Algebraic Breadth-First Search on Cerebras Wafer-Scale Engine

Elia Belli belli.2006305@studenti.uniroma1.it · **Daniele De Sensi** (Advisor) desensi@di.uniroma1.it · Department of Computer Science, Sapienza University of Rome

Sapienza Università di Roma · hlc High-Performance Large-Scale Computing

1 Motivation

Graph traversal is memory bound on conventional hardware: irregular access defeats the cache hierarchy. The Cerebras WSE-3 [1] removes that problem with distributed on-chip SRAM coordinating cores over a 2D mesh.

Prior WSE work targets regular dataflow kernels, and existing SpMV kernels require data movement between host and device at each iteration.

We design a **BFS** that runs **entirely on device**. We choose BFS because it is a foundational benchmarking kernel, most notably in **Graph500**, adopting its algebraic formulation which maps onto distributed memory naturally.

2 The Cerebras WSE-3 Architecture

key specs — Cerebras WSE-3 (CS-3) [1]

Topology	2D mesh
PEs	900,000
On-chip SRAM	44 GB
SRAM / PE	48 KB
Fabric BW	214 Pb/s
Memory BW	21 PB/s
System I/O	1.2 Tb/s

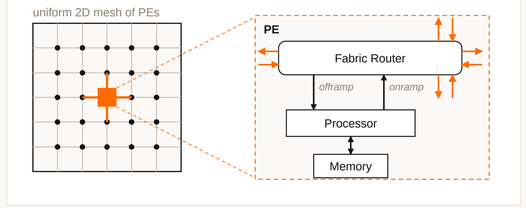


Figure 1. The wafer is one uniform 2D mesh of PEs; each PE couples a processor and private memory to a fabric router, whose offramp/onramp move wavelets to and from its four neighbours in a single cycle.

3 Algebraic BFS

The algebraic BFS is a sequence of **sparse matrix–vector multiplications (SpMV)** over the Boolean semiring. Each SpMV is one BFS step: $A \cdot \mathbf{f}_i$ gives the vertices reachable from the frontier, already visited vertices are masked out, and the process repeats until no new vertex is reached.

$$\mathbf{f}_{i+1} = \left[(A \cdot \mathbf{f}_i) \ \wedge \ (\neg\, \mathbf{v}_{i-1})\right]^{\mathsf{T}}$$

A: adjacency matrix
$\mathbf{f}_i$: frontier at step i
$\mathbf{v}_{i-1}$: visited set

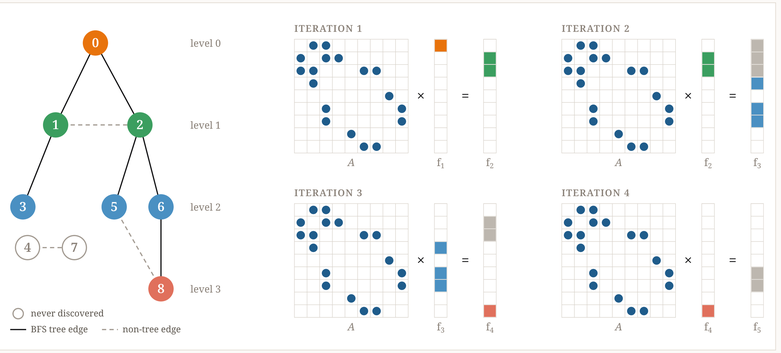


Figure 2. Graph components with colored BFS tree (left) and the four sparse matrix–vector products that traverse it (right): A · f_i masked by the visited set gives f_{i+1}, with the frontier starting at vertex 0. Grey cells are already visited vertices up to step i-1, filtered out before the next round;

4 Balancing the matrix

R-MAT degrees are **heavily skewed**, so a contiguous split of the adjacency matrix across the PE grid concentrates edges on a few ones.

We **balance the matrix** by searching for a legal permutation of row–column pairs for a given PE grid, distributing non-zero elements uniformly across tiles.

This also reduces overall compute time since the **work is evenly distributed across PEs**.

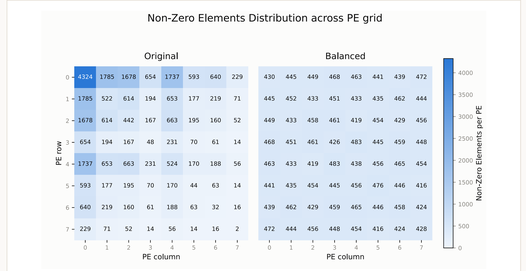


Figure 3. Non-zeros per PE before and after balancing: peak per-PE load falls from 4.1× the mean to 1.08×.

5 Mapping & data layout

The adjacency matrix uses a **2D-mapping** over the PE grid (one tile per PE), and is stored locally in **DCSC** (Doubly Compressed Sparse Columns), which keeps only the non-empty columns.

The frontier is a **dense bitmap** split along one dimension only, so each grid column owns a fragment replicated on every PE of that column; reduction and masking become **SIMD** word operations.

Parents are tracked in a **dense per-tile vector** p_{ij} aligned with the tile's rows, so each PE writes only the vertices it owns.

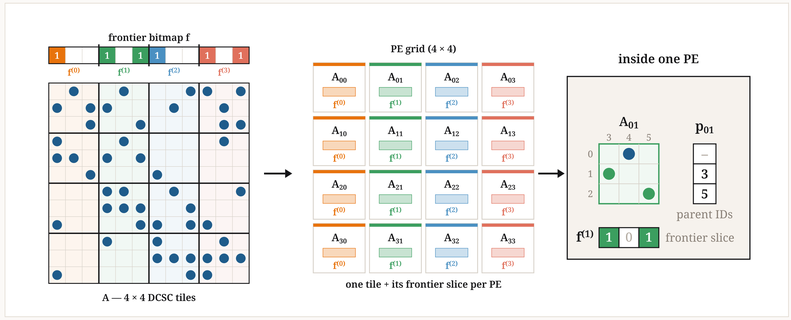


Figure 4. PE(0,1) is shown expanded: tile A_{01}, the frontier slice $f^{(1)}$ aligned horizontally with its columns, and the parent vector p_{01} aligned with its rows. Here $f^{(1)}$ is active on the tile's first and last column, so p_{01} records the global IDs 3 and 5 of the discovering vertices (green); "–" marks a vertex not reached. Grey numbers are global vertex IDs.

6 One BFS round

1. **Frontier broadcast**: the frontier fragments sit on the diagonal PE of each column and are broadcast over the column.
2. **Local SpMV**: each PE multiplies its tile by the frontier fragment, recording the parent of every newly discovered vertex.
3. **Result row reduce**: each PE row aggregates its local results onto the diagonal PE, revealing which vertices were discovered regardless of parent.
4. **Local termination test**: the diagonal PE masks out already-visited vertices and checks whether its portion holds anything new, storing the check result in a flag.
5. **All-reduce termination**: local flags are reduced onto the grid-center PE, which decides termination and either broadcasts the outcome back or gathers the parent vector and sends it to the host.

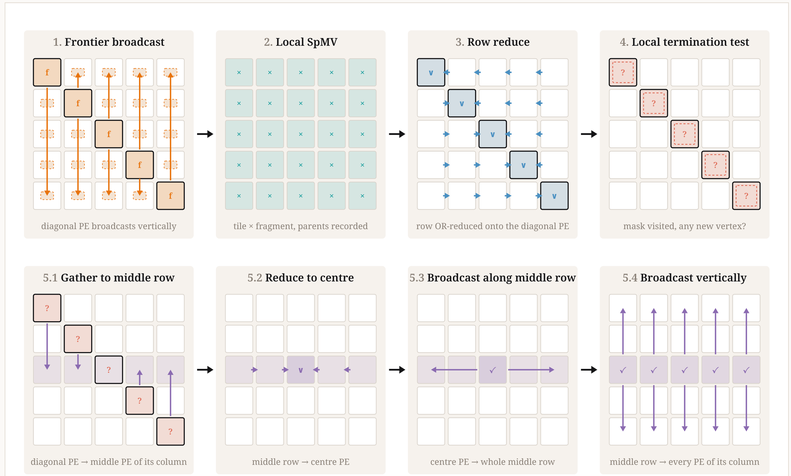


Figure 5. One BFS round step-by-step on a 5x5 PE grid. Panels 1-3 form the distributed SpMV: broadcast, local multiply, row reduce. Panel 4 is the local termination test, and 5.1-5.4 make it global, routing the all-reduce through the centre: gather to the middle row (5.1), reduce to the centre PE (5.2), broadcast back along the row (5.3) and every column (5.4).

7 Evaluation setup

We evaluate on a **CS-3 at Argonne National Laboratory**, on R-MAT graphs of edge factor 16 with source vertex 0. Per-round compute and communication times, host-transfer spans, and parent-resolve time are all measured with **on-device timestamp-counter brackets**.

WHY MEMORY RUNS OUT AT SCALE 20

Each PE already stores its code, the frontier at **1 bit per vertex**, the adjacency matrix at **2 bytes per edge**, and the parents at **4 bytes per vertex**. At s20 that'already using most of the space, and by s21 we run out of room.

8 Performance evaluation

Peak throughput reaches **45 GTEPS** at s20 on 750×750 and had not plateaued when s20 filled on-chip memory. Larger grids cut the nonzeros each PE holds but lengthen every collective, so the communication share rises with grid size at every scale; in the best configurations, communication takes **73–89%** of device time.

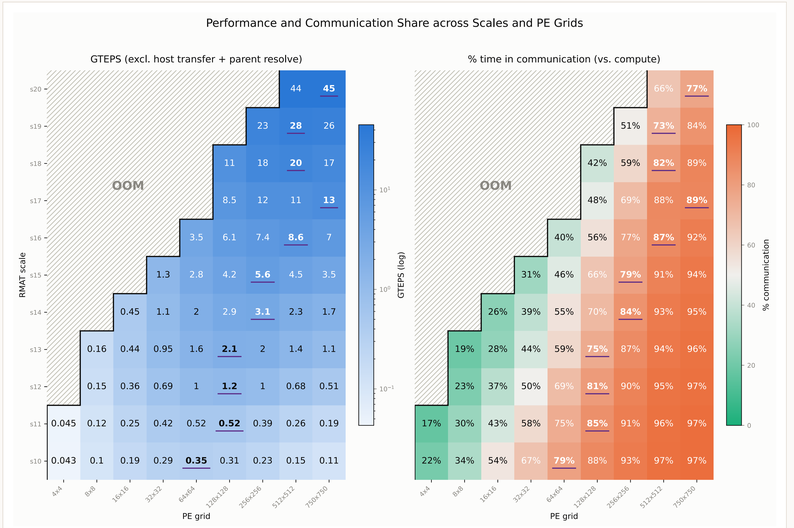


Figure 6. Throughput (GTEPS, billions of traversed edges per second, left) and share of device time in communication (right), kernel time only. Hatched: exceeds on-chip capacity; underlined: best grid per scale.

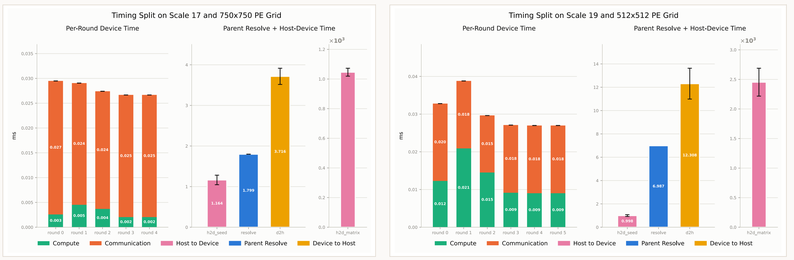


Figure 7. Per-round device time at the two extremes of the best-configuration band: communication-dominated (left) and compute-heavy (right). Both configurations were run 5 times.

9 Conclusions & future work

Cut communication S17, 750×750

Every best configuration is **dominated by collective communication**. So distributing the work across more PEs becomes counterproductive.

The SDK collectives module uses a chain topology; **tree-based collectives** such as **Bine Trees** [2], developed at our lab and presented in this session as *"Exploring Tree-Based Approaches for Collective Communications on Cerebras WSE-3"*, should reduce this cost.

Cut compute S19, 512×512

The least communication bound best configuration still spends a considerable share on compute, which matters more once collectives get cheaper.

Each round runs a **top-down BFS** locally, scanning every edge out of the frontier even when they lead to already-visited vertices. A **direction-optimizing BFS** [3] would switch to a bottom-up scan on large frontiers.

Host–Device Communication and Parent Resolve

Keeping the search on device removes the link from the inner loop, but not from a run.

Graph loading is paid once and can be amortized over searches from different sources; the extraction is not. It splits into resolving the parent vector, and exporting it to the host, but SDK overhead dominates rather than the kernel.

On-chip capacity

Higher performance needs larger graphs, and real-world graphs are larger still, so everything must all fit in the chip's limited SRAM.

Most of the footprint comes from the **parent vector**, followed by the dense frontier. Our next steps aim at **sparse representations** for both, leading to a **sparse matrix–sparse vector product (SpMSpV)**.

ACKNOWLEDGEMENTS & REFERENCES

This research used resources of the Argonne Leadership Computing Facility, a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

[1] S. Lie. Cerebras Wafer-Scale AI. Hot Chips 36 Symposium on High-Performance Chips, 2024.
[2] D. De Sensi et al. Bine Trees: Enhancing Collective Operations by Optimizing Communication Locality. SC '25, ACM, 2025, pp. 1901–1916.
[3] S. Beamer, K. Asanović, D. Patterson. Direction-Optimizing Breadth-First Search. SC '12, IEEE, 2012, pp. 1–10.