Algebraic Breadth-First Search on the Cerebras Wafer-Scale Engine

Elia Belli
Sapienza University of Rome

Daniele De Sensi (advisor)
Sapienza University of Rome

Abstract

Graph traversal is memory bound on conventional hardware: irregular access defeats the cache hierarchy. The Cerebras WSE-3 gives each of its 900,000 cores its own 48 KB of SRAM, turning irregular access into local access, but leaves them coordinating over a 2D mesh and a slow host-device link. We present an algebraic BFS that runs entirely on device, avoiding the link that host-callable SpMV kernels would cross every iteration. Sweeping Graph500-specification R-MAT graphs from scale 10 to 20 across PE grids up to 750x750, we find adding cores helps only up to a point, past which larger collectives outweigh the smaller share of work per PE. At that turning point up to 89% of device time goes into collectives, while exporting results costs two orders of magnitude more than the search itself. Within the kernel, distributed on-chip memory removes the memory wall and leaves collective communication as the next target.

1 Introduction

Graph traversal is memory bound on conventional hardware: irregular access defeats the cache hierarchy. The Cerebras WSE-3 [8] removes that wall, replacing DRAM with on-chip SRAM, but leaves its cores coordinating over a 2D mesh and a slow host-device link. Prior WSE work targets regular dataflow kernels [9] [7] [10] [4], and existing SpMV kernels require data movement between host and device at each iteration. We design an algebraic BFS that runs entirely on device.

2 Methodology

We choose BFS because it is a foundational benchmarking kernel, most notably in Graph500 [6], and adopt its algebraic formulation, which maps onto distributed memory naturally.

Algebraic BFS. The algebraic BFS is a sequence of Sparse Matrix-Vector multiplications (SpMV) over the Boolean semiring. Each SpMV is one BFS step: its result gives the vertices reachable from the frontier, visited vertices are masked out, and the process repeats until no new vertex is reached (Fig 1):

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

with A the adjacency matrix, $\mathbf{f}_i$ the frontier and $\mathbf{v}_i$ the visited vector at iteration i.

Data Layout and Mapping. The two operands are reused differently, so they are mapped differently. We store the adjacency matrix in DCSC format [2], tiled over the 2D PE grid, and the frontier as a dense bitmap replicated along one dimension, so reduction and masking become SIMD word operations (Fig 2).

Adjacency Matrix Balancing. That split assumes each PE holds a similar share, which R-MAT does not give: degrees are heavily skewed, so a contiguous split concentrates hubs on a few PEs, and the heaviest tile can overflow its memory. We relabel vertices so that each PE holds a comparable number of edges (Fig 3).

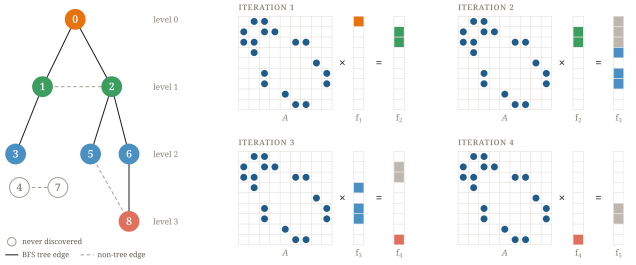


Figure 1: Graph with colored BFS tree (left) and the four sparse matrix–vector products that traverse it (right): $A \cdot f_i$ masked by the visited set gives f_{i+1}, with the frontier starting at vertex 0. Grey cells are already visited vertices up to step $i-1$, filtered out before the next round;

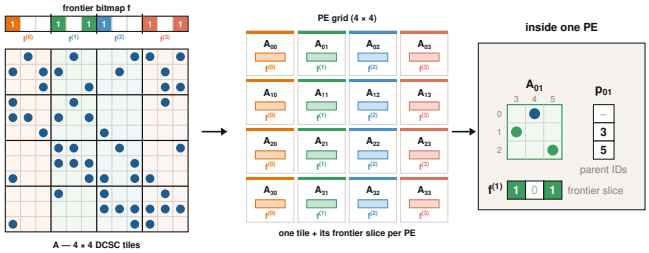


Figure 2: Mapping on a 4x4 PE grid. PE(0,1) is shown expanded: tile A_{01}, the frontier slice $f^{(1)}$ aligned horizontally with its columns, and the parent vector p_{01} aligned with its rows. Here $f^{(1)}$ is active on the tile's first and last column, so p_{01} records the global IDs 3 and 5 of the discovering vertices (green); "–" marks a vertex not reached. Grey numbers are global vertex IDs.

2.1 BFS Design and Phases

The BFS runs entirely on device. Each round is one distributed SpMV followed by a termination test. The SpMV leaves its result spread along rows, but the next round needs the frontier spread along columns, so every iteration has to transpose it. The diagonal does this for free: the row reduce leaves fragment i on PE (i, i), which also sits on column i, so broadcasting down that column feeds the next round already transposed. A round proceeds as follows (Fig 4):

(1) **Frontier Broadcast**: each fragment sits on the diagonal PE of its column and is broadcast down that column;

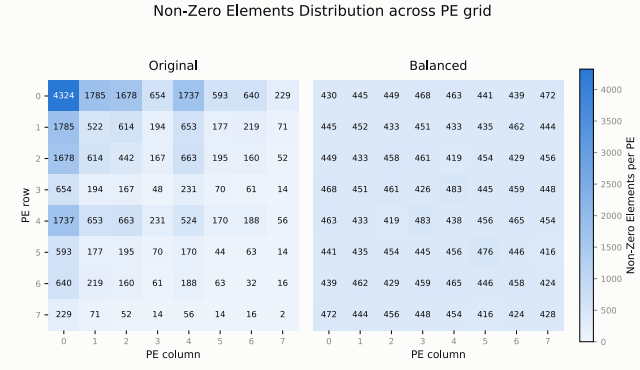


Figure 3: Nonzeros per PE before and after balancing: peak per-PE load falls from 4.1× the mean to 1.08×.

(2) **Local SpMV**: each PE multiplies its local block, recording a parent for every newly discovered vertex;
(3) **Result row Reduce**: each PE row reduces its results on the diagonal, so a vertex counts as discovered whichever PE found it;
(4) **Local Termination**: 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.

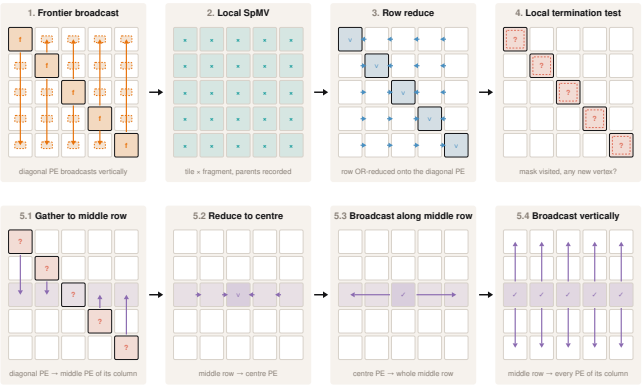


Figure 4: One BFS round step-by-step on a 5x5 PE grid. Panels 1-3 form the distributed SpMV: broadcast, local multiply, row reduce. Panel 4 is the local termination test, and 5.1-5.4 make it global, routing the all-reduce through the centre: gather to the middle row (5.1), reduce to the centre PE (5.2),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

3 Experimental Evaluation

We evaluate on a CS-3 at Argonne National Laboratory, on R-MAT graphs following the Graph500 BFS specification [5], sweeping scale S (a graph of 2^S vertices) from 10 to 20 against grids up to 750x750. Throughput is GTEPS over kernel time only (Fig 5); the host path is orders of magnitude slower and reported separately (Fig 6). Peak throughput grows with scale, from 0.35 GTEPS at scale 10 to 45 at scale 20, and had not saturated when the largest graph exhausted on-chip capacity. Adding PEs cuts the nonzeros each one holds, but spreads them over a wider grid, so every collective travels further: the communication share rises monotonically with grid size at every scale (Fig 5, right).

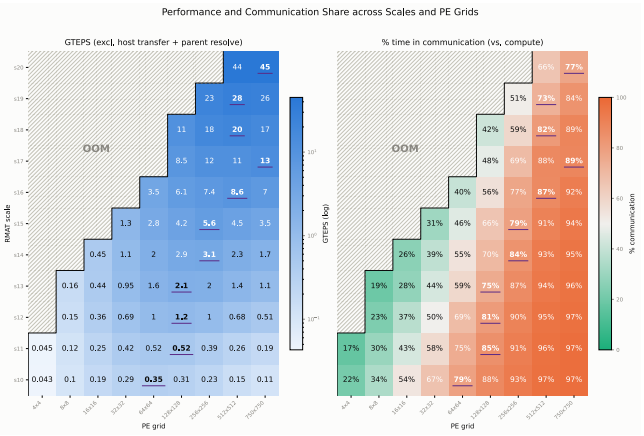


Figure 5: Throughput (GTEPS, billions of traversed edges per second, left) and share of device time spent in communication (right). Both are measured over kernel time only, excluding host transfer and parent resolve. Hatched cells exceed on-chip capacity; the best grid per scale is underlined in both panels.

4 Conclusion and Future Work

We analyze the most (89%) and least (73%) communication bound configurations (Fig 6). Collective communication dominates both, so replacing the platform's collectives is the highest-value direction; concurrent work at our institution implements Bine Trees collectives [3] for the WSE, which we intend to integrate. Compute, though smaller, remains substantial, so a direction-optimizing BFS [1] is worth exploring.

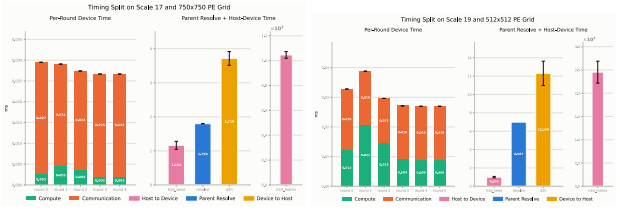


Figure 6: Communication and compute time (ms) split per round with host-device communication and parent resolve, for the most (left) and least (right) communication-bound configurations.

The kernel is the small part. Host-device time no longer appears in the inner loop, but still bounds a run end to end: on scale 19,

six rounds total 0.18 ms of device time against 19.3 ms to extract the result and 2.4 s to load the graph. The load is paid once, so searches from other sources amortize it; the extraction is not, and is dominated by exporting parents. Aggregating them on the host did not scale, so we reduce on device, but a parent is a 32-bit identifier where a frontier entry is a single bit, so the same hops carry 32 times the volume. R-MAT graphs are low diameter, the worst case here: more rounds would spread the same extraction cost over more device work.

Higher performance needs larger graphs, and real-world graphs are larger still. 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).

References

[1] Beamer, S., Asanovic, K., and Patterson, D. Direction-Optimizing Breadth-First Search. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (2012), pp. 1–10.

[2] Buluç, A., and Gilbert, J. R. Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments. *SIAM Journal on Scientific Computing 34*, 4 (Jan. 2012), C170–C191.

[3] De Sensi, D., Pasqualoni, S., Piarulli, L., Bonato, T., Ba, S., Turisini, M., Domke, J., and Hoefler, T. Bine trees: Enhancing collective operations by optimizing communication locality. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Nov. 2025), SC '25, ACM, p. 1901–1916.

[4] Essendelft, D. V., Almolyki, H., Shi, W., Jordan, T., Wang, M.-Y., and Saidi, W. Record acceleration of the two-dimensional ising model using high-performance wafer scale engine. *Comput. Phys. Commun. 315* (2024), 109734.

[5] Graph 500 Steering Committee. Graph 500 benchmark specification. https://graph500.org/?page_id=12, 2017. Accessed: 2026-07-30.

[6] Graph 500 Steering Committee. Graph 500: Large-scale benchmarks. https://graph500.org/, 2026. Accessed: 2026-08-03.

[7] Jacquelin, M., Araya–Polo, M., and Meng, J. Scalable Distributed High-Order Stencil Computations. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis* (Dallas, TX, USA, Nov. 2022), IEEE, pp. 1–13.

[8] Lie, S. Cerebras wafer-scale AI. Hot Chips 36 Symposium on High-Performance Chips, 2024. https://hc2024.hotchips.org/assets/program/conference/day2/72_HC2024.Cerebras.Sean.v03.final.pdf. Accessed: 2026-08-07.

[9] Rocki, K., Essendelft, D. V., Sharapov, I., Schreiber, R., Morrison, M., Kibardin, V., Portnoy, A., Dietiker, J., Syamlal, M., and James, M. Fast stencil-code computation on a wafer-scale processor. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis* (2020), pp. 1–14.

[10] Santos, K., Moore, S. G., Oppelstrup, T., Sharifian, A., Sharapov, I., Thompson, A. P., Kalchev, D., Perez, D., Schreiber, R., Pakin, S., León, E., Laros, J., James, M., and Rajamanickam, S. Breaking the molecular dynamics timescale barrier using a wafer-scale system. *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis* (2024), 1–13.