

Space-Efficient Simulations Beyond Multitape Turing Machines*

Danil Sibgatullin[†]

danil.sibgatullin@gmail.com

Ryan Williams[‡]

MIT

rrw@mit.edu

Abstract

We explore new directions in simulating complex computations with low space, building on the work of Williams [STOC'25] and Cook-Mertz [STOC'24, SICOMP'25]. We define and study a strengthened version of the parallel external memory model, in which there are P processors each with private internal memory M , all of which have shared access to an external memory. Like the standard external memory model, a processor is charged one step when it swaps up to M bits of its internal memory with the external memory; unlike the usual external memory model, each processor also has free access to the input at no cost, and may read any subset of M bits of external memory (not just a contiguous block) in a single I/O step.

We extend the reduction of Williams for multitape Turing machines to this vastly more general model, giving three main applications.

1. We show that the P-complete problem LEX FIRST 2SAT, where even the best-known RAM algorithm runs in $O(n^\omega) \leq O(n^{2.372})$ time on instances with $\Theta(n^2)$ clauses, nevertheless has an $\tilde{O}(\sqrt{n})$ -space algorithm for n -variable instances. For dense instances with $\Theta(n^2)$ clauses, our space usage is less than the *fourth root* of the best-known RAM running time.

2. We show that every *unbounded* fan-in circuit of n gates over the basis {NOT, AND, OR, XOR} can be evaluated on any given input in only $O(\sqrt{n \log n})$ space. That is, the space usage is nearly a fourth root of the running time for dense circuits with $\Theta(n^2)$ wires. This result generalizes a space-efficient simulation of Shalunov for bounded fan-in circuits.

3. We consider the problem of simulating a d -dimensional cellular automaton, where the value of each cell is determined by its immediate neighbors. We show how to simulate t time steps of a cellular automaton in only $O((t \log t)^{1-1/(d+1)})$ space. By an old theorem of Cook [1966], our space-efficient simulation of one-dimensional cellular automata yields another generalization of Williams' simulation of multitape time.

*This is the full version of a paper which will appear in FOCS 2026.

[†]Work done as part of the author's M.Eng thesis at MIT.

[‡]Work supported in part by NSF DMS-2424441 (at IAS) and NSF CCF-2420092 (at MIT).

1 Introduction

A central problem in computational complexity is to understand the relationships between time and space as resources. In particular, the following question is paramount:

Let f be a function with a time- t algorithm. How much space is required to compute f ?

If we know a bound on the running time for computing a function, what can we say about the space complexity of that function? For most reasonable computational models, the space usage is no more than about t : at most one new unit of space is consumed for each unit of time (however it is that units are being measured). Any nontrivial and generic space-bounded simulations of time-bounded algorithms imply lower bounds *against* time-bounded algorithms, by the space hierarchy theorem.

Already in the early days of complexity theory, researchers discovered interesting and highly nontrivial answers to this question. Hopcroft and Ullman [HU68] showed that the restricted model of single-tape Turing machines running in time t could be simulated using space only $O(\sqrt{t})$; this simulation was later improved [Pat72]. Paterson and Valiant [PV76] showed that size- s Boolean circuits can be simulated by depth- $O(s/\log s)$ Boolean circuits; in concurrent work, Hopcroft, Paul, and Valiant [HPV77] showed that multitape Turing machines running in time t can be simulated in space only $O(t/\log t)$. By the space hierarchy theorem, this yields functions computable in linear space that cannot be solved in $o(n \log n)$ time. Their result was later generalized to random-access models and pointer machines [PR81, HLMW86].

In a recent surprise, Williams [Wil26] showed that multitape Turing machines running in time t can in fact be simulated in space only $O(\sqrt{t \log t})$, by applying a breakthrough space-efficient algorithm of Cook and Mertz [CM25] for the TREE EVALUATION problem. Due to the power of multitape Turing machines, Williams' simulation implies novel low-space algorithms for a variety of problems, including the P-complete CIRCUIT VALUE PROBLEM; in particular, every bounded fan-in circuit of size s can be evaluated on any input using space only $\sqrt{s} \cdot \text{poly}(\log s)$. This space bound was recently improved to $O(\sqrt{s \log s})$ by Shalunov [Sha26]. Williams' result yields functions in linear space that require nearly quadratic time on a multitape Turing machine, a step towards $P \neq PSPACE$.

However, the multitape Turing machine model is not the most powerful model one could reasonably consider. For example, one could consider the standard Word RAM model used in modern core algorithms, and it is only known that time- t in such models can be simulated by multitape (and one-tape!) Turing machines using time about $t^2 \cdot \text{poly}(w)$ for wordsize w (see [AB09] for a related construction). So on the face of it, Williams' square-root result does not say anything nontrivial about the space complexity of time- t Word RAMs. It is possible that multitape Turing machines can simulate RAMs more time-efficiently, but this is a notorious decades-old open problem. Perhaps we can improve the space complexity of simulating more powerful computational models without resolving such questions. Arguably, the most significant open problem in this area is either to generalize Williams' space-efficient simulation to random-access models, or to give evidence that no such simulation exists.

1.1 Our Results

In this paper, we take concrete steps towards reducing the space complexity of computational models beyond multitape Turing machines.

Space-Efficient Simulation of Parallel External Memory Machines. We define a strengthened version of the parallel external memory model [AV88, AGNS08], which we call the FP_{EM} (Free-Input Parallel External Memory) model. In the usual parallel external memory (PEM) model [AGNS08], there are P

processors, each with private internal memory M and with shared access to external memory. In each I/O round, each processor may swap up to M bits of its internal memory with the external memory. (The usual model has a smaller blocksize B for I/O swaps; we will set $M = B$ for simplicity.) Our model has all these properties. However, unlike the usual external memory model, we also allow each processor *free access to any bits of the input* at no cost in each I/O round. Moreover, in our model the processors are not bound to merely read contiguous blocks of external memory, but rather they can randomly read any desired subset of M bits from the external memory in one I/O round. These extra properties make the FPEM model rather powerful, especially for large M and P : we effectively allow arbitrary $O(M)$ -space computations between each I/O round. (The model also circumvents many lower bounds in the external memory model which are proved by information-theoretic bottlenecks between the processors and the input.)

A naïve simulation of an FPEM model with a single (Turing-based) processor takes $O(T \cdot M)$ bits of space, where T is the total number of I/O rounds. We prove that the factor of $T \cdot M$ can be reduced to nearly $T + M$ when P is not enormous. Say that functions $T, S, P, M : \mathbb{N} \rightarrow \mathbb{N}$ are *good* if their values $T(n)$, $S(n)$, $P(n)$, $M(n)$ can be computed (given n in binary) in $O(T \cdot \log(P \cdot S \cdot M) + M)$ space.

Theorem 1.1. *Let T, S, P, M be good. Every algorithm in the free-input parallel external memory model (Section 3) using T I/O rounds, internal memory of size $M \geq \log n$ (where n is the input length), external memory of size S , and P processors can be simulated in $O(T \cdot \log(P \cdot S \cdot M) + M)$ space (by a single-processor machine).¹*

We observe that Theorem 1.1 subsumes Williams’ simulation of multitape time T in space $O(\sqrt{T \log T})$. The authors of [DLL⁺18] proved that multitape time- T can be simulated in the standard one-processor external memory model using only $O(T/B)$ I/Os (see Lemma 5.20 in the full version). Noting in our case that $M = B$, setting $M = \sqrt{T \log T}$, the simulation of Williams is an easy corollary.

Williams’ space-efficient simulation of multitape Turing machines [Wil26] proceeds via a series of reductions to the TREE EVALUATION problem, and calls to the Cook-Mertz algorithm [CM25]. To prove Theorem 1.1, we give a generalization of the Cook-Mertz algorithm for evaluating circuits space-efficiently, defining and utilizing a natural measure of *normalized depth* (see Section 2). Then, we reduce the simulation of an FPEM machine to our more general evaluation problem. Our reduction turns out to be quite intricate to handle the interactions between processors and shared memory.

Using Theorem 1.1, we are able to establish highly nontrivial space upper bounds on problems that do not appear to be efficiently solvable by multitape Turing machines.

Lex First 2SAT. The LEX FIRST 2SAT problem asks: *given a 2CNF formula F and an ordering on its variables, determine the lexicographically first satisfying assignment to F .* The decision version (where we ask for the i -th bit of such an assignment) is P-complete, so it is strongly believed that LEX FIRST 2SAT should *not* have highly space-efficient algorithms.² LEX FIRST 2SAT with n variables and $m \leq O(n^2)$ clauses can be solved in $O(n^\omega) \leq O(n^{2.372})$ time on a RAM (where ω is the matrix multiplication exponent [ADV⁺25]) by computing the transitive closure of the corresponding directed graph for 2SAT [APT79] and querying its edges appropriately.³ Thus, if $\omega = 2$, then there is a nearly-linear time algorithm for dense

¹In particular, any desired bit of the output of such an algorithm can be determined in such space.

²Although 2SAT is well-known to be NL-complete [AB09], finding the lex-first 2SAT assignment is P-complete: in particular, for a graph $G = ([n], E)$, the lex-first satisfying assignment of $\bigwedge_{\{u,v\} \in E} (x_u \vee x_v)$ sets $x_v = 0$ iff v is in the lexicographically first maximal independent set of G , which is P-complete [Coo85].

³In particular, when the formula is satisfiable, the lex-first assignment sets $x_j = 1$ iff the transitive closure has an edge $\neg x_j \rightarrow x_j$, or there are $i < j$ and an edge $\ell \rightarrow x_j$ from a literal ℓ on x_i that the assignment sets true. This follows from the proof of Lemma 4.2.

instances, but currently even the best-known RAM algorithm requires more than linear time. By designing a parallel external memory algorithm for the problem and applying Theorem 1.1, we can solve the problem with substantial space savings:

Theorem 1.2. *LEX FIRST 2SAT on 2CNFs with n variables and $O(n^2)$ clauses can be solved in $O(\sqrt{n \log n})$ space.*

Observe that for dense 2CNF formulas, our algorithm actually uses space *less than the fourth root* of the best-known running time on a RAM. This is a byproduct of our ability to simulate arbitrary space-limited computations in parallel.

Simulating Circuits. It is straightforward to design an efficient circuit evaluation procedure in our FPDM model; by doing so, we recover an $O(\sqrt{n \log n})$ -space circuit evaluation algorithm for bounded fan-in circuits, matching Shalunov [Sha26]. However, with our model we can go further: we can handle *unbounded fan-in* circuits with the same space bound.

Theorem 1.3. *Any size- n Boolean circuit with gates from $\{\text{NOT}, \text{AND}, \text{OR}, \text{XOR}\}$ and arbitrary fan-in can be evaluated in space $O(\sqrt{n \log n})$.*

Note that the evaluation of such a circuit may take $\Theta(n^2)$ time (there may be $\Theta(n^2)$ wires), even on a random-access machine. Therefore our $\tilde{O}(\sqrt{n})$ space savings is especially notable, and indicates that there is much more to be found.

Simulating Cellular Automata. Finally, we study the problem of simulating d -dimensional cellular automata, a massively parallel model where each cell’s value from one time step to the next depends on all neighboring values; see [Kar12] for a survey.

A d -dimensional cellular automaton (CA) is given by a positive integer constant r (the “radius” of each cell), a finite set of states Σ , an initial configuration $\sigma_0: \mathbb{Z}^d \rightarrow \Sigma$, and an evolution rule $\varphi: \Sigma^{(2r+1)^d} \rightarrow \Sigma$. The r -neighborhood of a cell c is the collection of cells within distance r of c . On each step, every cell $(x_1, \dots, x_d) \in \mathbb{Z}^d$ updates its state based on the values in its r -neighborhood:

$$\sigma_{t+1}(x_1, \dots, x_d) = \varphi\left(\left[\sigma_t(x_1 + \delta_1, \dots, x_d + \delta_d)\right]_{|\delta_i| \leq r}\right).$$

We define the d -dimensional cellular automata evaluation problem (CA^d) to be: given (for fixed d, r , Σ, φ) a non-negative integer t and the initial states of all $(2rt + 1)^d$ cells within ℓ^∞ distance rt of a cell c , compute $\sigma_t(c)$, the value of the cell c after t time steps.

Cook’s PhD thesis [Coo66] (see also [CA69, Theorem 5.3] and Cole [Col64]) demonstrates that one-dimensional automata (called *one-dimensional iterative arrays* in his work) can simulate arbitrary multitape Turing machines with constant runtime overhead. Therefore, our next theorem also subsumes Williams’ space-efficient simulation of multitape Turing machines, in a different way:

Theorem 1.4. *The CA^1 problem for t time steps can be computed in $O(\sqrt{t \log t})$ space.*

As with unbounded fan-in circuit evaluation, the space usage is again nearly a *fourth root* of the standard $O(t^2)$ evaluation time on a (single-processor) RAM. More generally:

Theorem 1.5. *For every fixed $d \geq 1$, the CA^d problem for t time steps can be computed in $O((t \log t)^{1-1/(d+1)})$ space.*

Note that the naïve simulation of CA^d on a RAM takes $\Theta(t^{d+1})$ time: there may be $\Omega(t^d)$ cells to track over t time steps. Theorem 1.5 extends an analogous result of [Wil26] for d -dimensional multitape Turing machines, to the massively parallel setting of cellular automata. By the space hierarchy theorem, it follows that for all $d \geq 1$, there are problems decidable in linear space which cannot be decided by d -dimensional cellular automata in $n^{1+1/d}/(\log^c n)$ steps for every $c > 1$.

Beyond the results we have shown in this paper, we are optimistic that Theorem 1.1 can be applied further to show that a variety of other problems have surprisingly space-efficient algorithms, leveraging the vast literature on external memory algorithms (see [Vit06] for a survey). It seems quite plausible to us that random-access machines running in time t can be simulated in $t^{1-\varepsilon}$ I/O rounds of an FPEM model with internal memory $M \leq t^{1-\varepsilon}$ and $P, S \leq \text{poly}(t)$, for some $\varepsilon > 0$. This would extend Williams' space-saving simulation to random-access models.

2 A Generalized Evaluation Problem and Cook-Mertz Procedure

We begin by defining a problem that generalizes TREE EVALUATION, followed by a generalization of the Cook-Mertz procedure that solves the problem.

Recall that in the TREE EVALUATION problem [CMW⁺12], for integer parameters k and h , we are given:

- a binary tree of height h with functions at each internal node,
- each internal node function takes two k -bit values from their children, and outputs a k -bit value, and
- k -bit values on all the leaves.

Note the functions are provided as full 2^{2k} -size lookup tables to the input. We can naturally define the value of every node of the tree inductively. The goal of the problem is to compute the value of the root of the tree. Cook and Mertz's amazing algorithm [CM25] solves this problem using only $O(h \log k + k)$ space.

In this section, we prove a generalization of the Cook-Mertz algorithm for TREE EVALUATION which can yield a more effective space bound when the vertices of the tree have varying indegrees at each node. We believe that our model and corresponding space bound yield some new insight into Cook-Mertz-like *catalytic computations*.

The b -Bit Circuit Value problem. Throughout this section, a b -bit circuit $\mathcal{C}$ is a directed acyclic graph on n vertices $\{v_1, \dots, v_n\}$ where $d_i = \text{in-deg}(v_i) \leq b$, together with an ordering on the edges entering every vertex. For all $i \in [n]$, we have a function $\varphi_i : \{0, 1\}^{d_i} \rightarrow \{0, 1\}$ that is *uniformly computable*: the bit $\varphi_i(x_1, \dots, x_{d_i})$ can be computed from the input $\langle i; x_1, \dots, x_{d_i} \rangle$ (with read-only access to the input) in $O(b + \log n)$ space. Moreover, given $i \in [n]$ in binary, we assume $d_i = \text{in-deg}(v_i)$ is computable in $O(b + \log n)$ space, and given $i \in [n]$ and $j \in [d_i]$, we assume the j -th predecessor of i is also computable in $O(b + \log n)$ space.

Note that for nodes v_i which are sources, φ_i has no inputs and defines a *bit assignment* to v_i . By induction, every node v in $\mathcal{C}$ has a value, obtained by applying φ_v to the values of v 's predecessors.

Given the above setup, we can naturally define the b -BIT CIRCUIT VALUE problem: given a b -bit circuit $\mathcal{C}$ of n nodes and $i \in [n]$, our task is to compute the value of v_i in $\mathcal{C}$.

This Generalizes Tree Evaluation? At this point, one may (rightly) worry that our b -BIT CIRCUIT VALUE problem does not look at all like the usual TREE EVALUATION problem [CMW⁺12], and yet our aim is to generalize the Cook-Mertz algorithm for TREE EVALUATION. As discussed above, the usual TREE EVALUATION problem has a tree of height h , functions at each internal node which take two k -bit values from the children and output k bits, and k -bit values on all the leaves. In our b -BIT CIRCUIT VALUE PROBLEM, node functions only have single bits on the wires, not k bits. Our nodes only take in up to b bits each (a bit from each predecessor) and output a single bit. Furthermore, we are “only” working with a DAG, and not a tree.

It turns out that b -BIT CIRCUIT VALUE does generalize TREE EVALUATION. The first thing to note is that we have not put a bound on the number of nodes in the DAG circuit, so trees are actually a special case for us. Secondly, each $2k$ -bit to k -bit function in a TREE EVALUATION instance can be readily implemented in our b -bit circuit model by setting $b = 2k$ and placing k nodes in parallel to compute each bit of the k -bit output (increasing the total number of nodes by a factor of k , but keeping the depth the same). Finally, in the usual TREE EVALUATION problem, the uniformity condition is obvious because the functions are given as entire lookup tables in the input.

Normalized depth. Next, we define our crucial notion of depth which will be used to bound the space complexity of our generalized algorithm.

Definition 2.1. *Let G be a directed acyclic graph and v be a vertex of G . Then the normalized depth of v is defined as follows:*

$$\bar{h}_G(v) = \max_{u_1, \dots, u_\ell} \sum_{i=1}^{\ell} \lceil \log_2(\text{in-deg}(u_i) + 1) \rceil,$$

where the maximum is taken over all directed paths $u_1, u_2, \dots, u_\ell = v$, from some source u_1 to the node v .

Likewise, the normalized depth of a DAG is the maximum normalized depth of its vertices:

$$\bar{h}(G) = \max_{v \in G} \bar{h}_G(v).$$

Finally, the normalized depth $\bar{h}(\mathcal{C})$ of a circuit is the normalized depth of the underlying DAG.

As an instructive example, suppose G is a tree of height h , in which all internal nodes have indegree $b = 2k$, as in our given reduction from TREE EVALUATION to b -BIT CIRCUIT VALUE. Then all leaf-to-root paths have length h , and $\bar{h}(G)$ is $O(h \cdot \log k)$, which is precisely one of the terms in the Cook-Mertz space bound for TREE EVALUATION. The main theorem of this section is the following result that relates the space complexity of b -BIT CIRCUIT VALUE to the normalized depth:

Theorem 2.1. *Every gate of a b -bit n -gate circuit $\mathcal{C}$ can be evaluated in $O(\bar{h}(\mathcal{C}) + b + \log n)$ space.*

2.1 Circuit evaluation

The evaluation procedure relies on representing gate functions as low-degree polynomials over a finite field, then recovering the output via Lagrange interpolation. We begin by recalling some necessary algebraic tools.

Proposition 2.2 (Low-Degree Extension cf. [BFL91]). *Let $S_1, \dots, S_m$ be subsets of a field $\mathcal{K}$ and set $S = S_1 \times \dots \times S_m$. Consider a function $\varphi : S \rightarrow \mathcal{K}$. Then there is a polynomial $f \in \mathcal{K}[t_1, \dots, t_m]$ that agrees with φ on S such that*

$$\deg(f) \leq \sum_{i=1}^m (|S_i| - 1).$$

Moreover, f is of the form

$$f(x_1, \dots, x_m) = \sum_{a \in S} \varphi(a) \cdot w_a(x_1, \dots, x_m),$$

where $w_a(x)$ is computable in $O(m \log |\mathcal{K}|)$ space given an efficient test of membership for all S_i .

Proof. Using the explicit formula

$$w_a(x) = \prod_{i=1}^m \prod_{b_i \in S_i \setminus \{a_i\}} \frac{x_i - b_i}{a_i - b_i},$$

we compute

$$f(x) = \sum_{a \in S} \varphi(a) \cdot w_a(x)$$

by iterating over all $a \in S$ and maintaining a running sum, using $O(m \log |\mathcal{K}|)$ space. $\square$

Corollary 2.3 (Lagrange Interpolation at Zero [CM25]). *Let $a_1, \dots, a_d$ be pairwise distinct non-zero elements of a field $\mathcal{K}$. There exist $c_1, \dots, c_d$ such that for any polynomial $f \in \mathcal{K}[t]$ with $\deg(f) < d$ it holds that*

$$f(0) = \sum_{j=1}^d c_j \cdot f(a_j).$$

Moreover, such c_j can be computed in space $O(\log |\mathcal{K}|)$. In particular, for any $r, x \in \mathcal{K}$ the following equality holds whenever $\deg(f) < d$:

$$\sum_{j=1}^d c_j \cdot f(a_j r + x) = f(x),$$

where we interpolate $f(tr + x)$ at $t = 0$.

To apply these interpolation tools to gate functions $\varphi_v : \{0, 1\}^d \rightarrow \{0, 1\}$, where d can vary from one node to the next, we need to embed the Boolean hypercube $\{0, 1\}^d$ into a product of a small number of subsets of a small finite field, in a way that keeps the interpolating polynomial's degree less than $2d^2$. The following construction achieves this by building a nested family of $\mathbb{F}_2$ -vector subspaces.

Definition 2.2. Assume that $\mathcal{F} = \mathbb{F}_{2^k}$ and fix some integer m . Select an element z such that $1, z, \dots, z^{k-1}$ are linearly independent over $\mathbb{F}_2$. For all $i \leq k$, define

$$W_i = \langle 1, z, \dots, z^{i-1} \rangle_{\mathbb{F}_2} \subseteq \mathcal{F}$$

to be an $\mathbb{F}_2$ -vector subspace of $\mathcal{F}$ of dimension $i \leq k$. Moreover, for all $r < m$, let

$$V_{\ell m + r} = W_\ell^{m-r} \times W_{\ell+1}^r \subseteq \mathcal{F}^m,$$

so that V_n is well-defined for all $n \leq km$ and $\dim(V_n) = n$.

Intuitively, V_n describes a way of storing n bits among m field elements. Think of an element of $\mathcal{F}$ as a k -bit word, where the j -th bit is the coefficient of z^j . Then W_i is the set of all words whose bits $i, \dots, k-1$ are all zero. V_n is the set of m -tuples of words in which n designated bit positions hold arbitrary values, and all other bits are zero. These n positions are spread out as evenly as possible among the m words, filling the least significant positions first: writing $n = \ell m + r$ with $0 \leq r < m$, the first $m - r$ words contain ℓ of the positions and the last r words contain $\ell + 1$ of the positions.

Proposition 2.4. *Let $\mathcal{F}$ and m be defined as in Definition 2.2, and let $\varphi : V_n \rightarrow \mathcal{F}$ be any function (not necessarily linear) with $1 \leq n \leq m \log_2(m)$ and $k \geq \log_2(m)$. Consider the polynomial $f \in \mathcal{F}[t_1, \dots, t_m]$ that agrees with φ on $V_n \subseteq \mathcal{F}^m$, constructed as in Proposition 2.2. Then $\deg(f) < 2n^2$.*

(Note we need the condition $k \geq \log_2(m)$ in the proposition, so that $n \leq km$ and V_n is well-defined.)

Proof. Let $n = \ell m + r$ with $r < m$. By Proposition 2.2,

$$\begin{aligned} \deg(f) &\leq \sum_{i=1}^m (|S_i| - 1) \\ &= (m - r) \cdot (|W_\ell| - 1) + r \cdot (|W_{\ell+1}| - 1) \\ &= (m - r) \cdot (2^\ell - 1) + r \cdot (2^{\ell+1} - 1) \\ &= (m + r) \cdot 2^\ell - m. \end{aligned}$$

When $n < m$, we have $\ell = 0$, so $\deg(f) \leq (m + r) - m = r = n < 2n^2$.

Otherwise, $n \geq m$. Observe that we must have $\ell \leq \log(n)$: since $\ell m \leq n \leq m \log_2 m$, we have $2^\ell \leq m \leq n$. Therefore we may conclude

$$\deg(f) \leq (m + r) \cdot 2^\ell - m < m \cdot 2^{\ell+1} \leq 2mn \leq 2n^2.$$

□

Thus, after identifying $\{0, 1\}^d$ with V_d , every gate function $\varphi_v : \{0, 1\}^d \rightarrow \{0, 1\}$ with $1 \leq d \leq m \log_2 m$ agrees on $V_d \subseteq \mathcal{F}^m$ with a polynomial of degree less than $2d^2$. As m and $\mathcal{F}$ do not depend on d , this will ultimately allow us to use the *same* global storage of m field elements to implement every gate function with a low-degree polynomial, even though d may vary wildly from one gate to the next.

Following the notation of Cook and Mertz [CM25], a *program* on $\mathcal{K}$ -registers $R_0, \dots, R_m$ is a finite sequence of *instructions*, where each instruction has one of the two forms:

- either $R_i \leftarrow aR_i$ with $a \in \mathcal{K}^\times$, or
- $R_i \leftarrow R_i + g(R_0, \dots, R_{i-1}, R_{i+1}, \dots, R_m)$ for a function $g : \mathcal{K}^m \rightarrow \mathcal{K}$.

Let $|P|$ be the number of instructions in P . Each instruction is a bijection on $\mathcal{K}^{m+1}$, so every program P has an *inverse* program P^{-1} (which runs the inverse instructions in reverse order) with $|P^{-1}| = |P|$. We say that P *cleanly adds* $y \in \mathcal{K}^{m+1}$ if, for every initial content $r \in \mathcal{K}^{m+1}$ of the registers, P ends with content $r + y$. Say that P cleanly adds $y_0 \in \mathcal{K}$ into register R_t if it cleanly adds the vector with y_0 in the coordinate of R_t and 0 elsewhere.

The following lemma refines a result of [CM25] by tracking the dependence on the degree $\deg(f)$ explicitly.

Lemma 2.5. Assume that $f : \mathcal{K}^m \rightarrow \mathcal{K}$ is a polynomial with $\deg(f) < d$, and let P_x be a program on $\mathcal{K}$ -registers $(R_0, R_1, \dots, R_m)$ that cleanly adds $(0, x_1, \dots, x_m) \in \mathcal{K}^{m+1}$ for some $x = (x_1, \dots, x_m) \in \mathcal{K}^m$. Then whenever $d < |\mathcal{K}|$ there exists a program P_f with $d \cdot (2|P_x| + 2m + 1)$ instructions that cleanly adds $f(x)$ into the register R_0 .

Proof. Let $a_1, \dots, a_d$ be any pairwise distinct elements of $\mathcal{K}^\times$, and let c_j be defined as in Corollary 2.3. Consider the following procedure:

```

1: for  $j = 1, \dots, d$  do
2:   for  $i = 1, \dots, m$  do
3:      $R_i \leftarrow a_j R_i$ 
4:   end for
5:    $P_x$ 
6:    $R_0 \leftarrow R_0 + c_j \cdot f(R_1, \dots, R_m)$ 
7:    $P_x^{-1}$ 
8:   for  $i = 1, \dots, m$  do
9:      $R_i \leftarrow a_j^{-1} R_i$ 
10:  end for
11: end for

```

Let r_i be the initial value of R_i . In iteration j , after the call to P_x , we have $R_i = a_j r_i + x_i$ for all $i \in \{1, \dots, m\}$, since P_x cleanly adds $(0, x)$. Therefore Line 6 adds $c_j \cdot f(a_j r_1 + x_1, \dots, a_j r_m + x_m)$ to R_0 . Calling P_x^{-1} cleanly adds $(0, -x)$, and the scaling by a_j^{-1} restores $R_i = r_i$ for all $i \in \{1, \dots, m\}$. Therefore R_i remains the same after each iteration of the main loop for all $i > 0$, for every initial content of the registers.

Let $\tilde{f}(t) = f(tr_1 + x_1, \dots, tr_m + x_m)$ be a single-variable polynomial; we know that $\deg(\tilde{f}) \leq \deg(f) < d$. Hence,

$$\sum_{j=1}^d c_j \cdot \tilde{f}(a_j) = \tilde{f}(0) = f(x),$$

which is exactly the value added to R_0 . The total instruction count is d iterations of $2|P_x| + 2m + 1$ instructions each. $\square$

Combining the polynomial representation of gate functions (Proposition 2.4) with the catalytic evaluation subroutine (Lemma 2.5), we obtain the following generalization of the main result of Cook and Mertz [CM25]. Whereas their space bound on TREE EVALUATION is $O(h \cdot \log b + b)$, where h is the height of the tree, the normalized depth $\tilde{h}(\mathcal{C})$ provides a finer measure by charging $\lceil \log(d_i + 1) \rceil$ per gate rather than a uniform $\lceil \log(b + 1) \rceil$.

Reminder of Theorem 2.1. Every gate of a b -bit n -gate circuit $\mathcal{C}$ can be evaluated in $O(\tilde{h}(\mathcal{C}) + b + \log n)$ space.

Proof. Assume b is sufficiently large (greater than 2), and let m be the smallest integer such that $m \log(m) \geq b$. We define $k = \lceil \log(m) \rceil$ (so that $km \geq b$ and $km = O(b)$), along with $\mathcal{F} = \mathbb{F}_{2^k}$, and select z and V_n as in Definition 2.2. We encode values in $\mathcal{F}$ as k -bit strings using their decomposition in the basis $\{z^i\}_i$, and fix a canonical inclusion of vector spaces $\mathcal{F} \subset \mathcal{F}^2 \subset \mathcal{F}^3 \subset \dots$ along the first coordinates. Furthermore, set $\mathcal{K} = \mathbb{F}_{2^{4k}}$. Since $k \mid 4k$, we have $\mathcal{F} \subseteq \mathcal{K}$, and $|\mathcal{K}| = 2^{4k} \geq m^4$.

Let v be one of $\mathcal{C}$'s gates, and let $\text{val}(v) \in \{0, 1\} \subseteq \mathcal{K}$ be the value at node v . We prove the following statement by strong induction on h .

Claim: Let v be one of $\mathcal{C}$'s gates with normalized depth $\bar{h}_{\mathcal{C}}(v) = h$. Then for every $t \in \{0, \dots, m\}$ and $c \in \mathcal{K}$, there is a program $P_{v,t,c}$ on $\mathcal{K}$ -registers $(R_0, R_1, \dots, R_m)$ that cleanly adds $c \cdot \text{val}(v)$ into R_t , with $|P_{v,t,c}| \leq C \cdot 64^h$ where $C = (2b)^3$.

Once we have proved the claim, we are done: starting from all registers set to zero, the effect of running $P_{v,0,1}$ is that R_0 is set to $\text{val}(v)$. We compute $P_{v,0,1}$ recursively: at a gate u of indegree d , run the loop of Lemma 2.5 with $2d^2$ iterations, and execute P_x (and P_x^{-1}) via recursive calls on each of u 's predecessors. A stack stores the loop index, the index of the current predecessor, and one bit recording whether the current call belongs to P_x or to P_x^{-1} , which costs $O(\lceil \log(d+1) \rceil)$ bits of space. Overall the stack requires $O(\bar{h}_{\mathcal{C}}(v))$ bits of space. The current gate's index is recomputed from the stack in $O(b + \log n)$ space, the registers take $O(mk) \leq O(b)$ bits, and each instruction is computed in $O(m \log |\mathcal{K}| + b + \log n)$ space by the formula of Proposition 2.2. Thus the total space is $O(\bar{h}_{\mathcal{C}}(v) + b + \log n)$.

Now we turn to proving the claim.

Base case. If $h = 0$, then v is a source, so its value $\text{val}(v)$ is the bit assigned to v by φ_v . Let $P_{v,t,c}$ be the single (clean) instruction $R_t \leftarrow R_t + c \cdot \text{val}(v)$. Note that $|P_{v,t,c}| = 1 \leq C$.

Induction step. Assume that the statement holds for all $h' < h$. Fix t and c , and let $R'_1, \dots, R'_m$ be the ordered list of registers other than R_t . Let $d = \text{in-deg}(v) \geq 1$ and let $u_1, \dots, u_d$ be the predecessors of v . By definition, $\bar{h}(u_j) \leq h - \log(d+1)$.

Let $x_j \in \{0, 1\}$ be the value at u_j . By the induction hypothesis, for every register R_s and every $c' \in \mathcal{K}$, there is a program $P_{u_j,s,c'}$ that cleanly adds $c' \cdot x_j$ into R_s , where

$$|P_{u_j,s,c'}| \leq C \cdot 64^{h - \log(d+1)}.$$

We identify $\{0, 1\}^d$, i.e., the domain of φ_v , with $V_d \subseteq \mathcal{F}^m$. In other words, we spread d bits among the m registers $R'_1, \dots, R'_m$ almost evenly, and designate these d bits as the input for φ_v . More precisely, for $d = \ell m + r$, we select the lower ℓ bits in $R'_1, \dots, R'_{m-r}$ and the lower $\ell + 1$ bits in $R'_{m-r+1}, \dots, R'_m$ (i.e., $\ell m + r = d$ bits in total), where the lower i bits of a register are the coefficients of $1, z, \dots, z^{i-1}$ in our encoding of $\mathcal{F}$. Let the j -th of these d bits be the coefficient of z^{p_j} in R_{s_j} (one of $R'_1, \dots, R'_m$), and let P_x be the concatenation of the programs $P_{u_1,s_1,z^{p_1}}, \dots, P_{u_d,s_d,z^{p_d}}$ given by the induction hypothesis. Each of them adds $z^{p_j} x_j$ to R_{s_j} and changes no other register, so P_x cleanly adds to $(R'_1, \dots, R'_m)$ the element of V_d that encodes $x = (x_1, \dots, x_d)$ and leaves R_t unchanged. In particular,

$$|P_x| = \sum_{j=1}^d |P_{u_j,s_j,z^{p_j}}| \leq d \cdot C \cdot 64^{h - \log(d+1)} = \frac{d}{(d+1)^6} \cdot C \cdot 64^h,$$

where we used $\log_2(64) = 6$.

Let $f \in \mathcal{F}[t_1, \dots, t_m]$ be the polynomial of Proposition 2.2 that agrees with φ_v on $V_d \subseteq \mathcal{F}^m$. By definition, $m \log(m) \geq b \geq d$, so Proposition 2.4 guarantees $\deg(f) < 2d^2$. Since for all but finitely many b , we have $m^4 > 2b^2$, we have for all large enough b that

$$|\mathcal{K}| = 2^{4k} \geq m^4 > 2b^2 \geq 2d^2.$$

Hence we can apply Lemma 2.5 to the polynomial $c \cdot f$ (of degree less than $2d^2$), with $2d^2$ as the degree parameter d , R_t in the role of register R_0 , and $R'_1, \dots, R'_m$ in the roles of $R_1, \dots, R_m$. The lemma provides

a program $P_v := P_{v,t,c}$ that cleanly adds $c \cdot f(x) = c \cdot \text{val}(v)$ into R_t and satisfies

$$|P_v| \leq 2d^2(2|P_x| + 2m + 1) \leq \frac{4d^3}{(d+1)^6} \cdot C \cdot 64^h + 2d^2(2m + 1).$$

To verify $|P_v| \leq C \cdot 64^h$, it suffices to check

$$\frac{4d^3}{(d+1)^6} + \frac{2d^2(2m+1)}{C} \leq 1$$

holds for all sufficiently large b . We claim that the first term is at most $1/4$, and the second term is at most $3/4$. For the first term, we have $4d^3 \leq \frac{1}{4} \cdot (d+1)^6 \iff 16d^3 \leq (d+1)^6 \iff 16^{1/3}d \leq (d+1)^2$, which holds for all $d \geq 1$ since $16^{1/3} < 3$. For the second term, we derive

$$\frac{2d^2(2m+1)}{C} \leq \frac{2b^2(2m+1)}{(2b)^3} \leq \frac{2m+1}{4b} \leq \frac{2b+1}{4b} = \frac{1}{2} + \frac{1}{4b} \leq \frac{3}{4},$$

since for all $b \geq 2$ we have $d \leq b$, $C = (2b)^3$, and $m \leq b$. This completes the proof. $\square$

3 Simulating Parallel External Memory Algorithms in Low Space

We begin this section by more formally defining the *free-input* Parallel External Memory Model (FPEM) mentioned in the introduction. It is an extension of the Parallel External Memory Model that allows reading bits of the input for free, as well as arbitrary subsets of bits of external memory. Just like the standard PEM, it consists of P uniform processors indexed from 1 to P , each being a RAM with M bits of memory, and there are S bits $a_1, \dots, a_S$ of shared external memory. Every processor p_i also has $S \cdot M$ *write-only* I/O registers, labeled $R_{j,\ell}^i$ for all $j \in [M]$ and $\ell \in [S]$, each holding a bit.

Computation is done in sequential *rounds*. Initially all memory bits and all a_ℓ 's are set to zero. In each round, each processor p_i has two stages:

- In the *reading* stage of a round, processor p_i runs an arbitrary (halting) computation in its internal memory, with read-only access to the input and its index i , and may fill some of its I/O registers. If $R_{j,\ell}^i$ ends up holding 1, then a_ℓ is used to change the value of the j -th bit of p_i 's memory. After that, all I/O registers are set to 0.
- In the *writing* stage of a round, processor p_i again runs a memory-bounded computation and may fill some I/O registers once again. If $R_{j,\ell}^i = 1$, then the j -th bit of p_i will be used to change the content of a_ℓ . Afterwards, the I/O registers are flushed again to 0.

The input is fully accessible to every processor at all times in a read-only manner, while one bit of the external memory is designated as the return bit.

3.1 Concurrent I/O

Although we assume every processor makes at most one write request to a bit of external memory a_ℓ , multiple processors may make such requests at once (*concurrent writes*). Moreover, a processor may request to read multiple bits from the external memory into a single location in its internal memory (*concurrent*

copying). There are several ways to deal with concurrent writes and copies, and the reduction in Section 3.2 can accommodate fairly complex models.

For every external memory bit a_ℓ , let $\Sigma_\ell = \{0, 1\}$ be the set of symbols that can be written in a_ℓ . Assume that Σ_ℓ has an associative operation $\circ$, and adjoin an identity e to get a monoid $\Sigma_\ell^e = (\{e, 0, 1\}, \circ)$.

Now suppose that p_i requests to write $w_i \in \Sigma_\ell^e$ into a_ℓ (where e corresponds to “no request”). Then the value of a_ℓ is set to be

$$w_1 \circ w_2 \circ \dots \circ w_P, \quad (1)$$

unless all $w_i = e$, in which case a_ℓ is kept the same.

In our case, there are three natural associative non-trivial operations on $\{0, 1\}$ to consider (up to an isomorphism induced by the action of S_2 on $\{0, 1\}$, and up to reversing the order of the operands):

- Right projection: $g_1 \circ g_2 = g_2$; this operation is equivalent to priority CRCW, where only the write request made by the processor with the highest index is recorded.
- Addition in $\mathbb{F}_2$: $g_1 \circ g_2 = g_1 \oplus g_2$.
- Multiplication in $\mathbb{F}_2$: $g_1 \circ g_2 = g_1 g_2$; this operation is also dual to logical OR.

We can take a similar approach to handle concurrent copies. We choose an associative operation $\circ$ on $\Gamma_j = \{0, 1\}$ for every $j \leq M$ (so that the operation can be computed in low space). Assume that p_i requested to read cells $a_{\ell_1}, \dots, a_{\ell_m}$ into its j -th bit, where $\ell_1 < \dots < \ell_m$. If $m = 0$, then the content of the bit stays the same; otherwise, it is set to

$$a_{\ell_1} \circ \dots \circ a_{\ell_m}. \quad (2)$$

For example, if we set one of the internal memory bits to support operation $\oplus$, then p_i can compute the binary sum of a very long block of the external memory in a single read request, even though the length of the block may exceed the entire internal memory.

In both cases, we require that for each index, the associative operation on Σ_ℓ or Γ_j is computable in logarithmic space.

Though not considered here, one can exploit concurrent writes/copies by expanding Σ_ℓ , i.e., replacing individual bits with blocks. If the operation is bit-wise, it brings no additional power, but operations like $(\mathbb{Z}_q, +)$ or $(\mathbb{Z}, \max)$ can potentially save a log factor in the number of rounds.

3.2 The Simulation

We are now ready to state our simulation result for this parallel external memory model. We prove the decision problem version of Theorem 1.1 from the introduction; the full version immediately follows.

Reminder of Theorem 1.1. *Let T, S, P, M be good. Suppose a language L can be decided in the FPME model with $P(n)$ processors each with $M(n) \geq \log n$ internal memory, and an external memory with $S(n)$ bits, in $T(n)$ rounds. Then*

$$L \in \text{SPACE} [T \cdot \log(P \cdot S \cdot M) + M].$$

Let $\mathcal{M}$ be an instance of the FPME model that decides L . As described in the introduction, we will give a space-efficient reduction from the simulation problem for $\mathcal{M}$ on an input x to an instance of b -BIT CIRCUIT VALUE (Section 2). Our reduction works by composing two such circuits. Both will take as input the current contents of the external memory $\mathbf{a} = (a_1, \dots, a_S)$ and the internal memories of the processors $(x_1^i, \dots, x_M^i)$ for all $i \in [P]$. Both circuits compute some updated values as outputs:

- The “reading circuit” $\mathcal{C}_1$ simulates the reading stage of a given round. This circuit will have normalized depth $\bar{h}(\mathcal{C}_1) \leq O(\log(S \cdot M))$.
- The “writing circuit” $\mathcal{C}_0$ simulates the writing stage of a given round, and has normalized depth $\bar{h}(\mathcal{C}_0) \leq O(\log(P \cdot M))$.

Now define a circuit

$$\mathcal{C} = (\mathcal{C}_0 \circ \mathcal{C}_1)^T,$$

obtained by composing T copies of $\mathcal{C}_1$ and $\mathcal{C}_0$ sequentially. The resulting M -bit circuit $\mathcal{C}$ decides L and has total normalized depth $O(T \cdot \log(PSM))$. Furthermore, $\mathcal{C}$ satisfies the uniformity requirements of Section 2, since $\mathcal{C}_0$ and $\mathcal{C}_1$ will satisfy them. In particular, their wiring is logspace-computable, and each gate function is computable in $O(M)$ space.

Applying the generalized Cook-Mertz algorithm for b -BIT CIRCUIT VALUE (Theorem 2.1) to $\mathcal{C}$, we can evaluate $\mathcal{C}$ in $O(T \cdot \log(P \cdot S \cdot M) + M)$ space. (Note that since the circuit $\mathcal{C}$ has $n = O(T \cdot P \cdot S \cdot M)$ gates, the $\log n$ term of Theorem 2.1 is $O(\log T + \log(PSM)) = O(T \log(PSM))$ and is absorbed in the big-O.)

The following binary tree construction will be useful in our b -BIT CIRCUIT VALUE instance:

Proposition 3.1. *Let $\circ$ be an associative binary operation on a set G . Assume that $|G| \leq 2^b$ and $\circ$ is computable in space b . Then there is a $2b$ -bit circuit $\text{Reduce}_n(\circ)$ with $\bar{h}(\text{Reduce}_n(\circ)) \leq \lceil \log(n) \rceil \lceil \log(2b+1) \rceil$ that takes values $x_0, \dots, x_{n-1} \in G$ and outputs $x_0 \circ \dots \circ x_{n-1}$.*

Proof. Simply take a binary tree of height $\lceil \log(n) \rceil$ whose internal nodes compute $\circ$ and whose leaves are $x_0, \dots, x_{n-1}$. $\square$

Note that we will use the proposition with $G = \{e, 0, 1\}$ and $b = 2$, so $\bar{h}(\text{Reduce}_n(\circ)) \leq 3 \lceil \log n \rceil$.

3.3 Reading bits from external memory: the circuit $\mathcal{C}_1$

We give a schematic of the circuit, then describe the functionality of each node in it.

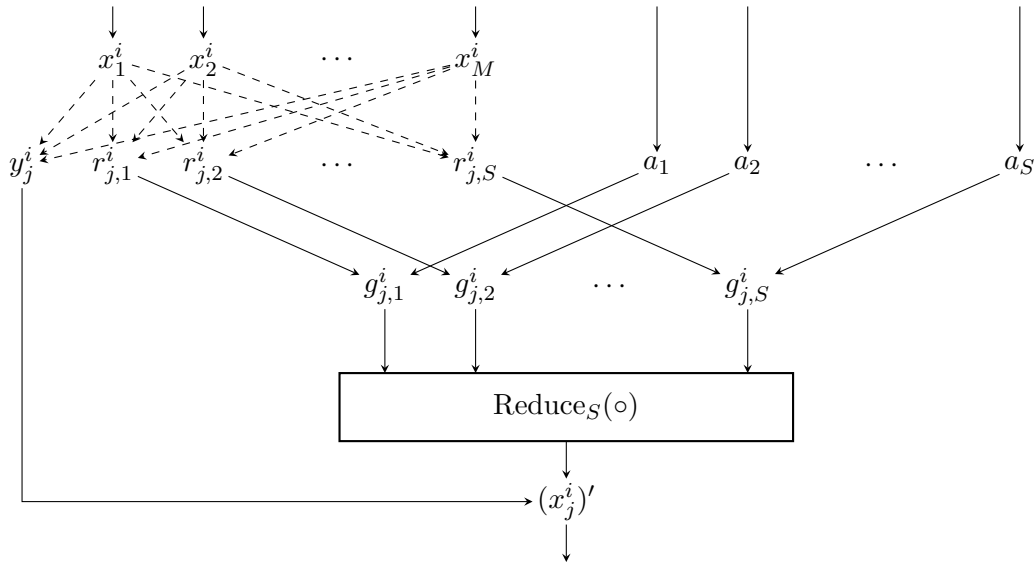


Figure 1: Circuit $\mathcal{C}_1$

The read queries from different processors are all independent, so for all $j \in [M]$ and $i \in [P]$, we construct a circuit reading the j -th bit requested by processor p_i , and mount these $P \cdot M$ copies in parallel. In the figure above, x_j^i is the j -th bit of processor p_i 's internal memory at the beginning of the reading stage, y_j^i is the value of the same bit after all reading requests are set, and $r_{j,\ell}^i \in \{0, 1\}$ is the content of the write-only register $R_{j,\ell}^i$ owned by p_i . The latter two are computed by simulating processor p_i .

For all $\ell \leq S$, each node $g_{j,\ell}^i$ outputs a pair of bits implemented by the control gate:

$$g_{j,\ell}^i(r_{j,\ell}^i, a_\ell) = \begin{cases} a_\ell, & r_{j,\ell}^i = 1, \\ e, & r_{j,\ell}^i = 0, \end{cases}$$

where we recall that e is the identity element. The node $(x_j^i)'$ computes the program output, also implemented by the control gate:

$$(x_j^i)'(y_j^i, g) = \begin{cases} g, & g \neq e, \\ y_j^i, & g = e. \end{cases}$$

The values among $g_{j,1}^i, \dots, g_{j,S}^i$ that are distinct from e are precisely $a_{\ell_1}, \dots, a_{\ell_m}$, as defined in (2), in the correct order. Since the identity e does not affect the product, the $\text{Reduce}_S(\circ)$ block computes $a_{\ell_1} \circ \dots \circ a_{\ell_m}$, and the value at $(x_j^i)'$ will coincide with the j -th bit requested by processor p_i . Applying Proposition 3.1, we observe that the normalized depth of $\mathcal{C}_1$ is

$$\hbar(\mathcal{C}_1) \leq \lceil \log(M+1) \rceil + \lceil \log(3) \rceil + \hbar(\text{Reduce}_S(\circ)) + \log(4) \leq O(\log(SM)),$$

where the $\log(M+1)$ term accounts for simulating p_i (a RAM with M bits of memory), and $\log(3)$ and $\log(4)$ correspond to $\text{in-deg}(g_{j,\ell}^i) = 2$ and $\text{in-deg}((x_j^i)') = 3$.

We emphasize that the $\text{Reduce}_S(\circ)$ circuit can use different operations $\circ$ for each x_j^i as long as the operation itself can be selected uniformly.

3.4 Writing bits to external memory: the circuit $\mathcal{C}_0$

As before, we give a schematic of the circuit, and describe the functionality of each node.

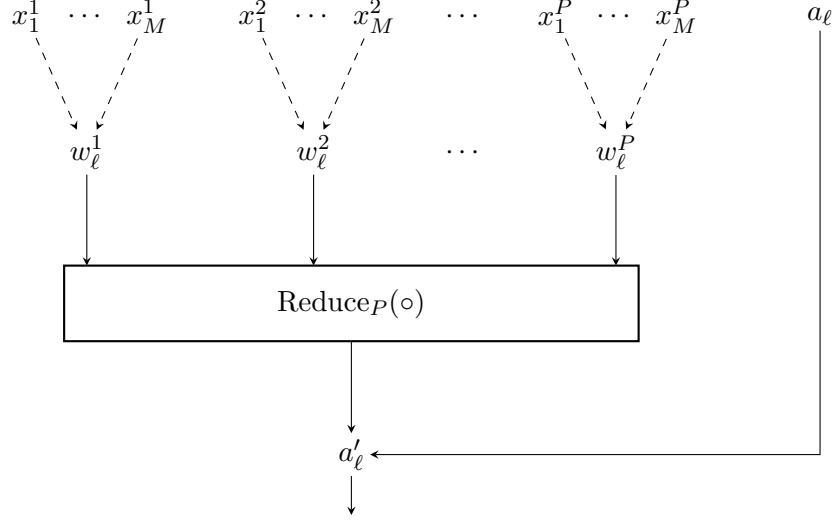


Figure 2: Circuit $\mathcal{C}_0$

Recall a_ℓ is the ℓ -th bit of the external memory. Similarly to $\mathcal{C}_1$, for each $\ell = 1, \dots, S$, we construct a circuit computing the bit a_ℓ and mount these S copies in parallel. Here $w_\ell^i \in \{0, 1, e\}$ is the write request sent by p_i to a_ℓ , which we can compute by simulating processor p_i using its entire internal memory state $(x_1^i, \dots, x_M^i)$: the value of w_ℓ^i is 0 or 1 if p_i wants to write that bit value into a_ℓ , and $w_\ell^i = e$ signifies that no such request has been made.

The final output a'_ℓ is a control gate:

$$a'_\ell(w, a_\ell) = \begin{cases} w, & w \neq e, \\ a_\ell, & w = e. \end{cases}$$

It is easy to see that $\text{Reduce}_P(\circ)$ will compute $w_\ell^1 \circ \dots \circ w_\ell^P$, precisely as defined in (1), so a'_ℓ will indeed output the updated value of a_ℓ . Furthermore, for all $i \in [P]$ and $j \in [M]$, $\mathcal{C}_0$ contains a gate z_j^i (of indegree M) with predecessors $x_1^i, \dots, x_M^i$ that outputs the j -th memory bit of p_i after its computation in the writing stage. Together with the bits a'_ℓ , these are the inputs to the next copy of $\mathcal{C}_1$.

Applying Proposition 3.1, the normalized depth of this circuit is

$$\bar{h}(\mathcal{C}_0) \leq \lceil \log(M+1) \rceil + \bar{h}(\text{Reduce}_P(\circ)) + \log(4) \leq O(\log(P \cdot M)).$$

Finally, we observe that both $\mathcal{C}_0$ and $\mathcal{C}_1$ satisfy the uniformity conditions of Section 2 with $b = M$: their wiring is logspace-computable, and each gate function can be computed in $O(M)$ space (with read-only access to the input). This completes the proof of Theorem 1.1.

4 Solving Lex First 2SAT

In this section, we describe the proof of Theorem 1.2. Let ϕ be a 2CNF formula with n variables $x_1, \dots, x_n$.

We recall the standard reduction from 2SAT to a directed graph [APT79]. Define $\Gamma(\phi)$ to be the implication graph associated with ϕ : its vertices are literals x_i and $\neg x_i$ for $1 \leq i \leq n$, and a directed edge $s \rightarrow t$ between two literals is present iff ϕ contains a clause $\neg s \vee t$. Moreover, we add an edge $\neg s \rightarrow s$ if

ϕ contains a unit clause (s) . Let $\bar{\Gamma}(\phi)$ be the transitive closure of $\Gamma(\phi)$. From the transitive closure we can immediately determine satisfiability:

Proposition 4.1 ([APT79]). *ϕ is unsatisfiable if and only if there is a variable x and edges $x \rightarrow \neg x$ and $\neg x \rightarrow x$ in $\bar{\Gamma}(\phi)$.*

For every variable x of ϕ , we write x^0 and x^1 for the literals $\neg x$ and x , respectively.

Definition 4.1. *Let ϕ be a 2CNF formula on n variables and a be a partial assignment to its variables (i.e., a partial function $[n] \rightarrow \{0, 1\}$). We say a is consistent with ϕ if a can be extended to a full satisfying assignment (a total function $[n] \rightarrow \{0, 1\}$). For an assignment a consistent with ϕ , we define its closure $\bar{a}$ to be $\bar{a}_i = \sigma \in \{0, 1\}$ if $a_i = \sigma$ or $x_j^{a_j} \rightarrow x_i^\sigma$ is an edge in $\bar{\Gamma}(\phi)$ for some j in the domain of a . (Otherwise, $\bar{a}_i$ is undefined.)*

We say that an assignment a is closed whenever $\bar{a} = a$.

As every edge of $\bar{\Gamma}(\phi)$ is a valid implication, $\bar{a}$ is a well-defined partial assignment consistent with ϕ , and it is also closed by the transitivity of $\bar{\Gamma}(\phi)$.

Lemma 4.2 (Locality of extension). *Let b be an assignment that is closed and consistent with a 2CNF ϕ , and let S be a subset of variables of ϕ . If a is a consistent extension of $b|_S$ to a subset of S , then $a \cup b$ is also consistent.*

Proof. Assume $a \cup b$ is inconsistent. We claim that there are two (not necessarily distinct) variables x_i and x_j with

$$(a \cup b)_i = \sigma, \quad (a \cup b)_j = \tau,$$

such that $x_i^\sigma \rightarrow x_j^{\neg\tau}$ is an edge in $\bar{\Gamma}(\phi)$. Note that this also implies $x_j^\tau \rightarrow x_i^{\neg\sigma}$ is an edge as well, since $s \rightarrow t$ is an edge of $\Gamma(\phi)$ iff $\neg t \rightarrow \neg s$ is an edge.

To prove the claim, let c be the partial assignment $a \cup b$ and suppose no such i, j exist. Note our supposition is equivalent to the condition that there are no literals p, q true under c such that $p \rightarrow \neg q$ is in $\bar{\Gamma}(\phi)$. By definition, a extends $b|_S$ so a and b must agree wherever both are defined. Let L be the set of literals that are either true under c , or are reachable in $\bar{\Gamma}(\phi)$ from a literal that is true under c .

First, we prove that L contains no complementary pair of literals. Suppose both ℓ and $\neg\ell$ are in L . By the definition of L , there is a literal p true under c such that either $p = \ell$ or $p \rightarrow \ell$ is in $\bar{\Gamma}(\phi)$, and a literal q true under c such that either $q = \neg\ell$ or $q \rightarrow \neg\ell$ is in $\bar{\Gamma}(\phi)$. If $p = \ell$ and $q = \neg\ell$, then c makes both ℓ and $\neg\ell$ true, which is a contradiction since c is a partial assignment. In every other case, we observe that $p \rightarrow \neg q$ is in $\bar{\Gamma}(\phi)$, utilizing the fact that $p \rightarrow \neg q \in \bar{\Gamma}(\phi)$ iff $q \rightarrow \neg p \in \bar{\Gamma}(\phi)$. This contradicts our supposition, so L contains no complementary pair.

Next, let e be a satisfying assignment of ϕ (which exists since b is consistent). Let f be the full variable assignment that sets every literal of L to true and agrees with e on all other variables. We claim every clause $u \vee v$ of ϕ is satisfied by f (with $v = u$ for unit clauses). If $u \in L$ or $v \in L$, this follows from the definition of f . Otherwise, if $u \notin L$ and $v \notin L$, then we also have $\neg u \notin L$ and $\neg v \notin L$, since $\neg v \rightarrow u$ and $\neg u \rightarrow v$ are edges in $\Gamma(\phi)$. Therefore f agrees with e on both literals u and v , and e satisfies $u \vee v$. Hence f is a satisfying assignment extending $a \cup b$, contradicting the assumption that $a \cup b$ is inconsistent. This proves the claim.

Finally, if one of a or b contains both variables, then that partial assignment cannot be consistent. Without loss of generality assume that $b_i = \sigma$ and $a_j = \tau$, where $x_j \in S$. Since b is closed, $b_j = \neg\tau$ for $x_i^\sigma \rightarrow x_j^{\neg\tau}$ is present in $\bar{\Gamma}(\phi)$, so a cannot extend $b|_S$. $\square$

Finally, we present the FPEM algorithm for LEX FIRST 2SAT.

Theorem 4.3. *For any parameter $k(n) \leq n$ with $k \in \Omega(\log(n)^2)$, there is an FPEM algorithm with $P \leq O(n^2)$, $M \leq O(k)$, and $S \leq O(n)$ that solves LEX FIRST 2SAT on n variables and $O(n^2)$ clauses in $T \leq O(n/k)$ I/O rounds.*

Applying Theorem 1.1 to the above, the following is immediate:

Reminder of Theorem 1.2. LEX FIRST 2SAT on 2CNFs with n variables and $O(n^2)$ clauses can be solved in $O(\sqrt{n \log n})$ space.

Proof. Set $k = \sqrt{n \log(n)}$ in Theorem 4.3. Then $T \leq O(\sqrt{n/\log n})$, $P \leq O(n^2)$, $S \leq O(n)$ and $M \leq O(\sqrt{n \log n})$. Applying Theorem 1.1, the theorem follows. $\square$

For the rest of this section, we will prove Theorem 4.3. Throughout the proof, we treat the input formula ϕ as freely accessible to all processors. The external memory will hold the current partial variable assignment $b = (b_1, \dots, b_n) \in \{0, 1, \perp\}^n$. The symbol $\perp$ means “no assignment yet”. Each b_i is stored using two bits, an “assigned” bit and a value bit, so that the all-zero initial external memory encodes $b = (\perp, \dots, \perp)$. Concurrent writes to a bit of external memory are resolved by priority (right projection, as in Section 3.1).

Let A be the adjacency matrix of $\bar{\Gamma}(\phi)$ (the transitive closure graph of ϕ). Since $k(n) \in \Omega(\log(n)^2)$, setting $M = \Theta(k(n))$ we can compute any desired entry of A in $O(M)$ bits of internal memory with *zero* I/Os, using the fact that every entry of the transitive closure can be computed in $O(\log^2 n)$ space and we can access the formula ϕ for free. Thus we may assume free random access to A .

We will define three subroutines:

INITASSIGNMENT computes the partial assignment defined on all variables x_i such that a unit clause (x_i) or $(\neg x_i)$ is present in ϕ :

procedure INITASSIGNMENT

Processor $Q_{i,i}$, **for** $i \in [n]$:

 If ϕ contains a unit clause (x_i^τ) , write τ into b_i

end procedure

CLOSURE uses $O(n^2)$ processors and computes the closure of the current partial assignment:

procedure CLOSURE

Processor $Q_{i,j}$, **for** $i, j \in [n]$:

 Read b_i, b_j and compute four entries of A corresponding to edges between $x_i^{\{0,1\}}$ and $x_j^{\{0,1\}}$

 If $b_j = \perp, b_i = \sigma \in \{0, 1\}$ and there is an edge $x_i^\sigma \rightarrow x_j^\tau$ in A , write τ into b_j

end procedure

Here all assigned variables are indeed contained in the closure, and the resulting assignment is closed. Suppose b is consistent at the start of the call (this holds at every call, see below). Then whenever $b_i = \sigma$ and $x_i^\sigma \rightarrow x_j^\tau$ is an edge in A , every satisfying extension of b sets $x_j = \tau$, so $b_j \in \{\tau, \perp\}$ and all processors writing into b_j write the same value τ . Indeed, assume that $Q_{i,j}$ set $b_j \leftarrow \tau$ since there is an edge $x_i^\sigma \rightarrow x_j^\tau$ in A and $b_i = \sigma$, but at the same time there is also an edge $x_j^\tau \rightarrow x_\ell^\rho$ while $b_\ell \neq \rho$. Given that A is transitively

closed and contains both $x_i^\sigma \rightarrow x_j^\tau$ and $x_j^\tau \rightarrow x_\ell^\rho$, it must also contain an edge $x_i^\sigma \rightarrow x_\ell^\rho$, so $b_\ell = \perp$ by the above, and $Q_{i,\ell}$ must have set $b_\ell \leftarrow \rho$. Contradiction.

LOCALLEX2SAT is a single-processor subroutine that takes in a range of indices $I = [i, j)$ and a partial assignment c to the $|I|$ variables $x_I = (x_i, \dots, x_{j-1})$. Given that c is the restriction of a closed assignment to x_I , this subroutine outputs a partial assignment a such that

- a extends c to all variables in x_I ,
- a can be extended to a satisfying assignment for ϕ , and
- a is lexicographically minimal among such assignments.

LOCALLEX2SAT calls an $O(\log^2 n)$ space algorithm for 2SAT on ϕ and a as a subroutine, to test if the current partial assignment a can be extended to a full satisfying assignment, while assigning variables greedily.

```

procedure LOCALLEX2SAT( $c, I$ )
   $a \leftarrow c$ 
  for each  $i \in I$  in increasing order do
    if  $c_i = \perp$  then
      if 2SAT( $\phi, a \sqcup \{x_i \leftarrow 0\}$ ), set  $a_i \leftarrow 0$  else set  $a_i \leftarrow 1$ 
    end if
  end for
  return  $a$ 
end procedure

```

The final algorithm finds the lex-first satisfying assignment in batches of k sequential variables. In the following, let a and c be $O(M)$ -bit registers of internal memory on an additional processor p_0 (distinct from the processors $Q_{i,j}$).

Algorithm 1 LEX 2SAT SOLVER

```

if  $\neg 2\text{SAT}(\phi)$  then reject.
INITASSIGNMENT
for  $t = 0$  to  $\lceil n/k \rceil - 1$  do
  CLOSURE
  Processor  $p_0$ :
    Read the block  $b|_I$  from external memory into  $c$ , where  $I = [tk + 1, \min((t + 1)k, n) + 1)$ .
     $a \leftarrow \text{LOCALLEX2SAT}(c, I)$ .
    Write all new values assigned in  $a$  (which were unassigned in  $c$ ) back to  $b$ .
end for

```

Correctness. First of all, note that the initial call to 2SAT in LEX 2SAT SOLVER can be simulated in $O(\log^2 n)$ space, since 2SAT is in $\text{NL} \subseteq \text{SPACE}[\log^2 n]$. Setting $M = \Theta(k) \geq \Omega(\log^2 n)$, we can determine whether ϕ is satisfiable entirely in internal memory with *zero* I/Os.

We claim that at the end of LEX 2SAT SOLVER, the vector b in external memory holds the lex-first satisfying assignment if ϕ is satisfiable.

It is clear from the construction that we assign every variable exactly once. Hence, to prove correctness by contradiction, we can consider the lex-first satisfying assignment b' and the first I/O round in which some b_i was assigned a value τ with $b'_i = \neg\tau$, and (within that round) the smallest such index i .

Before this “bad” round, the assignment in external memory is contained in b' . Thus this is not a round of INITASSIGNMENT, which only writes values forced by unit clauses, all satisfied by b' . This bad round also cannot be a round of CLOSURE, which only writes τ into b_j when $x_\ell^\sigma \rightarrow x_j^\tau$ is an edge of $\bar{\Gamma}(\phi)$ for some ℓ with $b_\ell = \sigma = b'_\ell$, and then $b'_j = \tau$ as well. Hence, we conclude that b_i was determined inside the call LOCALLEX2SAT(c, I). Let d be the state of the assignment in the external memory right before LOCALLEX2SAT(c, I) was called. We note $d \subseteq b'$ by the choice of the round (so d is consistent), and $c = d|_I$ by definition.

Let a be the value returned by that call. Observe that d is closed, since it is the result of a CLOSURE call on a consistent assignment. Furthermore, a is consistent by construction, so by Lemma 4.2, $a \cup d$ is also consistent. Moreover, every satisfying extension of $a \cup d$ agrees with b' on $x_1, \dots, x_{i-1}$: the variables of earlier blocks are assigned in $d \subseteq b'$, and for $j \in I$ with $j < i$ we have $a_j = b'_j$ by the minimality of i . Hence, $a_i = 1$ and $b'_i = 0$: otherwise, any extension of $a \cup d$ to a full satisfying assignment would precede b' in the lexicographical order.

Finally, let a' be the state of a right before we assigned $a_i \leftarrow 1$. Then $a' \subseteq b'$ by the above, as a' consists of c and the values a_j with $j \in I$ and $j < i$. Since LOCALLEX2SAT set $a_i \leftarrow 1$, the assignment $a' \sqcup \{x_i \leftarrow 0\}$ cannot be consistent, and neither can its extension b' . Contradiction.

The I/O Bound. LOCALLEX2SAT makes no calls to external memory, since we have free access to the input ϕ . It uses only $O(k + \log^2 n)$ internal memory: $O(k)$ to store the partial assignment a , and $O(\log^2 n)$ to run a 2SAT algorithm on the input ϕ and a . Both CLOSURE and INITASSIGNMENT take one I/O round each, and use $O(n^2)$ processors.

Finally, LEX 2SAT SOLVER takes $O(n/k)$ I/O rounds: one round for INITASSIGNMENT, and in each of the $\lceil n/k \rceil$ iterations, we make one round for CLOSURE and one round for p_0 , whose reading stage loads $b|_I$ and whose writing stage writes the new values. This completes the proof.

5 Space-Efficient Circuit Evaluation Via External Memory Algorithms

Recall in the *Circuit Value Problem* (CVP) we are given the encoding of a Boolean circuit $\mathcal{C}$ with fan-in at most d and values to its inputs $x_1, \dots, x_m$, and are asked to compute the value at a designated output of $\mathcal{C}$. Williams [Wil26] showed that for circuits of n gates and constant fan-in, CVP can be solved in $\sqrt{n} \cdot \text{poly}(\log n)$ space, and Shalunov [Sha26] showed the space bound can be improved to $O(\sqrt{n \log n})$.

First, we show how to recover Shalunov’s result with a very simple algorithm in our parallel external memory model, and then modify the approach to prove Theorem 1.3.

Proposition 5.1. *For any positive $\log^2 n \leq k(n) \leq n$, there exists an FPEM algorithm running on a single processor with $M = \Theta(dk)$ internal memory and $S = \Theta(n)$ external memory that solves CVP on size- n circuits with fan-in at most d in $T = \lceil n/k \rceil$ rounds.*

In particular, for $d \leq n/\log^3 n$, applying the space-efficient simulation of Theorem 1.1 with $k = \sqrt{n \log(n)/d}$ shows that CVP with gates of fan-in d can be solved in space $O(\sqrt{dn \log(n)})$, recovering the theorem of [Sha26] for $d = O(1)$.

Proof. We may assume that the gates of $\mathcal{C}$ are given in topological order. Otherwise, such an ordering can be computed for free in our model by counting the number of descendants of a given node, applying

$NL \subseteq \text{SPACE}[\log^2 n]$ [AB09]. (Note we have set $k(n) \geq \log^2 n$ to accommodate the computation of a topological ordering in internal memory for free.)

Let $v_1, \dots, v_n$ be the list of gates in topological order. Let $p(v_i)$ denote the set of parents of gate v_i , so that $|p(v_i)| \leq d$ for all i . We evaluate the gates in batches of k at a time, designating external memory bits $a_1, \dots, a_n$ to store the values of gates $v_1, \dots, v_n$.

Consider the following procedure:

Algorithm 2 CVP

```

1: for  $t = 0, \dots, \lceil n/k \rceil - 1$  do
2:    $u_j \leftarrow v_{j+tk}$  for all  $1 \leq j \leq \min(k, n - tk)$  ▷ assign all gates in the current batch
3:   Using zero-cost access to  $\mathcal{C}$ , compute  $Q \leftarrow \bigcup_j p(u_j)$  ▷ compute the set of all parents of gates
4:   read all  $a_i$  with  $v_i \in Q \setminus \{u_j\}_j$  from external memory ▷  $O(dk)$  bits being read, note  $M = \Theta(dk)$ 
5:   Evaluate all gates  $\{u_j\}_{j \leq \min(k, n - tk)}$  in increasing order of  $j$ 
6:   write the computed values back to external memory
7: end for

```

Since each gate has fan-in at most d , the set of parents satisfies $|Q| \leq dk$, so $O(dk)$ bits of internal memory suffice to hold the parent values for the current batch along with the k newly computed outputs. Each iteration performs a single read/write pair, giving $\lceil n/k \rceil$ rounds in total. Correctness follows by induction on t . $\square$

Now we turn to simulating *unbounded* fan-in circuits over the $\{\text{NOT}, \text{AND}, \text{OR}, \text{XOR}\}$ basis. We can give an FPEM algorithm with low internal memory for this task, by exploiting the associativity of the gate operations.

Theorem 5.2. *For any $\sqrt{n} \leq k(n) \leq n$, there exists an FPEM algorithm with $P = O(n)$ processors, $M = O(k)$ internal memory and $S = O(n)$ external memory that solves CVP on size- n circuits of arbitrary fan-in with gates from $\{\text{NOT}, \text{AND}, \text{OR}, \text{XOR}\}$ in $T = 3\lceil n/k \rceil$ rounds, where every processor reads at most k bits of external memory per round.*

The key improvement over Proposition 5.1 is that the internal memory is $O(k)$ rather than $O(dk)$: the values of the (up to n) parents of a gate are combined by a two-level tree of processors, each reading at most k bits per round, so that no processor ever stores them all. In particular, applying Theorem 1.1 with $k = \sqrt{n \log n}$ shows that CVP with $\{\text{NOT}, \text{AND}, \text{OR}, \text{XOR}\}$ gates can be solved in space $O(\sqrt{n \log n})$, independently of the fan-in d . This proves Theorem 1.3 from the introduction.

Proof. As in the proof of Proposition 5.1, we may assume without loss of generality that the gates $v_1, \dots, v_n$ are given in topological order. For every non-input gate v , we assign an associative operation $\circ_v$ on $\{0, 1\}$ with identity e_v :

- $(\oplus, 0)$ if v is an XOR or NOT gate,
- $(\wedge, 1)$ if v is an AND gate, and
- $(\vee, 0)$ if v is an OR gate.

The value of v is defined to be the $\circ_v$ -product of the values of its parents, plus 1 if v is a NOT gate.

The external memory holds bits $a_1, \dots, a_n$ for the gate values, bits $c_{j,r}$ for $j \in [k]$ and $r \in [\lceil n/k \rceil]$, and bits $s_1, \dots, s_k$. We evaluate the gates in batches $I_t = \{tk+1, \dots, \min((t+1)k, n)\}$ for $t = 0, \dots, \lceil n/k \rceil - 1$, spending three rounds on each batch. For $i \in I_t$ let $j = i - tk$, and list the parents of v_i outside v_{I_t} in increasing order of index, split into blocks of k consecutive parents. Note there are at most $\lceil n/k \rceil \leq k$ blocks.

1. Processor $h_{j,r}$ reads the values a_ℓ of the r -th block of v_i , computes their $\circ_{v_i}$ -product (e_{v_i} if the block is empty), and writes it into $c_{j,r}$.
2. Processor g_j reads $c_{j,1}, \dots, c_{j,\lceil n/k \rceil}$, computes their $\circ_{v_i}$ -product y_j , and writes it into s_j .
3. Processor g_0 reads $s_1, \dots, s_k$, evaluates the gates of I_t in increasing order, using y_j in place of the parents of v_i outside v_{I_t} (or the input bit, if v_i is an input gate), and writes the values into a_i for $i \in I_t$.

By induction on t , the cells of all earlier batches hold the correct values when batch t starts. Since we evaluate in topological order, every parent of v_i outside v_{I_t} lies in an earlier batch, so y_j is the $\circ_{v_i}$ -product over these parents, and g_0 combines it with the values of the parents inside v_{I_t} , which it has already computed in this round.

Each processor reads at most k bits per round and stores them, plus $O(\log^2 n) \leq O(k)$ bits to compute the topological order and to enumerate the parents of a gate (as in Proposition 5.1). Therefore $M = O(k)$. The number of processors is $P \leq k\lceil n/k \rceil + k + 1 = O(n)$, the external memory is $S \leq n + k\lceil n/k \rceil + k = O(n)$, and the number of rounds is $T = 3\lceil n/k \rceil$. $\square$

6 Simulating Cellular Automata in Low Space

Finally, we show how to simulate t steps of a d -dimensional cellular automaton in only $\tilde{O}(t^{1-1/(d+1)})$ space. Recall a d -dimensional cellular automaton (CA) is given by:

- a positive integer radius r ,
- a finite (constant) set of states Σ ,
- an initial configuration $\sigma_0: \mathbb{Z}^d \rightarrow \Sigma$, and
- an evolution rule $\varphi: \Sigma^{(2r+1)^d} \rightarrow \Sigma$.

The r -neighborhood of a cell c is the collection of cells within ℓ^∞ distance r of c . At each time step, every cell $(x_1, \dots, x_d) \in \mathbb{Z}^d$ updates its state based on the values in its r -neighborhood:

$$\sigma_{t+1}(x_1, \dots, x_d) = \varphi\left(\left[\sigma_t(x_1 + \delta_1, \dots, x_d + \delta_d)\right]_{|\delta_i| \leq r}\right).$$

The CA^d problem is: given (for fixed d, r, Σ, φ) a non-negative integer t and the initial states of all $(2rt+1)^d$ cells within ℓ^∞ distance rt of a cell c , compute $\sigma_t(c)$, the value of cell c after t time steps. Given our powerful FPEM model and Theorem 1.1, it is fairly straightforward to give a new space-efficient simulation for this problem.

Reminder of Theorem 1.5. *For every fixed $d \geq 1$, the CA^d problem for t time steps can be solved in space $O((t \log t)^{1-1/(d+1)})$.*

Proof. We prove the theorem by designing an appropriate FPEM algorithm, and appealing to Theorem 1.1.

Fix a parameter $k \in [t]$ to be set later. Given the index of cell c , we will simulate k steps of the d -dimensional automaton per I/O round, starting from the initial configuration σ_0 . Note that our last round will simulate only the remaining $t - k\lfloor t/k \rfloor$ steps left over. In what follows, let k denote the number of steps simulated in the current round, and let s be the number of CA steps simulated before the current round (so that the round computes σ_{s+k} from σ_s).

Since the automaton has radius r , only those cells within the d -dimensional hypercube H_c with side-length $2rt + 1$ centered at c can influence the value of $\sigma_t(c)$. Note that the number of cells in H_c is $O((2rt + 1)^d)$. We assign every cell x in H_c its own processor p_x , and a dedicated collection of bits a_x in external memory to hold the state of x . To compute $\sigma_{s+k}(x)$ from the configuration σ_s , the processor p_x simulates the “light cone” around x for k steps, which requires loading the rk -neighborhood of cell x from σ_s . Therefore each processor p_x can simulate at cell x for k time steps with internal memory $M \leq O((rk)^d) \leq O(k^d)$. In this round, only those cells x within ℓ^∞ distance $r(t - s - k)$ of c need to be updated, and every cell they load is within distance $r(t - s)$ of c . Therefore, by induction over the rounds, the external memory holds the correct states on all cells still needed, and $a_c = \sigma_t(c)$ at the end.

The resulting FPEM algorithm has a number of processors P and external memory S on the order of $O((2rt + 1)^d)$. Since r and d are fixed constants, this bound is $O(t^d)$. The FPEM algorithm runs for $T \leq O(t/k)$ total I/O rounds. Applying Theorem 1.1 to this algorithm, it can be simulated in space

$$O(T \cdot \log(P \cdot S \cdot M) + M) \leq O(t \log(t)/k + (rk)^d).$$

Setting $k = \lceil (t \log t)^{1/(d+1)} \rceil$ to balance the two terms, the space is $O((t \log t)^{1-1/(d+1)})$. □

Acknowledgments

Part of this work was done while R.W. was visiting the Institute for Advanced Study in Princeton, supported by DMS-2424441 (Von Neumann Fellowship).

AI Disclosure: We used Claude and ChatGPT to produce the figures of this paper. We also used Claude and ChatGPT for proofreading. Claude pointed out to us that computing the transitive closure of the 2CNF implication graph can be used to derive an $O(n^\omega)$ -time algorithm for LEX FIRST 2SAT.

References

- [AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009.
- [ADV⁺25] Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 2005–2039. SIAM, 2025. [doi:10.1137/1.9781611978322.63](https://doi.org/10.1137/1.9781611978322.63).
- [AGNS08] Lars Arge, Michael T. Goodrich, Michael J. Nelson, and Nodari Sitchinava. Fundamental parallel algorithms for private-cache chip multiprocessors. In *SPAA 2008: Proceedings of the 20th Annual ACM Symposium on Parallelism in Algorithms and Architectures, Munich, Germany, June 14-16, 2008*, pages 197–206. ACM, 2008. [doi:10.1145/1378533.1378573](https://doi.org/10.1145/1378533.1378573).

- [APT79] Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified Boolean formulas. *Inf. Process. Lett.*, 8(3):121–123, 1979. doi:[10.1016/0020-0190\(79\)90002-4](https://doi.org/10.1016/0020-0190(79)90002-4).
- [AV88] Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, 1988. doi:[10.1145/48529.48535](https://doi.org/10.1145/48529.48535).
- [BFL91] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Comput. Complex.*, 1:3–40, 1991. doi:[10.1007/BF01200056](https://doi.org/10.1007/BF01200056).
- [CA69] Stephen A. Cook and Stål O. Aanderaa. On the minimum computation time of functions. *Transactions of the American Mathematical Society*, 142:291–314, 1969. URL: <http://www.jstor.org/stable/1995359>.
- [CM25] James Cook and Ian Mertz. Tree evaluation is in space $O(\log n \cdot \log \log n)$. *SIAM J. Comput.*, 2025. doi:[10.1137/24M1690126](https://doi.org/10.1137/24M1690126).
- [CMW⁺12] Stephen A. Cook, Pierre McKenzie, Dustin Wehr, Mark Braverman, and Rahul Santhanam. Pebbles and branching programs for tree evaluation. *ACM Trans. Comput. Theory*, 3(2):4:1–4:43, 2012. URL: <https://doi.org/10.1145/2077336.2077337>.
- [Col64] Stephen N. Cole. *Real-Time Computation by Iterative Arrays of Finite-State Machines*. PhD thesis, Harvard University, Cambridge, Massachusetts, 1964. Also published as Report BL-36, Computation Laboratory, Harvard University.
- [Coo66] Stephen A. Cook. *On the Minimum Computation Time of Functions*. PhD thesis, Harvard University, Cambridge, Massachusetts, 1966.
- [Coo85] Stephen A. Cook. A taxonomy of problems with fast parallel algorithms. *Inf. Control.*, 64(1-3):2–21, 1985. doi:[10.1016/S0019-9958\(85\)80041-3](https://doi.org/10.1016/S0019-9958(85)80041-3).
- [DLL⁺18] Erik D. Demaine, Andrea Lincoln, Quanquan C. Liu, Jayson Lynch, and Virginia Vassilevska Williams. Fine-grained I/O complexity via reductions: New lower bounds, faster algorithms, and a time hierarchy. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, Cambridge, MA, USA, January 11-14, 2018*, LIPIcs, pages 34:1–34:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. URL: <https://doi.org/10.4230/LIPICS.ITCS.2018.34>, doi:[10.4230/LIPICS.ITCS.2018.34](https://doi.org/10.4230/LIPICS.ITCS.2018.34).
- [HLMW86] Joseph Y. Halpern, Michael C. Loui, Albert R. Meyer, and Daniel Weise. On time versus space III. *Math. Syst. Theory*, 19(1):13–28, 1986. doi:[10.1007/BF01704903](https://doi.org/10.1007/BF01704903).
- [HPV77] John E. Hopcroft, Wolfgang J. Paul, and Leslie G. Valiant. On time versus space. *J. ACM*, 24(2):332–337, 1977. doi:[10.1145/322003.322015](https://doi.org/10.1145/322003.322015).
- [HU68] John E. Hopcroft and Jeffrey D. Ullman. Relations between time and tape complexities. *J. ACM*, 15(3):414–427, 1968. URL: <https://doi.org/10.1145/321466.321474>.
- [Kar12] Jarkko J. Kari. Basic concepts of cellular automata. In Grzegorz Rozenberg, Thomas Bäck, and Joost N. Kok, editors, *Handbook of Natural Computing*, pages 3–24. Springer, 2012. doi:[10.1007/978-3-540-92910-9_1](https://doi.org/10.1007/978-3-540-92910-9_1).
- [Pat72] Mike Paterson. Tape bounds for time-bounded Turing machines. *J. Comput. Syst. Sci.*, 6(2):116–124, 1972. doi:[10.1016/S0022-0000\(72\)80017-5](https://doi.org/10.1016/S0022-0000(72)80017-5).
- [PR81] Wolfgang J. Paul and Rüdiger Reischuk. On time versus space II. (Turing machines). *J. Comput. Syst. Sci.*, 22(3):312–327, 1981. doi:[10.1016/0022-0000\(81\)90035-0](https://doi.org/10.1016/0022-0000(81)90035-0).
- [PV76] Mike Paterson and Leslie G. Valiant. Circuit size is nonlinear in depth. *Theor. Comput. Sci.*, 2(3):397–400, 1976. doi:[10.1016/0304-3975\(76\)90090-6](https://doi.org/10.1016/0304-3975(76)90090-6).

- [Sha26] Yakov Shalunov. Improved bounds on the space complexity of circuit evaluation. In Dana Moshkovitz, editor, *41st Computational Complexity Conference, CCC 2026, Lisbon, Portugal, August 3-6, 2026*, volume 383 of *LIPICs*, pages 3:1–3:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2026. doi:[10.4230/LIPICs.CCC.2026.3](https://doi.org/10.4230/LIPICs.CCC.2026.3).
- [Vit06] Jeffrey Scott Vitter. Algorithms and data structures for external memory. *Found. Trends Theor. Comput. Sci.*, 2(4):305–474, 2006. doi:[10.1561/0400000014](https://doi.org/10.1561/0400000014).
- [Wil26] Ryan Williams. Simulating time with square-root space. *J. ACM*, 73(3):1–25, June 2026. doi:[10.1145/3798104](https://doi.org/10.1145/3798104).