MICRO: G: A Scalable Memory-distributed Architecture for Memory-bound Applications

Marcelo Orenes-Vera (movera@princeton.edu), ACM ID: 6613993, Graduate
Computer Science, Princeton University, Princeton, NJ, 08540

1 PROBLEM AND MOTIVATION

Due to the growing size of AI models, computationally-intensive workloads, such as neural networks, are increasingly adopting sparse data structures to reduce memory footprints. Sparse structures, also central to graph analytics, recommendation systems, and computational chemistry, have exposed the limitations of computer systems when processing irregular, data-dependent memory accesses. These **limitations** arise from a combination of factors, including low compute-per-data resource ratios, inefficient memory hierarchies, and poor utilization of processing elements (PEs) due to atomic accesses, synchronization, and load imbalances.

Prior work in tackling these challenges has explored hardware-based graph processing pipelines [4, 5, 14, 23, 25], decoupling and prefetching [9, 13, 18, 28], and coalescing [11] to mitigate memory latency and reduce atomic update serialization. However, these approaches have not effectively addressed the problem of costly data movements and memory bandwidth bottlenecks. Processing-in-memory (PIM) proposals have attempted to leverage higher memory bandwidth by processing data near DRAM [1, 30], but their memory integration and dataset placement lead to load imbalances and synchronization overheads.

We identified **opportunities** to improve sparse data-structure traversal by minimizing data movement, exploiting both data and pipeline parallelism and avoiding serialization. To achieve these goals, we introduce Dalorex [21], a data-local execution model and distributed-memory architecture, where a program is split into tasks based on pointer indirection, and each task is executed on the core co-located with the memory region that the task operates on. Our hardware design supports efficient routing and scheduling of fine-grain tasks, efficiently enabling our data-local execution model. This results in system co-design that achieves near-linear speedup, as well as huge energy improvements, when scaling up to a million PEs without any dataset pre-processing.

Fig. 1 illustrates how Dalorex (right panel) minimizes data movement compared to architectures with hierarchical shared-memory structures (left panel). Instead of bringing data to the cores, Dalorex sends task-invoking messages to the tile containing the data to be processed next. This invocation message is one-way only, so the programming model is closer to dataflow than a shared-memory model. In contrast, architectures with a traditional memory hierarchy, sparse applications result in redundant data movement: irregular and long data-reuse distances lead to cache thrashing and over 50% cache miss rate. [9]. Dalorex design additionally offers the advantage of avoiding the need for cache coherence, virtual memory and atomics.

In addition to our data-local execution model and hardware support, we propose DCRA [22], a chiplet-based fabrication path that facilitates post-silicon integration of compute, memory, and I/O chiplets, optimizing chips for different targets of cost, energy

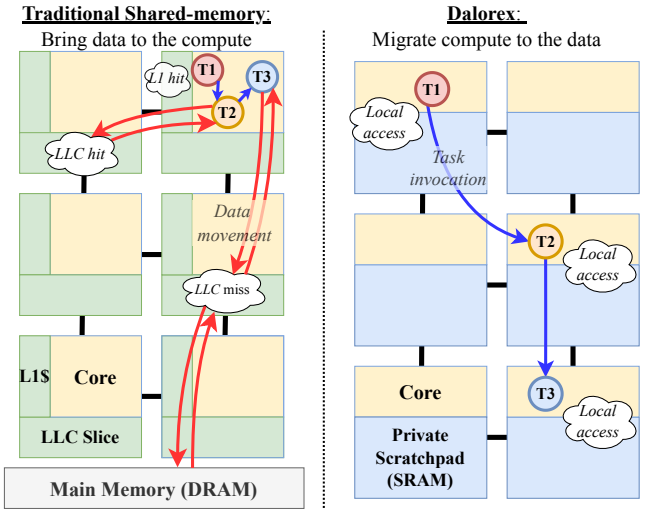

Figure 1: Program execution of three sequential graph-processing steps in a cache hierarchy (left), and Dalorex (right). Instead of moving data with little reuse, Dalorex invokes tasks where the data is local—reducing movement.

efficiency, and performance. Our design makes the network-on-chip (NoC) and memory hierarchy reconfigurable at the micro-code level, allowing for arbitrarily-sized NoC across chiplets and chips.

Our work re-thinks the way computers process sparse data structures by introducing innovations across the hardware-software stack, including a viable path for chip manufacturing. By addressing the limitations of prior work and exploiting new opportunities, our approach can improve the scalability and energy efficiency of systems handling sparse data structures in various domains, from AI to graph analytics and beyond.

2 BACKGROUND AND RELATED WORK

Dalorex aims to accelerate applications with memory-bound, irregular access patterns due to pointer indirection. Graphs are represented using adjacency matrices, with rows/columns representing vertices and values corresponding to weighted edges. Due to the sparse nature of these matrices, they are stored in formats like Compressed-Sparse-Row (CSR). Non-zero elements are accessed via pointer indirection.

Fig.2 presents the sequential code for Single Source Shortest Path (SSSP) and highlights the code sections that would be split into Dalorex tasks (at each level of pointer indirection). In a regular memory hierarchy, accesses to neighbor vertex data in the innermost loop (`line 8`) result in many cache misses and costly DRAM accesses. Moreover, the source vertex data (`lines 3 and 4`) and the neighbor index (`line 6`) are also accessed indirectly, and the cache's utility is limited.

```
1  while not frontier.isEmpty()
2    parallel for (v : frontier)
3  T1  node_dist = dist[v]
4      neigh_begin, neigh_end = ptr[v], ptr[v+1]
5      for i in range(neigh_begin, neigh_end):
6  T2    neigh_id = edge_idx[i]
7        new_dist = node_dist + edge_values[i]
8        curr_dist = dist[neigh_id]
9  T3    if (new_dist < curr_dist):
10         dist[neigh_id] = new_dist
11         new_frontier.push(neigh_id)
12 frontier = new_frontier
13 new_frontier = []
```

Figure 2: Pseudocode of the Single Source Shortest Path (SSSP) algorithm, and how it is split into Dalorex tasks based on indirect memory accesses. The colored variables determine the tile at which the next task is executed.

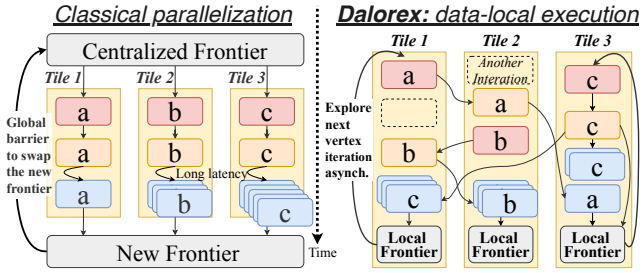

Figure 3: Program order for Bulk Synchronous Parallel (BSP), left, versus Dalorex, right. BSP leads to work imbalance since Task2 (orange) may generate several Task3 (blue). Columns show the tasks that are executed in each tile (arrows depict program order). Colors indicate task type based on the code of Fig. 2. Letters represent different vertex iterations. Dotted boxes depict that tasks from other iterations may interleave.

As Fig. 3 illustrates, Dalorex allows spatial interleavings within vertex iterations while preserving program order via sequential invocation of tasks. The classical time-wise interleavings of parallel-loop iterations are also allowed. Instead of using a centralized frontier, Dalorex uses local frontiers, which removes the synchronization overheads of frontier insertions and global barriers. As a result, our novel programming model maximizes program parallelization.

Prior work in graph processing can be divided into hardware and software techniques. Software techniques include prefetching [2, 28], and decoupling [9, 13, 18] to overlap data access and computation by running ahead in the loop iterations to bring the data asynchronously. These approaches perform program slicing on each software thread, creating a software pipeline effect [10, 27]. Others have proposed accelerators to perform the graph search as a hardware pipeline [5, 14, 23, 25]. Polygraph [4] generalizes prior accelerator designs to perform any of their algorithmic variants and optimizes work efficiency based on dataset characteristics, and Fifer [14] offers dynamic temporal pipelining to achieve load-balancing. Although effective for hiding latency and increasing load-balancing, these approaches remain highly energy inefficient due to excessive data movement and are ultimately limited by DRAM bandwidth. To increase memory bandwidth and reduce data movement, Tesseract [1] proposes in-memory graph processing by introducing cores into the logic layer of a 3D Hybrid Memory Cube (HMC)[6, 24]. Tesseract executes remote calls at the cores located near the data, similar to the execution-migration literature[8, 26]. However, their performance is limited because their vertex-based data distribution causes load imbalance since the highly-variable vertex degree in graphs causes a different workload per core; Tesseract remote calls are interrupting, incurring heavy penalties, and GraphQ's solution to overcoming this employs barriers for batch communication [30], causing high synchronization overheads.

3 APPROACH AND UNIQUENESS

In Dalorex, data arrays are scattered across the processing tiles as equal-sized chunks. A tile comprises a PU, an SRAM memory, and a router. Each PU can only access its fraction of the dataset, i.e., there is a single data owner. Because of that, a program must be split at each indirect memory access to create a sequence of tasks executed at the tiles containing the data to be operated on.

In our task-based model, the original program is split into tasks that are executed at the tile co-located with the memory region that the task operates on. A task with dependent tasks can spawn them by injecting inputs for each task in the output queue (OQ), which drains into a logical network channel (one per task type). The entire program is executed as a sequence of tasks, with no concept of execution threads (no main task). The parallelization level is determined by the number of tiles with tasks ready to execute, i.e., waiting on an input queue (IQ). There is one IQ per task type. An IQ is populated with invocation parameters created by either a prior task executed in the local tile or by incoming task messages at the network channels. While some spawned tasks are likely to be in the nearby tiles due to data placement policy, for others, the destination may be anywhere at random in the tile grid. As the size of the grid increases, so would the average number of router hops of a task invocation. To avoid distance communication as the grid size increases, DCRA introduces a hierarchical routing scheme based on the usage of proxies.

Tile Grid and Network Routing: The grid's size, hence the number of tiles a program is to run on, is determined by the user at compilation time. The logical tile ID is set by the microcode when a workload is selected to run on a grid, and it is used for XY routing. Since the dataset is statically partitioned across the tiles on the grid, and the first parameter of a task message is a global index to a data array, this index is used to route the message, avoiding message headers altogether. Based on the size of the array associated with each logical NoC for routing purposes, the router selects the bits that indicate the destination tile ID.

Task Prioritization: Work efficiency and PU utilization highly depend on the order of task executions. To make task invocations non-blocking and non-interrupting, the inputs for tasks arrive at a tile's task-specific queues through a unit called Task Scheduling Unit (TSU). Figure 4 shows the main components of each tile, and zooms into the TSU. TSU's main role is to determine the order of execution of tasks based on the occupancy of queues. Priority is given to tasks whose IQ is highly populated, or OQ is empty. This allows the tile to sense the network pressure and execute tasks that will relieve pressure when it is high (IQs are full) or increase it when it is low (OQs are nearly empty).

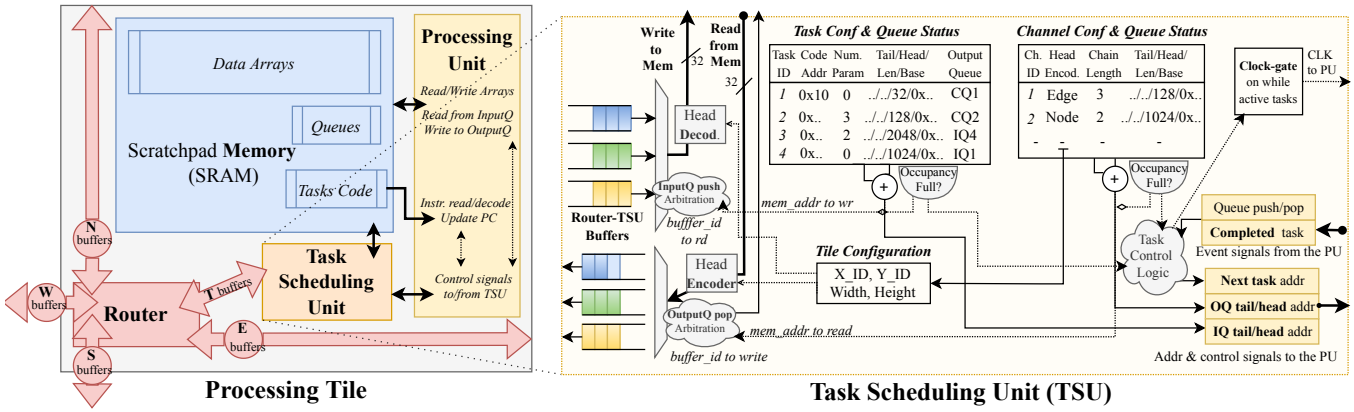

Figure 4: Tile organization. TSU feeds the PU the next task to execute based on the occupancy of the task queues. TSU uses SRAM to write incoming network data (push) to the IQs and reads (pop) outgoing data from the channel queues (CQs).

Reducing Network Contention by using Proxy Regions: In addition to the dataset chunking of Dalorex, DCRA introduces proxy ownership for tiles other than the owner by temporarily storing some remote data. We first define *proxy regions* by subdividing the tile grid. We distribute the proxy ownership for a select data array across the tiles of each proxy region such that the most recent updates coming from within the region can be stored inside the local proxy tile before being sent to the true owner. The purpose of proxy regions is twofold: it reduces network contention by reducing the average number of hops between task invocations and smooths out workload imbalance by allowing tiles other than the data owner to execute tasks.

Proxy Coherence: In many sparse algorithms, data updates follow the shape of reduction operations, i.e., they can happen in any order and the output would remain correct. This allows for loop iterations to be carried out independently of each other forming the basis of BSP where the output is guaranteed to converge to the correct values. We leverage these properties to reduce the number of tasks executed on the owner tile. The reduction is twofold: updates are not always successful (e.g., minimization in SSSP example), and several updates can be merged before being sent to the owner. In our design, proxy updates are propagated to the owner as either write-through (every time the value changes) or write-back (when proxy data gets invalidated).

Memory Hierarchy: The SRAM device on each tile is configured as a scratchpad or as a cache. To minimize the hardware and energy overhead of using the SRAM as a cache, we make the caches directly mapped. This cache mode is used to configure the *proxy cache* (P$), which holds proxy data, and the *data cache* (D$), which caches the dataset arrays when the package has integrated HBM [7]. When scaling out the parallelization of a dataset, if the memory footprint per tile is so low that everything fits in the local SRAM, the D$ would not be configured, and the PUs would access the data arrays as a scratchpad. (Both the memory controller and the HBM would be switched off). When using the cache mode, D$ misses fetch from the HBM without coherence checks, as data is not shared across tiles; D$ uses a write-back policy for evictions. For P$ misses, a default value is returned. Evictions are either write-through or write-back, based on the proxy region configuration. *Prefetching* is employed to minimize the impact of D$ misses. The TSU uses task message information to prefetch necessary data, and a next-line prefetcher is enabled when tasks access more than one element in a streaming pattern.

Chip Package reconfigurability: Fig. 5 shows two possible integrations of a chip package and the placement of chips within a PCB In terms of **memory**, whether to integrate HBM on the chip is a package-time decision based on the target metrics of the final product. Selecting the amount of HBM attached to a DCRA die depends on what level of parallelism is expected. If the chip is always expected to parallelize a dataset to the limit of strong scaling, HBM might not be necessary. In contrast, with target metrics like performance-per-watt, larger on-package memory capacity is essential. The 2D-torus **network** design facilitates more uniform utilization in dimension-ordered routing. To make implementation practical on 2D silicon, the torus is folded: one set of links connects the even-numbered tiles, and the other set of links connects the odd-numbered ones. The torus can be arbitrarily large, i.e., be confined within a die or span multiple dies, packages, or boards. We can reconfigure a 2D-torus interconnect to become two 2D-mesh networks by not connecting the wrap-around links on the routers at the edges. While bringing the data inside the package from disk, they both would be configured as a mesh to enable data streaming from the I/O dies. **Off-chip bandwidth** is determined during packaging based on the product's requirements. Thus, DCRA chiplets remain agnostic to specific protocols, delegating interconnect design between packages to I/O chiplets.

4 RESULTS AND CONTRIBUTIONS

Methodology: We used the following sparse workloads to evaluate our work: PageRank, Breadth-First Search (BFS), Single-Source Shortest Path (SSSP), Weakly Connected Components (WCC), Sparse Matrix-Vector Multiplication (SPMV), and Histogram. We used real-world networks and synthetic datasets of increasing sizes with up to 67 million vertices (V) and 1.3 billion edges (E), and a memory footprint of up to 12GB. Dalorex [21] introduced a cycle-level simulator to evaluate the performance and energy usage of the data-local execution model when the entire dataset is placed on SRAM, and DCRA [22] extended it to support data and proxy caches, the HBM model, three levels of network interconnect (between tiles, dies, and packages), and a cost model. (Details on the full paper).

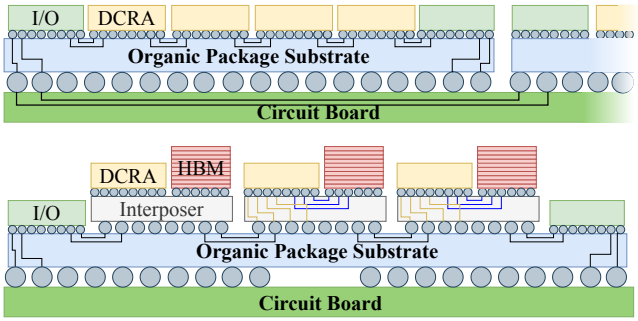

Figure 5: Two possible configurations: (top) two packages on a board, each featuring only DCRA chiplets, which optimizes for time-to-solution as it aims to parallelize a dataset much as possible (lower data footprint per die); and (bottom) a single package with DCRA chiplets and stacked DRAM for extended memory capacity, optimizes for performance-per-watt/$. (Insights based on our evaluation results.)

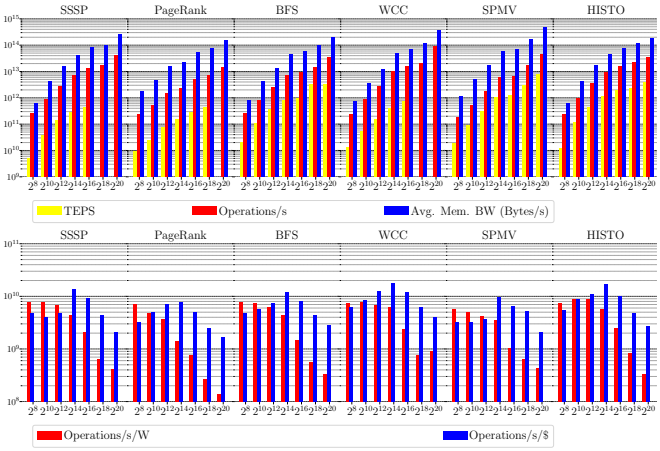

Figure 6: Throughput in operations, traversed edges per second (TEPS), and the average on-chip memory bandwidth needed to achieve that. Note that a couple of datapoints without TEPS are partial runs that the simulator could finish. Partial runs may show higher throughput as they have not reached the *cool-down* ending phase. The X-axis is the size of the tile grid used when analyzing strong scaling RMAT-26, ranging from 256 to over a million tiles. The Y-axis is logarithmic for both plots. The bottom plot shows throughput as a function of power and cost. Higher is better.

Comparing With Prior Work in PIM: Dalorex demonstrates significant performance and energy consumption improvements over prior work in PIM, such as Tesseract, by leveraging a combination of innovative architectural and software features. When compared to Tesseract, both using 256 cores (more evaluation details on the full paper [21]), Dalorex achieves up to two orders of magnitude improvement by employing: (1) a tile-based distributed-memory architecture that equally distributes data across processing tiles, ensuring all memory operations are local; (2) a task-based parallel programming model that enables tasks to be executed by the processing unit co-located with the target data; (3) An optimized network design for irregular traffic, featuring one-way communication and eliminating routing metadata from messages; (4) Novel traffic-aware task scheduling hardware that maintains high core utilization; (5) A data-placement strategy that enhances work balance. These innovations allow Dalorex to provide unprecedented performance for executing graph algorithms while maintaining ISA programmability for other domains.

Strong Scaling Up to a Million Tiles: When scaling to very large configurations, we leverage the proxy regions introduced in DCRA[22]. All our results use dies with 16×16 tiles, as that size is practical because it matches common HBM chiplets [7] ($110mm^2$) and leads to a good fabrication yield. The datapoints with grid sizes over 2^{12} tiles in Figure 6 use multiple chip packages of 64×64 tiles (each with 16 DCRA chiplets).

Throughput-per-watt likes small grids: From Figure 6 (bottom), we observe that when throughput-per-watt is the critical metric, staying within the smallest configuration that fits the dataset is the correct choice. The RMAT-26 dataset fits entirely on a 16×16 die with 8GB HBM. Throughput-per-watt drops significantly after using 2^{12} tiles, i.e., a chip package of 64×64 tiles. The inter-package links are more power-hungry than the links inside the package, hence, the drop in efficiency.

Throughput-per-dollar increases during superlinear performance scaling: Optimal throughput-per-dollar peaks at a 128×128 configuration (4 chip packages). This is because while the cost grows linearly with the number of chips, the performance grew super-linearly until that scaling step, where the decrease in footprint-per-tile also decreased the pressure in the memory controller of the HBM. At 128×128, RMAT-26 fits in SRAM. Beyond that, the gradually smaller footprint-per-tile reduces PU utilization since there are less parallel tasks to be executed. This leads to a less-than-linear scaling that makes the throughput-per-dollar to decrease.

Strong scaling for faster time-to-solution: For purposes of this experiment, we take the extreme approach of parallelizing a dataset with 2^{26} vertices (and $\sim 2^{28}$ edges) with up to 2^{20} tiles. This demonstrates that Dalorex+DCRA is suited for strong scaling, although it is not the most efficient way to use it.

Petabyte/s of memory bandwidth: We mentioned at the beginning of the paper that data-structure traversal have a high memory-to-compute ratio. We can observe in this plot how much memory bandwidth is required to maintain a high target throughput. Figure 6 (top) shows that for the 1-million-tile configuration, SPMV reads, on average, half a PB/s from their local memories. At peak, SPMV is reading 1.7 PB/s to perform 450 Teraops/s, of which 43 Teraops/s are dedicated to the multiplication of non-zero matrix elements. This is achieved by drawing 37KW across 256 chip packages (power density stays in the tens of mW/mm^2, air cooling).

Comparing with the state-of-the-art: From the Graph500 list, the current best performance for BFS on *RMAT-26* is 884 GTEPS—by the Tianhe Exa-node [12]. The next entry features an undisclosed *High Throughput Computer* with four NVidia V100-SXM2, which delivers 392 GTEPS. For RMAT-26, our work performs 3161 GTEPS on a 512×512 grid (256 chips) and 3323 GTEPS on a 1024×1024 grid (256 chips). Given the assumed chip I/O and the number of chips, bringing the dataset onto the chip would take < 0.1ms. For smaller datasets like *RMAT-22*, we found the fastest prior work to deliver 70 GTEPS [3] on a 32GB NVIDIA Tesla V100-SXM3, while our 64*x*64 configuration (single-chip) would achieve 362 GTEPS. (Note that DCRA assumes 1Ghz while this GPU runs at 1.6 GHz).

Contributions: In this work, we present a data-centric architecture that significantly improves the performance of sparse data applications, scaling to handle datasets with 2^{26} vertices across 2^{20} processing elements (PEs). This represents 2-3 orders of magnitude larger parallelization than existing work. Our architecture achieves this by utilizing the benefits of the Dalorex execution model and the DCRA reconfigurable tile fabric.

The key contributions of Dalorex [21] include: (1) A data-local execution model with equal dataset distribution across processing tiles, that maximizes work balance, avoids the serialization of atomic updates, and reduces communication overheads. (2) A tiled distributed-memory architecture connected by a network specifically designed for irregular fine-grain communication, utilizing data-driven headerless routing for enhanced efficiency. (4) A hardware unit (Task Scheduling Unit or TSU) that reduces task-invocation overheads and schedules tasks for optimal core utilization and work efficiency by sensing network traffic.

Moreover, DCRA [22] introduces several novel techniques to enhance work balance, scalable network communication, and reconfigurable memory hierarchy, such as: (1) Software-reconfigurable proxy regions that remove data-local architecture limitations, reducing distant communication of tasks and improving work balance. (2) A scalable architecture with post-silicon chip design decisions, such as on-chip memory, number of PEs, network topology, and off-chip bandwidth. (3) A practical path for manufacturing chip packages by composing DCRA dies, interleaving any number of compute and DRAM chiplets. (4) A detailed study of tradeoffs in cost, performance, and energy efficiency for various package designs. Through our evaluations (see full paper [22]), we demonstrate that: (a) Different packaging options of DCRA lead to different optimal design points for key metrics in high-performance computing (HPC), such as energy efficiency, cost, or throughput. (b) It is possible to construct a data-centric execution model exhibiting strong scaling performance up to a million PEs. (c) Our parallelization of the Breadth-First Search (BFS) algorithm is 3.8 times faster than the most performant entry of the Graph500 list for RMAT-26.

Overall, our work presents a novel data-centric architecture that enables massive parallelization without pre-processing or transforming datasets, achieving huge improvements in energy efficiency and performance for sparse data structure traversal.

Research Impacts: My Ph.D. has focused on building practical and efficient hardware-software co-design for sparse data structure traversal. This has led to seven first- and co-authored publications in premier Computer Architecture conferences (e.g., ISCA, ASPLOS, HPCA, ICS, DAC, ICCAD, ISPASS)[10, 18–21, 27, 29]. Especial thanks to my advisors, Margaret Martonosi and David Wenzlaff, and my collaborators Esin Tureci, Jonathan Balkind, Aninda Manocha, Tyler Sorensen, and the rest of the DECADES team.

My research achievements have been recognized with an IEEE Micro Top Picks' Honorable Mention for the hardware-software co-design proposed in MAPLE[18] and the Gold Medal in ACM/SIGMICRO student research competition (2022) for Dalorex [21]. Much of my work has been open-source and is being used by the research community [15–17]. In addition, MAPLE implementation was included in the DECADES chip tapeout (to appear at CICC). Aspects of my work have contributed to Cerebras Systems and AMD Research via two internships on hardware and software co-designs for memory- and communication-bound applications.

REFERENCES

[1] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2015. A scalable PIM accelerator for parallel graph processing. In *42nd ISCA*.

[2] A. Basak, S. Li, X. Hu, S. M. Oh, X. Xie, L. Zhao, X. Jiang, and Y. Xie. 2019. Analysis and Optimization of the Memory Hierarchy for Graph Processing. In *HPCA*.

[3] Luk Burchard, Johannes Moe, Daniel Thilo Schroeder, Konstantin Pogorelov, and Johannes Langguth. 2021. iPUG: Accelerating breadth-first graph traversals using manycore Graphcore IPUs. In *ISC*. Springer, 291–309.

[4] Vidushi Dadu, Sihao Liu, and Tony Nowatzki. 2021. Polygraph: Exposing the value of flexibility for graph processing accelerators. In *ISCA*. IEEE, 595–608.

[5] Tae Jun Ham, Lisa Wu, Narayanan Sundaram, Nadathur Satish, and Margaret Martonosi. 2016. Graphicionado: A high-performance and energy-efficient accelerator for graph analytics. In *MICRO*.

[6] Hybrid Memory Cube (HMC) Consortium 2018. Hybrid Memory Cube (HMC).

[7] Dong Uk Lee et al. 2020. 22.3 A 128Gb 8-High 512GB/s HBM2E DRAM with a pseudo quarter bank structure, power dispersion and an instruction-based at-speed PMBIST. In *2020 ISSCC*. IEEE, 334–336.

[8] Elliot Lockerman, Axel Feldmann, Mohammad Bakhshalipour, Alexandru Stanescu, Shashwat Gupta, Daniel Sanchez, and Nathan Beckmann. 2020. Livia: Data-Centric Computing Throughout the Memory Hierarchy *(ASPLOS '20)*. ACM.

[9] Aninda Manocha, Tyler Sorensen, Esin Tureci, Opeoluwa Matthews, Juan L Aragón, and Margaret Martonosi. 2021. GraphAttack: Optimizing Data Supply for Graph Applications on In-Order Multicore Architectures. *TACO* 18, 4 (2021).

[10] Opeoluwa Matthews, Aninda Manocha, Davide Giri, Marcelo Orenes-Vera, Esin Tureci, Tyler Sorensen, Tae Jun Ham, et al. 2020. MosaicSim: A Lightweight, Modular Simulator for Heterogeneous Systems. In *ISPASS*. IEEE.

[11] Anurag Mukkara, Nathan Beckmann, and Daniel Sanchez. 2019. PHI: Architectural Support for Synchronization-and Bandwidth-Efficient Commutative Scatter Updates. In *MICRO-52*.

[12] Richard C. Murphy, Kyle B. Wheeler, Brian W. Barrett, and James A. Ang. 2010. Introducing the Graph 500. http://www.graph500.org/specifications.

[13] Quan M Nguyen and Daniel Sanchez. 2020. Pipette: Improving Core Utilization on Irregular Applications through Intra-Core Pipeline Parallelism. In *MICRO*.

[14] Quan M. Nguyen and Daniel Sanchez. 2021. Fifer: Practical Acceleration of Irregular Applications on Reconfigurable Architectures. In *MICRO-54*. ACM.

[15] OpenSource. 2021. AutoSVA. https://github.com/PrincetonUniversity/AutoSVA.

[16] OpenSource. 2022. MAPLE. https://github.com/PrincetonUniversity/maple.

[17] OpenSource. 2023. Cohort. https://github.com/cohort-project/cohort-public.

[18] Marcelo Orenes-Vera, Aninda Manocha, Jonathan Balkind, Fei Gao, Juan L Aragón, David Wentzlaff, and Margaret Martonosi. 2022. Tiny but mighty: designing and realizing scalable latency tolerance for manycore SoCs.. In *ISCA*.

[19] Marcelo Orenes-Vera, Aninda Manocha, David Wentzlaff, and Margaret Martonosi. 2021. AutoSVA: Democratizing Formal Verification of RTL Module Interactions. In *Design Automation Conference (DAC)*.

[20] Marcelo Orenes-Vera, Ilya Sharapov, Robert Schreiber, Mathias Jacquelin, Philippe Vandermersch, and Sharan Chetlur. 2023. Wafer-Scale Fast Fourier Transforms. *To appear at ICS* (2023). https://arxiv.org/abs/2209.15040

[21] Marcelo Orenes-Vera, Esin Tureci, David Wentzlaff, and Margaret Martonosi. 2023. Dalorex: A Data-Local Program Execution and Architecture for Memory-bound Applications. In *2023 IEEE HPCA (HPCA)*. IEEE, 718–730.

[22] Marcelo Orenes-Vera, Esin Tureci, David Wentzlaff, and Margaret Martonosi. 2023. Massive Data-Centric Parallelism in the Chiplet Era. *Arxiv preprint* (2023). https://arxiv.org/pdf/2304.09389.pdf

[23] Muhammet Mustafa Ozdal, Serif Yesil, Taemin Kim, Andrey Ayupov, John Greth, Steven Burns, and Ozcan Ozturk. 2016. Energy efficient architecture for graph analytics accelerators. *ACM SIGARCH Computer Architecture News* 44, 3 (2016).

[24] J Thomas Pawlowski. 2011. Hybrid memory cube (HMC). In *2011 IEEE Hot Chips 23 Symposium (HCS)*. IEEE, 1–24.

[25] Shafiur Rahman, Nael Abu-Ghazaleh, and Rajiv Gupta. 2020. Graphpulse: An event-driven hardware accelerator for asynchronous graph processing. In *2020 53rd MICRO*. IEEE, 908–921.

[26] Keun Sup Shim, Mieszko Lis, Myong Hyo Cho, Omer Khan, and Srinivas Devadas. 2011. System-level optimizations for memory access in the execution migration machine (EM2). *CAOS* (2011).

[27] Tyler Sorensen, Aninda Manocha, Esin Tureci, Marcelo Orenes-Vera, Juan L Aragón, and Margaret Martonosi. 2020. A simulator and compiler framework for agile hardware-software co-design evaluation and exploration. In *ICCAD*. IEEE.

[28] Nishil Talati et al. 2021. Prodigy: Improving the Memory Latency of Data-Indirect Irregular Workloads Using Hardware-Software Co-Design. In *HPCA*. IEEE.

[29] Tianrui Wei, Nazerke Turtayeva, Marcelo Orenes-Vera, Omkar Lonkar, and Jonathan Balkind. 2023. Cohort: Software-Oriented Acceleration for Heterogeneous SoCs. In *ACM ASPLOS, Volume 3* (Vancouver, BC, Canada) *(ASPLOS 2023)*. ACM, New York, NY, USA, 105–117. https://doi.org/10.1145/3582016.3582059

[30] Youwei Zhuo, Chao Wang, Mingxing Zhang, Rui Wang, Dimin Niu, Yanzhi Wang, and Xuehai Qian. 2019. Graphq: Scalable pim-based graph processing. In *MICRO*.