2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS) | 979-8-3503-8711-7/24/$31.00 ©2024 IEEE | DOI: 10.1109/IPDPS57955.2024.00024

Low-Depth Spatial Tree Algorithms

Yves Baumann, Tal Ben-Nun, Maciej Besta, Lukas Gianinazzi, Torsten Hoefler, Piotr Luczynski
ETH Zurich, Switzerland

***Abstract*—Contemporary accelerator designs exhibit a high degree of spatial localization, wherein two-dimensional physical distance determines communication costs between processing elements. This situation presents considerable algorithmic challenges, particularly when managing sparse data, a pivotal component in progressing data science. The spatial computer model quantifies communication locality by weighting processor communication costs by distance, introducing a term named *energy*. Moreover, it integrates *depth*, a widely-utilized metric, to promote high parallelism. We propose and analyze a framework for efficient spatial tree algorithms within the spatial computer model. Our primary method constructs a spatial tree layout that optimizes the locality of the neighbors in the compute grid. This approach thereby enables locality-optimized messaging within the tree. Our layout achieves a *polynomial* factor improvement in energy compared to utilizing a PRAM approach. Using this layout, we develop energy-efficient treefix sum and lowest common ancestor algorithms, which are both fundamental building blocks for other graph algorithms. With high probability, our algorithms exhibit near-linear energy and poly-logarithmic depth. Our contributions augment a growing body of work demonstrating that computations can have both high spatial locality *and* low depth. Moreover, our work constitutes an advancement in the spatial layout of irregular and sparse computations.**

***Index Terms*—Graph algorithms, Parallel algorithms, Trees, Routing and layout, Distributed data structures, Dataflow architectures, Adaptable architectures**

I. Introduction

Spatial computing architectures have emerged as significant platforms in optimizing energy use and enhancing throughput, often outperforming traditional CPUs and GPUs in specific applications. Notable among these architectures are the Cerebras Wafer-Scale Engine (WSE) [17], [45], [48], [50], [53] and Coarse-Grained Reconfigurable Arrays [13], [18], [43], [47], [51]. These platforms employ highly localized memory systems, wherein each processor is allocated its own segment of fast memory, and feature a communication network where the cost between processors is dictated by their relative proximity. This paradigm proves particularly advantageous in applications such as matrix computations [36] and stencils [45]. Yet, it also introduces programming and algorithmic challenges, especially when computing on sparse data—a widespread and growing challenge in domains like machine learning [8], [29] and deep learning [5], [7], [23], [54].

The emerging focus on sparsity and its resulting irregular access patterns underscore a critical challenge for spatial architectures, prompting the exploration and development of innovative algorithmic approaches to address these complexities. This work introduces the first *spatial tree algorithms*, presenting a noteworthy advancement in optimizing irregular access patterns for spatial architectures. Trees are fundamental graph structures, and tree algorithms often underpin more complex graph algorithms [27]. They offer wide applicability and utility across various scientific and technological domains.

The optimization of tree layouts has direct applications in diverse fields such as machine learning and computational biology. In the former, models like decision trees [29] and random forests [8] can realize enhanced performance through spatial locality. Meanwhile, in computational biology, the study of phylogenetic trees [42], [52] provides vital insights into evolutionary pathways [41] by extensively analyzing tree structures. This highlights the broad relevance and widespread applicability of optimized tree layouts and algorithms in various scientific domains.

A. Methodology

We analyze our results in a model that captures the main characteristics of *spatial* architectures [20]. The model abstracts certain lower-level details such as the interconnection network, but emphasizes the main drivers of performance. Let us consider the Cerebras WSE-2 as an example [48]. It has 850,000 cores with around 50KB of fast local memory per core. Each processor can send and receive a 32-bit message per cycle. The time to reach the destination has an initial latency of 2 cycles and is then proportional to the distance on the chip. Hence, it is important to minimize (1) the distance traveled and (2) the number of hops in the communication chains. The *spatial computer model* [20] encourages these qualities and enables productive algorithm design by focusing on the main performance factors.

The model considers a two-dimensional grid of processors. Each processor $p_{i,j}$ has a position (i, j) and a local memory containing a *constant* number of words. In each round, a processor can send and receive a constant number of messages and then perform a constant number of arithmetic operations on its local memory. The *energy* cost of a processor $p_{i,j}$ sending a message to a processor $p_{x,y}$ is equal to $|x-i|+|y-j|$, i.e., the Manhattan distance. The energy of an algorithm is the sum of the energy of all its messages. Essentially, the energy is the distance-weighted communication volume. The *depth* of a spatial computation is the longest chain of messages depending on each other.

Reducing the *energy* leads to more spatially local algorithms that minimize the distance traveled; this, in turn, enhances the efficiency and sustainability of large-scale applications. By minimizing the *depth* of the model, shorter communication chains can be achieved, enabling algorithms to effectively leverage the vast number of compute cores available in modern spatial architectures. Although the model is formulated for processors with asymptotically constant-size memory, its results are applicable in settings where processors have larger memories of size M [20]. Hence, by focusing on bounded memory, we obtain results for general memory sizes.

B. Limitations of State-of-the-Art

None of the existing approaches capture the unique requirements of having high spatial locality and low depth:

PRAM: There is a rich literature of low-depth and work-optimal algorithms on trees [6], [33], [34], [37], [46]. However, the PRAM model does not have any notion of spatial locality and consequently its algorithms exhibit suboptimal energy. However, we will show how to adapt principles of parallel algorithm design. When combined with our specialized data layouts, this adaptation achieves low-depth *and* low-energy.

Parallel External Memory: Arge et al. [3] presented parallel external memory (PEM) algorithms for list ranking, expression tree evaluation, and lowest common ancestors. In PEM [14], it is crucial to subdivide the working set into chunks that fit into the processors' local fast memory. This issue is even more pronounced in the spatial setting, as the processors have only constant-sized memory. In contrast to the PEM setting, obtaining near-linear energy bounds precludes sorting the data.

CGM: Dehne et al. [16] considered lowest common ancestors and expression tree evaluation in the Coarse-Grained Multicomputer model (CGM). In CGM [10], [15], the number of processors is smaller than the input size by a polynomial factor. Hence, CGM work focuses on reducing communication at the expense of parallelism. In contrast, the spatial computer considers the setting where the processor count is on a similar order of magnitude as the input size, thereby demanding maximized parallelism.

FCN: Leighton [30] proposed graph algorithms for fixed-connection networks (FCN), where the communication network has a fixed topology, for example, a mesh of trees [32]. For dense graphs, their approach embeds the graph in a fixed network of *quadratic size* in the number of nodes. For sparse graphs, their approach relies on PRAM simulation. In contrast, the spatial computer is network-oblivious in that the energy and depth terms are meaningful for a variety of topologies [20]. Moreover, we avoid costly PRAM simulations.

VLSI Complexity: The problem of embedding graphs onto a two-dimensional plane has received much attention in VLSI Complexity [31], [35]. Similar to our setting, reducing wire lengths is a central goal in VLSI. However, our model differs significantly in both its goals and approaches. VLSI concentrates on hardware wiring, which is inherently static, focusing on the physical layout of connections. Our model is tailored for algorithm design, emphasizing dynamic message passing rather than static wire configuration. This fundamental difference in objectives leads to distinct methodologies. For instance, unlike VLSI, our model does not prioritize crossing-free wire placement. Instead, our emphasis is on optimizing the efficiency of message passing. Consequently, we use proxy metrics such as energy and depth to evaluate the routing of messages. These metrics are versatile, applicable across a variety of communication networks and routing policies.

C. Key Insights and Contributions

Tree algorithms are nontrivial to make spatially local because traditional algorithms exhibit irregular access patterns. Consider, for instance, a fundamental tree kernel that sends a message from each vertex to its children. Although this *local messaging* kernel has constant depth, its energy depends on the layout of the tree on the processor grid. The average distance between neighbors varies dramatically based on vertex and edge placements: A suboptimal layout can yield a communication distance between vertices that deviates polynomially from the optimum.

We present a locality-optimized tree layout to obtain an efficient message-passing kernel on trees. The layout has two main steps: First, we map the tree onto a linear order based on the sizes of the subtrees using an order we call *light-first*. Second, the linear order is mapped to the 2D grid with a space-filling curve [4]. Our analysis shows that this layout leads to a message-passing kernel with *linear energy*, which is optimal up to constant factors. This is achieved while preserving the low depth characteristic of the local messaging kernel. For this result, we bound the distances in several space-filling curves. Trees of unbounded degree need special care because of the limitations imposed by having $O(1)$ memory per processor. Nevertheless, we attain the same energy bounds on general trees for the following useful communication patterns: when children uniformly receive a message from a parent and when a parent receives reduced messages from its children.

We demonstrate how to utilize this locality-optimal layout to implement treefix sums and lowest common ancestors (LCA), two pivotal tree operations, with near-linear energy and poly-logarithmic depth. These operations are subroutines for other graph algorithms, such as the computation of minimum cuts [27]. To formulate efficient algorithms within the local messaging framework, specific adaptations are necessary. For treefix sums, this requires adapting low-depth techniques to the spatial setting. For LCA, this requires the design of a new approach based on a cover of the tree with carefully chosen subtrees. This cover uses our efficient treefix sum algorithm. We present randomized (Las Vegas) algorithms for treefix sum and batched lowest common ancestors that take $O(n \log n)$ energy and $O(\log^2 n)$ depth with high probability on a tree with n vertices. If the tree has bounded degree, the depth of the treefix sum algorithm is $O(\log n)$. In comparison, the simulation of a work-optimal PRAM algorithm would take $\Theta(n^{\frac{3}{2}})$ energy and $O(\log^4 n)$ depth.

D. Limitations of the Proposed Approach

To achieve the best performance, we must store the trees in our efficient layout before executing the tree algorithms. We can mitigate this limitation and amortize the layout costs when using the same tree across multiple iterations, a common scenario in machine learning and phylogenetic applications.

TABLE I
TREE AND COST MODEL NOTATION

Symbol	Description
$T, \hat{T}, T' \dots$	A rooted tree
n	Number of vertices in T = number of processors
deg(v)	degree of vertex v
Δ	Maximum vertex degree in the tree T
$s(v)$	number of descendants of v
$E(n), E_d, \dots$	Energy, i.e., total Manhatten distance traversed

II. BACKGROUND

We proceed to give the background needed for our technical discussion, including on our model of computation and space-filing curves, which are a crucial tool in our constructions.

A. Model of Computation

In spatial architectures, the physical distance has a direct impact on the cost of communication [24], [25]. Longer distances increase latency, indicate potential congestion, and the active energy is proportional to total distance traveled. Because of manufacturing constraints, many commercially available chips, including the Cerebras WS-2, use mesh-like interconnects, exacerbating these issues [26].

The spatial computer model provides a suitable abstraction for such architectures [20]. It considers a $\sqrt{n} \times \sqrt{n}$ grid of processors with constant-sized memory each. We consider a computation as a directed acyclic graph, where the vertices correspond to computation at a given processor in the grid and the edges correspond to communication. Each edge has a weight corresponding to the Manhattan distance between the processors of the computations it connects. Processors may compute independently, but a computation waits for all incoming messages on which it depends. The largest number of messages in a chain of dependent messages is the **depth** of the computation. The depth provides a measure of how often the computation switches between communication and computation steps, which can be costly. A low-depth algorithm can be scheduled in a small number of parallel steps [9]. The total weight of the communicating edges is the **energy** of the computation. The energy provides an estimate on the cost of routing the messages, as a larger energy corresponds to sending messages for more network hops. Note that the energy is bounded by the work, i.e., the total number of operations, because the constant-sized memory implies that an algorithm performs $O(1)$ computations between sending two messages.

We employ the following foundational spatial algorithms. The **Broadcast** of a scalar value from one processor to all other processors in the grid takes $O(n)$ energy and $O(\log n)$ depth. A **reduce** operation computes the sum of a set of n values and takes $O(n)$ energy and $O(\log n)$ depth. An *all-reduce*, which is a reduce followed by a broadcast, has the same energy and depth bounds [20]. Parallel **prefix sum** takes $O(n)$ energy and poly-logarithmic depth. **Sorting** n numbers takes $\Theta(n^{\frac{3}{2}})$ energy and poly-logarithmic depth, which matches the $\Omega(n^{\frac{3}{2}})$ energy lower bound of a global permutation on a $\sqrt{n} \times \sqrt{n}$ grid [20].

The spatial computer can perform a **PRAM simulation** of any shared-memory parallel algorithm. If an algorithm uses p processors, m memory cells, and T_p time steps, it takes $O(p(\sqrt{p} + \sqrt{m})T_p)$ energy with poly-logarithmic depth overhead. While typically resulting in sub-optimal algorithms, this simulation easily provides upper bounds.

A **Las Vegas** algorithm always produces the correct result with a probabilistic guarantee on its costs. For all our Las Vegas algorithms, the bounds hold with high probability. This means that given any constant $c > 0$, the probability that the bound holds is greater than 1-$\frac{1}{n^c}$.

B. Space-Filling Curves

A key element used in our tree layout schemes are *space-filling curves* [4]. A discrete space-filling curve maps a subset of the natural numbers onto a subset of the 2D grid.

We define the **Hilbert curve** [4], [22] of k-th order for a grid of size $4^{\frac{k}{2}} \times 4^{\frac{k}{2}}$ by dividing it into four subgrids, each with a Hilbert curve of $(k-1)$-th order, flipping the two lower curves across the diagonals and then connecting the subgrids. A Hilbert curve of 0-th order is a 1×1 square. If a curve has order k, we assume it is defined on a $4^{\frac{k}{2}} \times 4^{\frac{k}{2}}$ subgrid. See Figure 1 (right) for an example.

The **Z-order curve** [4], [21] is defined by dividing the grid into four quadrants. Visit the four quadrants recursively in the following order: upper left; upper right; lower left; lower right. See Figure 2 for an example. Note that the Z-order curve is not distance-bound. However, we will show that it provides the same spatial locality properties as the other curves.

C. Trees

Throughout, we consider a rooted tree $T = (V, E)$ with $n = |V|$ vertices. The degree of a vertex v counting its parent and children is deg(v) and the maximum degree in the tree T is Δ. The descendants of a vertex v contains v, its children, and recursively the descendants of its children. A vertex u is an ancestor of a vertex v if v is a descendant of u. Table I provides a summary of our notation.

III. SPATIAL TREE LAYOUTS

We show how to embed a tree into the two-dimensional grid so neighbors can communicate with *constant* average energy. This embedding is the key tool for our tree algorithms in Sections V and VI. The main idea is as follows: First, compute a linear order of the vertices of the tree. Second, embed the linear order using a space-filling curve.

The challenge lies in finding the correct linear order and proving that the linear order indeed results in a low-energy layout. Naïve solutions, such as using breadth-first order or depth-first order, do not yield the desired results. In particular, a perfect binary tree will have a breadth-first layout where the average distance between neighbors is $\Omega(\sqrt{n})$. For depth-first order, a tree formed by adding an additional vertex as a child

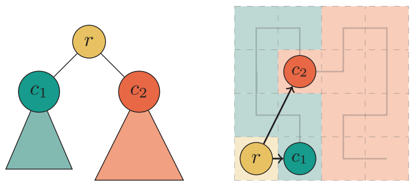

Fig. 1. Part of a tree stored in Hilbert-light-first order. The smaller subtree is stored first, then the larger subtree follows. Both subtrees are stored similarly recursively. Mapping this linear order onto the Hilbert curve yields an energy-efficient two-dimensional layout.

of each vertex in a path graph provides similarly poor results. Hence, it is crucial to optimize the tree layout.

We will first consider trees of bounded degree and then generalize the results to arbitrary degrees in Section III-D.

A. Light-First Order

We define a tree layout, called **light-first order**, and show that trees stored in this order allow for energy-efficient messaging. We can define light-first order for any space-filling curve. Let v be some vertex with children $\{c_1, ..., c_{\deg(v)}\}$. Let $s(c_i)$ be the size of the subtree rooted at the vertex c_i. Let us now assume that a vertex v is stored in a processor p_v and that the children are indexed in increasing order of their subtree size. We say that a vertex v has a *neighborhood* stored in *light-first order* if each c_i is stored in $(1 + p_v + \sum_{j=1}^{i-1} s(c_j))$-th position according to the space-filling curve. Note that a vertex with no children has a neighborhood in light-first order by default. We say that a tree T is in *light-first order* if every vertex v in T has a neighborhood stored in light-first order. We can say *K-light-first order* to specify that it is defined for the space-filling curve K. See Figure 1 for an example of a tree stored in Hilbert-light-first order.

B. Distance-Bound Curves

Consider a tree T with a degree bounded by some constant Δ. Let that tree be stored in light-first order for some space-filling curve. For a space-filling curve, we say that a processor is i-th if its location on the grid corresponds to the i-th element in that curve's order. We say that a curve is **distance-bound** if for all natural numbers i and j, sending a message from the i-th processor to the $(i+j)$-th processor takes $O(\sqrt{j})$ energy. We will get different constant factors depending on the type of distance-bound curve. That is, sending the message takes $c\sqrt{j} + o(\sqrt{j})$ energy for some constant α. Examples include the Hilbert curve with $\alpha = 3$ [39], the Peano curve with $\alpha = \sqrt{10 + 2/3}$ [4], the $\beta\Omega$ curve with $\alpha = 3$ [4], [21], or the H-index where $\alpha = 2\sqrt{2}$ [4], [38]. There are also other space-filling curves for which the constant α has been proven [4], [38], [39]. The definition also applies to space-filling curves not defined for a square grid, e.g., the Gosper Flowsnake. However, in a practical sense, non-square curves may be rather inefficient.

We say that a light-first order is **energy-bound** if, for any tree stored in this order, the total energy of each vertex sending a message to all its children is $O(n)$. In this section, we prove that any light-first order for a distance-bound space-filling curve is also *energy-bound*.

Let $\text{dist}(i, j)$ be the energy cost of sending a message from the i-th position to the j-th position. If the curve is distance-bound, $\text{dist}(i, j) \leq c \cdot \sqrt{|j - i|}$ for some $c \in O(1)$. For simplicity, we assume that all vertices have exactly Δ children. If a vertex has less than Δ children, we can introduce empty subtrees to make it so.

Let $E(n)$ be the energy cost of sending the messages in a subtree of size n. It is bounded as follows:

Lemma 1. $E(n) \leq (\sum_{i=1}^{\Delta} E(s(c_i)) + (\Delta - i) \cdot c\sqrt{s(c_i)}) + \Delta \cdot c\sqrt{2}$.

Proof: We can express $E(n)$ recursively:

$$E(n) \leq \sum_{i=1}^{\Delta} E(s(c_i)) + \text{dist}\left(p_r, p_r + 1 + \left(\sum_{j=1}^{i-1} s(c_j)\right)\right) \ .$$

From the definition of distance-bound, $\text{dist}(p_r, p_r + 1 + \sum_{j=1}^{i-1} s(c_j)) \leq c\sqrt{2 + \sum_{j=1}^{i-1} s(c_j)}$, which implies the result. ■

We need the following lemma to prove the main theorem.

Lemma 2. *If* $s(c_i) \leq s(c_j), \forall j \geq i$ *and* $\sum_{i=1}^{d} s(c_i) = n$*, the equation* $\sum_{i=1}^{\Delta} (\Delta + i) \cdot c\sqrt{s(c_i)}$ *is minimized for* $s(c_d) = n$.

For the proof of Lemma 2, see Appendix A.

Theorem 1. *Light-first order defined on a distance-bound space-filling curve is energy-bound.*

Proof: We prove by strong induction on the number n of elements in the tree that

$$E(n) \leq \Delta \cdot 8c \cdot n - \Delta \cdot 2c\sqrt{n} - 2c\sqrt{2}, \forall n \in \mathbb{N} \ .$$

The base case for $n = 1$ follows from $c \geq 1$. The inductive step goes as follows:

$$\begin{aligned} E(n+1) &\leq \Delta \cdot c\sqrt{2} + \sum_{i=1}^{\Delta} E(s(c_i)) + (\Delta - i) \cdot c\sqrt{s(c_i)} \\ &\overset{\text{(I.H.)}}{\leq} \Delta \cdot c\sqrt{2} + \sum_{i=1}^{\Delta} \Delta \cdot 8c \cdot s(c_i) - \Delta \cdot 2c\sqrt{s(c_i)} \\ &\quad - 2c\sqrt{2} + (\Delta - i) \cdot c\sqrt{s(c_i)} \\ &\leq \Delta \cdot 8c \cdot n - \Delta \cdot c\sqrt{2} - \sum_{i=1}^{\Delta} (\Delta + i) \cdot c\sqrt{s(c_i)} \ . \end{aligned}$$

By Lemma 2, the expression is minimized for $s(c_\Delta) = n$. Hence,

$$E(n+1) \leq \Delta \cdot 8c(n+1) - \Delta \cdot 2c\sqrt{2} - \Delta \cdot 2c\sqrt{n+1}$$

As Δ is constant, we conclude that $E(n) \in O(n)$. ■

Note that this implies that the reverse operation in which each vertex sends a message to its parent also takes $O(n)$ energy. Moreover, observe that all the space-filling curves apart from the Z-order mentioned in Section II satisfy the distance-bound property.

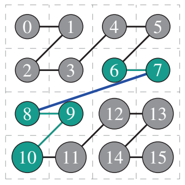

Fig. 2. 16 elements stored in Z-order. Given $i = 6$ and $j = 10$ the longest diagonal would be the blue one. The x-length of the diagonal would be 3 and the y-length 1. Moreover, we have that $E_d(6, 10) = 4$.

C. Z-Light-First Order

Proving that Z-light-first-order is energy-bound is more involved since, in contrast to the other space-filling curves, it is not distance-bound. This is because the distance between two arbitrary processors is not bounded except by the side lengths of the computational grid. Nevertheless, we will prove the necessary bound on the communication distances by showing that Z-order, in essence, consists of multiple distance-bound curves connected by 'diagonals'. The proof relies on showing that each such diagonal is used a bounded number of times.

Theorem 2. *Z-light-first order is energy-bound.*

If for a space-filling curve for any k every 4^k consecutive elements are stored in a subgrid of size at most $2 \cdot 4^{\frac{k}{2}} \times 2 \cdot 4^{\frac{k}{2}}$, we call the curve *aligned*. An example of such a curve is the Hilbert curve. We define $E_b(i, j)$ as the largest energy of sending a message from i-th to j-th processor stored on an aligned curve.

We define $E_d(i, j)$ as the Manhattan distance of the longest diagonal between i and j. Refer to Figure 2 for an example. Observe that the Manhattan length of a diagonal is always one larger than the side length of the smallest power-of-two-aligned square subgrid. For simplicity of notation, we define the *length* of a diagonal to be one less than its Manhattan distance.

Lemma 3. *The energy $E(i, j)$ of sending a message from i-th to j-th processor stored in z-light-first order is at most $E_b(i, j) + E_d(i, j)$.*

Proof: In Z-order, every 4^k consecutive and aligned elements are stored in a subgrid of size $4^{\frac{k}{2}} \times 4^{\frac{k}{2}}$ [4]. If they are not aligned they would be stored in at most two subgrids of size $4^{\frac{k}{2}} \times 4^{\frac{k}{2}}$ connected by some diagonal and could therefore be far apart. The energy to send a message from i to j is then bounded by the energy of sending a message as if those two subgrids were next to each other (all elements stored in $2 \cdot 4^{\frac{k}{2}} \times 2 \cdot 4^{\frac{k}{2}}$) plus the energy of sending a message across a diagonal. This is bounded by $E_b(i, j) + E_d(i, j)$. ■

Lemma 4. *An aligned curve is distance bound.*

Proof: The distance between the i-th and $(i + j)$-th processor is at most $8\sqrt{j}$. Hence, the curve is distance-bound. ■

Therefore, we conclude by Lemma 4 and Theorem 1:

Corollary 1. *The total E_b cost of each vertex sending a message to all its children is in $O(n)$.*

It remains to prove an $O(n)$ bound for the energy of the diagonals E_d. Given a diagonal D of length k, we want to bound how many times it is the longest diagonal when each vertex u sends a message to its child c_i. We say that D is the *longest diagonal* in a subtree rooted at some vertex v if and only if D is the longest diagonal when v sends a message to one of its children.

Lemma 5. *Consider a vertex v with children $c_1, ..., c_k$ and let D be the longest diagonal when v sends a message to one of its children. Then D can be the longest diagonal in at most one of the children's subtrees, which has size at most $\frac{1}{2} \cdot s(u)$.*

Proof: Note that D can only be part of one of the subtrees $c_1, ..., c_{k-1}$: If D were in the subtree c_k, then v would not send a message to any of its children over D. Because only the last subtree may have a size greater than $\frac{1}{2} \cdot s(u)$, but it is not included, D will be the longest diagonal in a subtree of size at most $\frac{1}{2} \cdot s(u)$. ■

Lemma 6. *Let D be some diagonal of length $k = 2^c$, for some $c \in \mathbb{N}^+$. D is the longest diagonal at most $\Delta \cdot \lceil \log_2(4 \cdot k^2) \rceil$ times.*

Proof: Observe that for every $4 \cdot k^2$ elements, the longest diagonal has a length of at least $2 \cdot k$. Moreover, diagonals can only have lengths that are powers of 2. Hence, if a diagonal of length k is the longest diagonal when i sends a message to j, then $j - i < 2 \cdot k$.

Let T' be the smallest subtree such that for all vertices not in the subtree, D is never the longest diagonal. It follows that T' has size at most $4 \cdot k^2$. Now, observe that every time D is the longest diagonal in some subtree, the size of the next subtree in which D can be the longest diagonal is at least halved, by Lemma 5. This means that there are at most $\lceil \log_2(4 \cdot k^2) \rceil$ subtrees in which D is the longest diagonal. Since each vertex sends at most Δ messages, D is the longest diagonal at most $\Delta \cdot \lceil \log_2(4 \cdot k^2) \rceil$ times. ■

Lemma 7. *The total E_d cost of each vertex sending a message to all its children is in $O(n)$.*

Proof: Consider n processors on a $\sqrt{n} \times \sqrt{n}$ grid. For each $i \in \{0, ..., \lceil \log_2(\sqrt{n}) \rceil\}$ we have less than $2 \cdot 4^{\lceil \log_2(\sqrt{n}) \rceil - i}$ diagonals of length 2^{i+1}, by Lemma 6. We now bound the diagonal energy:

$$E_d \leq \sum_{i=0}^{\lceil \log_2 \sqrt{n} \rceil} 2 \cdot 4^{\lceil \log_2(\sqrt{n}) \rceil - i} \cdot \Delta \cdot 2^{i+1} \cdot \lceil \log_2(4 \cdot 2^{2i+2}) \rceil$$

$$\leq 16 \cdot \Delta \sum_{i=0}^{\log_2 \sqrt{n}+1} \frac{n}{2^i} \cdot (4i + 5)$$

$$\leq 8 \cdot \Delta \sum_{i=1}^{\infty} \frac{n}{2^i} \cdot (4i + 1) \leq 8 \cdot \Delta \cdot n(8 + 1) \in O(n)$$

Theorem 2 now follows from Lemmas 3, 1 and 7. ■

D. Unbounded Degree Trees

So far, we have relied on a restriction to bounded degree trees. When the degree is unbounded, we cannot store all the neighbors of a vertex in one processor because it has $O(1)$ memory. Moreover, energy bounds would deteriorate with the degree even if we allowed that. Instead, we show how to transform the tree of unbounded degree into a tree of bounded degree that can simulate the original tree with *linear energy and using constant space per processor*.

For this to work out, we need to restrict the messaging operations. These operations suffice to implement the two tree algorithms we consider, treefix sum and LCA. We allow so-called *local messaging*, which includes two operations:

- *Local broadcast:* Each vertex sends a single message to all its children. Note that the message each child receives has to be the same.
- *Local reduce:* Each parent receives a reduction of the messages from its children. The reduction can use any associative function, such as sum or maximum.

We show that for such operations, the total energy cost is $O(n)$ and the depth is $O(\log n)$. In the following, light-first order refers to some arbitrary fixed energy-bound light-first order.

We first show how to conceptually transform an unbounded degree tree T into a tree $\hat{T}$ with a bounded degree. $\hat{T}$ defines the order in which the messages are sent and propagated. We then show that assuming that T is in light-first order, $\hat{T}$ will also be in light-first order, i.e., we do not have to change the processors in which vertices are stored. Finally, we show how to implement the construction of $\hat{T}$, given T.

For a vertex v, we initialize a set $C(v)$ of *current children* that initially contains all of the children of v and a set $A(v)$ of *appended children* that is initially empty.

Let us now assume that we are given a vertex v with the set of current children $C(v) = \{c_1, ..., c_d\}$ and the set of appended children $A(v) = \{a_1, ..., a_{d'}\}$. We define TRANSFORM(V):

1) Set $A(c_1)$ to $\{c_2, ..., c_{\lfloor \frac{d}{2} \rfloor}\}$ and $A(c_{\lfloor \frac{d}{2} \rfloor+1})$ to $\{c_{\lfloor \frac{d}{2} \rfloor+2}, ..., c_d\}$. Update the set of current children $C(v)$ to be $\{c_1, c_{\lfloor \frac{d}{2} \rfloor+1}\}$.
2) Set $A(a_1)$ to $\{a_2, ..., a_{\lfloor \frac{d'}{2} \rfloor}\}$ and $A(a_{\lfloor \frac{d'}{2} \rfloor+1})$ to$\{a_{\lfloor \frac{d'}{2} \rfloor+2}, ..., a_{d'}\}$. Update the set of appended children $A(v)$ to $\{a_1, a_{\lfloor \frac{d'}{2} \rfloor+1}\}$.
3) Transform the vertices $c_1, c_{\lfloor \frac{d}{2} \rfloor+1}, a_1$ and $a_{\lfloor \frac{d'}{2} \rfloor+1}$.

See Figure 3 for an example of the TRANSFORM. We define a *virtual* tree as a tree where the children of a vertex v are a union of $C(v)$ and $A(v)$. Now let $\hat{T}$ be a virtual tree which at first is the same as T. We then use the algorithm defined above to conceptually transform $\hat{T}$ starting from the root. The final result is a binary virtual tree $\hat{T}$ which defines the messaging order.

We now show that after the transformation, the children of every vertex are still sorted by the size of their subtree. This means that the physical placement of the vertices does not need to change:

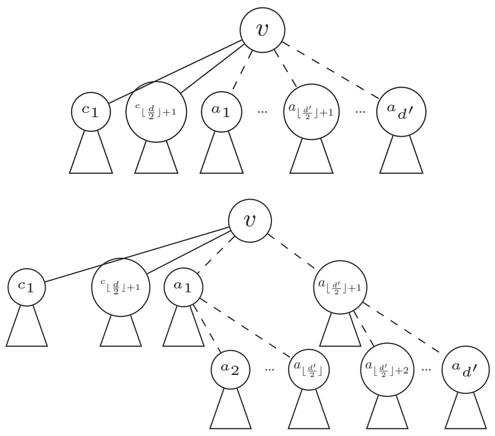
v
c_1
$c_{\lfloor \frac{d}{2} \rfloor+1}$
a_1
...
$a_{\lfloor \frac{d'}{2} \rfloor+1}$
...
$a_{d'}$
v
c_1
$c_{\lfloor \frac{d}{2} \rfloor+1}$
a_1
$a_{\lfloor \frac{d'}{2} \rfloor+1}$
a_2
...
$a_{\lfloor \frac{d'}{2} \rfloor}$
$a_{\lfloor \frac{d'}{2} \rfloor+2}$
...
$a_{d'}$

Fig. 3. Example of a virtual tree rooted at a vertex v before and after the second step of TRANSFORM. The vertex v has degree 4 after the transform. Solid lines connect to the current children, whereas dashed lines connect to the appended children.

Lemma 8. *Let $\hat{T}$ be a tree resulting from transformation of the tree T. If T is in light-first order and the vertices in $\hat{T}$ are stored in the same order as in T, then $\hat{T}$ is also in light-first order.*

Proof: We assume that the vertices in $\hat{T}$ are stored in the same processors as in T. We prove the statement by induction. $\hat{T}$ is in light-first order at the beginning since it is the same as T. Let T be the state of $\hat{T}$ before some step of the transformation and T' the state of $\hat{T}$ after that step. We now show that assuming a tree T is in light-first order, the tree T' is also in light-first order. We consider some vertex v and first look at step 1. Let $C(v) = \{c_1, ..., c_d\}$ and $A(v) = \{a_1, ..., a_{d'}\}$. We execute step 1 of TRANSFORM(V). The correct positions of the vertex v and the vertices outside of the subtree rooted at v stay the same since the size of the subtree rooted at v does not change. Notice that the children of v stay sorted by size, since $|A(c_1)| \leq |A(c_{\lfloor \frac{d}{2} \rfloor+1})|$ and $s(c_i) \leq s(c_j)$ for all $i \leq j$ before the operation. It therefore follows that $s(c_1) \leq s(c_{\lfloor \frac{d}{2} \rfloor+1})$ after the operation. Note that step 2 can be analyzed analogously. In particular, if $A(v)$ is not empty, then $s(c_{\lfloor \frac{d}{2} \rfloor+1}) \leq s(a_1) \leq s(a_{\lfloor \frac{d'}{2} \rfloor+1})$ after the operation. Since the vertices remain sorted by their subtree size, the positions in which the subtrees need to be stored stay the same. As the algorithm only executes those steps, the virtual tree $\hat{T}$ is always in light-first order. ■

We now assume that we are given the tree $\hat{T}$ as input and it is in light-first order. Note that $\hat{T}$ has degree bounded by 4. Let $v \in B$ with $C(v) = \{c_1, c_2\}$ and $A(v) = \{a_1, a_2\}$. For each vertex v we define the local broadcast:

1) Send the message to its children c_1 and c_2.
2) Unless v is the root, wait until v receives a message from the parent and then propagate it to a_1 and a_2.

The local reduce can be defined in a similar way using reduction.

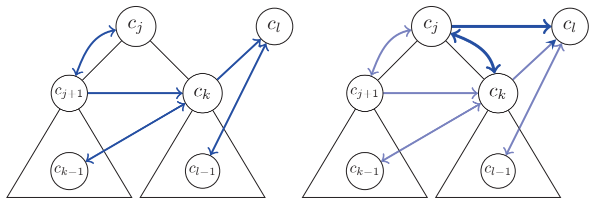

Fig. 4. Example of a procedure for passing the references. The black edges represent the tree $\hat{T}$. The directed blue edges represent each vertex's references before and after the operation.

Lemma 9. *Local messaging in a virtual tree $\hat{T}$ stored in energy-bound light-first order takes $O(n)$ energy and $O(\log n)$ depth.*

Proof: Since the tree has constant degree and is in energy-bound light-first order it follows from Theorem 1 that the local messaging is done with $O(n)$ energy. The depth follows from the fact that if we look at a single vertex we are doubling the number of vertices that distribute its message every round. After $O(\log n)$ rounds all the messages will therefore be sent. ■

Note that we have shown this assuming that the virtual tree $\hat{T}$ is given as input. We now show how to construct $\hat{T}$ given some unbounded degree tree T as input. Note that every processor has $O(1)$ memory, so a vertex cannot store a reference to all its children. Next, we show how each vertex can compute its parent and children in the virtual tree $\hat{T}$. Each vertex has at most two current children, so it can pass their references to its parent. Next, we focus on the appended children.

Let $v \in U$ with $C(v) = \{c_1, ..., c_d\}$. We assume that each $c_j \in C(v)$ knows its index j and the degree d of its parent. Moreover, c_j has a reference to its left sibling c_{j-1}, its right sibling c_{j+1}, and its parent v. Assume that $A(c_j) = \{c_{j+1}, c_k\}$ for some k and that c_j's parent is c_p. Observe that c_k is the right sibling (in the list of children of v) of the rightmost descendant of c_j. The procedure works bottom-up in the subtree rooted at the appended children and maintains this right sibling of the rightmost descendant of the current subtree. Assume that c_{j+1} is either a leaf or has finished local messaging with its children. Then, c_{j+1} sends c_j the reference to c_k. If c_{j+1} is a leaf then $c_k = c_{j+2}$ hence it already has that reference. Otherwise, c_{k-1} is in the subtree rooted at c_{j+1}. Therefore, c_{j+1} would have received that reference. Next, c_j sends a message to c_k which responds with the reference to c_l, where c_l is the only vertex that is not in the subtree rooted at c_k where c_{l-1} is in that subtree. See Figure 4 for an illustration. To compute the parent c_p, observe that if c_j is a left child of its parent, then $c_p = c_{j-1}$. Otherwise, c_j waits until it gets the reference to the parent from its sibling. Note that finding the references to the children and parents is enough to build the virtual $\hat{T}$ because, as proven in Lemma 8, we do not have to change the position of the vertices.

Theorem 3. *Local messaging in a tree T stored in energy-bound light-first order takes $O(n)$ energy and $O(\log n)$ depth.*

Proof: We construct $\hat{T}$ as shown above. This requires $O(\log n)$ depth and $O(1)$ memory for each vertex to compute its children and its parent. It then takes $O(n)$ energy and $O(\log n)$ depth to pass the references, which follows from Lemma 9. Once we have constructed $\hat{T}$, every local messaging operation takes $O(n)$ energy and $O(\log n)$ depth. ■

IV. Creating the Layout

We have shown how energy-bound light-first order is a storage format that leads to linear-energy messaging within the tree. It remains to show how a light-first order tree layout can be efficiently computed. Our goal is to obtain $O(n^{\frac{3}{2}})$ energy, which matches the permutation lower bound, and $O(\log n)$ depth. The main challenge lies in reducing the depth: Simulating a work-optimal PRAM algorithm would give $\Theta(n^{\frac{3}{2}})$ energy and $\Theta(\log^4 n)$ depth [20].

Theorem 4. *Light-first order takes $O(n^{\frac{3}{2}})$ energy and $O(\log n)$ depth to compute, with high probability.*

The approach is as follows:

1) Compute the size of each subtree via an Euler Tour [49]:
   a) Drop all but the first and last occurrence of a vertex using a parallel prefix sum and compact the result.
   b) The size of a vertex v's subtree is half the difference between the first and last index of v in the tour.
2) Create an Euler Tour of the tree, where the children are visited in increasing order of their subtree size.
3) Drop all but the first occurrence of each vertex using a parallel prefix sum.
4) Permute the vertices according to a space-filling curve.

Next, we show to compute an Euler Tour using list ranking. List ranking is the problem of determining the index of each element in a linked list. By duplicating every edge in a rooted tree and ranking the resulting list, it computes an Euler Tour.

We adapt a contraction-based algorithm [2]. The idea is to repeatedly contract a large independent set of edges.

Initially, each processor stores the index of one vertex and one pointer to the processor that stores the next vertex in the list. To compute the list ranking, for a given list, first select a subset S of non-adjacent vertices using random-mate [2], [3], [44]. Then, do one step of pointer jumping over the vertices from S. When doing the pointer jumping step, store the information of the current iteration number. Increase the temporary vertex rank of any vertex that has its pointer adjusted by the temporary rank of the vertex of which it copies the pointer. Once $\Theta(\log n)$ vertices remain, solve the list ranking problem sequentially. Finally, revert each step of the algorithm, where in each step the vertices that were in S during that iteration, compute their local rank with respect to the vertex that copied their pointer.

Theorem 5. *We can compute the list ranking of a list with n vertices with $O(n^{\frac{3}{2}})$ energy and $O(\log n)$ depth, with high probability.*

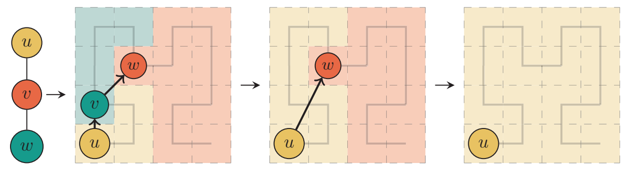

Fig. 5. Example illustrating the compression of supervertices. Every supervertex corresponds to several vertices as indicated by the contiguous space taken up in the grid drawing. We first compress u with v and then u with w.

Proof Sketch: As long as $\Omega(\log n)$ vertices remain, the number of edges decreases by a constant factor in each iteration by a Chernoff bound [11]. The energy of an iteration where n' vertices remain is $O(n'\sqrt{n})$ and its depth is $O(1)$. The base case takes $O(\sqrt{n}\log n)$ energy and $O(\log n)$ depth. ■

As the bottleneck of computing the Euler Tour is in the list ranking, we conclude:

Corollary 2. *Computing an Euler tour on a tree with n vertices takes $\mathcal{O}(n^{\frac{3}{2}})$ energy and $\mathcal{O}(\log n)$ depth with high probability.*

Since all the operations involved in creating the layout can be reduced to the computation of Euler tours, sorting, and parallel prefix sums, Theorem 4 follows. We proceed to show two tree algorithms using our energy-efficient layout.

V. Treefix Sum

We showcase the application of our tree layouts to a classic algorithmic problem. This problem generalizes prefix sums and is related to the parallel evaluation of arithmetic expressions [37]. Given a rooted tree T with a value in each node, the goal of a *treefix sum* is to compute for each vertex v the sum of the values in the subtree rooted at v. Any associative operator may be used instead of a sum. This problem has applications in minimum cut computations [1], [19], [27].

We present a Las Vegas algorithm which solves the treefix sum problem with $O(n\log n)$ energy and $O(\log^2 n)$ depth. The algorithm has $O(\log n)$ depth for trees of bounded degree. The input is a tree T stored in energy-bound light-first order. Each vertex u initially holds a value *val*(u). After the algorithm finishes, each vertex v holds the sum *sum*(v) of the values of its descendants, including its own value *val*(v).

The algorithm has two phases, *tree contraction* and a *uncontraction*. In the tree contraction phase, we use modified versions of the *rake* and *compress* operations [37] and maintain partial sums over merged subtrees. After contracting the whole tree, we undo the contractions to compute the final results using those partial sums. Using this framework ensures that the algorithm has low depth for any tree. The challenges are twofold. First, we need to use local messaging to ensure energy-efficiency. Second, we need to maintain the state using only constant memory per processor.

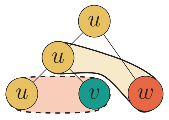

Fig. 6. Example illustrating a contraction tree. The tree is a result of the contractions on the supervertices shown in Figure 5. The solid border represents *last_contracted*(u), whereas the dashed one *saved_state(w)* .

A. Tree Contraction

We adapt the classic rake and compress [37] to our spatial setting. A subset of vertices that induces a single connected component in the tree is a *supervertex*. A supervertex u is represented by and identified with its vertex closest to the root, called its *representative* $R(u)$. This is the first vertex in light-first order of the subgraph induced by the supervertex. Throughout, we will maintain the invariant that every supervertex u stores the *partial sum* P_u of its values in its representative $R(u)$. During the contraction process, we consider the tree of supervertices, which is the tree we get when we merge all vertices in the supervertices. We can contract two supervertices and merge their sets of vertices when they are adjacent in the tree of supervertices. Doing so merges their sets of vertices.

When merging two supervertices, we need to maintain their children before and after the merge. This allows us to undo the sequence of contractions later. We want to store the contraction tree (see Figure 6), which shows the order in which the supervertices were merged. Simply storing a list of operations in the vertex representing the supervertex would lead to unbounded storage. Instead, we distribute this list among the vertices in the supervertex. Each supervertex u stores the representatives of the supervertices that were *last* contracted to create u in its representative $R(u)$ as *last_contracted*(u). If *last_contracted*(u) is non-empty, then the previous value of *last_contracted* before the contraction is stored in *saved_state*(*last_contracted*$(u)[0]$). As long as *last_contracted*(u) has constant size, this ensures the storage remains $O(1)$ throughout. Similarly, we maintain which type of contraction operation was used to create a supervertex (Rake or Compress).

1) Compress: A Compress operation on the supervertices u and v can be performed when v is *the only* child of u and u has a single child as well. Such a compress operation contracts u and v. We increment the partial sum P_u of the supervertex u by the partial sum P_v of the supervertex v. After that, u inherits the children of v. Finally, we set v to be *inactive*, meaning it will wait for the uncontraction to be reactivated. This is illustrated in Figure 5.

2) Rake: A Rake operation (see Figure 7) on u and a subset of its children $v_1, ..., v_i$ can be performed when all these children are leaves and u has at most one other child w. Such a rake operation contracts u and $v_1, ..., v_i$.

Observe that when we perform such a Rake, the supervertex u consists of just a single vertex that is the direct parent of the vertices that represent the supervertices $v_1, ..., v_i$. Hence, we can perform local messaging to efficiently update

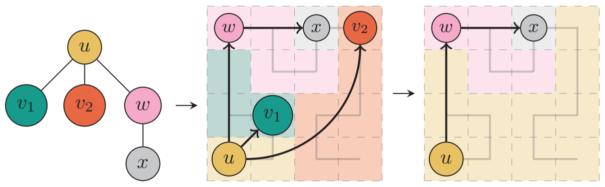

Fig. 7. The example illustrates the rake of the supervertex u and its children v_1 and v_2. Note that w is not part of the rake since it is not a leaf.

the state even when the degree is unbounded. During the rake, we increase the partial sum P_u of the supervertex u by the value of a local reduce with an addition operator, where each child v except w sends its partial sum P_v. A child w that is not part of the rake, if it exists, sends the value 0. The supervertices $v_1, ..., v_i$ are set to *inactive*.

We assumed that *last_contracted* has constant size. Since $\deg(u)$ might be unbounded, we cannot store a list of all the vertices that were part of the rake. Instead, we store the representative of the vertex w that *was not* part of the rake. This is sufficient to undo the RAKE, by doing a local broadcast that omits $R(w)$.

3) Compact: The COMPACT subroutine iteratively applies COMPRESS to reduce the length of the paths in the tree and then RAKE to contract the leaves. A supervertex v is *viable* if and only if its parent is *non-branching* (i.e., it is the only child) and v has exactly one child. Only the *viable* supervertices can be compressed with their parents. COMPACT works as follows:

1) Each supervertex sends a message to its child supervertices indicating if it is *branching*.
2) Find a set R of independent viable supervertices. This set can be found using local messaging with a randomized approach called random-mate [2], [3], [44]: Independently, each vertex chooses heads or tails with probability $p = \frac{1}{2}$. Include in R every viable vertex that chose heads and whose predecessor chose tails.
3) COMPRESS every vertex in R with its parent.
4) Repeat step (1).
5) RAKE the supervertices where it is possible.

Lemma 10. COMPACT *takes $O(n)$ energy and its depth is $O(1)$ or $O(\log n)$ for bounded and unbounded degree trees, respectively.*

Proof: (i) We first note that by the triangle inequality, the energy of sending a message from u to v and then from v to w is at least the energy of sending a message from a supervertex u to a supervertex w (which was created by contracting v and w). (ii) In every round, each supervertex sends a constant number of messages to its children and parents. (iii) By Theorem 3, the total energy of every vertex sending a message to its parent and children is $O(n)$. From (i), (ii), and (iii) it follows that the total energy spent per round of COMPACT is in $O(n)$. The depth follows from the local messaging bounds in Theorem 1 and Theorem 3. ∎

B. Uncontraction

We can use the partial sums and the contraction tree to undo the contractions and compute the result. We undo the contractions using local messaging based on *last_contracted* and *saved_state*.

We maintain a value A_u for every vertex u that represents the sum of the values of descendants of the current supervertex represented by u that are not in u. This value needs to be added to the partial sum of the supervertex u to get the sum at u. The invariant is that for every supervertex u, its result is $\mathit{sum}(u) = P_u + A_u$. Initially, $A_u = 0$. This invariant can be maintained as follows. When undoing a COMPRESS of supervertex u with child supervertex v, increment A_u by P_v and decrement P_u by P_v. For the supervertex v, set $A_v = A_u$. When undoing a RAKE of supervertex u with children $v_1, \ldots v_i$, set $P_u = \mathit{val(u)}$ and $A_u = P_u - \mathit{val}(u)$.

C. A Local Messaging Treefix Sum Algorithm

The local messaging treefix sum algorithm works as follows: As long as there exist two or more supervertices, apply COMPACT to the supervertices. Once there is one supervertex left, perform uncontraction to get the results. Synchronization between the rounds would be a bottleneck (in the bounded degree case) since it requires $O(\log n)$ depth. Note that in each round of COMPACT, a supervertex only needs to communicate with its parent and children. Therefore, we can safely avoid global synchronization by having each vertex execute the steps as soon as possible.

Lemma 11. *For a bounded degree tree, treefix sum takes $O(n \log n)$ energy and $O(\log n)$ depth, with high probability.*

Proof: It takes $O(\log n)$ repetitions of COMPACT to contract the tree to a single vertex, with high probability. This follows from the analysis of the Random-mate approach to find an independent set [3], [37]. Note that this routine only requires messages between neighboring supervertices. It follows from Lemma 10 that it takes $O(n \log n)$ energy and $O(\log n)$ depth to contract the tree, with high probability. The uncontraction essentially reverses the operations. ∎

Lemma 12. *Treefix sum takes $O(n \log n)$ energy and $O(\log^2 n)$ depth, with high probability.*

Proof: The proof is analogous to the one for constant degree trees except each round of COMPACT has depth $O(\log n)$. ∎

D. Top-down Treefix Sum

Similarly, we can also compute for every vertex u the sum $\mathit{sum}'(u)$ of the values along the path from the root to u. The only change is in the uncontraction. We maintain the invariant that for every supervertex u, $\mathit{sum}'(u) = \mathit{val}(u) + A_u$. Initially, A_r is zero for the root supervertex r. When undoing a COMPRESS of a supervertex u with child supervertex v, set $A_v = A_u + P_u - P_v$. Then, decrement P_u by P_v. When undoing a RAKE of a supervertex u with supervertices $v_1, \ldots, v_i$, set $A_v = A_u + \mathit{val}(u)$ and set P_u to $\mathit{val}(u)$.

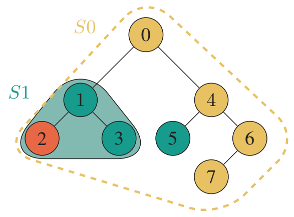

Fig. 8. The path decomposition creates a set of $O(\log n)$ layers consisting of disjoint paths. The yellow path $(0, 4, 6, 7)$ is in layer 0, the green paths $(1, 3)$ and (5) are in layer 1, and the red path (2) is in layer 2. We get the subtree cover by extending each path to contain all the descendants of the root of the path. Each vertex is labeled with its light-first order. The subtree $S0$ corresponds to the range $[0, 7]$ and the subtree $S1$ to $[1, 3]$.

VI. Lowest Common Ancestor

The *lowest common ancestor* (LCA) of two vertices u and v is the lowest vertex w, such that both u and v are descendants of w. We design an algorithm that processes multiple queries of the form LCA(u, v) with $O(n \log n)$ energy and $O(\log^2 n)$ depth, where each vertex appears in at most a constant number of queries. If this is not the case, the tree can be preprocessed by splitting a vertex with many queries into multiple vertices that form a path and distributing the queries among them.

Our approach is to design an algorithm for LCA that fits into the local messaging framework. Hence, we can benefit from the linear energy bounds of local messaging. We cannot directly use previous approaches [3], [12], [16], as they require non-local messaging. Using treefix sums, we can answer queries where one vertex is a descendant of the other. For the other types of queries, we introduce a notion of *subtree covers* of the tree. Every vertex is part of at least one and at most $O(\log n)$ subtrees in the cover. The cover guarantees that there is a subtree S such that exactly one of the query vertices is in S. This means that the lowest common ancestor is the parent of the root of S. Because of this simple structure, we can identify the solution using a series of broadcasts and reductions on the subtrees in the cover.

A. Path Decomposition

A *path decomposition* $\mathcal{P}$ is a partition of the tree into a set of disjoint paths. Consider a path $P \in \mathcal{P}$ rooted at vertex v. If the root-to-v path intersects i other paths in $\mathcal{P}$, then P and v are in the i-th layer of $\mathcal{P}$. Note that if a vertex v is a child of a vertex u and they are in different paths of the decomposition, then u is in some i-th layer and v is in the $(i + 1)$-th layer. See Figure 8 (top) for an example.

A suitable decomposition, the heavy-light decomposition [28], can be directly constructed from light-first order: Always connect a vertex with its heaviest child. This is the rightmost child in light-first order. Observe that every time we go down to a child that is not rightmost, the size of the subtree decreases at least by a factor 2. Hence, connecting vertices via their rightmost children constitutes a path decomposition with $O(\log n)$ layers. We can compute the layer of its path for every vertex with a top-down treefix sum (see Section V-D) and an appropriate associative operator. It takes $O(n \log n)$ energy and $O(\log n)$ depth to construct such a decomposition with local messaging.

B. Subtree Cover

A *subtree cover* is defined given a path decomposition $\mathcal{P}$, as follows. For each path P in the path decomposition $\mathcal{P}$, the subtree cover contains the subtree rooted at the root of that path P. We say that a subtree is i-th if it is rooted at a vertex on the i-th layer. Note that all i-th subtrees are pairwise vertex disjoint, but subtrees from different levels overlap. See Figure 8 for an example subtree cover.

Next, we show the main structural lemma regarding subtree covers. We say that a vertex w is a parent of a subtree S rooted at vertex v if w is a parent of v.

Corollary 3. *Consider a subtree cover and let LCA(u, v) = w. Then, either $w \in \{u, v\}$ or w is a parent of a subtree in the cover that contains exactly one of those vertices.*

Proof: We assume that $w \notin \{u, v\}$ and show that w is a parent of a subtree in the cover that contains one of the vertices. Let $\mathcal{P}$ be the path decomposition that defined the subtree cover. Because $w \notin \{u, v\}$, there must be two children x_u and x_v of w that are ancestors of u and v, respectively. Only one of those two vertices can be in the same path of the path decomposition $\mathcal{P}$ as w. The other vertex must therefore be the root of a subtree in the cover. ■

C. Local Messaging LCA Algorithm

We assume that the input tree is stored in energy-bound light-first order. For each query LCA(u, v), we assume the input is stored in both u and v. The result of the query is stored in one of the two vertices.

Consider a *range* $[a, b]$ for integers $0 \leq a \leq b \leq n$. We say a vertex $v \in [a, b]$ if v is stored in the i-th processor in the message bound curve order for some i with $a \leq i \leq b$. Each subtree S rooted at v corresponds to exactly one contiguous range $r(v) = [a, b]$ that contains all its descendants. We can test if a vertex is in a subtree given the range of its root. See Figure 8 for an example.

Our local messaging LCA algorithm has four main steps:

1) Let each vertex have value 1 and run the treefix sum algorithm such that each vertex u contains a value $sum(u)$, which is the size of the subtree rooted at the vertex u. The range of a subtree rooted at a vertex u stored in i-th position in light-first order is now $r(u) = [i, i+sum(u)-1]$. Then, answer all queries where LCA$(u, v) \in \{u, v\}$. If $v \in r(u)$ then u answers that query; otherwise, v answers it.
2) Every vertex w does a local broadcast to send the range of its subtree $r(w)$ to its children.
3) Compute the path decomposition using a top-down treefix sum.
4) Let $\mathcal{B}$ be the subtree cover. For each layer i in increasing order, consider all subtrees S in that layer. Let w be the parent of the subtree S and let x be its root.

a) Broadcast $r(w) \backslash r(x)$ within S, where $r(w) \backslash r(x)$ is the range of w excluding the range of x.
b) If u is in the subtree S, we know that $LCA(u, v) = w$ if $v \in r(w) \backslash r(x)$. We answer those queries.

Perform a synchronization barrier before proceeding to the next layer. This can be performed by doing an all-reduce on the compute grid, where a processor starts the all-reduce once it hits the barrier.

The correctness of the algorithm follows directly from Theorem 3. If $LCA(u, v) = w \notin \{u, v\}$, at some point exactly one of the vertices will be part of a subtree which is a child of w. The query will therefore always have a correct answer.

Lemma 13. *It takes $O(b-a)$ energy and $O(\log(b-a))$ depth to broadcast a message on the range $[a, b]$ with $a < b$ when the tree is stored in energy-bound light-first order.*

Proof: We build a virtual broadcast tree on the processors in the range, such that the tree is a complete binary tree in energy-bound light-first order. Because the range is contiguous, such a tree can be constructed top-down using only the indexes of the range. The statement now follow from light first order being message bound. ■

Theorem 6. *The local messaging LCA algorithm takes $O(n \log n)$ energy and $O(\log^2 n)$ depth with high probability.*

Proof: With high probability, the first three steps take $O(n \log n)$ energy and $O(\log^2 n)$ depth, by Theorem 12. Note that every subtree in the cover corresponds to a range and that subtrees from the same level are disjoint. Hence, the sum of their ranges is at most n. By Lemma VI-C, we can do the broadcast within all the i-th subtrees in $O(n)$ energy. The barrier takes $O(n)$ energy and $O(\log n)$ depth using an all-reduce [20]. We conclude that the fourth step takes $O(n \log n)$ energy overall. For each level, the depth is $O(\log n)$ for the broadcasts and the barrier. Hence, the depth is $O(\log^2 n)$. ■

This near-linear energy and low depth algorithm constitutes a significant improvement on the naïve PRAM simulation, which would take $\Omega(n^{\frac{3}{2}})$ energy [20].

VII. Conclusion

This paper presents spatial tree algorithms that minimize communication distances between vertices on a 2D computational grid through a twofold framework:

1) *Data Layout*: Our strategy mapped high-dimensional tree structures onto a localized linear layout. This was subsequently lifted to the 2D grid using space-filling curves. This layout reduces the average distance between neighboring vertices to a *constant*, optimizing communication.
2) *Logical Operations*: By separating logical operations from the layout, we could leverage existing strategies from the PRAM environment. This separation allowed us to maintain low depth and work while also achieving low energy. We addressed two foundational tree problems: treefix sum and lowest common ancestors. For both approaches, our algorithms exhibited near-linear energy and poly-logarithmic depth.

This framework could be fruitful for optimizing other sparse workloads, including sparse matrix-vector multiplication and graph clustering. Future exploration of layouts supporting dynamic updates may enhance the real-time adaptability of our framework. Not only could this address current limitations that require layouts to be precomputed, but it could also pave the way for more dynamic and versatile applications.

In conclusion, this research constitutes a novel approach to deploying specialized architectures, such as the Wafer-Scale engine [48] and CGRAs [51]. Our work addresses irregular sparse workloads and unlocks new possibilities in large-scale data science applications.

Acknowledgment

This project received funding from the European Research Council (Project PSAP, No. 101002047). This project was supported by the ETH Future Computing Laboratory (EFCL), financed by a donation from Huawei Technologies. This project received funding from the European Research Council under the European Union's Horizon 2020 programme GLACIATION, No. 101070141. T.B.N. was supported by the Swiss National Science Foundation (Ambizione Project No. 185778). The language in the abstract, Section I, and Section VII has been enhanced with a large language model [40].

Appendix A
Proof of Lemma 2

We begin with the following inequality:

Lemma 14. *Let a, b be positive constants with $\frac{b}{2} \leq a \leq b$, then $\forall 0 \leq 2x \leq y$ we have $b\sqrt{y} \leq a\sqrt{x} + b\sqrt{y-x}$.*

Proof: Let $f(x, y) = b\sqrt{y}$ and $g(x, y) = a\sqrt{x} + b\sqrt{y-x}$. We show that $f^2(x, y) - g^2(x, y) \leq 0$, which implies the result. If $x = 0$, then we have equality trivially. For $x > 0$, we have:

$$\begin{aligned}
&f^2(x, y) - g^2(x, y) \\
&= b^2 y - a^2 x - 2ab\sqrt{x(y-x)} - b^2 y + b^2 x \\
&\leq \left(b^2 - \left(\frac{b}{2}\right)^2 - 2\frac{b}{2}b\right) x \\
&\leq -\frac{b^2}{4} \cdot x \leq 0 \ .
\end{aligned}$$

We conclude using that $f(x, y)$ and $g(x, y)$ are nonnegative. ■

We continue with the proof of Lemma 2.

Proof of Lemma 2: Let us assume we start with $s(c_\Delta) = n$ and the other variables set to 0, i.e., our value is $2\Delta \cdot c\sqrt{n}$. We can create any combination that fulfils the constraints by applying $\Delta - 1$ transformations. For $i = 1, ..., \Delta - 1$, we increment $s(c_i)$ by some value x_i and subtract that value from $s(c_d)$. The problem is equivalent to the one in Lemma 14: Let y_i be the value of $s(c_\Delta)$ before the i-th transformation. In our case we are comparing $b\sqrt{y_i}$ with $a\sqrt{x_i} + b\sqrt{y_i - x_i}$, where $\frac{b}{2} = \Delta \cdot c \leq a = (\Delta + i) \cdot c \leq 2\Delta \cdot c = b$. Since $s(c_\Delta) \geq s(c_i)$, we have $0 \leq 2x_i \leq y_i$. Hence, no such transformation decreases the sum; the function is minimized for $s(c_\Delta) = n$. ■

REFERENCES

[1] D. Anderson and G. E. Blelloch, “Parallel minimum cuts in $O(m \log^2 n)$ work and low depth,” 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. 71–82. [Online]. Available: https://doi.org/10.1145/3409964.3461797

[2] R. J. Anderson and G. L. Miller, “A simple randomized parallel algorithm for list-ranking,” *Inf. Process. Lett.*, vol. 33, no. 5, pp. 269–273, 1990. [Online]. Available: https://doi.org/10.1016/0020-0190(90)90196-5

[3] L. Arge, M. T. Goodrich, and N. Sitchinava, “Parallel external memory graph algorithms,” in *24th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2010, Atlanta, Georgia, USA, 19-23 April 2010 - Conference Proceedings*. IEEE, 2010, pp. 1–11. [Online]. Available: https://doi.org/10.1109/IPDPS.2010.5470440

[4] M. Bader, *Space-Filling Curves*, ser. Texts in Computational Science and Engineering. Berlin, Heidelberg: Springer, 2013, vol. 9. [Online]. Available: http://link.springer.com/10.1007/978-3-642-31046-1

[5] T. Ben-Nun, M. Besta, S. Huber, A. N. Ziogas, D. Peter, and T. Hoefler, “A modular benchmarking infrastructure for high-performance and reproducible deep learning,” in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2019, pp. 66–77.

[6] O. Berkman and U. Vishkin, “Recursive star-tree parallel data structure,” *SIAM J. Comput.*, vol. 22, no. 2, pp. 221–242, 1993. [Online]. Available: https://doi.org/10.1137/0222017

[7] M. Besta and T. Hoefler, “Parallel and distributed graph neural networks: An in-depth concurrency analysis,” *arXiv preprint arXiv:2205.09702*, 2022.

[8] L. Breiman, “Random forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[9] 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

[10] A. Chan and F. K. H. A. Dehne, “A note on coarse grained parallel integer sorting,” *Parallel Process. Lett.*, vol. 9, no. 4, pp. 533–538, 1999. [Online]. Available: https://doi.org/10.1142/S0129626499000499

[11] H. Chernoff, “A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the sum of Observations,” *The Annals of Mathematical Statistics*, vol. 23, no. 4, pp. 493 – 507, 1952. [Online]. Available: https://doi.org/10.1214/aoms/1177729330

[12] Y. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter, “External-memory graph algorithms,” in *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1995. San Francisco, California, USA*, K. L. Clarkson, Ed. ACM/SIAM, 1995, pp. 139–149. [Online]. Available: http://dl.acm.org/citation.cfm?id=313651.313681

[13] S. A. Chin, N. Sakamoto, A. Rui, J. Zhao, J. H. Kim, Y. Hara-Azumi, and J. H. Anderson, “CGRA-ME: A unified framework for CGRA modelling and exploration,” in *28th IEEE International Conference on Application-specific Systems, Architectures and Processors, ASAP 2017, Seattle, WA, USA, July 10-12, 2017*, 2017, pp. 184–189. [Online]. Available: https://doi.org/10.1109/ASAP.2017.7995277

[14] F. K. H. A. Dehne, W. Dittrich, and D. A. Hutchinson, “Efficient external memory algorithms by simulating coarse-grained parallel algorithms,” in *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA ’97, Newport, RI, USA, June 23-25, 1997*, C. E. Leiserson and D. E. Culler, Eds. ACM, 1997, pp. 106–115. [Online]. Available: https://doi.org/10.1145/258492.258503

[15] F. K. H. A. Dehne, A. Fabri, and A. Rau-Chaplin, “Scalable parallel computational geometry for coarse grained multicomputers,” *Int. J. Comput. Geom. Appl.*, vol. 6, no. 3, pp. 379–400, 1996. [Online]. Available: https://doi.org/10.1142/S0218195996000241

[16] F. K. H. A. Dehne, A. Ferreira, E. Cáceres, S. W. Song, and A. Roncato, “Efficient parallel graph algorithms for coarse-grained multicomputers and BSP,” *Algorithmica*, vol. 33, no. 2, pp. 183–200, 2002. [Online]. Available: https://doi.org/10.1007/s00453-001-0109-4

[17] N. Dey, G. Gosal, Z. Chen, H. Khachane, W. Marshall, R. Pathria, M. Tom, and J. Hestness, “Cerebras-gpt: Open compute-optimal language models trained on the cerebras wafer-scale cluster,” *CoRR*, vol. abs/2304.03208, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2304.03208

[18] B. Gaide, D. Gaitonde, C. Ravishankar, and T. Bauer, “Xilinx adaptive compute acceleration platform: Versal$^{\text{tm}}$ architecture,” in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA 2019, Seaside, CA, USA, February 24-26, 2019*, K. Bazargan and S. Neuendorffer, Eds. ACM, 2019, pp. 84–93. [Online]. Available: https://doi.org/10.1145/3289602.3293906

[19] B. Geissmann and L. Gianinazzi, “Parallel minimum cuts in near-linear work and low depth,” in *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, C. Scheideler and J. T. Fineman, Eds. ACM, 2018, pp. 1–11. [Online]. Available: https://doi.org/10.1145/3210377.3210393

[20] L. Gianinazzi, T. Ben-Nun, M. Besta, S. Ashkboos, Y. Baumann, P. Luczynski, and T. Hoefler, “The spatial computer: A model for energy-efficient parallel computation,” 2022. [Online]. Available: https://arxiv.org/abs/2205.04934

[21] H. J. Haverkort and F. van Walderveen, “Locality and bounding-box quality of two-dimensional space-filling curves,” in *Algorithms - ESA 2008, 16th Annual European Symposium, Karlsruhe, Germany, September 15-17, 2008. Proceedings*, ser. Lecture Notes in Computer Science, D. Halperin and K. Mehlhorn, Eds., vol. 5193. Springer, 2008, pp. 515–527. [Online]. Available: https://doi.org/10.1007/978-3-540-87744-8_43

[22] D. Hilbert, “Ueber die stetige abbildung einer line auf ein flächenstück,” *Mathematische Annalen*, vol. 38, no. 3, pp. 459–460, Sep 1891. [Online]. Available: https://doi.org/10.1007/BF01199431

[23] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, “Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks,” *J. Mach. Learn. Res.*, vol. 22, no. 1, jan 2021.

[24] P. Iff, M. Besta, M. Cavalcante, T. Fischer, L. Benini, and T. Hoefler, “Hexamesh: Scaling to hundreds of chiplets with an optimized chiplet arrangement,” in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[25] P. Iff, B. Bruggmann, M. Besta, L. Benini, and T. Hoefler, “Rapidchiplet: A toolchain for rapid design space exploration of chiplet architectures,” *arXiv preprint arXiv:2311.06081*, 2023.

[26] 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

[27] D. R. Karger, “Minimum cuts in near-linear time,” in *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, G. L. Miller, Ed. ACM, 1996, pp. 56–63. [Online]. Available: https://doi.org/10.1145/237814.237829

[28] P. N. Klein, “Computing the edit-distance between unrooted ordered trees,” in *Algorithms - ESA ’98, 6th Annual European Symposium, Venice, Italy, August 24-26, 1998, Proceedings*, ser. Lecture Notes in Computer Science, G. Bilardi, G. F. Italiano, A. Pietracaprina, and G. Pucci, Eds., vol. 1461. Springer, 1998, pp. 91–102. [Online]. Available: https://doi.org/10.1007/3-540-68530-8_8

[29] S. B. Kotsiantis, “Decision trees: a recent overview,” *Artif. Intell. Rev.*, vol. 39, no. 4, pp. 261–283, 2013. [Online]. Available: https://doi.org/10.1007/s10462-011-9272-4

[30] F. T. Leighton, “Introduction to parallel algorithms and architectures: Arrays, trees, hypercubes,” 1991.

[31] ——, “Complexity issues in vlsi: Optimal layouts for the shuffle-exchange graph and other networks,” 2003.

[32] T. Leighton, “Parallel computation using meshes of trees,” in *Proceedings of the WG ’83, International Workshop on Graphtheoretic Concepts in Computer Science, June 16-18, 1983, Haus Ohrbeck, near Osnabrück, Germany*, M. Nagl and J. Perl, Eds. Universitätsverlag Rudolf Trauner, Linz, 1983, pp. 200–218.

[33] R. Lin and S. Olariu, “A simple optimal parallel algorithm to solve the lowest common ancestor problem,” in *Advances in Computing and Information - ICCI’91, International Conference on Computing and Information, Ottawa, Canada, May 27-29, 1991, Proceedings*, ser. Lecture Notes in Computer Science, F. K. H. A. Dehne, F. Fiala, and W. W. Koczkodaj, Eds., vol. 497. Springer, 1991, pp. 455–461. [Online]. Available: https://doi.org/10.1007/3-540-54029-6_194

[34] ——, “A fast cost-optimal parallel algorithm for the lowest common ancestor problem,” *Parallel Comput.*, vol. 18, no. 5, pp. 511–516, 1992. [Online]. Available: https://doi.org/10.1016/0167-8191(92)90086-M

[35] R. J. Lipton and R. Sedgewick, "Lower bounds for VLSI," in *Proceedings of the 13th Annual ACM Symposium on Theory of Computing, May 11-13, 1981, Milwaukee, Wisconsin, USA*. ACM, 1981, pp. 300–307. [Online]. Available: https://doi.org/10.1145/800076.802482

[36] 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," 2023. [Online]. Available: http://hdl.handle.net/10754/694388

[37] G. L. Miller and J. H. Reif, "Parallel tree contraction and its application," in *26th Symposium on Foundations of Computer Science*. Portland, Oregon: IEEE, October 1985, pp. 478–489.

[38] R. Niedermeier, K. Reinhardt, and P. Sanders, "Towards optimal locality in mesh-indexings," *Discrete Applied Mathematics*, vol. 117, no. 1, pp. 211–237, 2002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166218X00003267

[39] R. Niedermeier and P. Sanders, "On the Manhattan-distance between points on space-filling mesh-indexings," 1996. [Online]. Available: https://publikationen.bibliothek.kit.edu/17796

[40] OpenAI, "Chatgpt (september 25 version)," 2023. [Online]. Available: https://chat.openai.com/chat

[41] E. Pennisi, "Modernizing the tree of life," *Science*, vol. 300, no. 5626, pp. 1692–1697, 2003. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.300.5626.1692

[42] W. H. Piel, L. Chan, M. J. Dominus, J. Ruan, R. A. Vos, and V. Tannen, "Treebase v. 2: A database of phylogenetic knowledge," in *e-BioSphere 2009*, 2009.

[43] A. Podobas, K. Sano, and S. Matsuoka, "A survey on coarse-grained reconfigurable architectures from a performance perspective," *IEEE Access*, vol. 8, pp. 146 719–146 743, 2020. [Online]. Available: https://doi.org/10.1109/ACCESS.2020.3012084

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

[45] 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

[46] B. Schieber and U. Vishkin, "On finding lowest common ancestors: Simplification and parallelization," *SIAM J. Comput.*, vol. 17, no. 6, pp. 1253–1262, 1988. [Online]. Available: https://doi.org/10.1137/0217079

[47] I. Swarbrick, D. Gaitonde, S. Ahmad, B. Gaide, and Y. Arbel, "Network-on-chip programmable platform in versal$^{\text{tm}}$ ACAP architecture," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA 2019, Seaside, CA, USA, February 24-26, 2019*, K. Bazargan and S. Neuendorffer, Eds. ACM, 2019, pp. 212–221. [Online]. Available: https://doi.org/10.1145/3289602.3293908

[48] 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

[49] R. Tarjan and U. Vishkin, "Finding biconnected componemts and computing tree functions in logarithmic parallel time," in *25th Annual Symposium onFoundations of Computer Science, 1984.*, 1984, pp. 12–20.

[50] A. Trifan, D. Gorgun, M. Salim, Z. Li, A. Brace, M. Zvyagin, H. Ma, A. Clyde, D. Clark, D. J. Hardy, T. Burnley, L. Huang, J. McCalpin, M. Emani, H. Yoo, J. Yin, A. Tsaris, V. Subbiah, T. Raza, J. Liu, N. Trebesch, G. Wells, V. Mysore, T. Gibbs, J. Phillips, S. C. Chennubhotla, I. Foster, R. Stevens, A. Anandkumar, V. Vishwanath, J. E. Stone, E. Tajkhorshid, S. A. Harris, and A. Ramanathan, "Intelligent resolution: Integrating cryo-em with ai-driven multi-resolution simulations to observe the severe acute respiratory syndrome coronavirus-2 replication-transcription machinery in action," *Int. J. High Perform. Comput. Appl.*, vol. 36, no. 5–6, p. 603–623, nov 2022. [Online]. Available: https://doi.org/10.1177/10943420221113513

[51] K. A. Vissers, "Versal: The xilinx adaptive compute acceleration platform (ACAP)," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA 2019, Seaside, CA, USA, February 24-26, 2019*, K. Bazargan and S. Neuendorffer, Eds. ACM, 2019, p. 83. [Online]. Available: https://doi.org/10.1145/3289602.3294007

[52] R. A. Vos, J. P. Balhoff, J. A. Caravas, M. T. Holder, H. Lapp, W. P. Maddison, P. E. Midford, A. Priyam, J. Sukumaran, X. Xia, and A. Stoltzfus, "Nexml: rich, extensible, and verifiable representation of comparative data and metadata," *Systematic Biology*, vol. 61, no. 4, pp. 675–689, 2012.

[53] M. Woo, T. Jordan, R. Schreiber, I. Sharapov, S. Muhammad, A. Koneru, M. James, and D. V. Essendelft, "Disruptive changes in field equation modeling: A simple interface for wafer scale engines," *CoRR*, vol. abs/2209.13768, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2209.13768

[54] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 1, pp. 249–270, 2020.