A model of computation for Fourier optical processors

Thomas J. Naughton

Department of Computer Science, National University of Ireland/Maynooth, Maynooth, Ireland

ABSTRACT

We present a novel and simple theoretical model of computation that captures what we believe are the most important characteristics of an optical Fourier transform processor. We use this abstract model to reason about the computational properties of the physical systems it describes. We define a grammar for our model's instruction language, and use it to write algorithms for well-known filtering and correlation techniques. We also suggest suitable computational complexity measures that could be used to analyze any coherent optical information processing technique, described with the language, for efficiency. Our choice of instruction language allows us to argue that algorithms describable with this model should have optical implementations that do not require a digital electronic computer to act as a master unit. Through simulation of a well known model of computation from computer theory we investigate the general-purpose capabilities of analog optical processors.

Keywords: optical information processing, analog optical computing, computational complexity, models of computation, optical computing architectures and algorithms, Fourier transform processor, general-purpose optical computer, roadmaps for optics in computing

1. INTRODUCTION

The discovery of efficient algorithms is important to the development of optical information processing. Computational complexity analysis[1,2] is the standard mathematical tool for classifying algorithms according to their performance characteristics. Primarily, computational complexity is a measurement of the amount of resources an algorithm requires. The first stage of this analysis requires one to define a model of computation: essentially a list of assumptions made about any physical processor running an instance of the algorithm. A simple model of the processor is chosen that hides various technical details while still making accurate predictions over all final implementations. Models of computation are usually ideal (reliable and error free), simple (have a limited instruction set of basic operations), unrestricted (have unlimited resources, e.g. memory) and abstract (hide technical details, e.g. memory management overheads). In order to measure and compare the efficiency of optical information processing algorithms we therefore require a theoretical optical processor that is straightforward to analyze and yet yields meaningful results by corresponding closely to physical implementations. The ability of Fourier processors to take constant-time Fourier transforms and constant time image multiplication (but little else) requires us to construct a specialized model of computation.

A model of computation is only part of the story. In the second stage of computational complexity analysis, we define suitable complexity measures, thus giving units (dimensions) to our measurements. The standard units describe space (typically, how much memory is used) and time (how long will the computation take). For an unconventional model of computation, such as that describing an analog optical processor, we may require additional measures, to cost such implementation concerns as required number of optical passes, spatial light modulator (SLM) resolution, SLM dynamic range, or photon budget.

Previous complexity studies in this area have sometimes acknowledged the lack of suitable models and used a conventional model[3] or simply added a unit-time Fourier transform instruction to a theoretical model of a digital electronic computer.[4] We explain why these and other approaches are not suitable for analyzing analog information processing algorithms and present a novel and simple theoretical machine that captures what we believe are the most important characteristics of an optical Fourier transform processor.

The benefits of this paper are two-fold. Firstly, our theoretical machine is shown to be suitable for the representation (and subsequent analysis) of a range of Fourier optical information processing algorithms. Secondly, the machine is a theoretical description of what could be a general-purpose analog optical computer that does

Further author information. Fax: +353-1-7083848 Email: tomn@cs.may.ie

not require a digital electronic master unit. As such, we hope to illuminate another avenue of research in the investigation of whether a primarily-optical general-purpose computer can be built or not.

In Sect. 2 we expand on our motivations and review existing optical models of computation. In Sect. 3 we describe our model. In this idealistic model, the basic data unit is a continuous image of unbounded resolution. The model has image copying functionality, constant-time Fourier transformation/multiplication/complex conjugation, and minimal flow control in the form of unconditional branching (`goto`). We define a grammar for our model's instruction language and use it in Sect. 4 to express a selection of well-known filtering and correlation techniques. In this section we also suggest suitable computational complexity measures that could be used to analyze any coherent optical information processing technique, described in the model, for efficiency. In Sect. 5 we look at our model in the light of computer theory. Since this very restricted instruction language has neither conditional branching (`if`) nor iteration (`for`, `while`) we argue that algorithms describable with this model should have optical implementations that do not require a digital electronic computer to act as a master unit. We also investigate this simple processor's capabilities for general-purpose computation. In a different approach to that of previous research[5] we use our model to simulate a standard (albeit non-universal) model of computation from computer theory.

2. EXISTING MACHINE MODELS

Before it became obvious that we needed a new model of computation to analyze the algorithms of our Fourier optical processors[6,7] we looked at existing models from computer theory and optical information processing literature. Most were found unsuitable due to their discrete nature.

2.1. Random-access machine

A natural choice for a Fourier model of computation is the random-access machine[8] (RAM), an idealization of the digital electronic computer, with one extra operation: a constant-time FT. However, such a choice would have some drawbacks. Firstly, it would be difficult to encode images of variable resolution in the register-based memory. Discrete raster images stored in a digital machine invariably have fixed dimensions and resolution. Any encoding that did allow variable resolution (such as a vector format, or a discrete quad-tree encoding) would require a specially tailored FT operation. Conversion from a discrete variable resolution format to a raster format of arbitrary resolution, prior to Fourier transformation, would of course be possible but would cost time proportional to the number of pixels in the image. This brings us to the second drawback.

In order to permit constant-time image move and copy (or format conversion) operations we would need to add extra functionality to our machine. This would mean significant changes to our standard RAM model and ultimately forfeit the advantages of using a predefined and simple textbook model.

Thirdly, by using a RAM as our model of computation we run the risk of restricting the range of possible implementations to those with digital electronic master units. By incorporating into our model all of the functionality of a RAM we are forced to build a physical machine that has all of this functionality. We would ideally like a model that incorporated Fourier operations with a minimum of control flow mechanisms in its instruction set. The pragmatic reason for this is that we hope eventually to construct a physical implementation of such a model and it is not immediately obvious how to effect conditional control flow mechanisms (such as branching or iteration) in a purely FT-based processor. Even worse, this may not be due to a lack of ingenuity on our part. We have concerns over whether a fundamental operation of digital architectures, the comparison of two data values, can be performed at all in an analog system. In digital systems, a stepwise comparison of pairs of digits will conclusively, and in a finite time, determine the equality of two digital values. In our analog optical processors, we cannot assume that image comparison is a logical operation; real or complex images can contain values with an infinite digital representation, and correlation, if we choose to use it, gives a probability rather than a truth value.

2.2. Optical models of computation

A number of researchers have analyzed optical information processing algorithms for computational efficiency. However, all have chosen discrete models to perform the analysis. Reif and Tyagi[4] employ a model that is closest to our requirements. They add a constant-time discrete FT (DFT) operation to a 3-D VLSI model of computation. The reasons for its unsuitability, however, are the same as those for not choosing the RAM. Louri and Post[3] cite a lack of suitable optical model as a reason for their choice of a conventional model based on (Boolean) logic circuits.

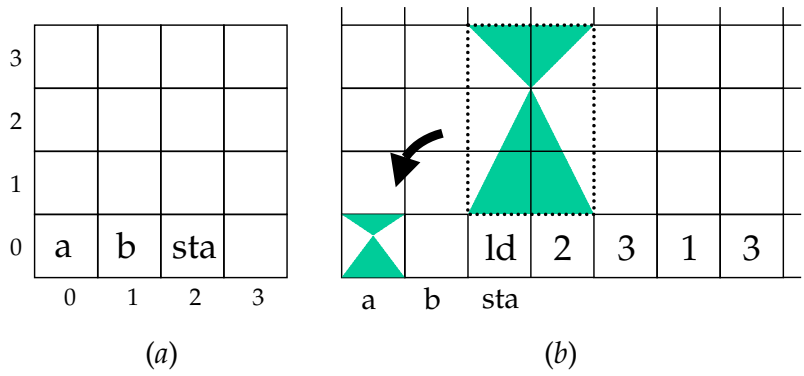


Figure 1. Schematics of (a) the grid memory structure of our theoretical machine, showing possible locations for the 'well known' addresses **a**, **b** and **sta**, and (b) loading (and automatically rescaling) a subset of the grid into grid element **a**. The program | ld | 2 | 3 | 1 | 3 | . . . | *hlt* | instructs the machine to load into default location **a** the portion of the grid addressed by columns 2 through 3 and rows 1 through 3.

Both Athale et al.[9] and Goutzoulis et al.[10] define discrete 2-D array models of computation to analyze their lookup table algorithms, using Boolean logic circuit complexity in the analysis. Caulfield[11] analyzes an optical processor that solves a set of linear equations. This processor has the property that a matrix-vector product can be taken in constant time. However, his implicit model is also a discrete one.

Some analysis appears in the literature for specialized optical systems or individual optical components. One example is Pelevin and Filimonov's[12] analysis of the time required to detect unknown signals in acousto-optic systems. Theirs, and similar models and measures, are too limited to be applied to Fourier optical information processing systems or general-purpose optical computers.

3. THEORETICAL MACHINE

The following description of our model of computation is based on van Emde Boas' formalisation of machine models.[13] In this framework, our theoretical machine is a collection of 6 things: a memory, a finite control, control flow mechanisms, information flow pathways, configurations (or states), and a tuple representation.

1. **Memory.** The memory structure is in the form of a 2-D grid of rectangular elements, as shown in Fig. 1(a). The grid has finite size and a scheme to address each element uniquely [usually the natural numbers, starting with $(0,0)$]. Each grid element is a 2-D continuous complex image, bounded between 0.0 and 1.0 in both directions. Three of the images are known to the machine by the identifiers **a**, **b**, and **sta** (two global storage locations and a program start location, respectively).

2. **Finite control.** The finite control is essentially an instruction pointer that contains the address of one of the grid elements. At any instant it is therefore in one of a finite set of possible states. The finite control can be instructed to move to any explicitly addressed memory location or by default to the grid element immediately to the right of the current one.

3. **Control flow.** Each program (instance of the theoretical machine) has a finite sequence of operations to manage the memory contents and to direct the finite control. The sequence will be encoded as a rectangle of images and stored in memory. The most basic operations, **ld** and **st**, copy rectangular $m \times n$ subsets of the grid ($m, n \in \mathbb{N}$, $m, n >= 1$) into and out of image **a**, respectively. Upon such loading and storing the image information is rescaled to the full extent of the target location (as depicted in Fig. 1(b)). Two additional real numbers z_{lower} and z_{upper}, specifying upper and lower cut-off values, filter the rectangle's contents by amplitude before rescaling,

$$f(i,j) = \begin{cases} z_{\text{lower}} & : \quad \text{Re}\,[f(i,j)] < z_{\text{lower}} \\ z_{\text{upper}} & : \quad \text{Re}\,[f(i,j)] > z_{\text{upper}} \\ f(i,j) & : \quad \text{otherwise} \end{cases} , \tag{1}$$

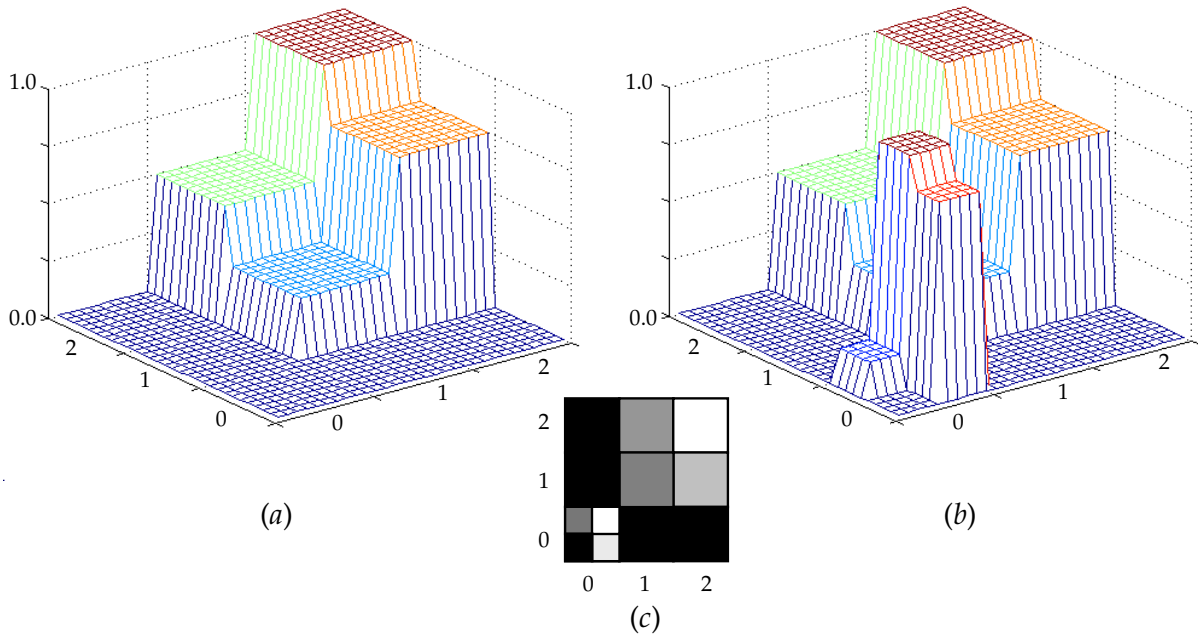


Figure 2. The operation to move a subset of the grid into image **a** has the form $\mathrm{ld}(x_1, x_2, y_1, y_2, z_{\mathrm{lower}}, z_{\mathrm{upper}})$. (a) depicts the initial state of a 3×3 memory grid and (b) depicts the state of the memory after the $\mathrm{ld}(1, 2, 1, 2, 18/40, 32/40)$ operation has been performed. (c) is a 2-D representation of (b). In this simple example we assume imaginary components of 0i everywhere.

as shown in Fig. 2. Other possible atomic operations perform horizontal and vertical 1-D FTs (**h** and **v**, respectively) on the 2-D image **a**, multiply $(\cdot)$ **a** by **b** (point by point), perform a complex addition $(+)$ of **a** and **b**, and produce the complex conjugate $(*)$ of the image in **a**. By default, the result of any such operation will be found in **a**. In addition, we have a system of specifying the order of execution of the list of operations; a left to right ordering implies sequential composition. The finite control will be told automatically to move right whenever the instruction at the current image address has been performed. There are only two occasions when the finite control will not move directly to the right. The first occurs after an unconditional branching instruction (**br**) directs the finite control to some other location in the memory grid. The second occurs at the end of every normally terminating instance of the theoretical machine: encountering the halt command symbol (**hlt**).

4. **Information flow.** A portion of the memory grid (≥ 0 images) will be reserved for input. When the finite control encounters a halt symbol the output will be in **a**.

5. **Configurations.** At each timestep, the memory grid plus finite control defines a unique configuration. We define the transition relation between two configurations of our machine $C_1 \vdash C_2$ as the application of one of {**ld**, **st**, **h**, **v**, $*$, $\cdot$, $+$, **br**, **hlt**} to our memory grid (which, of course, includes the input). We define the initial configuration as that configuration with the finite control at the start (**sta**) location in memory and **a** and **b** empty (or undefined). Any configuration with the finite control at a halt symbol is a final configuration.

6. **Tuples.** We are now in a position to describe each instance of our machine as a quintuple $M = \langle D, L, Z, I, P \rangle$, in which

   - $D = \langle x, y \rangle\,, \quad x, y \in \mathbb{N}$: grid dimensions
   - $L = \langle a_{\mathrm{x}}, a_{\mathrm{y}}, b_{\mathrm{x}}, b_{\mathrm{y}}, s_{\mathrm{x}}, s_{\mathrm{y}} \rangle\,, \quad a, b, s \in \mathbb{N}$: locations of **a**, **b**, and **sta**
   - $Z = \langle z_{\mathrm{MIN}}, z_{\mathrm{MAX}}, r \rangle\,, \quad z \in \mathbb{C}, \quad r \in \mathbb{Q}$: bounds and amplitude resolution on image values

- $I = [(i_{1\mathrm{x}}, i_{1\mathrm{y}}, \psi_1), \ldots, (i_{n\mathrm{x}}, i_{n\mathrm{y}}, \psi_n)]$, $i \in \mathbb{N}$, $\psi \in Image$: the n inputs and locations, where $Image$ is a finitely describable complex surface bounded by 0 and 1 in both spatial directions and with values limited by Z
- $P = [(p_{1\mathrm{x}}, p_{1\mathrm{y}}, \pi_1), \ldots, (p_{m\mathrm{x}}, p_{m\mathrm{y}}, \pi_m)]$, $p \in \mathbb{N}$, $\pi \in \{\mathrm{ld}, \mathrm{st}, \mathrm{h}, \mathrm{v}, *, \cdot, +, \mathrm{br}, \mathrm{hlt}, /, N\} \subset Image$: the m programming symbols, for a given instance of the theoretical machine, and their locations. An explanation for symbols / and N follows.

3.1. Grammar for the machine's language

In our definition of P above we allow two additional programming symbols which are not themselves operations. One, $N \in Image$, encodes a natural number under some encoding scheme and is used to specify a row or column of the memory grid. Such an encoding scheme would have to be determined by the designer of any physical realization of the theoretical machine. The second, $/ \in Image$, is used to separate the numerator and denominator when specifying the amplitude filter $(z_{\mathrm{lower}}, z_{\mathrm{upper}})$ with rational numbers.

There is a restriction on the possible sequences of programming symbols that can appear in any instance of P. Each syntactically-correct program must form a word in the language accepted by whatever compiler or interpreter we employ for an implementation of our theoretical machine. This language is generated from the following context-free grammar[14] in Backus normal form notation,

$$\begin{aligned}
S &\to MS \mid FS \mid M \mid F \\
M &\to \mathrm{ld}A \mid \mathrm{st}A \mid \mathrm{br}N;N; \mid \mathrm{hlt} \mid \square \\
F &\to \mathrm{h} \mid \mathrm{v} \mid * \mid \cdot \mid + \\
A &\to N;N;N;N;Q;Q; \\
N &\to ND \mid D \\
Q &\to N/N \\
D &\to 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \qquad ,
\end{aligned}$$

where we use capital letters for nonterminals and lowercase letters for terminals. Some explanation of the symbols follows. S, by convention, is the first nonterminal. Memory and control operations, M, and processing operations, F, can be combined sequentially. Load and store operations address a portion of the memory using two column, two row, and the two real-valued numbers (explained earlier) that are expressed here as quotients. Branching requires row and column coordinates. F operations require no parameters; they act on images **a** and **b** by default. The $\square$ symbol represents an empty or undefined image. Although not part of the programmer's set of operations, empty or undefined grid elements can appear in the absence of a programming symbol and so may be encountered by the preprocessor.

4. EXAMPLE THEORETICAL MACHINES

We demonstrate the operation of our theoretical machine with three examples. Each machine effects a well-known convolution technique that employs the FT operation. The first, matched filtering[15] of a signal $f(x, y)$ with a reference signal $s(x, y)$,

$$g(-u, -v) = f \circledast s = \mathcal{F}\{F(\alpha, \beta) S^*(\alpha, \beta)\} \quad , \tag{2}$$

where $\circledast$ denotes correlation, $*$ means the complex conjugate, u and v define the correlation plane, α and β define the Fourier plane, $\mathcal{F}$ denotes application of a FT operation, and $F = \mathcal{F}\{f\}$ and $S = \mathcal{F}\{s\}$. The following matched filtering pseudo-code algorithm multiplies $\mathcal{F}\{\mathbf{i1}\}$ by the complex conjugate of $\mathcal{F}\{\mathbf{i2}\}$ and takes a FT of the result,

```
ld i2       :  load image i2 into a
h v         :  horizontal FT of a followed by vertical FT of a
* st b      :  get complex conjugate of a and place in b
ld i1 h v   :  load image i1 into a and take horizontal and vertical FTs      ,
·           :  point-by-point multiplication of a and b, store result in a
h v hlt     :  perform FTs again and halt
```

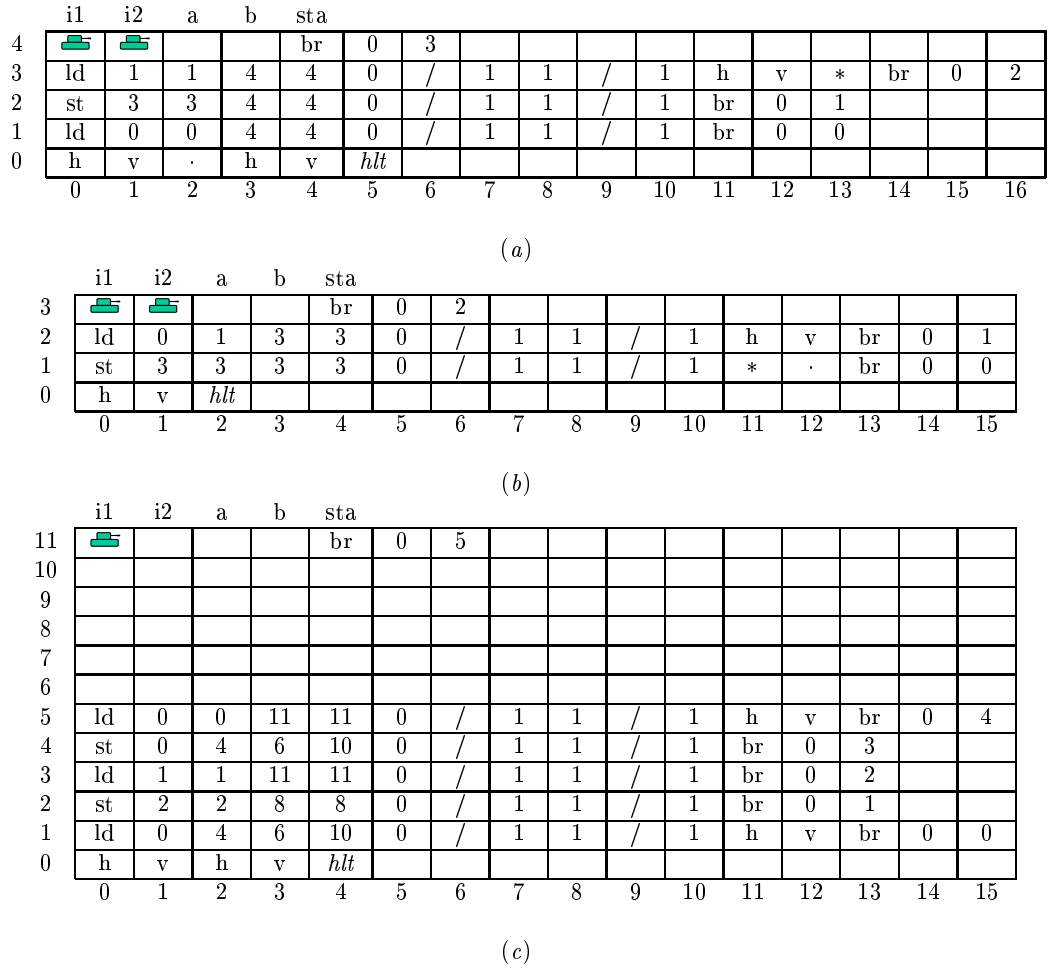


Figure 3. Theoretical machine performing (a) matched filtering correlation, (b) joint transform correlation, and (c) spatial filtering, with input images **i1** and **i2**. Computations begin at address **sta** and terminate at a **hlt** symbol.

where images **i1** and **i2** are the two inputs, and a sequential composition of **h** and **v** is equivalent to a 2-D FT. As required, upon halting the output image can be found in **a**. Fig. 3(a) shows the machine in its entirety. Unless otherwise stated, we assume that the bounds on image values for theoretical machines are $z_{\mathrm{MIN}} = 0 + 0\mathrm{i}$ and $z_{\mathrm{MAX}} = 1 + 0\mathrm{i}$. The load and store commands contain 0/1 (=0) and 1/1 (=1) for their z_{lower} and z_{upper} parameters, respectively, indicating that the complete image is to be accessed. Notice also the two grid elements reserved for input and the computation beginning at image **sta**. In this machine, the **br** commands are present largely for aesthetic reasons, to give a rectangular shape to the program. As a convention (for the reader's and programmer's convenience) we use boldface and underlining in program grid elements whose images can be changed by the machine and italics to highlight points of machine termination within the grid.

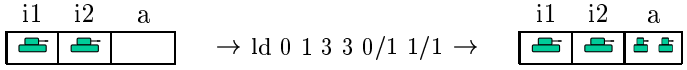
The second example machine performs joint transform correlation[16] on its two inputs $f(x, y)$ and $s(x, y)$,

$$g(-u, -v) = f \circledast f + s \circledast s + f \circledast s + s \circledast f = \mathcal{F}\left\{ |F(\alpha, \beta) + S(\alpha, \beta)|^2 \right\} \quad , \tag{3}$$

where the previous explanation of notation applies. The following is a pseudo-code algorithm for the machine,

ld i1 i2	:	load both images into **a**
h v st b	:	2-D FT and copy to **b**
$* \cdot$	:	multiply complex conjugate of **a** by **b**
h v hlt	:	2-D FT and halt

Although both images are loaded side-by-side and rescaled as shown here,



this rescaling will not lower the spatial resolution of the resulting FT as, by definition, grid images have infinite spatial resolution. The full theoretical machine is shown in Fig. 3(b). The third example machine [Fig. 3(c)] performs high-pass filtering on its input image **i1**, by multiplying it by a mask **i2**, also supplied as input. The image is rescaled relative to the mask to achieve the required degree of filtering.

We can perform various computational complexity comparisons between machines. Appropriate measures of complexity are suggested in the next section.

4.1. Complexity Measures

The standard computational complexity measures of time and space could be used to analyze instances of our machine. TIME would be measured by counting the number of times any program grid square was accessed by the finite control. Particular operations could be weighted (**h** could have a different cost to **br**) and parameter values taken into account (**ld** and **st** would have costs proportional to the number of grid elements accessed, and **br** would have a cost proportional to the distance of the next instruction). This would also accommodate encoding schemes where decoding a numerical value from an image took TIME proportional to the size of that value. It is important to recognize that the temporal cost of grid square accesses would differ by no more than a constant (worst case being the size of the grid) and does not otherwise depend on the amount or type of information stored in an image. SPACE could be a straightforward static measure of the number of elements in the grid. Such a measure would include storage of the program and input data. An alternative costing, and the standard practice in computer theory, counts the number of grid elements that were overwritten at least once. This latter measure does not count the grid elements reserved for the program or the input.

Caulfield[11] has incorporated an additional complexity measure into his optical algorithm analysis: that of fan-in. This discrete measure is not suitable for an analog model of computation, such as ours, but a comparable measure would be RESOLUTION. RESOLUTION is a measure of the spatial compression of image data due to rescaling, relative to the input resolution. For example, the joint transform correlator machine has a RESOLUTION complexity of 2 since at most two images are compressed without loss into one grid element. This imposes the following implementation limitation: any physical realization of such a machine could only accept input images with a resolution strictly less than 50% of the SLM resolution. Both the spatial filtering and matched filtering machines have RESOLUTION complexity of 1. For more sophisticated optical algorithms RESOLUTION will become a critical concern. In the PDA simulation in Fig. 4 it could be used to measure the greatest number of symbols stored in the stack during the computation.

Another measure is RANGE, which is used to describe the amplitude resolution (or dynamic range) of the images in memory. This is a static measure, not dependent on the inputs. It would be useful if any instance of the machine employed amplitude quantization or noise modeling.

5. COMPUTATIONAL PROPERTIES OF THE MODEL

As with all abstract models of the physical world, an abstract model of a physical processor allows us to reason about its intrinsic properties. So, as well as enabling us to analyze Fourier optical processing algorithms, we could determine some fundamental computational limits, such as lower bounds on the complexity of problems solved through Fourier optics and an upper bound on the computational power of Fourier optical processors. Although computational power is often associated with the speed of a processor, it is also a technical term describing the set of all functions that a processor can realize. A personal computer is more powerful than the simplest electronic

calculator; it uses memory to store intermediate results and, for example, can work with numbers larger than its register capacity. This definition of 'power' has nothing to do with processor speed or efficiency.

We often use a technique called simulation to measure computational power. If we can show that machine B can simulate every operation that A performs, we can say that B is at least as powerful as A, without ever having to explicitly compare functionality. By subsequently finding a single operation that B computes that A cannot, we have a constructive proof that B is more powerful than A.

5.1. General-purpose analog optical processing

There has been at least one attempt to define a generalized or general-purpose FT processor. In 1994, Caulfield et al.[5] produced a description of what they called a generalized FT (GFT) processor. They sought to keep the space-invariance of Fourier processors while extending them by increasing the set of functions they are capable of realizing. This was achieved (in matrix representation) by allowing off-diagonal terms in the 2-D spatial frequency filter of a 1-D convolution operation. In such a GFT processor framework the conventional FT (CFT) processor becomes a special case. Computer simulations reveal at least one instance where the GFT produces a superior result to that of the CFT, but Caulfield et al. do not demonstrate or claim general-purpose capabilities.

In an alternative approach, we investigate the general-purpose properties of this simple processor by simulating a standard (albeit non-universal) model of computation from computer theory: the push-down automaton[14] (PDA). By attempting to define the computational power of our theoretical analog information processor we investigate if analog optics has anything to offer the mainly digital approach[17] to general-purpose optical computer design. An instance of our machine simulating an instance of a PDA is shown in Fig. 4. This PDA recognizes the language of character strings of the form $S \rightarrow \mathrm{a}S\mathrm{b}|\epsilon$, i.e. the set $\{\epsilon, \mathrm{ab}, \mathrm{aabb}, \mathrm{aaabbb}, \ldots\}$. Language recognition problems are the most general form of computation; all computations (squaring an integer, taking a Fourier transform) can be expressed in this form. Appendix A contains some background work.

The instruction language we have chosen is restricted to operations for which it should be possible to have physical optical implementations (image multiplication, addition, Fourier transformation). As might be expected for an analog machine, we do not permit comparison of arbitrary image values; correlation cannot be used to definitively compare two arbitrary analog values in a finite number of steps. Fortunately, not having such a comparison operator will not impede us from implementing a branching operation. Correlation can be used for address resolution since (*i*) our set of possible images is finite (each memory grid has a fixed size), and (*ii*) we anticipate no false positives (we will never seek an address not from this finite set). The possibility of future implementation of our theoretical machine makes its simulation of an infinite-state model of computation all the more significant.

6. CONCLUSION

Our theoretical machine can be used to describe a range of Fourier optical information processing algorithms. Complexity measures are outlined with which the machine's algorithms can be analyzed. The instruction language we have chosen is restricted to operations that should be possible to physically implement in optics.

It is also programmable, in the sense that we have simulated a limited (but infinite state) model of computation from computer theory: the PDA. (To convincingly demonstrate general-purpose capabilities we must simulate a universal model of computation, possibly by incorporating additional memory stacks into our current simulation.) Being programmable suggests that a general-purpose analog (electro-)optical processor could be built that did not require a digital electronic computer to act as its control unit.

Further work for the author includes (*i*) showing how the measures of complexity, defined here, could be applied to the analysis of any Fourier processor, (*ii*) expressing a wider range of algorithms with the machine, such as instances of nonlinear correlators, and (*iii*) investigating if these results for Fourier processors can be generalized to other forms of analog optical computation.

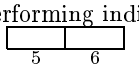
Figure 4. Theoretical machine simulating a PDA that recognizes the language $S \rightarrow \mathrm{a}S\mathrm{b}|\epsilon$. Rows 0 through 3 store the PDA algorithm, the image at grid element $(1, 99)$ encodes the input, and $(3, 99)$ encodes the stack. Computation begins at (**sta**, 99) and terminates in either an accepting or rejecting state (loads either 1 or 0 into **a**, respectively). The push and pop subroutines employ indirect addressing [location (**stk**, 99) contains the address of the stack].

6	ld	5	5	7	7	0	/	1	1	/	1	br	0	5
5	st	1	1	0	0	0	/	1	1	/	1	br	0	4
4	st	2	2	0	0	0	/	1	1	/	1	br	0	3
3	ld	6	6	7	7	0	/	1	1	/	1	br	0	2
2	st	3	3	0	0	0	/	1	1	/	1	br	0	1
1	st	4	4	0	0	0	/	1	1	/	1	br	0	0
0	ld					0	/	1	1	/	1			
	0	1	2	3	4	5	6	7	8	9	10	11	12	13

(a)

4	ld	5	5	7	7	br	0	3
3	st	&t1	st	&t2	br	0	2	
2	ld	6	6	7	7	br	0	1
1	st	&t3	st	&t4	br	0	0	
0	ld	**t1**	**t2**	**t3**	**t4**			
	0	1	2	3	4	5	6	7

(b)

Figure 5. Performing indirect addressing with the theoretical machine: (a) loading the contents of the address specified by the images 

and (b) its abbreviation in shorthand.

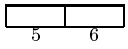
APPENDIX A. SIMULATING A PDA

The instruction set of our machine consists of a minimal list of operations that use direct addressing to manipulate memory and an unconditional jump command that also uses direct addressing. Much research has already been done on defining minimal instruction sets for general-purpose computation. One of the earliest and most celebrated was for Minsky's program machines[18] in 1967. His elegant instruction set consists of little more than an operation to decrement a counter and, based on its value, branch to another instruction. Unfortunately, our machine does not even have the comparison operation in its instruction set. This is a serious situation. A comparison operation of some sort seems to be the most basic requirement of any general-purpose instruction set; it is required to build counters, `while` or `for` loops (in fact, any terminating loop with more than one iteration), and conditional jumps. Fortunately, we show that two innocuous properties of our theoretical machine are sufficient to simulate any existing universal instruction set:

1. direct addressing; our machine can decode memory addresses stored as images, and
2. self-modification; no restriction has been made on which grid elements can be modified. We will allow programs to modify themselves while executing.

Our technique involves first simulating indirect addressing with properties 1 and 2 above, and then simulating conditional branching with a combination of indirect addressing and unconditional branching.

A.1. Indirect addressing and conditional branching

Our model employs direct addressing: the **br**, **ld**, and **st** operations must be followed by an explicit address. Neither "branch to the location in **a**" nor "load the grid element whose address is at (x, y)" are directly possible with our instruction set. To effect indirect addressing we use the self-modification ability of our machine. The instruction "load the contents of the image specified at images 

" would be expressed as shown in Fig. 5(a). A more compact shorthand version is given in Fig. 5(b). Note that in this shorthand, instead of specifying specific addresses we give images a temporary name (such as 't1') and refer to the address of that image with the ampersand ('&') character. Expansion from this shorthand to the long-form programming language is a mechanical procedure that could be performed as a 'tidying-up' phase by the programmer or by a preprocessor.

Next, we use indirect addressing and unconditional branching to simulate conditional branching. We base our technique on that of Rojas.[19] If we assume that our set of symbols is finite (all Turing-computable problems can

be solved with a finite set of symbols) then the conditional branching instruction "if (α=1) then jump to address X, else jump to Y" could be written as unconditional branching instruction "jump to address α". All we are required to ensure is that the code corresponding to addresses X and Y are at addresses 1 and 0, respectively. In a 2-D memory, multiple such branching instructions are possible. The stack for our automaton is encoded here as a sequence of images, compressed recursively into a single grid element. Each push operation costs one unit of RESOLUTION complexity as the new image and the stack, positioned side-by-side, are compressed into a single image. The PDA simulator is shown in Fig. 4, using the shorthand notation illustrated in Fig. 5(b).

ACKNOWLEDGMENTS

Sincere thanks to the Department of Computer Science, NUI Maynooth, and to Head of Department, Stephen Brown, for financial assistance. Thanks also to Damien Woods and Paul Gibson for many valuable and fruitful discussions.

REFERENCES

1. C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, Massachusetts, 1995.
2. R. Sedgewick and P. Flajolet, *An Introduction to the Analysis of Algorithms*, Addison-Wesley, Reading, Massachusetts, 1996.
3. A. Louri and A. Post, "Complexity analysis of optical-computing paradigms," *Applied Optics* **31**, pp. 5568–5583, Sept. 1992.
4. J. H. Reif and A. Tyagi, "Efficient parallel algorithms for optical computing with the discrete Fourier transform (DFT) primitive," *Applied Optics* **36**, pp. 7327–7340, Oct. 1997.
5. H. J. Caulfield, H. Peng, and J. Hereford, "Generalized Fourier transform processor," in *Optical Pattern Recognition V*, D. P. Casasent and T.-H. Chao, eds., Proceedings of SPIE vol. 2237, pp. 320–328, (Orlando, Florida), Apr. 1994.
6. T. Naughton, Z. Javadpour, J. Keating, M. Klíma, and J. Rott, "General-purpose acousto-optic connectionist processor," *Optical Engineering* **38**, pp. 1170–1177, July 1999.
7. T. J. Naughton, M. Klíma, and J. Rott, "Improved joint transform correlator performance through spectral domain thresholding," in *Optical Engineering Society of Ireland/Irish Machine Vision and Image Processing Joint Conference*, D. Vernon, ed., pp. 199–214, (Maynooth, Ireland), Sept. 1998.
8. J. C. Shepherdson and H. E. Sturgis, "Computability of recursive functions," *Journal of the Association for Computing Machinery* **10**(2), pp. 217–255, 1963.
9. R. A. Athale, M. W. Haney, J. J. Levy, and G. W. Euliss, "Minimum complexity optical architecture for look-up table computation in the residue number system," in *Optical Technology for Microwave Applications VI and Optoelectronic Signal Processing for Phased-Array Antennas III*, S.-K. Yao and B. M. Hendrickson, eds., Proceedings of SPIE vol. 1703, pp. 411–418, (Orlando, Florida), Apr. 1992.
10. A. P. Goutzoulis, "Complexity of residue position-coded lookup table array processors," *Applied Optics* **26**, pp. 4823–4831, Nov. 1987.
11. H. J. Caulfield, "Space-time complexity in optical computing," in *Optical Information-Processing Systems and Architectures II*, B. Javidi, ed., Proceedings of SPIE vol. 1347, pp. 566–572, July 1990.
12. V. Pelevin and R. Filimonov, "Temporal complexity and efficiency of hybrid acousto-optic systems for signals detection," in *Second International Conference on Optical Information Processing*, Z. I. Alferov, Y. V. Gulyaev, and D. R. Pape, eds., Proceedings of SPIE vol. 2969, pp. 479–482, (St. Petersburg, Russia), June 1996.
13. P. van Emde Boas, "Machine models and simulations," in *Handbook of Theoretical Computer Science*, J. van Leeuwen, ed., vol. A, chapter 1, pp. 1–66, Elsevier, Amsterdam, 1990.
14. D. I. A. Cohen, *Introduction to Computer Theory*, Wiley, New York, second ed., 1997.
15. A. VanderLugt, "Signal detection by complex spatial filtering," *IEEE Transactions on Information Theory* **IT-10**, pp. 139–145, Apr. 1964.
16. C. S. Weaver and J. W. Goodman, "A technique for optically convolving two functions," *Applied Optics* **5**, pp. 1248–1249, July 1966.
17. A. Huang, "Architectural considerations involved in the design of an optical digital computer," *Proceedings of the IEEE* **72**, pp. 780–786, 1984.

18. M. L. Minsky, *Computation: Finite and Infinite Machines*, chapter 11, pp. 199–216. Series in Automatic Computation, Prentice Hall, Englewood Cliffs, New Jersey, 1967.
19. R. Rojas, "Conditional branching is not necessary for universal computation in von Neumann computers," *Journal of Universal Computer Science* **2**(11), pp. 756–768, 1996.