Near-Optimal Wafer-Scale Reduce

Piotr Luczynski*
ETH Zurich
Department of Computer Science
Zurich, Switzerland
piotr.luczynski@inf.ethz.ch

Lukas Gianinazzi*
ETH Zurich
Department of Computer Science
Zurich, Switzerland
lukas.gianinazzi@inf.ethz.ch

Patrick Iff
ETH Zurich
Department of Computer Science
Zurich, Switzerland

Leighton Wilson
Cerebras Systems
Sunnyvale, CA, USA

Daniele De Sensi
Sapienza University of Rome
Department of Computer Science
Rome, Italy

Torsten Hoefler
ETH Zurich
Department of Computer Science
Zurich, Switzerland

ABSTRACT

Efficient Reduce and AllReduce communication collectives are a critical cornerstone of high-performance computing (HPC) applications. We present the first systematic investigation of Reduce and AllReduce on the Cerebras Wafer-Scale Engine (WSE). This architecture has been shown to achieve unprecedented performance both for machine learning workloads and other computational problems like FFT. We introduce a performance model to estimate the execution time of algorithms on the WSE and validate our predictions experimentally for a wide range of input sizes. In addition to existing implementations, we design and implement several new algorithms specifically tailored to the architecture. Moreover, we establish a lower bound for the runtime of a Reduce operation on the WSE. Based on our model, we automatically generate code that achieves near-optimal performance across the whole range of input sizes. Experiments demonstrate that our new Reduce and AllReduce algorithms outperform the current vendor solution by up to 3.27×. Additionally, our model predicts performance with less than 4% error. The proposed communication collectives increase the range of HPC applications that can benefit from the high throughput of the WSE. Our model-driven methodology demonstrates a disciplined approach that can lead the way to further algorithmic advancements on wafer-scale architectures.

1 INTRODUCTION

1.1 Motivation

Communication collectives are essential in numerous distributed applications [3, 37]. Consequently, their efficient implementation is crucial to achieve high communication performance. Among these, Reduce and AllReduce are the collectives most frequently utilized in typical HPC workloads [10, 33]. Specifically, these operations are critical in GEMV and GEMM kernels for fields like deep learning [5, 9, 38, 50, 54], bioinformatics [41, 49], and physics simulations [6, 36]. These heterogeneous applications demand a variety of input shapes.

The Cerebras WSE represents a groundbreaking architecture designed specifically to expedite machine learning workloads. Traditional architectures such as CPUs and GPUs use shared DRAM memories, which can lead to long access latencies. Instead, the WSE

features hundreds of thousands of processing elements (PE), each equipped with a local fast static random-access memory (SRAM), thereby enabling single-cycle access latencies. These PEs communicate via an on-wafer 2D mesh network that supports multicasting, which influences communication efficiency and patterns, setting it apart from contemporary distributed systems that typically utilize low diameter networks [4, 13, 29, 30]. The architecture of the WSE delivers high throughput for machine learning training [14, 35, 50] and various other HPC applications [36, 39, 52, 59]. However, maximizing performance on this architecture necessitates tailoring communication patterns to its unique characteristics. This need motivates our investigation of Reduce and AllReduce on the WSE.

1.2 Limitations of state-of-the-art

Current wafer-scale Reduce and AllReduce implementations are primarily optimized for extreme vector sizes. This means they are suboptimal for the intermediate and variable vector lengths typical in HPC applications. Furthermore, certain implementations like ring, though efficient in conventional systems, underperform on specialized hardware like the WSE. Existing approaches for wafer-scale algorithms employ ad-hoc theoretical modeling on a per-problem basis [39] or depend solely on experimental validation [25, 36, 44, 52], leading to time-consuming trial-and-error or suboptimal performance. The results from traditional distributed memory computing models like the $\alpha - \beta$ model [7] do not consider features such as pipelining and multicasting, which are essential in the wafer-scale setting. Therein we identify the gap for a model-driven approach to optimizing communication collectives.

1.3 Key Insights and Contributions

This work presents a robust methodology for designing, analyzing, and implementing algorithms tailored to architectures similar to the WSE. The specific contributions are:

(1) **Model.** We propose a novel model-driven approach. Our performance model accurately predicts execution times on the Cerebras WSE, providing a significant improvement over trial-and-error and ad-hoc methods. Our experiments confirm the model's accuracy in predicting execution times, accurately characterizing the relative strengths of each approach.

(2) **Algorithms.** We illustrate the effective use of this model by characterizing the performance trade-offs of *Broadcast*, *Reduce*, and *AllReduce* algorithms. For Broadcast, our analysis shows

---
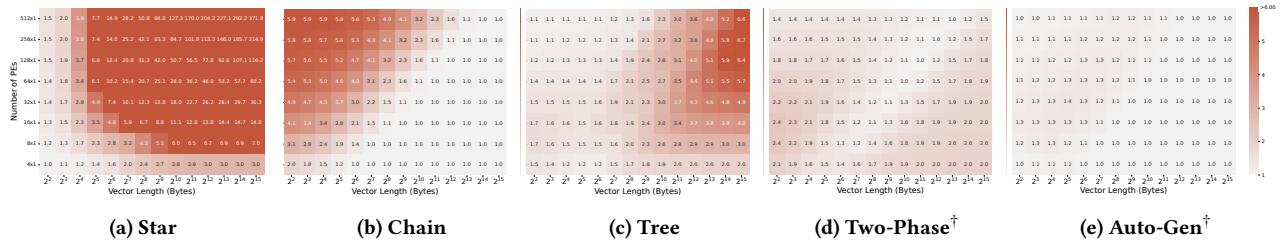
*Both authors contributed equally.

Figure 1: Optimality ratios of 1D Reduce algorithms, where 1.0 is optimal. (†) *our contribution.*

that multicast support means that a simple flooding approach is optimal. We introduce a new (All)Reduce algorithm tailored to the architecture, which we call Two-Phase. Previous Reduce and AllReduce excel only within narrow ranges of input size. Two-Phase is the only approach that performs well for a wide range of vector lengths. In particular, on 512 × 512 PEs, Two-Phase is up to 3.32× and 2.56× faster than the current vendor solution for Reduce and AllReduce, respectively. We adapt classic ring AllReduce algorithm and asses its performance on the Cerebras WSE. Our model-driven approach enables a comparison of ring's performance against a direct Reduce-then-Broadcast method. Interestingly, the direct approach frequently outperforms the classical algorithm not designed with the Cerebras WSE's particular hardware features, such as multicast, in mind.

(3) **Automatically Generated Collectives** Furthermore, we demonstrate how our model-driven approach facilitates the automatic generation of code for Reduce, significantly enhancing efficiency. This novel automatically-generated (*Auto-Gen*) algorithm not only streamlines the optimization of complex kernels but also presents a less time-consuming alternative to manual tuning. Our experiments show that the Auto-Gen Reduce consistently matches or exceeds the performance of the best manual implementations across various input sizes.

(4) **Lower Bound** Additionally, our model is instrumental in establishing robust lower bounds for the runtime of Reduce. As summarized in Figure 1, we prove that for the 1D case, our Auto-Gen Reduce is at most 1.4× away from optimal across all input sizes. Two-Phase gives the best optimality ratio of the manual algorithms, being at most 2.4× away from optimal. In contrast, previous algorithms are all up to 5.9× away from optimal for some input size.

(5) **Implementation** We implement the proposed communication collectives for the second generation WSE, the Cerebras CS-2.

1.4 Experimental Methodology

We perform an extensive evaluation of our collectives on the CS-2 device. The benchmarks consider a row of PEs as well as a 2D grid of PEs. High precision measurements are of crucial importance when measuring short durations. Common problems with time measurements for distributed architectures [23, 60] need to be addressed. We propose a solution to synchronizing the clock between PEs and establishing a common start time. Each benchmark is evaluated 5 times with negligible standard deviation (< 4%). The small number of evaluations suffice because the CS-2 exhibits small runtime variance. Because execution of a thread cannot be preempted, the PE

programs exhibit deterministic, state-machine like behavior which can be modeled with a cycle-accurate fabric simulator. The only notable deviation between the fabric simulator and the physical chip is overheating, which can cause a PE to insert no-ops to prevent wafer cracking. The source code is available on GitHub[1].

1.5 Limitations of the Proposed Approach

Although we demonstrate that our lower bounds are near-optimal in a 1D row or column of PEs, for a 2D grid of PEs the optimality gap remains large. This is in part due to the lack of a strong lower bound for the 2D case. In turn, our model suggests further improvements are possible for the general 2D case.

2 BACKGROUND

2.1 Communication Collectives

The *Message Passing Interface* (MPI) standard [19, 37] defines semantics for collective operations in systems with distributed memory. MPI collectives have been extensively studied and optimized for a variety of network topologies [7, 22, 27, 28, 46, 51, 53]. In a *Reduce*, initially each PE holds a vector of equal length. The goal is to compute the sum of the vector and store it at a designated *root* PE. We consider the sum over the vector, although any associative operation may be used interchangeably. In an *AllReduce*, the result must be stored in every PE. Many patterns and techniques have been developed for *Reduce* and *AllReduce* [26, 40, 42, 43] offering different tradeoffs.

The ring algorithm is a bandwidth optimal AllReduce [1, 21], but it is mostly used to reduce large vectors or when running on a few nodes, since it performs a number of steps equal to the number of nodes minus one. Another notable example is the butterfly pattern [42], which relies on recursive halving and doubling to reduce the number of steps compared to the ring algorithm.

Although some algorithms have been optimized for torus [11, 26, 47] or mesh networks [32], the specific features of Cerebras CS-2, like hardware support for multicast and pipelining require the design of novel algorithms that can fully exploit those capabilities.

Some AllReduce implementations exploit the network hardware support for multicast [31] or in-network compute capabilities to perform vector aggregation in the network switches [12, 18].

2.2 Wafer-Scale Engine

The Cerebras CS-2 [24, 25, 34] consists of around 750,000 processing elements (PEs) structured in a 2D grid. See Figure 2 for an overview

---

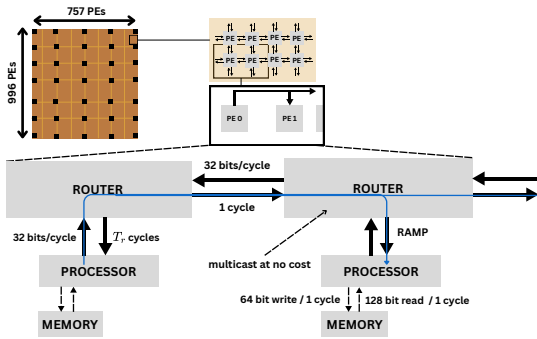[1]https://github.com/spcl/spatial-collectives

Figure 2: PE 0 sends a wavelet to the neighbouring PE 1 on the blue color. The router connected to PE 1 forwards the wavelet to the right *and* sends it up the ramp towards PE 1. This demonstrates the *multicasting* capability of the network.

of the hardware architecture. Each PE consists of a router connected to a processor with 48KB of dedicated SRAM memory. In each cycle, a PE can read up to 128 bits from memory and write up to 64 bits. It can also execute up to 8 16-bit operations per cycle. The router manages 5 bidirectional links: 4 between the neighboring routers and a *ramp* link connected to the processor. A link has a bandwidth of 32 bits/cycle in each direction. Data is sent in 32-bit packets called *wavelets*. A wavelet can travel any link in a single cycle, but it takes T_R cycles between when it enters the router and when an instruction can be issued by the processor using that wavelet and similarly between after a send instruction completes and when the resulting wavelet enters the router. T_R is a small value, which Tramm et al. [52] say to be around 7. We found it to equal 2 by inspection of the cycle-accurate simulator, which models ideal conditions.

Routing. Each wavelet is assigned a *color*, which determines its routing. When a router receives a wavelet, it selects in which directions to forward the wavelet based on the current configuration for the color the wavelet arrived from. Routers support *multicast*, which allows them to duplicate a wavelet and send it in multiple directions at no additional cost. If a wavelet arrives at a router from some direction from which the router is currently not accepting wavelets, it stalls until the routing configuration changes accordingly. If two wavelets arrive at a router on the same color in the same cycle, the behaviour is undefined. For each color, a router stores up to four routing configurations. Initially, one of those is active. Control wavelets allow cycling through those configurations. For more flexibility, a *teardown* wavelet on a color allows re-configuration from scratch. A normal wavelet can also advance the routing configuration to receive from a different direction. See Figure 3 for an example on how routing configuration changes in order to receive vectors from two different PEs.

Dataflow. The Cerebras chip has a dataflow architecture [55]. Tasks can be activated by wavelets of a specific color that arrive at a PE. This means that the order of tasks may differ depending on the order in which the wavelets arrive. Tasks may also be activated at compile time or by other tasks. Most of the operations are described using Data Structure Descriptors (DSDs), a way to describe certain vectorized operations. DSDs represent some part of memory or a sequence of incoming or outgoing wavelets. By using DSDs, we can simplify repeated operations down to a single hardware instruction.
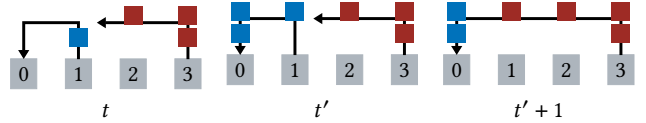

Figure 3: Synchronization on the WSE occurs through routing configurations. In cycle t, router 1 is configured to forward the blue wavelets it gets from PE 1 towards PE 0. As a result, the red wavelets from PE 3 stall at router 2. At cycle t', the last element of the vector from PE 1 arrives at the router 1. This triggers a change in routing configuration, such that in cycle $t' + 1$ the red wavelets are propagated towards PE 0.

Table 1: Summary of our Performance Model.

Symbol	Description
E	**Energy** is the *total* number of hops the network needs to route wavelets for. I.e., every wavelet m that travels E_m hops increments the energy by E_m.
L	**Distance** is the largest number of hops a wavelet has to travel.
D	**Depth** is the longest sequence of PEs that perform operations that depend on each other's output.
C	**Contention** is the largest number of wavelets a PE sends/receives.
N	**Number of links** being used overall
T_R	**Ramp latency** from a processor to its router
P	**Number of PEs**
B	**Vector length** in wavelets

3 PERFORMANCE MODEL

To effectively design algorithms, it is paramount to be guided by a performance model. An ideal performance model should be accurate and straightforward to evaluate, facilitating quick design iterations without extensive implementation and measurement.

We base our performance model on the *spatial computer* model, which provides a general framework for assessing algorithm performance in a disaggregated on-chip setting [17]. For cycle-level predictions, we parameterize the model to the properties of the WSE. The model isolates the individual contributions to the application performance. Then, the individual contributions are aggregated into a runtime estimate. This approach allows us to identify and name the bottlenecks of a given algorithm and simplifies the analysis. Table 1 explains the the individual cost terms that contribute to the model, namely *depth*, *distance*, *contention*, and *energy*.

Intuitively, a high depth means that the computation is highly sequential. A large distance means a higher communication latency. A high contention means that wavelets might stall at the contended PE. Lastly, a high energy indicates that the network might become congested. By reducing all these contributing costs, we can obtain a high-performance algorithm.

We synthesize the spatial cost metrics into an estimate for the cycle count for the WSE. Receiving and sending a wavelet costs $2T_R$ cycles to go down and up the ramp. Additionally, it costs 1 cycle to

store the received element. Hence, the number of cycles increases by $(2T_R + 1)$ times the depth D. Note that on the WSE-2, T_R is about 2 cycles on average. Moreover, a wavelet needs at least L cycles to travel the distance L. Next, observe that with a bandwidth of 1 element per link per cycle, using N links it takes at least E/N cycles to route a communication pattern with energy E. Finally, when a PE experiences contention C, it takes at least C cycles to receive those elements. We observe that when PEs experience high contention relative to the congestion in the network, the system approaches the behavior of a pipeline, where in each of the C cycles one element arrives at each PE. In this case, the congestion and latency in the network becomes negligible, as the system will stall at the contended PE.

Synthesizing these observations, we propose the following estimate T for the number of cycles on a grid using N links:

$$T = \max\left(C, \frac{E}{N} + L\right) + (2T_R + 1)D \ . \tag{1}$$

Note that the number of links being used should be determined based on the algorithm at hand. For example, if only the links in one direction are used, those should be used for estimating the contribution of energy to the number of cycles.

Equipped with a performance model, we now dive into the design and analysis communication collectives. We begin with the 1D case where we are operating on a part of a row or column of the device. This case is important in its own right for applications such as GEMV [50]. In Section 7, we consider the general 2D case.

4 1D BROADCAST

A broadcast is a simple, yet ubiquitous communication collective which can be used to build an AllReduce from a Reduce. A crucial feature of the WSE is the multicast support. It greatly simplifies operations where all PEs end up with the same data. We show with our model that this allows us to perform a broadcast with the same runtime as sending a message.

4.1 Message

The simplest communication primitive is sending a vector of length B to some PE. Consider a setting where we are sending a vector from the rightmost to the leftmost PE in the row.

The depth when sending a message is 1, the distance is $P - 1$, the energy $B(P - 1)$, and the contention is B. The number of links is $P - 1$. Hence, we conclude:

$$T_{\text{MESSAGE}} = B + P + 2T_R$$

Note that this cost is optimal for sending such a message as there is no benefit of using the links that point towards the sender, there is no benefit in using other PEs, and so there is a single path.

4.2 Flooding Broadcast

We consider a broadcast where the root is the rightmost PE. It is implemented by sending a message from the rightmost to the leftmost PE. Each router is configured such that it duplicates the message and sends it to the corresponding processor as well as in the direction of the leftmost PE. See Figure 4 for an illustration. We model the runtime as:
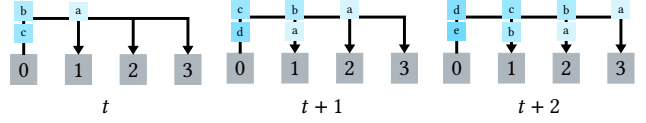

Figure 4: Broadcast for 3 consecutive cycles. This example showcases both pipelining and multicasting.

LEMMA 4.1. $T_{BCAST} = B + P + 2T_R = T_{MESSAGE}$

PROOF. The depth of the broadcast is 1, the distance is $P - 1$, the energy $B(P - 1)$, and the contention is B. Since the messages are sent only towards the root, the number of links is $P - 1$. □

Because broadcasting sends the message to each PE, the optimality of flooding follows from the lower bound on messaging.

5 1D REDUCE

Guided by our performance model, we analyze the tradeoffs of different Reduce patterns. In this section, we will focus on reduction to the leftmost PE in a single row of PEs. We show how to generalize these ideas to the full 2D grid in Section 7. We first discuss two patterns which have already been introduced in previous works. We then introduce two of our own patterns.

5.1 Star Reduce

If we minimize the depth, we get the following algorithm: every PE sends its vector directly to the root. See Figure 5a for an illustration. This pattern has been used as part of a stencil computation algorithm for the CS-1 [44]. We can model its performance as follows:

LEMMA 5.1. $T_{STAR} \leq \max\left(B(P - 1), \frac{P}{2}B + P - 1\right) + 2T_R + 1$

PROOF. The depth is 1, because messages go from each PE to the root directly. The distance is $P - 1$, because the message from PE P to PE 1 needs $P - 1$ hops. Each PE sends B messages to PE 1. Each message from PE i will need $i - 1$ hops, which leads to $B\frac{P(P-1)}{2}$ energy. The contention is $B(P - 1)$ at PE 1. □

We can actually find here a better performance prediction than our model would suggest. For the case $B = 1$, the direct upper bound predicts that the energy plus distance term would dominate. However, a closer look reveals that actually there is no congestion in the network in this case. Instead, the communication forms a perfect pipeline. Hence, the runtime is still $P - 1$, rather than $\frac{3P}{2} - 1$. We conclude that

$$T_{\text{STAR}} = B(P - 1) + 2T_R + 1 \ .$$

From the model, we expect this pattern to perform well when reducing a scalar, i.e., $B = 1$. In this case, the runtime approaches the distance lower bound $P - 1$.

5.2 Chain Reduce

A lot of distributed applications require reduction of longer vectors, where Star-Reduce would be very inefficient. We could use the chain pattern for that, which is currently implemented as part of the existing collectives library [58] and used in Cerebras' matrix multiplication algorithm [50]. Every PE sends its vector to its left
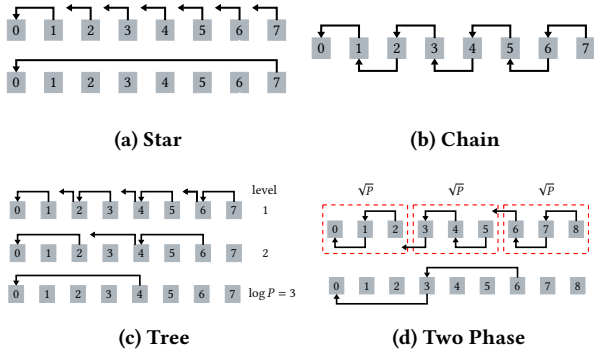
Figure 5: Routing configurations for 1D Reduce schemes. Each row shows a configuration. When a PE has sent all its data, it switches to the next configuration. Observe that every path is set up to process a vector of elements in a *pipeline*. However, if a router is not ready yet to forward data because its PE is still in a previous configuration, this will stall the preceding PE. In this way, the operation is loosely synchronized between configurations.

neighbor, forming a chain as shown in Figure 5b. The operation is pipelined, i.e., when a PE is receiving wavelets it is also sending out the already processed ones. We will later show that this pattern is optimal for very large vectors.

The pattern uses two colors. Every PE receives on the red color and sends them out on the blue color. Routing decisions cannot depend on where a wavelet came from. If we had only one color, we would need to treat the wavelets coming from the RAMP differently than the ones coming from the EAST (see also Figure 2).

LEMMA 5.2. $T_{CHAIN} = B + (2T_R + 2)(P - 1)$

PROOF. The depth is $P - 1$, because PE i can only start sending messages after it receives them from PE $i + 1$. The distance is $P - 1$ as it is the number of hops from PE P to PE 1. Each PE $(i + 1)$ sends B messages to PE i. This requires 1 hop per message, which leads to $(P - 1)B$ energy. Every PE receives B messages from its right neighbouring PE, which results in B contention. □

The Chain-Reduce shines for vector lengths $B \gg T_R P$, when its runtime approaches the contention lower bound B.

5.3 Tree Reduce

The main issue with the Chain Reduce is that the runtime increases linearly with the number of PEs. Therefore, we propose a binary tree reduction pattern. We assume in our description that the number of PEs is a power of two. This assumption can be easily removed. The reduction proceeds in $\log P$ rounds. Initially, all PEs are *active*. In every round, every second active PE sends a message containing its partial result to the previous active PE and then becomes inactive. This way, we halve the number of active PEs in every round until the root holds the result. See Figure 5c for an illustration.

Note that the router configuration changes during the execution, is achieved using control wavelets. An active PE that is sending wavelets has a router configuration to receive from the RAMP and propagate to the WEST. Then, when it has sent everything out and

becomes inactive, it switches the configuration to receive from the EAST and propagate to the WEST.

LEMMA 5.3.

$$T_{TREE} = \max\left(B \log_2 P, \frac{B \cdot P}{2(P - 1)} \log_2 P + P - 1\right) + (2T_R + 1) \log_2 P$$

PROOF. For a Tree-Reduce on P processors, where P is a power of two, the depth is $\log_2 P$ because we halve the number of active PEs each round. The distance is $P - 1$. In the i-th round, we have $\frac{P}{2^{i-1}}$ active PEs. Half of those send B wavelets that travel a distance of 2^{i-1}. The energy round i is therefore $\frac{PB}{2^i} 2^{i-1} = \frac{PB}{2}$. Because we have a total of $\log_2 P$ rounds, the energy is $\frac{B}{2} P \log_2 P$. The root will receive B messages in each round which leads to $B \log_2 P$ contention. □

The tree pattern overcomes the large depth of the Chain Reduce. However, it comes at the cost of a significantly increased contention. This becomes an issue for large vector lengths.

5.4 Two Phase Reduce

We introduce an approach that combines the beneficial aspects of the Tree and the Chain pattern, namely low depth and low contention. The algorithm is parameterized by the group size S and has two phases. In the first phase, we perform Chain-Reduce in groups of S consecutive PEs. Only the leftmost PE in each group participate in the second phase. In that phase, we perform Chain-Reduce on the remaining $\lceil \frac{P}{S} \rceil$ PEs. It is important that we assign the groups from the end, i.e., starting from p_{P-1}. See Figure 5d for an illustration of the approach. A choice of $S = \sqrt{P}$ reduces the depth and energy costs. Hence, we use this choice of S throughout.

LEMMA 5.4. *When $P = S^2$, we have:*

$$T_{TwoPHASE} \leq \max\left(2B, 2B - 2\frac{B}{\sqrt{P}} + P\right) + (2\sqrt{P} - 2) \cdot (2T_R + 1)$$

PROOF. The depth is $S - 1 + \lceil \frac{P}{S} \rceil - 1$. Executing chain reduce on S PEs has depth $S - 1$. In the second phase, we have $\lceil \frac{P}{S} \rceil$ PEs left, which again requires $\lceil \frac{P}{S} \rceil - 1$ depth with chain reduce. The energy of the first phase equals $\lceil \frac{P}{S} \rceil$ times that of a chain pattern on S PEs, that is, $(S - 1)B\lceil \frac{P}{S} \rceil$ energy. In the second phase, we have $\lceil \frac{P}{S} \rceil - 1$ vectors of length B that travel S hops. This totals $SB(\lceil \frac{P}{S} \rceil - 1)$ energy. When $S = P^2$, the energy of each phase simplifies to $PB - B\sqrt{P}$. The result follows since there are at most P links active. □

We observe that the two phase pattern has a contention that is only a factor 2 worse than the chain reduce, while vastly reducing the depth from $P - 1$ to $2\sqrt{P}$. Hence, we expect it to perform well for intermediate ranges of vector sizes.

5.5 Auto-Gen Reduce

Our model reveals that none of the existing algorithms provide a consistent performance throughput the whole range of vector lengths B and number of PEs P. In this section, we provide a method that achieves good performance across the board.
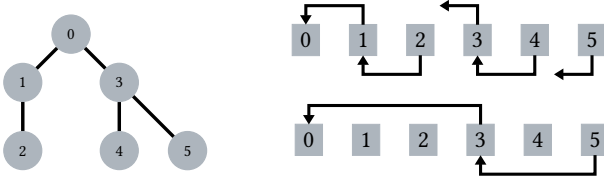
Figure 6: Reduction tree labelled in pre-order and the corresponding Auto-Gen Reduce routing configuration.

This Reduce algorithm generates a different reduction tree for a given set of input sizes and PE counts. We call this algorithm the Auto-Gen algorithm, because it traverses an automatically generated reduction tree in pre-order. Each tree stored in pre-order represents some reduce execution. In such an execution, each vertex, representing a unique PE, receives messages from its children in-order. During the execution, each PE can send messages only to one other PE. Moreover, we do not allow for overlapping communication edges. This means that if PE 3 is sending messages to PE 0, then PE 4 can send messages to neither PE 1 or PE 2. See Figure 6 for an illustration. Note that this general approach generalizes every algorithm we have presented so far. For example, a Star reduce is represented by a star graph and a Chain reduce by a path. Hence, by finding the optimal tree, we can guarantee to match or outperform those fixed algorithms.

Let $E_{\text{AUTO-GEN}}(P, B, D, C)$ be the minimum energy of Auto-Gen reduce with P PEs, B vector length, D depth and contention C. We can calculate it recursively as:

$$\min_i E_{\text{AUTO-GEN}}(i, B, D, C - 1) + E_{\text{AUTO-GEN}}(P - i, B, D - 1, C) + i$$

The root PE needs to receive at least one message. Let the last message it receives be from PE $i + 1$. When it receives that message, it needs to already have the sum of i PEs. This needs to be done with contention at most $C - 1$, since it will receive one more message. The message sent from PE $i + 1$ needs to have the value of reduce on $P - i$ rightmost PEs. This can be done with depth at most $D - 1$, because it is followed by sending a message. We can now calculate $T_{\text{AUTO-GEN}}(P, B)$ as:

$$\min_{(D,C)} \max\left(C, \frac{E_{\text{AUTO-GEN}}(P, B, D, C)}{P - 1} + P - 1\right) + D(2T_R + 1)$$

We add $P - 1$, because the message from the rightmost PE needs $P - 1$ hops. We divide the energy by $P - 1$, because we assume that messages are sent towards the root.

To generate the code for the Auto-Gen reduce, we first compute the best pre-order tree via backtracking. We can compute this optimal tree in $O(P^4)$ by finding the lowest energy tree according to the formula for $T_{\text{AUTO-GEN}}$. Based on the tree we compute the routing configuration for each PE. This includes the colors on which the wavelets are sent and received. We need to compute whether a PE should be sending a control wavelet in order to update the routing configuration of the receiver. We implement Auto-Gen by providing a python program which computes the optimal tree and generates the code with the routing and PE code.

5.6 Lower Bound

We present a lower bound on the cost of 1D Reduce for a broad class of algorithms in our model. The idea is to bound the energy required for a given depth recursively. We can then find the best algorithm by considering every possible depth and summing all cost terms. We assume that PEs send messages in the direction of the root. However, we allow for a PE to send one wavelet to PE i and another to PE j.

Let $T^\star(P, B)$ be the minimum time it takes to Reduce a vector of length B between P consecutive PEs. Let $E^\star(P, B, D)$ be the minimum energy needed for this reduction using depth at most D.

LEMMA 5.5.

$$E^\star(P, 1, D) \geq \min_{0 < i < P} E^\star(i, 1, D) + E^\star(P - i, 1, D - 1) + \min(i, P - i + 1)$$

PROOF. In order to decompose the energy term, we consider a generalized problem, where the distance between the j-th and $j + 1$-th PE is some integer $s_j \geq 1$. Then, $E^\star(i, 1, D, S)$ denotes the energy to perform an optimal reduce on i PEs with depth at most D where S denote the sum of the distances s_j. Note that $S = i - 1$ if and only if all PEs are neighbouring. Let now $S = i - 1 + k$, we want to show that $E^\star(i, 1, D, S) \geq E^\star(i, 1, D, i - 1) + k = E^\star(i, 1, D) + k$. To see that the first inequality holds, consider the pattern used in $E^\star(i, 1, D, S)$. If we made all PEs neighbouring, i.e., decreased the total distance by k, the energy would need to decrease by at least k. This is because each link in the reduce (in the direction of the root) needs to be used at least once. By decreasing the sum of distances by k we are shortening the links by a total of k. Since each of such link was used at least once, they contribute at least k to the energy. The second equality holds because when all PEs are neighbouring we are in the usual cost setting.

Now, we are ready to derive the main recursion of the lemma. Let the last message received by the root contain a partial sum of $P - i$ PEs for some i. Since this is the last message, the root must already have a partial sum of i PEs. To reduce i PEs with depth D, we need at least $E^\star(i, 1, D)$ energy. The reduction of $P - i$ PEs needs to be done with depth at most $D - 1$, which needs $E^\star(P - i, 1, D - 1)$ energy. Let S_3 be the energy of the last message. Then, for some total distances S_1 and S_2 we have:

$$E^\star(P, 1, D) = E^\star(i, 1, D, S_1) + E^\star(P - i, 1, D - 1, S_2) + S_3 \ .$$

Using our previous observation, we get

$$E^\star(P, 1, D) \geq E^\star(i, 1, D) + E^\star(P - i, 1, D - 1) + S_1 + S_2 + S_3 - (P - 1) \ .$$

It remains to bound $S_1 + S_2 + S_3$. Let $x_{1,1}$ and $x_{1,2}$ be the leftmost and rightmost PEs in the first reduction, respectively. Define $x_{2,1}$ and $x_{2,2}$ similarly for the second reduction. Observe that:

$$S_1 + S_2 + S_3 = (x_{1,2} - 1) + (x_{2,2} - x_{2,1}) + x_{2,1}$$

We know that $x_{1,2} \geq i$ and $x_{2,2} \geq P - i + 1$. We know that either $x_{1,2} = P$ or $x_{2,2} = P$. Hence, we conclude that

$$S_1 + S_2 + S_3 = P - 1 + \min(i, P - i + 1) \ .$$

□

We can compute the energy for reducing a scalar on P PEs with depth at most D in $O(P^2)$ with a dynamic programming approach. We use this result to bound the optimal runtime $T^\star$:

$$T^\star(P,B) \geq \min_D \frac{E^\star(P,B,D)}{P-1} + P - 1 + D(2T_R+1)$$
$$\geq \min_D B\frac{E^\star(P,1,D)}{P-1} + P - 1 + D(2T_R+1)$$

The first inequality follows because we can omit contention when calculating the lower bound. The second inequality follows because the energy to Reduce a vector of length B needs to be at least B times the energy to Reduce a scalar. Solving the dynamic program takes $O(P^3)$ time.

5.7 Comparison

We run the predictions of the Auto-Gen reduce and compare them against the lower bound and the fixed patterns. Figure 1 compares each pattern against the lower bound. Star-Reduce is effective at $B = 1$, Tree-Reduce is effective for slightly larger B, and Chain Reduce excels for large B. Finally, the Two-phase pattern is effective for intermediate vector sizes, that is when $P \approx B$. We can see that each pattern outside of its ideal range is often at least 3x worse than the best one. The Two-Phase pattern performs quite well throughout the whole range, although it is up to 2.4x away from the lower bound. Our Auto-Gen reduce strictly dominates all other patterns and is at most 1.4x away from our lower bound.

6 1D ALLREDUCE

We now consider different AllReduce patterns and analyze them using our performance model. We focus on AllReduce in a single row or column, but show in Section 7 how to generalize those ideas to the whole 2D grid.

6.1 Reduce-then-Broadcast

We first consider a Reduce-then-Broadcast implementation of AllReduce. Let us assume that we use a reduction pattern REDUCE. The total predicted runtime is simply

$$T_{\text{Naive}} = T_{\text{REDUCE}} + T_{\text{BCAST}} \ .$$

Note that this naive implementation could be further optimized by choosing an optimal root to reduce to. We could choose it based on our performance model. This is done in optimized stencil implementations [25], in which they first reduce to the middle PE and broadcast from there.

6.2 Ring AllReduce

The main issue with a Reduce-then-Broadcast approach is that the root receives the whole vector and then sends it out, which has a runtime of at least $2B$ because of contention. This is suboptimal for larger vector lengths.

To address this problem, we consider the ring AllReduce [1, 21] pattern. Because the network is a mesh and not a torus, we cannot have a ring in which a PE communicates with its nearest left and right neighbours. Instead, we propose two different mappings, which as we show result in the same predicted performance. The simplest way to map a ring is to to have each PE receive from its left neighbour and send to its right neighbour. Since the rightmost PE does not have a right neighbour, it sends a message to the leftmost PE in the row. See Figure 7a for an illustration. A problem with
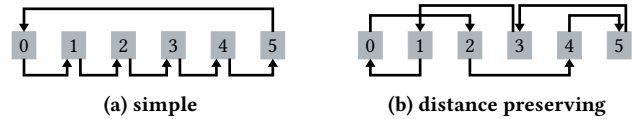

(a) simple   (b) distance preserving

Figure 7: Different ring pattern implementations.

this design could be that the longest link is a bottleneck. We can also map a ring such that a PE will be communicating with PEs at a distance of at most two. See Figure 7b for an illustration. Notice that in both patterns we are utilizing bidirectional links.

The ring AllReduce first performs $P-1$ rounds of reduce-scatter, after which each PE has a $\frac{B}{P}$ elements of the final vector. It then executes $P-1$ rounds of allgather, after which each PE has the final vector. Let us assume that B is divisible by P. In each round a PE sends a vector of length $\frac{B}{P}$ and receives a vector of length $\frac{B}{P}$.

LEMMA 6.1.
$$T_{RING} = \frac{2(P-1)B}{P} + 4P - 6 + 2(P-1)(2T_R+1)$$

PROOF. We analyse the two mappings together. The depth is $2(P-1)$, because each round depends on the previous one. In the first $P-1$ rounds a wavelet traverses the whole ring minus one link. Since the ring maps $2(P-1)$ links and we have two rounds, we get a distance of $2 \cdot (2P-3)$. In each round of the algorithm, we have $\frac{B}{P}$ wavelets travelling over each link. Since we have $2(P-1)$ links and $2(P-1)$ rounds, the energy is $2(P-1) \cdot \frac{2(P-1)B}{P}$. The contention is $\frac{2(P-1)}{P}$. Notice that in this case, the number of links is $2(P-1)$ instead of $P-1$, because we are using bidirectional links. □

6.3 Comparison

Analytically, just like for reduce, we can determine the best choice of algorithm for a given B and P. We plot the result in the heatmap Figure 8. It shows which algorithm the model predicts to perform best for a given combination of vector size B and PE count P. As we could expect, different Reduce-then-Broadcast AllReduce patterns perform best for the same parameters as the underlying Reduce does. However, there is a part where the Chain+Broadcast is outperformed by the ring pattern. This is when the runtime is dominated by the contention due to a large vector length.

7 2D COLLECTIVES

We discuss how to design collectives in a 2D setting. One problem we encounter is that the CS-2 has only one port from the processor to the router. This means we cannot e.g. send one packet on the y-axis and another on the x-axis each cycle. We would need to alternate between them each cycle. This means that in contrast to other works on AllReduce [11] where usually multiple ports per dimension are considered, we benefit less from the 2D setting. However, we still show how certain collectives, such as broadcast greatly benefit from this setting. In this section we assume that dimensions of the PE grid are $M \times N = P$.

7.1 2D Broadcast

Let us first analyse a broadcast where the root has position $(0, 0)$. A 2D broadcast can be implemented by performing 1D broadcast
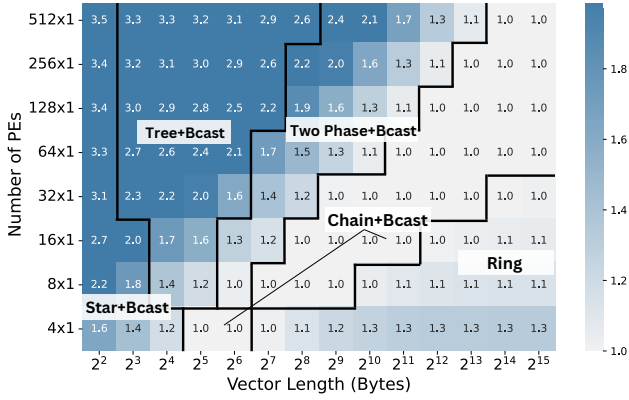
Figure 8: Speedup of *1D AllReduce* algorithm over Chain+Bcast, which is used by the vendor. The regions indicate which of the algorithms is the best fixed algorithm for the given combination of vector length and PE count.
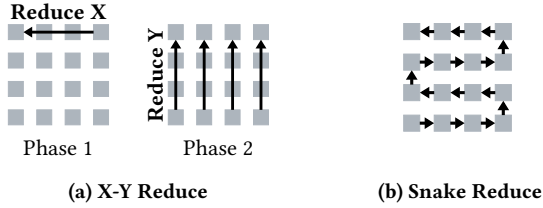

Figure 9: 2D Reduce patterns.

on the x-axis and multicasting to perform it simultaneously on the y-axis. We get the following bound:

LEMMA 7.1. $T_{2D\ BROADCAST} = B + M + N - 2 + 2T_R + 1$

PROOF. The depth is 1. The distance is $M + N - 2$. The energy is $B(P - 1)$ with $P - 1$ links used. The contention is B. □

This means that if we had P processes in a $\sqrt{P} \times \sqrt{P}$ grid, the broadcast would take $2\sqrt{P} + 2T_R - 1 + B$. This is much more efficient compared to the row broadcast on P processes. Specifically, we see that there is a lot to benefit from the 2D setting.

7.2 X-Y Reduce

The simplest approach to performing 2D reduce, would be utilizing our existing 1D implementations. We can perform reduce on the x-axis followed by reduce on the y-axis. See Figure 9a for an illustration. Then, the predicted runtime is going to be:

$$T_{REDUCE\ X} + T_{REDUCE\ Y}$$

7.3 Snake Reduce

One problem with the previous approach is that the root PE will receive the vector at least twice, which is sub-optimal for $B \gg P$. We know that in the 1D setting, the chain pattern performs best in this case. We therefore propose to map the chain implementation in a snake-like pattern. See Figure Figure 9b for an illustration. Notice that this way the runtime is going to be the same as T_{CHAIN}.

7.4 2D Allreduce

The simplest approach is executing AllReduce on the x-axis followed by AllReduce on the y-axis. Then, the predicted runtime is:

$$T_{ALLREDUCE\ X} + T_{ALLREDUCE\ Y}$$

By performing AllReduce on the x-axis and y-axis, we are essentially going to be broadcasting twice, which is very bandwidth inefficient. Remember that we have a very efficient broadcast implementation. To improve the 2D AllReduce, we could perform first a 2D Reduce and then a 2D broadcast. The runtime is:

$$T_{ALLREDUCE} = T_{2D\ REDUCE} + T_{2D\ BROADCAST}$$

7.5 Lower Bound

We provide a simple lower bound for the general 2D Reduce. Let $T^{\star}(M, N)$ be the time of an optimal Reduce on an $P = M \times N$ grid of PEs.

LEMMA 7.2. $T^{\star}(M, N) \geq \max\left(B, \frac{B}{8} + M + N - 1\right) + 2T_R + 1$

PROOF. The contention is at least B because if the root receives less than B values there is no way to construct the result. Similarly, the energy is at least PE because every PE has to send a value for each of the entries in the vector. There are at most $8P$ bidirectional links in a 2D grid of P PEs. The distance is at least $M + N - 1$ (from the bottom-right PE to the top-left PE). Finally, the depth is trivially at least 1. The result follows by combining the terms. □

When comparing with the lower bound, we see that for $B \gg P$, the snake Reduce is optimal. When the vector length B does not dominate the number of PEs P, there is room for improvement.

7.6 Discussion

Analytically, we can determine the best choice of algorithm for a given B and P. We plot the resulting heatmap in Figure 10, where we show which algorithm we predict to perform best for a given combination of vector size B and PE count P. As expected, the results are very similar to what we observed in the 1D setting. However, here the bandwidth-limited area is occupied by the ring AllReduce in 1D is replaced by the snake pattern in 2D.

8 EXPERIMENTS

We evaluate the performance of different collectives for a row of PEs as well as a square grid of PEs. For each of the collective, we perform two types of experiments. All data types are 32-bit floating point numbers. In the first experiment type, we fix the number of PEs P to be the maximum row length or grid size that is still a power of two and vary the vector length B. A large grid size caters to the fact that HPC applications need to utilize most of the PEs to achieve the best performance. The second experiment type fixes the vector length B to 256 values (1 KB) and varies the number of PEs in a row or grid. This allows us to see how performance and our predictions are impacted by the number of PEs.

8.1 Benchmarking

All benchmarks are on a CS-2 running at 850 MHz with 40GB of on-chip SRAM. We run each benchmark 5 times and plot the mean
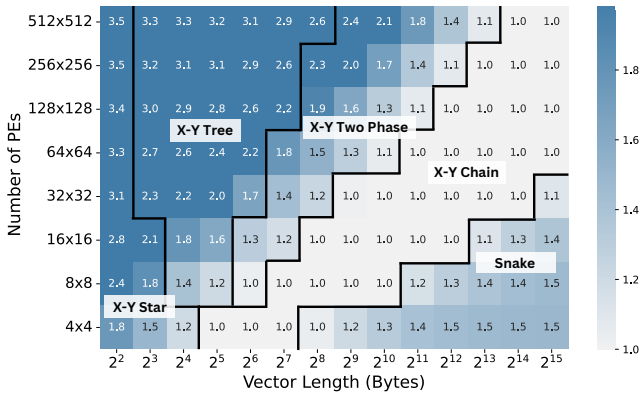
Figure 10: Speedup of *2D AllReduce* algorithm over the X-Y Chain, which is used by the vendor. The regions indicate which of the proposed algorithms is the best fixed algorithm for the given combination of vector length and PE count.

runtime. The observed standard deviation is negligible (< 4%) [20]. Five measurements is enough because the CS-2 is mostly deterministic unlike traditional architectures. There is no caching and memory access time is deterministic. The time to travel between the routers is also always 1 cycle. One source of non-determinism is that PEs may insert no-ops to regulate thermal stress of the wafer. Additionally, although the cores all run at around 850 MHz, they are truly independent cores, with independent clocks. For these reasons we still see some deviation from the mean. The performance will also depend the specific CS-2 chip. If there are any defects, a proprietary process will route around them.

8.2 Implementation.

We implement all algorithms with the newest version of the Cerebras SDK 1.0 [48] and runtime. For our Auto-Gen reduce, we compute the necessary parameters in Python. Based on that we generate the source code for each PE. Note that we provide our own (equivalent) implementation of Chain Reduce instead of the one provided in the SDK. A library call would cause reconfiguration of the routers which would yield artificially slow results.

When implementing collectives for the CS-2 it is important to limit the number of colors as there are only 24 of them. Our 1D implementations utilize up to 3 colors, while the 2D implementations use up to 5. When using our collectives, the rest of the colors would be available to the application. The routing for the CS-2 is requires avoiding race conditions. Having two wavelets arrive at a router on the same color in the same cycle leads to undefined behaviour. To avoid this, we configure the routers such that at a given cycle they accept wavelets only from a single direction. We do this using control wavelets which alter the routing configuration at runtime.

8.3 Time Measurements

Time measurements in distributed systems have been extensively studied [23, 60] and pose challenges, such as lack of a shared clock. A proper measurement methodology is particularly important for the

sub-microsecond runtimes we observe. For broadcast, the methodology is easier because the computation starts from a single root, which synchronizes the start time. However, for Reduce we need to synchronize the clocks and ensure that each processor starts at the same time. For all measurements, we start by performing a Reduce to the PE at position $(0, 0)$. This constitutes a barrier and ensures that all processes have no other ongoing computations.

To measure 1D broadcast performance, we use a ping-pong like approach. We execute a broadcast from the leftmost PE, then from the rightmost PE. We repeat this procedure k times and report the end clock time - start clock time at the leftmost PE divided by $2k$.

For Reduce and AllReduce we need a different approach, because the execution might start at multiple PEs. Hence, we need to ensure that all PEs start execution at the same time. We perform several calibration runs until a synchronized clock tells us that all PEs start at roughly the same time. The calibration adjusts a so-called *wait parameter* α until the condition is satisfied. Initially, $\alpha = 1$.

First, the PE at position $(0, 0)$ performs a broadcast that triggers each PE at position (i, j) to sample its local reference clock, called $T_R(i, j)$. Now, each PE (i, j) executes $\alpha(M + N - i - j)$ writes to an empty memory location. After that, each PE samples the start clock $T_S(i, j)$, performs the collective and samples the end clock $T_E(i, j)$. For each PE we calibrate the start and end clock as follows:

$$T_S(i, j)' = T_S(i, j) - (T_R(i, j) + (i + j + 2))$$
$$T_E(i, j)' = T_E(i, j) - (T_R(i, j) + (i + j + 2))$$

This accounts for the difference in time when a PE samples the reference clock, which is $i + j + 2$ for a PE at position (i, j) as we use a broadcast to initiate the sampling. We adjust the wait parameter α and repeat the calibration until the difference in calibrated start times $\max_{i,j} T_S(i, j)' - \min_{i,j} T_S(i, j)'$ is small enough. We obtain a start difference below 57 cycles for 1D and 129 cycles for 2D Then, the final measurement is $\max_{i,j} T_E(i, j)' - \min_{i,j} T_S(i, j)'$. This methodology ensures accurate measurements in the wafer-scale setting. In an ideal system $\alpha = 1$ would make all PEs start at the same time since each write takes 1 cycle. However, in order to prevent the PE from overheating, the machine will start inserting no-ops, which we need to adjust for.

8.4 1D Broadcast

Scaling Vector Length. Figure 11a shows the broadcast results for 512 PEs and increasing vector length. As expected, for small vector lengths the runtime is dominated by the distance. Hence, the runtime only grows slowly with the vector length. For vector lengths larger than 512 bytes, the runtime grows roughly linearly with the vector length. The model matches the predictions closely, with a relative error of at most 21%.

Scaling PE count. Figure 12a shows the results for fixed vector length of 1 KB and increasing number of PEs. Our model predicts the correct trend, with a large initial runtime that accounts for sending the message (contention or energy term) and a gradually increasing contribution of the distance term. The relative error of the prediction is 8%-21%.
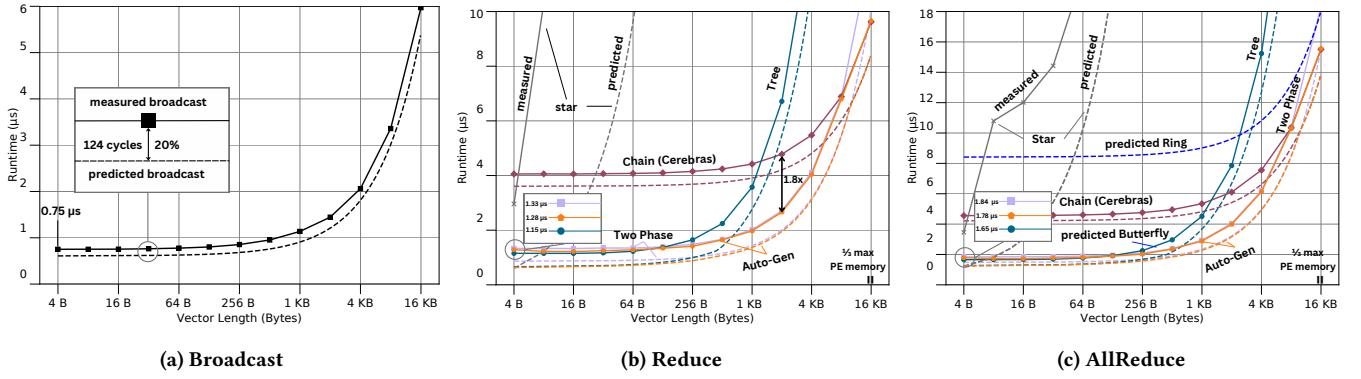
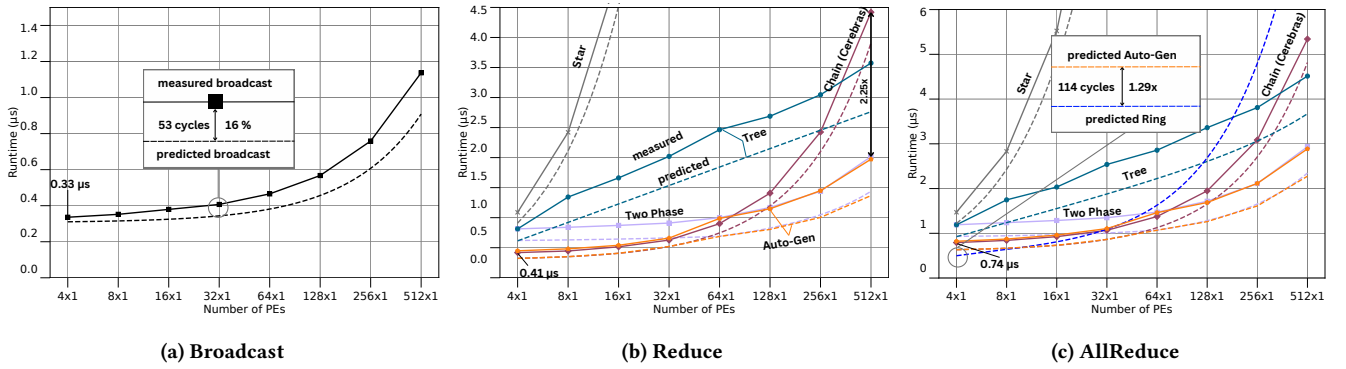**Figure 11: Benchmarks for 1D row of 512x1 PEs and increasing vector length.**

Figure 12: Benchmarks for fixed vector length of 1 KB and increasing number of PEs.

8.5 1D Reduce

Scaling Vector Length. We evaluate Reduce for 512 PEs with increasing vector length in Figure 11b. As predicted, among the fixed implementations, low depth patterns like the tree excel for small vector lengths, while those with greater depth, like the chain pattern, show inferior performance. With increasing vector lengths, energy and contention begin to outweigh depth in impact, resulting in the Two-Phase pattern outperforming others. Finally, for the largest vectors, the contention dominates the runtime and the chain pattern performs best.

The only exception to the above mentioned predictions is the star pattern. It performs worse than predicted in this scenario. This is likely because of the overhead associated when a PE starts receiving from another PE. Because the star pattern receives elements from all PEs, this is much more pronounced than for other implementations, especially for large number of PEs. Except for scalars, our experiments and model both suggest that other low-depth patters, such as Tree or our Two-Phase algorithm, are faster.

Overall, our Auto-Gen Reduce is the fastest pattern except when reducing a scalar. There, it is slower by at most 110 cycles. It outperforms the chain pattern, based on the current library implementation, by up to 3.16x. This further confirms the choice of our model as a tool for automatic performance tuning.

With mean relative error per pattern ranging from 12% to 35%, the general performance trends are well captured by our model. When choosing two different patterns, our model is able to very accurately predict which of the two performs best for the given vector length and number of PEs. In the cases where it mispredicts, the difference is at most 114 cycles. This means that, even if the model's predictions are not perfect, the chosen algorithm remains highly competitive and close to the best possible performance.

Scaling PE count. Additionally, we evaluate the Reduce operation for a fixed vector length of 1 KB across an increasing number of PEs, as illustrated in Figure 12b. Our model correctly predicts that initially the chain pattern performs best because with very few PEs, contention has a larger impact than the depth. With increasing number of PEs, the depth becomes more significant, and, as expected, the two phase pattern performs better.

Just as for the results with the fixed number of PEs, we see that our Auto-Gen Reduce implementation is the fastest throughout. Two-Phase offers similar performance as Auto-Gen for 64 or more PEs. Interestingly, predictions for the star pattern have high accuracy, with a 10% minimum relative error. This is likely because the runtime is dominated by the vector length rather than the number of PEs in this case. Overall, our model predicts performance trends accurately, with a mean relative error between 13% and 28%.
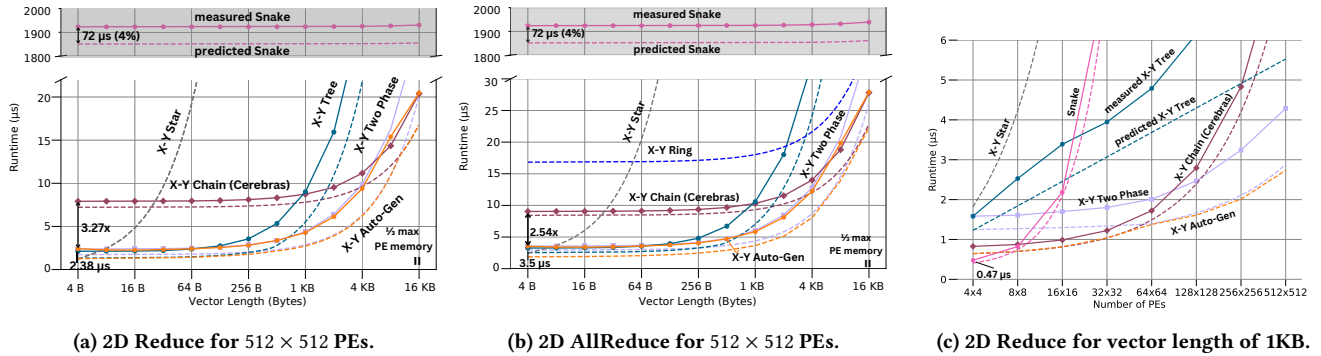
(a) 2D Reduce for 512×512 PEs.

(b) 2D AllReduce for 512×512 PEs.

(c) 2D Reduce for vector length of 1KB.

Figure 13: 2D Reduce and AllReduce benchmarks.

8.6 1D AllReduce

Scaling Vector Length. We evaluate AllReduce for 512 PEs and increasing vector length, see Figure 11c. As expected, for the reduce-then-broadcast AllReduce implementations the runtime increases by the cost of performing a broadcast with respect to the corresponding Reduce variant. We can draw the same insights as from the Reduce results, e.g., how the optimal pattern changes based on the vector length. Our Auto-Gen AllReduce gains a 2.47x improvement over the chain-then-broadcast approach, which the current library is based on. Additionally, just like for Reduce, our model accurately predicts performance trends and which pattern will perform best.

We also plotted predicted performance for ring pattern. We have already observed that our model accurately predicts which algorithm performs best. Even accounting for a 15% prediction error, the largest observed overall, the ring algorithm is never be the best choice. Hence, we refrain from providing an implementation. This underscores the utility of our model is saving engineering effort on sub-optimal and unpromising approaches. Moreover, we see that algorithms designed for the traditional distributed memory setting do not translate well into the wafer-scale setting, where it is important to leverage multicasting and pipelining.

Scaling PE count. Moreover, we evaluate AllReduce for a fixed vector length of 1 KB and increasing number of PEs, see Figure 12c. Again, for the reduce-then-broadcast implementations, we observe similar results as for Reduce. We see that for 4 PEs, the predicted ring performance is a bit better than the chain AllReduce. However, the expected performance gain is not significant. For numbers of PEs larger than 8, we see that the reduce-then-broadcast implementations would perform significantly better than the ring, outperforming it by possibly even 1.4x. This further shows how powerful the multicast feature for the CS-2 is.

8.7 2D Collectives

Scaling Vector Length. We also evaluate our implementations on the full chip of 512×512 PEs and increasing vector length. For Reduce, the results can be found in Figure 13a. The performance trends are as predicted. They are similar to to 1D setting. Our X-Y Auto-Gen Reduce outperforms the X-Y Chain by up to 3.27×.

The predictions for the snake pattern are at most 10% off. As expected, it performs very poorly. This is because of its linear depth

in the number of PEs, which is over 200'000 for this experiment. Interestingly, these results indicate that $T_R = 2$ on average. Any other choice of T_R would lead to significantly worse predictions.

Moreover, we have a relative error that is very similar to what we were seeing for 1D. The predictions are slightly worse because when we execute X-Y Reduce, after reducing on the X-axis, we need to load some values into registers which adds additional overhead. Again, our model is able to predict very well which pattern will perform best for a given vector length.

Figure 13b shows the 2D AllReduce results for fixed number of PEs and increasing vector length. The relative error of our predictions remains almost the same. Moreover, our X-Y Auto-Gen AllReduce implementation outperforms the X-Y Auto-Gen AllReduce implementation by up to 2.54×.

Scaling PE count. Just like for 1D benchmarks, we measure performance for fixed vector length and increasing 2D grid of PEs from 4×4 to 512×512. The results can be found in Figure 13c. As expected, when we have few PEs and we are bandwidth bound, the Snake pattern performs best. Then as the number of PEs grows, the X-Y Chain and finally the X-Y Two Phase pattern are best.

Our X-Y Auto-Gen reduce once again achieves good overall performance across the board. The only exception is for 4×4 PEs, where the Snake is better. This demonstrates that generating code based on our model works well both in the 1D and 2D setting.

9 RELATED WORK

9.1 Reduce on the WSE

Orenes-Vera et al. [39] introduced and implemented a wafer-scale **3D FFT algorithm**. The main communication bottleneck in their implementation is an all-to-all collective. They model the communication time by considering the most heavily congested link in the network. While the all-to-all is limited by congestion, our research demonstrates that, for the case of reduce and AllReduce, accurately modeling performance necessitates considering depth and distance, in addition to congestion factors.

Rocki et al. [44] designed a wafer-scale **SpMV stencil**. It involves an AllReduce operation at its core. They use a variant of a 2D Star AllReduce with two PEs accumulating all the results and then broadcasting them. As our evaluation shows, such an approach

is only efficient for small vector lengths because it creates large contention at the PEs that aggregate the vectors.

Hall et al. [50] designed a matrix-multiplication kernel as part of their **neural network training** stack. They utilize there the chain pattern to perform column-reduction, which as we have shown, is the best when the vector length is much larger than the number of PEs. They also perform row-reduce to specific PE by mapping the chain pattern onto a ring like in Figure 7b.

9.2 Distributed Models of Computation

The $\alpha - \beta$ **model** of parallel computation [7] considers the latency and bandwidth requirements of distributed programs. Specifically, sending a message of length m costs $\alpha + m\beta$. Unlike our model, all processors are assumed to be at the same distance to each other – the cost is independent of sender and receiver. More general models [56], such as **LogGP** [51] share the same limitations. This means that those models are unable to capture the aspects of distance and energy. These are essential for accurately estimating performance in the spatial wafer-scale setting.

The **spatial computer model** [2, 17] introduced an asymptotic model that considers energy, depth, and distance. However, it assumes that the bandwidth of the PE is of the same order of magnitude as its local memory. This does not correspond to our practical setting, where the local SRAM memory is a several thousand times larger than the bandwidth from and to the PE. This difference in the model leads to a lack of pipelining in spatial computer algorithms. As we have seen, pipelining is necessary to obtain the best performance for very large vector lengths. To address issues arising from unequal bandwidth and memory, we introduced the contention term into the model. It reflects that PE-bandwidth is a scarce resource on the device.

9.3 Developments in Accelerator Architectures

The Versal ACAP [16] is a Course-Grained Reconfigurable Array (CGRA) [8] that contains both programmable FPGA fabric and software programmable accelerators. These accelerators consist of 8×50 tiles with 48KB of local memory connected via a mesh network [45, 57], similarly to the WSE. While the programming models differ, one can observe much of the same distance-dependent performance characteristics [45]. Therefore, our model could provide a useful basis for algorithmic design on the Versal ACAP.

The SambaNova Reconfigurable Dataflow Unit (DFU) [15] is a CGRA machine learning accelerator. In contrast to the WSE, it is not based on general-purpose cores. Instead, a large grid of tiny hardware units are reconfigured to perform the given operation in a pipelined dataflow. Note that the problem of mapping operations, such as communication collectives, onto a grid of compute elements is a shared problem. Hence, there could be insights and tangible benefits of our work that carry over to the DFU.

10 CONCLUSION

We provided the first in-depth exploration of communication collectives on the Cerebras WSE. We introduced and analyzed different collective algorithms which outperform previous algorithms by up to 3.27×. Given the widespread use of these collectives in HPC

applications, our improvements promise to significantly boost computational efficiency in various scientific fields.

To achieve those improvements, we introduced a streamlined model to design algorithms for the hardware. We demonstrated that this model accurately predicts performance on the WSE. While previous works focused on communication for specific problems on the Cerebras WSE, our approach is the first to enable systematic analysis of any program on this hardware.

Our findings demonstrate that achieving optimal performance on the WSE is contingent upon automatic code generation. Manual optimizations for the WSE, hindered by the complexity of hardware features like routing, are both tedious and challenging. Our newly introduced model significantly advances the creation of effective code generators by enabling accurate performance prediction. Utilizing our model-driven approach, we generate code that achieves near-optimal performance across a broad spectrum of input sizes. This marks the first time such an approach has been successfully adapted for a wafer-scale processor.

Overall, our work enhances the performance of communication collectives and provides valuable insights into optimizing wafer-scale programs for emerging architectures like the Cerebras WSE. The accurate theoretical framework for modeling algorithms significantly advances our understanding of the hardware and its limitations. Hence, our study represents a significant advancement in unlocking the full potential of this emerging architecture and boosting the efficiency of HPC applications.

ACKNOWLEDGMENTS

REFERENCES

[1] Michael Barnett, Richard J. Littlefield, David G. Payne, and Robert A. van de Geijn. 1995. Global Combine Algorithms for 2-D Meshes with Wormhole Routing. *J. Parallel Distributed Comput.* 24, 2 (1995), 191–201. https://doi.org/10.1006/jpdc.1995.1018
[2] Yves Baumann, Tal Ben-Nun, Maciej Besta, Lukas Gianinazzi, Torsten Hoefler, and Piotr Luczynski. 2024. Low-Depth Spatial Tree Algorithms. arXiv:2404.12953 [cs.DC]
[3] Tal Ben-Nun and Torsten Hoefler. 2019. Demystifying Parallel and Distributed Deep Learning: An In-depth Concurrency Analysis. *ACM Comput. Surv.* 52, 4 (2019), 65:1–65:43. https://doi.org/10.1145/3320060
[4] Maciej Besta and Torsten Hoefler. 2014. Slim Fly: A Cost Effective Low-Diameter Network Topology. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2014, New Orleans, LA, USA, November 16-21, 2014*, Trish Damkroger and Jack J. Dongarra (Eds.). IEEE Computer Society, 348–359. https://doi.org/10.1109/SC.2014.34
[5] Maciej Besta and Torsten Hoefler. 2022. Parallel and Distributed Graph Neural Networks: An In-Depth Concurrency Analysis. *CoRR* abs/2205.09702 (2022). https://doi.org/10.48550/ARXIV.2205.09702 arXiv:2205.09702
[6] Achi Brandt and AA Lubrecht. 1990. Multilevel matrix multiplication and fast solution of integral equations. *J. Comput. Phys.* 90, 2 (1990), 348–370.
[7] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert A. van de Geijn. 2007. Collective communication: theory, practice, and experience. *Concurr. Comput. Pract. Exp.* 19, 13 (2007), 1749–1783. https://doi.org/10.1002/cpe.1206
[8] S. Alexander Chin, Noriaki Sakamoto, Allan Rui, Jim Zhao, Jin Hee Kim, Yuko Hara-Azumi, and Jason Helge Anderson. 2017. 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*. IEEE Computer Society, 184–189. https://doi.org/10.1109/ASAP.2017.7995277
[9] Benjamin Y. Cho, Jeageun Jung, and Mattan Erez. 2021. Accelerating bandwidth-bound deep learning inference with main-memory accelerators. In *International

Conference for High Performance Computing, Networking, Storage and Analysis, SC 2021, St. Louis, Missouri, USA, November 14-19, 2021, Bronis R. de Supinski, Mary W. Hall, and Todd Gamblin (Eds.). ACM, 44. https://doi.org/10.1145/3458817.3476146

[10] Sudheer Chunduri, Scott Parker, Pavan Balaji, Kevin Harms, and Kalyan Kumaran. 2018. Characterization of MPI usage on a production supercomputer. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018*. IEEE / ACM, 30:1–30:15. http://dl.acm.org/citation.cfm?id=3291696

[11] Daniele De Sensi, Tommaso Bonato, David Saam, and Torsten Hoefler. 2024. Swing: Short-cutting Rings for Higher Bandwidth Allreduce. In *21th USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA.

[12] Daniele De Sensi, Salvatore Di Girolamo, Saleh Ashkboos, Shigang Li, and Torsten Hoefler. 2021. Flare: Flexible in-Network Allreduce. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (St. Louis, Missouri) *(SC '21)*. Association for Computing Machinery, New York, NY, USA, Article 35, 16 pages. https://doi.org/10.1145/3458817.3476178

[13] Daniele De Sensi, Salvatore Di Girolamo, Kim H. McMahon, Duncan Roweth, and Torsten Hoefler. 2020. An In-Depth Analysis of the Slingshot Interconnect. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Atlanta, Georgia) *(SC '20)*. IEEE Press, Article 35, 14 pages. https://doi.org/10.1109/sc41405.2020.00039

[14] Nolan Dey, Gurpreet Gosal, Zhiming Chen, Hemant Khachane, William Marshall, Ribhu Pathria, Marvin Tom, and Joel Hestness. 2023. Cerebras-GPT: Open Compute-Optimal Language Models Trained on the Cerebras Wafer-Scale Cluster. *CoRR* abs/2304.03208 (2023). https://doi.org/10.48550/ARXIV.2304.03208 arXiv:2304.03208

[15] Murali Emani, Venkatram Vishwanath, Corey Adams, Michael E. Papka, Rick Stevens, Laura Florescu, Sumti Jairath, William Liu, Tejas Nama, Arvind Sujeeth, Volodymyr V. Kindratenko, and Anne C. Elster. 2021. Accelerating Scientific Applications With SambaNova Reconfigurable Dataflow Architecture. *Comput. Sci. Eng.* 23, 2 (2021), 114–119. https://doi.org/10.1109/MCSE.2021.3057203

[16] Brian Gaide, Dinesh Gaitonde, Chirag Ravishankar, and Trevor Bauer. 2019. Xilinx Adaptive Compute Acceleration Platform: Versal^{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*, Kia Bazargan and Stephen Neuendorffer (Eds.). ACM, 84–93. https://doi.org/10.1145/3289602.3293906

[17] Lukas Gianinazzi, Tal Ben-Nun, Saleh Ashkboos, Yves Baumann, Piotr Luczynski, and Torsten Hoefler. 2022. The spatial computer: A model for energy-efficient parallel computation. *CoRR* abs/2205.04934 (2022). https://doi.org/10.48550/arXiv.2205.04934 arXiv:2205.04934

[18] Richard L. Graham, Lion Levi, Devendar Bureddy, Gil Bloch, Gilad Shainer, David Cho, George Elias, Daniel Klein, Joshua Ladd, Ophir Maor, Ami Marelli, Valentin Petrov, Evyatar Romlet, Yong Qin, and Ido Zemah. 2020. Scalable Hierarchical Aggregation and Reduction Protocol (SHARP)TM Streaming-Aggregation Hardware Design and Evaluation. *High Performance Computing* 12151 (2020), 41 – 59.

[19] William Gropp, Ewing L. Lusk, Nathan E. Doss, and Anthony Skjellum. 1996. A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. *Parallel Comput.* 22, 6 (1996), 789–828. https://doi.org/10.1016/0167-8191(96)00024-5

[20] Torsten Hoefler and Roberto Belli. 2015. Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2015, Austin, TX, USA, November 15-20, 2015*, Jackie Kern and Jeffrey S. Vetter (Eds.). ACM, 73:1–73:12. https://doi.org/10.1145/2807591.2807644

[21] Torsten Hoefler, Tommaso Bonato, Daniele De Sensi, Salvatore Di Girolamo, Shigang Li, Marco Heddes, Jon Belk, Deepak Goel, Miguel Castro, and Steve Scott. 2022. HammingMesh: A Network Topology for Large-Scale Deep Learning. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, Dallas, TX, USA, November 13-18, 2022*. IEEE, 1–18. https://doi.org/10.1109/SC41404.2022.00016

[22] Torsten Hoefler and D. Moor. 2014. Energy, Memory, and Runtime Tradeoffs for Implementing Collective Communication Operations. *Journal of Supercomputing Frontiers and Innovations* 1, 2 (Oct. 2014), 58–75.

[23] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. 2010. Accurately Measuring Overhead, Communication Time and Progression of Blocking and Nonblocking Collective Operations at Massive Scale. *International Journal of Parallel, Emergent and Distributed Systems* 25, 4 (Jul. 2010), 241–258.

[24] Cerebras Systems Inc. 2021. Cerebras Systems: Achieving Industry Best AI Performance Through A Systems Approach. (2021).

[25] Mathias Jacquelin, Mauricio Araya-Polo, and Jie Meng. 2022. Massively scalable stencil algorithm. *CoRR* abs/2204.03775 (2022). https://doi.org/10.48550/arXiv.2204.03775 arXiv:2204.03775

[26] Nikhil Jain and Yogish Sabharwal. 2010. Optimal bucket algorithms for large MPI collectives on torus interconnects. In *Proceedings of the 24th International Conference on Supercomputing, 2010, Tsukuba, Ibaraki, Japan, June 2-4, 2010*, Taisuke Boku, Hiroshi Nakashima, and Avi Mendelson (Eds.). ACM, 27–36. https://doi.org/10.1145/1810085.1810093

[27] S. Lennart Johnsson and Ching-Tien Ho. 1989. Optimum Broadcasting and Personalized Communication in Hypercubes. *IEEE Trans. Computers* 38, 9 (1989), 1249–1268. https://doi.org/10.1109/12.29465

[28] Nicholas T. Karonis, Bronis R. de Supinski, Ian T. Foster, William Gropp, Ewing L. Lusk, and John Bresnahan. 2000. Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance. In *Proceedings of the 14th International Parallel & Distributed Processing Symposium (IPDPS'00), Cancun, Mexico, May 1-5, 2000*. IEEE Computer Society, 377–384. https://doi.org/10.1109/IPDPS.2000.846009

[29] John Kim, William J. Dally, and Dennis Abts. 2007. Flattened butterfly: a cost-efficient topology for high-radix networks. In *34th International Symposium on Computer Architecture (ISCA 2007), June 9-13, 2007, San Diego, California, USA*, Dean M. Tullsen and Brad Calder (Eds.). ACM, 126–137. https://doi.org/10.1145/1250662.1250679

[30] John Kim, William J. Dally, Steve Scott, and Dennis Abts. 2008. Technology-Driven, Highly-Scalable Dragonfly Topology. In *35th International Symposium on Computer Architecture (ISCA 2008), June 21-25, 2008, Beijing, China*. IEEE Computer Society, 77–88. https://doi.org/10.1109/ISCA.2008.19

[31] Sameer Kumar and Daniel Faraj. 2013. Optimization of MPI_Allreduce on the Blue GeneQ Supercomputer. In *Proceedings of the 20th European MPI Users' Group Meeting* (Madrid, Spain) *(EuroMPI '13)*. Association for Computing Machinery, New York, NY, USA, 97–103. https://doi.org/10.1145/2488551.2488557

[32] Sameer Kumar and Norm Jouppi. 2020. Highly Available Data Parallel ML training on Mesh Networks. *CoRR* abs/2011.03605 (2020). arXiv:2011.03605 https://arxiv.org/abs/2011.03605

[33] Ignacio Laguna, Ryan J. Marshall, Kathryn Mohror, Martin Ruefenacht, Anthony Skjellum, and Nawrin Sultana. 2019. A large-scale study of MPI usage in open-source HPC applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2019, Denver, Colorado, USA, November 17-19, 2019*, Michela Taufer, Pavan Balaji, and Antonio J. Peña (Eds.). ACM, 31:1–31:14. https://doi.org/10.1145/3295500.3356176

[34] Sean Lie. 2021. Multi-Million Core, Multi-Wafer AI Cluster. In *IEEE Hot Chips 33 Symposium, HCS 2021, Palo Alto, CA, USA, August 22-24, 2021*. IEEE, 1–41. https://doi.org/10.1109/HCS52781.2021.9567153

[35] Sean Lie. 2023. Cerebras Architecture Deep Dive: First Look Inside the Hardware/Software Co-Design for Deep Learning. *IEEE Micro* 43, 3 (2023), 18–30. https://doi.org/10.1109/MM.2023.3256384

[36] Hatem Ltaief, Yuxi Hong, Leighton Wilson, Mathias Jacquelin, Matteo Ravasi, and David Elliot Keyes. 2023. 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*, Dorian Arnold, Rosa M. Badia, and Kathryn M. Mohror (Eds.). ACM, 6:1–6:12. https://doi.org/10.1145/3581784.3627042

[37] Message Passing Interface Forum. 2021. *MPI: A Message-Passing Interface Standard Version 4.0*. https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf

[38] G. F. Oliveira, J. Gomez-Luna, S. Ghose, A. Boroumand, and O. Mutlu. 2022. Accelerating Neural Network Inference With Processing-in-DRAM: From the Edge to the Cloud. *IEEE Micro* 42, 06 (nov 2022), 25–38. https://doi.org/10.1109/MM.2022.3202350

[39] Marcelo Orenes-Vera, Ilya Sharapov, Robert Schreiber, Mathias Jacquelin, Philippe Vandermersch, and Sharan Chetlur. 2023. Wafer-Scale Fast Fourier Transforms. In *Proceedings of the 37th International Conference on Supercomputing, ICS 2023, Orlando, FL, USA, June 21-23, 2023*, Kyle A. Gallivan, Efstratios Gallopoulos, Dimitrios S. Nikolopoulos, and Ramón Beivide (Eds.). ACM, 180–191. https://doi.org/10.1145/3577193.3593708

[40] Pitch Patarasuk and Xin Yuan. 2009. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel Distributed Comput.* 69, 2 (2009), 117–124. https://doi.org/10.1016/j.jpdc.2008.09.002

[41] Jianlong Qi, Hassan Foroughi Asl, Johan Björkegren, and Tom Michoel. 2014. kruX: matrix-based non-parametric eQTL discovery. *BMC bioinformatics* 15 (2014), 1–7.

[42] Rolf Rabenseifner. 2004. Optimization of Collective Reduction Operations. In *Computational Science - ICCS 2004, 4th International Conference, Kraków, Poland, June 6-9, 2004, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 3036)*, Marian Bubak, G. Dick van Albada, Peter M. A. Sloot, and Jack J. Dongarra (Eds.). Springer, 1–9. https://doi.org/10.1007/978-3-540-24685-5_1

[43] Rolf Rabenseifner and Jesper Larsson Träff. 2004. More Efficient Reduction Algorithms for Non-Power-of-Two Number of Processors in Message-Passing Parallel Systems. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, September 19-22, 2004, Proceedings (Lecture Notes in Computer Science, Vol. 3241)*, Dieter Kranzlmüller, Péter Kacsuk, and Jack J. Dongarra (Eds.). Springer, 36–46.

https://doi.org/10.1007/978-3-540-30218-6_13

[44] Kamil Rocki, Dirk Van Essendelft, Ilya Sharapov, Robert Schreiber, Michael Morrison, Vladimir Kibardin, Andrey Portnoy, Jean-Francois Dietiker, Madhava Syamlal, and Michael James. 2020. 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*, Christine Cuicchi, Irene Qualters, and William T. Kramer (Eds.). IEEE/ACM, 58. https://doi.org/10.1109/SC41405.2020.00062

[45] Niklas Roemer. 2023-08-06. *Designing of a communication library for Versal devices using Stream-Based API*. Bachelor Thesis. ETH Zurich, Zurich. https://doi.org/10.3929/ethz-b-000635928

[46] Yousef Saad and Martin H. Schultz. 1989. Data communication in parallel architectures. *Parallel Comput.* 11, 2 (1989), 131–150. https://doi.org/10.1016/0167-8191(89)90024-0

[47] Paul Sack and William Gropp. 2015. Collective Algorithms for Multiported Torus Networks. *ACM Trans. Parallel Comput.* 1, 2 (2015), 12:1–12:33. https://doi.org/10.1145/2686882

[48] Justin Selig. 2023. *The Cerebras Software Development Kit: A Technical Overview*. Technical Report. Cerebras Systems, Inc. 8 pages. https://f.hubspotusercontent30.net/hubfs/8968533/Cerebras%20SDK%20Technical%20Overview%20White%20Paper.pdf

[49] Andrey A Shabalin. 2012. Matrix eQTL: ultra fast eQTL analysis via large matrix operations. *Bioinformatics* 28, 10 (2012), 1353–1358.

[50] Sean Lie Stewart Hall, Rob Schreiber. 2023. *Training Giant Neural Networks Using Weight Streaming on Cerebras Wafer-Scale Clusters*. Technical Report. Cerebras Systems, Inc. 34 pages. https://f.hubspotusercontent30.net/hubfs/8968533/Virtual%20Booth%20Docs/CS%20Weight%20Streaming%20White%20Paper%20111521.pdf

[51] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of Collective Communication Operations in MPICH. *Int. J. High Perform. Comput. Appl.* 19, 1 (2005), 49–66. https://doi.org/10.1177/1094342005051521

[52] John Tramm, Bryce Allen, Kazutomo Yoshii, Andrew Siegel, and Leighton Wilson. 2024. Efficient algorithms for Monte Carlo particle transport on AI accelerator hardware. *Computer Physics Communications* 298 (2024), 109072. https://doi.org/10.1016/j.cpc.2023.109072

[53] Sathish S. Vadhiyar, Graham E. Fagg, and Jack J. Dongarra. 2000. Automatically Tuned Collective Communications. In *Proceedings Supercomputing 2000, November 4-10, 2000, Dallas, Texas, USA. IEEE Computer Society, CD-ROM*, Jed Donnelley (Ed.). IEEE Computer Society, 3. https://doi.org/10.1109/SC.2000.10024

[54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[55] Arthur H. Veen. 1986. Dataflow Machine Architecture. *ACM Comput. Surv.* 18, 4 (dec 1986), 365–396. https://doi.org/10.1145/27633.28055

[56] Udayanga Wickramasinghe and Andrew Lumsdaine. 2016. A Survey of Methods for Collective Communication Optimization and Tuning. *CoRR* abs/1611.06334 (2016). arXiv:1611.06334 http://arxiv.org/abs/1611.06334

[57] Max Wierse. 2023-02. *Evaluation of Xilinx Versal Device*. Bachelor Thesis. ETH Zurich, Zurich. https://doi.org/10.3929/ethz-b-000600880

[58] Leighton Wilson. 2023. What's New in R0.6 of the Cerebras SDK. https://www.cerebras.net/blog/whats-new-in-r0.6-of-the-cerebras-sdk. Accessed: 2023-08-09.

[59] Mino Woo, Terry Jordan, Robert Schreiber, Ilya Sharapov, Shaheer Muhammad, Abhishek Koneru, Michael James, and Dirk Van Essendelft. 2022. Disruptive Changes in Field Equation Modeling: A Simple Interface for Wafer Scale Engines. *CoRR* abs/2209.13768 (2022). https://doi.org/10.48550/arXiv.2209.13768 arXiv:2209.13768

[60] Thomas Worsch, Ralf H. Reussner, and Werner Augustin. 2002. On Benchmarking Collective MPI Operations. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 9th European PVM/MPI Users' Group Meeting, Linz, Austria, September 29 - October 2, 2002, Proceedings (Lecture Notes in Computer Science, Vol. 2474)*, Dieter Kranzlmüller, Péter Kacsuk, Jack J. Dongarra, and Jens Volkert (Eds.). Springer, 271–279. https://doi.org/10.1007/3-540-45825-5_43