Energy-Optimal and Low-Depth Algorithmic Primitives for Spatial Dataflow Architectures

Lukas Gianinazzi, Tal Ben-Nun, Maciej Besta, Saleh Ashkboos,
Yves Baumann, Piotr Luczynski, and Torsten Hoefler
ETH Zurich, Switzerland
Emails: {lukas.gianinazzi, tal.bennun, maciej.besta, saleh.ashkboos, yves.baumann, piotr.luczynski, hhtor}@inf.ethz.ch

***Abstract*—Spatial dataflow architectures, characterized by large arrays of processing elements communicating over a spatially localized on-chip network, offer unprecedented parallelism but introduce unique challenges for algorithm design. Unlike traditional shared-memory parallel systems, these architectures rely on local interconnects, where the distance between elements directly impacts communication efficiency. This necessitates new algorithmic approaches that balance parallelism and spatial locality. Fundamental operations like Parallel Scans, Rank Selection, and Sorting, which form the backbone of many algorithms—including those in graph neural networks and scientific computing—must be carefully adapted to minimize communication overhead. To address these challenges, we adopt the Spatial Computer Model, which quantifies communication costs through two key metrics: energy, representing the total distance traveled by messages (a proxy for network load), and depth, indicating the largest chain of dependent messages (critical for parallelism). In this work, we present the first energy-optimal algorithms for parallel scans, rank selection, and sorting within this model, achieving poly-logarithmic depth while maintaining tight upper and lower bounds on energy and distance. We demonstrate the applicability of these algorithms to the critical problem of sparse matrix-vector multiplication, which is central to scientific workloads and machine learning models. Our results lay the groundwork for designing energy-efficient and scalable algorithms on spatial dataflow architectures, highlighting the potential for further exploration of sparse algorithms and neural networks optimized for these systems.**

***Index Terms*—Sorting and searching, Routing and layout, Computations on matrices**

I. Introduction

Spatial dataflow architectures have emerged as powerful platforms for both data science and traditional high-performance computing (HPC) workloads due to their high throughput and parallelism [1], [2], [3], [4], [5], [6], [7], [8], [9]. These architectures, exemplified by cutting-edge systems like the Cerebras Wafer-Scale Engine [10], feature up to hundreds of thousands of parallel processing elements on a single chip. Each processing element is equipped with its own fast, local memory. Rather than using global memory, processing elements communicate through an on-chip mesh network, with communication costs varying based on the distance. While these innovations eliminate the shared memory bottleneck and enable high memory bandwidth, they also introduce unique challenges for algorithm design, especially in managing communication across the chip.

High throughput is only achievable on a spatial dataflow architecture when an algorithm exploits the problem's *spatial locality*, meaning communication primarily occurs between nearby processing elements on the chip. However, this locality should not come at the cost of reducing parallelism, which would reduce the scalability of the algorithms. Hence, traditional algorithms must be adapted to balance spatial locality and parallelism to operate efficiently on spatial dataflow architectures [11], [12]. The cost of communication between processing elements, which increases with physical distance, must be carefully managed to avoid performance bottlenecks. This requires developing and adapting key algorithmic primitives that minimize communication distance and leverage parallelism, making them suitable for the unique properties of spatial dataflow architectures.

Many complex algorithms rely on a small set of fundamental primitives, such as parallel scans (prefix sums), rank selection, and sorting. These operations form the backbone of numerous algorithms in both data science and machine learning. Specifically, they are key to efficient sparse matrix-vector multiplication (SpMV), a fundamental operation in scientific workloads [13], [14] and graph algorithms [15]. Moreover, they are integral to the construction of advanced neural networks, including graph neural networks (GNNs) with sort pooling layers [16], which rely on sorting as a critical operation for feature extraction and learning. Given the broad applicability of these primitives, it is essential to understand how to optimize their execution on spatial dataflow architectures. Traditional algorithms either ignore spatial locality because they assume all processors are symmetric with respect to each other [17], [18], [19] or provide insufficient parallelism [20], [21], [22], [23], showcasing a gap in the understanding of highly parallel, but spatially local computation primitives. Efficient algorithm designs for these operations will not only enhance individual algorithm performance but also increase the range of workloads these architectures can support, paving the way for more advanced algorithms and neural network models.

A. Methodology

We use the Spatial Computer Model [24], [11], a machine abstraction that enables algorithm designers to focus on optimizing for spatial locality and parallelism, rather than specific hardware details. It considers an unbounded number of processors with constant-sized memory organized in a Cartesian 2-dimensional grid. While the grid allows unrestricted

communication between processors, it imposes costs based on the distance between them. Specifically, sending a message from processor $p_{i,j}$ in the grid to processor $p_{x,y}$ in the grid has distance $|x-i|+|y-j|$. The largest total distance of any chain of dependent messages is the *distance* of an algorithm, which measures the latency incurred by sending the messages along wires.

The energy of a computation is the sum of the distance cost of the sent messages, and it measures the total load on the communication network. The longest chain of messages that consecutively depend on each other is the *depth* of the computation. A low depth indicates that the algorithm has many independent operations, allowing for a high degree of parallelism. In algorithm design, optimizing for depth and energy often involves trade-offs [11]. While minimizing depth allows for faster execution due to high parallelism, managing energy ensures efficient resource usage.

B. Limitations of State-of-the-Art Approaches

The problems of parallel scans, rank selection, and sorting have been widely studied across various architectures and models of computation [25], [26], [27]. Notably, sorting networks [28], [29], [30], [31] have been explored as a general-purpose approach for parallel sorting, offering a highly parallel solution that can work across different systems. In addition, significant work has focused on designing efficient parallel algorithms for shared-memory architectures [32], [33], [34], leveraging the availability of fast, concurrent access to a unified memory space. Other efforts have concentrated on distributed algorithms [35], [36], where processors communicate over message-passing protocols, and mesh-connected networks [20], [37], where communication happens between processors organized in a grid-like structure. Despite this wealth of research, none of these prior models or approaches directly address the unique challenges of spatial dataflow architectures. Specifically, unlike spatial dataflow architectures, these models do not need to balance spatial locality—quantified in our model as energy and distance—with parallelism, measured by depth. In spatial architectures, both factors are critical for achieving peak performance.

Previous work targeting the Spatial Computer Model has largely focused on basic collective communication primitives [11] such as reductions, broadcasts, and efficient tree layouts, along with operations performed on these trees [24]. While these primitives are foundational, they are not sufficient for handling more complex algorithms that operate on graphs, sparse matrices, or other irregular data structures. These problems introduce a higher level of complexity due to their irrecular access patterns, and existing solutions in the spatial model do not extend to these domains. Consequently, the question of which complex algorithms can be made both energy-efficient and low depth within this model has remained open until now.

TABLE I: Summary of our Spatial Computer Model bounds. We provide tight energy upper and lower bounds for the computational primitives of scan, selection, and sorting, from which we derive results for sparse matrix-vector multiplication (SpMV).

Problem	Energy	Depth	Distance
Parallel Scan §IV	$\Theta(n)$	$O(\log n)$	$\Theta(\sqrt{n})$
Sorting §V	$\Theta\left(n^{\frac{3}{2}}\right)$	$O(\log^3 n)$	$\Theta(\sqrt{n})$
Rank Selection* §VI	$\Theta(n)$	$O(\log^2 n)$	$\Theta(\sqrt{n})$
SpMV §VIII	$\Theta\left(n^{\frac{3}{2}}\right)$	$O(\log^3 n)$	$\Theta(\sqrt{n})$

$^{(\star)}$ randomized algorithm; bounds hold with high probability
$\mathbf{n}$ input size, grid size $\sqrt{n} \times \sqrt{n}$

C. Key Insights and Contributions

In this paper, we present the first energy- and distance-optimal algorithms with poly-logarithmic depth for three fundamental operations: parallel scans, rank selection, and sorting. These results establish new upper and lower bounds on the computational complexity of these operations in the Spatial Computer Model. An overview of the specific energy, depth, and distance bounds we achieve for each algorithm can be found in Table I. We provide a lower bound on the energy of the permutation operation, which states that there are permutations that take $\Omega(n^{\frac{3}{2}})$ energy. Because sorting can be used to implement any permutation, this result implies that our sorting algorithm is energy-optimal.

Sorting allows us to simulate any Parallel Random Access Machine (PRAM) algorithm on the Spatial Computer Model (see Section VII). While this simulation typically incurs additional energy, distance, and depth costs, it provides a convenient framework for transferring well-established algorithms into the spatial context without the need for detailed re-implementation. Although PRAM simulations generally do not yield optimal algorithms in spatial dataflow architectures due to the overhead of sorting the simulated shared memory, they are particularly useful for subroutines that do not dominate overall runtime and provide upper bounds quickly.

Our algorithmic tools serve as building blocks for a broader class of algorithms. To illustrate their utility, we demonstrate how these primitives can be combined to form more complex algorithms for the case study of sparse matrix multiplication, which is central to many applications in scientific computing and machine learning. We demonstrate how to apply our PRAM simulation results, and how a careful use of our algorithmic primitives can improve on the simulation result. The modularity of these building blocks makes them valuable for tackling a wide range of computational problems on spatial dataflow architectures.

D. Limitations of the Proposed Approach

Our approach assumes that each processing element has $O(1)$ memory, a reasonable assumption for large systems where the total memory far exceeds that of individual elements. A promising direction for future research is to generalize our algorithms for cases where local memory constitutes a significant fraction of total memory, which would be beneficial for systems with fewer processing elements. In this work, we emphasize parallel scalability and tailor our algorithms to systems like the Cerebras WSE-2 [10], which features *hundreds of thousands of processing elements*.

II. Related Work

A. Spatial Algorithms

Tree Algorithms. Previous work has investigated both theoretical and experimental aspects of the Spatial Computer Model. Baumann et al. [38] introduce low-depth, linear-energy algorithms on trees, including treefix sums (a generalization of parallel scans) and lowest common ancestor computations. Their algorithms use a spatially optimized tree layout, enabling neighboring nodes to exchange information with constant energy cost on average. While their treefix sum algorithm supports parallel scans, it requires $\Theta(n \log n)$ energy, which is not energy-optimal. Our results reduce the energy cost by a factor of $\Theta(\log n)$ for the case where the tree is a path.

Communication Collectives. Luczynski et al. [11] present several algorithms for *broadcast*, *reduce*, and *all-reduce* communication collectives, comparing theoretical bounds from the Spatial Computer Model with experimental results on the Cerebras WSE-2, a wafer-scale chip with approximately 850,000 processing elements [10]. Their findings reveal a trade-off between low-depth and low-energy algorithms, with low-depth algorithms showing superior performance given a larger number of processing elements. Moreover, they demonstrate that a key advantage of low-depth algorithms is reducing data movement between compute elements and network routers on the chip, which adds significant overhead in practice. Moreover, we introduce a parallel scan as a new spatial communication collective and present a logarithmic depth broadcast and reduce with optimal $O(n)$ energy without the need for multicasting–a $\log n$ improvement.

B. Sorting, Scan, and Selection in Other Models

Sorting Networks. Some of the earliest sorting algorithms were sorting networks [28], [29], [30], [31], [39]. Their fixed communication patterns make them suitable for hardware implementation. For dataflow architectures, these *data-oblivious* algorithms are advantageous, as their routing patterns depend only on input size. In this paper, we demonstrate how sorting networks can be mapped onto a spatial dataflow architecture, achieving low-depth execution and spatial locality. However, this approach is energy-inefficient because sorting networks typically operate on a one-dimensional array, failing to leverage the two-dimensional nature of spatial architectures.

Fixed-Connection Network Model. Fixed-connection network models [40], [41] study computation on specific communication topologies, including tori, hypercubes, and meshes. Mesh-connected computers [20], [21], [22], [23] are particularly relevant for our work because they optimize for spatial locality. Any algorithm on a mesh network can be adapted to the Spatial Computer Model, where an algorithm requiring K rounds on a $\sqrt{n} \times \sqrt{n}$ mesh network incurs $O(Kn)$ energy with depth K and distance $O(K)$. However, many problems such as sorting cannot be solved in sub-polynomial rounds on a mesh-network. In particular, the optimal sorting algorithm takes $\Theta(\sqrt{n})$ rounds [42], resulitng in $\Theta(\sqrt{n})$ depth. We improve on this significantly, reducing the depth to poly-logarithmic while maintaining optimal energy and distance.

Work-Depth/PRAM. The work-depth model models computation as a directed acyclic graph and counts the total number of operations (work) and the length of the critical path (depth). This model is closely related to the PRAM model [17], which assumes a single shared memory with uniform cost random access. An algorithm with work W and depth D takes $O(W/p + D)$ time on p PRAM processors [43]. Optimal parallel sorting takes $O(\log^2 n)$ time with $O(n/\log n)$ processors on an exclusive-write exclusive-read machine [44]. Executing such an algorithm would lead to sub-optimal energy and distance (see Section VII). Similarly, while selection and scanning takes $O(\log n)$ time using $O(n/\log n)$ processors [45], [17], simulating these algorithms would require $\Omega(n^{\frac{3}{2}})$ energy – a polynomial factor worse than our results. In summary, PRAM algorithms, designed for random access memory with no consideration for spatial locality, are highly inefficient for spatial dataflow architectures, resulting in significantly higher energy consumption and distance costs.

BSP. The bulk-synchronous parallel model [19] (BSP), operates in synchronous rounds, or supersteps, where processors can exchange arbitrarily large messages. The model aims to minimize the number of rounds, communication volume, and computation time. Comunication costs are uniform between processors. BSP is well-suited for coarse-grained parallelism, where the number of processors is much smaller than the input size, and each processor has a large local memory—common in large clusters. While efficient sorting, selection, and scan algorithms have been developed for this model [35], [36], [46], they are ill-suited to our setting. In spatial dataflow architectures, execution is asynchronous, memory per processing element is limited, and communication costs depend heavily on the spatial proximity of processors. As a result, BSP-based algorithms fail to effectively utilize these architectures.

III. Preliminaries

We introduce some preliminaries that we use throughout.

Spatial Computer Notation. Initially, an input of size n is distributed in some predefined format on an $h \times w$ sized processor *subgrid* with $n = h \cdot w$. We refer to the processor at coordinate (i, j) as processor $p_{i,j}$. For simplicity, we assume w.l.o.g. that n is a power of 4.

Z-Order Curve. As observed in previous work [24], storing arrays according to a space-filling curve's traversal of the grid enables improving the spatial locality of certain parallel algorithms. The *Z-Order curve* (sometimes called Morton space filling curve [47], [48]) is one such traversal of a grid. We can define it recursively: Traverse the four quadrants of the grid in order, visiting the top two quadrants first, left to right, then the bottom two quadrants, left to right.

Observation 1. *Sending a message along each edge of a Z-Order curve of a $\sqrt{n} \times \sqrt{n}$ subgrid takes $O(n)$ energy [11].*

High Probability. A bound $f(n) \in O(g(n))$ holds with high probability (w.h.p.), if for any constant d, there exist constants $n_0 > 0$ and $c > 0$ such that for all $n \geq n_0$, $f(n) \leq cg(n)$ with probability at least $1 - n^{-d}$.

IV. Communication Collectives

We introduce our improved broadcast and reduce collectives and then introduce our novel scan algorithm design.

A. Broadcast Without Multicasting

Consider the problem of broadcasting a value from the processor $p_{0,0}$ to all other processors in an $h \times w$ sub-grid that contains $p_{0,0}$. Let us first consider the square $w \times w$ case (2D broadcast). We can solve the problem efficiently by subdividing the grid into quadrants and proceeding recursively on them: Send the value to the three processors in the top-left corners of the other quadrants, then proceed recursively on each quadrant. Next consider the $h \times 1$ case (1D broadcast). We build a binary broadcast tree, as follows. The root has a child node directly next to it and a child node at an offset $(h-1)/2$. Then, recursively build a binary tree for each child's subtree (each contains $(h-1)/2$ elements).

Now, consider the general case, where we want to broadcast on an $h \times w$ grid, where $h \geq w$. First, do a 1D broadcast on the first column. Then, subdivide the grid into square $w \times w$ subgrids and run a 2D broadcast on each of them.

Lemma IV.1. *Broadcast on an $h \times w$ subgrid takes $O(hw + h \log h)$ energy, $O(\log n)$ depth, and $O(w + h)$ distance.*

Proof: The 1D broadcast takes $O(h \log h)$ energy [11]. The following $\lceil \frac{h}{w} \rceil$ 2D broadcasts take $O(w^2)$ energy each: Let $E(w)$ be the energy required for the 2D broadcast on a $w \times w$ subgrid. Then, we have that:

$$E(w) \leq \begin{cases} 0 \text{ if } w \leq 1 \\ \frac{3w}{2} + 3 + 4E(w/2) \text{ otherwise,} \end{cases}$$

which solves to $O(w^2)$. The depth is clearly $O(\log w + \log h) = O(\log n)$. Finally, the distances of both the 1D and 2D broadcast form geometric series and solve to $O(w + h)$. □

B. Low-Depth Reduce

Given an associative and commutative operator $\circ$ and n inputs $A_0, \ldots, A_{n-1}$ stored in arbitrary order on an $h \times w$ subgrid containing the processor $p_{0,0}$, a *reduce* computes $A_0 \circ A_1 \circ \ldots \circ A_{n-1}$ and leaves the result in the root processor $p_{0,0}$. To compute a reduce, we can use the reverse commmunication pattern as the broadcast. Hence, the result follows:

Corollary IV.2. *Reduce on an $h \times w$ subgrid takes $O(hw + h \log h)$ energy, $O(\log n)$ depth, and $O(w + h)$ distance.*

On a square subgrid, previous $O(\log n)$-depth Reduce took $\Omega(n \log n)$ energy [11], so this constitutes a $\Theta(\log n)$ factor energy improvement in the logarithmic-depth regime.

C. Parallel Scan

The parallel scan is a fundamental communication primitive [49], [34]. In this section, we introduce an energy-optimal scan algorithm with logarithmic depth, which will be applied in Section VIII for sparse matrix-vector multiplication.

Consider an array $A_0, \ldots, A_{n-1}$ stored on a $w \times w$ grid. Our goal is to compute the prefix sums

$$A_0,\ A_0 + A_1,\ A_0 + A_1 + A_2, \ldots, \sum_{i=0}^{n-1} A_i$$

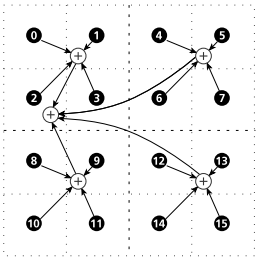
where the i-th result $\sum_{j=0}^{i} A_j$ is stored in the same location as the i-th input. While this operation can be generalized for any associative operator, we will focus on addition for clarity.

A naive 1D parallel prefix sum, implemented via a binary tree over the array in row-major order, would incur $\Omega(n \log n)$ energy, similar to the energy cost of a binary tree broadcast [11]. A sequential prefix sum, while requiring only $O(n)$ energy, would suffer from $\Omega(n)$ depth. In contrast, our 2D scan algorithm reduces the energy cost to $\Theta(n)$ while achieving $O(\log n)$ depth by using a 4-ary summation tree structured in Z-Order. This method fully takes advantage of the 2D grid layout, featuring small communication distances while achieving high parallelism.

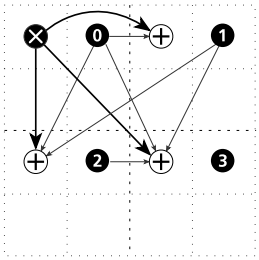
Energy-Optimal Scan. Our scan algorithm consists of two phases: an *up-sweep* followed by a *down-sweep*. During the up-sweep phase, partial sums are computed across quadrants, progressively aggregating results from smaller subgrids. The down-sweep phase then uses these partial sums to compute the final prefix sums. This process forms a 4-ary summation tree over the grid, where the root corresponds to the entire subgrid and its children represent the four quadrants, recursively. By keeping communication primarily within quadrants, this structure ensures efficient energy usage. The prefix sums are computed in Z-Order, which optimizes the spatial locality of the computation (see Section III for details). A visual representation of the grid mapping is provided in Figure 1a.

a) Up-sweep: For each node in the quadrant tree, we want to compute the sum of its leaf elements. If the current subgrid contains a single processor, its element equals the value at a leaf. Otherwise, proceed recursively.

- Recurse over all quadrants to obtain the sum of the element of the children.
- Sum those values, store the result in the i-th processor of the current subgrid in Z-order, where i is the distance to a leaf in the 4-ary summation tree.



(a) The up-sweep computes partial sums along a 4-ary tree. The root of a height i subtree is in the i-th node in Z-Order of the subtree's quadrant.



(b) The down-sweep computes prefix sums over the values from the up-sweep. It sends these prefix sums to the quadrants recursively.

Fig. 1: The energy-optimal scan operates recursively in Z-order. It first runs an up-sweep, followed by a down-sweep.

b) Down-sweep: Now, we use the values from the up-sweep to compute the prefix sum. The algorithm again recursively considers the subgrid's quadrants. At each step, a value x gets passed down from above. For the first invocation, $x = 0$. This values x is added to all values in the current quadrant to account for values that occur outside the current subgrid.

If the current subgrid has size 1, add x to the value of A in the subgrid's only processor. Otherwise, proceed recursively. Consider the four quadrants in the current subgrid, S_0, S_1, S_2, S_3 in Z-order and their respective values s_0, s_1, s_2, s_3 that were computed during the up-sweep. The value that gets passed down to quadrant S_i is $x + \sum_{j=0}^{i-1} s_j$. These values are passed down to the top left processor of each subgrid. Figure 1b illustrates one step of the down-sweep.

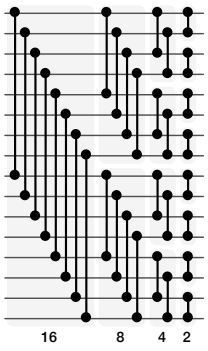
Lemma IV.3. *A scan on an array of n elements in Z-order takes $O(n)$ energy, $O(\log n)$ depth, and $O(\sqrt{n})$ distance.*

Proof: The energy equals that of a Z-order curve up to constant factors. In the up-sweep, each processor stores at most 2 values of the summation tree. Because of the recursive construction, the distance forms a geometric series. □

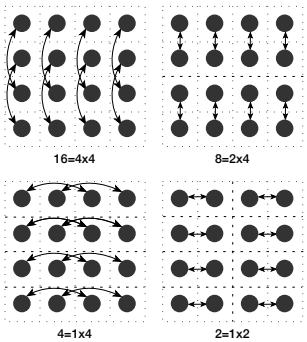
Segmented Scan. To operate efficiently on multiple consecutively stored arrays, we consider *segmented scans*. The input array is partitioned into consecutive *segments*. The result of a segmented scan is the same as executing a scan on each segment. For any associative operator, we can define a segmented associative operator that has the logic of the segments built-in [17]. Hence, we can use the same algorithm for a segmented scan.

V. Sorting

We present a low-depth, energy-optimal sorting algorithm based on mergesort, tailored for spatial dataflow architectures. The primary challenge in this approach is designing a spatially local, yet low-depth, merge subroutine to minimize communication distances while maintaining logarithmic depth. Our energy-optimal mergesort improves significantly on the energy cost of bitonic sorting networks, reducing the overall energy consumption by a factor of $O(\log n)$.



(a) 1D Bitonic Merge



(b) 2D Bitonic Merge

Fig. 2: Bitonic Merge network in 1D and 2D layout. In 2D, the recursion first splits into fewer columns (4x4 - 2x4 - 1x4), then fewer rows (1x2). Because the recursion does not reduce both rows and columns simultaneously, 2D Bitonic Mergesort is suboptimal in terms of energy.

A. Lower Bound

It is easy to prove a non-linear lower bound on the energy to permute (and hence sort) the input.

Lemma V.1. *Permuting $h \times w$ elements on an $h \times w$ subgrid takes $\Omega(\max(w,h)^2 \cdot \min(w,h))$ energy.*

Proof: Assume w.l.o.g. that $h > w$. Otherwise, the situation is transposed. Consider the permutation that reverses the order in which elements appear in a row-wise layout. The elements in the first $h/3$ rows need to be sent to one of the last $h/3$ rows, which takes at least $h/3$ energy per element (of which there are $hw/3$). □

The lowest permutation cost is obtained when $w = h$. Since permutation reduces to sorting, the lower bound follows:

Corollary V.2. *Sorting n elements takes $\Omega(n^{\frac{3}{2}})$ energy.*

By the permutation lower bound, the matching upper bound can only be obtained when the input is contained in a $h \times w$ subgrid where $w = \Theta(h)$. Hence, we will focus on the case where $w = h$. We begin by analysing the energy of a sorting network in our model, which we show to be sub-optimal. Then, we present an energy-optimal 2D Mergesort algorithm, which has $O(\log^3 n)$ depth and $O(n^{3/2})$ energy on a $\sqrt{n} \times \sqrt{n}$ grid.

B. Sorting Networks

Sorting networks are data-oblivious (and stable) sorting algorithms of oftentimes low depth [30], [28], [29], [31] . For each time step, they define a set of pairs of indices to compare (and swap if necessary). Each index into the array is thought of as a "wire". In each step, a wire can be compared with at most one other wire. A natural idea is to map a sorting network to our processor grid: each wire in the sorting network is assigned to a processor. For example, we can assign wires to processors in row-major order. Interestingly, this approach does not easily lead to energy-optimal sorting algorithms. We present the results for Bitonic Sort [30].

The Bitonic Sort is a simple network with $O(\log^2 n)$ depth and $O(n \log^2 n)$ comparisons. As it is defined recursively on

halves of its input, it exhibits some degree of spatial locality. A Bitonic Sort makes use of a Bitonic Merge network, which can be defined recursively: For an input of size n, compare and swap each wire i with index less than $n/2$ with wire $i + n/2$. Then, recursively merge both halves. See Figure 2 for an illustration of a 4×4 Bitonic Merge with wires mapped to the compute grid in row-major order. A Bitonic Sort recursively invokes itself on both halves of its input, then invokes a Bitonic Merge on the input. We begin with the analysis of the recursive Bitonic Merge.

Lemma V.3. *On an $h \times w$ subgrid, Bitonic Merge takes $\Theta(h^2w+w^2h)$ energy, $\Theta(\log n)$ depth, and $\Theta(w+h)$ distance.*

Proof: We split the energy cost into two parts: (1) when there is more than one row left; (2) when there is a single row left. When there are $h > 1$ rows left, the network sends $\Theta(w \cdot h)$ messages across a distance of $h/2$. Hence, the energy $E_1(h)$ for this part is $E_1(h) = \Theta(h^2w) + 2E_1(h/2)$, which solves to $\Theta(h^2w)$. When there is a single row left of length w, the network sends $\Theta(w)$ messages across a distance of $\Theta(w)$. Hence, the energy $E_2(w)$ for this part is $E_2(w) = \Theta(w^2)+2E_2(w/2)$, which is in $\Theta(w^2)$. The algorithm reaches the situation h times, so the total cost of this part is $O(hw^2)$. The distance is a geometric series. □

Next, we describe and analyze the cost of the bitonic sorting network. Because the Bitonic Sort has a 1D recursive pattern that first reduces the number of rows, and only then the number of columns, it has to pay the energy of the Bitonic Merge a logarithmic number of times in one dimension, leading to the following bound:

Lemma V.4. *On an $h \times w$ subgrid, Bitonic Sort takes $\Theta(h^2w + w^2h \log h)$ energy, $\Theta(\log^2 n)$ depth, and $\Theta(h + w \log h)$ distance.*

Proof: We again divide the energy cost into the part where there is more than one row and the part when there is a single row left. The energy $E_1(h)$ when there are $h > 1$ rows left is

$$E_1(h) \leq O(h^2w + w^2h) + 2E(h/2) \text{ if } h > 1.$$

We see that $E_1(h) = O(h^2w + w^2h \log h)$. When there is a single row left of length w, the energy $E_2(w)$ for this part is

$$E_2(w) \leq O(w^2) + 2E(w/2) \ ,$$

which solves to $O(w^2)$. This cost occurs h times.
For the distance, observe that as long as there are is more than one row, the cost is $O(w+h)$. Because w stays the same while there is more than one row, the distance is $O(w \log h + h)$. □

Discussion. In conclusion, Bitonic Sort takes $O(n^{\frac{3}{2}} \log n)$ energy to sort n numbers, a logarithmic factor away from optimal. Note that the sub-optimality is not because of the suboptimal number of comparisons performed by the bitonic sorting network, but instead because the network eventually turns into a 1D algorithm (when the recursion becomes smaller than a single row). Moreover, Bitonic Sort is also not distance optimal, as it has $\Theta(\sqrt{n} \log n)$ distance. We now present an energy and distance optimal algorithm.

C. Energy-Optimal Sorting

We design a spatial energy-optimal variant of Mergesort [50], [51], [52]. On a high level, the 2D Mergesort works similarly to traditional Mergesort. However, instead of recursing on two halves of the array, we recurse on the four sub-quadrants:

- Recursively sort the four quadrants of the subgrid.
- Merge the two top quadrants
- Merge the two bottom quadrants
- Merge the results of the two previous merges

The challenge lies in an energy-efficient implementation of the Merge subroutine, that we present in the rest of this subsection. Our merging subrouting relies on a naive sorting routine (All-Pairs Sort), which we discuss next.

a) All-Pairs Sort: A simple idea for a low-depth sorting algorithm is to compare every element with every other element. This can be done by using our efficient broadcast and reduce patterns. The implementation "explodes" the computation onto a larger subgrid. This leads to low depth. However, because the computation grid has a larger diameter, the energy cost increases to more than quadratic.

1) Subdivide an $n \times n$ subgrid into n subgrids Γ_i of size $\sqrt{n} \times \sqrt{n}$ each. Scatter the elements of A such that A_i is sent to the first processor in subgrid Γ_i, for each i.
2) Within each subgrid Γ_i, broadcast the element A_i.
3) Copy the array A to each grid Γ_i by using the same communication pattern as for the 2D broadcast (by treating the array A and the subgrids Γ_i as units).
4) Now, each processor compares its two elements.
5) Each subgrid Γ_i performs a reduction to compute the rank of element A_i in the sorted sequence. Gather these ranks.

Lemma V.5. *All-Pairs Sort of n elements takes $O(n^{5/2})$ energy, $O(\log n)$ depth, and $O(n)$ distance.*

Proof: Scattering the n elements over a distance of at most $O(n)$ takes $O(n^2)$ energy. Let $E(k)$ be the energy of the broadcast when $k \cdot k$ subgrids remain. Then, we have the recurrence

$$E(k) \leq \begin{cases} O(n) \text{ if } k \leq 1 \ ; \\ n^{3/2}k + 4E(k/2) \text{ otherwise,} \end{cases}$$

which is in $O(n^{3/2}k^2)$. As initially, $k = \sqrt{n}$, we get that the energy is $O(n^{5/2})$. Computing the reductions takes $O(n^2)$ energy and gathering these ranks takes $O(n^2)$ energy. The depth is bottlenecked by the broadcasts. Finally, the distance is bottlenecked by scattering the elements in the $n \times n$ subgrid. □

b) Merging Two Sorted Arrays: The challenging part of our merging algorithm is an energy-efficient and low-depth merging subroutine. We cannot use classical approaches because they do not lead to balanced recursive cases (which is needed to organize them into square subgrids). Moreover, using a binary search as a subroutine leads to a sub-optimal distance. Consider two arrays A and B containing n_A and n_B

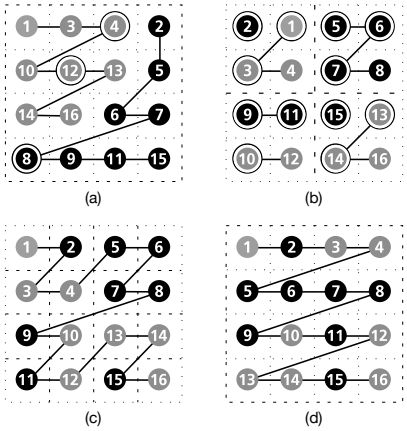


Fig. 3: (a-c) The 2D merge recursively splits the two sorted arrays (colored in black and grey) by the encircled rank $n/4$, $n/2$ and $\frac{3}{4}n$ elements into quadrants. (d) Finally, it permutes the array from Z-Order into row-major order.

sorted numbers in row-major order. The goal is to construct an array C that contains the $n = n_A + n_B$ elements of A and B in sorted row-major order.

At any point in the recursion, the algorithm operates on a square subgrid Γ. The larger of the two current subarrays A and B is stored in a square subgrid $\Gamma' \subseteq \Gamma$, while the other array is stored in row-major order filling up the rest of the subgrid Γ, in a "mirrored L" shape. See Figure 3 for an example. The idea of the algorithm is as follows.

1) Let $A\|B$ be the concatenation of A and B. Split A and B by the rank $n/4$, rank $n/2$ and rank $3n/4$ elements of $A\|B$ into 4 subarrays each.
2) Reorganize the resulting subarrays into the four quadrants, such that the first quadrant contains the $n/4$ smallest elements of $A\|B$, and so on.
3) Recursively merge each quadrant.
4) After finishing the recursion, the array is sorted in Z-Order. Hence, permute the array to row-major order.

c) Rank Selection in Two Sorted Arrays: To implement the merge, we need to find the rank k element of $A\|B$ (in particular for $k = n/4, n/2, 3n/4$), known as a multiselection problem [53]. The idea is to quickly rank a small subset of the elements using All-Pairs Sort and determine much smaller subarrays of A and B that contain the rank-k element.

1) Gather every $\lfloor\sqrt{n}\rfloor$-th element of A and B into a sample S. Specifically, select from A the elements at indices $i\lfloor\sqrt{n}\rfloor$ for i in the range $0, \ldots, \lfloor n_A/\lfloor\sqrt{n}\rfloor\rfloor$ and similarly for B.
2) Sort the sample S with an All-Pairs Sort.
3) Let $l = \lfloor\frac{k-1}{\lfloor\sqrt{n}\rfloor}\rfloor$.
4) Find S_l, the l-th ranked element in S. Using binary search, locate its predecessors A_a in array A and B_b in array B. If no predecessor exists, use the first element of the respective array.
5) Narrow the search to the two subarrays $A_a, \ldots A_{\min(n_A, a+2\lfloor\sqrt{n}\rfloor)}$ and $B_b \ldots B_{\min(n_B, b+2\lfloor\sqrt{n}\rfloor)}$.
6) Within these narrowed subarrays, locate the rank-$(k-a-b)$ element using All-Pairs Sort.

Lemma V.6. *Given two sorted arrays A and B, we can determine the rank k element in $O((n_A+n_B)^{5/4})$ energy, $O(\log n)$ depth, and $O(\sqrt{n})$ distance.*

Proof: Energy, depth, and distance are bottlenecked by the All-Pairs Sort of the sample S. Because S has $O(\sqrt{n_A+n_B})$ elements, the result follows by Lemma V.5.

Next, we prove the correctness of the algorithm. *Case 1*: $l = 0$. Then k is at most $\lfloor\sqrt{n}\rfloor$. Hence, it must be contained within the first $\sqrt{n}$ elements of one of the two subarrays. *Case 2*: $l > 0$. Because we sample every $\lfloor\sqrt{n}\rfloor$ element, the rank of S_l in $A\|B$ is at most $l\lfloor\sqrt{n}\rfloor \leq k-1$. Hence, by removing elements smaller than s_l we do not exclude the rank k element. Moreover, the rank of s_l in $A\|B$ is at least $k-1-2\lfloor\sqrt{n}\rfloor$. Since we always consider the next $2\lfloor\sqrt{n}\rfloor+1$ ranked elements, correctness follows.

□

We now bound the cost of the merging subroutine.

Lemma V.7. *Merging two arrays A and B with n_A and n_B elements located on adjacent square subgrids takes $O((n_A+n_B)^{3/2})$ energy, $O(\log^2(n_A+n_B))$ depth, and $O(\sqrt{n})$ distance.*

Proof: Let $E(n_A, n_B)$ be the energy of merging arrays A and B with n_A and n_B elements in total. Then, the energy in each step of the recursion is $O((n_A+n_B)^{5/4})$ for determining the splitting elements and $O((n_A+n_B)^{3/2})$ for performing the necessary permutations. Each of the recursive calls operates on four pairs of subarrays. We get the energy recurrence

$$E(n_A, n_B) \leq \begin{cases} (n_B)^{\frac{3}{2}} & \text{if } n_A = 0 \\ (n_A)^{\frac{3}{2}} & \text{if } n_B = 0 \\ O((n_A+n_B)^{3/2}) + \sum_{i=1}^{4} E(n_A^i, n_B^i) & \text{else,} \end{cases}$$

where $n_A^i + n_B^i = (n_A+n_B)/4$. Since the number of elements remains constant when summed over all nodes in the same level of the recursion, and the 'diameter' term $\sqrt{n_A+n_B}$ decreases geometrically in each level of the recursion (for all recursive calls), the recurrence solves to $E(n_A, n_B) \leq O((n_A+n_B)^{3/2})$. □

Note that for the case where $n_A = n_B$, the energy to merge the subarrays is $O(n^{\frac{3}{2}})$, the permutation bound.

Finally, we can bound the costs of 2D Mergesort:

Theorem V.8. *2D Mergesort takes $O(n^{3/2})$ energy, $O(\log^3 n)$ depth, and $O(\sqrt{n})$ distance on a square subgrid.*

Proof: By Lemma V.7 , the energy $E(n)$ is

$$E(n) \leq \begin{cases} 0 & \text{if } n \leq 1; \\ O(n^{\frac{3}{2}}) + 4E(n/4) & \text{else,} \end{cases}$$

which solves to $O(n^{\frac{3}{2}})$. □

Discussion. We introduced an energy-optimal 2D mergesort algorithm tailored for spatial dataflow architectures. To handle smaller portions of data, we employed an auxiliary sorting

algorithm with $O(\log n)$ depth, albeit with a higher energy cost of $O(n^{\frac{5}{2}})$. Notably, the permutation lower bound is a factor $\sqrt{n}$ higher than the linear energy bounds for communication collectives such as scan and reduce, as well as tree operations [38]. This reveals a polynomial separation between energy consumption in the Spatial Computer Model and the processor-time product in PRAM machines [17].

VI. Rank Selection

Selecting an element of a certain *rank* plays a crucial role in nonparametric statistics [54]. We show that we can determine the median of n elements, and more generally the rank-k element with *linear energy* and poly-logarithmic depth. This demonstrates a polynomial factor energy separation between selection and sorting in the spatial model. We can assume without loss of generality that $k \leq \lceil n/2 \rceil$, as we can reverse the order of the elements and select the rank $n-k$ element.

The idea is similar to the deterministic ranking we used for merging sorted subarrays. However, since we do not start with a partially sorted array, we have to sample randomly. We create a sample that is as large as possible, rank this sample, and use the ranking of the sample to eliminate a polynomial fraction of the input elements. Because it takes $O(\sqrt{n})$ energy to gather an element in a subgrid, the largest sample we can gather in $O(n)$ energy contains $O(\sqrt{n})$ elements. The sampling idea is similar to an idea by Reishuk [32].

Initially, all elements are marked as *active*. Elements are gradually marked *inactive* until we find the rank-k element. Let N be the current number of *active* elements. Choose a constant c with $c \geq 3$. Repeat the following until $N \leq c\sqrt{n}$:

1) Create a sample S by including every active element independently with probability $cN^{-1/2}$.
2) Gather those elements in a square subgrid, using a scan to assign each sampled element an index within the subgrid and a broadcast to communicate the size of the sample.
3) Choose *two pivots*. The first pivot is the element s_r at rank $r = \min(|S|, ckN^{-1/2} + \frac{c}{2}N^{1/4}\sqrt{\ln n})$. If $k \geq \frac{1}{2}N^{3/4}\sqrt{\ln n}$, the second pivot is the element s_l at rank $l = ckN^{-1/2} - \frac{c}{2}N^{1/4}\sqrt{\ln n}$. Otherwise, the second pivot is the dummy element $s_l = -\infty$. Sort the sample S using Bitonic Sort to determine s_l and s_r.
4) Broadcast s_l and s_r in the original subgrid.
5) Count the number of active elements $N_{<l}$ smaller than s_l and the number of active elements $N_{>r}$ larger than s_r with an all-reduce. If $N_{<l} \geq k$ or $N_{>r} \geq N-k$, sort the input using 2D Mergesort and return the rank k element. Otherwise, set $k = k - N_{<l}$ and continue.
6) For each active element a, inactivate it if $a < s_l$ or $a > s_r$.
7) Count the number of remaining active elements N with an all-reduce. If $k > \lceil N/2 \rceil$, set $k = N-k$ and reverse the order of the elements (logically, that is, by henceforth flipping the result of the comparator).

Once the iteration terminates, gather the elements in a square subgrid, sort them and return the rank k element.

Our goal is to show that $O(1)$ iterations suffice. The idea behind the energy efficiency proof is that the number of input elements of rank at most k is highly concentrated around their expectation. Hence, the probability that the true rank k element is between the pivot elements s_l and s_r is high.

Lemma VI.1. *The probability that $N_{<l} \geq k$ or $N_{>r} \geq N-k$ is at most $2n^{-c/6}$.*

Proof: Let K be the random variable denoting the number of rank at most k elements of the input that are sampled. Let $\delta = \frac{cN^{1/4}\sqrt{\ln n}}{2\,\mathbb{E}[K]}$. We first consider the case where $k \geq \frac{1}{2}N^{3/4}\sqrt{\ln n}$ and there are two non-trivial pivots. Observe that $N_{<l} \geq k$ occurs when $K > l$ and $N_{>r} \geq N-k$ occurs when $K < r$. Note that $\mathbb{E}[K] = ckN^{-1/2}$. Hence, it remains to bound the probability that K deviates from its expectation by more than $\frac{c}{2}N^{1/4}\sqrt{\ln n}$. Note that $0 < \delta \leq 1$. By a Chernoff bound [55], we get

$$\begin{aligned} P\left[|K - \mathbb{E}[K]| \geq \frac{c}{2}N^{1/4}\sqrt{\ln n}\right] &= P\left[|K - \mathbb{E}[K]| \geq \delta\,\mathbb{E}[K]\right] \\ &\leq 2e^{-\delta^2\,\mathbb{E}[k]/3} \\ &\leq 2e^{-\frac{c^2 N^{1/2}\ln n}{12\,\mathbb{E}[K]}} \\ &\leq 2n^{-c/6} \ . \end{aligned}$$

For the case where $k < \frac{1}{2}N^{3/4}\sqrt{\ln n}$, we have that $N_{<l} = 0$. Thus, we only need to bound $P[K \geq (1+\delta)\,\mathbb{E}[k]]$. Note that $\delta > 1$. We can conclude by another Chernoff bound [55]:

$$P\left[K \geq (1+\delta)\,\mathbb{E}[k]\right] \ \leq \ e^{\delta\,\mathbb{E}[k]/3} \ < \ e^{-2N^{1/4}} \ .$$

□

Next, we bound the size of the number of active elements after one iteration. The idea is that it is unlikely that more than the expected number of elements are between s_l and s_r.

Lemma VI.2. *Let N_t be the number of active elements after the t-th iteration, $N_0 = n$. Given $N_t = n_t$ and any constant $0 < \epsilon < 1$, with probability at least $1 - e^{-c\epsilon n_t^{1/4}\sqrt{\ln n}/4}$, we have that $N_{t+1} \leq (1+\epsilon)n_t^{3/4}\sqrt{\ln n}$.*

Proof: We define a binomially distributed random variable X to bound the probability, as follows. Consider the rank of s_l within the array after the t-th iteration. Now consider the next $(1+\epsilon)n_t^{3/4}\sqrt{\ln n}$ subsequently ranked elements (after the t iteration) in order. If an element is sampled in the $(t+1)$-th iteration, it is counted as a success. Recall that this occurs with probability $cn_t^{-1/2}$. The event that $N_{t+1} > (1+\epsilon)n_t^{3/4}\sqrt{\ln n}$ occurs exactly when $X \leq cn_t^{1/4}\sqrt{\ln n}$. Note that $\mathbb{E}[X] = (1+\epsilon)cn_t^{1/4}\sqrt{\ln n}$. We bound the tail probability of X by a Chernoff bound (for $\delta = \frac{\epsilon}{1+\epsilon}$):

$$\begin{aligned} P[X \leq cn_t^{1/4}\sqrt{\ln n}] &= P[X \leq (1-\delta)\,\mathbb{E}[X]] \\ &\leq e^{-\delta^2\,\mathbb{E}[X]/2} \\ &= e^{-\frac{(\frac{\epsilon}{1+\epsilon})cn_t^{1/4}\sqrt{\ln n})}{2}} \\ &\leq e^{-\frac{c\epsilon}{4}n_t^{1/4}\sqrt{\ln n}} \ . \end{aligned}$$

□

Theorem VI.3. *Rank Selection takes $O(n)$ energy, $O(\log^2 n)$ depth, and $O(\sqrt{n})$ distance with high probability in n. The same bounds also hold in expectation.*

Proof: It takes $O(n)$ energy to send the $O(\sqrt{n})$ sampled elements across the $O(\sqrt{n})$ diameter compute grid. Sorting the sample takes $O(n^{3/4}\log n) = o(n)$ energy. The remaining operations take $O(n)$ energy using our communication primitives. Hence, each iteration takes $O(n)$ energy. The depth is bottlenecked by the Bitonic Sort, which takes $O(\log^2 n)$ depth. The distance is $O(\sqrt{n})$ in each iteration. For all N_t larger than a constant, by Lemma VI.2 we have that $N_{t+1} \leq N_t^{4/5}$ with high probability in n. Hence, the algorithm performs a $O(1)$ iterations. By Lemma VI.1, each of those iterations resorts to sorting the whole input with probability at most $2n^{-c/6} \leq 2n^{-1/2}$, which implies the expectation bounds. Adjusting the constant c boost the probability of success to $1 - n^{-d}$ for any constant d. □

VII. PRAM Simulation

Simulating PRAM algorithms in our spatial model offers a convenient way to quickly derive upper bounds for various problems. The spatial layout of processors allows us to map PRAM computations to a 2D grid, where shared memory can be emulated by a dedicated subgrid of processors, and PRAM processors operate in another subgrid. By organizing memory and computation in this way, we can efficiently simulate the PRAM's operations in the spatial model. For problems involving concurrent reads and writes, we can employ our energy-optimal sorting and scan algorithms to manage concurrency.

A. EREW Simulation.

Let us start with the *Exclusive Read Exclusive Write* (EREW) PRAM simulation. In each synchronous time step, an EREW PRAM processor can read $O(1)$ word-sized memory cells, perform $O(1)$ computation, and write to $O(1)$ memory cells. No two processors can write or read the same memory cell in the same time step. By simulating the shared memory as a square subgrid of processors and placing the PRAM processors into a square subgrid (next to the memory), we obtain a simulation result for EREW PRAM:

Lemma VII.1. *Consider an algorithm A on an EREW PRAM that uses m memory cells and runs in T_p time steps on p processors. Simulating algorithm A takes $O(p(\sqrt{p}+\sqrt{m})T_p)$ energy, $O(T_p)$ depth, and $O((\sqrt{p}+\sqrt{m})T_p)$ distance.*

Proof: Organize the PRAM processors on a $\sqrt{p} \times \sqrt{p}$ subgrid and the PRAM memory cell on a $\sqrt{m} \times \sqrt{m}$ subgrid next to it. Each processor sends a message to each memory cell to request it. Each memory cell that received a request answers it with a message. The processors compute their result and send the result back to the memory cell. Each of the T_p simulated PRAM step takes $O(1)$ depth, $O(\sqrt{p}+\sqrt{m})$ distance and $O(p(\sqrt{p}+\sqrt{m}))$ energy. □

B. CRCW Simulation.

In the *Concurrent Read Concurrent Write* (CRCW) PRAM model, multiple processors can simultaneously access the same memory cell for reading or writing. When several processors attempt to write to the same cell, one of the writes arbitrarily succeeds. In our spatial model, we simulate this concurrency by using energy-optimal sorting to resolve both read and write conflicts. Sorting ensures that concurrent read values are broadcast efficiently, and only one write is applied to each memory cell. However, it increases the simulation depth due to additional sorting steps required to coordinate memory access.

Lemma VII.2. *Consider an algorithm A on an CRCW PRAM that uses m memory cells and runs in T_p time steps on p processors. Simulating algorithm A takes $O(p(\sqrt{p}+\sqrt{m})T_p)$ energy, $O(T_p \log^3 p)$ depth, and $O((\sqrt{p}+\sqrt{m})T_p)$ distance.*

Proof: Index the PRAM processors and memory cells with one-dimensional indexes. Organize the PRAM processors on a $\sqrt{p} \times \sqrt{p}$ subgrid indexed in Z-order and the PRAM memory cells on a $\sqrt{m} \times \sqrt{m}$ subgrid indexed in row major order (next to it). We show how to simulate a sub-step where each processor reads at most one value from the simulated global memory and writes at most one value from the simulated global memory. To simulate a PRAM step, execute $O(1)$ such sub-steps.

Let us begin with the read step. If processor with index i wants to read a value at cell k, it creates a tuple (i, k). Then, these tuples are sorted according to the last component. Each processor $i > 0$ inspects its tuple (i, k') and the tuple $(i-1, k'')$ of processor $i-1$. If $k' \neq k''$ or $i = 0$, then this processors reads the value at cell k by sending a message there and waiting for the value. The processors perform a segmented broadcast on the received values (with segments determined by the processors that read the same cell). Finally, we need to send the results back to the processors that initiated the read. A processor j that got tuple (i, k) from the first sorting step and received v from the segmented broadcast creates a tuple (i, v). Sort these tuples by first component, interpreted as a location in the Z-order curve of the processors (convert index i to a 2D location on the grid). Now, each processor has read a value from global memory.

The write step is similar. If processor with index i wants to write a value v to cell k, it creates a tuple (v, i, k). Then, these tuples are sorted according to the last component. Each processor $i > 0$ inspects its tuple (v', i, k') and the tuple $(v'', i-1, k'')$ of processor $i-1$. If $k' \neq k''$ or $i = 0$, then this processors sends the value v' to memory cell k'.

Each read/write step takes $O(p\sqrt{p})$ energy to sort the tuples, and $O(p)$ energy for the segmented broadcast. Since there are at most $2p$ accesses to the simulated shared memory, each taking $O(\sqrt{p}+\sqrt{m})$ energy, the total energy of one step is $O(p\sqrt{p}+p\sqrt{m})$. The depth is bottlenecked by the depth of the sorting, which is $O(\log^3 n)$ for each of the T_p sequential steps. The distance is $O(\sqrt{p}+\sqrt{m})$ for each of the T_p sequential steps. Summing over the $O(T_p)$ steps yields the result. □

VIII. Applications to Sparse Computations

Next, we demonstrate the utility of our algorithmic primitives by applying them to sparse matrix-vector multiplication (SpMV), a fundamental operation in many scientific and machine learning applications. We show how SpMV can be efficiently implemented on spatial dataflow architectures using a combination of sorting and scanning primitives, achieving the same energy, depth, and distance bounds as our optimized sorting algorithm.

Let $\boldsymbol{A}$ be an $n \times n$ sparse matrix with m non-zero entries, where $m \geq n$, and let $\boldsymbol{x}$ be a vector of size n. We want to compute the matrix-vector product $\boldsymbol{Ax}$. The matrix is stored in *coordinate format* (COO), where each non-zero entry is represented as a triple $(i, j, A_{i,j})$. Initially, the matrix $\boldsymbol{A}$ is distributed across a $\sqrt{m} \times \sqrt{m}$ subgrid of processors, with each processor holding a single arbitrary of those triples. The vector $\boldsymbol{x}$ is in a $\sqrt{n} \times \sqrt{n}$ subgrid of processors, with each processor holding one entry of the vector $\boldsymbol{x}$.

Lower Bound on Energy. To establish the optimality of our algorithm, we provide a matching lower bound on the energy required for SpMV for the case where $m = \Theta(n)$:

Lemma VIII.1. *Sparse matrix-vector multiplication requires at least $\Omega(n^{3/2})$ energy.*

Proof: This result follows from the lower bound on permutation energy (Lemma V.1). Since any permutation can be represented as a matrix-vector multiplication by constructing a corresponding permutation matrix, the energy bound for permutations applies directly to SpMV. □

PRAM Simulation Upper Bound. To establish an initial upper bound, we apply a PRAM simulation of the sparse matrix-vector multiplication algorithm. A PRAM algorithm can compute the products $A_{i,j}x_j$ in parallel. Then, it forms the sums $\sum_i A_{i,j}x_j$ using parallel sums. This CRCW PRAM algorithm runs in $O(\log n)$ time using $O(m/\log n)$ processors and $O(m)$ memory cells. By applying Lemma VII.2, this translates to $O(m^{3/2})$ energy, $O(\log^4 n)$ depth, and $O(\sqrt{m}\log n)$ distance. While this bound is energy-optimal for $m = O(n)$, we can further reduce the distance and depth by a logarithmic factor by applying our optimized primitives.

Low-Depth SpMV Algorithm. Our improved SpMV algorithm proceeds by sorting and scanning over the matrix and vector entries to compute the product with lower depth and distance compared to a PRAM simulation. Below are the key steps of the algorithm:

1) Sort the non-zero matrix entries by column index (i.e., the second coordinate of each triple $(i, j, A_{i,j})$). This step groups entries into segments corresponding to the same column.
2) For each segment, designate the first processor holding a matrix entry as the *column leader*. Each processor sends its column index to the next processor in the sequence; if the received index differs from its own or no message is received, it becomes a leader.
3) Each column leader requests the corresponding vector entry x_j and broadcasts it to all processors in its segment using a segmented broadcast implemented via a parallel scan.
4) Each processor holding a matrix entry $(i, j, A_{i,j})$ computes the partial product $A_{i,j}x_j$.
5) Sort the partial products by row index (i.e., the first coordinate of each triple), grouping products contributing to the same row of the result.
6) As with columns, identify *row leaders* by comparing row indices. Row leaders manage the summation for each segment.
7) Perform a segmented scan to sum the partial products $A_{i,j}x_j$ for each row. The row leader stores the final result $(\boldsymbol{Ax})_i$.

Theorem VIII.2. *Sparse matrix-vector multiplication takes $O(m^{3/2})$ energy, $O(\log^3 n)$ depth, and $O(\sqrt{m})$ distance.*

Proof: The cost is dominated by the sorting and scanning primitives (Theorem V.8 and Lemma IV.3). □

Discussion. We have demonstrated that SpMV can be implemented efficiently using the spatial dataflow model, achieving energy-optimal performance when the number of non-zero entries $m = O(n)$. Starting from a looser bound derived from PRAM simulation, we improved both the depth and distance metrics by carefully applying sorting and scanning primitives. These results illustrate how our algorithmic tools can be used to formulate efficient algorithms for sparse computations, with optimal energy for large, sparse matrices.

IX. Conclusion

In this paper, we presented the first energy- and distance-optimal algorithms with poly-logarithmic depth for fundamental operations, including parallel scan, rank selection, and sorting, within the Spatial Computer Model. These primitives form the foundation for more complex algorithms, such as sparse matrix-vector multiplication, demonstrating their broad applicability to spatial dataflow architectures. Our results pave the way for extending spatial architectures to a wider range of computations, particularly sparse ones. This has promising implications for fields like graph neural networks, where sparse data structures and irregular access patterns pose significant challenges that our optimized algorithms can help address.

Despite these contributions, open questions remain. Further simplification of our sorting algorithm could reduce implementation complexity, and while our sparse matrix-vector multiplication is energy-optimal when the number of non-zero elements is proportional to the number of rows, the optimal energy for denser matrices is still unknown.

In summary, this work establishes crucial foundations for spatial dataflow algorithm design through energy-optimal primitives, creating new possibilities for efficiently executing complex, sparse computations on emerging hardware platforms.

Acknowledgment

This project received funding from the European Research Council (Project PSAP, No. 101002047). This project received funding from the European Research Council under the European Union's Horizon 2020 programme GLACIATION, No. 101070141.

This work's language was refined for clarity, flow, impact, and conciseness using large language models [56], [57].

References

[1] M. Emani, V. Vishwanath, C. Adams, M. E. Papka, R. Stevens, L. Florescu, S. Jairath, W. Liu, T. Nama, A. Sujeeth, V. V. Kindratenko, and A. C. Elster, "Accelerating scientific applications with sambanova reconfigurable dataflow architecture," *Comput. Sci. Eng.*, vol. 23, no. 2, pp. 114–119, 2021. [Online]. Available: https://doi.org/10.1109/MCSE.2021.3057203

[2] S. Lie, "Cerebras architecture deep dive: First look inside the hardware/software co-design for deep learning," *IEEE Micro*, vol. 43, no. 3, pp. 18–30, 2023. [Online]. Available: https://doi.org/10.1109/MM.2023.3256384

[3] S. L. Stewart Hall, Rob Schreiber, "Training giant neural networks using weight streaming on cerebras wafer-scale clusters," Cerebras Systems, Inc., Tech. Rep., March 2023. [Online]. Available: https://f.hubspotusercontent30.net/hubfs/8968533/Virtual%20Booth%20Docs/CS%20Weight%20Streaming%20White%20Paper%20111521.pdf

[4] K. Rocki, D. V. Essendelft, I. Sharapov, R. Schreiber, M. Morrison, V. Kibardin, A. Portnoy, J. Dietiker, M. Syamlal, and M. James, "Fast stencil-code computation on a wafer-scale processor," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*, C. Cuicchi, I. Qualters, and W. T. Kramer, Eds. IEEE/ACM, 2020, p. 58. [Online]. Available: https://doi.org/10.1109/SC41405.2020.00062

[5] M. Jacquelin, M. Araya-Polo, and J. Meng, "Scalable distributed high-order stencil computations," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, Dallas, TX, USA, November 13-18, 2022*, F. Wolf, S. Shende, C. Culhane, S. R. Alam, and H. Jagode, Eds. IEEE, 2022, pp. 30:1–30:13. [Online]. Available: https://doi.org/10.1109/SC41404.2022.00035

[6] R. Sai, M. Jacquelin, F. P. Hamon, M. Araya-Polo, and R. R. Settgast, "Massively distributed finite-volume flux computation," in *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis, SC-W 2023, Denver, CO, USA, November 12-17, 2023*. ACM, 2023, pp. 1713–1720. [Online]. Available: https://doi.org/10.1145/3624062.3624252

[7] J. Tramm, B. Allen, K. Yoshii, A. Siegel, and L. Wilson, "Efficient algorithms for monte carlo particle transport on ai accelerator hardware," *Computer Physics Communications*, vol. 298, p. 109072, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010465523004174

[8] H. Ltaief, Y. Hong, L. Wilson, M. Jacquelin, M. Ravasi, and D. E. Keyes, "Scaling the "memory wall" for multi-dimensional seismic processing with algebraic compression on cerebras CS-2 systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2023, Denver, CO, USA, November 12-17, 2023*, D. Arnold, R. M. Badia, and K. M. Mohror, Eds. ACM, 2023, pp. 6:1–6:12. [Online]. Available: https://doi.org/10.1145/3581784.3627042

[9] K. Santos, S. G. Moore, T. Oppelstrup, A. Sharifian, I. Sharapov, A. P. Thompson, D. Z. Kalchev, D. Perez, R. Schreiber, S. Pakin, E. A. Leon, J. H. L. III, M. James, and S. Rajamanickam, "Breaking the molecular dynamics timescale barrier using a wafer-scale system," *CoRR*, vol. abs/2405.07898, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2405.07898

[10] C. Systems, Inc., "Cerebras systems: Achieving industry bestai performance through a systems approach," Apr. 2021. [Online]. Available: https://cerebras.net/wp-content/uploads/2021/04/Cerebras-CS-2-Whitepaper.pdf

[11] P. Luczynski, L. Gianinazzi, P. Iff, L. Wilson, D. D. Sensi, and T. Hoefler, "Near-optimal wafer-scale reduce," in *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing, HPDC 2024, Pisa, Italy, June 3-7, 2024*, P. Dazzi, G. Mencagli, D. K. Lowenthal, and R. M. Badia, Eds. ACM, 2024, pp. 334–347. [Online]. Available: https://doi.org/10.1145/3625549.3658693

[12] M. Orenes-Vera, I. Sharapov, R. Schreiber, M. Jacquelin, P. Vandermersch, and S. Chetlur, "Wafer-scale fast fourier transforms," in *Proceedings of the 37th International Conference on Supercomputing, ICS 2023, Orlando, FL, USA, June 21-23, 2023*, K. A. Gallivan, E. Gallopoulos, D. S. Nikolopoulos, and R. Beivide, Eds. ACM, 2023, pp. 180–191. [Online]. Available: https://doi.org/10.1145/3577193.3593708

[13] H. M. Aktulga, A. Buluç, S. Williams, and C. Yang, "Optimizing sparse matrix-multiple vectors multiplication for nuclear configuration interaction calculations," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium, Phoenix, AZ, USA, May 19-23, 2014*. IEEE Computer Society, 2014, pp. 1213–1222. [Online]. Available: https://doi.org/10.1109/IPDPS.2014.125

[14] M. R. Hestenes, E. Stiefel *et al.*, *Methods of conjugate gradients for solving linear systems*. NBS Washington, DC, 1952, vol. 49, no. 1.

[15] A. Ashari, N. Sedaghati, J. Eisenlohr, S. Parthasarathy, and P. Sadayappan, "Fast sparse matrix-vector multiplication on gpus for graph applications," in *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2014, New Orleans, LA, USA, November 16-21, 2014*, T. Damkroger and J. J. Dongarra, Eds. IEEE Computer Society, 2014, pp. 781–792. [Online]. Available: https://doi.org/10.1109/SC.2014.69

[16] M. Zhang, Z. Cui, M. Neumann, and Y. Chen, "An end-to-end deep learning architecture for graph classification," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, S. A. McIlraith and K. Q. Weinberger, Eds. AAAI Press, 2018, pp. 4438–4445. [Online]. Available: https://doi.org/10.1609/aaai.v32i1.11782

[17] J. H. Reif, *Synthesis of Parallel Algorithms*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[18] H. Kang, P. B. Gibbons, G. E. Blelloch, L. Dhulipala, Y. Gu, and C. McGuffey, "The processing-in-memory model," in *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, K. Agrawal and Y. Azar, Eds. ACM, 2021, pp. 295–306. [Online]. Available: https://doi.org/10.1145/3409964.3461816

[19] L. G. Valiant, "A bridging model for parallel computation," *Commun. ACM*, vol. 33, no. 8, pp. 103–111, 1990. [Online]. Available: https://doi.org/10.1145/79173.79181

[20] M. Kaufmann, S. Rajasekaran, and J. F. Sibeyn, "Matching the bisection bound for routing and sorting on the mesh," in *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '92, San Diego, CA, USA, June 29 - July 1, 1992*, 1992, pp. 31–40. [Online]. Available: https://doi.org/10.1145/140901.140905

[21] V. Leppänen and M. Penttonen, "Simulation of PRAM models on meshes," in *PARLE '94: Parallel Architectures and Languages Europe, 6th International PARLE Conference, Athens, Greece, July 4-8, 1994, Proceedings*, 1994, pp. 146–158. [Online]. Available: https://doi.org/10.1007/3-540-58184-7_97

[22] ——, "Work-optimal simulation of PRAM models on meshes," *Nord. J. Comput.*, vol. 2, no. 1, pp. 51–69, 1995.

[23] S. Goddard, S. Kumar, and J. F. Prins, "Connected components algorithms for mesh-connected parallel computers," in *Parallel Algorithms, Proceedings of a DIMACS Workshop, Brunswick, New Jersey, USA, October 17-18, 1994*, 1994, pp. 43–58. [Online]. Available: https://doi.org/10.1090/dimacs/030/03

[24] Y. Baumann, T. Ben-Nun, M. Besta, L. Gianinazzi, T. Hoefler, and P. Luczynski, "Low-depth spatial tree algorithms," in *IEEE International Parallel and Distributed Processing Symposium, IPDPS 2024, San Francisco, CA, USA, May 27-31, 2024*. IEEE, 2024, pp. 180–192. [Online]. Available: https://doi.org/10.1109/IPDPS57955.2024.00024

[25] C. D. Thompson, "The VLSI complexity of sorting," *IEEE Trans. Computers*, vol. 32, no. 12, pp. 1171–1184, 1983. [Online]. Available: https://doi.org/10.1109/TC.1983.1676178

[26] E. W. Mayr and C. G. Plaxton, "Pipelined parallel prefix computations, and sorting on a pipelined hypercube," *J. Parallel Distributed Comput.*, vol. 17, no. 4, pp. 374–380, 1993. [Online]. Available: https://doi.org/10.1006/jpdc.1993.1037

[27] C. Wu and S. Horng, "Fast and scalable selection algorithms with applications to median filtering," *IEEE Trans. Parallel Distributed Syst.*, vol. 14, no. 10, pp. 983–992, 2003. [Online]. Available: https://doi.org/10.1109/TPDS.2003.1239867

[28] M. Ajtai, J. Komlós, and E. Szemerédi, "An o(n log n) sorting network," in *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, D. S. Johnson, R. Fagin, M. L. Fredman, D. Harel, R. M. Karp, N. A. Lynch, C. H. Papadimitriou, R. L. Rivest, W. L. Ruzzo, and J. I. Seiferas, Eds. ACM, 1983, pp. 1–9. [Online]. Available: https://doi.org/10.1145/800061.808726

[29] M. Paterson, "Improved sorting networks with o(log N) depth," *Algorithmica*, vol. 5, no. 1, pp. 65–92, 1990. [Online]. Available: https://doi.org/10.1007/BF01840378

[30] K. E. Batcher, "Sorting networks and their applications," in *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference, Atlantic City, NJ, USA, 30 April - 2 May 1968*, ser. AFIPS Conference Proceedings, vol. 32. Thomson Book Company, Washington D.C., 1968, pp. 307–314. [Online]. Available: https://doi.org/10.1145/1468075.1468121

[31] R. Müller, J. Teubner, and G. Alonso, "Sorting networks on fpgas," *VLDB J.*, vol. 21, no. 1, pp. 1–23, 2012. [Online]. Available: https://doi.org/10.1007/s00778-011-0232-z

[32] R. Reischuk, "Probabilistic parallel algorithms for sorting and selection," *SIAM J. Comput.*, vol. 14, no. 2, pp. 396–409, 1985. [Online]. Available: https://doi.org/10.1137/0214030

[33] S. Saxena, P. C. P. Bhatt, and V. C. Prasad, "On parallel prefix computation," *Parallel Process. Lett.*, vol. 4, pp. 429–436, 1994. [Online]. Available: https://doi.org/10.1142/S0129626494000399

[34] G. E. Blelloch, "Scans as primitive parallel operations," *IEEE Trans. Computers*, vol. 38, no. 11, pp. 1526–1538, 1989. [Online]. Available: https://doi.org/10.1109/12.42122

[35] A. V. Gerbessiotis and C. J. Siniolakis, "Deterministic sorting and randomized median finding on the BSP model," in *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '96, Padua, Italy, June 24-26, 1996*, 1996, pp. 223–232. [Online]. Available: https://doi.org/10.1145/237502.237561

[36] ——, "A randomized sorting algorithm on the BSP model," in *11th International Parallel Processing Symposium (IPPS '97), 1-5 April 1997, Geneva, Switzerland, Proceedings*, 1997, pp. 293–297. [Online]. Available: https://doi.org/10.1109/IPPS.1997.580912

[37] P. K. Jana, B. D. Naidu, S. Kumar, M. Arora, and B. P. Sinha, "Parallel prefix computation on extended multi-mesh network," *Inf. Process. Lett.*, vol. 84, no. 6, pp. 295–303, 2002. [Online]. Available: https://doi.org/10.1016/S0020-0190(02)00317-4

[38] Y. Baumann, T. Ben-Nun, M. Besta, L. Gianinazzi, T. Hoefler, and P. Luczynski, "Low-depth spatial tree algorithms," in *IEEE International Parallel and Distributed Processing Symposium, IPDPS 2024, San Francisco, CA, USA, May 27-31, 2024*. IEEE, 2024, pp. 180–192. [Online]. Available: https://doi.org/10.1109/IPDPS57955.2024.00024

[39] S. Lakshmivarahan, S. K. Dhall, and L. L. Miller, "Parallel sorting algorithms," *Adv. Comput.*, vol. 23, pp. 295–354, 1984. [Online]. Available: https://doi.org/10.1016/S0065-2458(08)60467-2

[40] F. T. Leighton, "Introduction to parallel algorithms and architectures: Arrays, trees, hypercubes," 1991.

[41] D. Nassimi and S. Sahni, "Parallel permutation and sorting algorithms and a new generalized connection network," *J. ACM*, vol. 29, no. 3, pp. 642–667, 1982. [Online]. Available: https://doi.org/10.1145/322326.322329

[42] C. D. Thompson and H. T. Kung, "Sorting on a mesh-connected parallel computer," *Commun. ACM*, vol. 20, no. 4, pp. 263–271, 1977. [Online]. Available: https://doi.org/10.1145/359461.359481

[43] R. P. Brent, "The parallel evaluation of general arithmetic expressions," *J. ACM*, vol. 21, no. 2, pp. 201–206, 1974. [Online]. Available: https://doi.org/10.1145/321812.321815

[44] T. Hagerup and C. Rüb, "Optimal merging and sorting on the erew pram," *Inf. Process. Lett.*, vol. 33, no. 4, pp. 181–185, 1989. [Online]. Available: https://doi.org/10.1016/0020-0190(89)90138-5

[45] M. Aigner, "Parallel complexity of sorting problems," *J. Algorithms*, vol. 3, no. 1, pp. 79–88, 1982. [Online]. Available: https://doi.org/10.1016/0196-6774(82)90010-4

[46] A. Jakobsson, "Automatic cost analysis for imperative BSP programs," *Int. J. Parallel Program.*, vol. 47, no. 2, pp. 184–212, 2019. [Online]. Available: https://doi.org/10.1007/s10766-018-0562-1

[47] G. M. Morton, "A computer oriented geodetic data base and a new technique in file sequencing," 1966.

[48] C. Burstedde, J. Holke, and T. Isaac, "On the number of face-connected components of morton-type space-filling curves," *Found. Comput. Math.*, vol. 19, no. 4, pp. 843–868, 2019. [Online]. Available: https://doi.org/10.1007/s10208-018-9400-5

[49] P. Sanders and J. L. Träff, "Parallel prefix (scan) algorithms for MPI," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 13th European PVM/MPI User's Group Meeting, Bonn, Germany, September 17-20, 2006, Proceedings*, ser. Lecture Notes in Computer Science, B. Mohr, J. L. Träff, J. Worringen, and J. J. Dongarra, Eds., vol. 4192. Springer, 2006, pp. 49–57. [Online]. Available: https://doi.org/10.1007/11846802_15

[50] R. Cole, "Parallel merge sort," *SIAM J. Comput.*, vol. 17, no. 4, pp. 770–785, 1988. [Online]. Available: https://doi.org/10.1137/0217049

[51] L. G. Valiant, "Parallelism in comparison problems," *SIAM J. Comput.*, vol. 4, no. 3, pp. 348–355, 1975. [Online]. Available: https://doi.org/10.1137/0204030

[52] B. Huang and M. A. Langston, "Practical in-place merging," *Commun. ACM*, vol. 31, no. 3, pp. 348–352, 1988. [Online]. Available: https://doi.org/10.1145/42392.42403

[53] N. Deo, A. Jain, and M. Medidi, "An optimal parallel algorithm for merging using multiselection," *Inf. Process. Lett.*, vol. 50, no. 2, pp. 81–87, 1994. [Online]. Available: https://doi.org/10.1016/0020-0190(94)00009-3

[54] R. J. Hyndman and Y. Fan, "Sample quantiles in statistical packages," *The American Statistician*, vol. 50, no. 4, pp. 361–365, 1996. [Online]. Available: http://www.jstor.org/stable/2684934

[55] B. Doerr, *Probabilistic Tools for the Analysis of Randomized Optimization Heuristics*. Cham: Springer International Publishing, 2020, pp. 1–87. [Online]. Available: https://doi.org/10.1007/978-3-030-29414-4_1

[56] OpenAI, "Chatgpt 4o," 2024. [Online]. Available: https://chatgpt.com

[57] Anthropic, "Claude 3.7 sonnet," 2025. [Online]. Available: https://claude.ai/