Jesse Lu, David Qu, Jim Qu, Ryan Fong, Geun Ho Ahn, and Jelena Vučković

Overcoming Memory Bandwidth Limitations In GPU-Accelerated Finite-Difference Time-Domain Simulations

A systolic update scheme

The exponential growth of artificial intelligence (AI) has fueled the development of high-bandwidth photonic interconnect fabrics as a critical component of modern AI supercomputers [1]. As the demand for ever-increasing AI compute and connectivity continues to grow, the need for high-throughput photonic simulation engines to accelerate and even revolutionize photonic design and verification workflows will become an increasingly indispensable capability for the integrated photonics industry. Unfortunately, the mainstay and workhorse of photonic simulation algorithms, the finite-difference time-domain (FDTD) method [10], is a memory-intensive and computationally lightweight algorithm that is fundamentally misaligned with modern computational platforms, which are equipped to deal with compute-heavy, memory-light workloads instead. This article proposes a systolic update scheme for the FDTD method, which circumvents this mismatch by reducing the need for global synchronization as well as minimizing the amount of global memory access needed per cell update by aggressively reusing existing field values already present within the GPU cache. We present an initial prototype of our scheme on a full 3D FDTD algorithm that achieves a performance of ~0.15 trillion cell updates per second (TCUPS) on a single Nvidia H100 GPU. Our work paves the way for the increasingly efficient, cost-effective, and

Digital Object Identifier 10.1109/MAP.2026.3668263

high-throughput photonic simulation engines needed to continue powering the AI era.

MOTIVATION

The insatiable demand for computational power for AI applications has resulted in a new breed of AI supercomputers that critically rely on advanced, high-bandwidth photonic interconnect fabrics for their operation [1]. As integrated photonics embeds itself further and further into the core of the modern computer hardware stack [3], the availability of fast, scalable photonic simulation engines to accelerate and even revolutionize photonic simulation, design, and verification workflows becomes increasingly necessary to continue the pace of innovation [9].

A high-throughput, cost-effective photonic simulation engine could drive such a "revolution within a revolution"—revolutionizing photonics to, in turn, continue to power the AI revolution—by replacing the expensive, months-long development cycle of multiproject wafer runs and laboratory-based device characterization with a quick-feedback workflow that instead leverages high-fidelity simulation running on massively parallel compute clusters. In addition, such a fast, and widely available engine would enable new paradigms like inverse design [8], which require high-throughput simulation capabilities to automatically optimize existing structures in a free-form manner, or else to search for better device topologies across vast design spaces.

Unfortunately, the mainstay and workhorse of photonic simulation algorithms, the FDTD method [10], because it is a memory-intensive but computationally lightweight algorithm, is fundamentally misaligned with modern computational platforms, which are, instead, equipped with massive computational throughput and relatively meager memory bandwidth.

Here, we present an implementation of the original FDTD method that circumvents these limitations by use of a "systolic" update scheme, which relaxes both synchronization and memory bandwidth requirements, while still performing the exact same update equations. Our initial prototype achieves a performance of ~0.15 TCUPS on a single Nvidia H100 GPU. Critically, we motivate the need for our systolic scheme by showing that it is able to recover the simulation speed lost due to simulation domains, exceeding the capacity of the top-level caches in the GPU memory hierarchy.

Our work builds on top of key innovations in bandwidth-efficient tiling schemes for FDTD methods [2], [6]. It paves the way for increasingly efficient implementations of the FDTD method on modern computational platforms, and even holds the possibility of designing custom compute hardware tailored exactly for such a systolic implementation of the FDTD method.

INHERENT MISMATCH BETWEEN FDTD AND GPUs

The FDTD method presents a straightforward method for updating Maxwell's equations in the time domain and consists of update equations in the vein of

$$E_x^{i,j,k} = E_x^{i,j,k} + \frac{\Delta t}{\Delta y}\left(H_z^{i,j,k} - H_z^{i,j-1,k}\right) - \frac{\Delta t}{\Delta z}\left(H_y^{i,j,k} - H_y^{i,j,k-1}\right) \quad (1)$$

which demonstrates that the update of an E_x field value requires:

- six memory operations:
  - one load of E_x
  - two loads of H_z
  - two loads of H_y
  - one store of E_x
- six floating-point operations:
  - four additions/subtractions
  - two multiplications (assuming $\Delta t/\Delta y$ and $\Delta t/\Delta z$ have been precomputed).

If we assume 4-byte floating-point numbers, this simplified analysis reveals a compute-to-memory ratio of 0.25 floating-point operations per byte of memory transferred.

In contrast, modern compute platforms, such as the Nvidia H100 NVL [5], which boasts 60×10^9 floating-point operations per second but a bandwidth of only $\sim 4 \times 10^9$ bytes per second, have a compute-to-memory ratio of ~15 (note that if we assume tensor-core operations, this ratio increases yet further by over an order of magnitude). Herein lies the fundamental mismatch between the FDTD method and modern compute platforms, that the FDTD method is a computational workload that is nearly two orders-of-magnitude more memory-intensive than those for which modern compute platforms are optimized. Or, put another way, that a naive implementation of the FDTD method is expected to unlock only ~1% of the computational capability of modern compute platforms.

That said, we extend our simple analysis by examining the performance of the FDTD method as the size of the simulation domain is increased. To do this, we implement the FDTD method in a very straightforward manner and observe that the performance:

- It bottoms out at a per-thread performance of ~0.2 cell updates per microsecond, or ~0.026 TCUPS (128K threads in use) as the simulation domain increases, which is consistent with prior art [4].
- But, it also has the critical characteristic of actually increasing dramatically by a factor of ~10 for much smaller simulations.

This order-of-magnitude performance gap can be attributed to three primary factors, as denoted in Figure 1:

1) The need for global synchronization when the number of threads used to update the simulation domain exceeds 1,024, so that multiple thread blocks must now be used.
2) The L1 cache limit of 228 KB on the H100 GPU.
3) The L2 cache limit of 50 MB on the H100 GPU.

Note that we choose to use the terms *shared memory* and *L1 cache* interchangeably; however, in practice their flexibility and use differs in that they necessitate explicit and implicit control of what values are being cached, respectively.

We conclude, then, that although there is a large, fundamental compute-to-memory ratio mismatch between the FDTD method and modern compute platforms, this mismatch is partially alleviated for very small simulation domains that both fit within a high-throughput, low-latency cache that is located near the arithmetic units of the processor (such as the L1 cache on

the H100 GPU), and can bypass the need for the synchronization of threads at a global, chip-level scale.

SYSTOLIC FDTD UPDATE SCHEME

We now leverage the insight from our previous experiment and adopt the following strategy for circumventing the memory bandwidth limitations of a GPU-accelerated FDTD engine. Specifically, we propose that, instead of seeking a fast FDTD update over a single, monolithic simulation domain, we should instead seek to implement the FDTD update over a series of smaller simulation subdomains, which are connected only via one-way data dependencies where the need for global communication can be amortized by extending the buffer sizes associated with this communication. We would then seek to circumvent the order-of-magnitude performance gap seen in Figure 1 by designing subdomains that both fit within the L1 cache of the GPU, and minimize the need of global synchronization between neighboring subdomains.

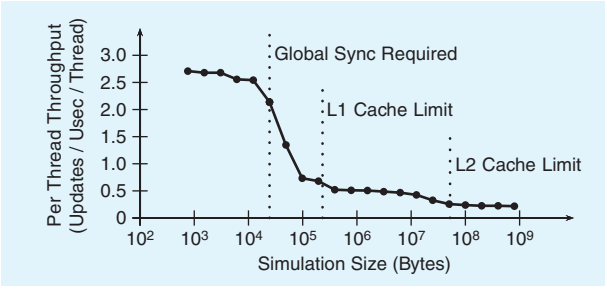

FIGURE 1. Empirical performance of a naive FDTD implementation for increasingly large simulation domains. The performance of a naive implementation of the FDTD update algorithm on a H100 Nvidia GPU is shown for increasingly large simulation domains on a per thread basis. At a high-level, our results show that there is an order-of-magnitude loss in performance that is predominantly attributable to the onset of the need to perform global synchronization operations, and to access memory resources, which exceed first the capacity of the L1 cache, and then the capacity of the L2 cache, until finally the global memory pool is accessed. The bottomed-out performance at large simulation domains is congruent with previous work reported in the literature [4]. At the same time, our results in this figure hint that better performance may be possible to be achieved by devising an update strategy that can capture the performance attainable when the simulation domain is so miniscule that it can fit entirely within the L1 cache of the GPU, and somehow extend that performance to arbitrary simulation domain sizes.

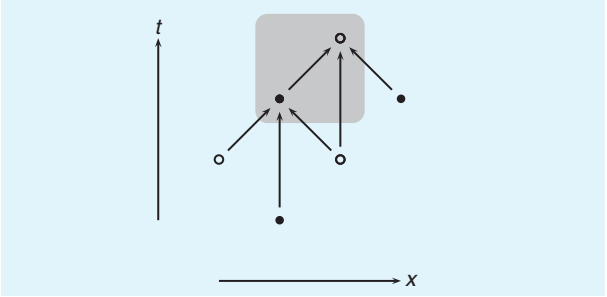

FIGURE 2. Dependency structure for a single Yee cell in the 1D FDTD algorithm. The dependency structure of a 1D FDTD update algorithm is shown in the x–t plane, for a single Yee cell where E-field nodes are represented by solid circles and H-field nodes are represented by hollow circles. The arrows indicate dependencies between field values; namely, that to compute the value of any given node, we must first obtain the values of all of the nodes that point to it. As can be seen in the figure, each node is dependent on the value of the node at the previous timestep, as well as the adjacent nodes of the complementary field immediately adjacent to it.

We term this strategy the *systolic FDTD update scheme*, since it is reminiscent of the systolic array architecture in that each subdomain can be considered to be a single "node," which communicates, as we shall see, only boundary values to neighboring nodes, and communicates them in a directional (as opposed to bidirectional) manner, which minimizes the need for global synchronization.

While our strategy is easily stated, the interdependencies of the FDTD update equations do not, at first glance, naturally yield to such a strategy. Because the full FDTD update in three spatial dimensions is actually an acyclic dependency graph in four dimensions (three spatial and one temporal dimension), we instead illustrate the principles of our strategy for the case where there is a single spatial dimension (and therefore two dimensions in total).

For a single spatial dimension, the dependency structure of the FDTD update can be simplified to a scalar electric field interacting with a scalar magnetic field, where both are shifted by half a unit cell in both spatial and temporal dimensions. In this case, a single Yee cell can be considered as an E-field node paired with a neighboring H-field node, as shown in Figure 2.

In our quest to devise our systolic FDTD update scheme, we first attempt the most natural subdivision strategy, which is to simply divide the simulation domain both spatially and temporally: spatially, into groups of adjacent Yee cells, and temporally by updating each spatial block independently of the others, deferring interblock synchronization and communication to later timesteps. This strategy is illustrated in Figure 3, which shows an implementation that utilizes spatiotemporal blocks of shape 2×2.

Unfortunately, Figure 3 also reveals that this naive strategy is not only inefficient, but also results in a circular dependency, or deadlock condition, between blocks neighboring blocks. The deadlock condition means that a practical implementation of this strategy actually requires the use of "halos," as depicted in Figure 4. Halo regions break the deadlocking condition, but come at the cost of increased memory usage, as well as the need to perform extra redundant computation.

An alternative strategy is to break the deadlock condition without the cost of redundant computation or memory usage, and comes in the form of the "diamond" block, which is a 45° rotation of the computational cell, as shown in Figure 5. Such a strategy, by virtue of more naturally aligning with the dependency structure of the FDTD update, allows for computational subdomains that are deadlock-free, because each diamond effectively acts as a "metanode" in that the interdiamond

dependency structure is exactly the same as the original FDTD dependency structure; critically, each are only composed of one-directional dependencies.

We can further improve the performance of the diamond block by scanning the diamond along the x–t diagonal, as shown in Figure 6. This strategy is a natural extension of the diamond block, which naturally demands to be flattened along the temporal dimension (since there is no advantage in keeping track of field values for past timesteps), and gives us a first glimpse of how a systolic FDTD update scheme can be implemented. This is because this is the first strategy that we have seen that exhibits the key property of being able to communicate boundary

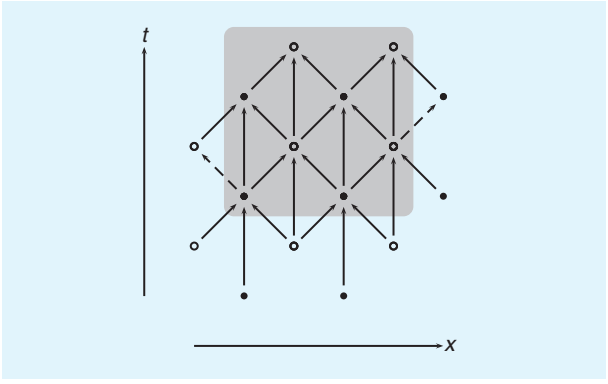

FIGURE 3. The 2 × 2 block of Yee cells as a computational subdomain. An $M \times N$ block of Yee cells is the most natural subdivision of the FDTD computational domain but immediately reveals the problem of circular dependencies that arise at the block boundaries. This figure highlights the deadlocking condition found in a 2 × 2 block of Yee cells, in that the dashed arrows indicate nodes, which are dependent on values within the computational subdomain, which the subdomain itself is dependent on. The deadlock condition generated by the circular dependencies means that such a tiling of the computational domain cannot be implemented in practice.

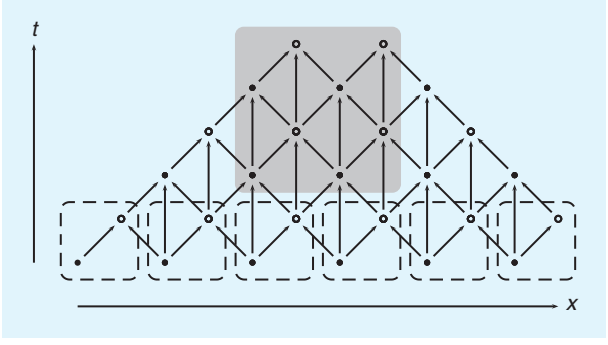

FIGURE 4. Utilization of halo regions with a 2 × 2 block subdomain. A 2 × 2 block updated exclusively from the field values of previous timesteps sidesteps the circular dependencies problem encountered in Figure 3 at the cost of incurring the penalty of increased memory loads (as denoted by the nodes in the dashed boxes) and redundant computation in the overlapping regions of adjacent subdomains (as denoted by the nodes outside of the both the dashed boxes and outside of the 2 × 2 subdomain block).

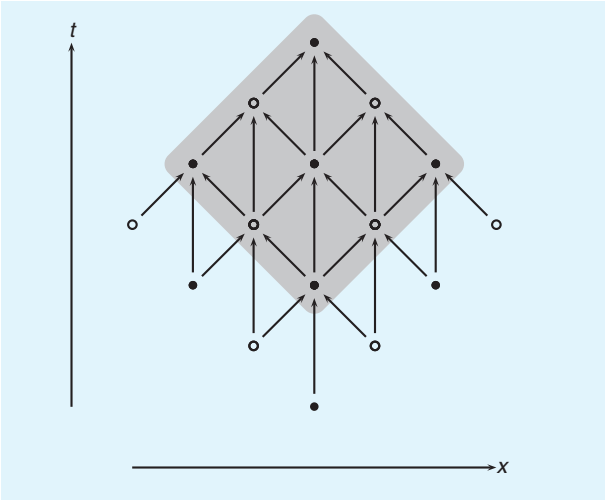

FIGURE 5. The "diamond"-shaped subdomain. A diamond-shaped subdomain successfully avoids the deadlock conditions between adjacent subdomains encountered in Figure 3, while also not requiring the use of halo regions and their associated overhead, as seen in Figure 4. The virtue of the diamond-shaped subdomain is that it is a natural subdivision of the computational domain in that it mimics the underlying dependency structure of the domain itself. In fact, the resulting intersubdomain dependency structure that arises when the simulation domain is decomposed into diamonds is identical to the original dependency structure of the FDTD update.

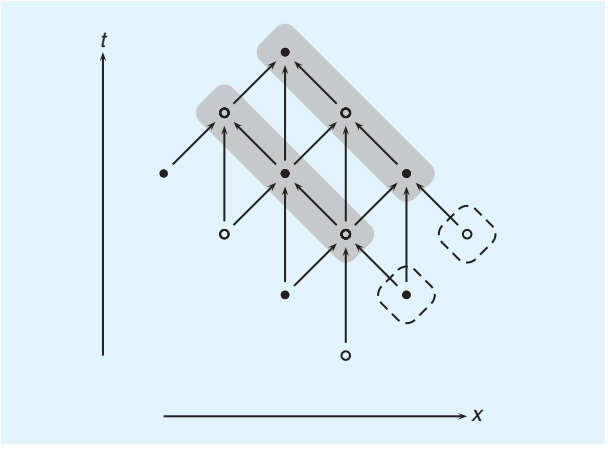

FIGURE 6. A diagonal "bar" scanned along x–t. Since the value of nodes at the same spatial location but at past timesteps are not needed in general, the flattening of the diamond-shaped subdomain depicted in Figure 5 is a natural improvement. Such a bar-shaped subdomain is scanned along the x–t diagonal because all of the nodes that are needed to resolve the nodes at the new scan point are already present as nodes in the previous scan point, with the exception of a single extra node, which must be loaded from global memory. As such, the diagonal bar subdomain represents a critical step toward a systolic update scheme. This is because it is the first subdomain which, by virtue of scanning, requires a lower-dimensional (zero-dimensional point, in this case) shape to be loaded to resolve the nodes of a higher-dimensional (in this case, 1D) shape. As such, it provides a critical stepping stone in achieving a systolic update scheme where communication between subdomains via global memory is necessary for boundary values of the subdomain only.

values at reduced dimensionality (in this case, a single zero-dimensional node) while continuing to update the interior of a computational cell at higher dimensionality (in this case, a 1D diagonal bar).

A modified and final version of the strategy keeps the dynamic of scanning along the x–t diagonal, but allows for much simpler inner update in the utilization of a horizontal 1D bar, as shown in Figure 7 (where all of the field values are located at the same timestep and can all be updated in a simultaneous manner), instead of the diagonal bar of Figure 6 (where the update of field values must proceed sequentially, starting from the node at the earliest timestep). As such, we consider the strategy depicted in Figure 7 to be the ideal form of our systolic FDTD update scheme, in that:

- It allows for the update of the (in this case 1D, but in our practical implementation 3D) subdomains to be proceed entirely in a local cache, while only requiring lower-dimensional (a point in this case, but a 2D slice in our practical implementation) set of boundary values to be communicated to neighboring subdomains via (much slower) global memory.
- It allows for delayed synchronization of neighboring subdomains since an "upwind" subdomain can leave behind a trail of boundary values for a "downwind" subdomain to pick up whenever it is ready to consume them.

We note that this dynamic is only possible because of the one-directional dependencies between adjacent subdomains, which is directly a consequence of forcing the subdomains to traverse along the x–t diagonal, thus creating a one-directional dependency structure. The downwind-only flow of boundary values in our systolic FDTD update scheme is simply a consequence of continually pushing the subdomains upwind. We reiterate that what this scheme achieves is working with small local simulation domains to stay in the "high-throughput" region of Figure 1, while at the same time allowing for arbitrarily large simulation domains to be solved via a systolic data flow. As such, the throughput does not have any inherent dependence on the size of the larger, macroscopic simulation domain.

Finally, we also note that the number of boundary values can be reduced by using a diamond- or octohedral-type of cell similar to [2] instead of the rectangular prism that we use here; we leave these optimizations as potential future improvements.

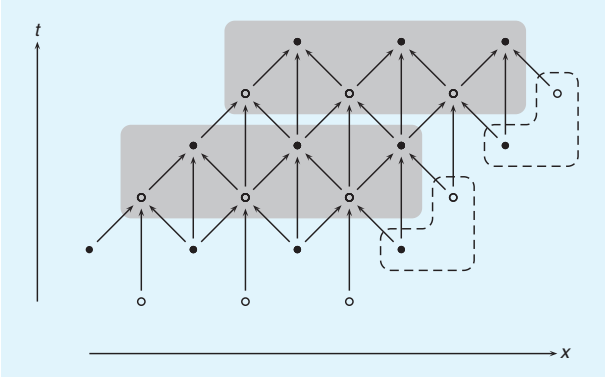

FIGURE 7. Horizontal-bar subdomain scanned along x–t. The same diagonal scanning technique introduced in Figure 6 can also be applied to a horizontal bar of N Yee cells. This results in an ideal subdomain scheme which possesses all of the advantages of the diagonal bar in Figure 6, with the added advantage of being able to trivially resolve the nodes within each scan point. This is because in the case of the diagonal bar, each node must be resolved in a sequential manner, beginning from the node at the earliest timestep (because nearly each node in the diagonal bar is dependent on a node from an earlier timestep that is in the diagonal bar subdomain and must first be resolved), while in the case of the horizontal bar, all of the nodes within the domain can be resolved in parallel (that is, all of the E-field nodes can be resolved simultaneously, after which all of the H-field nodes can then be resolved together).

PRACTICAL PROTOTYPE ON AN NVIDIA H100 GPU

Having established the principles of our systolic FDTD update scheme for the case of 1D FDTD, we extend our strategy into a practical implementation of 3D FDTD on an Nvidia H100 GPU, where subdomains are now 3D blocks, which traverse along the major axes of the (4D) simulation domain, that is, along the x–y–z–t diagonal. Communication between subdomains now occurs in the form of 2D slices, or faces, of the subdomains, which are stored in global memory, while the interior of the subdomains reside completely in L1 cache.

The details of our prototype are listed here:

- Subdomains are formed from 30×15 Yee cells, although a $31 \times 16 \times 16$ block of Yee cells is stored in cache, to allow for loading of boundary values from global memory.
- Each iteration involves the update of $30 \times 15 \times 15$ Yee cells, and the loading and writing of $\sim 31 \times 16 \times 31 \times 16 \times 16 \times 16$ cells to global memory (one slice or face for each of the three spatial dimensions).
- Compared to an implementation that loads, updates, and stores exactly one Yee cell per iteration, this implementation requires $\sim$5.41 times fewer global memory operations.
- Compared to the naive implementation consisting of a compute-to-memory ratio of $\sim$0.25 as depicted earlier, this implementation, in our estimation, achieves a compute-to-memory ratio of $\sim$4, which is over an order-of-magnitude improvement, but still far from being a compute-limited implementation.
- Our implementation doubles the number of memory operations needed per iteration, because it loads and stores not only electric- and magnetic-field values, but also structural fields corresponding to permittivity and conductivity values (our implementation does not use perfectly matched layers (PMLs), but instead relies on the imaginary part of the permittivity to allow for the formation of adiabatic absorbers [7]. We choose adiabatic absorbers because they offer the equivalent performance of PMLs, but avoid the use of more complicated update equations and additional auxiliary fields. We do not support periodic boundary conditions.
- On the other hand, our implementation also halves the number of bytes needed to be loaded per field value, by utilizing 16-bit floating-point representations for the field values.

Because the performance of an FDTD simulation engine does indeed depend on simulation-specific details, such as the size of the simulation domain, the number of timesteps, the source profile, and the output volume and format, we choose to present the performance of a stripped-down version of our engine, in line with what others have done in the past [4]. To this end, we choose to present the performance of our prototype engine for a "bare-bones" simulation setup, which is devoid of sources or output fields, but does include permittivity and conductivity fields.

For such a setup, we find that our prototype achieves a performance of ~0.15 TCUPS. This represents an ~5.5 times improvement over the performance of the naive implementation, as measured at large simulation domains. In this sense, we have demonstrated that it may indeed by possible to achieve the goal of circumventing the order-of-magnitude loss in performance due to the relatively slow memory bandwidth of the GPU and the need to perform global synchronization.

By devising a systolic update scheme that confines updates to smaller subdomains, which individually fit within the L1 cache of the GPU, and by scanning these subdomains along the x–y–z–t diagonal to minimize the frequency of global synchronization operations, we have shown that it indeed may be possible extend the memory performance of the miniscule simulation residing only in the L1 cache of the GPU, to a large, practical-sized simulation domain only limited by the amount of global memory available to the GPU.

We attribute the remaining gap in performance to the fact that global synchronization operations are still required, that we have only been able to achieve partial overlap between the loading of boundary values from global memory and the update of the interior of the subdomain, and that there are still further optimizations that can be made to our implementation. At the same time, we acknowledge that the addition of sources and output functionality will degrade the performance of a fully working implementation.

CONCLUSIONS AND FUTURE WORK

The systolic FDTD update scheme presented here is a first step toward a future where high-throughput, large-scale photonic simulation capability is widely and generally available. We expect schemes, such as the one presented here, to be increasingly essential as the compute capabilities of processors continue to outpace memory and interconnect bandwidths in future systems. Such capabilities will likely come in the form of optimized computational kernels, which continue to piggyback on the massive compute power of modern AI supercomputing centers (such as the Nvidia H100 GPU, as in our case), but may also evolve into a new generation of customized hardware, which implement the systolic FDTD update scheme at the transistor-level, since in either case, the systolic FDTD update scheme will be pivotal in efficiently updating Maxwell's equations in the time domain.

We have motivated the need for such a systolic FDTD update scheme by showing both theoretically and empirically that the memory bandwidth of compute systems is a very significant bottleneck to such a memory-intensive algorithm as FDTD, and we have shown how such a systolic update scheme can be used to circumvent the global memory bandwidth bottleneck of the GPU to enable a large-scale simulation of Maxwell's equations with performance mostly limited by the L1 cache bandwidth.

The future of photonics is bright, and as it continues to play an increasingly critical role in the continued development of the AI compute stack, we believe that the availability of increasingly powerful photonic simulation engines will also be an increasingly critical part of the photonic engineer's toolbox. To that end, we have presented a general strategy as well as a practical prototype for performing large-scale FDTD simulations without being limited by the memory bandwidth of the global memory pool that such large-scale simulations must necessarily draw from.

ACKNOWLEDGMENT

We thank the Advanced Research Projects Agency-Energy DIFFERENTIATE program's critical support for this effort.

AUTHOR INFORMATION

Jesse Lu (jesselu@spinsphotonics.com) is with SPINS Photonics, Inc., Dublin, CA 94568 USA. His research interests include nanophotonic inverse design. Lu received his Ph.D. degree in electrical engineering from Stanford in 2013.

David Qu (davidqu@spinsphotonics.com) is a software engineer at SPINS Photonics Inc., Dublin, CA 94568 USA. Qu received his B.A. from the University of California, Berkeley. His research interests include building scalable computational infrastructure for nanophotonic simulation and optimization.

Jim Qu (jimqu@spinsphotonics.com) is a founding engineer at SPINS Photonics Inc., Dublin, CA 94568 USA. Qu received his B.A. in computer science from the University of California, Berkeley, and specializes in machine learning for nanophotonic inverse design.

Ryan Fong (ryanfong@spinsphotonics.com) is with SPINS Photonics Inc., Dublin, CA 94568 USA, leading company business strategy and partnerships.

Geun Ho Ahn (gahn@stanford.edu) is with Stanford University, Stanford, CA 94305 USA.

Jelena Vučković (jela@stanford.edu) is the Jensen Huang Professor of Global Leadership and professor of electrical engineering at Stanford University, Stanford, CA 94305 USA, where she leads the Nanoscale and Quantum Photonics Group. She is a cofounder and lead scientific advisor of Spins Photonics Inc., Dublin, CA 94568 USA. She is a Fellow of IEEE.

REFERENCES

[1] N. Jouppi et al., "Tpu v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings," in *Proc. 50th Annu. Int. Symp. Comput. Archit.*, 2023, pp. 1–14.

[2] V. Levchenko, A. Perepelkina, and A. Zakirov, "Diamondtorre algorithm for high-performance wave modeling," *Computation*, vol. 4, no. 3, 2016, Art. no. 29, doi: 10.3390/computation4030029.

[3] C.-H. Lu et al., "High bandwidth and energy efficient electrical-optical system integration using COUPE technology," in *Proc. IEEE 74th Electron. Compon. Technol. Conf. (ECTC)*, Piscataway, NJ, USA: IEEE Press, 2024, pp. 893–897, doi: 10.1109/ECTC51529.2024.00144.

[4] M. Minkov, P. Sun, B. Lee, Z. Yu, and S. Fan, "GPU-accelerated photonic simulations," *Optics Photon. News*, vol. 35, no. 9, pp. 44–50, 2024, doi: 10.1364/OPN.35.9.000044.
[5] "NVIDIA H100 GPU," NVIDIA, Santa Clara, CA, USA, 2024. [Online]. Available: https://www.nvidia.com/en-us/data-center/h100/
[6] D. A. Orozco and G. R. Gao, "Mapping the FDTD application to Many-Core chip architectures," in *Proc. Int. Conf. Parallel Process.*, Piscataway, NJ, USA: IEEE Press, 2009, pp. 309–316, doi: 10.1109/ICPP.2009.44.
[7] A. F. Oskooi, L. Zhang, Y. Avniel, and S. G. Johnson, "The failure of perfectly matched layers, and towards their redemption by adiabatic absorbers," *Opt. Express*, vol. 16, no. 15, pp. 11,376–11,392, 2008, doi: 10.1364/OE.16.011376.
[8] A. Y. Piggott, J. Petykiewicz, L. Su, and J. Vučković, "Fabrication-constrained nanophotonic inverse design," *Sci. Rep.*, vol. 7, no. 1, 2017, Art. no. 1786, doi: 10.1038/s41598-017-01939-2.
[9] M. Wade et al., "TeraPHY: A chiplet technology for low-power, high-bandwidth in-package optical I/O," *IEEE Micro*, vol. 40, no. 2, pp. 63–71, Mar./Aug. 2020, doi: 10.1109/mm.2020.2976067.
[10] K. Yee, "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media," *IEEE Trans. Antennas Propag.*, vol. 14, no. 3, pp. 302–307, May 1966, doi: 10.1109/TAP.1966.1138693.