High Throughput Optical Algorithms for the FFT and Sorting via Data Packing

Keren Bergman§ Geoffrey L. Burdge∗ David A. Carlson† Neil B. Coletti†
Harry F. Jordan‡ Rajgopal Kannan¶ Kyungsook Y. Lee¶ Phillip R. Merkey⋆
Paul R. Prucnal§ Coke S. Reed†,§ Douglas E. Straub‡

†Center for Computing Sciences
17100 Science Drive
Bowie, MD 20715

‡Electrical and Computer Engineering Dept.
University of Colorado
Boulder, CO 80309

§Electrical Engineering Dept.
Princeton University
Princeton, NJ 08544

¶Computer Science Dept.
University of Denver
Denver, CO 80208

∗Laboratory for Physical Sciences
University of Maryland
College Park, MD 20742

⋆USRA CESDIS
Goddard Space Flight Center
Greenbelt, MD 20771

Abstract

This paper demonstrates that the potential for very high speed computation provided by optical technology can be achieved for important computational problems including the fast Fourier transform and sorting. First a programming model that captures the 'time-of-flight' characteristics of optical fibers and switching elements is presented. Then it is shown that the FFT and the radix sort algorithms can both be reduced to the computational kernel of 'packing' intermediate data results that are continuously moving through an optical fiber. New algorithms are developed for the general packing problem, and the details of how to implement them in optics are presented. The resulting systems have the potential for operating at clock rates that are two or more orders of magnitude higher than conventional computers.

1. Introduction

Optical technology promises a substantial increase in raw clock speed over conventional computing devices. However, the programming model imposed by these devices is not necessarily suited to algorithms that have been designed and tuned to run well on existing machines. The high latency for memory access (relative to clock speed), the necessity to stream data, and the high cost of individual logic gates is reminiscent of early electro-mechanical devices. The fact that the entire optical computer must adhere to time-of-flight scheduling constraints [7] offers a difficult programming challenge.

This paper presents an optical implementation of a new algorithm that packs data in an optical stream. Furthermore, it is shown that the computation in optics of both the fast Fourier transform (FFT) and the radix sort contain packing as a kernel operation. Thus, the packing algorithm solves the time-of-flight problem for the FFT and the radix sort, i.e. it reduces the problem of implementing these algorithms using optical hardware to standard computer science techniques.

The paper is organized as follows. Section 2 gives an introduction to optical devices and shows how their characteristics determine the programming model. Sections 3 and 4 review FFT algorithms and the radix sort, showing that they share a similar data movement pattern which reduces to the data packing problem when time-of-flight constraints are imposed. Sections 5 and 6 give a detailed description of the implementation of the algorithms including the use of the packer.

2. Optical Technology Overview

Recent advances in optical technology have produced data channels and switching elements both with incredibly high throughput rates. An interesting question is whether a high speed computational device can be built using this technology. That is, can an algorithm be mapped onto an architecture comprised solely of optical or opto-electronic parts. In this section, the characteristics of the devices are examined along with the programming model that they imply.

The first important consideration when designing an optical algorithm is the time-of-flight nature of data movement in the fibers. The data processed by the algorithms here is photonic, having been injected into an optical fiber using a laser. Once the data has been injected into a fiber, it remains moving continuously, through both the fiber and any switching elements it may encounter along the way. Thus the programming model must be RAMless - the only memory in the system is the delay-line memory of an optical fiber. It must also ensure that the desired data is accessed and switched as it flies by, hence the term time-of-flight. Furthermore, when there is a choice of fibers to switch the data onto, the fiber not receiving the data will have a logical blank inserted onto it. At certain points in the computation, these blanks may have to be removed in order to keep the data size manageable.

The second important consideration is the expense of the optical switching elements along with their limited functionality and fan-in, fan-out capabilities. The production of integrated optical devices is in its infancy stage. Only a few switches can be integrated on a common substrate and substantial costs are expended in doing so (an integrated device containing only five to ten lithium niobate optoelectronic switches can cost up to ten thousand dollars). Thus algorithms must be designed that minimize the number of switching elements and that take into account their restrictions.

One commonly used opto-electronic switching element is the lithium niobate gate. It has two fiber optic data inputs, two fiber optic data outputs, and an electronic control input [11, 15]. In an all optical system, the control input is converted from optics to electronics just before being fed to the gate. The functionality of the gate is a 2×2 crossbar switch. If there is no electronic input at the control port then the switch is in the cross position, if there is an electronic input, then the switch is in the straight through position.

Data can pass through a lithium niobate gate at the terabit rate and the gate can switch at below a nanosecond (hero experiments have been performed where these devices switch at rates up to 40 GHz [11]). Therefore the optical algorithms presented here will be designed to use switching elements whose throughput is much higher than the switching rate. That is, the data passes through the gates at the bit rate but is switched at the word rate, and a blank space is required between words to allow the switches to transition from one state to the other. Furthermore, when an optical signal passes through the gate there is a certain amount of loss. The optical implementations given here are designed to minimize the number of the gates and pass signals through each gate only once so that only a small number of amplifiers and regenerators are required during the course of the computation.

All of the optical algorithms and implementations presented in this paper will be based on these lithium niobate switching elements. For the optical sorting algorithm, a second optical component that extracts a bit from each word in the data stream is required. One component that has this capability is the TOAD (Terabit Optical Demultiplexing Device) developed at Princeton University [12, 13]. The key property of the device is that it is capable of removing a single bit from a high data rate stream. However once it has removed a bit, it must remain inactive for a time until it relaxes and is ready to remove another bit.

A second component with the capability to demultiplex a high data rate signal is the NOLM (nonlinear optical loop mirror) or Sagnac loop [6, 9]. It also operates as a switch, but in contrast to the lithium niobate gate, it is all optical and has a much higher switching rate. NOLMs have been fabricated that switch at the rate of 100 GHz [9], and are believed to have the potential for switching at a rate approaching 1000 GHz. Hence they can easily operate at the bit rate of the photonic data flowing through them. Data enters the NOLM and is split into signals that counter propagate around a loop. A control signal causes either the signals to add back together so that the original signal is output, or to cancel in which case the control signal is output. There is a latency of about five microseconds between paired inputs and outputs due to the length of the fiber loop. For latency tolerant algorithms such as the FFT and radix sort, this does not affect the overall throughput of the system. In fact, the TOAD is a particular type of NOLM.

3. FFT Overview

In this section, an algorithm for the FFT is described and an overview of how it can be implemented using optical technology is given. However, the details of the implementation are deferred until later in the paper so that the relationship between the optical FFT algorithm and the optical sorting algorithm can first be established.

The dataflow graph for the well known 'out-of-place' FFT algorithm is illustrated in Figure 1 for eight inputs. Notice that the interconnection pattern for the out-of-place FFT is the same at each stage. This makes it a good candidate for implementation in hardware [14], since a savings can be made by using the same hardware to perform successive stages. However, the memory requirement is twice the width of a stage, since outputs of a stage cannot directly overwrite inputs without producing incorrect results. As a contrast, the in-place FFT algorithm [4] has a different interconnection pattern at each stage, but its memory requirement is only the width of a stage. Also, since the Fourier transform is its own inverse, the out-of-place network can be reversed (start the computation at the outputs of the graph and proceed towards inputs) and a Fourier transform will still be performed. This dataflow graph for eight data points is also illustrated in Figure 1.

As a precursor to an optical FFT algorithm, an algorithm for computing the FFT that uses four tapes and the reversed out-of-place dataflow graph is now discussed. It is similar

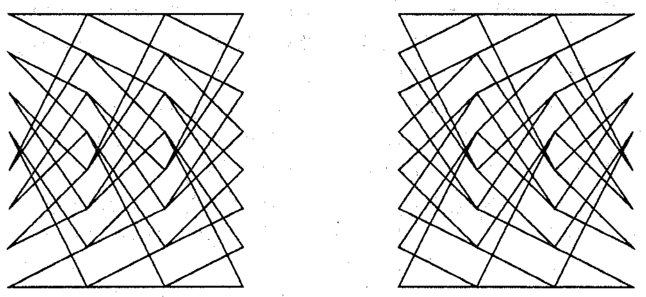

Figure 1. The dataflow graph for the out-of-place FFT

to the balanced tape merge discussed in [10], and can be used when the number of inputs to the FFT is much larger than the size of the memory (while no longer true, this was typically the case in the early days of computing).

The four tape FFT algorithm works as follows. To compute a stage of the FFT, start with the first half of the inputs on tape one and the second half of the inputs on tape two (assume the previous stage has produced its outputs in this format). Now, read the first two inputs from tape one, compute the two outputs of a butterfly, then write the left output to tape three and the right output to tape four. Continue this until the middle of tape three (four also) is reached. At this point, switch reading the inputs of the butterflies from tape one to tape two. When the end of tape two is encountered, the computation of a stage is complete, and the next stage simply begins with its inputs on tapes three and four. Refer to Figure 2 for an illustration of this process.

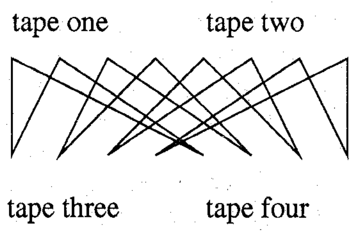

Figure 2. A stage of the four tape FFT algorithm

The placement and movement of data on a tape is roughly analogous to what happens in an optical fiber with one major exception. When programming for a tape, the algorithm designer has control over starting and stopping the movement of the data. However, once data has been injected into an optical fiber, it remains moving continuously. This is equivalent to a tape that must advance one position at each time step. Algorithms designed with this restriction in mind are referred to as time of-flight [7].

An overview of an FFT algorithm suitable for implementation on a special purpose optical computer will now be given. The algorithm carries out the FFT dataflow graph in a manner similar to the four tape FFT algorithm. Consider how the four tape FFT algorithm must be modified if the tapes are required to move one position during each time step. A problem occurs when both inputs of a butterfly are read from the same tape - the output tapes continue to move while waiting for the next pair of inputs to be read, which causes a blank to appear between consecutive outputs. Note that reading the inputs in parallel does not solve this problem. But it can be overcome if, after the output tapes are produced, a separate computation is done that removes the blanks between the outputs in preparation for the next stage.

The optical FFT algorithm explained in detail later in this paper has the same data movement pattern as the four tape FFT algorithm, along with a subsystem referred to as the 'packer' for removing blanks between data that is constantly moving along optical fibers. It will be shown that this packer can be designed using a linear number of optical fiber loops and only a logarithmic number of optical switching elements. Recall that minimizing the number of optical switching elements is important due to their expense.

4. Sorting Overview

The second problem that optical implementations are developed for in this paper is sorting. Similar problems involving network applications and time slot interchangers have been previously studied in [2, 7, 8].

It is well known that a dataflow graph for the bitonic sorting algorithm can be constructed recursively using the FFT dataflow graph [1, 5]. Having just presented a sketch of how the FFT algorithm can be implemented using optical fibers and switching elements, it is straightforward to extend this to yield an optical sorting device. However, since the basic operation in a bitonic sort is a comparison of an entire data word, the optical implementation turns out to be expensive to build due to the large number of optical switching elements required by a comparator. See [3] for details.

Other sorting algorithms, while just as efficient as bitonic sort, are at first glance not as easy to map into hardware in general (optics in particular) since their corresponding dataflow graphs are not regular. Here, it is show that the radix sort can be implemented efficiently in optics even though it has an irregular dataflow graph. The main reason for choosing radix sort is that comparisons are done on the bit level rather than the word level, which greatly reduces the number of optical switching elements required.

A radix sort proceeds in stages as does the FFT algorithm. Each stage operates on a different bit position within a word, beginning with the low order bit position and ending with the high order bit position. A stage simply separates the current partially ordered list of data words into sublists, one with zeroes and the other with ones in bit position j, then concatenates them. Each sublist must maintain the ordering of data in the input list.

An overview of the radix sort implementation using optical fibers and switching elements is as follows. To compute a stage, first assume that there is an input fiber and two output fibers, and that the input fiber is densely packed. Each data word will be placed on output fiber zero or output fiber one depending on the value of the jth bit (an optical component will examine the jth bit and make this decision). At the end of this process each output fiber will contain a sublist of the inputs with variable numbers of blanks between each data word. The blanks arise from the fact that while a data word is being written to one output fiber, nothing is being written to the other fiber and the data on it continues in motion. In order to start the next stage, these blanks must be extracted and the lists must be concatenated. A subsystem referred to as the 'packer' will be designed to perform this computation, again using a only small number of optical switching elements.

There is a fundamental point to be noted here. Both the optical FFT and the optical radix sort share the kernel operation of packing data that is in continuous motion along an optical fiber. The only significant difference is that in the FFT the blanks form a regular pattern, whereas in the radix sort the blanks are irregular. The major contribution of this paper is to give a new algorithm that solves the packing problem in its full generality, and to show how it can be implemented using optical fiber loops and a small number of optical switching elements.

5. Optical Implementation of an FFT

This section presents the details of an implementation of the FFT algorithm using optical fibers and lithium niobate gates. The dataflow is the main emphasis here, i.e. the algebra on the elements of the data can essentially be ignored. Start with a mathematical description of the data as it appears on an optical fiber.

Let $X(0) = x_0(0), x_1(0), \ldots, x_{n-1}(0)$ denote the original data sequence to be transformed. Suppose that there is an integer k such that $n = 2^k$. The FFT algorithm produces the n long sequences $X(1), X(2), \ldots, X(k)$ so that $X(k)$ is the Fourier transform [4]. Although in practice each data word $x_i(j)$ may be composed of multiple signals in the fiber, it will be regarded as a single entity occupying a unit amount of length. Thus for simplicity, assume that at time zero, $x_i(0)$ is at position i in the fiber, and that the data moves through the fiber one unit per time step, i.e. if a data item z is at position p at time t, then z is at position $p + \tau$ at time $t + \tau$.

The optical implementation of the FFT algorithm must contain hardware that forms the outputs of a butterfly. The only constraint on this hardware is that it be fast enough or parallel enough to keep up with the dataflow, since the butterfly operation is latency tolerant. Thus, this component is not required to be optical. But if the optical signal is at a high data rate, and the butterfly unit is electronic, it may need to be pipelined.

Assume that the butterfly hardware reads from position 0 of an input fiber and after a delay Δ writes the butterfly outputs to positions 0 and n of an output fiber. These writing positions can be physically close together by placing a coil in the fiber. At times 0 and 1, $x_0(0)$ and $x_1(0)$ are read from the input fiber. The butterfly computation is performed, which takes time Δ to produce two intermediate results that need to be placed on the output fiber. So at time Δ, $x_0(1)$ is inserted into position 0 of the output fiber and $x_1(1)$ is inserted into position n of the output fiber. Next at times 2 and 3, $x_2(0)$ and $x_3(0)$ are at the reading stations. At time $\Delta + 2$, $x_2(1)$ is inserted into position 2 and $x_3(1)$ is inserted into position $n + 2$ of the output fiber. Continuing to times $n - 2$ and $n - 1$, $x_{n-2}(0)$ and $x_{n-1}(0)$ are read, and at time $\Delta + n - 2$, $x_{n-2}(1)$ and $x_{n-1}(1)$ are inserted into positions $n - 2$ and $2n - 2$ respectively. This finishes the production of the sequence $X(1)$.

Notice that the absence of writes at time $\Delta + 1, \Delta + 3, \ldots$ has produced blanks at positions $1, 3, \ldots$ and $n + 1, n + 3, \ldots$. It is not possible to directly produce the output fiber without blanks because the data is flowing into each half at a rate that is half as much as the data flow from the input fiber. Thus, before this data can be fed back through the logic to produce $X(2)$, it must be fed through a subsystem that removes the blanks.

5.1. The Packer

At this time it would be convenient if there were no blanks between the data items $x_i(1)$ and $x_{i+1}(1)$; then this stream could be fed back through the computational units to produce $X(2)$. A subsystem that receives the unpacked data and produces data in the same order except with no blanks between adjacent data items will now be described. This subsystem is referred to as the packer.

The initial state of the data can be viewed as n words in even positions and n blanks in odd positions. The first stage of the packer will produce a sequence that has the format: pairs of words separated by pairs of blanks. Thus the output is equivalent to the input except that the word size (and blank size) has doubled. This will also be true for subsequent stages, so that the only difference between stages is the word size that must be handled.

A stage of the packer is composed of an optical switching device (e.g. a lithium niobate gate) and an optical fiber loop. The loop is used to delay some of the data elements entering the stage, and the switch is used to determine which ones to delay. The size of the loop is determined by the data element size - the loop in stage j holds 2^j words.

The data elements flow through a stage of the packer as follows. Denote the data elements as $d_0, d_1, \ldots$ If i is even, then d_i will be delayed one time unit so that it appears consecutively with d_{i+1} in the output stream. This is done by switching d_i onto the loop when it enters the stage. It stays there for one time unit after which it takes the place of the blank between d_i and d_{i+1} in the output stream. Then d_{i+1} is switched straight through so that d_{i+1} immediately follows d_i in the output stream. Notice that two blanks will appear in the output stream between d_{i+1} and d_{i+2} , one when the blank in the input stream between d_{i+1} and d_{i+2} is switched straight through, and the second is 'produced' when d_{i+2} is switched onto the loop. Figure 3 shows the dataflow through a stage of the packer.

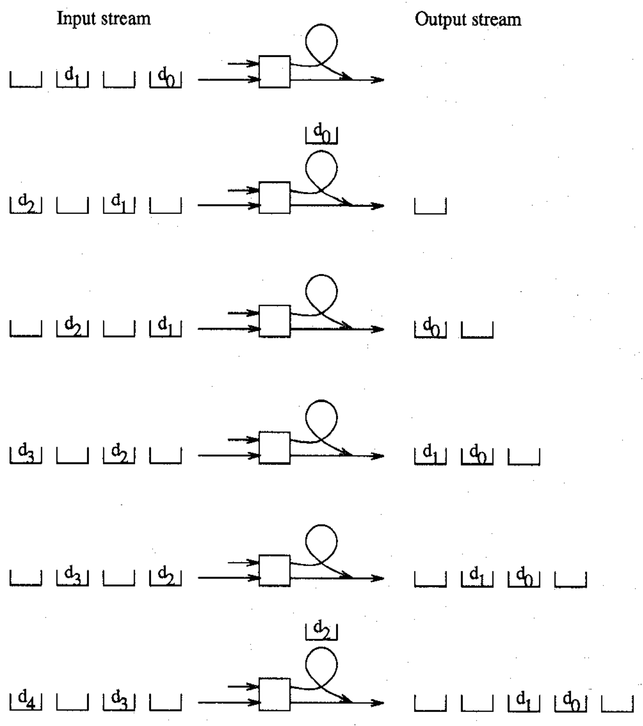

Figure 3. The dataflow through a stage of the packer

The switching element in a stage of the packer simply controls the data elements being placed on the loop. It is in the cross state only when a d_i, i even appears in the input stream, i.e. it switches at the rate that even data elements appear. Note that the control for a stage is straightforward because the data elements and blanks have a regular format. A stage simply delays all the even elements, and the packer iterates this over successive stages that double the word size. In the forthcoming discussion of sorting, the blanks will be irregularly placed amongst the data elements. This will lead to a packer with the same overall fiber optic loop structure, but with more complicated circuitry for deciding which data elements are fed through which loops.

From the preceding discussion, it should be obvious that after passing through a stage, the data has the format of two packed elements, two blanks, two packed elements, two blanks, etc. The order of the data is preserved and k stages are required to produce an output stream that is densely packed (no blanks between words). The last stage actually produces n packed words followed by n blanks, but the blanks are just ignored by the computations that follow. Also, to pack a data set of size 2^k, only k switches are required, which is logarithmic in the data size. The number of unit length optical fiber loops required is $1+2+\ldots+2^{k-1} = 2^k - 1$. The packer is illustrated in Figure 4 for a data set of size eight.

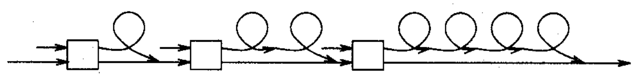

Figure 4. The packer component of the FFT

There are a few things worth noting here about this optical implementation of the FFT. First, the beginning of the stream can enter the second stage as soon as it emerges from the first stage so that the beginning of the data is entering the second stage just as the second word of data is entering the first stage. In this way, the computational units can begin producing $X(2)$ from $X(1)$ as soon as they are finished producing $X(1)$ from $X(0)$. Also, because lithium niobate gates have a certain amount of loss and introduce a certain amount of noise, it is necessary to amplify and regenerate the signal at certain points in the process. The details of the amplifiers and regenerators are not covered in this paper, except to say that only a small number are required. There is another slightly more complicated design that requires only k gates for a data set of size 3^k and has the same throughput. A drawback of this design is that the data must make more than one pass through some of the gates and loops. This introduces additional signal degradation and thus requires more amplification/regeneration. Finally, an optical FFT based on the forward out-of-place algorithm would be very similar to the one presented here. Instead of a packer subsystem, it would produce a data stream in dense format that required unpacking. The details are left to the reader.

6. Optical Implementation of a Radix Sort

This section continues the presentation of optical algorithms with the details of an implementation of the radix sort using optical fibers and lithium niobate gates. The radix sort is a deterministic algorithm that sorts $L = 2^l$, m-bit data items in m passes. In the jth pass, $0 \leq j \leq m - 1$, the list is partitioned into two sublists according to the value of the jth bit of each number. Without changing their relative order all the numbers with a zero in the jth bit are assembled into a sublist (the 'zero' sublist) as are all the numbers with a one in the jth bit (the 'one' sublist). Then these two sublists are concatenated, the zero sublist is followed by the one sublist, to complete this jth pass. After m such passes the list will be in ascending order.

In what preceded, all of the devices switched at the word rate. For the sort, there are two devices that must switch at the bit rate. Either the TOAD (Terabit Optical Demultiplexing Device) or the NOLM (nonlinear optical loop mirror) can be used. Recall that the TOAD is capable of removing a single bit from a high data rate stream, but after it has removed a bit it must remain inactive for a time until it relaxes and is ready to remove another bit. This is not a problem for this application because only one bit (the radix bit) is removed per word per pass. The NOLM is a classical Sagnac device [6, 9]. It would also work in this application because the algorithm is latency tolerant and therefore, the long Sagnac loop would not cause a degradation of throughput performance.

Corresponding to the two operations in each pass, the optical sorter has two main components: 1) the 'splitter' and 2) the 'packer'. The splitter is the component that separates the zero sublist from the one sublist. The packer is responsible for concatenating and assembling the sublists into the new list for the next pass.

To form an optical sorting system, one splitter and two packers are connected as shown in Figure 5. Two packers are used so that all components can be kept busy at all times. The data stream enters the splitter on the input line. The data moves from the splitter to the 'even packer' and its delay line. The data flows from the even packer back to the splitter, thus completing an even pass of the algorithm. In an odd pass of the algorithm, the data flows through the splitter to the odd packer and back to the splitter. This completes two passes of the algorithm and one cycle of the system. A more complete description of the flow of data throughout the system will be given after the discussion of each of the components.

6.1. The Splitter

The splitter performs the following operation on the jth pass. The data stream runs through a coupler so that there

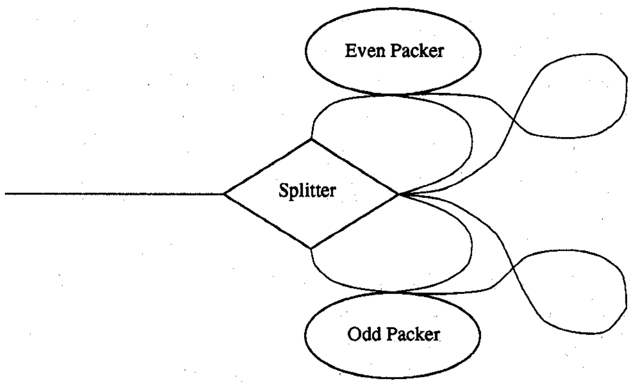

Figure 5. The splitter and packer components of the optical radix sorter

are two copies of the stream. One of the copies is sent through a TOAD/NOLM that picks off the jth bit and sends this bit ahead to be converted to electronics. After the one bit is read, this copy of the data is discarded. The extracted bit drives the control port of a lithium niobate gate. The other copy of the data is sent to one of the data input ports of the gate.

One output of the gate is connected directly to the packer, the other is connected to a delay line. If the jth bit of the word in question is a zero, then the switch is set to the straight through position. The word is sent directly to the packer and a blank word is sent to the delay line. Conversely, if the jth bit is a one, the switch is set to the cross position. A blank word is sent to packer and the data word is sent to the delay line. The delay line is exactly L words long so that it can hold the entire unpacked 'ones' sublist. As the last word leaves the splitter exchange switch, the first word put into the delay line is being output from the delay line. At this time the input to the packer is switched from the splitter to the delay line. This implements the step of concatenating the 'zero' sublist to the 'one' sublist. The result is a string of length $2L$ that contains L words and L blanks. This is similar to the situation that occurred in the FFT. However the sort presents a more difficult problem in that the blanks in the data are not in any regular order.

6.2. The General Packer

The packer will take the $2L$ long stream of data and blanks as input and output a stream consisting of L blanks followed by L words. Adopt the convention that at time zero, the first word or blank enters the packer from the splitter. Then at time $L - 1$ the last output (word or blank) from the splitter enters the packer. At time L the first word or blank leaving the delay line is switched into the packer. Also at time L, the first word leaves the packer. For the next $L - 1$ time steps, the rest of the delay line is fed into the packer and the rest of the $L - 1$ packed words leave the packer. Thus the packer must provide each word a path that is 0 to L words long. The challenge is to do this without a global view of the data stream and to do it with minimal logic.

Each data word has one header bit indicating whether the word is full or empty. As the data stream enters the packer it passes through a second TOAD/NOLM that sends forward the bit indicating whether each data slot is full or empty. There are a total of $l + 1$ lithium niobate gates in sequence that either pass data straight through or send it through delay loops, see Figure 6. This is almost the same in structure as the FFT packer, except now the first two loops will have a length of one word (this is necessary so that the entire packer can provide a delay of L units), and after the second loop, each subsequent loop will be twice the length of its predecessor. As is the case of the FFT packer, each data word will go through a subset of the delay loops and no data word will go through any delay loop twice. It will be the case that if there is a full data word at position zero at time zero (this would be the case if first word entering the splitter had a zero in the jth bit) then this word will go through all of the data loops so that it would be shifted back a total of $1 + 1 + 2 + 4 + \cdots + 2^{l-1}$ slots which is a total of 2^l slots. It would therefore leave the packer in slot L. Notice that the total amount that a word should be shifted back depends only on the number of empty slots that pass in front of it in the original stream.

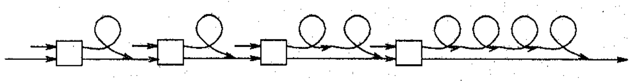

Figure 6. Delay loop structure of the packer

Figure 7 tracks a sequence containing eight full and eight empty slots as the data enters stages 1,2,3 and 4 and also illustrates the data as it leaves stage four. In fact, one could spread out the stages of the packer so that the strings were produced between them. The figure depicts the stream moving from left to right, so that position 0 is the right most slot and position $2^4 - 1$ is the left most slot. The example illustrates the fact that the first delay loop insures that there is a blank in position zero, the second delay loop insures that all the empty slots are in groups of length divisible by 2, the next loop produces empty groups of length divisible by 4 and so on. As before, the number of consecutive data words in a cluster may be arbitrary.

The logic that drives the gates is shown in Figure 8 and operates as follows. Each gate has an associated logic unit,

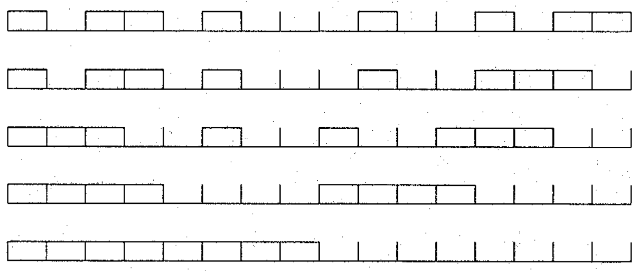

Figure 7. Sequence of data movements in the optical sort

which may be optical or electronic. The gates and their associated logic units are numbered from 1 to $l + 1$. The TOAD/NOLM sends a signal forward to logic unit one each time an empty slot passes. Note that this is the only device in the packer that looks at the data stream. The first gate starts off in the cross position (the position that sends data through its one long loop). It remains in that position until its logic unit receives a signal from the control line indicating that a blank has passed. It then moves to the straight through state and remains in that state until the entire data stream has passed. It then resets to the cross position for the next pass of the data on the next radix bit. The timing is such that it transits from the cross state to the straight through state in the middle of the first blank slot. When the first logic unit receives its second signal indicating that an empty slot has passed, it sends a signal to the second logic unit. The second gate starts out in the cross position (the position that sends data through its one word long loop). When it receives a signal indicating that the second empty slot has passed, it switches to the straight through position. It does this switching in the middle of the second empty slot. The process continues in this fashion, with the first logic unit sending signals to the second logic unit every time it receives a signal indicating that an empty slot has passed. Each time the second logic unit receives a signal, it switches the second gate in the middle of an empty data slot.

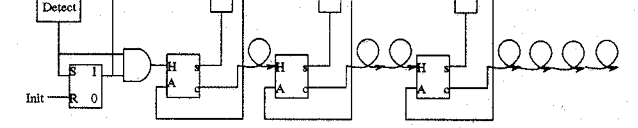

Figure 8. The packer line and its control

The second logic unit keeps track of the parity of the number of signals that it receives from the first logic unit. When the second logic unit receives its second signal from the first logic unit, it sends a signal to the third logic unit. The third logic unit uses this signal to switch the third gate in the middle of the second pair of empty slots (it switches between empty slots three and four). The second logic unit sends a bit to the third logic unit every other time it receives a signal from the first logic unit. The configuration is replicated down the entire cascade of gates and logic units. That is, the ith logic unit receives signals from the $(i-1)$th unit and passes the parity of the signals it receives to the $(i+1)$th unit. Also, the gates toggle in the middle groups of empty slots (of increasing length). The signals that pass among the logic units can be thought of as the carry bits for a counter that counts the number of blank slots that enter the packer. In fact, the ith bit of the sum of the number of empty slots arrives at the ith logic unit at the proper time for the ith gate to be set appropriately.

Notice that gates that are further along in the sequence switch in the middle of progressively longer blanks; therefore, the system is more and more jitter tolerant further down the line. Notice also that the switching rate of the gates is slower further down the line. For this reason, with the possible exception of the first few logic units, all of the logic can possibly be made out of electronics.

The packer described in this paper can be viewed as providing each word in the data stream with a variable length delay. Thus it resembles the Tunable Optical Delay described in [12, 13].

6.3. Combining the Splitter and the Packer

In order to completely sort a string of data, it is necessary to look at each bit of each word. That is, the packed data must be fed back through the splitter for each radix bit. Figure 9 shows a full period of the data moving from one packer through the splitter and into the other packer. The heavy solid line represents packed data while the heavy dashed line is the unpacked data stream.

The lines from the splitter to the delay line and the packer add a constant to the overall latency of the loop but are not involved in the variable delays required for data packing. Stepping through Figure 9 we have:

1. At $t = 0$, the data stream enters the splitter for the first time.

2. At $t = L/2$, half the data has gone through the splitter. The even packer is half full of blanks and 'zero words' and the even delay line is half full of blanks and 'one words'.

3. At $t = L - 1$, the last data word is processed by the splitter, so at $t = L$ the first packed word leaves the even packer and enters the splitter.

4. At $t = 3L/2$, half the packed words have left the even packer, been split and have filled the odd packer and delay line with unpacked words and blanks.

5. At $t = 2L - 1$, the last word out of the even packer is split. Thus at $t = 2L$ the odd packer and delay line are filled and the first packed word leaves the odd packer. This completes the cycle.

The cycle continues until all m passes of the sort have been completed.

Notice that this configuration allows for the time-of-flight passing of data through the splitter and enjoys full utilization of all the components. An important consequence of this is that the high speed device that examines the radix bit (the TOAD/NOLM in the splitter) is busy all the time. Since the algorithm looks at each bit exactly once with this device, the total time to sort a stream is equal to the total number of bits in the stream times the rate at which this TOAD/NOLM is picking bits out of the stream.

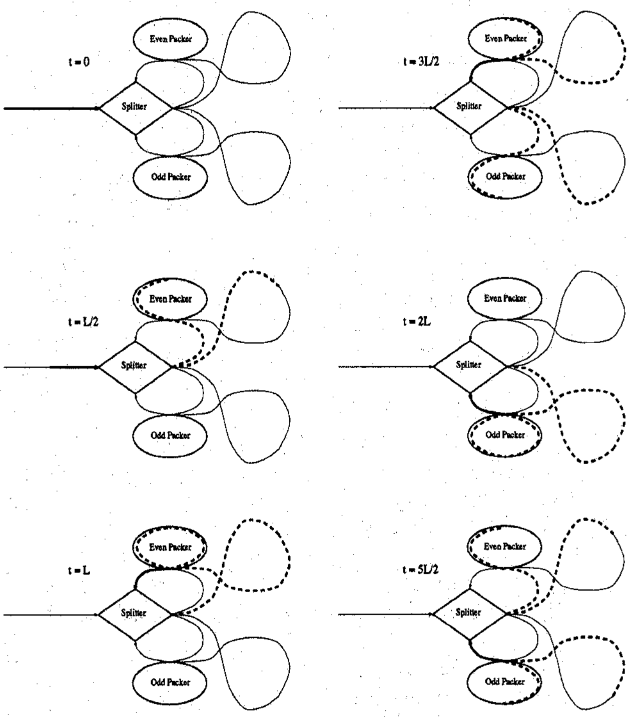

Figure 9. A full period of data movement through the packers

7. Conclusions

It has been shown from an algorithmic perspective that optical systems with the common kernel of data packing can be constructed to perform an FFT and a radix sort at very high throughput rates. New algorithms that adhere to time-of-flight programming constraints were designed for the data packing problem. Implementations using optical fiber loops and a minimum number of optical switching elements were presented.

Hardware details were avoided unless directly related to the flow of data and the bit rate versus the word rate of the systems. For example, the arithmetic unit of the FFT system could be designed using heavily pipelined or parallel electronics instead of optics without affecting the overall flow of data in the algorithm. Likewise, in the sorter, the technology for the logic that drives the 2×2 switches in a packer was less important than the fact that each stage counted the parity of the groups of blanks that passed through the previous stage, and that the entire cascade of logic only required a single look at the data stream.

References

[1] K. E. Batcher. Sorting networks and their applications. *AFIPS Conference Proceedings, 1968 SJCC*, 32:307–314, 1968.

[2] D. J. Blumenthal, K. Y. Chen, J. Ma, R. J. Feuerstein, and J. R. Sauer. Demonstration of a deflection routing 2x2 photonic switch for computer interconnects. *IEEE Photonics Technology Letters*, 4:169–173, 1992.

[3] N. B. Coletti. The TAO of optics: Sorting, FFT's and other transforms on the transpose architecture for optics. Technical report, IDA Center for Computing Sciences, 1994.

[4] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19:297–301, 1965.

[5] R. Fuller. Formal derivation of interconnection schema for parallel computation. *Congressus Numerantium*, 70:249–254, 1990.

[6] A. Huang et al. Sagnac fiber logic gates and their possible applications: a system perspective. *Applied Optics*, 33(26):6254–6267, 1994.

[7] H. F. Jordan, V. P. Heuring, and R. F. Feuerstein. Optoelectronic time-of-flight design and the demonstration of an all-optical, stored program, digital computer. *Proceedings of the IEEE, Special Issue on Optical Computing*, 82(11):1678–1689, 1994.

[8] H. F. Jordan, D. Lee, K. Y. Lee, and S. V. Ramanan. Serial array time slot interchangers and optical implementations. *IEEE Transactions on Computers*, 43(11):1309–1318, 1994.

[9] S. Kawanishi, H. Takara, K. Uchiyama, T. Kitoh, and M. Saruwatari. 100 Gbit/s, 50 km optical transmission employing all-optical multi/demultiplexing and PLL timing extraction. *Conference on Optical Fiber Communication (OFC 93)*, 1993. paper PDP2.

[10] D. E. Knuth. *The Art of Computer Programming Vol 3: Sorting and Searching*. Addison-Wesley, New York, NY, 1994.

[11] S. Korotky, 1994. Private communication, AT&T Bell Laboratories.

[12] P. R. Prucnal. Photonic fast packet switching. In *Photonics in Switching Vol II*, page 303, New York, NY, 1993. Academic Press.

[13] J. P. Sokoloff, P. R. Prucnal, I. Glesk, and M. Kane. A terahertz optical asymmetric demultiplexer (TOAD). *IEEE Photonics Technology Letters*, 5(7):787–790, 1993.

[14] H. S. Stone. Parallel processing with the perfect shuffle. *IEEE Transactions on Computers*, 20(2):153–161, 1971.

[15] United Technology Photonics, Part No. APE-YBBM-1.5-18-T-02.