will each handle a limited portion of the data (a frequency range, say) but requires input from each antenna. We present a low-cost solution allowing the correlator to transpose its data in real time, without contending for bandwidth, via a butterfly network requiring neither additional RAM memory nor expensive general-purpose switching hardware. We discuss possible implementations of this using FPGA, CMOS, analog logic and optical technology, and conclude that the corner turner cost can be small even for upcoming massive radio arrays.

I. INTRODUCTION

There is now strong community interest in building more sensitive radio telescopes, stemming from diverse science opportunities that range from planets to pulsars, from black holes to cosmology [1–6]. However, greater sensitivity requires greater collecting area, which in turn increases cost. For large collecting area, interferometers become more cost-effective than single-dish radio telescopes, but pose interesting engineering challenges.

One such challenge is that for a general large interferometer, the cost grows quadratically with area, because for an array of N antennas, all $N(N-1)/2 \lesssim N^2$ pairs of antennas need to be correlated to calculate the so-called visibilities. The computational costs thus scale as N^2 and dwarf all other costs for large enough N. Fortunately, there are attractive design approaches such as tiling, the MOFF-correlator [7] and the omniscope [11, 12] which enable more competitive cost scaling, some having costs that grow as slowly as $N \log N$ with size without losing any information [12].

A second bottleneck, common to all the above-mentioned approaches, is what is known as the *corner turning problem*. The problem is essentially that of rapidly transposing a matrix that is too large to store on one single device; in radio interferometry, it occurs because data from each antenna needs to be routed to an array of processors that will each handle a limited portion of the data (a frequency range, say) but requires input from each antenna. Most often, the data from each of the N antenna signals can be filtered, amplified, digitized and decomposed into channels separately from all other antenna signals. Suppose that the signal from each antenna is digitized and processed so as to

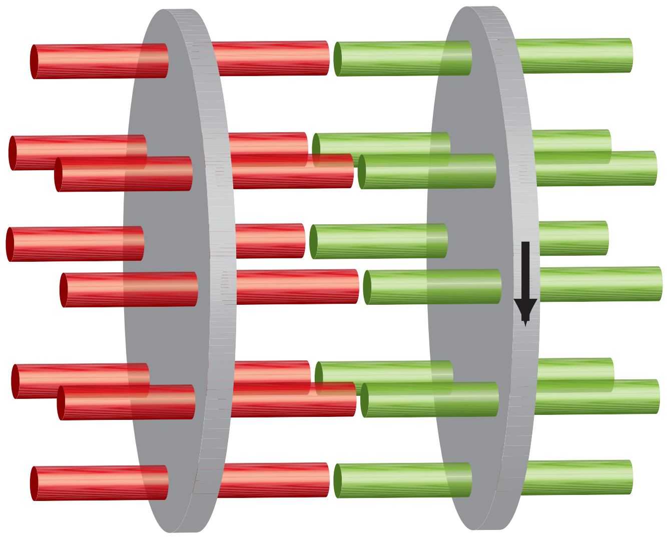

FIG. 1: Toy example of how the $N = 8$ corner turning problem could be solved using moving parts, with 8 devices simultaneously transmitting data into the 8 red/dark grey links on the left and 8 other devices receiving data from the 8 green/light grey links on the right. After 8 successive 45° rotations of the right wheel, all input devices have transmitted the required data to all output devices.

produce a sample stream from each of M separate frequency channels.[1] If we imagine all this data arranged in

[1] For simplicity, we ignore the polarization issue in our discussion, since it can be trivially incorporated by simply doubling N and

an $N \times M$ matrix, then each *row* of the matrix thus resides on a separate physical device (typically an FPGA, a field-programmable gate array). However, the subsequent computation of UV plane visibilities needs to combine information from all N antennas, separately for each frequency, *i.e.*, each *column* of the matrix needs to be processed separately and rapidly. For a traditional interferometer, this second stage involves multiplying the numbers from the $N(N-1)/2$ pairs of numbers, the so-called "x-operation", as distinguished from the preceding so-called f operation that Fourier transformed in the time domain. For an omniscope, this second stage involves performing a Fourier transform in two or more dimensions [12]. In either case, the amount of computation of required for the second stage is at least proportional to N, so we need $M \gtrsim N$ channels to distribute the computation. In either case, the corner turning is a major bottleneck, since transferring data between all $N \times M$ pairs of first-stage and second-stage devices requires moving the entire contents of the matrix across a large network. For example, for an $N = 64 \times 64$ dual polarization omniscope sampling at 400 MHz, the corner turner has to route about 13 terabytes per second. Once this bottleneck has been passed and the second stage has been completed, however, the resulting sky maps (or their Fourier transforms) can be time-averaged, dramatically reducing the data rate to manageable levels.

Modern radio telescopes have typically adopted one of the following solutions to the corner turning problem:

1. Writing the entire matrix to a single, extremely fast, giant memory module where it can be read out transposed, or using some other device with size $O(M^2)$, for example enough wires to turn the corner directly.

2. Routing all the data through an off-the-shelf non-blocking switch.

3. Using enough wires to make all the connections directly.

The first approach has been used by numerous experiments, and the second has been successfully implemented in the packetized CASPER correlator [13, 14] used by the PAPER experiment [15], where $N = 32$ (including polarization) is small enough to be handled by a single 10 GB Ethernet switch. The third is used in some very large correlators such as EVLA and ALMA. Unfortunately, all of these approaches become expensive for very large N, which makes it timely to explore alternative solutions.

Another way of thinking about the corner turning problem is that it involves the only part of an interferometer that is not embarrassingly parallel[2]: it is easy to build many antennas, many A/D converters, many time-domain Fourier transformers and many correlators acting on separate frequency bands. The corner turn is the piece that transposes the data matrix to keep the processing embarrassingly parallel.

The rest of this paper is organized as follows. In Section II, we present our solution to the corner turning problem, which requires neither general-purpose network switches nor additional memory. We discuss various physical implementation options in Section III and summarize our conclusions in Section IV.

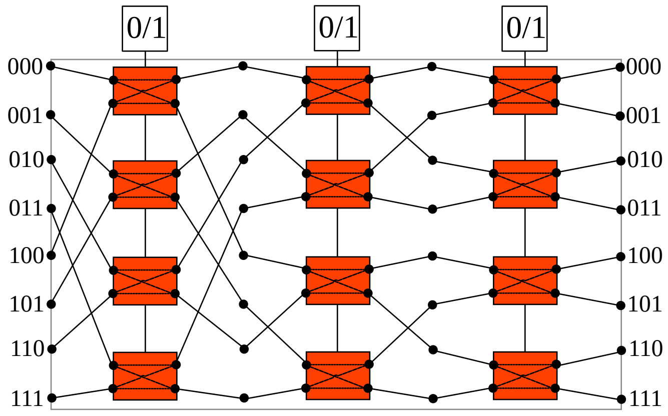

FIG. 2: How to solve the $N = 8$ corner turning problem using a butterfly network, with 8 devices simultaneously transmitting data from the left and 8 other devices receiving data from the right. After the three control bits shown at the top have looped through all 8 combinations 000, 001, 010, 011, 100, 101, 110 and 111, all input devices have transmitted the required data to all output devices. The boxes ("controlled swappers") have two input wires and either pass them straight through to their output wires or swap them, depending on whether the control bit (drawn as entering from above) is 0 or 1, respectively. The 8 inputs are numbered in binary on the left hand side, and we see that the 1st row of swappers can flip their 1st address bit, the 2nd row of swappers can flip their 2nd bit, and the 3rd row of swappers can flip their 3rd bit.

II. THE BUTTERFLY ALGORITHM

Below we will limit our discussion to the special case where $M = N$, *i.e.*, where the matrix to be transposed is square, or, equivalently, where there are equal numbers of devices writing to and reading from the corner turner, since if one has an efficient corner turner for this case, then the general case becomes easy to solve. For exam-

treating each of the two polarization channels from each antenna as an independent data stream.

[2] Computer scientists say that a problem is "embarrassingly parallel" if it can be trivially distributed across a large number of processors that do not need to communicate with each other.

ple, one can simply pad the matrix with zeros, which is equivalent to inserting dummy data sources or sinks, or combine a few adjacent entries into one larger entry, splitting it at the end if necessary [10]. We will present an optimized solution for the case where N is a power of 2.

A. The problem

In the discussion below, we will refer to the concept of a "link," by which we mean some connection between two computers, FPGAs, or any other devices (nodes) that can carry data at a rate of N matrix entries each time a matrix is transposed. In most cases, this will simply be the rate at which the f stage of the correlator outputs data *for a single antenna*. In other words, the data rate on a given link is independent of the size of the interferometer. If the data rate for a single channel exceeds that of the technology we use to build our corner turner (*e.g.*1 Gbps if we used gigabit Ethernet connections), then we will use bundles of identical connections as our links.

It is easy to solve the corner turning problem with a non-blocking switch that can connect $2N$ links: each source node that starts with one row of the matrix simply transmits all of its data, with each entry in the matrix addressed to the sink node labeled with the column number of that entry. Each node sends or receives at exactly the link rate, so a general-purpose nonblocking switch can handle all of the data.

Large general-purpose high-speed switches are expensive because they are fully non-blocking, allowing any set of input devices to simultaneously transmit to any set of output devices. This is overkill for the corner turning problem, since we have complete prior knowledge of how data needs to be distributed. This suggests the possibility of reducing cost by giving up complete generality.

We need each of the N source nodes to transmit data through our corner turner to each of the N sink nodes, with each source node transmitting exactly a $1/N$ fraction of its total data rate to each sink node. This can be done with a very restricted kind of switch using no memory at all. Such a switch has N states, selected by a control input $c \in \{0, \ldots, N-1\}$, where the source node labeled with a number i is connected to the sink node $j = p(c, i)$ where the function p has the following properties:

1. For fixed i, all $p(c, i)$ are unique and in the range $0, ..., N-1$.
2. For fixed c, all $p(c, i)$ are unique and in the range $0, ..., N-1$.

In other words, the corner turner performs a different permutation of the inputs at each time step, such that after N steps, every input node has been connected to every output node exactly once.

With such a switch, each source node i transmits the i, j entry of the matrix exactly when $p(c, i) = j$, and each sink node j will receive the entire column j, albeit in some arbitrary order. This means that each source node transmits data in a different order, but most receiver or f stage designs should be able to handle this without difficulty.

B. Our solution

How should we choose the sequence of permutations p? There are clearly vast numbers of permutation sequences that satisfy the two requirements above, since we have $N!$ choices even for $p(0, j)$ alone.

1. *A mechanical solution*

One simple solution is that defined by the cyclic permutations

$$p(c, i) \equiv c + i \mod N. \tag{1}$$

This choice is illustrated in Figure 1 for a toy example where where $N = 8$. If we connect the input devices to the metal bars protruding on the left side and the output devices to the bars protruding to the right, then the N successive 45° rotations of the right wheel will achieve a complete corner turn where every input device has transmitted to every output device.

2. *The butterfly algorithm*

In practice, one of course needs to accomplish all operations electronically without large moving parts. An elegant method for implementing precisely the cyclic permutations of Figure 1 electronically was discovered about a decade ago by Lynn Urry and implemented for the Allen Telescope Array [8, 9], but this was unfortunately never published in a journal and did not become as widely known as it deserves to be. The other authors of this paper independently discovered the methods that we will describe below, which have the further advantage of being even cheaper to implement.

We schematically illustrate a simple solution in Figure 2, where the boxes ("controlled swappers") have two input wires and either pass them straight through to their output wires or swap them, depending on whether a control bit (drawn as entering from above) is 0 or 1, respectively. If the $N = 8$ inputs i are numbered in binary, then the 1st row of swappers can flip their 1st bit, the 2nd row of swappers can flip their 2nd bit, and the 3rd row of swappers can flip their 3rd bit. This means that this corner turner implements the permutations

$$p(c, i) \equiv c \operatorname{xor} i, \tag{2}$$

where the integers c, i and j are written in binary on the top, left and right sides of Figure 2, respectively.

This basic network topology where a given node successively "talks" with nodes separated by 2^0, 2^1, 2^2, *etc.* appears in a wide range of electrical engineering and software applications, including the Fast Fourier transform of N numbers, and is often referred to as a "butterfly network". When the "talk" part is a swapper like in Figure 2, the resulting network is a special case of a Banyan network [16] — a type of general-purpose network switch which is nonblocking for certain permutations (as opposed to fully nonblocking, which would allow nodes to talk to each other in any permutation.) A key point about the corner turner that we are proposing is that we are *not* using it as a general-purpose switch but rather with a specific algorithm: to cycle through N very particular permutations, which is precisely what is needed to solve the problem at hand.

For comparison, the method in [8, 9] also uses a Butterfly network, but changes all $N \log N$ controlled swappers independently at each control step c instead of using the same setting for each of the $\log N$ columns. The latter implementation thus requires N times fewer control input wires, and we will see below how it can be further simplified to cut cost.

The butterfly algorithm we have proposed requires that N be a power of 2. As we will see in Section III, the cost of a butterfly corner turner is likely to constitute only a small fraction of the total cost of a large N radio array, so for a general number of antennas, a one can simply round N up to the nearest power of two for the corner turner.

3. An even cheaper corner turner using perfect shufflers

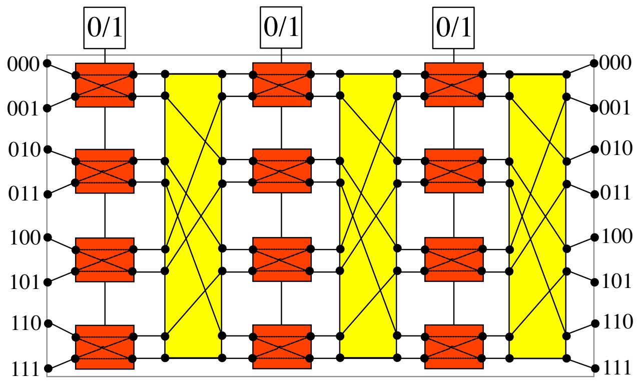

FIG. 3: An equivalent network to Figure 2, but reorganized so that the connections after each row of swappers are identical. These connections, contained within the tall boxes, are seen to correspond to a "perfect shuffle", whereby the top half gets interleaved with the bottom half, and corresponds to cyclically shifting the address bits that label the inputs on the left-hand-side.

An obvious drawback to the configuration in Figure 2 is that the wiring layer between each stage is different, which complicates the manufacturing of a device to implement the network. It turns out that by a suitable permutation of the nodes after each row of swappers, one can simplify the wiring diagram so that all layers becomes identical, as illustrated in Figure 3.

The required permutation performed between the wires after each row of swappers, contained within the tall boxes in the figure, turns out to be a so called "perfect shuffle" permutation on N elements, which draws its name from card shuffling. The perfect shuffle (or Faro shuffle) is defined as

$$\begin{cases} j \mapsto 2j & \text{if } j < N/2, \\ j \mapsto 2j - N + 1 & \text{otherwise,} \end{cases} \tag{3}$$

and corresponds to interleaving the top and bottom halves in the input [17].

Figure 3 illustrates that if we write the input row j as a binary number composed of $\log_2 N$ bits, a perfect shuffle simply permutes its bits cyclically, shifting them all one notch to the left and moving the leftmost bit all the way to the right. $\log_2 N$ perfect shuffles thus restores j to its original value. Since the rows of swappers in Figure 3 can flip the rightmost bit (exchanging two neighboring rows), the net effect of the n^{th} control bit from the left at the top of the figure is thus to control the n^{th} bit from the right of j. In other words, Figure 3 corresponds to the same permutation sequence $p(c, i) = c \operatorname{xor} i$ as Figure 2 except for the trivial modification that the control variable i has its bits in reverse order.

To further reduce cost, we can omit the last perfect shuffle layer, since the resulting network is equivalent to the butterfly network with its outputs permuted and retains all of the necessary properties.

III. IMPLEMENTATION

A. Layout

Section II described the basic layout of the butterfly network. We can solve the corner turning problem for an N-element interferometer using an N-link butterfly network. Since this technique is meant to scale to *large* radio telescopes, an ideal implementation of the butterfly network would be built out of large (but not too large) numbers of identical, inexpensive, and easy-to-connect parts.

In the discussion below, we use $n = \log_2 N$ to refer to the number of layers in the network.

1. *Network cost*

If we built a butterfly network corner turner out of modular components, then the cost could be roughly computed by counting the number of each type of component. To build a very large network, or to build many smaller networks, it would be worth the extra effort to design the components so that they would be inexpensive to manufacture and so that as few as possible would be needed.

The simplest set of components to use would be discrete 2×2 switches and single-link cables. Each controlled swapper layer is $N/2$ identical 2×2 switches arranged in a column and each perfect shuffle layer consists of N cables running between pairs of computers or switches. For a hypothetical $2^{20} = 1048576$-node interferometer, this comes out to 20 layers, for a total of 10,485,760 switches and 20,971,520 cables. This is doable (the interferometer would be expensive enough that this would most likely be only a small part of the cost), but connecting 20 layers of over one million cables without making mistakes would be tedious at best.

2. *How to further reduce the cost*

There is room for a large improvement in the number of parts needed, though: printed circuit boards containing hundreds of components are inexpensive (in 2009, 12 inch by 14 inch circuit boards can be fabricated for less than \$30 each, even in small volumes, assuming that the circuit fits on two layers) and cables are available that can carry many links worth of bandwidth. (Of course, using large cables may only be useful internally—unless multiple source or sink nodes are on the same board, the input and output links must each be on its own cable.)

To optimize the perfect shuffles, we will use a simple property of a cyclic bit shift: an n-bit binary number $b_{n-1}b_{n-2}\dots b_0$ can be circularly shifted one bit to the left by first circularly shifting the leftmost $n-k$ bits one bit to the left, giving $b_{n-2}\dots b_{k+1}b_kb_{n-1}b_{k-1}\dots b_0$ and then shifting the rightmost $k+1$ bits one bit to the left, giving $b_{n-2}\dots b_{k+1}b_kb_{k-1}\dots b_0b_{n-1}$ as desired. (For example, $n = 5$ and $k = 2$ starts with $b_4b_3b_2b_1b_0$, maps it to $b_3b_2b_4b_1b_0$, and finally maps it to $b_3b_2b_1b_0b_4$.)

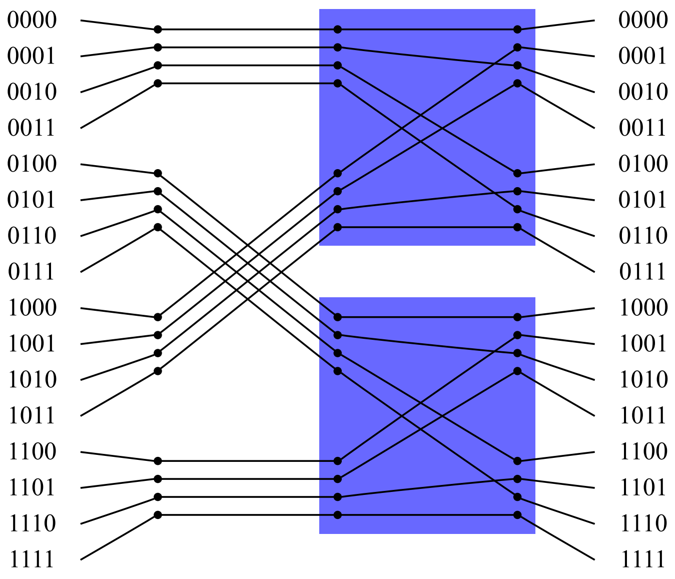

FIG. 4: A 16-link perfect shuffler built out of four cables, each carrying four links, and two eight-link perfect shufflers, shaded blue.

If the left sides of 2^{n-k} cables, each carrying 2^k links were lined up such that the first cable (cable 0) carried links $0, \dots, 2^k-1$, the second carried links $2^k, \dots, 2\cdot 2^k-1$, etc, then the leftmost $n-k$ bits of the link number could be circularly shifted one bit to the left by arranging the *cables* into a perfect shuffle. The rightmost $k+1$ bits could be circularly shifted one bit to the left by connecting each pair of adjacent cables (*i.e.*all of the links sharing the leftmost $n-k-1$ bits after the first circular shift) into a board that had two cable inputs and two cable outputs, with the individual links in the cables arranged into a perfect shuffle. This construction with $n = 4$ and $k = 2$ is shown in Figure 4.

From these two components, using $k = 6$ (digital cables with 64 digital links are commercially available), each perfect shuffle layer in a 2^{20}-link butterfly network would require 16,384 cables and 8192 two-cable perfect shufflers. This is a nice saving over using single-link cables, especially in terms of the labor needed to assemble the layers.

By using larger circuit boards, many controlled swappers and many two-cable perfect shufflers could fit on a single circuit board, further cutting the number of parts.

As an additional improvement, the layers in a butterfly network can be rearranged—rather than using n layers, each of which circularly shifts left by one bit and then xors the least significant bit, the network could instead use n/ℓ layers, each of which xors the least significant ℓ bits with an ℓ-bit control and then circularly shifts by ℓ bits (if n is not a multiple of ℓ, then one layer would involve fewer bits). A device that toggles the least significant ℓ bits of the address does not mix between blocks of 2^ℓ bits, so a simple row of 2^ℓ-link ℓ-bit togglers would make a 2^n-link ℓ bit toggler. This saves a factor of ℓ in the number of togglers needed; ℓ perfect shuffle layers could go between each toggler layer.

Putting this together for a 2^{20}-link butterfly network, using $k = 6$ and $\ell = 8$ gives 20 perfect shuffle layers for a total of 327,680 cables and 163,840 two-cable perfect shufflers and three controlled xor layers containing a total of 12,288 256-link 8-bit togglers (of which one of the three layers would only use half of its control inputs).

Some further improvements would be possible by building layers that circularly shifted by more than one bit at a time—these could still use multiple-link cables but would need larger, although still probably inexpensive, multiple-bit perfect shuffle boards.

B. Technology

So far, we have discussed layouts from an abstract point of view, without considering the particular technology used. We will now briefly survey some attractive options for implementing this in practice.

1. *Off-the-shelf hardware*

The easiest design to prototype would use small (16-port, perhaps) nonblocking Ethernet switches and standard Ethernet cables, with one link per cable. The switches are physically capable of performing any combination of swapping and shuffling on their ports, but they are not generally meant to change their connections on the fly. Most so-called managed switches can, however, store a small number (often 4096) of fixed routes indexed by the destination of the packet that they are switching (this is called the forwarding table). This means that, with some care, the destination field could be used to control the entire route taken by each packet traversing a butterfly network of Ethernet switches. This could be inexpensive (a few tens of dollars per switch port in 2009 for one Gbps per link for a programmable managed switch and significantly more for 10 Gbps) but does not scale well beyond the square root of the size of the forwarding table, giving an upper bound of $N \leq 64$. More flexible switches and routers are available, but tend to be far more expensive and slower.

FPGAs also allow rapid development. Most FPGAs have both standard I/O pins, where a high voltage held for one clock cycle indicates a 1 bit and a low voltage indicates a 0 bit (these pins are limited to relatively low data rates on all but the most expensive FPGAs), and high-speed serial I/O pins, which can send or receive several gigabits per second on differential pairs of pins. Any FPGA can easily act as an arbitrary shuffler or swapper, limited only by the numbers and types of I/O pins it has. We experimented with FPGA implementations and found that a 2×2 controlled swapper, even on a bottom-of-the-line Xilinx FPGA, could switch once per clock cycle, which was 64 million times per second on our device. The downsides of FPGAs are their price and the fact that most FPGAs have relatively few high-speed I/O pins, keeping the cost per link quite high if data rates higher than one bit per clock per link are needed.

2. *Digital ASICs*

Custom application-specific integrated circuits (ASICs) can be fabricated in 2009 for a few hundred thousand dollars to make a set of masks plus very little for each part produced. Any technology that can transmit and receive high-speed digital data can also be used to switch it. For example, CMOS devices can switch at moderate speeds (several Gbps) and current-mode devices can operate in excess of 10Gbps per differential pair. The number of links switched on a chip is limited only by the number of pins available on the chip, and a printed circuit board can hold as many of these chips as will fit at very low cost.

In fact, with a signaling technology that can tolerate enough loss, shufflers could be build on circuit boards containing no chips at all, keeping costs even lower.

3. *Analog switching*

Many protocols for transmitting large data rates over copper cables use advanced modulation techniques. For example, gigabit Ethernet over CAT5 cabling uses multilevel signaling to achieve two gigabits per seconds (1Gbps each way) over four wire pairs at low frequency. Devices to encode, decode, and error correct these kinds of protocols are complex and require significant power to operate, so it would be useful to minimize the number of times that data is modulated and demodulated in a butterfly network. If we used analog switches that could exchange two *modulated* signals with little enough loss, then we could have several layers of controlled swappers between each modulator/demodulator pair.

4. *Cable technologies*

Technologies to send large data rates over copper cable are well established. Over short distances, a single conductor can carry one link. Over longer distances, differential signals are usually sent over one pair of conductors per link, arranged into some form of transmission line, and different techniques can be used to modulate the signal depending on the frequency response of the transmission line, the available transmission power, and cost considerations. Copper cables have the advantages of being relatively easy to construct, easy to connect, and they can interface easily with switching electronics.

Optical fibers, on the other hand, can carry much higher data rates than copper over a single fiber, and all-optical switching technologies (photonic or mechanical) can rapidly switch these high data rates with little loss. Optical cables are inexpensive, but connecting them is labor-intensive and they are far more expensive to interface with electronics than copper.

Finally, it is possible to transmit very high data rates between boards without any cables at all using free-space optical communication, in which boards have lasers and photodiodes aimed at each other. These laser beams can freely intersect each other, and devices that automatically aim and focus free-space optical links are available. See [18] for an example of a large network switch built out of free-space optical links. This technology could eliminate the need to hand-wire perfect shufflers altogether.

IV. CONCLUSIONS

The corner turner in a conventional interferometer ranges from moderately expensive (if the telescope is small enough to use an off-the-shelf Ethernet switch) to extremely expensive (if the number of antennas N is very large). We have presented a corner turning algorithm based on a butterfly network topology which can solve the corner turning problem for any N for an $\mathcal{O}(N \log N)$ hardware cost with a constant prefactor if inexpensive custom network parts are used. Even for an interferometer with over one million antennas, the corner turner could require well under one million network parts, each of which could cost only a few dollars in large volumes. By eliminating a key bottleneck, this bodes well for future large-N radio telescopes.

Acknowledgements: The authors wish to thank Richard Bradley, Adrian Liu, Jacqueline Hewitt, Michael Matejek, Miguel Morales, Edward Morgan, Courtney Peterson, Scott Morrion, Joel Villasenor, Dan Werthimer, Chris Williams and an anonymous referee for helpful comments. This work was supported by NASA grants NAG5-11099 and NNG 05G40G, NSF grants AST-0607597, AST-0708534, AST-0907969, AST-0908848 and PHY-0855425, the DoD through the NDSEG Program, and fellowships from the David and Lucile Packard Foundation.

[1] C. Carilli and S. Rawlings eds., *Science with the Square Kilometre Array* (Elseview: Amsterdam, `http://www.skads-eu.org/p/SKA_SciBook.php`, 2004)
[2] S. Jester and H. Falcke, *New Astron. Rev.*, **53**, 1 (2009)
[3] S. Furlanetto *et al.*, arXiv:0902.3011 [astro-ph] (2009)
[4] S. Furlanetto *et al.*, arXiv:0902.3259 [astro-ph] (2009)
[5] E. Komatsu *et al.*, arXiv:0902.4759 [astro-ph] (2009)
[6] J. Aguirre *et al.*, arXiv:0903.0902 [astro-ph] (2009)
[7] M. F. Morales, arXiv:0812.3669 [astro-ph] (2008)
[8] W. L. Urry, "A Corner Turn Architecture." ATA Memo No. 14, 2000 Nov 17.
[9] W. L. Urry, M. Wright, M. Dexter & D. MacMahn, "The ATA Correlator", ATA memo 73.
[10] L. D'Addario, "Generalization of the Memoryless Corner Turner to the Non-Square Case." ATA Memo No. 22, 2001 April 2.
[11] M. Tegmark and M. Zaldarriaga, *PRD*, **79**, 083530 (2009)
[12] M. Tegmark and M. Zaldarriaga, arXiv:0909.0001 [astro-ph] (2009)
[13] `http://casper.berkeley.edu/`
[14] A. Parsons, arXiv:0809.2266 [astro-ph] (2008)
[15] A. R. Parsons, arXiv:0904.2334 (2009)
[16] A. Youssef and B. Arden, *Microprocess. Microsyst.*, **16**, 3 (1992)
[17] H. S. Stone, *Computers, IEEE Transactions on*, **C-20**, 2 (1971)
[18] K. Hirabayashi, T. Yamamoto, S. Matsuo, and S. Hino, *Appl. Opt.*, **37**, 2985 (1998)