Lower bounds on the computational power of an optical model of computation

Damien Woods[1] and J. Paul Gibson[2]

[1] Boole Centre for Research in Informatics, School of Mathematics, University College Cork, Ireland. Corresponding author: http://www.bcri.ucc.ie/~dw5
[2] Theoretical Aspects of Software Systems Research Group, Department of Computer Science, National University of Ireland, Maynooth, Ireland.

Abstract. We present lower bounds on the computational power of an optical model of computation called the $\mathcal{C}_2$-CSM. We show that $\mathcal{C}_2$-CSM time is at least as powerful as sequential space, thus giving one of the two inclusions that are necessary to show that the model verifies the parallel computation thesis. Furthermore we show that $\mathcal{C}_2$-CSMs that simultaneously use polynomial space and polylogarithmic time decide at least the class NC.

1 Introduction

The computational model we study is relatively new and is called the continuous space machine (CSM) [6–8, 14–16]. The CSM is inspired by classical Fourier optical computing architectures and uses complex-valued images, arranged in a grid structure, for data storage. The program also resides in images. The CSM has the ability to perform Fourier transformation, complex conjugation, multiplication, addition, thresholding and resizing of images. It has simple control flow operations and is deterministic. We analyse the model in terms of seven complexity measures inspired by real-world resources.

A rather general variant of the model was previously shown [14, 16] to decide the membership problem for all recursively enumerable languages, and as such is unreasonable in terms of implementation. Also, the growth in resource usage was shown for each CSM operation, which in some cases was unreasonably large [14, 15]. These results motivated the definition of the $\mathcal{C}_2$-CSM, a restricted CSM.

Recently [14] we have given upper and lower bounds on the computational power of the $\mathcal{C}_2$-CSM by showing that it verifies the parallel computation thesis. This thesis [2–5, 9, 12] states that parallel time corresponds, within a polynomial, to sequential space for reasonable parallel models. Furthermore we have characterised the class NC in terms of the $\mathcal{C}_2$-CSM [14].

Here we present one of the two inclusions that are necessary in order to verify the parallel computation thesis; we show that the languages accepted by nondeterministic Turing machines in $S(n)$ space are accepted by $\mathcal{C}_2$-CSMs computing in TIME $O(S(n) + \log n)^4$.

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

For example polynomial TIME $\mathcal{C}_2$-CSMs accept the PSPACE languages. Also we show that $\mathcal{C}_2$-CSMs that simultaneously use polynomial SPACE and polylogarithmic TIME accept the class NC.

$$\text{NC} \subseteq \mathcal{C}_2\text{-CSM–SPACE, TIME}(n^{O(1)}, \log^{O(1)} n)$$

These inclusions are established via $\mathcal{C}_2$-CSM simulation of index-vector machines.

2 The CSM

We begin by informally describing the model, this brief overview is not intended to be complete: Detailed definitions and discussions are to be found in [14, 16].

Definition 1 (Image). *A complex-valued image, or simply an 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 all complex-valued images. Let $\mathbb{N}^+ = \{1, 2, 3, \ldots\}$ and $\mathbb{N} = \mathbb{N}^+ \cup \{0\}$. An address is an element of $\mathbb{N} \times \mathbb{N}$. For a given CSM M we let $\mathcal{N}$ be a countable set of images that encode M's addresses. Also 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 [14].

Definition 2 (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,*
$I = ((\iota_{1_\xi}, \iota_{1_\eta}), \ldots, (\iota_{k_\xi}, \iota_{k_\eta}))$ *are the addresses of the k input images,*
$P = \{(\zeta_1, p_{1_\xi}, p_{1_\eta}), \ldots, (\zeta_r, p_{r_\xi}, p_{r_\eta})\}$ *are the r programming symbols and their addresses where* $\zeta_j \in (\{h, v, *, \cdot, +, \rho, st, ld, br, hlt\} \cup \mathcal{N}) \subset \mathcal{I}$*,*
$O = ((o_{1_\xi}, o_{1_\eta}), \ldots, (o_{l_\xi}, o_{l_\eta}))$ *are the addresses of the l output images.*

Each address is an element from $\{0, 1, \ldots, \Xi - 1\} \times \{0, 1, \ldots, \mathcal{Y} - 1\}$ *where* $\Xi, \mathcal{Y} \in \mathbb{N}^+$*. Addresses a and b are distinct.*

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 the above definition to mean that M is (initially) defined on a grid of images bounded by the constants Ξ and $\mathcal{Y}$, in the horizontal and vertical directions respectively.

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 at the lower left-hand corner of the grid. The images have the same orientation as the grid. 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.

In Definition 2 the tuple P specifies the grid locations of programming symbols ζ_j that are from the (low-level) CSM programing language [14, 16]. Here we introduce a less cumbersome, high-level, language. Figure 1 gives the basic instructions of this high-level language. There are also **if/else** and **while** control flow instructions with conditions of the form $(f_\psi == f_\phi)$ where f_ψ, f_ϕ are binary symbol images (see Figure 2(a)). Finally there are user defined functions. For

$\mathrm{h}(i_1;i_2)$	: replace image at i_2 with horizontal 1D Fourier transform (FT) of i_1.
$\mathrm{v}(i_1;i_2)$	: replace image at i_2 with vertical 1D FT 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 2(c).

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

convenience we write function input addresses before a ';' and function output addresses after the ';'. In Section 4 the reader will find example programs written in this programming language. See [14] for technical details and arguments to show that the low-level and high-level languages are equivalent.

Next we define some CSM complexity measures. All resource bounding functions map from $\mathbb{N}$ into $\mathbb{N}$ and have the usual properties [1].

Definition 3. *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.*

Definition 4. *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.*

For example suppose M accepts language L, then the GRID complexity of M is the minimum number of images accessible by M and arranged in a rectangular grid, such that M accepts exactly L.

Let $S : \mathcal{I}\times(\mathbb{N}\times\mathbb{N}) \to \mathcal{I}$, where $S(f(x,y),(\Phi,\Psi))$ is an image, with $\Phi\Psi$ constant-valued pixels arranged in Φ columns and Ψ rows, that approximates f. If we choose a reasonable and realistic S then the details of S are unimportant.

Definition 5. *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 6. *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.*

In previous treatments we also defined the complexity measures AMPLRES, PHASERES and FREQ [14–16]. AMPLRES and PHASERES are measures of the cardinality of discrete amplitude and phase values respectively of the complex numbers in the range of CSM images. In the present work AMPLRES and PHASERES both have constant value of 2 (due to Definition 8) which means that all images are of the form $f : [0,1) \times [0,1) \to \{0, \pm\frac{1}{2}, \pm 1, \pm\frac{3}{2}, \ldots\}$. Furthermore we are studying a restricted CSM to which FREQ does not apply. Often we wish to make analogies between space on some well-known model and CSM 'space-like' resources. For this purpose we define the following convenient term.

Definition 7. *The* SPACE *complexity of a CSM M is the product of all of M's complexity measures except* TIME.

We have defined the complexity of a computation (sequence of configurations) for each measure. We extend this definition to the complexity of a (possibly non-final) configuration in the obvious way. Also, we sometimes talk about the complexity of an image, this is simply the complexity of the configuration that the image is in. More details on the complexity measures are be found in [16].

In [14, 15] we defined the $\mathcal{C}_2$-CSM, a restricted class of CSM.

Definition 8 ($\mathcal{C}_2$-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 FT (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.*

3 Index-vector machines and representations

Here we introduce vector machines, and the variant that we simulate called index-vector machines. We then describe our image representation of vectors.

The vector machine model was originally described by Pratt, Rabin and Stockmeyer [10], here we mostly use the conventions of Pratt and Stockmeyer [11]. A vector V is a binary sequence that is infinite to the left only and is *ultimately constant* (after a finite number of bits every bit to the left is either always 0 or always 1). The length of V, denoted $|V|$, is the length of the non-ultimately constant part of V. An ultimately 0 sequence represents a positive number and an ultimately 1 sequence represents a negative number [11, 1]. The non-constant part represents a positive binary integer in the usual way, with the rightmost vector bit representing the least significant integer bit. The negative integer $-n$ is represented by the bitwise complement of the vector representing n. A vector machine (program) is a list of instructions where each is of the form given in the following definition.

Definition 9 (Vector machine instructions and their meanings [1]).

Vector instruction	*Meaning*
$V_i := x$	*Load the positive constant binary number x into vector V_i.*
$V_k := \neg V_i$	*Bitwise parallel negation of vector V_i.*
$V_k := V_i \wedge V_j$	*Bitwise parallel 'and' of two vectors.*
$V_k := V_i \uparrow V_j$	*If V_j is ultimately 0 (resp. 1) then shift V_i to the left (resp. right) by the distance given by the binary number v_j and store the result in V_k. If $V_j = 0$ then V_i is copied to V_k.*
$V_k := V_i \downarrow V_j$	*If V_j is ultimately 0 (resp. 1) then shift V_i to the right (resp. left) by the distance given by the binary number v_j and store the result in V_k. If $V_j = 0$ then V_i is copied to V_k.*
goto m *if* $V_i = 0$	*If $V_i = 0$ then branch to the instruction labelled m.*
goto m *if* $V_i \neq 0$	*If $V_i \neq 0$ then branch to the instruction labelled m.*

Instructions are labelled to facilitate the goto instruction. Configurations, (accepting) computations and computation time are all defined in the obvious way. Computation space is the maximum over all configurations, of the sum of the lengths of the vectors in each configuration. A language accepting vector machine on input w has an input vector of the form $...000w$ where $w \in 1\{0,1\}^*$. In this work we consider only deterministic vector machines. See [1] for details.

Definition 10 (Index-vector machines [11]). *A vector machine is of class $\mathcal{V}_I$ (equivalently, an index-vector machine) if its registers are partitioned into two disjoint sets, one set called index registers and the other called vector registers, such that (i) each Boolean operation in the program involves either only index registers or only vector registers; and (ii) each shift instruction is of the form*

$$V_1 := V_2 \uparrow I, \quad V_1 := V_2 \downarrow I, \quad I := J \uparrow 1, \quad I := J \downarrow 1$$

where V_1 and V_2 are vector registers, and I and J are index registers. For language recognition the input register is a vector register.

It is straightforward to prove the following lemma by induction on t.

Lemma 1 ([11]). *Given index-vector machine $M \in \mathcal{V}_I$ with n as the maximum input length, there is a constant c such that vector length in index (respectively vector) registers is bounded above by $c+t$ (respectively $2^{c+t}+n$) after t timesteps.*

Pratt and Stockmeyer's [11] main result is a characterisation of the power of index-vector machines. The characterisation is described by two inclusions, proved for time bounded index-vector machines and space bounded Turing machines:

$$\mathrm{NSPACE}(S(n)) \subseteq \mathcal{V}_I\text{–TIME}(O(S(n) + \log n)^2) \tag{1}$$

$$\mathcal{V}_I\text{–TIME}(T(n)) \subseteq \mathrm{DSPACE}(O(T(n)(T(n) + \log n))) \tag{2}$$

In other words, index-vector machines verify the parallel computation thesis and are a member of the second machine class [12]. Modulo a polynomial, deterministic and nondeterministic vector machines have equal power [10].

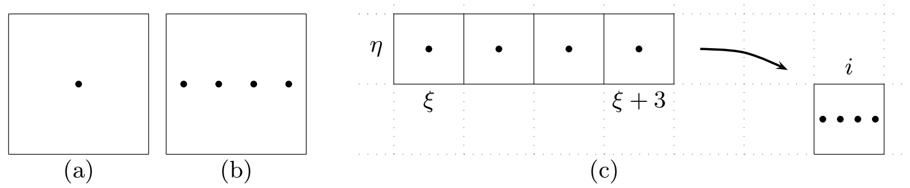

Fig. 2. Representing data by images. To represent the value ψ, the black point has value ψ. The white area denotes value zero. (a) *Binary symbol image* f_ψ where $\psi \in \{0, 1\}$, or *number image* where $\psi \in \{0, \pm\frac{1}{2}, \pm 1, \pm\frac{3}{2}, \ldots\}$, (b) *binary list image* where $\psi \in \{0, 1\}$. (c) Illustration of the instruction $i \leftarrow [\xi, \xi + 3, \eta, \eta]$ that copies four images to a single image that is denoted i.

3.1 Image representation of vectors

Let $v_i \in \{0, 1\}^*$ denote the non-'ultimately constant' part of vector V_i. If the ultimately constant part of V_i is 0^ω (respectively 1^ω) then let $\mathrm{sign}(v_i) = 0$ (respectively let $\mathrm{sign}(v_i) = 1$). In this work we use binary symbol images, number images and binary list images. These represent binary symbols, numbers from $\{0, \pm\frac{1}{2}, \pm 1, \pm\frac{3}{2}, \ldots\}$, and binary words in a straight-forward way that is illustrated in Figure 2. Further details are to be found in [14, 16].

The vector V_i is represented by three images: $\overline{v_i}$, $\overline{|v_i|}$ and $\overline{\mathrm{sign}(v_i)}$. The image $\overline{v_i}$ is the binary list image representation of v_i. Image $\overline{|v_i|}$ is the natural number image represention of $|v_i|$ (the length of v_i). Accessing these images respectively incurs SPATIALRES and DYRANGE costs that are linear in $|v_i|$. Image $\overline{\mathrm{sign}(v_i)}$ is f_0 (the binary symbol image representing 0) if $\mathrm{sign}(v_i) = 0$ and f_1 if $\mathrm{sign}(v_i) = 1$. We use the same representation scheme for vector program constants. The simulation uses natural number images as addresses, which are clearly reasonable in the sense of the $\mathcal{C}_2$-CSM definition. Hence addressing incurs a (linear) DYRANGE cost. The three images types are illustrated in Figures 2(a) and 2(b).

Another issue to consider is the layout of the grid of images; where to place input, program constants (f_0, f_1, f_{-1}, $f_{\frac{1}{2}}$, f_2), local variables, etc. There are only a constant number of such images hence there a number of layouts that work, a specific grid layout is given in [14]. Rows 0 and 1 are used to store temporary images. The only images explicitly referred to by numerical addresses are in these two rows (all other addresses have identifier names from the outset).

4 $\mathcal{C}_2$-CSM simulation of index-vector machines

In this section we prove that $\mathcal{C}_2$-CSMs are at least as powerful as index-vector machines (up to a polynomial in time). More precisely

$$\mathcal{V}_I\text{–TIME}(T(n)) \subseteq \mathcal{C}_2\text{-CSM–TIME}(O(T^2(n))). \tag{3}$$

To prove this we simulate each index-vector machine instruction in $O(\log |V_{\max}|)$ TIME where $|V_{\max}| \in \mathbb{N}$ is the maximum length of (the non-ultimately constant part of) any of the vectors mentioned in the instruction. Additionally we simulate the index-vector shifts in linear TIME. From Lemma 1 this TIME bound ensures that our overall simulation executes in quadratic TIME, which is sufficient for the inclusion given by Equation (3). The SPACE bound on the simulation is $O(|V_{\max}|^3)$. Some of the simulations are merely sketched, full proofs are to be found in [14].

We begin by giving a straightforward simulation of vector assignment.

Theorem 1 **($V_i := x$).** *The vector machine assignment instruction $V_i := x$ is simulated by a $\mathcal{C}_2$-CSM in $O(1)$ TIME, $O(1)$ GRID, $O(|x|)$ DYRANGE and $O(\max(|x|, |v_i|))$ SPATIALRES.*

Proof. The images representing x are simply copied to those representing V_i:

assignment($\overline{x}$, $\overline{|x|}$, $\overline{\mathrm{sign}(x)}$; $\overline{v_i}$, $\overline{|v_i|}$, $\overline{\mathrm{sign}(v_i)}$)

  $\overline{v_i} \leftarrow \overline{x}$
  $\overline{|v_i|} \leftarrow \overline{|x|}$
  $\overline{\mathrm{sign}(v_i)} \leftarrow \overline{\mathrm{sign}(x)}$

end // assignment

We require $O(\max(|x|, |v_i|))$ SPATIALRES to represent x and v_i as binary list images. DYRANGE of $O(|x|)$ is needed to represent $|x|$ as a natural number image. No address goes beyond the initial grid limits hence we use constant GRID. □

A $\mathcal{C}_2$-CSM can quickly generate a list image g, where each list element is identical. We state the following lemma for the specific case that each list element is a binary symbol image f_ψ. By simply changing the value of one input, the algorithm generalises to arbitrary repeated lists (with a suitable change in resource use, dependent only on the complexity of the new input image element).

Lemma 2 **(generate_list($f_\psi, l; g$)).** *A list image g that contains l list elements, each of which is a copy of input binary image f_ψ, is generated in $O(\log l)$ TIME, $O(l)$ GRID, SPATIALRES and DYRANGE.*

Proof (Sketch). The algorithm horizontally juxtaposes two copies of f_ψ and rescales them to a single image. This juxtaposing and rescaling is repeated on the new image; the process is iterated a total of $\lceil \log l \rceil$ times to give a list of length $2^{\lceil \log l \rceil}$, giving the stated TIME bound. In constant TIME, the list image is then stretched to its full length across $2^{\lceil \log l \rceil}$ images, l juxtaposed images are then selected and rescaled to a single output image g. $O(l)$ SPATIALRES is necessary to store the list in a single image. $O(l)$ GRID is used to stretch the list out to its full length. Recall that we are using natural number images for addresses, hence $O(l)$ DYRANGE is used to stretch the list across $2^{\lceil \log l \rceil}$ images. □

Theorem 2 **($V_k := \neg V_i$).** *The vector machine negation instruction $V_k := \neg V_i$ is simulated by a $\mathcal{C}_2$-CSM in $O(\log |v_i|)$ TIME, $O(|v_i|)$ GRID and DYRANGE, and $O(\max(|v_k|, |v_i|))$ SPATIALRES.*

$\neg(\overline{v_i}, \overline{|v_i|}, \overline{\mathrm{sign}(v_i)}; \overline{v_k}, \overline{|v_k|}, \overline{\mathrm{sign}(v_k)})$
  generate_list(f_{-1}, $\overline{|v_i|}$; list_neg_ones) // generate list of -1s
  $\cdot$ ($\overline{v_i}$,list_neg_ones;$\overline{-v_i}$) // change each 1 in $\overline{v_i}$ to -1
  generate_list(f_1, $\overline{|v_i|}$; list_ones) // generate list of 1s
  +($\overline{-v_i}$,list_ones;$\overline{v_k}$) // change -1s to 0s, 0s to 1s, & place in $\overline{v_k}$
  if ($\overline{\mathrm{sign}(v_i)}$ == f_1) **then** $\overline{\mathrm{sign}(v_k)} \leftarrow f_0$
  else $\overline{\mathrm{sign}(v_k)} \leftarrow f_1$ **end if**
end // $\neg$

Program 4.1: Simulation of $V_k := \neg V_i$.

Proof. Program 4.1 simulates $V_k := \neg V_i$. The program generates a list of -1s of length $|v_i|$. This list image is then multiplied by $\overline{v_i}$; changing each 1 in $\overline{v_i}$ to -1 and leaving each 0 unchanged. Then we add 1 to each element in the resulting list. A simple **if** statement negates $\overline{\mathrm{sign}(v_i)}$. Each call to the function generate_list($\cdot$) requires $O(\log|v_i|)$ TIME, otherwise TIME is constant. The remaining resource usages are for accessing vectors and rescaling them to their full length. □

The proof of the following straightforward lemma gives a program that decides which of two vectors is the longer in constant TIME. It also shows that we can decide the max or min of two integer images in constant TIME.

Lemma 3 (max($\cdot$) and min($\cdot$)). *The* max *(or* min*) length of the vectors V_i and V_j is decided in $O(1)$* TIME*, $O(1)$* GRID*, $O(\max(|v_i|, |v_j|))$* SPATIALRES*, $O(\max(|v_i|, |v_j|))$* DYRANGE*.*

Proof (Sketch). The function header for max($\cdot$) is formatted as follows:
max($\overline{v_i}$, $\overline{|v_i|}$, $\overline{\mathrm{sign}(v_i)}$, $\overline{v_j}$, $\overline{|v_j|}$, $\overline{\mathrm{sign}(v_j)}$; longest, |longest|, sign(longest))
The encoding of $-|v_i|$ is created by the instruction $\cdot$ ($\overline{|v_i|}$,f_{-1};$\overline{-|v_i|}$), then the max($\cdot$) algorithm thresholds the value $|v_j| - |v_i|$ to the range $[0, 1]$. If the result is the zero image f_0 then V_i is the longer vector and its representation is copied to the three output addresses, else the representation of V_j is output. In a similar way we decide the min length of two vector images, the function header for min($\cdot$) has the format:
min($\overline{v_i}$, $\overline{|v_i|}$, $\overline{\mathrm{sign}(v_i)}$, $\overline{v_j}$, $\overline{|v_j|}$, $\overline{\mathrm{sign}(v_j)}$; shortest, |shortest|, sign(shortest)) □

Theorem 3 ($V_k := V_i \wedge V_j$). *The vector machine instruction $V_k := V_i \wedge V_j$ is simulated by a $\mathcal{C}_2$-CSM in $O(\log\max(|v_i|, |v_j|))$* TIME*, $O(\max(|v_i|, |v_j|, |v_k|))$* SPATIALRES*, and $O(\max(|v_i|, |v_j|))$* GRID *and* DYRANGE*.*

Proof. Program 4.2 simulates $\wedge$. It uses multiplication of vector images to simulate $V_i \wedge V_j$ in parallel. However if $|v_i| \neq |v_j|$, we first pad the shorter vector image with zeros so that both have equal length. To find the longer and shorter of the two vectors we make use of the max($\cdot$) and min($\cdot$) routines given above.

The program requires $O(\log\max(|v_i|, |v_j|))$ TIME for the generate_list($\cdot$) call (the worst case is when exactly one of the vectors is of length 0). The remainder of the program runs in $O(1)$ TIME, including determining which vector is longer,

```
∧ (v̄_i, |v_i|‾, sign(v_i)‾, v̄_j, |v_j|‾, sign(v_j)‾; v̄_k, |v_k|‾, sign(v_k)‾ )
  max(v̄_i, |v_i|‾, sign(v_i)‾, v̄_j, |v_j|‾, sign(v_j)‾; longest, |longest|, sign(longest))
  min(v̄_i, |v_i|‾, sign(v_i)‾, v̄_j, |v_j|‾, sign(v_j)‾; shortest, |shortest|, sign(shortest))
  if ( sign(longest) == f_1 ) then
    ·(f_{-1},|shortest|;−|shortest|)
    +(longest,−|shortest|;difference)
    generate_list(f_1, difference; pad)
    [1,|shortest|,1,1] ← shortest
    +(|shortest|,f_1;|shortest|+1)
    [|shortest|+1,|longest|,1,1] ← pad
    padded_shortest ← [1,|longest|,1,1]
  else
    [1,|longest|,1,1] ← f_0
    [1,|shortest|,1,1] ← shortest
    padded_shortest ← [1,|longest|,1,1]
  end if
  ·(longest,padded_shortest;v̄_k)        // a single multiplication simulates v_i ∧ v_j
  ·(sign(longest),sign(shortest);sign(v_k)‾)
  |v_k|‾ ← |longest|
end // ∧
```

Program 4.2: Simulation of $V_k := V_i \wedge V_j$.

padding of the shorter vector and parallel multiplication of vectors. The remaining resource usages on vector images in the theorem statement are for accessing and storing to a single image, and stretching to full length. $\square$

Next we give algorithms to simulate vector left shift and right shift. The main idea is to copy large numbers of images to simulate shifting.

Lemma 4 (left_shift$(n, \overline{v_i}, \overline{|v_i|}, \overline{\mathrm{sign}(v_i)}; \overline{v_k}, \overline{|v_k|}, \overline{\mathrm{sign}(v_k)})$)**.** *A left shift of distance $n \geqslant 0$ on a vector V_i, to create vector V_k, is simulated in $O(1)$* TIME*, $O(|v_i + n|)$* GRID *and* DYRANGE*, and $O(\max(|v_i + n|, |v_k|))$* SPATIALRES*.*

Proof (Sketch). The algorithm assumes that n is given as a natural number image. We simulate the shift by stretching $\overline{v_i}$ out to its full length, placing n zero images to the right of the stretched $\overline{v_i}$, and then selecting all of $\overline{v_i}$ along with the n zeros and rescaling back to one image. After the shift (in accordance with the definition of vector shift), 0s are to be placed in the rightmost positions. $\square$

An algorithm for right_shift$(\cdot)$ would work similarly. However this time we select the leftmost $|v_i| - n$ images of the stretched $\overline{v_i}$. If $n \geqslant |v_i|$ the output is the representation of the zero vector.

Theorem 4 ($V_k := V_i \uparrow V_j$)**.** *The vector machine instruction $V_k := V_i \uparrow V_j$ is simulated by a $\mathcal{C}_2$-CSM in $O(|v_j|)$* TIME*, $O(|v_i| + 2^{|v_j|})$* GRID *and* DYRANGE*, and $O(\max(|v_k|, |v_i| + 2^{|v_j|}))$* SPATIALRES*.*

$\uparrow$ ($\overline{v_i}$, $\overline{|v_i|}$, $\overline{\mathrm{sign}(v_i)}$, $\overline{v_j}$, $\overline{|v_j|}$, $\overline{\mathrm{sign}(v_j)}$; $\overline{v_k}$, $\overline{|v_k|}$, $\overline{\mathrm{sign}(v_k)}$)
  shift_distance $\leftarrow f_0$
  current_bit $\leftarrow \overline{|v_j|}$
  current_power_2 $\leftarrow f_1$
  [1,$\overline{|v_j|}$,0,0] $\leftarrow \overline{v_j}$
  ρ($\overline{|v_j|}$,f_0,f_1;flag)
  while (flag == f_1) **do**
    if ($\overline{\mathrm{sign}(v_j)}$ == f_0) **then**
      if ([current_bit,current_bit,0,0] == f_1) **then**
        +(shift_distance,current_power_2;shift_distance)
      end if
    else
      if ([current_bit,current_bit,0,0] == f_0) **then**
        +(shift_distance,current_power_2;shift_distance)
      end if
    end if
    $\cdot$ (current_power_2,f_2;current_power_2)
    +(current_bit,f_{-1};current_bit)
    ρ(current_bit,f_0,f_1;flag)
  end while
  if ($\overline{\mathrm{sign}(v_j)}$ == f_0) **then**
    left_shift(shift_distance, $\overline{v_i}$, $\overline{|v_i|}$, $\overline{\mathrm{sign}(v_i)}$; $\overline{v_k}$, $\overline{|v_k|}$, $\overline{\mathrm{sign}(v_k)}$)
  else right_shift(shift_distance, $\overline{v_i}$, $\overline{|v_i|}$, $\overline{\mathrm{sign}(v_i)}$; $\overline{v_k}$, $\overline{|v_k|}$, $\overline{\mathrm{sign}(v_k)}$) **end if**
end // $\uparrow$

Program 4.3: Simulation of $V_k := V_i \uparrow V_j$.

Proof. Program 4.3 simulates the shift by stretching V_i out to its full length; then selecting either part of V_i, or V_i and some extra zero images; and finally rescaling back to one image. The simulator's addresses are represented by natural number images whereas vectors are represented by binary list images. In order to perform the stretching the program converts the binary number defined by V_j to a natural number image called shift_distance.

The **while** loop efficiently generates a value of $O(2^{|v_j|})$ in $O(|v_j|)$ TIME. At different stages of the algorithm each of $\overline{v_i}$ and $\overline{v_j}$ are rescaled to their full length, across $|v_i|$ and $|v_j|$ images respectively. We get the value $O(|v_i| + 2^{|v_j|})$ for GRID since in the worst case V_i is left shifted by the value $2^{|v_j|}$, and (when stretched) the resulting vector spans $O(|v_i| + 2^{|v_j|})$ images. This upper bound also covers the right shift case (when V_j is negative). Analogously we get the same value for SPATIALRES and DYRANGE (except $|v_k|$ is also in the SPATIALRES expression as it could contain some values before the program executes). $\square$

The converse shift instruction ($V_k := V_i \downarrow V_j$) is simulated by Program 4.3 except that the calls to right_shift($\cdot$) and left_shift($\cdot$) are exchanged. The resource usage remains the same.

The proof of the following lemma gives a log TIME algorithm to decide if a list or vector image represents a word that consists only of zeros. It is possible to give a constant TIME algorithm that makes use of the FT (to 'sum' the entire list in constant TIME). Not using the FT enables us to state Corollary 6.

Lemma 5. *A $\mathcal{C}_2$-CSM that does not use Fourier transformation decides whether or not a list (equivalently vector) image $\overline{v_i}$ represents the word $0^{|v_i|}$ in $O(\log|v_i|)$ TIME, $O(|v_i|)$ GRID, SPATIALRES and DYRANGE.*

Proof (Sketch). The binary list image $\overline{v_i}$ is padded with zeros so that is of length $2^{\lceil \log|v_i| \rceil}$. The algorithm splits $\overline{v_i}$ into a left half and a right half, adds both halves (in a one step parallel pointwise fashion), and repeats until the list is of length 1. A counter image keeps track of list length. The resulting image is thresholded below by f_0 and above by f_1. If the result is the zero image then $\overline{v_i}$ represents a list of zeros, otherwise $\overline{v_i}$ represents a list with at least one 1. □

Theorem 5 (goto m if $V_i = 0$**).** *The vector machine instruction* goto m if $V_i = 0$ *(or* goto m if $V_i \neq 0$*) is simulated by a $\mathcal{C}_2$-CSM in $O(\log|v_i|)$ TIME, $O(|v_i|)$ GRID, SPATIALRES, and DYRANGE.*

Proof. Due to the vector machine number representation, there are exactly two representations for 0; the constant sequences $\ldots 000$ and $\ldots 111$. Using our $\mathcal{C}_2$-CSM representation of vectors, if $\overline{|v_i|} = 0$ then the vector V_i is constant, and hence represents 0. We can test $\overline{|v_i|} = 0$ in constant time with an **if** statement.

However, it may be the case that $\overline{|v_i|} = n > 0$ and yet V_i represents 0. In this case $\overline{v_i}$ represents a list of 0s (respectively 1s) and $\overline{\mathrm{sign}(v_i)}$ represents 0 (respectively 1). A sequential search through $\overline{v_i}$ will require exponential TIME (worst case) and as such is too slow. Instead we use the log TIME technique given by the previous lemma. In the case that V_i is ultimately 1 we make use of the $\neg(\cdot)$ program defined in Theorem 2. For the goto part of the instruction we merely note that gotos are simulated by **if**s and **while**s. Clearly the related instruction 'goto m if $V_i \neq 0$' is simulated with the same resource usage. □

Given a vector machine M there is a $\mathcal{C}_2$-CSM M' that simulates M. In particular, if vector machine M decides a language L then we can easily modify our simulation of vector machines so that M' decides L.

Theorem 6. *Let M be an index-vector machine that decides $L \in \{0,1\}^*$ in time $T(n)$ for input length n. Then L is decided by a $\mathcal{C}_2$-CSM M' in $O(T^2(n))$ TIME, $O(2^{T(n)})$ GRID, SPATIALRES and DYRANGE.*

Proof. By Lemma 1 M's index-vectors have length $O(T(n))$, while unrestricted vectors have length $O(2^{T(n)})$. From the above simulation theorems, any non-shifting instruction is simulated in TIME that is log of the length of the vectors. The remaining operations, right and left shift, are simulated in TIME that is linear in the length of their index-vector input. From these bounds it is straightforward to work out that M decides L in $O(T^2(n))$ TIME and that each of GRID, SPATIALRES and DYRANGE is $O(2^{T(n)})$. □

From the previous theorem M' uses $O(2^{3T(n)})$ SPACE to decide L, hence our simulation uses SPACE that is cubic in the space of M.

Corollary 1. $\mathcal{V}_I$–TIME$(T(n)) \subseteq \mathcal{C}_2$-CSM–TIME$(O(T^2(n)))$

Let $S(n) = \Omega(\log n)$. From the inclusion in Equation (1) we get:

Corollary 2. NSPACE$(S(n)) \subseteq \mathcal{C}_2$-CSM–TIME$(O(S(n) + \log n)^4)$

Combining this result with the upper bound on TIME bounded $\mathcal{C}_2$-CSM power [14]:

Corollary 3. NSPACE$(S(n)) \subseteq \mathcal{C}_2$-CSM–TIME$(O(S(n) + \log n)^4)$
$\subseteq$ DSPACE$(O(S(n) + \log n)^8))$

To summarise, the $\mathcal{C}_2$-CSM satisfies the parallel computation thesis:

Corollary 4. NSPACE$(S^{O(1)}(n)) = \mathcal{C}_2$-CSM–TIME$(S^{O(1)}(n))$

This links space bounded sequential computation and TIME bounded $\mathcal{C}_2$-CSM computation. For example $\mathcal{C}_2$-CSM–TIME$(n^{O(1)})$ = PSPACE. We strengthen this result by restricting the $\mathcal{C}_2$-CSM. Let a 1D-$\mathcal{C}_2$-CSM be a $\mathcal{C}_2$-CSM with constant GRID and SPATIALRES, in one of the vertical or the horizontal directions.

Corollary 5. *The* 1D-$\mathcal{C}_2$-CSM *verifies the parallel computation thesis.*

Proof. The index-vector machine simulation used only constant GRID and SPATIALRES in the vertical direction. Moreover we can rotate the grid layout and all images by 90°, to obtain a simulation where GRID and SPATIALRES are constant in the horizontal direction only. □

Corollary 6. *The* $\mathcal{C}_2$-CSM *without the DFT operations h and v verifies the parallel computation thesis.*

Proof. Our $\mathcal{C}_2$-CSM simulation of index-vector machines did not use h nor v. □

The thesis relates parallel time to sequential space, however in our simulations we explicitly gave *all* resource bounds. As a final result we show that the class of $\mathcal{C}_2$-CSMs that simultaneously use polynomial SPACE and polylogarithmic TIME decide at least the languages in NC. Let $\mathcal{C}_2$-CSM–SPACE, TIME$(S(n), T(n))$ be the class of languages decided by $\mathcal{C}_2$-CSMs that use SPACE $S(n)$ and TIME $T(n)$. It is known [5] that $\mathcal{V}_I$–SPACE, TIME$(n^{O(1)}, \log^{O(1)} n)$ = NC. From the resource overheads in our simulations:

$$\begin{aligned} &\mathcal{V}_I\text{–SPACE, TIME}(O(2^{T(n)}), T(n)) \\ &\subseteq \mathcal{C}_2\text{-CSM–SPACE, TIME}(2^{O(T(n))}, T^{O(1)}(n)) \end{aligned}$$

For the case of $T(n) = \log^{O(1)} n$ we have our final result.

Corollary 7. NC $\subseteq \mathcal{C}_2$-CSM–SPACE, TIME$(n^{O(1)}, \log^{O(1)} n)$

In [14] it was shown that the converse inclusion also holds.

5 Discussion

We have given lower bounds on $\mathcal{C}_2$-CSM power in terms of space and NC complexity classes via simulation of index-vector machines. The quadratic time bound for index-vector simulation is reasonably tight. In a certain sense this is not surprising; both are SIMD models. Possibly the power in Corollary 2 could be reduced from 4 to 2 by direct Turing machine simulation.

The simulation uses the reasonable (we argue) natural number representation of images. This incurs a DYRANGE cost that is a constant times the longest vector. Since the binary values in vectors are represented by images with constant DYRANGE, it would be interesting if another addressing scheme could be employed that works (say) on binary values (e.g. binary list image addresses). We believe that DYRANGE could be reduced without a significant increase in the other measures. Simultaneously, a constant GRID simulation might be possible, the main problem is to simulate vector shifts while using at most constant GRID. By using these trade-offs we conjecture that SPACE can be reduced to be linear in index-vector machine space, with only a polynomial increase in TIME.

In addition to fulfilling our needs of giving a lower bound on $\mathcal{C}_2$-CSM power, the results in this paper are useful to the practitioner since we have given a method to directly translate vector machine algorithms to optical algorithms.

Interestingly we did not make use of Fourier transformation in the simulation. Optical computers are often celebrated for having a constant time FT operation. Our results prove that the $\mathcal{C}_2$-CSM has remarkable power without explicitly using the FT. However in a more fine grained analysis, say using the $\mathcal{C}_2$-CSM to design algorithms for NC and AC problems, some advantages of Fourier transformation would be observed.

Acknowledgements

We thank Tom Naughton for interesting discussions. The first author is funded by the Irish Research Council for Science, Engineering and Technology.

References

1. J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity, vols I and II.* EATCS Monographs on Theoretical Computer Science. Springer, Berlin, 1988.
2. A. K. Chandra and L. J. Stockmeyer. Alternation. In *In 17th annual symposium on Foundations of Computer Science*, pages 98–108, Houston, Texas, Oct. 1976. IEEE. Preliminary Version.
3. L. M. Goldschlager. *Synchronous parallel computation.* PhD thesis, University of Toronto, Computer Science Department, Dec. 1977.
4. R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to parallel computation: P-completeness theory.* Oxford university Press, Oxford, 1995.
5. R. M. Karp and V. Ramachandran. *Parallel algorithms for shared memory machines.* Volume A of van Leeuwen [13], 1990.

6. 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, San Diego, California, Aug. 2000.
7. 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.
8. 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*, volume 2055 of *Lecture Notes in Computer Science*, pages 288–299, Chişinău, Moldova, May 2001. Springer.
9. I. Parberry. *Parallel complexity theory.* Wiley, 1987.
10. 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.
11. 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.
12. P. van Emde Boas. *Machine models and simulations.* Volume A of van Leeuwen [13], 1990.
13. J. van Leeuwen, editor. *Handbook of Theoretical Computer Science*, volume A. Elsevier, Amsterdam, 1990.
14. D. Woods. *Computational complexity of an optical model of computation.* PhD thesis, National University of Ireland, Maynooth, 2004. Submitted.
15. 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*, volume 3526 of *Lecture Notes in Computer Science*, pages 540–551, Amsterdam, June 2005. Springer.
16. D. Woods and T. J. Naughton. An optical model of computation. *Theoretical Computer Science*, 334(1–3):227–258, Apr. 2005.