Optical computing and computational complexity

Damien Woods

Boole Centre for Research in Informatics
School of Mathematics
University College Cork
Ireland
http://www.bcri.ucc.ie/~dw5
d.woods@bcri.ucc.ie

Abstract. This work concerns the computational complexity of a model of computation that is inspired by optical computers. The model is called the continuous space machine and operates in discrete timesteps over a number of two-dimensional images of fixed size and arbitrary spatial resolution. The (constant time) operations on images include Fourier transformation, multiplication, addition, thresholding, copying and scaling. We survey some of the work to date on the continuous space machine. This includes a characterisation of the power of an important discrete restriction of the model. Parallel time corresponds, within a polynomial, to sequential space on Turing machines, thus satisfying the parallel computation thesis. A characterisation of the complexity class NC in terms of the model is also given. Thus the model has computational power that is (polynomially) equivalent to that of many well-known parallel models. Such characterisations give a method to translate parallel algorithms to optical algorithms and facilitate the application of the complexity theory toolbox to optical computers. In the present work we improve on these results. Specifically we tighten a lower bound and present some new resource trade-offs.

1 Introduction

Over the years, optical computers were designed and built to emulate conventional microprocessors (digital optical computing), and for image processing over continuous wavefronts (analog optical computing). Here we are interested in the latter class: optical computers that store data as images. Numerous physical implementations exist and example applications include fast pattern recognition and matrix-vector algebra [9, 24]. There have been much resources devoted to designs, implementations and algorithms for such optical information processing architectures (for example see [1, 4, 6, 9, 12–15, 22, 24, 31] and their references). However the computational complexity theory of optical computers[1] has received

[1] That is, finding lower and upper bounds on computational power in terms of known complexity classes.

relatively little attention when compared with other nature-insired computing paradigms. Some authors have even complained about the lack of suitable models [6, 13].

We investigate the computational complexity of a model of computation that is inspired by such optical computers. The model is relatively new and is called the continuous space machine (CSM) [16–18, 26–30]. The model was originally proposed by Naughton [16, 17]. The CSM computes in discrete timesteps over a number of two-dimensional images of fixed size and arbitrary spatial resolution. The data and program are stored as images. The (constant time) operations on images include Fourier transformation, multiplication, addition, thresholding, copying and scaling. We analyse the model in terms of seven complexity measures inspired by real-world resources.

Subsequent to the original [17] CSM definition, Naughton [16] showed that the CSM (sequentially) simulates Turing machines, with a constant factor slowdown in time, thus giving a lower bound on its computational power. Later it was shown [18] that the model could simulate Type-2 Turing machines [25]. It was also shown that the CSM definition was perhaps too general as there is an ω-language that is Type-2 (and Turing machine) undecidable, but is CSM decidable [18], and furthermore any language is decided in finite time (and infinite space) [30]. In this paper we mostly focus on computational complexity results for a restricted CSM called the $\mathcal{C}_2$-CSM. Section 2 surveys some of the work to date on the model. This includes an analysis of complexity resources relevant to the CSM. Optical information processing is a highly parallel form of computing and we have made this intuition more concrete by relating the $\mathcal{C}_2$-CSM to parallel complexity theory. We discuss characterisations of $\mathcal{C}_2$-CSM computational power in terms of sequential space complexity classes and NC. Section 3 presents a new result that improves the lower bound for $\mathcal{C}_2$-CSM simulation of sequential space.

2 CSM and $\mathcal{C}_2$-CSM

We begin by describing the model in its most general sense, this brief overview is not intended to be complete and more details are to be found in [26].

2.1 CSM

A complex-valued image (or simply, image) is a function $f : [0, 1) \times [0, 1) \to \mathbb{C}$, where $[0, 1)$ is the half-open real unit interval. We let $\mathcal{I}$ denote the set of complex-valued images. Let $\mathbb{N}^+ = \{1, 2, 3, \ldots\}$, $\mathbb{N} = \mathbb{N}^+ \cup \{0\}$, and for a given CSM M let $\mathcal{N}$ be a countable set of images that encode M's addresses. Additionally, for a given M there is an *address encoding function* $\mathfrak{E} : \mathbb{N} \to \mathcal{N}$ such that $\mathfrak{E}$ is Turing machine decidable, under some *reasonable* representation of images as words. An address is an element of $\mathbb{N} \times \mathbb{N}$.

Definition 1 (CSM). *A CSM is a quintuple $M = (\mathfrak{E}, L, I, P, O)$, where*
$\mathfrak{E} : \mathbb{N} \to \mathcal{N}$ *is the address encoding function,*
$L = ((s_\xi, s_\eta), (a_\xi, a_\eta), (b_\xi, b_\eta))$ *are the addresses: sta, a and b, where $a \neq b$,*
I and O are finite sets of input and output addresses, respectively,
$P = \{(\zeta_1, p_{1_\xi}, p_{1_\eta}), \ldots, (\zeta_r, p_{r_\xi}, p_{r_\eta})\}$ *are the r programming symbols ζ_j and their addresses where $\zeta_j \in (\{h, v, *, \cdot, +, \rho, st, ld, br, hlt\} \cup \mathcal{N}) \subset \mathcal{I}$.*
Each address is an element from $\{0, \ldots, \varXi-1\} \times \{0, \ldots, \mathfrak{Y}-1\}$ where $\varXi, \mathfrak{Y} \in \mathbb{N}^+$.

Addresses whose contents are not specified by P in a CSM definition are assumed to contain the constant image $f(x, y) = 0$. We interpret this definition to mean that M is (initially) defined on a grid of images bounded by the constants $\varXi$ and $\mathfrak{Y}$, in the horizontal and vertical directions respectively. The grid of images may grow in size as the computation progresses.

In our grid notation the first and second elements of an address tuple refer to the horizontal and vertical axes of the grid respectively, and image $(0, 0)$ is located at the lower left-hand corner of the grid. The images have the same orientation as the grid. For example the value $f(0, 0)$ is located at the lower left-hand corner of the image f.

In Definition 1 the tuple P specifies the CSM program using programming symbol images ζ_j that are from the (low-level) CSM programing language [26, 30]. We refrain from giving a description of this programming language and instead describe a less cumbersome high-level language [26]. Figure 1 gives the basic instructions of this high-level language. The copy instruction is illustrated in Figure 3. There are also **if/else** and **while** control flow instructions with conditions of the form $(f_\psi == f_\phi)$ where f_ψ and f_ϕ are *binary symbol images* (see Figures 2(a) and 2(b)).

Address *sta* is the start location for the program so the programmer should write the first program instruction at *sta*. Addresses a and b define special images that are frequently used by some program instructions. The function $\mathfrak{E}$ is specified by the programmer and is used to map addresses to image pairs. This enables the programmer to choose her own address encoding scheme. We typically don't want $\mathfrak{E}$ to hide complicated behaviour thus the computational power of this function should be somewhat restricted. For example we put such a restriction on $\mathfrak{E}$ in Definition 7. Configurations are defined in a straightforward way as a tuple $\langle c, e\rangle$ where c is an address called the control and e represents the grid contents.

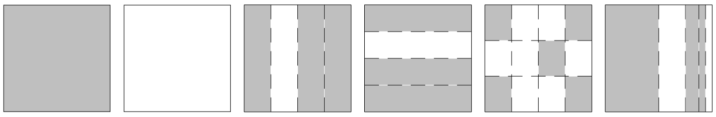
2.2 Complexity measures

Next we define some CSM complexity measures. All resource bounding functions map from $\mathbb{N}$ into $\mathbb{N}$ and are assumed to have the usual properties [2]. Logarithms are to the base 2.

Definition 2. *The* TIME *complexity of a CSM M is the number of configurations in the computation sequence of M, beginning with the initial configuration and ending with the first final configuration.*

$\mathrm{h}(i_1;i_2)$: replace image at i_2 with horizontal 1D Fourier transform of i_1.
$\mathrm{v}(i_1;i_2)$: replace image at i_2 with vertical 1D Fourier transform of image at i_1.
$*(i_1;i_2)$: replace image at i_2 with the complex conjugate of image at i_1.
$\cdot\,(i_1,i_2;i_3)$: pointwise multiply the two images at i_1 and i_2. Store result at i_3.
$+(i_1,i_2;i_3)$: pointwise addition of the two images at i_1 and i_2. Store result at i_3.
$\rho(i_1,z_\mathrm{l},z_\mathrm{u};i_2)$: filter the image at i_1 by amplitude using z_l and z_u as lower and upper amplitude threshold images, respectively. Place result at i_2.
$[\xi_1', \xi_2', \eta_1', \eta_2'] \leftarrow [\xi_1, \xi_2, \eta_1, \eta_2]$: copy the rectangle of images whose bottom left-hand address is (ξ_1, η_1) and whose top right-hand address is (ξ_2, η_2) to the rectangle of images whose bottom left-hand address is (ξ_1', η_1') and whose top right-hand address is (ξ_2', η_2'). See illustration in Figure 3.

Fig. 1. CSM high-level programming language instructions. In these instructions $i, z_\mathrm{l}, z_\mathrm{u} \in \mathbb{N} \times \mathbb{N}$ are image addresses and $\xi, \eta \in \mathbb{N}$. The control flow instructions are described in the main text.



(a) (b) (c) (d) (e) (f)

Fig. 2. Representing binary data. The shaded areas denote value 1 and the white areas denote value 0. (a) Binary symbol image representation of 1 and (b) of 0, (c) list (or row) image representation of the word 1011, (d) column image representation of 1011, (e) 3×4 matrix image, (e) binary stack image representation of 1101. Dashed lines are for illustration purposes only.

Definition 3. *The* GRID *complexity of a CSM M is the minimum number of images, arranged in a rectangular grid, for M to compute correctly on all inputs.*

Let $S : \mathcal{I} \times (\mathbb{N} \times \mathbb{N}) \to \mathcal{I}$, where $S(f(x,y), (\Phi, \Psi))$ is a raster image, with $\Phi\Psi$ constant-valued pixels arranged in Φ columns and Ψ rows, that approximates $f(x,y)$. If we choose a reasonable and realistic S then the details of S are not important.

Definition 4. *The* SPATIALRES *complexity of a CSM M is the minimum $\Phi\Psi$ such that if each image $f(x,y)$ in the computation of M is replaced with $S(f(x,y),(\Phi,\Psi))$ then M computes correctly on all inputs.*

Definition 5. *The* DYRANGE *complexity of a CSM M is the ceiling of the maximum of all the amplitude values stored in all of M's images during M's computation.*

We also use complexity measures called AMPLRES, PHASERES and FREQ [26, 30]. Roughly speaking, the AMPLRES of a CSM M is the number of discrete, evenly spaced, amplitude values per unit amplitude of the complex numbers in the range of M's images. The PHASERES of M is the total number (per 2π)

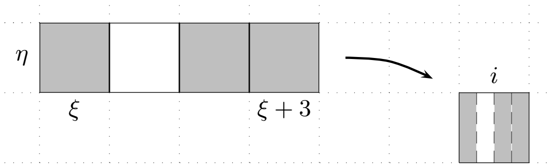


Fig. 3. Illustration of the instruction $i \leftarrow [\xi, \xi + 3, \eta, \eta]$ that copies four images to a single image that is denoted i.

of discrete evenly spaced phase values in the range of M's images. FREQ is a measure of the optical frequency of M's images [30].

Often we wish to make analogies between space on some well-known model and CSM 'space-like' resources. Thus we define the following convenient term.

Definition 6. *The* SPACE *complexity of a CSM M is the product of all of M's complexity measures except* TIME.

2.3 Representing data as images

There are many ways to represent data as images. Here we mention some data representations that are commonly used and moreover are used in Section 3. Figures 2(a) and 2(b) are the binary symbol image representations of 1 and 0 respectively. These images have an everywhere constant value of 1 and 0 respectively, and both have SPATIALRES of 1. The row and column image representations of the word 1011 are respectively given in Figures 2(c) and 2(d). These row and column images both have SPATIALRES of 4. In the matrix image representation in Figure 2(e), the first matrix element is represented at the top left corner and elements are ordered in the usual matrix way. This 3×4 matrix image has SPATIALRES of 12. Finally the binary stack image representation, which has exponential SPATIALRES of 16, is given in Figure 2(f). Section 3.1 discusses the manipulation of stack images.

Figure 3 shows how we might form a list image by copying four images to one in a single timestep. All of the above mentioned images have DYRANGE, AMPLRES and PHASERES of 1.

2.4 Worst case CSM resource usage

For the case of sequential computation it is usually obvious how the execution of a single operation will effect resource usage. In parallel models, execution of a single operation can lead to large growth in a single timestep. Characterising resource growth is useful for proving upper bounds on power and choosing reasonable model restrictions.

We investigated the growth of complexity resources over TIME, with respect to CSM operations [26, 28]. As expected, under certain operations some measures do not grow at all. Others grow at rates comparable to massively parallel models.

	GRID	SPATIALRES	AMPLRES	DYRANGE	PHASERES	FREQ
h	G_T	∞	∞	∞	∞	∞
v	G_T	∞	∞	∞	∞	∞
$*$	G_T	$R_{\text{S},T}$	$R_{\text{A},T}$	$R_{\text{D},T}$	$R_{\text{P},T}$	ν_T
$\cdot$	G_T	$R_{\text{S},T}$	$(R_{\text{A},T})^2$	$(R_{\text{D},T})^2$	$R_{\text{P},T}$	ν_T
$+$	G_T	$R_{\text{S},T}$	∞	$2R_{\text{D},T}$	∞	ν_T
ρ	unbounded	$R_{\text{S},T}$	$R_{\text{A},T}$	$R_{\text{D},T}$	$R_{\text{P},T}$	ν_T
st	unbounded	$R_{\text{S},T}$	$R_{\text{A},T}$	$R_{\text{D},T}$	$R_{\text{P},T}$	ν_T
ld	unbounded	unbounded	$R_{\text{A},T}$	$R_{\text{D},T}$	$R_{\text{P},T}$	unbounded
br	G_T	$R_{\text{S},T}$	$R_{\text{A},T}$	$R_{\text{D},T}$	$R_{\text{P},T}$	ν_T
hlt	G_T	$R_{\text{S},T}$	$R_{\text{A},T}$	$R_{\text{D},T}$	$R_{\text{P},T}$	ν_T

Table 1. CSM resource usage after one timestep. For a given operation and complexity measure pair, the relevant table entry defines the worst case CSM resource usage at TIME $T+1$, in terms of the resources used at TIME T. At TIME T we have GRID $= G_T$, SPATIALRES $= R_{\text{S},T}$, AMPLRES $= R_{\text{A},T}$, DYRANGE $= R_{\text{D},T}$, PHASERES $= R_{\text{P},T}$ and FREQ $= \nu_T$.

By allowing operations like the Fourier transform we are mixing the continuous and discrete worlds, hence some measures grow to infinity in one timestep. This gave strong motivation for CSM restrictions.

Table 1 summarises these results; the table defines the value of a complexity measure after execution of an operation (at TIME $T+1$). The complexity of a configuration at TIME $T+1$ is at least the value it was at TIME T, since complexity functions are nondecreasing. Our definition of TIME assigns unit time cost to each operation, hence we do not have a TIME column. Some entries are immediate from the complexity measure definitions, for others proofs are given in the references [26, 28].

2.5 $\mathcal{C}_2$-CSM

Motivated by a desire to apply standard complexity theory tools to the model, we defined [26, 28] the $\mathcal{C}_2$-CSM, a restricted and more realistic class of CSM.

Definition 7 ($\mathcal{C}_2$-CSM)**.** *A $\mathcal{C}_2$-CSM is a CSM whose computation* TIME *is defined for $t \in \{1, 2, \ldots, T(n)\}$ and has the following restrictions:*

- *For all* TIME *t both* AMPLRES *and* PHASERES *have constant value of 2.*
- *For all* TIME *t each of* GRID*,* SPATIALRES *and* DYRANGE *is $O(2^t)$ and* SPACE *is redefined to be the product of all complexity measures except* TIME *and* FREQ*.*
- *Operations h and v compute the discrete Fourier transform (DFT) in the horizontal and vertical directions respectively.*
- *Given some* reasonable *binary word representation of the set of addresses $\mathcal{N}$, the address encoding function $\mathfrak{E} : \mathbb{N} \to \mathcal{N}$ is decidable by a logspace Turing machine.*

Let us discuss these restrictions. The restrictions on AMPLRES and PHASERES imply that $\mathcal{C}_2$-CSM images are of the form $f : [0,1)\times[0,1) \to \{0, \pm\frac{1}{2}, \pm 1, \pm\frac{3}{2}, \ldots\}$. We have replaced the Fourier transform with the DFT [3], this essentially means that FREQ is now solely dependent on SPATIALRES; hence FREQ is not an interesting complexity measure for $\mathcal{C}_2$-CSMs and we do not analyse $\mathcal{C}_2$-CSMs in terms of FREQ complexity [26, 28]. Restricting the growth of SPACE is not unique to our model, such restrictions are to be found elsewhere [8, 19, 21].

In Section 2.1 we stated that the address encoding function $\mathfrak{E}$ should be Turing machine decidable, here we strengthen this condition. At first glance sequential logspace computability may perhaps seem like a strong restriction, but in fact it is quite weak. From an optical implementation point of view it should be the case that $\mathfrak{E}$ is not complicated, otherwise we cannot assume fast addressing. Other (sequential/parallel) models usually have a very restricted 'addressing function': in most cases it is simply the identity function on $\mathbb{N}$. Without an explicit or implicit restriction on the computational complexity of $\mathfrak{E}$, finding non-trivial upper bounds on the power of $\mathcal{C}_2$-CSMs is impossible as $\mathfrak{E}$ could encode an arbitrarily complex halting Turing machine. As a weaker restriction we could give a specific $\mathfrak{E}$. However, this restricts the generality of the model and prohibits the programmer from developing novel, reasonable, addressing schemes.

2.6 $\mathcal{C}_2$-CSM and parallel complexity theory

We have given lower bounds on the computational power of the $\mathcal{C}_2$-CSM by showing that it is at least as powerful as models that verify the parallel computation thesis [26, 29]. This thesis [5, 7] states that parallel time corresponds, within a polynomial, to sequential space for reasonable parallel models. See, for example, [10, 11, 19, 23] for details. Let $S(n)$ be a space bound that is $\Omega(\log n)$. The languages accepted by nondeterministic Turing machines in $S(n)$ space are accepted by $\mathcal{C}_2$-CSMs computing in TIME $O(S^4(n))$.

Theorem 1 ([26, 29]). $\mathrm{NSPACE}(S(n)) \subseteq \mathcal{C}_2\text{-CSM–TIME}(O(S^4(n)))$

For example polynomial TIME $\mathcal{C}_2$-CSMs accept the PSPACE languages. (Of course any polynomial TIME $\mathcal{C}_2$-CSM algorithm that we could presently write to solve PSPACE-complete or NP-complete problems would require exponential SPACE.) Theorem 1 is established via $\mathcal{C}_2$-CSM simulation of vector machines [2, 20, 21]. In the simulation the SPACE overhead is polynomial in vector machine space. Using this fact we find that $\mathcal{C}_2$-CSMs that simultaneously use polynomial SPACE and polylogarithmic TIME accept the class NC.

Corollary 1 ([26, 29]). $\mathrm{NC} \subseteq \mathcal{C}_2\text{-CSM–SPACE, TIME}(n^{O(1)}, \log^{O(1)} n)$

We have also given the other of the two inclusions that are necessary in order to verify the parallel computation thesis; $\mathcal{C}_2$-CSMs computing in TIME $T(n)$ are no more powerful than $O(T^2(n))$ space bounded deterministic Turing machines.

Theorem 2 ([26, 27]). $\mathcal{C}_2\text{-CSM-TIME}(T(n)) \subseteq \mathrm{DSPACE}(O(T^2(n)))$

Via the proof of Theorem 2 we get another (stronger) result. $\mathcal{C}_2$-CSMs that simultaneously use polynomial SPACE and polylogarithmic TIME accept at most NC.

Corollary 2 ([26, 27]). $\mathcal{C}_2\text{-CSM-SPACE}, \text{TIME}(n^{O(1)}, \log^{O(1)} n) \subseteq \text{NC}$

The latter two inclusions are established via $\mathcal{C}_2$-CSM simulation by logspace uniform circuits of size and depth polynomial in SPACE and TIME respectively. Thus $\mathcal{C}_2$-CSMs that simultaneously use both polynomial SPACE and polylogarithmic TIME characterise NC.

It turns out that the $\mathcal{C}_2$-CSM simulation of sequential space can be made more efficient. Theorem 3 in the next section improves the lower bound given by Theorem 1.

3 Improved $\mathcal{C}_2$-CSM lower bound

In this section we improve the lower bound given by Theorem 1 by proving the following result.

Theorem 3. $\text{NSPACE}(S(n)) \subseteq \mathcal{C}_2\text{-CSM–TIME}(O(S^2(n)))$

Moreover the GRID and DYRANGE complexities are both reduced from $O(2^{S(n)})$ to $O(1)$. However we see a trade-off here as the reduction in GRID and DYRANGE is swapped for an increase[2] in SPATIALRES from $O(2^{S(n)})$ to $O(2^{3S(n)}S^3)$. Thus the SPACE overhead in Theorem 3 has not decreased, nevertheless the trade-off is interesting. Also the simulation is achieved[3] with AMPLRES of 1 and PHASERES of 1. In summary, we have tightened the relationship between the $\mathcal{C}_2$-CSM and sequential space:

Corollary 3.

$$\text{NSPACE}(S(n)) \subseteq \mathcal{C}_2\text{-CSM–TIME}(O(S(n))^2) \subseteq \text{DSPACE}(O(S(n))^4)$$

We prove Theorem 3 by giving a $\mathcal{C}_2$-CSM that efficiently generates (Lemma 1) and squares (Lemma 2) the transition matrix of a $S(n) = \Omega(\log n)$ space bounded Turing machine.

We assume that Turing machines have a single tape, use only binary tape symbols $\{0, 1\}$ and are nondeterministic. At each timesstep the tape head moves either left (denoted L) or right (denoted R). The proofs below are sketched in the sense that we refrain from giving explicit code.

---

[2] On a technical note we are abusing notation here. $\mathcal{C}_2$-CSMs are defined to use SPATIALRES $O(2^t)$ after t timesteps. To save the reader the burden of new notation we overload the notation "$\mathcal{C}_2$-CSM" by using it to also describe machines that are $\mathcal{C}_2$-CSMs except for the fact that they have a $O(2^{O(1)t})$ upperbound on SPATIALRES. Although we omit the details, we note that Theorem 2 and Corollary 2 still hold for such (more general) definitions of $\mathcal{C}_2$-CSM.

[3] This is in contrast to the proof of the previous lower bound proof [26, 29] where AMPLRES and PHASERES were both 2. Subtraction (via addition of negative numbers) and devision by 2 (via multiplication by 1/2) are not needed in the present proof.

3.1 Iteration

In order to bound iterative loops we use a 'counter image'. In previous work [26, 29] we used an image with value/range of k (and thus of dyRange k) as a counter for k iterations. At each iteration the counter image is decremented by 1 (by adding an image of value -1), and tested for equality with 0 (by addressing).

Here we are restricted to constant dyRange so a different approach is adopted. Our counter image for value k is a *unary stack image* that represents 1^k. A unary stack image is just like the binary stack illustrated in Figure 2(e) except that the represented word is a list of ones. To access the i^{th} bit in a stack image we 'pop' the stack i times. Popping involves spreading the stack over two horizontally adjacent images, the leftmost image now contains the topmost stack element, the rightmost image contains the remainder of the stack. Popping the stack in this way uses grid $O(1)$ and time $O(k)$ to pop the entire stack. After each pop we test if the popped element is 0 by addressing, this happens only on pop $k+1$. The unary stack image representation requires spatialRes of $O(2^k)$, and amplRes, phaseRes and dyRange of 1.

In the sequel we simply write $S(n)$ as S. In the proof of Lemma 1 below all loops run for S or $\log S$ iterations. Thus their counter images have spatialRes of $O(2^S)$, which is no more than the spatialRes of other parts of the algorithm. A similar argument holds for Lemma 2.

3.2 Generating the transition matrix

The configuration graph of a space bounded Turing machine M is a graph with exactly one node for each configuration of M. There is a path from node i to node j iff configuration c_i leads to configuration c_j in exactly one step via some transition rule of M (formally we write $c_i \vdash_M c_j$). On input w, given the pair of nodes corresponding to the (unique) initial and accepting configurations, simulating the computation of $M(w)$ is the same as asking if there is a path from the initial node to the accepting node. We simulate M by computing the reflexive transitive closure of the transition graph. To do this we represent the graph by a binary matrix which we call the transition matrix of M. There is one row (respectively column) for each node. Entry (i, j) is 1 iff there is a path from node i to node j. The reflexive transitive closure is computed by squaring the matrix a number of times that is logarithmic in the number of nodes. Motivations and further details are to be found in van Emde Boas' survey [23].

We begin by constructing the binary transition matrix.

Lemma 1. *Let M be a Turing machine that accepts $L \in \{0,1\}^*$ in space $S = 2^i$ for some $i \in \mathbb{N}$. Then there is a $\mathcal{C}_2$-CSM that generates the transition matrix of M in* time $O(S)$, spatialRes $O(2^{2S}S^2)$, grid $O(1)$, dyRange $O(1)$, amplRes 1 *and* phaseRes 1.

Proof (sketch). Let Q be the states of M and $t = (q_x, \sigma_1, \sigma_2, D, q_y)$ be an arbitrary transition rule of M, with initial state q_x, next state q_y, read symbol $\sigma_1 \in \{0,1\}$, write symbol $\sigma_2 \in \{0,1\}$, and tape head move direction $D \in \{L, R\}$.

Before generating the matrix we precompute some special images.

A Turing machine tape word is represented in a straightforward way as a binary list image. We quickly generate all 2^S possible words of length S in TIME $O(\log S)^2$ and SPATIALRES $O(2^S S)$. The output, denoted TapesVertical, is a list-matrix image with 2^S rows and S columns, where each row represents a unique tape word. To do this we use an algorithm that (recursively) generates the matrix image $\text{TapesVertical}_{s/2}$ of all words of length $S/2$. We let $f = \text{TapesVertical}_{s/2}$ then the following is repeated $\log S$ times: place one copy of f immediately above another, scale the two to one image, call the new image f. After this repeated scaling f contains S copies of $\text{TapesVertical}_{s/2}$. We place f immediately to the right of $\text{TapesVertical}_{s/2}$ and the two are scaled to a single image to give TapesVertical.

We generate the image TapesHorizontal that represents each possible tape word repeated S times. More precisely, TapesHorizontal is the list image representation of the binary word

$$(0^S)^S(0^{S-1}1)^S(0^{S-2}10)^S(0^{S-2}11)^S \dots (1^S)^S$$

TapesHorizontal is generated in TIME $O(S)$ and SPATIALRES $O(2^S S^2)$ from TapesVertical by copying and shifting subimages, the details are omitted.

A tape head position $k \in \{1, \dots, S\}$ is encoded as the list image representation of the word $0^{k-1}10^{S-k}$. There are S such words and we generate these in TIME $O(\log S)$ and SPATIALRES $O(S^2)$ by copying and scaling. The output P is a $S \times S$ matrix image with ones on the diagonal ($P_{i,i} = 1$) and zeros elswhere. Each row represents a unique tape head position. The image PositionsVertical consists of 2^S vertically juxtaposed copies of P and is easily generated in TIME $O(S)$.

We generate the image PositionsHorizontal that represents the list of all possible position words, repeated 2^S times. More precisely, PositionsHorizontal is the list image representation of the binary word

$$\left((10^{S-1})(010^{S-2})(0010^{S-3})\dots(0^{S-1}1)\right)^{2^S}$$

PositionsHorizontal is generated in TIME $O(S)$ and SPATIALRES $O(2^S S^2)$ from PositionsVertical by copying and shifting subimages, the details are omitted.

Finally we precompute the image P_R which is identical to P except that the represented tapes have their head positions moved one cell to the right (if the head was on the rightmost tape cell then it is moved to the leftmost tape cell).

We are now ready to generate the transition matrix. The Turing machine has at most $8|Q|^2$ transition rules. For simplicity we assume that all $8|Q|^2$ possible transition rules are explicitly given. We begin by generating the transition matrix for one of these transition rules t that changes the machine from state q_ℓ to state q_m as follows: $t = (q_\ell, 1, 1, R, q_m)$. Thus we are generating a matrix image that represents a binary matrix with entry (i, j) equal to 1 iff $c_i \vdash c_j$ via t.

First we generate a column image, denoted $\bar{\sigma}_1$, with entry $i \in \{1, \dots, 2^S S\}$ equal to 1 iff the read symbol of c_i is $\sigma_1 = 1$. We use PositionsVertical as a mask to isolate the read symbols from TapesVertical; that is we pointwise multiply

PositionsVertical and TapesVertical in $\textsc{time}$ $O(1)$. The resulting matrix is called MaskedReadSymbols. We vertically split MaskedReadSymbols into a left image and a right image, pointwise add the two, and repeat; after $\log S$ iterations the output is the column image $\bar{\sigma}_1$.

Secondly we generate a row image, denoted $\bar{\sigma}_2$, where entry $j \in \{1, \ldots, 2^S S\}$ is 1 iff the write symbol of c_j is $\sigma_2 = 1$. We use PositionsHorizontal as a mask to isolate the write symbols from TapesHorizontal; that is we pointwise multiply PositionsHorizontal and TapesHorizontal in $\textsc{time}$ $O(1)$. The resulting matrix is called MaskedWriteSymbols. We 'shuffle' this row of 2^S lists to a column of 2^S lists, that is we repeat the following S times: vertically split into a left image and a right image, place the left image above the right and scale to one image. Then we vertically split the result (in half) into a left image and a right image, pointwise add the two, and repeat for a total of $\log S$ iterations. We then 'unshuffle' this column to a row in $\textsc{time}$ $O(S)$ to get $\bar{\sigma}_2$.

Thirdly we generate a $2^S S \times 2^S S$ binary matrix image called $\overline{\text{positions}}$, where entry (i, j) is 1 iff the tape head position on configuration c_i, *after* a move to the right (recall $D = R$), is equal to the tape head position of configuration c_j. To do this we generate P'_R which is a $S \times S^2$ matrix image with S copies of P_R side by side. We then pointwise multiply P'_R by the row image that represents

$$(10^{S-1})(010^{S-2})(0010^{S-3}) \ldots (0^{S-1}1)$$

The result of this multiplication is a $S \times S^2$ matrix image. Then (using the technique of shuffling and adding mentioned above) this $S \times S^2$ matrix image is 'shuffled' $\log S$ times, vertically split and added $\log S$ times, and 'unshuffled' $\log S$ times. The resulting $S \times S$ matrix image is replicated 2^{2S} times to create a 'square' $2^S S \times 2^S S$ matrix image denoted $\overline{\text{positions}}$.

We pointwise multiply $\bar{\sigma}_1$, $\bar{\sigma}_2$ and $\overline{\text{positions}}$ in $\textsc{time}$ $O(1)$, and threshold between 0 and 1, to get a $2^S S \times 2^S S$ binary matrix image. Entry (i, j) of this matrix image is 1 iff c_i yields c_j in one step under the read symbol 1, write symbol 1 and tape head direction R.

This above procedure is repeated 8 times with different values for the triple (σ_1, σ_2, D) where $\sigma_1, \sigma_2 \in \{0, 1\}$ and $D \in \{L, R\}$. The resulting 8 matrix images are pointwise added in $\textsc{time}$ $O(1)$ to give a matrix image denoted B. Entry (i, j) in B is 1 iff c_i yields c_j in one step under any (σ_1, σ_2, D). We then create a $|Q| \times |Q|$ matrix image where entry (i, j) is 1 iff state q_i yields q_j via some transition rule (this can be computed sequentially in a straightforward way in $\textsc{time}$ $O(|Q|^2)$, or in parallel $\textsc{time}$ $O(\log |Q|)$ using techniques similar to those above). We multiply this by a $2^S S|Q| \times 2^S S|Q|$ matrix image that consists of $|Q|^2$ copies of B. The result is the binary matrix image that represents the transition matrix of M. $\square$

3.3 Squaring the transition matrix

Lemma 2. *Let n be a power of 2 and let A by a $n \times n$ binary matrix. The matrix A^2 is computed by a $\mathcal{C}_2$-CSM, using the matrix image representation, in* $\textsc{time}$ $O(\log n)$, $\textsc{spatialRes}$ $O(n^3)$, $\textsc{grid}$ $O(1)$, $\textsc{dyRange}$ $O(1)$, $\textsc{amplRes}$ 1 *and* $\textsc{phaseRes}$ 1.

Proof (sketch). In this proof the matrix, and its matrix image representation are both denoted A. We being with some precomputation, then one parallel pointwise multiplication step followed by $\log n$ additions completes the algorithm.

We generate the matrix image A_1 that consists of n vertically juxtaposed copies of A. This is computed by placing one copy of A above the other, scaling to one image, and repeating to give a total of $\log n$ iterations. The image A_1 is constructed in TIME $O(\log n)$, GRID $O(1)$ and SPATIALRES $O(n^3)$.

Next we transpose A to the column image A_2. The first n elements of A_2 are row 1 of A, the second n elements of A_2 are row 2 of A, etc. This is computed in TIME $O(\log n)$, GRID $O(1)$ and SPATIALRES $O(n^2)$ as follows.

Let $A' = A$ and $i = 2n$. We horizontally split A' into a left image A'_L and a right image A'_R. Then A'_L is pointwise multiplied (or masked) by the column image that represents $(10)^i$, in TIME $O(1)$. Similarly A'_R is pointwise multiplied (or masked) by the column image that represents $(01)^i$. The masked images are added. The resulting image has half the number of columns as A' and double the number of rows, and for example: row 1 consists of the first half of the elements of row 1 of A' and row 2 consists of the latter half of the elements of row 1 of A'. We call the result A' and we double the value of i. We repeat the process to give a total of $\log n$ iterations. After these iterations the resulting column image is denoted A_2.

We pointwise multiply A_1 and A_2 to give A_3 in TIME $O(1)$, GRID $O(1)$ and SPATIALRES $O(n^3)$.

To facilitate a straightforward addition we first transpose A_3 in the following way: A_3 is vertically split into a bottom and a top image, the top image is placed to the left of the bottom and the two are scaled to a single image, this splitting and scaling is repeated to give a total of $\log n$ iterations and we call the result A_4. Then to perform the addition, we vertically split A_4 into a bottom and a top image. The top image is pointwise added to the bottom image and the result is thresholded between 0 and 1. This splitting, adding and thresholding is repeated a total of $\log n$ iterations to create A_5. We 'reverse' the transposition that created A_4: image A_5 is horizontally split into a left and a right image, the left image is placed above the right and the two are scaled to a single image, this splitting and scaling is repeated a total of $\log n$ iterations to give A^2. $\square$

3.4 Proof of main result

At this point we have all the main ingredients for the proof of Theorem 3 which goes as follows. Using Lemma 1 we generate the $2^S S|Q| \times 2^S S|Q|$ binary transition matrix within the stated resource bounds. We put ones on the diagonal of this matrix by pointwise adding it to the $2^S S|Q| \times 2^S S|Q|$ identity matrix and thresholding the result between 0 and 1, all in constant TIME (however generating the identity matrix takes TIME $O(S)$). In TIME $O(S^2)$ we compute the reflexive and transitive closure of this matrix by squaring it $O(S)$ times via Lemma 2. In terms of M's input length n, the overall TIME is $O(S^2(n))$ and both SPATIALRES and SPACE are $O(2^{3S(n)} S^3(n))$.

Acknowledgements

Thanks to Cristian Calude, Grzegorz Rozenberg and the conference organisers for inviting me to UC'06. Also thanks to Tom Naughton and Paul Gibson who were collaborators on much of the previous work that is surveyed in Section 2. This work is funded by the Irish Research Council for Science, Engineering and Technology.

References

1. H. H. Arsenault and Y. Sheng. *An introduction to optics in computers*, volume TT 8 of *Tutorial texts in optical engineering.* SPIE, 1992.
2. J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural complexity II*, volume 22 of *EATCS Monographs on Theoretical Computer Science.* Springer, Berlin, 1988.
3. R. N. Bracewell. *The Fourier transform and its applications.* Electrical and electronic engineering series. McGraw-Hill, second edition, 1978.
4. H. J. Caulfield. Space-time complexity in optical computing. In B. Javidi, editor, *Optical information-processing systems and architectures II*, volume 1347, pages 566–572. SPIE, July 1990.
5. A. K. Chandra and L. J. Stockmeyer. Alternation. In *17th annual symposium on Foundations of Computer Science*, pages 98–108, Houston, Texas, Oct. 1976. IEEE. Preliminary Version.
6. D. G. Feitelson. *Optical Computing: A survey for computer scientists.* MIT Press, 1988.
7. L. M. Goldschlager. *Synchronous parallel computation.* PhD thesis, University of Toronto, Computer Science Department, Dec. 1977.
8. L. M. Goldschlager. A universal interconnection pattern for parallel computers. *Journal of the ACM*, 29(4):1073–1086, Oct. 1982.
9. J. W. Goodman. *Introduction to Fourier optics.* McGraw-Hill, New York, second edition, 1996.
10. R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to parallel computation: P-completeness theory.* Oxford university Press, Oxford, 1995.
11. R. M. Karp and V. Ramachandran. *Parallel algorithms for shared memory machines*, volume A. Elsevier, Amsterdam, 1990.
12. J. N. Lee, editor. *Design issues in optical processing.* Cambridge studies in modern optics. Cambridge University Press, 1995.
13. A. Louri and A. Post. Complexity analysis of optical-computing paradigms. *Applied optics*, 31(26):5568–5583, Sept. 1992.
14. A. D. McAulay. *Optical computer architectures.* Wiley, 1991.
15. T. Naughton, Z. Javadpour, J. Keating, M. Klíma, and J. Rott. General-purpose acousto-optic connectionist processor. *Optical Engineering*, 38(7):1170–1177, July 1999.
16. T. J. Naughton. Continuous-space model of computation is Turing universal. In S. Bains and L. J. Irakliotis, editors, *Critical Technologies for the Future of Computing*, Proceedings of SPIE vol. 4109, pages 121–128, San Diego, California, Aug. 2000.
17. T. J. Naughton. A model of computation for Fourier optical processors. In R. A. Lessard and T. Galstian, editors, *Optics in Computing 2000*, Proc. SPIE vol. 4089, pages 24–34, Quebec, Canada, June 2000.

18. T. J. Naughton and D. Woods. On the computational power of a continuous-space optical model of computation. In M. Margenstern and Y. Rogozhin, editors, *Machines, Computations and Universality: Third International Conference (MCU'01)*, volume 2055 of *LNCS*, pages 288–299, Chişinău, Moldova, May 2001. Springer.
19. I. Parberry. *Parallel complexity theory.* Wiley, 1987.
20. V. R. Pratt, M. O. Rabin, and L. J. Stockmeyer. A characterisation of the power of vector machines. In *Proc. 6th annual ACM symposium on theory of computing*, pages 122–134. ACM press, 1974.
21. V. R. Pratt and L. J. Stockmeyer. A characterisation of the power of vector machines. *Journal of Computer and Systems Sciences*, 12:198–221, 1976.
22. J. H. Reif and A. Tyagi. Efficient parallel algorithms for optical computing with the discrete Fourier transform (DFT) primitive. *Applied optics*, 36(29):7327–7340, Oct. 1997.
23. P. van Emde Boas. Machine models and simulations. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 1. Elsevier, Amsterdam, 1990.
24. A. VanderLugt. *Optical Signal Processing.* Wiley Series in Pure and Applied Optics. Wiley, New York, 1992.
25. K. Weihrauch. *Computable Analysis: An Introduction.* Texts in Theoretical Computer Science. Springer, Berlin, 2000.
26. D. Woods. *Computational complexity of an optical model of computation.* PhD thesis, National University of Ireland, Maynooth, 2005.
27. D. Woods. Upper bounds on the computational power of an optical model of computation. In *16*$^{\text{th}}$ *International Symposium on Algorithms and Computation (ISAAC 2005)*, volume 3827 of *LNCS*, pages 777–788, Sanya, China, Dec. 2005. Springer.
28. D. Woods and J. P. Gibson. Complexity of continuous space machine operations. In S. B. Cooper, B. Löewe, and L. Torenvliet, editors, *New Computational Paradigms, First Conference on Computability in Europe (CiE 2005)*, volume 3526 of *LNCS*, pages 540–551, Amsterdam, June 2005. Springer.
29. D. Woods and J. P. Gibson. Lower bounds on the computational power of an optical model of computation. In C. S. Calude, M. J. Dinneen, G. Păun, M. J. Pérez-Jiménez, and G. Rozenberg, editors, *Fourth International Conference on Unconventional Computation (UC'05)*, volume 3699 of *LNCS*, pages 237–250, Sevilla, Oct. 2005. Springer.
30. D. Woods and T. J. Naughton. An optical model of computation. *Theoretical Computer Science*, 334(1–3):227–258, Apr. 2005.
31. F. T. S. Yu, S. Jutamulia, and S. Yin, editors. *Introduction to information optics.* Academic Press, 2001.