In Proceedings of the 2024 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)

MuchiSim: A Simulation Framework for Design Exploration of Multi-Chip Manycore Systems

Marcelo Orenes-Vera, Esin Tureci, Margaret Martonosi, David Wentzlaff
Princeton University
{movera, esin.tureci, mrm, wentzlaf} @princeton.edu

***Abstract*—The design space exploration of scaled-out manycores for communication-intensive applications (e.g., graph analytics and sparse linear algebra) is hampered due to either lack of scalability or accuracy of existing frameworks at simulating data-dependent execution patterns. This paper presents MuchiSim, a novel parallel simulator designed to address these challenges when exploring the design space of distributed multi-chiplet manycore architectures. We evaluate MuchiSim at simulating systems with up to a million interconnected processing units (PUs) while modeling data movement and communication cycle by cycle. In addition to performance, MuchiSim reports the energy, area, and cost of the simulated system. It also comes with a benchmark application suite and two data visualization tools.**

MuchiSim supports various parallelization strategies and communication primitives such as task-based parallelization and message passing, making it highly relevant for architectures with software-managed coherence and distributed memory. Via a case study, we show that MuchiSim helps users explore the balance between memory and computation units and the constraints related to chiplet integration and inter-chip communication. MuchiSim enables evaluating new techniques or design parameters for systems at scales that are more realistic for modern parallel systems, opening the gate for further research in this area.

***Index Terms*—graph, sparse, communication-intensive kernels, simulator, design-exploration, multi-chip, scale-out, manycore**

I. Introduction

The end of Moore's law and the increase in application dataset sizes have driven the recent surge in development of scale-out manycore systems [2], [16], [18], [38], [43], [92], [95], [102]. Design-space exploration for these architectures at large scales employs roofline and analytical models to estimate performance, by relying on the deterministic nature of communications as well as computations for many ML and dense applications [52]. However, communication pattern in data-dependent communication-intensive (DCI) applications, such as graph analytics, sparse linear algebra, and database operations, is not deterministic, and therefore estimation of performance for running these applications on a system under design requires simulation tools. Current simulators cannot evaluate DCI applications at scale due to the limited scalability of detailed processing unit models. Thus, recent work on architecture designs for DCI applications are evaluated on relatively small scales, and their network design—whose performance impact would be exacerbated at larger scales—is not well explored [19], [20], [30], [53], [57], [61], [62], [65].

Current manycore architectures typically emphasize accelerating computation, dedicating significant chip area to it, at the expense of memory and network resources. However, when the workload is highly parallelized, data movement increasingly becomes the limiting factor for performance. This is especially important when using such manycore systems to accelerate communication-intensive applications such as graph analytics, sparse linear algebra, and spectral methods [66], [70], which necessitate a careful balance between computational, memory and network resources. Thus, in manycore systems with thousands to millions of cores, factors such as the number of cores per silicon die, the number of dies per package, memory hierarchy, network-on-chip (NoC) and inter-chip interconnect, become critical design considerations. The inability to evaluate these aspects at a large scale creates significant gaps between designed systems and their actual realizations.

To address these issues, we introduce MuchiSim[1], a parallel simulator for exploring the manycore architecture design space, with a special focus on communication-intensive applications[2]. MuchiSim scales up to millions of interconnected PUs by relying on user instrumentation for code runtime on the compute units while simulating data movement and communication every cycle. It also includes energy, area and cost models for multi-chip distributed systems; these are critical to reason about design tradeoffs. In addition, MuchiSim includes an application benchmark suite and data visualization tools to facilitate the analysis of its outputs.

As industry trends towards explicit data movement for large manycores due to the prohibitive cost of hardware-based coherence [2], [16], [92], [102], MuchiSim becomes particularly relevant. MuchiSim is applicable for architectures with distributed memory and software-managed coherence, which is becoming increasingly popular in modern scale-out systems. MuchiSim facilitates exploring the integration granularity and balance between memory and computation units, as well as constraints related to chiplet integration and inter-chip communication.

With a multifaceted analysis aimed at expanding the current understanding of manycore architectures for DCI applications, MuchiSim offers a flexible yet powerful framework for exploring many design configurations, thereby contributing to the design and optimization of future large-scale systems.

[1] MuchiSim is derived from the Spanish word *muchisimo* (very much), alluding to its ability to simulate a large number of PUs in a multi-chip setting. MuchiSim is pronounced as ”moo-chee-sim”.

[2] At large scale, memory-intensive applications lead to a communication bottleneck; we call this category of applications communication-intensive.

Our main technical contributions are:

- A novel performance modeling approach for distributed manycore architectures that allows for scalable simulations of data-dependent and communication-intensive applications across millions of PUs.
- Performance, energy, area and cost modeling of multi-chip module (MCM) and interposer-based integrations, critical in the design of scalable manycore systems.
- Support for different parallelization strategies (do-all and task-based) and communication primitives (e.g., message-passing and reduction trees).
- A benchmark suite of eight applications especially programmed for distributed scale-out systems.
- Visualization tools that allow comparing system-wide metrics for different evaluations (i.e., design configurations, applications and datasets) and analyzing per-PU metrics throughout the entire execution of a particular evaluation.

We evaluate MuchiSim and demonstrate that:

- MuchiSim is the first open-source framework that precisely simulates DCI applications with billion-element datasets parallelized across a million PUs within tens of hours.
- Its parallelization achieves linear speedups up to a host thread count equal to the number of columns of the manycore grid being simulated.
- MuchiSim closely matches runtime and area of the real runs of the Cerebras Wafer-Scale Engine when using their reported workload implementation and network specification.
- MuchiSim is a powerful design-exploration tool that helps identify optimal configurations for different metrics (e.g., performance-per-dollar) and applications.

The rest of the paper is organized as follows: §II provides background on existing simulation approaches and motivates our work; §III details the architecture class that MuchiSim simulates, the design of MuchiSim and capabilities, and the benchmark and visualization tool that it includes; §IV presents the validation and scalability analysis of MuchiSim, in addition to a case study that uses MuchiSim to study different memory integrations; and §V concludes the paper and discusses future work.

II. Background and Motivation

As the number of processing units (PUs) used to parallelize a workload increases, the network bandwidth and topology become critical factors in the performance of the system. Depending on the architecture, bottlenecks may also shift or worsen with the level of parallelism. Thus, detailed simulation of large manycore systems is crucial for design-space exploration, to avoid making suboptimal design decisions that may be hard to correct later in the design process.

The computer architecture community has seen a lot of promising research on hardware-software co-designs and optimizations for data-dependent communication-intensive (DCI) applications [1], [19], [24], [30], [35], [61], [62], [65], [67], [70], [73], [77], [79], [91], [97]. However, these ideas are often evaluated on a limited number of PUs (hundreds to a few thousand) or relatively small datasets (up to millions of data elements), as increasing either of these dimensions significantly increases simulation time. MuchiSim enables the exploration of architectural ideas on larger scales, as well as offers key parameters to explore the balance of hardware resources and power allocated to memory, network and compute.

Maximizing accuracy as well as performance is the main objective for the design and engineering an architecture simulator, as it dictates reliability as well practicality of design experiments [100]. For the rest of this section, we outline the critical features of a scale-out manycore system simulator (and their potential trade-offs) and review existing simulators.

A. Full-system vs. Application-level Simulation

Full-system simulators offer the ability to evaluate the performance of a target design in the context of other system components as well as a complete software stack including the operating system (OS). However, this comes at a great expense of speed and scalability even for small-scale systems. Nevertheless, when the execution significantly relies on OS and I/O processes, full-system simulators are essential when evaluating the performance of a design. Simulators such as GEM5, SimFlex, COTSon, RAMP Gold are some of the most commonly used full-system simulators with varying scope and capabilities [8], [13], [31], [93].

In the manycore architectures we are considering in this study, PUs are not expected to run the OS but rather behave as accelerator processing elements. These manycores save silicon area and power by not implementing hardware coherence and instead having an explicit view of the memory. In addition, PUs rely on software to orchestrate the data movement—optimized based on application specifics [16], [22], [70], [71], [95]. This makes an application/user-level simulator for these systems more appropriate.

Application-level simulators can scale to higher degrees of parallelism than full-system simulators by simulating up to a couple of thousand PUs [4], [8], [27], [56], [81]. However, existing simulators cannot efficiently simulate systems beyond this size, and some of them lack detailed network modeling, which significantly affects the accuracy of larger system simulations. BigSim [103] is a simulator originally developed to simulate Blue Gene that performs detailed network simulation after the emulation of the program on a real system to account for network traffic, achieving scaling in tens to hundreds of thousands of cores. BigSim relies on the program to be deterministic, but certain DCI workloads like graph algorithms do not converge deterministically unless additional synchronization steps are employed.

B. Simulating DCI Applications on Manycores

This paper focuses on a class of tiled, distributed manycore architectures. This class of architectures is composed of an interconnected number of machine nodes, each with a board of chip packages, further subdivided into chiplets and processing tiles. Therefore the entire compute system can be viewed as a hierarchically connected grid of tiles.

MuchiSim is a framework to explore the design space of this class of architectures, with a focus on DCI applications. The challenge with these applications is that their data-dependent execution requires functional simulation to precisely capture performance and energy usage. When parallelizing DCI applications across thousands of PUs and billions of data elements, faithfully simulating all architecture components every cycle becomes infeasible.

PU modeling: Since prior literature has identified the memory bandwidth and inter-PU communication to be the main bottlenecks of these applications, MuchiSim focuses on modeling the network and memory systems, while executing compute tasks natively on the host and relying on the user to provide a performance model for the compute time (e.g., cycles per basic block). A similar approach has been previously used by PriME [27], but PriME simplifies the PU model by simulating each instruction as 1 cycle, whereas MuchiSim can use that or a detailed user-provided model (§III). Because the PUs used in scale-out manycores do not interact with the OS and have a simple view of the memory system, it is possible to model the execution using user-provided cycle counts and without detailed core simulations. Simulators such as SimpleScalar [9], SimFlex [31], and Gem5 [13] offer detailed core modeling with a wide range of supported ISAs. In addition, simulators such as MosaicSim [54] and SimTRaX [85] provide LLVM-based instruction sets. When precise PU models are desired, these simulators can be used to provide the task-specific cycle counts to be incorporated into MuchiSim simulations.

Network modeling: Since inter-PU communication becomes a significant performance factor for large-scale manycore systems, the capability to model different configurations of the network-on-chip (NoC) and the inter-chip interconnect in detail is essential when modeling these systems. While there are several cycle-accurate NoC simulators such as Xpipes [11], NOXIM [15], SICOSYS [78], BookSim [36], among others [32], most of the existing multicore system simulators [27], [56], [88] do not model the NoC in detail. Although some implementations incorporate these NoC models within system simulators (e.g. Xpipes with Gem5), a functional architecture simulator (with cycle-accurate NoC modeling) able to simulate beyond a thousand PUs has not been offered so far. MuchiSim fills this gap.

C. Scope of Applicability

The class of architectures for which MuchiSim is applicable includes—but not limited to—tiled accelerators [5], [19], [69], [70], [104] and dataflow machines [2], [16], [22], [26], [95]. MuchiSim can also be used to simulate systolic array architectures like Google's TPU [38], but these designs usually execute deterministic workloads with near-neighbor communication, and thus they would be modeled much faster using analytical models.

Unlike general-purpose simulators for HPC such as SST [80], focusing on this architecture class allows MuchiSim to be a highly-scalable, parallel, light-weight simulator while providing detailed simulations of critical aspects of DCI workloads such as NoC modeling at flit-level granularity. MuchiSim could also potentially be used to model the performance of workloads running on more general-purpose manycores [21], [28], [50], [102] and server-class chip [6], [23], [59], provided that the application is written in a way that does not require hardware coherence.

III. MuchiSim Simulation Framework

In §III-A we first present the hardware components of MuchiSim's target architecture class, and how they are hierarchically organized. Then, §III-B explains the execution models supported in MuchiSim and how to describe applications. Next, §III-C elaborates on how MuchiSim simulates the execution of the applications on the target architecture.

MuchiSim is a parallel simulator written in C/C++ which does not use external libraries beyond threading, and thus, it can be compiled with any C++ compiler that includes OpenMP (`-fopenmp`) or pthreads support (`-lpthread`). The code can be found in our repository [68] under the `src` folder, where the `configs` subfolder contains the files to configure the system to simulate, and its latency, energy, area and cost parameters (described in §III-D and §III-E). The `gui` and `plot` folders contain the visualization tools to analyze and compare simulation logs (§III-F). The `apps` and `datasets` folders contain the benchmark suite of applications and datasets (§III-G). In addition, the repository includes a Readme file explaining how to configure and run MuchiSim for different experiments.

A. The Target Architecture Class

Throughout the paper, we refer to the simulated architecture as the *design under test* (DUT). We call the machine that runs the simulator, the *host*. We use *DUT's host* to refer to the machine for which the DUT may behave as an accelerator.

Figure 1 shows the hierarchy of MuchiSim's target architecture class. A cluster is composed of interconnected boards (∼nodes) that contain one or more chip packages (∼sockets). Each package is composed of one or several compute chiplets of any size (from a few mm^2 to as big as a wafer), optionally integrating DRAM chiplets with the compute chiplets. Each compute chiplet has a grid of tiles, where each tile contains one or more PUs, a private local memory in SRAM, a router, and a task scheduling unit (TSU).

Processing Units (PUs): A PU can be an ISA-programmable core, a CGRA unit, or a hardware accelerator, depending on the performance, energy and area models specified. We have only evaluated cases where the PUs are homogeneous across the chiplet, but MuchiSim could potentially simulate heterogeneous manycores [17], [22], [28] since a different performance model could be provided for different PU IDs.

Network-on-chip (NoC): MuchiSim supports evaluating 2D mesh and folded torus topologies [25] (with dimension-ordered routing) for the NoC that connects the routers of every tile. This NoC can span multiple chiplets and chip packages, up to the level of a cluster node. Several physical NoCs can be evaluated, with the same or different NoC widths. Additional

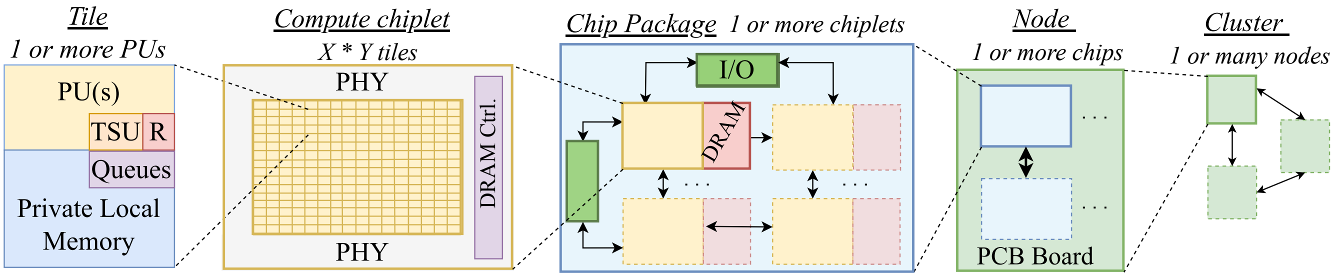

Fig. 1. Hierarchical overview of how tiles can be organized on MuchiSim's target architecture. The board of a cluster node may contain one or multiple chip packages, each with one or multiple chiplets. Packages can be composed of only compute chiplets (with a grid of tiles), or also DRAM chiplets (adjacent to the compute chiplets). Chiplets also include the physical layer (PHY) for inter-chiplet communication. A tile contains one or more processing units (PUs), a private local memory (PLM), a network router (R), and a task scheduling unit (TSU). The task queues are mapped into the PLM.

NoCs can be regular (i.e., connecting all routers) or Ruche [39], [72] (i.e., connecting to one for every *R* routers).

Routers generally have five bidirectional ports: North, South, East, West, and the PU ports (connected to the PU's input and output queues). When configuring hierarchical inter-chiplet [67] or Ruche NoCs, some or all routers have an additional set of cardinal ports (for a total of nine). MuchiSim can also evaluate the Tascade router support for asynchronous and opportunistic reduction trees [71].

Task Scheduling Unit (TSU): To support task-based parallelizations and message-triggered tasks (see §III-B), MuchiSim considers a TSU per tile, which takes messages from the input queues (one per task type) and schedules them to a PU. The TSU decides which task to schedule next—from the ones available at the queues—based on a scheduling policy. The policies currently supported are round-robin, priority-based (i.e., prioritizing tasks from a particular queue), or occupancy-based (to prevent full queues from backpressuring the network).

Private Local Memory (PLM): The size of each tile's memory is specified in Kibibytes (KiB); it is the same for all the tiles. MuchiSim supports using the PLM as a write-back cache, when DRAM is integrated on-package, or as a scratchpad (given the tile's local address space) when the chip's main memory is the tile-distributed SRAM. In cache mode, we model the area and energy of the tags (and valid/dirty bits) using part of the local SRAM; the width of a cacheline is equal to that of the bitline of the DRAM memory controller (set to 512 by default). Cache misses are directed to the on-chip memory controller, which fetches the full cacheline from DRAM directly since there is no hardware coherence in our target DUT. Dirty lines are written back to DRAM upon eviction.

Queues: Inputs and outputs for message triggered tasks (see III-B for details) are mapped into the PLM of each tile and currently modeled as circular FIFOs, which allows for compile-time configuration on the DUT. Thus, in our energy model, queue reads and writes are similar to load and store operations. Alternatively, one could change the model to consider them as separate structures, with the corresponding tradeoff in area and energy.

Prefetching: When DRAM is present in the design, MuchiSim supports modeling next-line and pointer-indirection data prefetching from DRAM into the PLM (used as a cache). The latter is enabled by the fact that tasks can be split at pointer indirection and the TSU can prefetch the data that tasks (waiting in the input queue) will use.

Chiplet Integration: The network routers at the edges of the compute chiplets are connected to the PHYs for inter-chiplet communication [7]. When DRAM is integrated on-package, MuchiSim models the pairs of DRAM and compute chiplets to be connected via a silicon interposer, while these interposers lay on an organic substrate (details in §III-D). Alternatively, MuchiSim can model the compute chiplets to be connected via a single silicon or organic interposer. In our model, the choice of the interposer mostly affects maximum communication bandwidth, area of the PHY, and energy per bit (see Table I).

Interconnect links: MuchiSim offers parameters to set the width and the number of interconnect links across chiplets (within a package) and across chip packages. Across cluster nodes, one can also set the multiplexing factor of the links between nodes, i.e., how many edge tiles share an inter-node link. To interconnect nodes, MuchiSim currently supports evaluating a mesh topology, but this could be extended to support other cluster-level topologies [3], [10], [12], [41], [47].

B. Describing and Mapping an Application

The code for an application and its potential DUT compile-time configurations are described as C/C++ header files in a structured manner using a template that defines a series of functions. The file for the application to simulate (`src/apps`) is included via preprocessor macros so that the simulator code is compiled together with the application code.

Configuration functions: The functions with the prefix `config_` are invoked only once at the beginning of the simulation. They allow overriding software parameters of the DUT for which the configuration files have set default values. These parameters include the sizes of the queues per task, as well as the parameters for prioritizing the scheduling of certain task IDs. Other software parameters of the DUT are macros that need to be set when compiling the simulator. MuchiSim uses pre-processor macros in some places to save branches and improve performance.

Address space and dataset layout: MuchiSim maps the memory address space contiguously such that the PLM of each tile is assigned a chunk of this contiguous space. The dataset arrays for a given application are allocated and mapped to memory across tiles (at config_app) depending on how a user or mapping technique [96] decides so. On our application suite, the dataset is scattered so that each tile has an equal chunk of each data array.

Message-triggered tasks (MTTs): The functions associated with an identifier (e.g., task1) describe the different types of MTTs that are invoked upon receiving a message into their corresponding input queue. Each MTT is associated with an IQ matching its identifier (ID). Defining MTTs enables simulating communication primitives like message passing, active messages, and non-blocking remote procedure calls (RPCs). Tasks can invoke other tasks by placing messages directly into another task's input queue (IQ)—if the task is going to be executed on the same PU—or into a channel queue (CQ)—if the task is meant to be executed on a different PU. Each CQ drains into its logical channel in the network, and each channel is associated with an IQ ID. To avoid network deadlocks, loops in the association between MTTs are not allowed (e.g., task1 invoking task2 and task2 invoking task1). In other words, the dependency chain must end on a leaf task (a task not invoking other tasks); a new phase of computation can start after a local or global barrier.

Init task: The function with the suffix _init is invoked once at the beginning of each kernel. The kernel_count variable can be used to distinguish behavior between different kernels. Multiple kernels can be simulated in sequence (with global barrier synchronization between them) to compose an application. The combination of the Init task and MTTs per kernel enables the simulation of a variety of programming models. However, recursion is not supported.

Supported parallelization modes: On the one extreme, the Init task can be empty if the entire kernel only comprises MTTs invoking each other, starting from a seed invocation coming from the DUT's host. On the other extreme, a kernel can be entirely included in the Init task (not invoking remote tasks). All task functions can access the identifier of the PU for which the task is being simulated and the size of the grid of PUs, and thus, kernels can be parallelized across PUs similarly to using *pthreads*. §III-C describes how MuchiSim iterates over every PU to check whether there are tasks ready to execute.

Result-check function: Since MuchiSim is a functional simulator, the application results can be compared with reference values. The compare_out function can be used to compare the outputs directly or write them into a file to be compared outside the simulator.

C. Simulating the Application Runtime

Although the tiles of the DUT may be organized hierarchically, for parallelization purposes, the simulator considers a global grid of tiles, of which each host thread simulates a slice of one or more columns. There are two types of threads, the ones that simulate the TSU and the PU tasks (*execution threads*) and the ones that simulate the network (*router threads*).

The ***execution threads*** iterate over every tile in their slice of the grid, checking if there are tasks ready to be executed in the tile's IQs, in addition to the Init task. If there are, the TSU selects the next task type to execute based on the configured policy. Then, the function associated with that task is executed, and the task delay is obtained from the latency-instrumented code. The codes included in our application suite are instrumented considering the execution of simple in-order PUs. However, users could change that to reflect other PU models. For memory operations, MuchiSim offers a special dcache function that returns the latency to fetch a given memory address, depending on whether it hits in the data cache, and the configured memory system. The function call simulating the task may have pushed messages into other IQs or CQs, with the corresponding timestamps, so they cannot be routed until the router thread is executed for that timestamp. Once the task function returns, the clock for that PU advances as many cycles as the task delay, and the execution thread continues evaluating the next PU. The execution thread keeps iterating over the PUs until there are no more tasks to be done, and the network is empty (i.e., no more messages to be routed). This models a hardware-based termination condition based on idleness, and by default, we set its latency to two times the diameter of the network.

The ***router threads*** iterate every cycle over every router in their slide of the grid. For every input port, they check whether there is a message to be routed and a free buffer slot in the destination port (the buffer size is also parameterizable). When two or more input ports want to route to the same output, the priority is determined via a round-robin policy (static priority could also be configured). A message is only routable if the timestamp (set at injection and updated every hop) is less than or equal to the current router cycle; the timestamps do not exist in the DUT, but they are used to allow PUs and routers to be simulated in parallel. The synchronization between the execution and router threads is done based on their wall clock time, i.e., a router can never be simulated ahead of its tile's PUs. In addition, a PU cannot start executing a new task until the router thread has caught up with the clock time of the slowest PU in the tile. Thanks to this type of synchronization, MuchiSim can seamlessly support different frequencies for the PUs and the network, with any ratio between them.

Frequency: The DUT configuration file in MuchiSim distinguishes between the peak frequency supported by the design and the operating frequency (at which the DUT is evaluated) for the PUs and the NoCs. Peak design frequency is going to affect the silicon area of the design, while the operating frequency affects the power consumption due to voltage scaling. The default model for this is described in §III-D). Peak and operating frequencies can be changed independently for PUs and NoCs, which is useful to evaluate architectures looking to support applications with different compute and network demands. For example, having a higher peak PU frequency allows raising its operating frequency when needed (e.g., to increase compute capacity) while lowering it to save power for applications with low arithmetic intensity. The configuration file has by default a peak and operating frequency of 1 GHz for all components.

SRAM model: The default latency, area and energy parameters of the SRAM memories are based on 7nm technology at 1 Ghz [99] (see Table I). Unlike the PU and NoC, SRAM memories have by design a narrower operating voltage and frequency range, therefore we do not support changing these and we only use the default values for which we have a reference (Table I). MuchiSim models scaling the size of SRAM memories by increasing the number of banks. Only the active banks consume static energy, and we model the energy of the multiplexor tree that selects the bank to grow by 50% at each doubling step. The access latency is also modeled to increase by 1 nanosecond at quadrupling steps beyond 512 KiB. All these modeling options and values can be changed in the simulator parameters. On the DUT configurations with multiple PUs per tile, the SRAM is shared among them, but bank conflicts are not modeled yet for this case.

DRAM model: Each compute chiplet may be paired with a memory device of a given number of channels. By default, MuchiSim considers an HBM2E device with eight 64GB/s channels. On the 2.5D integration shown in Figure 1, the memory controller sits on one edge of the chiplet, adjacent to the closely integrated DRAM device. The bus connecting the tiles to the memory controller is separate from the task-communication network. Since each memory channel is shared by many tiles, the contention is modeled by imposing that the memory channel can only take one request per cycle, and keeping the count of the transactions of each channel. For example, if a request is done at cycle X, but the memory channel has received Y transactions (where $Y > X$), then the delay if this request is $Y - X +$ the round-trip to the memory channel. Increasing the capacity of HBM increases the cost linearly (see §III-E) but not the device area since it is a 3D stack. The HBM bandwidth available to the processing units can be defined by changing the number of channels or devices integrated with each compute chiplet, or by changing the number of tiles or PUs per tile of these chiplets.

D. Energy and Area Model

Table I summarizes some of the energy, latency, and area parameters for communication links and memory technology, while the full set of default energy, area and performance parameters of MuchiSim can be found in our repo [68]. The simulator periodically logs performance and energy during the simulation, in what we call *frames*, at a rate that can be configured. This name will become more apparent in §III-F, which describes the visualization tools. Recording frames allows the user to observe the progress of the simulation, by visualizing several metrics (per PU or averaged) for each frame and aggregating until that frame. At the end of the simulation, the simulator also outputs statistics over metrics like throughput, average power, and network traffic at different levels of the hierarchy.

The files starting with `calc_` under the `src/common` folder contain the functions that calculate the performance metrics, energy, area and cost of the different DUT components based on the simulation of the application execution.

Because there is often no ground truth for energy and cost parameters but rather estimations or assumptions, ***MuchiSim allows post-processing a given simulation to re-calculate the energy and cost with different model parameters***. In addition to the execution log file, MuchiSim creates a separate file with many performance counters collected during the simulation process, such as messages hops and memory accesses at different levels, and instructions executed for each type. The counters file is then provided as an argument to our separate post-processing executable—which uses the same configuration and parameter files as the simulator, with potentially new values—to re-calculate the energy usage, and the DUT area and cost.

The model for how area and voltage grows with peak and operating frequency can be re-evaluated during post-processing. By default, MuchiSim simplistically models the area of the PUs and routers to grow by 50% of the increase in their peak frequency. This model can be refined by synthesizing particular RTL components with different peak frequencies and measuring their area. A simulation can be post-processed with a new area model to re-calculate energy and cost.

For our ***voltage scaling model***, we fit a ridge regression to the frequency and voltage data from the shmoo plots of chips with 5, 7 and 12 nm transistor nodes [28], [82], [83]. The current model grows voltage with $0.06 + 0.13 * freq + 0.06 * node$, which could be adjusted by adding data to `src/voltage_model.py`.

DRAM integration: The standard integration of DRAM and compute chiplets in MuchiSim is a 2.5D integration via a passive interposer [18], [60]. However, since the adjacent compute chiplet has dedicated access to the DRAM device, another possible integration would be having the DRAM device on top of the compute chiplet, which would need to be redimensioned to behave as an active interposer, in addition to posing a power and thermal challenge [14], [33]. MuchiSim supports studying this mode by adjusting the area, cost, and wire energy with the DRAM integration, but we do not consider a different latency. MuchiSim reports power density, which can be used to estimate the thermal feasibility of a 3D integration.

Chiplet PHY and Package I/O are affected by the width, frequency and topology of the on-chip and off-chip network, respectively. MuchiSim configures one physical network by default, but up to three independent NoCs have been evaluated (one for each of the task types of the benchmark suite, §III-G).

E. Cost Model

We also added a fabrication cost model to the simulator to study the cost-effectiveness of different architecture configurations. Similarly to the energy model, the cost model is decoupled from the runtime simulation process, i.e., cost and energy can be re-calculated post-simulation for different parameters. This is useful to study how price variations can change the cost-effectiveness of different DUT configurations.

Silicon cost: By default, we consider 7 nm technology for transistors; we assume that a 300 mm wafer with this transistor process costs $6,047 [37]. We obtain the cost per die by dividing

TABLE I
DEFAULT VALUES OF ENERGY (E), BANDWIDTH, LATENCY, AND AREA PARAMETERS OF LINKS AND MEMORY DEVICES MODELED IN MUCHISIM.

Memory Model Parameters	**Values**
SRAM Density	3.5 MB/mm^2 [99]
SRAM R/W Latency & E.	0.82 ns & 0.18 / 0.28 pJ/bit [99]
Cache Tag Read & cmp. E.	6.3 pJ [99], [101]
HBM2E 4-high Density	8GB on 110mm^2 (75 MB/mm^2) [46]
Mem.Channels & Bandwidth	8 x 64GB/s [46]
Mem.Ctrl-to-HBM Latency & E.	50 ns & 3.7 pJ/bit [40], [74]
Bitline Refresh Period & E.	32 ms & 0.22 pJ/bit [29], [87]
Wire & Link Model Parameters	**Values**
MCM PHY Areal Density	690 Gbits/mm^2 [7]
MCM PHY Beachfront Density	880 Gbits/mm [7]
Si. Interposer PHY Areal Density	1070 Gbits/mm^2 [7]
Si. Interposer PHY Beachfront Density	1780 Gbits/mm [7]
Die-to-Die Link Latency & E.	4 ns & 0.55 pJ/bit (<25 mm) [64]
NoC Wire Latency & E.	50 ps/mm & 0.15 pJ/bit/mm [42]
NoC Router Latency & E.	500 ps & 0.1 pJ/bit
I/O Die RX-TX Latency	20 ns [84]
Off-Package Link E.	1.17 pJ/bit (upto 80mm) [98]

the wafer cost by the number of good dies, which we calculate using 0.2 mm scribes, 4 mm edge loss, and 0.07 defects per mm^2. We integrate and validate [34] die yield calculations in our cost model using Murphy's model. When comparing the cost-effectiveness of the simulated architecture, we do not include the non-recurring engineering cost of the compute dies since all the options use the same technology.

Packaging cost: We model the cost of the 65 nm silicon interposer connecting a compute die with a DRAM device (including bonding) to be 20% of the price of a compute die [94], and the cost of an organic substrate to be 10% of the price of an equal-sized compute die, and the bonding to add an extra 5% overhead [45], [89].

DRAM cost: While the cost of HBM2 is not disclosed, we made an educated guess using public sources [37], [76]. By default, the cost model assumes $7.5/GB, which is more affordable than when HBM was first released in 2017. One could expect this price to decrease over time as more vendors fabricate HBM or the process matures [46], [55], [63], [75]. MuchiSim's post-processing tool allows evaluating the performance-per-dollar of a given simulation in the light of different DRAM cost scenarios.

F. Visualization Tools

MuchiSim comes with two data visualization tools: a command-line interface (CLI) tool to generate plots including multiple application executions, and a graphical user interface (GUI) tool—coded in PyQt5—to visualize MuchiSim metrics over time for a single execution. Both tools are written in Python and use matplotlib to generate the plots.

The CLI tool allows **plotting multiple metrics for combinations of DUT configurations and sizes, and different applications and datasets**. These metrics include runtime, throughput (FLOPS or TEPS), energy (and its breakdown), cost (in USD), simulator time, arithmetic intensity (FLOPS divided by data loads or network traffic), overall network traffic (in message hops), and cache hit-rate, among others. They can be plotted as absolute numbers (as in Figure 3) or normalized to a baseline configuration (as in Figure 5).

The GUI tool allows **visualizing performance counters throughout the application execution**, such as router port collisions and utilization, end-point contention, PU utilization, cache hit-rate and memory controller requests and average latency. For these metrics, we can plot various **statistics** such as the average, maximum and minimum values across all tiles as well as boxplot and standard deviation, to observe the work distribution throughout the execution. This time series also provides insights into the length and challenges of the tail of the execution, where the maximum and minimum values are far apart. In addition to plotting statistics about these performance counters, the GUI can also generate a **heatmap** of the tile grid, where the color represents the percentage of the duration of the frame (~microseconds) that the counter was activated. The frames are then played in sequence (by creating a GIF) to visualize the evolution of that metric on the tile grid over time. For example, Figure 2 shows that for the routing activity when using a mesh (top), a torus (middle), and a torus with reduction trees (bottom). The left panels show the routing activity and the right panels show the PU (core) activity. Overall, MuchiSim's visualization tool helps identify the bottlenecks of different applications and datasets, and how they change with different DUT configurations.

Verbosity: MuchiSim supports four levels of output verbosity: (v=0) only reports aggregated statistical metrics at the end of the execution; (v=1) reports these metrics for each time frame; (v=2) reports the metrics for each tile in the grid, which is required to plot heatmaps; (v=3) also shows the capacity of the input and output queues of each task type for every tile. Higher verbosity levels increase the log file size and the simulator runtime, proportionally to the configured frame rate.

G. Benchmark Suite

Applications: MuchiSim includes four graph algorithms, two sparse linear algebra, and two HPC kernels [90]. *Breadth-First Search (BFS)* determines the number of hops from a root vertex to all vertices reachable from it; *Single-Source Shortest Path (SSSP)* finds the shortest path from the root to each reachable vertex; *PageRank* ranks websites based on the potential flow of users to each page [44]; *Weakly Connected Components (WCC)* finds and labels each set of vertices reachable from one to all others in at least one direction (using graph coloring [86]); *Sparse Matrix-Vector Multiplication (SPMV)* multiplies a square sparse matrix with a dense vector. *Sparse Matrix-Matrix Multiplication (SPMM)* multiplies a square sparse matrix with a dense matrix and stores the result in a dense matrix. *3D Fast Fourier Transform (FFT)* computes the Fourier Transform of a 3D tensor. *Histogram* counts the values that fall within a series of intervals. The implementation of BFS, SSSP and WCC, supports running with local or global barrier synchronization

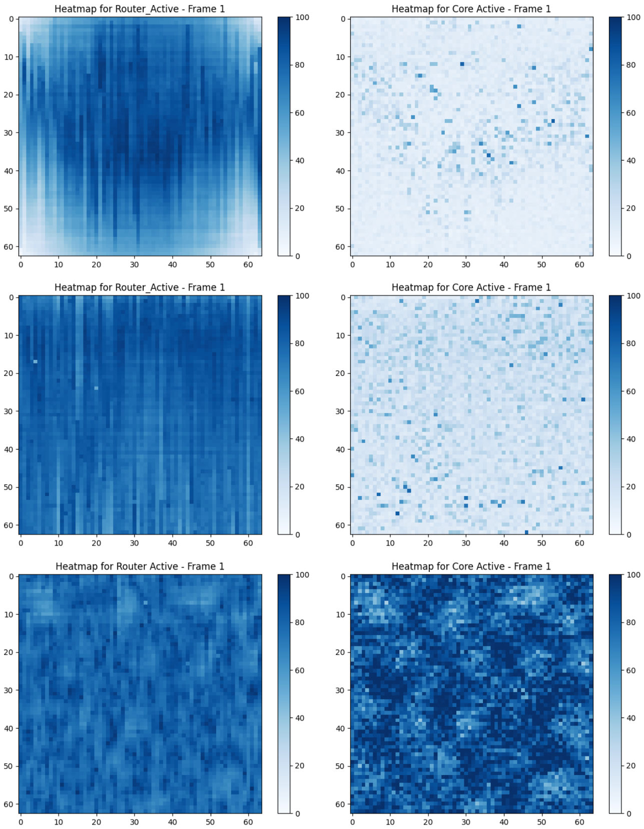

Fig. 2. Animation of the router activity when running BFS on RMAT-22 for three different NoCs: 2D mesh (top), 2D torus (middle), and 2D torus with reduction trees (bottom). The left panels show the routing activity and the right panels show the PU (core) activity. No router activity can mean that the router has no messages to route, or that the NoC is clogged and messages are stuck. The animation is composed of snapshots at a rate of a frame every 40 microseconds (this rate is configurable in MuchiSim). Since this rate is the same for these plots, the number of frames (50, 28, and 16, from top to bottom) is proportional to the execution time. The animation can be visualized by opening this PDF with Adobe. Visualizing the router and core activity simultaneously helps understand the effect of NoC congestion on core utilization. In addition, plotting the destination-port collisions helps understand the router activity. We evaluated the version of BFS with barrier synchronization at the end of each epoch to showcase the effect of the network on the tail execution time (3 major epochs can be observed). A finer time resolution allows observing the evolution of the execution in more detail, but it increases the size of the GIF.

at the end of each epoch, where the next vertices to explore are stored in the frontier.

As throughput, MuchiSim reports FLOPS (considering the dataset as arrays of FP32) and traversed edges per second (TEPS) as $TEPS = m/time$ where m is the number of edges connected to the vertices in the graph starting from the search key. When reporting TEPS for SPMV, SPMM, FFT and Histogram, MuchiSim considers the non-zero elements to multiply, and elements to process, respectively.

Datasets: MuchiSim includes ten datasets, including six sizes of the RMAT [48] graphs—standard on the Graph500 list [58]—RMAT-16, 21, 22, 25, 26, and 27, which are named after their scale. For example, RMAT-26 contains 2^{26}, i.e., 67M vertices (V) and 1.3B edges (E), and has a memory footprint of 12GB. We also include the real-world graphs from Wikipedia (V=4.2M, E=101M), LiveJournal (V=5.3M, E=79M), Amazon (V=262K, E=1.2M) and Twitter (V=81K, E=2.4M) [49]. For SPMV and SPMM, the graphs are seen as a square sparse matrix with V rows and columns and E non-zero elements. The graphs (as sparse matrices) are stored in the Compressed Sparse Row (CSR) format without any partitioning, i.e, the dataset contains three input arrays, one for the values of the non-zeros, one for the column indices of those non-zeros, and one for the pointers to the beginning of each row in the previous two arrays. For Histogram, the output array is the count of the column indices of the non-zeros.

IV. Results

This section first presents our validation of MuchiSim by simulating the Cerebras Wafer Scale Engine (WSE) [51] and comparing the results with an existing experimental evaluation [66]. Then, §IV-B shows the speedup of MuchiSim with the number of host threads and its throughput when simulating up to a million tiles. Finally, §IV-C presents a case study of MuchiSim to explore the optimal memory integration for a given application domain, for several target metrics.

Our repo [68] includes additional experiments and case studies that explore the impact of (1) the NoC topology, width and frequency, (2) the reduction-tree size (3) the number of PUs per tile and the PU frequency, (4) integrating DRAM or not, which sets the minimum parallelization required for a given dataset, and (5) the size of the tile input and output queues, among others.

A. Validation

We compare the runtimes reported on the WSE for Fast Fourier Transforms (FFTs) [66] with the runtimes obtained with our simulator. We perform simulations for all their reported datapoints, i.e., parallelizing FFT of n^3 tensors across n^2 processors, for n equal to 32, 64, 128, 256, and 512. In doing so, we show that MuchiSim has a high level of scalability and the accuracy is not impacted by the size of the DUT.

The WSE mounted on the CS-2 [51] is a 46,225mm^2 monolithic die with 850,000 cores and 40GB of SRAM on 7 nm technology. To model the WSE, we configure the DUT as a package with a single chiplet (32-bit 2D mesh NoC) with no integrated DRAM [51]. The WSE has a circuit-switched NoC with traffic filtering, while our simulator models a packet-switched NoC. The advantage of the circuit-switched NoC is to avoid message headers and simplify the routers, but it requires more synchronization when setting up a path to send data. To overcome this difference, we do not consider the destination header in this case, and rather consider the same amount of data sent by the PUs in both cases. The performance model for the PUs is set based on their reported numbers [66].

Runtime: The runtimes of the n^3 Fast Fourier Transform (FFT) problems parallelized across n^2 tiles reported on the WSE [66] are 1.2× of the runtimes reported by MuchiSim, consistently, for n ranging from 32 to 512 (i.e., from a thousand to a quarter million tiles). We believe the reason for this slightly optimistic runtime is that MuchiSim does not model

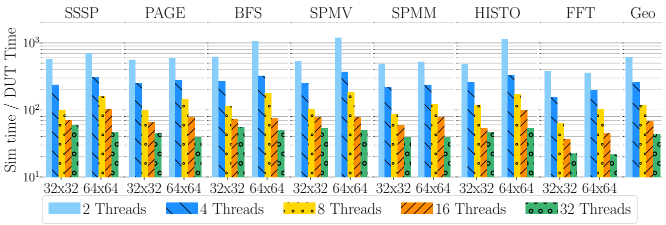

Fig. 3. Ratio between the simulator and the DUT runtime, for two DUT sizes (32x32 and 64x64 tiles, monolithic, connected via a 64-bit 2D torus), evaluated with an increasing number of host threads to process the same RMAT-22 dataset. The DUT time is considered as the aggregated runtime of all tiles. The simulator runtime decreases close to linearly with the number of host threads.

the synchronization overhead of the circuit-switched NoC of the WSE. We also simulated these datapoints with up to 32 host threads to measure the speedup of the simulator, shown in Figure 3 for FFT and other applications.

In terms of ***chip area***, the area reported by the simulator is 8.8% larger than the area of the WSE. We attribute this to WSE's NoC, where their routers are likely smaller than the default area of MuchiSim's routers. Regarding ***energy***, the simulator reports an average power of the processing tiles of a bit over 1 KW. While the WSE study did not report energy for the FFT evaluation, we know that the entire CS-2 system (including cooling) draws up to 20 KW [51]. Considering that 512x512 is less than a third of the CS-2, and that PU utilization is low (∼30%) because of the communication bottleneck of FFT, the energy estimation seems reasonable.

B. Parallelization Speedup and Scaling Throughput

Figure 3 shows the performance of MuchiSim with the number of host threads, for two DUT sizes. This is shown as the ratio between the host runtime for the simulator and the simulated runtime of executing these applications on the DUT runtime. This ratio decreases from a geomean ratio of 614 to 43 (a 12× speedup) across the datapoints, when scaling from 2 to 32 host threads (Intel Xeon Gold 6342 at 2.8Ghz). The speedup in MuchiSim's parallelization is linear until the point where each host thread only processes a couple of tile columns (32-thread 32x32 DUT). Figure 3 demonstrates the efficiency of MuchiSim and its parallelization. On a 32-thread host, simulating the execution of applications on the DUT takes only 43× (on geomean, and down to 21× for FFT) longer than the expected runtime of every tile of the DUT. This ratio is remarkable considering that MuchiSim performs a functional simulation of data-dependent workloads, where all NoC routers are evaluated every cycle at the flit level.

Figure 4 shows the simulation time and throughput of MuchiSim (in host seconds) when using increasingly large DUT configurations (from a thousand to a million tiles) to execute the RMAT-26 dataset. The first three datapoints see a decreased or nearly constant simulation time because we are increasing the parallelization of MuchiSim from 16 to 64 host threads. The NoC simulation throughput ranges from a few million message flits routed per second (PageRank) to over 40 million (SSSP). This NoC throughput does not count the routing attempts that fail due to contention (two input ports compete for the same output) or backpressure (the buffer of the output port is full). The throughput of operations—natively executed on the host—reaches up to a few billion Ops/s.

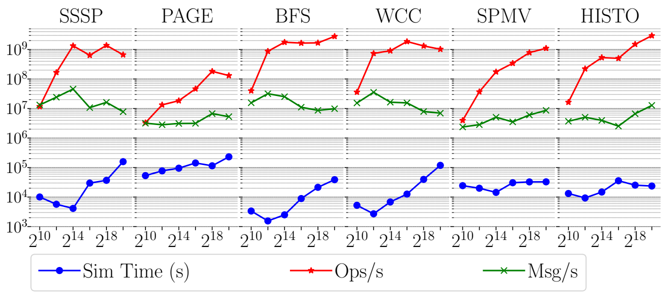

Fig. 4. Simulation time (in host seconds) and throughput in DUT operations and NoC message fits routed per second (y-axis), for scaling DUT sizes from a thousand to a million tiles (x-axis) when processing the RMAT-26 dataset. (This evaluation models 32x32 tiles per chiplet, connected via a 64-bit hierarchical 2D torus.) The 2^{10} and 2^{12} datapoints are evaluated with 16 and 32 host threads, respectively, of a single-socket Intel Xeon Gold 6342 at 2.8Ghz, the datapoints from 2^{14} to 2^{18} with 64 and 2^{20} with 128 threads, respectively, of a 4-socket Intel Xeon Gold 6230 at 2.1GHz.

The absolute runtimes in Figure 4 (blue) showcase that even million-tile DUT evaluations on a billion-element dataset—unattainable before MuchiSim—are simulated in under half a day for BFS, SPMV and Histo, and in up to two days for SSSP and PageRank, on a single host server.

FFT is not shown in Figure 4 since for this benchmark we scale the problem size with the system size (i.e., the FFT of a n^3 tensor is parallelized across n^2 processors) but we mention some runtimes here. FFT is simulated in under 100 seconds for up to $n = 128$ (i.e, 2^{14} tiles) and it takes around a day to simulate $n = 1024$ (a million tiles on a billion-element tensor), where the all-to-all communication of FFT dominates the runtime.

C. Case Study: Memory Integration

As a case study, we studied the performance, energy efficiency, and performance per dollar improvements of different SRAM sizes and numbers of tiles per chiplet (same number of tiles in total) and show the results in Figure 5. Since all applications but SPMM have a very low **arithmetic intensity** with respect to the data movement, they require a large amount of memory bandwidth. The top plot of (Figure 5 shows a strong performance increase with SRAM size, 3.5× on geomean when increasing from 64 KiB to 256 KiB, with the same chiplet configuration of 32x32 tiles, and an additional 2× increase when using 16x16-tile chiplets (32 Tile/Ch), for the same SRAM size of 256 KiB. The **hit-rate** of the data cache (not displayed) increases from 83% to 95% on geomean with the SRAM size increase, but since the effective bandwidth of a tile is $SRAM\_bandwidth \times hit\_rate + DRAM\_bandwidth \times (1 - hit\_rate)$, the hit-rate has a larger impact on the bandwidth when more tiles share the same DRAM channel (128 Tile/Ch).

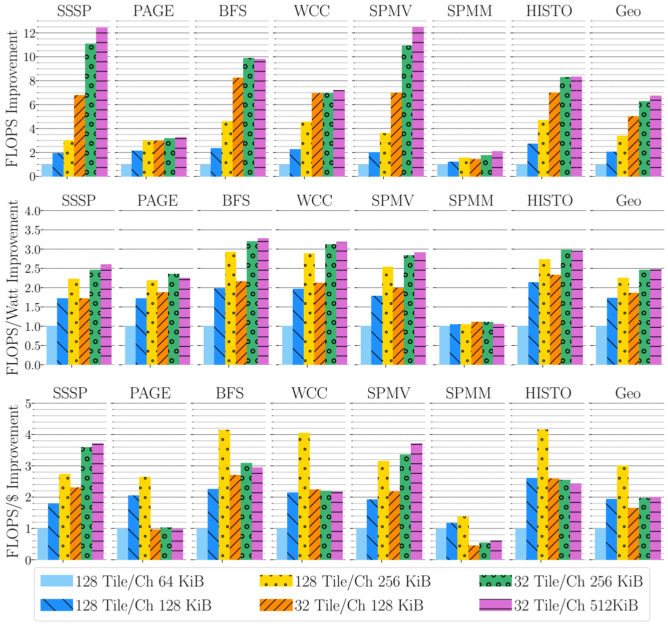

Fig. 5. Performance, energy efficiency, and performance per dollar improvements of the DUT using different SRAM sizes and number of tiles per HBM channel, over a baseline of 64 KiB SRAM and 128 tiles per HBM channel (Tile/Ch). In this study, a chiplet is always attached to a single 8-channel HBM device, and thus, the number of tiles per chiplet (16x16 or 32x32) determines the ratio of tiles per HBM channel. The RMAT-25 dataset is studied on a DUT with 1024 tiles; the dataset footprint per tile ranges from 4 MiB (Histogram) to 8 MiB per tile (SPMV).

The **throughput per dollar** is lower with 16x16-tile chiplets for all applications but SSSP and SPMV, due to the extra cost of having four times more HBM devices. The throughput per dollar estimates for SPMM—with over an order of magnitude more arithmetic intensity than the rest of the applications for the dataset evaluated (RMAT-25)—showcases that when targeting performance per dollar, the optimal memory integration depends on the application domain.

V. Conclusions

MuchiSim is a simulation framework aimed to explore the design space of scale-out architectures for communication-intensive applications like graph analytics and sparse linear algebra. It supports simulating a class of tiled manycore architectures with different hierarchical organizations, network topologies, and memory integrations. It also supports evaluating various parallelization strategies (do-all and task-based) and communication primitives (e.g., message-passing and reduction trees). As demonstrated via a case study, MuchiSim is well suited to explore the optimal balance between resources dedicated to compute, memory and network, for different application domains and target metrics (e.g., performance, power and cost).

The distinguishing factor of MuchiSim is its scalability in the challenging domain of data-dependent and communication-intensive applications, which requires functional simulation (since the execution depends on the data) as well as modeling the communication cycle by cycle. To demonstrate that, we simulated applications with billion-element datasets parallelized with up to a million processing units and presented the simulation times and throughput metrics. We achieve this scalability by optimizing the design and parallelization of the simulator for the target class of tiled manycore architectures.

The MuchiSim framework is open-source and available at our repository [68], including the simulator, applications, datasets, and visualization tools. It also contains the scripts to reproduce the experiments presented in this paper and other case studies, as a tutorial to allow researchers to explore the design space of scale-out manycore architectures. Future work on the simulator itself includes multi-node MPI parallelization and support for low-diameter cluster interconnects, to enable simulating even larger systems.

Acknowledgements

This material is based upon work supported in part by National Science Foundation (NSF) award No. 1763838, and based on research sponsored by the Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement FA8650-18-2-7862. In addition, Martonosi received separate NSF support while serving at NSF as an IPA rotator. [3]

References

[1] M. Abeydeera and D. Sanchez, "Chronos: Efficient speculative parallelism for accelerators," in *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020, pp. 1247–1262.

[2] D. Abts, J. Kim, G. Kimmell, M. Boyd, K. Kang, S. Parmar, A. Ling, A. Bitar, I. Ahmed, and J. Ross, "The Groq software-defined scale-out tensor streaming multiprocessor: From chips-to-systems architectural overview," in *IEEE Hot Chips 34 Symposium (HCS)*. IEEE Computer Society, 2022, pp. 1–69.

[3] N. R. Adiga, G. Almási, G. S. Almasi, Y. Aridor, R. Barik, D. Beece, R. Bellofatto, G. Bhanot, R. Bickford, M. Blumrich *et al.*, "An overview of the BlueGene/L supercomputer," in *SC'02: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*. IEEE, 2002, pp. 60–60.

[4] J. H. Ahn, S. Li, S. O, and N. P. Jouppi, "Mcsima+: A manycore simulator with application-level+ simulation and detailed microarchitecture modeling," in *Proceedings of the 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 74–85.

[5] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA)*, 2015, pp. 105–117.

[6] Ampere Computing, "AmpereOne 192-core server processor," https://amperecomputing.com/products/processors.

[7] S. Ardalan, B. Vinnikota, T. Arabi, and E. Alon, "What is the right die-to-die interface? a comparison study," 2022, https://www.opencompute.org/events/past-events/hipchips-chiplet-workshop-isca-conference.

[8] E. Argollo, A. Falcón, P. Faraboschi, M. Monchiero, and D. Ortega, "Cotson: Infrastructure for full system simulation," *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 1, p. 52–61, jan 2009. [Online]. Available: https://doi.org/10.1145/1496909.1496921

[9] T. Austin, E. Larson, and D. Ernst, "Simplescalar: an infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, 2002.

[10] A. Benner, "Optical interconnect opportunities in supercomputers and high end computing," in *OFC/NFOEC*. IEEE, 2012, pp. 1–60.

[3] The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF, AFRL, DARPA or the U.S. Government.

[11] D. Bertozzi and L. Benini, “Xpipes: a network-on-chip architecture for gigascale systems-on-chip,” *IEEE Circuits and Systems Magazine*, vol. 4, no. 2, pp. 18–31, 2004.

[12] M. Besta and T. Hoefler, “Slim fly: A cost effective low-diameter network topology,” in *SC’14: proceedings of the international conference for high performance computing, networking, storage and analysis*. IEEE, 2014, pp. 348–359.

[13] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, “The gem5 simulator,” *SIGARCH Comput. Archit. News*, vol. 39, no. 2, p. 1–7, aug 2011. [Online]. Available: https://doi.org/10.1145/2024716.2024718

[14] B. Black, “Die stacking is happening,” in *46th International Symposium on Microarchitecture (MICRO). Keynote*, 2013.

[15] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, “Cycle-accurate network on chip simulation with noxim,” *ACM Trans. Model. Comput. Simul.*, vol. 27, no. 1, aug 2016. [Online]. Available: https://doi.org/10.1145/2953878

[16] Cerebras Systems Inc., “The second generation wafer scale engine,” https://cerebras.net/wp-content/uploads/2021/04/Cerebras-CS-2-Whitepaper.pdf.

[17] T.-J. Chang, A. Li, F. Gao, T. Ta, G. Tziantzioulis, Y. Ou, M. Wang, J. Tu, K. Xu, P. J. Jackson, A. Ning, G. Chirkov, M. Orenes-Vera, S. Agwa, X. Yan, E. Tang, J. Balkind, C. Batten, and D. Wentzlaff, “CIFER: A 12nm, 16mm2, 22-core SoC with a 1541 LUT6/mm2 1.92 MOPS/LUT, fully synthesizable, cache-coherent, embedded FPGA,” in *2023 IEEE Custom Integrated Circuits Conference (CICC)*, 2023, pp. 1–2. [Online]. Available: https://doi.org/10.1109/CICC57935.2023.10121294

[18] J. Choquette and W. Gandhi, “NVIDIA A100 GPU: Performance & innovation for GPU computing,” in *IEEE Hot Chips 32 Symposium (HCS)*. IEEE Computer Society, 2020, pp. 1–43.

[19] V. Dadu, S. Liu, and T. Nowatzki, “Polygraph: Exposing the value of flexibility for graph processing accelerators,” in *Proceedings of the 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 595–608.

[20] V. Dadu and T. Nowatzki, “Taskstream: accelerating task-parallel workloads by recovering program structure,” in *Proceedings of the 27th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2022, pp. 1–13.

[21] S. Davidson, S. Xie, C. Torng, K. Al-Hawai, A. Rovinski, T. Ajayi, L. Vega, C. Zhao, R. Zhao, S. Dai, A. Amarnath, B. Veluri, P. Gao, A. Rao, G. Liu, R. K. Gupta, Z. Zhang, R. Dreslinski, C. Batten, and M. B. Taylor, “The Celerity open-source 511-core RISC-V tiered accelerator fabric: Fast architectures and design methodologies for fast chips,” *IEEE Micro*, vol. 38, no. 2, pp. 30–41, 2018.

[22] M. Emani, V. Vishwanath, C. Adams, M. E. Papka, R. Stevens, L. Florescu, S. Jairath, W. Liu, T. Nama, and A. Sujeeth, “Accelerating scientific applications with sambanova reconfigurable dataflow architecture,” *Computing in Science & Engineering*, vol. 23, no. 2, pp. 114–119, 2021.

[23] Esperanto Technologies, “Esperanto’s ET-Minion on-chip RISC-V cores,” https://www.esperanto.ai/technology/.

[24] A. Feldmann and D. Sanchez, “Spatula: A hardware accelerator for sparse matrix factorization,” in *Proceedings of the 56th Annual International Symposium on Microarchitecture (MICRO)*. ACM, 2023, p. 91–104. [Online]. Available: https://doi.org/10.1145/3613424.3623783

[25] K. Feng, Y. Ye, and J. Xu, “A formal study on topology and floorplan characteristics of mesh and torus-based optical networks-on-chip,” *Microprocessors and Microsystems*, vol. 37, no. 8, pp. 941–952, 2013.

[26] D. Fox, J. M. Diaz, and X. Li, “A gem5 implementation of the sequential codelet model: Reducing overhead and expanding the software memory interface,” in *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA*, 2023.

[27] Y. Fu and D. Wentzlaff, “PriME: A parallel and distributed simulator for thousand-core chips,” in *Proceedings of the 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE Press, March 2014.

[28] F. Gao, T.-J. Chang, A. Li, M. Orenes-Vera, D. Giri, P. J. Jackson, A. Ning, G. Tziantzioulis, J. Zuckerman, J. Tu, K. Xu, G. Chirkov, G. Tombesi, J. Balkind, M. Martonosi, L. Carloni, and D. Wentzlaff, “DECADES: A 67mm2, 1.46 TOPS, 55 giga cache-coherent 64-bit RISC-V instructions per second, heterogeneous manycore SoC with 109 tiles including accelerators, intelligent storage, and eFPGA in 12nm FinFET,” in *2023 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2023, pp. 1–2.

[29] S. Ghose, A. G. Yaglikçi, R. Gupta, D. Lee, K. Kudrolli, W. X. Liu, H. Hassan, K. K. Chang, N. Chatterjee, A. Agrawal, M. O’Connor, and O. Mutlu, “What your DRAM power models are not telling you: Lessons from a detailed experimental study,” *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 3, pp. 1–41, 2018.

[30] T. J. Ham, L. Wu, N. Sundaram, N. Satish, and M. Martonosi, “Graphicionado: A high-performance and energy-efficient accelerator for graph analytics,” in *Proceedings of the 49th Annual International Symposium on Microarchitecture (MICRO)*, 2016. [Online]. Available: https://doi.org/10.1109/MICRO.2016.7783759

[31] N. Hardavellas, S. Somogyi, T. F. Wenisch, R. E. Wunderlich, S. Chen, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk, “Simflex: A fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture,” *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 4, p. 31–34, mar 2004. [Online]. Available: https://doi.org/10.1145/1054907.1054914

[32] M. Horro, G. Rodríguez, and J. Touriño, “Simulating the network activity of modern manycores,” *IEEE Access*, vol. 7, pp. 81 195–81 210, 2019.

[33] X. Hu, D. Stow, and Y. Xie, “Die stacking is happening,” *IEEE micro*, vol. 38, no. 1, pp. 22–28, 2018.

[34] Isine, “Die yield calculator,” https://isine.com/resources/die-yield-calculator/.

[35] M. C. Jeffrey, S. Subramanian, C. Yan, J. Emer, and D. Sanchez, “A scalable architecture for ordered parallelism,” in *Proceedings of the 48th Annual International Symposium on Microarchitecture (MICRO)*. ACM, 2015, p. 228–241. [Online]. Available: https://doi.org/10.1145/2830772.2830777

[36] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, “A detailed and flexible cycle-accurate network-on-chip simulator,” in *Proceedings of the 2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 86–96.

[37] S. W. Jones, “Lithovision: Economics in the 3d era,” https://semiwiki.com/wp-content/uploads/2020/03/Lithovision-2020.pdf.

[38] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 1–12.

[39] D. C. Jung, S. Davidson, C. Zhao, D. Richmond, and M. B. Taylor, “Ruche networks: Wire-maximal, no-fuss nocs: Special session paper,” in *2020 14th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. IEEE, 2020, pp. 1–8.

[40] D.-H. Kim, B. Song, H.-a. Ahn, W. Ko, S. Do, S. Cho, K. Kim, S.-H. Oh, H.-Y. Joo, G. Park, J.-H. Jang, Y.-H. Kim, D. Lee, J. Jung, Y. Kwon, Y. Kim, J. Jung, S. O, S. Lee, J. Lim, J. Son, J. Min, H. Do, J. Yoon, I. Hwang, J. Park, H. Shim, S. Yoon, D. Choi, J. Lee, S. Woo, E. Hong, J. Choi, J.-S. Kim, S. Han, J. Bang, B. Park, J. Kim, S.-K. Choi, G.-H. Han, Y.-C. Sung, W.-I. Bae, J.-D. Lim, S. Lee, C. Yoo, S. J. Hwang, and J. Lee, “A 16Gb 9.5Gb/S/pin LPDDR5X SDRAM with pow-power schemes exploiting DVFS and offset-calibrated readout sense amplifiers in a fourth generation 10nm DRAM process,” in *2022 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 448–450.

[41] J. Kim, W. J. Dally, S. Scott, and D. Abts, “Technology-driven, highly-scalable dragonfly topology,” *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3, pp. 77–88, 2008.

[42] S. Kim, S. Kim, K. Cho, T. Shin, H. Park, D. Lho, S. Park, K. Son, G. Park, and J. Kim, “Processing-in-memory in high bandwidth memory (pim-hbm) architecture with energy-efficient and low latency channels for high bandwidth system,” in *2019 IEEE 28th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*, 2019, pp. 1–3.

[43] S. Knowles, “Graphcore,” in *IEEE Hot Chips 33 Symposium (HCS)*. IEEE, 2021, pp. 1–25.

[44] P. Lawrence, B. Sergey, R. Motwani, and T. Winograd, “The pagerank citation ranking: Bringing order to the web,” Stanford University, Technical Report, 1998.

[45] C.-C. Lee, C. Hung, C. Cheung, P.-F. Yang, C.-L. Kao, D.-L. Chen, M.-K. Shih, C.-L. C. Chien, Y.-H. Hsiao, L.-C. Chen, M. Su, M. Alfano, J. Siegel, J. Din, and B. Black, "An overview of the development of a GPU with integrated HBM on silicon interposer," in *2016 IEEE 66th Electronic Components and Technology Conference (ECTC)*, 2016, pp. 1439–1444.

[46] D. U. Lee, H. S. Cho, J. Kim, Y. J. Ku, S. Oh, C. D. Kim, H. W. Kim, W. Y. Lee, T. K. Kim, T. S. Yun *et al.*, "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 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2020, pp. 334–336.

[47] C. E. Leiserson, "Fat-trees: Universal networks for hardware-efficient supercomputing," *IEEE transactions on Computers*, vol. 100, no. 10, pp. 892–901, 1985.

[48] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *Journal of Machine Learning Reseach (JMLR)*, vol. 11, pp. 985–1042, Mar. 2010.

[49] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.

[50] A. Li, T.-J. Chang, F. Gao, T. Ta, G. Tziantzioulis, Y. Ou, M. Wang, J. Tu, K. Xu, P. Jackson, A. Ning, G. Chirkov, M. Orenes-Vera, S. Agwa, X. Yan, E. Tang, J. Balkind, C. Batten, and D. Wentzlaff, "CIFER: A cache-coherent 12nm 16mm2 SoC with four 64-bit RISC-V application cores, 18 32-bit RISC-V compute cores, and a 1541 LUT6/mm2 synthesizable eFPGA," *IEEE Solid-State Circuits Letters*, pp. 1–1, 2023.

[51] S. Lie, "Multi-million core, multi-wafer AI cluster," in *IEEE Hot Chips 33 Symposium (HCS)*. IEEE Computer Society, 2021, pp. 1–41.

[52] Y. J. Lo, S. Williams, B. Van Straalen, T. J. Ligocki, M. J. Cordery, N. J. Wright, M. W. Hall, and L. Oliker, "Roofline model toolkit: A practical tool for architectural and program analysis," in *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation*, S. A. Jarvis, S. A. Wright, and S. D. Hammond, Eds. Cham: Springer International Publishing, 2015, pp. 129–148.

[53] A. Manocha, T. Sorensen, E. Tureci, O. Matthews, J. L. Aragón, and M. Martonosi, "Graphattack: Optimizing data supply for graph applications on in-order multicore architectures," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 18, no. 4, pp. 1–26, 2021.

[54] O. Matthews, A. Manocha, D. Giri, M. Orenes-Vera, E. Tureci, T. Sorensen, T. J. Ham, J. L. Aragon, L. P. Carloni, and M. Martonosi, "MosaicSim: A lightweight, modular simulator for heterogeneous systems," in *Proceedings of the 2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2020, pp. 136–148.

[55] Micron, "High Bandwidth Memory with ECC," 2018, https://media-www.micron.com/-/media/client/global/documents/products/data-sheet/dram/hbm2e/8gb_and_16gb_hbm2e_dram.pdf.

[56] J. E. Miller, H. Kasture, G. Kurian, C. Gruenwald, N. Beckmann, C. Celio, J. Eastep, and A. Agarwal, "Graphite: A distributed parallel simulator for multicores," in *HPCA*. IEEE Press, 2010.

[57] F. Muñoz-Martínez, R. Garg, M. Pellauer, J. L. Abellán, M. E. Acacio, and T. Krishna, "Flexagon: A multi-dataflow sparse-sparse matrix multiplication accelerator for efficient dnn processing," in *Proceedings of the 28th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 3*, 2023, pp. 252–265.

[58] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, "Introducing the Graph 500," http://www.graph500.org/specifications, Cray User's Group (CUG), 2010.

[59] S. Naffziger, N. Beck, T. Burd, K. Lepak, G. H. Loh, M. Subramony, and S. White, "Pioneering chiplet technology and design for the amd epyc™ and ryzen™ processor families," in *Proceedings of the 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE Press, 2021, p. 57–70.

[60] N. Nassif, A. O. Munch, C. L. Molnar, G. Pasdast, S. V. Lyer, Z. Yang, O. Mendoza, M. Huddart, S. Venkataraman, S. Kandula *et al.*, "Sapphire Rapids: The next-generation Intel Xeon scalable processor," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 44–46.

[61] Q. M. Nguyen and D. Sanchez, "Pipette: Improving core utilization on irregular applications through intra-core pipeline parallelism," in *Proceedings of the 53rd Annual International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 596–608.

[62] Q. M. Nguyen and D. Sanchez, "Fifer: Practical acceleration of irregular applications on reconfigurable architectures," in *Proceedings of the 54th Annual International Symposium on Microarchitecture (MICRO)*, ser. MICRO '21. ACM, 2021, p. 1064–1077. [Online]. Available: https://doi.org/10.1145/3466752.3480048

[63] C.-S. Oh, K. C. Chun, Y.-Y. Byun, Y.-K. Kim, S.-Y. Kim, Y. Ryu, J. Park, S. Kim, S. Cha, D. Shin *et al.*, "22.1 A 1.1 V 16GB 640GB/s HBM2E DRAM with a data-bus window-extension technique and a synergetic on-die ECC scheme," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2020, pp. 330–332.

[64] Open Compute Group, "Bunch of wires PHY specification," https://opencomputeproject.github.io/ODSA-BoW/bow_specification.html.

[65] M. Orenes-Vera, A. Manocha, J. Balkind, F. Gao, J. L. Aragón, D. Wentzlaff, and M. Martonosi, "Tiny but mighty: designing and realizing scalable latency tolerance for manycore SoCs." in *Proceedings of the 48th Annual International Symposium on Computer Architecture (ISCA)*, 2022, pp. 817–830.

[66] M. Orenes-Vera, I. Sharapov, R. Schreiber, M. Jacquelin, P. Vandermersch, and S. Chetlur, "Wafer-scale fast fourier transforms," in *Proceedings of the 37th International Conference on Supercomputing (ICS)*. ACM, 2023, p. 180–191. [Online]. Available: https://doi.org/10.1145/3577193.3593708

[67] M. Orenes-Vera, E. Tureci, M. Martonosi, and D. Wentzlaff, "DCRA: A distributed chiplet-based reconfigurable architecture for irregular applications," 2023, https://doi.org/10.48550/arXiv.2311.15443.

[68] M. Orenes-Vera, E. Tureci, M. Martonosi, and D. Wentzlaff, "MuchiSim simulation framework and artifacts," 2023, https://github.com/PrincetonUniversity/muchisim.git.

[69] M. Orenes-Vera, E. Tureci, D. Wentzlaf, and M. Martonosi, "Massive data-centric parallelism in the chiplet era," 2023, https://doi.org/10.48550/arXiv.2304.09389.

[70] M. Orenes-Vera, E. Tureci, D. Wentzlaff, and M. Martonosi, "Dalorex: A data-local program execution and architecture for memory-bound applications," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 718–730.

[71] M. Orenes-Vera, E. Tureci, D. Wentzlaff, and M. Martonosi, "Tascade: Hardware support for atomic-free, asynchronous and efficient reduction trees," 2023, https://doi.org/10.48550/arxiv.2311.15810.

[72] Y. Ou, S. Agwa, and C. Batten, "Implementing low-diameter on-chip networks for manycore processors using a tiled physical design methodology," in *2020 14th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. IEEE, 2020, pp. 1–8.

[73] M. M. Ozdal, S. Yesil, T. Kim, A. Ayupov, J. Greth, S. Burns, and O. Ozturk, "Energy efficient architecture for graph analytics accelerators," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 166–177, 2016.

[74] M. O'Connor, N. Chatterjee, D. Lee, J. Wilson, A. Agrawal, S. W. Keckler, and W. J. Dally, "Fine-grained dram: Energy-efficient dram for extreme bandwidth systems," in *Proceedings of the 50th Annual International Symposium on Microarchitecture (MICRO)*. IEEE, 2017, pp. 41–54.

[75] M.-J. Park, H. S. Cho, T.-S. Yun, S. Byeon, Y. J. Koo, S. Yoon, D. U. Lee, S. Choi, J. Park, J. Lee *et al.*, "A 192-gb 12-high 896-gb/s hbm3 dram with a tsv auto-calibration scheme and machine-learning-based layout optimization," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 444–446.

[76] A. Patrizio, "High-bandwidth memory (hbm) delivers impressive performance gains," https://semiengineering.com/whats-next-for-high-bandwidth-memory/.

[77] G. Posluns, Y. Zhu, G. Zhang, and M. C. Jeffrey, "A scalable architecture for reprioritizing ordered parallelism," in *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*. ACM, 2022, p. 437–453. [Online]. Available: https://doi.org/10.1145/3470496.3527387

[78] V. Puente, J. Gregorio, and R. Beivide, "Sicosys: an integrated framework for studying interconnection network performance in multiprocessor systems," in *Proceedings 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, 2002, pp. 15–22.

[79] S. Rahman, N. Abu-Ghazaleh, and R. Gupta, "Graphpulse: An event-driven hardware accelerator for asynchronous graph processing," in *Proceedings of the 53rd Annual International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 908–921.

[80] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "The structural simulation toolkit," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 4, p. 37–42, mar 2011. [Online]. Available: https://doi.org/10.1145/1964218.1964225

[81] D. Sanchez and C. Kozyrakis, "Zsim: Fast and accurate microarchitectural simulation of thousand-core systems," *ACM SIGARCH Computer architecture news*, vol. 41, no. 3, pp. 475–486, 2013.

[82] D. Schor, "TSMC demonstrates a 7nm ARM-based chiplet design for HPC," 2019, https://fuse.wikichip.org/news/2446/tsmc-demonstrates-a-7nm-arm-based-chiplet-design-for-hpc/.

[83] D. Schor, "TSMC Details 5 nm," 2020, https://fuse.wikichip.org/news/3398/tsmc-details-5-nm/.

[84] D. D. Sharma, "PCI express 6.0 specification: A low-latency, high-bandwidth, high-reliability, and cost-effective interconnect with 64.0 gt/s pam-4 signaling," *IEEE Micro*, vol. 41, no. 1, pp. 23–29, 2020.

[85] K. Shkurko, T. Grant, E. Brunvand, D. Kopta, J. Spjut, E. Vasiou, I. Mallett, and C. Yuksel, "Simtrax: Simulation infrastructure for exploring thousands of cores," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, 2018, pp. 503–506.

[86] G. M. Slota, S. Rajamanickam, and K. Madduri, "BFS and coloring-based parallel algorithms for strongly connected components and related problems," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium, Phoenix, AZ, USA, May 19-23, 2014.* IEEE Computer Society, 2014, pp. 550–559. [Online]. Available: https://doi.org/10.1109/IPDPS.2014.64

[87] K. Sohn, W.-J. Yun, R. Oh, C.-S. Oh, S.-Y. Seo, M.-S. Park, D.-H. Shin, W.-C. Jung, S.-H. Shin, J.-M. Ryu, H.-S. Yu, J.-H. Jung, H. Lee, S.-Y. Kang, Y.-S. Sohn, J.-H. Choi, Y.-C. Bae, S.-J. Jang, and G. Jin, "A 1.2 v 20 nm 307 gb/s hbm dram with at-speed wafer-level io test scheme and adaptive refresh considering temperature distribution," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 250–260, 2017.

[88] T. Sorensen, A. Manocha, E. Tureci, M. Orenes-Vera, J. L. Aragón, and M. Martonosi, "A simulator and compiler framework for agile hardware-software co-design evaluation and exploration," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.

[89] D. Stow, Y. Xie, T. Siddiqua, and G. H. Loh, "Cost-effective design of scalable high-performance systems using active and passive interposers," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 728–735.

[90] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W.-m. Hwu, "Parboil: A revised benchmark suite for scientific and commercial throughput computing," University of Illinois at Urbana-Champaign, Tech. Rep. IMPACT-12-01, 2012.

[91] N. Talati, K. May, A. Behroozi, Y. Yang, K. Kaszyk, C. Vasiladiotis, T. Verma, L. Li, B. Nguyen, J. Sun, J. M. Morton, A. Ahmadi, T. Austin, M. O'Boyle, S. Mahlke, T. Mudge, and R. Dreslinski, "Prodigy: Improving the memory latency of data-indirect irregular workloads using hardware-software co-design," in *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA).* IEEE, 2021, pp. 654–667.

[92] E. Talpes, D. Williams, and D. D. Sarma, "Dojo: The microarchitecture of tesla exa-scale computer," in *IEEE Hot Chips 34 Symposium (HCS).* IEEE Computer Society, 2022, pp. 1–28.

[96] Z. Wang, C. Liu, N. Beckmann, and T. Nowatzki, "Affinity alloc: Taming not-so near-data computing," in *Proceedings of*

[93] Z. Tan, A. Waterman, R. Avizienis, Y. Lee, H. Cook, D. Patterson, and K. Asanović, "Ramp gold: An fpga-based architecture simulator for multiprocessors," in *Proceedings of the 47th Design Automation Conference (DAC).* ACM, 2010, p. 463–468. [Online]. Available: https://doi.org/10.1145/1837274.1837390

[94] T. Tang and Y. Xie, "Cost-aware exploration for chiplet-based architecture with advanced packaging technologies," *arXiv preprint arXiv:2206.07308*, 2022.

[95] J. Vasiljevic, L. Bajic, D. Capalija, S. Sokorac, D. Ignjatovic, L. Bajic, M. Trajkovic, I. Hamer, I. Matosevic, A. Cejkov, U. Aydonat, T. Zhou, S. Z. Gilani, A. Paiva, J. Chu, D. Maksimovic, S. A. Chin, Z. Moudallal, A. Rakhmati, S. Nijjar, A. Bhullar, B. Drazic, C. Lee, J. Sun, K.-M. Kwong, J. Connolly, M. Dooley, H. Farooq, J. Y. T. Chen, M. Walker, K. Dabiri, K. Mabee, R. S. Lal, N. Rajatheva, R. Retnamma, S. Karodi, D. Rosen, E. Munoz, A. Lewycky, A. Knezevic, R. Kim, A. Rui, A. Drouillard, and D. Thompson, "Compute substrate for software 2.0," *IEEE Micro*, vol. 41, no. 2, pp. 50–55, 2021.
the 56th Annual International Symposium on Microarchitecture (MICRO). ACM, 2023, p. 784–799. [Online]. Available: https://doi.org/10.1145/3613424.3623778

[97] T. Wei, N. Turtayeva, M. Orenes-Vera, O. Lonkar, and J. Balkind, "Cohort: software-oriented acceleration for heterogeneous SoCs," in *Proceedings of the 28th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).* ACM, 2023, p. 105–117. [Online]. Available: https://doi.org/10.1145/3582016.3582059

[98] J. Wilson, "High-bandwidth density, energy-efficient, short-reach signaling that enables massively scalable parallelism," 2022, https://www.opencompute.org/events/past-events/hipchips-chiplet-workshop-isca-conference.

[99] Y. Yokoyama, M. Tanaka, K. Tanaka, M. Morimoto, M. Yabuuchi, Y. Ishii, and S. Tanaka, "A 29.2 mb/mm2 ultra high density SRAM macro using 7nm FinFET technology with dual-edge driven wordline/bitline and write/read-assist circuit," in *Proceedings of the IEEE Symposium on VLSI Circuits*, 2020, pp. 1–2.

[100] J. Zarrin, R. L. Aguiar, and J. P. Barraca, "Manycore simulation for peta-scale system design: Motivation, tools, challenges and prospects," *Simulation Modelling Practice and Theory*, vol. 72, pp. 168–201, Mar. 2017. [Online]. Available: https://doi.org/10.1016/j.simpat.2016.12.014

[101] F. Zaruba and L. Benini, "The cost of application-class processing: Energy and performance analysis of a linux-ready 1.7-GHz 64-bit RISC-V core in 22-nm FDSOI technology," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 11, pp. 2629–2640, Nov 2019, https://github.com/openhwgroup/cva6.

[102] F. Zaruba, F. Schuiki, and L. Benini, "Manticore: A 4096-core RISC-V chiplet architecture for ultraefficient floating-point computing," *IEEE Micro*, vol. 41, no. 2, pp. 36–42, 2020.

[103] G. Zheng, G. Kakulapati, and L. Kale, "Bigsim: a parallel simulator for performance prediction of extremely large parallel machines," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, 2004, pp. 78–.

[104] Y. Zhuo, C. Wang, M. Zhang, R. Wang, D. Niu, Y. Wang, and X. Qian, "GraphQ: Scalable PIM-based graph processing," in *Proceedings of the 52nd Annual International Symposium on Microarchitecture (MICRO)*, 2019, pp. 712–725.