Communication Strategies for Wafer-Scale Iterative Solvers

Hayden Estes*, Nathan Hanford†, Kirk W. Cameron*, Edgar A. León†
* *Virginia Tech*
Alexandria, Virginia, USA
† *Lawrence Livermore National Laboratory*
Livermore, California, USA

***Abstract*—Collective communication strongly affects the performance of iterative scientific applications. The conjugate gradient repeatedly computes global dot products that require reductions across processing elements. This work evaluates chain and hierarchical XY two-phase reductions on a Cerebras Wafer-Scale Engine. We developed a reproducible workflow for compiling wavelet programs, executing them on hardware, collecting distributed timer values, and producing graph-ready data. Measurements across 262,144 processing elements show a payload-dependent crossover. XY two-phase communication performs best for small payloads, but chain communication becomes faster as the payload grows. These results motivate adaptive collective selection based on message size.**

***Index Terms*—wafer-scale computing, collective communication, all-reduce, conjugate gradient, performance analysis**

I. Motivation

The conjugate gradient (CG) repeatedly performs global dot products that require communication across all participating processing elements (PEs). On a wafer-scale system, the collective communication algorithm can determine whether an application benefits from the available parallelism.

This work compares two AllReduce strategies and measures how their performance changes with payload size. The experiments also evaluate whether fusing local computation with the beginning of a reduction improves end-to-end CG dot product performance.

Our programs are written in the Tungsten language developed by Cerebras to leverage the wafer architecture for scientific computing. This architecture was originally designed for AI workloads.

II. Wafer-Scale Architecture

The experimental region contains a 512×512 array with 262,144 PEs. Each PE executes a local program, called a wavelet, and stores its data in local scratchpad memory. Wavelets communicate through sockets connected to hardware routes during compilation.

The implementation assigns separate wavelets to the root PE, the remaining PEs in the top row, and the PEs in lower rows. Local communication routes perform reductions, and a broadcast route distributes the completed result from the root across the PE grid.

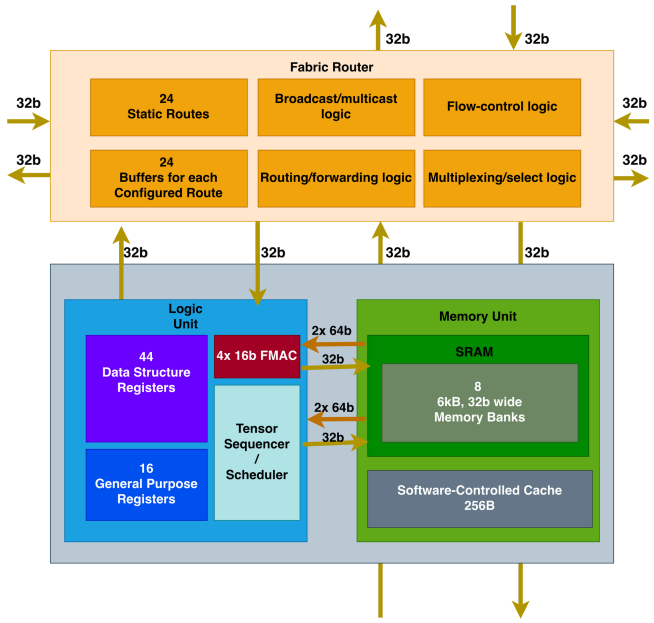


Fig. 1. Processing-element architecture and communication paths.

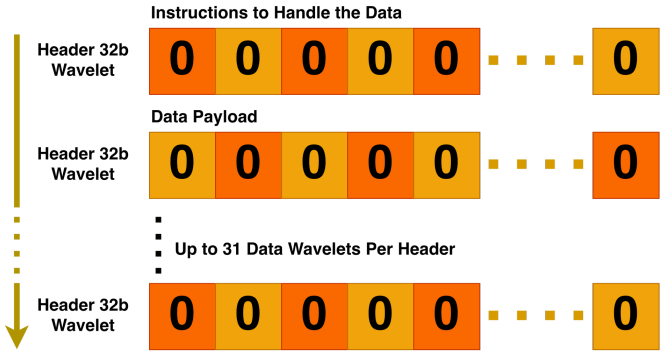


Fig. 2. Wavelets map local computation and communication onto each PE.

III. AllReduce Implementations

The chain algorithm reduces every row from right to left, reduces column zero from bottom to top, and broadcasts the result. Its simple pipeline avoids additional reduction phases.

The XY two-phase algorithm divides each dimension into groups of approximately 23 PEs. The first phase reduces values within each group. Group leaders then perform a second reduction phase. This structure shortens individual communication paths but introduces additional phases and synchronization.

The CG kernels compute B output lanes using K local matrix and vector elements. Each PE produces local contributions, the collective reduces the B lanes, and the root folds them into a scalar dot product. The partially fused implementation injects each completed local lane directly into the first horizontal reduction phase.

IV. Experimental Workflow

The automated workflow generates inputs, compiles wavelets, constructs a run image, executes the program, and collects timing information.

TABLE I
Experimental Workflow

Stage	Primary tool or output
Route and compile	`paint`
Link and prepare	`milink`, `weight`
Install input data	`miset`
Execute on wafer	`csclient job`
Collect symbols	`csclient fetch`
Analyze timings	Python and CSV output

Timer snapshots from the root, top row, and lower rows reconstruct a global timeline. The analysis converts cycles to microseconds using an 850 MHz clock and writes graph-ready CSV files.

V. Results

The complete $K = 1$ sweep shows a clear payload-dependent crossover. At a 4-byte payload, XY two-phase completes the standard CG dot product in 11.30 microseconds, compared with 52.99 microseconds for chain. XY two-phase is approximately 4.7 times faster.

At 256 bytes, XY two-phase remains approximately twice as fast. The methods become nearly equal at 1 KiB. At 4 KiB, chain completes in 195.68 microseconds, compared with 323.48 microseconds for XY two-phase. Chain is approximately 1.65 times faster at this payload.

The completed $K = 2$ measurements reproduce the same behavior through the available 1 KiB payload. Increasing K from 1 to 2 changes most overlapping totals by approximately 1 percent or less. Communication topology and payload size therefore dominate performance in the measured range.

TABLE II
CG Dot Product Results for $K = 1$

Payload bytes	Chain μs	XY μs	Faster
4	52.99	11.30	XY
16	53.41	12.25	XY
64	55.69	15.96	XY
256	62.35	30.62	XY
1024	88.08	89.18	Chain
4096	195.68	323.48	Chain

VI. Conclusions

No single AllReduce algorithm performs best for every payload. Hierarchical XY two-phase communication performs best for small messages because it shortens the critical communication path. Chain communication performs better for larger messages because its simple pipeline avoids repeated phase overhead.

These results support an adaptive strategy that selects a collective strategy based on the size of the payload. Future work will complete the remaining parameter sweep and evaluate dynamic selection inside a full CG solver.

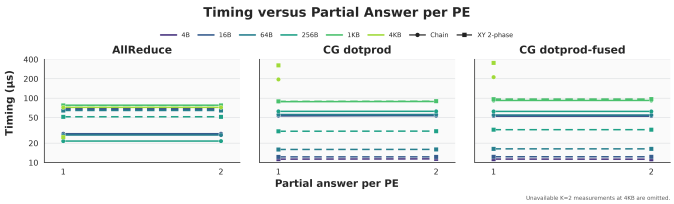


Fig. 3. Perfect scaling from 1 to 2 partial answers. Ongoing work to measure larger K values.

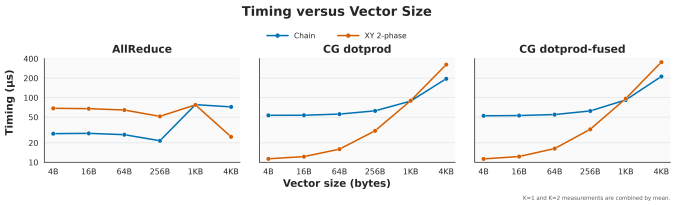


Fig. 4. The best communication algorithm varies based on the vector size.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-ABS-2024370.