At the Locus of Performance: A Case Study in Enhancing CPUs with Copious 3D-Stacked Cache

Jens Domke*,§, Emil Vatai*,§, Balazs Geroﬁ†, Yuetsu Kodama*, Mohamed Wahib*,†, Artur Podobas‡, Sparsh Mittal**, Miquel Pericàs¶, Lingqi Zhang††, Peng Chen†, Aleksandr Drozd*, and Satoshi Matsuoka*,**††

* RIKEN Center for Computational Science, Japan
† National Institute of Advanced Industrial Science and Technology, Japan
‡ KTH Royal Institute of Technology, Sweden
** Chalmers Institute Of Technology, Gothenburg, Sweden
¶ Indian Institute Of Technology, Roorkee, India
†† Tokyo Institute of Technology, Japan

Abstract—Over the last three decades, innovations in the memory subsystem were primarily targeted at overcoming the data movement bottleneck. In this paper, we focus on a specific market trend in memory technology: 3D-stacked memory and caches. We investigate the impact of extending the on-chip memory capabilities in future HPC-focused processors, particularly by 3D-stacked SRAM. First, we propose a method oblivious to the memory subsystem to gauge the upper-bound in performance improvements when data movement costs are eliminated. Then, using the gem5 simulator, we model two variants of LARC, a processor fabricated in 1.5 nm and enriched with high-capacity 3D-stacked cache. With a volume of experiments involving a board set of proxy-applications and benchmarks, we aim to reveal where HPC CPU performance could be circa 2028, and conclude this immediate remedy to the end of Dennard’s scaling applies to this day in the form of processors such as Fujitsu A64FX [5], AMD Ryzen [6], or NVIDIA GPUs [7], [8].

I. INTRODUCTION

Historically, the reliable performance increase of von Neumann-based general-purpose processors (CPUs) was driven by two technological trends. The first, observed by Gordon E. Moore [1], is that the number of transistors in an integrated circuit doubles roughly every two years. The second, called Dennard’s scaling [2], postulates that as transistors get smaller their power density stays constant. These trends synergized well, allowing computer architectures to continuously improve performance through, for example, aggressive pipelining and superscalar techniques without running into thermal limitations by, e.g., reducing the operating voltage. In the early 2000s, Dennard’s scaling ended [3] and forced architects to shift their attention from improving instruction-level parallelism to exploiting on-chip multiple-instruction multiple-data parallelism [4]. This immediate remedy to the end of Dennard’s scaling applies to this day in the form of processors such as Fujitsu A64FX [5], AMD Ryzen [6], or NVIDIA GPUs [7], [8].

Unfortunately, Moore’s law is impending termination [9], and we are entering a post-Moore era [10], home to a diversity of architectures, such as quantum-, neuromorphic-, or reconfigurable computing [11]. Many of these prototypes hold promise but are still immature, focus on a niche use case, or incur long development cycles. At the same time, there is one salient post-Moore architecture that is growing in maturity and which can facilitate performance improvements in the decades to come even for the classic von Neumann CPUs we have come to rely upon—3D integrated circuit (IC) stacking [12].

3D integrated circuits refer to the general technologies of vertically building integrated circuits and can be done in multiple ways, such as by stacking multiple discrete dies and connecting them using coarse through-silicon vias (TSVs) or growing the 3D integrated circuit monolithically on the wafer [13]. 3D integrated circuits have multiple benefits [14], including (i) shorter wire lengths in the interconnect leading to reduction in power consumption, (ii) improved memory bandwidth through on-chip integration that can alleviate performance bottlenecks in memory-bound applications, (iii) higher package density, which yields more compute and smaller system footprint, and (iv) possibly lower fabrication cost due to smaller die size (thus improved yield). All these are very desirable benefits in today’s exascale (and future) High-Performance Computing (HPC) systems. Recent advances in
3D integrated circuits have enabled many times higher capacity on-chip memory (caches) than traditional systems (e.g., AMD V-Cache [15]) in the hope that the larger cache sizes will materialize—as conventional wisdom and maxims tell us—in higher performance. But how far can 3D ICs (with a focus on increased on-chip cache) take us in HPC?

Intuition tells us that an increased cache size will help alleviate the performance bottlenecks of many key HPC applications. To demonstrate this, we conduct a pilot study where we execute one of the important proxy-apps from the DoE Exascale Computing Project (ECP) suite, MiniFE [16] (see Section III-C for details), on AMD EPYC Milan and Milan-X CPUs—two architecturally similar processors with vastly different L3 cache sizes [17]. Figure 1 overviews our result, and we see that for a subset of problem sizes, the 3-times larger L3 capacity of Milan-X yields up-to 3.4x improvements over baseline Milan for this memory-bound application, which motivates us to further research 3D-stacked caches.

**Contributions:** In this study, we seek to unveil the impact that future large caches could have on the next generation of HPC systems. We study our research questions from three different levels of abstraction: (i) we design a novel exploration framework that allows us to simulate HPC applications running on a hypothetical processor having infinitely large L1D cache. We use this framework, that is orders of magnitude faster than with eight stacked SRAM dies under 1.5 nm manufacturing where we execute one of the important proxy-apps from the processor (LARC’s Core Memory Group (CMG) compared to our A64FX CMG baseline at one-fourth of the area. For applications that are responsive to larger cache capacity, this would translate to an average improvement of 9.77x (geometric mean) when we assume ideal scaling and compare at the full chip level.

**II. CPUs Empowered with High-Capacity Cache: the Future of HPC?**

The memory bandwidth of modern systems has been the bottleneck (the “memory wall” [18]) ever since CPU performance started to outgrow the bandwidth of memory subsystems in the early 1990s [19]. Today, this trend continues to shape the performance optimization landscape in high-performance computing [20], [21]. Diverse memory technologies are emerging to overcome said data movement bottleneck, such as Processing-in-Memory (PIM) [22], 3D-stackable High-Bandwidth Memory (HBM) [23], deeper (and more complex) memory hierarchies [24], and—the topic of the present paper—novel 3D-stacked caches [12], [25], [26].

In this study, our aspiration is to gauge the far end of processor technology and how we foresee it to evolve in six to eight years from now, circa 2028, when processors using 1.5 nm technology are expected to be available [27]. More specifically, as 3D-stacked SRAM memory [28] becomes more common. What are the performance implications for common HPC workloads, and what new challenges lie ahead for the community? However, before attempting to understand what performance may look like six years from now, we must describe how the processor itself will change. In this section, we introduce, motivate, and reason about our design choices of what we envision as a future processor that capitalizes on large capacity 3D-stacked cache, briefly called LARC (LARge Cache processor). Before looking at LARC, we must first set and analyze a baseline processor.

**A. LARC’ Baseline: The A64FX Processor**

We choose to base our future processor design on the A64FX processor [29]. The Arm-based A64FX processor from Fujitsu is currently powering Supercomputer Fugaku [5], which, as of April 2022, leads the TOP500 [30] (for HPL and HPCG; cf. Section II-C) and Graph500 performance charts. The A64FX is a many-core processor build on 7 nm technology, developed jointly by Fujitsu and the RIKEN research institute. It has a total of 52 ARM-capable cores (including Scalable Vector Extensions [31]) distributed across four internal compute clusters, called Core Memory Groups (CMGs). Each CMG hosts 13 cores, where 12 are available to the user, and one core is exclusively used for management. Each core has a local 64 KiB instruction and data-cache, and is capable of delivering 70.4 Gflop/s (IEEE-754 double-precision) performance. Each CMG also contains one of the four slices of the processor’s 32 MiB L2 cache, delivering over 900 GB/s bandwidth to the CMG [29]. The L2 cache, which is the CPU’s last level cache (LLC), is kept coherent through a ring interconnect that connects the four CMGs. Inside the CMG, a crossbar switch is used to connect the cores and the L2 slice. The cache line-size is 256 bytes, and the bus-width between the L1 and L2 cache is set to be 128 bytes (read) and 64 bytes (write). The L2 cache has 16-way set associativity, and the total performance per CMG is 845 Gflop/s (user cores) or 3.4 TFlop/s for the entire chip.

We emphasize that our aim is not to propose a successor of A64FX, nor are we particularly restricting our vision by the design constrains of A64FX (e.g., power budget). However, we build our design on A64FX because: (i) as mentioned above, A64FX represents the pinnacle of CPU performance in commercially available CPUs, so it is natural to use it as a starting point. (ii) A64FX is the only commercially-available CPU, currently in continued production, with HBM. The expected bandwidth ratio between future HBM and future 3D-stacked caches is similar to the ratio between traditional DRAM and LLC bandwidths [32], which is what applications and performance models are accustomed to. (iii) The A64FX LLC cache design (particularly the L2 slices connected by a crossbar switch) happens to be convenient and thus, requires a minimal effort to extend the design in a simulated environment.

In conclusion, while we extend the A64FX architecture, our workflow itself can be generalized to cover any of the
processors supported by CPU simulators (e.g., variants of gem5 [33] can simulate other architectures, including x86).

B. Floorplan Analysis for Fujitsu A64FX

In order to estimate the floorplan of the future LARC processor built on 1.5 nm technology, we first need the floorplan of the current A64FX processor built at 7 nm; but detailed floorplan information is very sparse and often not readily available. Fortunately, we do know that the die size of A64FX is ≈400 mm² [5]. Given this knowledge, and the openly-available die shots with processor core segments highlighted [34], we can estimate most of the A64FX floorplan, including the size of CMGs and processor cores. We performed this exercise, and the result is shown in Figure 2. Overall, each CMG is ≈48 mm² in area, where a A64FX core occupies ≈2.25 mm² area. The remaining parts of the CMG consist of the L2 cache slice and controller and the interconnect for intra-CMG communication.

C. From A64FX’s CMG Layout to LARC’s CMG Layout

Knowing the floorplan, we now proceed to describe how we envision the CMG design with 1.5 nm technology. We scale the CMG by moving three generations, from 7 nm to 1.5 nm, and reduce the silicon footprint by around 8x (∼2x per generation) for the entire CMG [35]. The new CMG consumes as little as 6 mm² of silicon area. Next, we reclaim the area of the CMG currently occupied by the L2 cache and controller and replace it with three additional CPU cores, yielding a total of 16 per CMG. Further, inline with the projected year 2019→2028 growth in the number of cores [36], we double the core count of the CMG to 32, which leads to it occupying ≈12 mm² of silicon area. We pessimistically leave the area for the interconnect the same and continue to use it as the primary means for communication with the CMG. We call this new variant as LARC’s CMG. Finally, we conservatively assume the same die size, and therefore, LARC would have 16 CMGs, each with 32 cores, in comparison to A64FX’s 4 CMG with 12+1 cores each. In LARC, we ignore the management core. However, our performance analysis will remain on the CMG level, instead of full chip, due to limitations which we detail in Section III-B.

D. LARC’s Vertically Stacked Cache

In the above design, we removed the L2 cache and controller from the CMG of LARC. We now assume that the L2 cache can be directly placed vertically on the CMG through 3D stacking [25]. We build our estimations based on experiments from Shiba et al. [26], who demonstrate the feasibility of stacking up-to-eight SRAM dies on top of a processor using a ThruChip Interface (TCI). The capacity and bandwidth of stacked memory is a function of several parameters: the number of channels available (N_Ch), the per-channel capacity (N_Cap in KiB), their width (W in bytes), the number of stacked dies (N_dies), and the operating frequency (f_clk in GHz). Shiba et al. [26] estimated that at a 10 nm process technology, eight stacks would provide ≈512 MiB of aggregated SRAM capacity for a footprint of ≈121 mm². In their design, each stack has 128 channels of 512 KiB capacity. In our work, we conservatively assume an 8x scaling from 10 nm to 1.5 nm, and thus, at 12 mm² area (the size of one LARC CMG), N_Ch on each die would be ≈102 (=128*8/10). Keeping the area and total capacity fixed, we reduce per-channel capacity by 4x and increase the number of channels by 4x, thus, N_Ch = 408. We approximate it a nearby sum of power-of-two number, viz., N_Ch = 384. Thus, with eight stacked dies (N_dies = 8), our 3D SRAM cache has a total storage capacity of N_dies·N_Ch·N_Cap = 384 MiB per CMG. We estimate the bandwidth in a similar way. We know from previous studies [26] that 3D-stacked SRAM, built on 40 nm technology, can operate at 300 MHz. We conservatively expect the same SRAM to operate at (f_clk=1 GHz when moving from 40 nm→1.5 nm. Each channel’s width is given in [26] and amounts to (W=)4 byte (32-bit). With this, the CMG bandwidth becomes: N_Ch·f_clk·W = 1536 GB/s. The read- and write-latency of this SRAM cache is 3 cycles [26].

A challenge in the use of large caches is their large tag size and high bandwidth consumption. Several solutions to this issue have been proposed in the context of stacked-DRAM caches [23]. For our cache design, we assume a 256 B cache block design, which avoids bandwidth bloat. Each tag takes 6 B and as such, the total tag array size for each CMG becomes 9 MiB. This tag array can be easily placed in the cache itself. We assume that tag and data accesses happen sequentially. The tags and data of a cache set are stored on a single die. Hence, on every access, only one die needs to be activated. Since this takes only few cycles, the overall miss penalty remains small and comparable to that of A64FX LLC.

To show that our cache projections are realistic, we compare it with AMD’s 3D V-cache design. It uses a single stacked die providing 64 MiB capacity (in addition to the 32 MiB cache in the base die) at 7 nm [15], [37], and only 3 to 4 cycles of extra latency [38]. It has 36 mm² area and has a bandwidth of 2 TB/s. When stacking additional dies on top, and assuming...
an 8x scaling of the area by going from 7 nm to 1.5 nm, we speculate that the LLC capacity of this commercial processor could easily exceed that of our proposed LARC.

E. LARC’s Core Memory Group (CMG)

At last, we detail our experimental CMG built on a hypothetical 1.5 nm technology: the LARC CMG. An illustration of this system is shown in Figure 2. Each CMG consists of 32 A64FX-like cores, which keeps the L1 instruction- and data-cache to 64 KiB each, yielding a per CMG performance of \( \approx 2.3 \text{Tflop/s} \) (IEEE-754 double-precision). A 384 MiB L2 cache is stacked vertically on the top of the CMG through eight SRAM layers. We also keep the bandwidth of external HBM memory to its current A64FX value of 256 GB/s to be able to quantify performance improvements from the proposed large capacity 3D cache in isolation from any improvements that would come from increased HBM bandwidth.

To summarize, a complete, hypothetical LARC chip would contain 512 processing cores, 6 GiB of stacked L2 cache, a peak L2 bandwidth of 24.6 TB/s, a peak HBM bandwidth of 4.1 TB/s, and 36 Tbit/s of raw, double-precision, compute.

III. PROJECTING PERFORMANCE IMPROVEMENT IN SIMULATED ENVIRONMENTS

Analyzing the feasibility of a future CPU architecture is only one side of the coin, and hence we have to demonstrate the effects of the proposed changes on real applications to allow a meaningful cost-benefit analysis by CPU vendors. This section focuses on two simulation approaches (one novel; one established) and discusses the HPC applications, which we evaluate later extensively in Section IV.

A. Machine Code Analyzers Assuming Unrestricted Locality

Before launching multi-months executions of cycle-accurate simulations, one would want to have a first-order approximation of a very large and fast cache. We design a simulation approach, using Machine Code Analyzers (MCA), which can estimate the speedup for a given application orders-of-magnitude faster than gem5 (cf. next section). This upper bound in expected performance improvement allows us to: (i) get a perspective on the best possible performance improvement if all read/writes can be satisfied from the cache; and (ii) justify more accurate simulations and classify their results with respect to the baseline and the upper bound.

Machine Code Analyzers, such as llvm-mca \[39\], have been designed to study microarchitectures, improve compilers, and investigate resource pressure for application kernels. Usually, the input for these tools is a short Assembly sequence and they output, among other things, an expected throughput for a given CPU when the sequence is executed many times and all data is available in L1 data cache. For most real applications the latter assumption is obviously incorrect, however it is ideal to gauge an upper bound when all memory-bottlenecks disappear.

Unfortunately, it is neither feasible to record all executed instructions in one long sequence, nor to analyze a full program sequence with llvm-mca. Hence, we break the program execution into basic blocks (at most tens or hundreds of instructions) and evaluate their throughput individually. For a given combination of a program and input (called workload hereafter), the basic blocks and their dependencies create a directed Control Flow Graph (CFG) \[40\] with one source (program start) and one sink (program termination). All intermediate nodes (representing basic blocks) of the graph can have multiple parent- and dependent-nodes, as well as self-references (e.g. basic blocks of \( f_{oz}\)-loops). Knowing the “runtime” of each basic block and the number of invocations per basic block, we can estimate the runtime of the entire workload by summation of the parts.

We utilize Intel’s Software Development Emulator (SDE) \[41\] to record the basic blocks and their caller/callee dependencies for a workload with modest runtime overhead (typically in order of 1000x slowdown). SDE also notes down the number of invocations per CFG edge for a workload, i.e., how often the program counter (PC) jumped from one specific basic block to another specific block. We developed a program which parses the output of Intel SDE and establishes an internal representation of the Control Flow Graph. The internal CFG nodes are then amended with Assembly extracted from the program’s binary, since SDE’s Assembly output is not compatible with Machine Code Analyzers. Our program subsequently executes a Machine Code Analyzer for each basic block, getting in return an estimated cycles-per-iteration metric (CPI). We record the per-block CPI at the directed CFG edge from caller to callee, which already holds the number of invocations of this edge, effectively creating a “weighted” graph. Figure 3 showcases the result and it is easy to see that the summation of all edges in the CFG is equivalent to the estimated runtime of the entire workload (assuming all data is inside the L1 data cache).

The above outlined approach works not only for sequential programs, but also for parallel applications. Intel SDE can record the instruction execution and caller/callee dependencies for thread-parallel programs, e.g. pThreads, OpenMP, or TBB. Furthermore, we can attach SDE to individual MPI ranks to get the data for it. Therefore, we are able to estimate the runtime for MPI-X parallelized HPC applications by the following equation:

\[
L_{app} := \max_{v \in \text{ranks}} \left( \max_{t \in \text{threads}} \left( \sum_{\text{edges } e \in \text{CFG}_{v,t}} \text{CPL}_e \cdot \#\text{calls}_e \right) / \text{processor frequency in Hz} \right)
\]

under the assumption that MPI ranks and threads do not share computational resource \(^1\).

The self-imposed restriction of Machine Code Analyzers is the limited accuracy compared to cycle-accurate simulators, due to their distinct design goal. To improve our CPI estimate, we rely on four different MCAs, namely llvm-mca \[39\], Intel ACA (IACA) \[42\], uiCA \[43\], and OSACA \[44\], and take the median of the results. Another shortcoming of MCA tools is that most of them estimate the throughput of basic blocks in isolation while assuming looping behavior of the assembly block (PC

\(^1\)Resource over-subscription is outside the scope of this study and our tool.
jumps from last back to first instruction). Neither “block looping” nor an empty instruction pipeline (single iteration of the block) are realistic for some blocks. Hence, for non-looping basic blocks, we estimate the CPI by the MCA tool with the blocks of caller and callee, and the callee’s CPI is calculated by subtracting the cycle of retirement of its last instruction from the caller’s last instruction retirement (instead of when the callee’s first instructions are decoded, which can overlap with execution of caller instructions). Further, we correct some cycle estimates for specific instructions within our tool in a post-processing step, since we encountered a few unsupported or grossly mis-estimated instructions while validating our tool against known benchmarks. We refer the reader to Section [V-A] for more details.

B. Cycle-level Accuracy: gem5-based Processor Simulations

While the machine code analyzers give us a first-order approximation, we are still interested in highly accurate predictions for our proposed 3D-stacked, cache-rich CPU. Hence, we employ an open-source system architecture simulations, called gem5 [33]. This simulator supports Arm, x86, and RISC-V CPUs to varying degrees of accuracy, and can be extended with memory models and plugins for higher simulation fidelity of the memory subsystem. We use gem5’s “syscall emulation” to executes applications directly without booting a Linux kernel.

Fortunately, RIKEN released their gem5 version which is specially tailored for A64FX’s co-design to support SVE, HBM2, and other advanced features [45]. Hence it is well suited to simulate our LARC proposal in Section II-D. This version of gem5 has been validated for A64FX [46], and can be used with production compilers from Fujitsu.

While experimenting with RIKEN’s gem5, we noticed a few drawbacks, such as the lack of support for: (i) dynamically linked binaries; (ii) adequate memory management (freeing memory after application’s free() calls); (iii) simulating more than 16 CPU cores due to limits in the cache coherence protocol; (iv) multi-rank MPI-based programs; and (v) simulating more than one A64FX CMG.

We modify gem5 to remedy the first three problems. However, the last two problems remain intractable without major changes to the simulator’s codebase, and hence we limit ourselves to single-CMG simulations (with one MPI rank). Relying on the assumption that most HPC codes are weak scaled across multiple NUMA domains and compute nodes, we believe the single-rank approach still serves as a solid foundation for future performance projection.

However, even single-rank MPI binaries require numerous unsupported system calls. To circumvent this problem, we extend and deploy a MPI stub library [47].

C. Relevant HPC (Proxy-)Applications and Benchmarks

Instead of relying on a narrow set of cherry-picked applications, we attempt to cover a broad spectrum of typical scientific workloads executed on modern supercomputers. We customize and extend a publicly available benchmarking framework [48], with a few additional benchmarks and necessary features to perform the MCA- and gem5-based simulations. Hereafter, we detail the list of 127 included workloads, summed up across all benchmark suites, which are sized to fit within a single node and which could be simulated with gem5 in a reasonable time (≤ six months).

1) Polyhedral Benchmark Suite: The PolyBench/C suite contains 30 single-threaded, scientific kernels which can be parameterized in memory occupancy (∈ [16 KiB, 120 MiB]) [50]. Unless stated otherwise, we use the largest configuration.

2) TOP500, STREAM, and Deep Learning Benchmarks:
   a) High Performance Linpack (HPL) [51] solves a dense system of linear equations $Ax = b$ of size 36,864 in our case. High Performance Conjugate Gradients (HPCG) [52] applies a conjugate gradient solver to a system of linear equation (represented by a sparse matrix $A$). We choose $120^3$ for HPCG’s global problem size. BabelStream [53] evaluates the memory subsystem of CPUs and accelerators, and we configure 2 GiB input vectors. Additionally, we implement a micro-benchmark to approximate the single-precision GEMM operation ($m = 1577088; n = 27; k = 32$) which is commonly found in 2D deep convolutional neural networks (CNN), such as 224×224 ImageNet classification workloads [54].

3) NASA Advanced Supercomputing Parallel Benchmarks: The NAS Parallel Benchmarks (NPB) [55], [56] consists of nine kernels and proxy-apps which are common in computational fluid dynamics (CFD). The original MPI-only set has been expanded with ten OpenMP-only benchmarks [57] and we select the class B input size for all of them.

4) RIKEN’s Fiber Mini-Apps and TAPP Kernels: To aid the co-design of Supercomputer Fugaku, RIKEN developed the Fiber proxy-application set [58], a benchmark collection representing the scientific priority areas of Japan. Additionally, RIKEN released numerous scaled-down TAPP kernels [59].
of their priority applications which are tailored for fast simulations with gem5 [46]. The proxy-apps and workloads we use are as follows. FFB [60] with the 3D-flow problem discretized into 50×50×50 sub-regions; FFVC [61] using 144×144×144 cuboids; MODYLAS [62] with the wat222 workload; mVMC [63] with the strong-scaling test reduced to 1/8th of the samples and 1/3rd of the lattice size; NICAM [64] with a single (not 11) simulated day; NTChem [65] with the H₂O workload; QCD [66] with the class 2 input.

5) Exascale Computing Project Proxy-Applications: Similar to RIKEN, the US-based supercomputing centers curated a co-design benchmarking suite for their recent exascale efforts [67]. We select eleven applications of the aforementioned benchmarking framework with the following workloads. AMG [68] with the problem 1 workload; CoMD [69] with the 256,000-atom strong-scaling test; Laghos [70] modelling a 3D Šedoň blast but with 1/6th of the timesteps; MACSio [71] with an ≈ 1.14 GiB data dump distributed across many JSON files; MiniAMR [16] simulating a sphere moving diagonally through 3D space; MiniFE [16] with 128×128×128 grid size; MiniTri [72] testing triangle- and largest clique-detection on BCSSTK30 (MatrixMarket [73]); Nekbone [74] with 8,640 elements and polynomial order of 8; SW4lite [75] simulating a pointsource; SWFFT [76] with 32 forward and backward tests for a 128×128×128 grid; XSBench [77] with the small problem and 15 million particle lookups.

6) SPEC CPU and SPEC OMP Benchmarks: The Standard Performance Evaluation Corporation [78] offers, among other benchmarks, two HPC-focused test suits: SPEC CPU[speed] (ten integer-heavy, single-threaded; ten OpenMP-parallelized, floating-point benchmarks) and SPEC OMP (14 OpenMP-parallelized benchmarks). All SPEC tests hereafter are based on non-compliant runs with the train input configuration.

IV. SIMULATOR VALIDATION AND SIMULATION RESULTS

Sections IV-A and IV-B are dedicated to our MCA-based estimation of the upper bound on performance improvement with abundant L1 cache. First, we evaluate the accuracy of this approach, and then apply the methodology to our benchmarking sets. Similarly, Sections IV-C, IV-D, and IV-E highlight our validation of gem5 for our proposed CPU architecture and evaluate numerous benchmarks and proxy applications on said architecture.

A. MCA-based Simulator Validation

During the development of our MCA-based simulator, we implemented numerous micro-benchmarks to fine-tune the CPI estimation capabilities while comparing the results to an Intel Broadwell (E5-2650V4). Our micro-benchmarks comprise MPI/OpenMP-only, MPI+OpenMP, and single-threaded tests (exercising recursive functions, floating-point- or integer-intensive operations, L1-localised, or stream-like operation).

Needless to say, applying MCA-based simulations to full workloads or complex application kernels is still error-prone, since these tools are designed to analyze small Assembly sequences without guarantee for accurate absolute performance numbers. Regardless, we validate the current status of our tool using PolyBench/C with MINI inputs. In theory, these input sizes (≈ 16 KiB) should all fit into the 32 KiB L1D cache of the Broadwell. Hence, measuring the kernel execution time for these PolyBench tests should yield numbers close to MCA-based runtime estimates. For the baseline measurements, we set all cores of the Broadwell to 2.2 GHz, set the uncore to 2.7 GHz, and disable turbo boost; compile each workload with Intel’s Parallel Studio XE 14 and execute every test for 100 times (since many only run for a few ms) to determine the fastest possible execution time. The difference between the real baseline results and our MCA-based estimates is visualized in Figure 4 as projected relative runtime difference.

The data shows that on average our MCA-based method slightly overestimates a faster execution. Only seven out of 30 workloads are expected to run slower than what we observe on the real Broadwell (i.e., y-value ≤ 1). For eight of the PolyBench tests, our tool estimates the runtime to be over 2x faster than our measurements. Hence, we can conclude that for 73% of the micro-benchmarks the MCA-based method is reasonably accurate: within 2x slower-to-2x faster. While a 2x discrepancy might appear high, we have to point out that our cross-validations using SST [79], [80] and third-party gem5 models [81] for Intel CPUs yield similar inaccuracies but our MCA-based method is substantially faster.

Another indicator for the accuracy of our MCA-approach can be drawn from DGEMM. Theoretically, DGEMM performs close to peak and is not memory-bound for large matrices, and hence the measured runtime and MCA-based estimates are expected to match. Unfortunately, PolyBench’s Gflop/s rate for gemm is far from peak (due to its hand-coded loop-nest), and therefore we replace it with an Intel MKL-based implementation of equal matrix dimensions. For the PolyBench input sizes MINI,...,EXTRALARGE in our MKL-based implementation, our MCA tool estimates a faster runtime by 6.4x, 75%, 11%, 1.9%, and 1.5%, respectively. This closely matches the achievable single-core Gflop/s of the E5-2650V4: for MINI and the MKL-based runs, we measure only 2 Gflop/s, while for EXTRALARGE we peak out at the

Fig. 4. Validation of MCA-based performance estimation against PolyBench/C MINI with inputs fitting into L1D; Relative runtime shown (vs. Intel E5-2650V4 measurements); Values ≥ 1 show prediction of faster execution

2For details of flags, tools, versions, and exec. env, please refer to AD/AE.

3A large-scale survey of academic simulators in realistic scenarios, beyond carefully selected and tuned micro-kernels, is—in our humble opinion—consequential yet outside the scope of this paper.
expected 32 Gflop/s. The low Gflop/s measurements for MINI (and SMALL) demonstrate that MKL is not yet compute-bound, and hence causes the 6.4x (and 75%) misprediction.

B. Speedup-potential with Unrestricted Locality

Knowing the limits of the MCA-based approach, we can take on the entire benchmark suite from Section III-C and evaluate their speedup potential when all data fits into L1.

The baseline measurements are conducted similar to Section IV-A on a dual-socket Intel E5-2650v4 system with 48 cores (2-way hyper-threading enabled). For all listed benchmarks, excluding SPEC CPU and OMP, we focus on the solver times only, i.e., we ignore data initialization and post-processing phases. Since most of the proxy-apps are parallelized with MPI and/or OpenMP, we perform an initial sweep of possible configurations of ranks and threads to determine the fastest time-to-solution (TTS) for our strong-scaling benchmarks, and the highest figure-of-merit (as reported by the benchmark) for weak-scaling benchmarks. The highest performing configuration is executed ten times to determine the TTS of the kernel as our reference point in Figure 5.

The MCA-based estimate. Under the assumption that some MPI-parallelized benchmarks experience imbalances, we randomly sample up to nine ranks (in addition to rank 0) execute the selected rank with Intel SDE (and the remaining ranks normally), and calculate the estimated runtime using Equation 1 and the 2.2 GHz processor frequency. The resulting runtime is divided by the measured runtime to determine the upper-bound speedup potential per application when all its data would fit into L1D, see Figure 5.

For PolyBench/C workloads, we see similar speedup trends as for its smallest inputs which we used in Figure 4 although the expected speedup for EXTRALARGE increases to a peak of 8.4x for the ludcomp kernel. Only four kernels show no performance increase, presumably by being compute-bound and not bandwidth-bound: 2mm, 3mm, doitgen, and trisolv. Overall, the MCA-based approach estimates a geometric mean ($GM$) performance increase of 2.9x from fitting all data into L1D. RIKEN’s TAPP kernels benefit the most from unrestricted locality. Especially Kernel 20, which represents one core of the FFB application, shows a speedup of 20x. Altogether, we see a projection of $(GM)=2.6x$ increased performance, but also two cases (Kernel 5 and 9) where the MCA tool estimates nearly 100% slowdown. These two are from GENESIS and NICAM, respectively, but as detailed in Section IV-A some inaccuracy is expected as the trade-off for the faster simulation time.

NPB’s OpenMP version of a conjugate gradient (CG) solver is another workload with large theoretical performance gain of 13.1x. In total, we expect a $(GM)=3x$ gain for all NAS Parallel Benchmarks; specifically, $(GM)=4x$ for the OpenMP versions and $(GM)=2.3x$ for the MPI versions, respectively. The potential gain for CG is not surprising, since these solvers are predominantly bound by memory bandwidth and are sensitive to memory latency [83]. High Performance Linpack is unsurprisingly not expected to gain any performance by placing all its data into L1 cache, as this benchmarks is compute-bound. In fact, our MCA tool expected a small runtime decrease of 11%. By contrast, DLproxy, which uses MKL’s SGEMM, would benefit from a large L1, since MKL cannot achieve peak Gflop/s for the tall/skinny matrix in this workload (cf. Section III-C). XSBench and miniAMR show the highest gains for ECP’s and RIKEN’s proxy-apps, with a value of 7.3x and 4.5x, respectively. This appears to be in line with expectation from the roofline characteristics of the benchmarks when measured on a similar compute node [84].

A deeper look at roofline analysis in [84] reveal that there is no strong correlation between the position of an application on the roofline model and the expected performance gain.
from solely running out of L1D cache. We speculate that other, hidden bottlenecks are exposed by our MCA approach, such as data dependencies and lack of concurrency in the applications, which limit the expected speedup. Apart from noticeable outliers in the expected speedup, such as lbm, bwaves, botsspar, and iblbc, the potential from enlarged L1D is rather slim for SPEC, and only \((G:M)=1.9x\) runtime reduction can be expected across all 33 workloads. Our simulation with gem5, in the following sections will reveal if those speedups can also be obtained by sufficiently large, 3D-stacked L2 cache, or if we fall short on our expectations.

C. LARC Simulation in gem5 and the A64FX3 Baseline

As we discussed in Section II-D we envision one LARC CMG to have 32 cores, 384 MiB L2 cache, and 1.6 TB/s L2 bandwidth. Regrettfully, gem5 (at least RIKEN's version) can only be configured with L2 cache sizes that are \(2^X\), and therefore we either have to scale up or down LARC's L2 cache size. The compromise is to explore both as distinct options, one conservative and one technologically aggressive configuration. The conservative option, called LARC\(_C\), is limited to 256 MiB L2 cache at \(\sim\) 800 GB/s, while the aggressive version, LARC\(_A\), doubles both values, to 512 MiB and \(\sim\) 1.6 TB/s, respectively.

Starting at a baseline, i.e., a simulated version of A64FX which we label as A64FX\(_S\), and in order to materialize the properties of the LARC CMG (cf. Section II-D), we modify three parameters in our gem5 model. We modify: (i) the number of cores in the system to match 32 (up from A64FX\(_S\)'s baseline of 12); (ii) the size of the total L2 cache to match the capacity of the eight stacked layers (256/512 MiB, up from A64FX\(_S\)'s L2 size of 8 MiB per CMG); and (iii) we adjust the number of L2 banks in LARC\(_A\) to control the bandwidth.

Furthermore, we introduce a fourth gem5 configuration, called A64FX\(_{32}\), which simulates one baseline A64FX\(_S\) CMG but with 32 cores. These four configurations A64FX\(_S\)→A64FX\(_{32}\)→LARC\(_C\)→LARC\(_A\) should provide us with the means to determine the speedup gains from the larger core count and larger L2 cache, individually. The processor's core frequency is universally set to 2.2 GHz. Table I summarizes the four gem5 configurations.

D. gem5-based Simulation and Configuration Validation

We perform OpenMP tests to verify our gem5 simulator for up to 32 cores. For the L2 cache size and bandwidth changes, we employ a STREAM Triad benchmark, parameterized to avoid cache line conflicts among participating threads. Splitting the A64FX\(_S\) CMG L2 cache into 12 chunks (one per thread) yields a working size of 683 KiB. Hence, the three 128 KiB vectors of the Triad operation will fit into the L2 cache. We increase the total vector size in proportion to the number of threads and test the achievable L2 bandwidth for LARC\(_C\) and LARC\(_A\). Additionally, Figure 6 includes the baseline A64FX\(_S\) CMG scaled to 12 cores. The simulation shows that LARC\(_C\)'s L2 bandwidth peaks out at 792 GB/s and LARC\(_A\)'s bandwidth goes up to 1450 GB/s for this particular test case, which is, respectively, 1% and 9% lower than our estimates shown above. The baseline A64FX\(_S\) closely matches the bandwidth of the real A64FX CPU executing this test.

Another validation test we perform is setting the number of cores to the maximum (12 and 32, respectively) and scale the vector size from 2 KiB per core to a total of 1 GiB for the three vectors. Figure 7 shows the results for this simulation. In the memory range of tens to hundreds of KiB, the Triad operation can be done from L1 cache, for which LARC\(_C\) and LARC\(_A\) show higher bandwidth. Their 2.7x higher core count results in 2.6x higher aggregated L1 bandwidth. For the Triad, for the memory sizes that fit into L2 cache, we see a behavior similar to Figure 6. Past 8 MiB, the A64FX\(_S\) configuration shows the expected bandwidth drop to HBM2 level, while for LARC\(_C\) and LARC\(_A\), the expected L2 cache bandwidth is maintained.

### Table I

| Configuration   | A64FX\(_S\) | A64FX\(_{32}\) | LARC\(_C\) | LARC\(_A\) |
|-----------------|-------------|---------------|------------|------------|
| Area (per CMG)  | 48 mm\(^2\) | N/A           | 12 mm\(^2\) | 12 mm\(^2\) |
| Cores (per CMG) | 12          | 32            | 32         | 32         |
| Core config.    | Arm v8.2 + SVE, 512bit SIMD width, 2.2 GHz | OoO 128 ROB entries, dispatch width: 4 |
| Branch pred.    | Bi-mode: 16K global predictor, 16K choice predictor |
| Per-core L1D    | 64 KiB 4-way set-associative, 3 cycles, adjacent line pref. |
| L2 cache/BW (per CMG) | \(~\) 800 GB/s | \(~\) 800 GB/s | \(~\) 1600 GB/s |
| L2 config.      | 16-way set-associative, 37 cycles, inclusive, 256B block |
| Main Memory     | 32 GiB HBM2, 4 channels, 256 GB/s |
E. Speedup-potential with Restricted Locality

To further refine our projections gained by abundant L2 cache, we proceed with the cycle-level simulations of the proxy-applications and benchmarks listed in Section III-C.

We compile all benchmarks with Fujitsu’s Software Technical Computing Suite (v4.6.1) targeting the real A64FX, and simulate the single-rank workloads in gem5 for our four versions. Unfortunately, three of our MPI-based benchmarks require multi-rank MPI: MODYLAS (requires ≥ 8 ranks), NICAM (≥ 10 ranks), and NTChem (complex message pattern with rank 0 sending/receiving from itself), and hence we omit them. Furthermore, we skip the MPI-only versions of NPB.

The per-configuration speedup is given relative to the baseline A64FXs configuration. We exclude initialization and post-processing times, and measure only the main kernel runtime, except for the SPEC benchmarks as described in Section IV-B. These results are presented in Figure 8 and show the effects of the gradual expansion of simulated resources. The average (single CMG) speedups from LARC_C and LARC_A are ≈ 1.9x and ≈ 2.1x, respectively, with some applications reaching ≈ 4.4x for LARC_C and ≈ 4.6x for LARC_A.

As expected, most benchmarks benefit from the additional cores and increased cache size, most prominently MG-OMP which gains a small speedup of ≈ 1.3x from the extra cores, ≈ 2x speedup from the extra cache, and with 512 MiB cache and higher bandwidth reaches a speedup of ≈ 4.6x. Comparable incremental improvements with all three architecture steps are observable in other workloads, such as TAPP Kernel 7 and 17, showing good scaling on multiple cores and being memory-bound since they benefit from both the higher core count and the cache size upgrade. Kernel 19 and 20, XSBench, roms, and imagick (SPEC OMP) show similar gain in runtime, but the difference between LARC_C and LARC_A is significantly smaller, implying that the problem size either fits into the 256 MiB L2 (e.g., XSBench) or the workload arrives at a point of diminishing returns from the 2x larger cache. Kernel 8, 9, 12–15, and FT-OMP suffer a slowdown from cache contention in LARC_A. LARC_C and LARC_A mitigate the cache contention resulting in speedups similar to the benchmarks discussed earlier. EP-OMP, CoMD, and other compute-bound benchmarks benefit only from the higher core count, with both LARCs providing similar speedup as A64FX.

Expectedly, single-threaded workloads (all of PolyBench’s benchmarks) show little to no improvements over A64FX_S, i.e., they do not benefit from more cores. However, these benchmarks also do not show a performance gain from a larger 3D-stacked L2 cache, albeit their working set size exceeding A64FX_S, i.e., 8 MiB L2 but fitting into LARC’ larger cache. We only see a limited speedup of (G/M)=4.3% across all of them and no noteworthy outliers, and hence omit them in Figure 8. We attribute other outliers, such as the slowdown of imagick (SPEC-CPU), to similar intrinsic property of the benchmark: our testing on a real A64FX reveals that imagick has a sweetspot at 8 OpenMP threads, and scales negatively thereafter; and the TAPP Kernels 3–6 and 18 were customized for the 12-core A64FX CMG and cannot run effectively on 32 threads without a rewrite. Hence, we limit gem5 to 12 cores for these TAPP kernels, and we see that only Kernel 18 benefits from a larger L2. Further proxy-applications and benchmarks missing from Figure 8 yet appearing in Figure 5 are the unfavorable result of persistent, repeatable simulator errors—sometimes occurring after months of simulation.

We should note that in some cases the benchmarks’ implementation and the quality of the compiler may skew the results, for instance, BabelStream measuring memory bandwidth on a 2 GiB buffer. Being unoptimized for A64FX, BabelStream’s baseline underperforms in terms of per-core BW (compared to STREAM Triad tests in Figure 6 and 7) which in turn results in performance gain when the number of cores increases to 32. Overall, the speedup on A64FX can originate from the
following reasons: (i) the program is compute-bound (a valid result); (ii) the workload exhibits both compute-bound and memory-bound tendencies in different components of a proxy-application (a valid result); (iii) the program is highly latency-bound, and hence the speedup can be the result of the larger aggregate L1 cache (a valid result); or (iv) a poor baseline resulting in a slightly misleading result.

We confirm the validity of attributing improvement to the high capacity L2 by inspecting the L2 cache-miss rates of our gem5 simulations (miss rate of some selected examples listed in Table II). The reduction in cache-miss rates reported in the table is consistent with the performance improvements we observe in Figure 8.

F. Summary of the Results

In summary, our gem5 simulations indicate that more than half (29 out of 52) of the applications experienced a larger than two times speedup on LARC\textsuperscript{A} compared to the baseline A64FX\textsubscript{S} CMG. For over two-thirds (23 out of 29) of these applications, the performance gains are directly attributed to the larger (3D-stacked) cache. Most notably, out of all the RIKEN TAPP kernels that experienced meaningful speedup on this new platform, a majority benefited from the expanded cache, rather than the core increase. This carried particular importance as these kernels are highly tuned for A64FX.

V. DISCUSSION AND LIMITATIONS

In this study, we simulated a single LARC CMG in gem5, and its potential future effect on common HPC workloads.

A. The Prospect of LARC

In reality, if a LARC processor were incepted in 2028, it would contain 16 LARC CMGs, which correspond to the same silicon area as the current A64FX CPU, and it is important to understand what impact such a processor would have on the HPC community and its applications. Unfortunately, it is hard to give a conclusive answer to such a forward looking question today. However, if we do ideal scaling of both A64FX and LARC CMGs and compare at the full chip level, then a LARC system in 2028 could give between 4.91x ($\times z$; SPEC CPU) and 18.57x ($MG$—OMP; NPB) performance improvements over the current A64FX processor with an average improvement of ($GM=9.77x$ for applications that are responsive to larger cache capacity. For applications that do not obviously benefit from a larger cache, future studies should (continue to) consider algorithmic improvements, as well as investigate the potential of allocating parts of the cache to vary compute capabilities, for example, processing-in-memory (PIM) [22] or alternative compute modules, e.g., CGRAs [85].

B. Considerations and Limitations

Our MCA-based estimation framework only gives a first-order approximation for a hypothetical CPU with sufficiently large L1 cache to host the entire data structures of a specific workload. This approach has some advantages and disadvantages and should be used with caution, but it also has capabilities which we have not yet detailed, such as estimating the runtime of the same binary/workload for different (ISA-compatible) x86 systems by simply replacing the MCA target architecture and altering the assumed CPU clock frequency.

We emphasize that we run applications as they are, i.e., without any algorithmic optimizations to the larger last level cache, in our MCA- and gem5-based simulators. This is also true to our motivating experiment shown in Figure 1. While the cache capacity of AMD’s Milan-X CPU is about three times that of Milan, it is far from what we envision in 2028. Hence, our Milan-X results serve as a first-order indication of what SRAM—in its current available SoTA—can offer.

Another notable aspect, which is outside the scope of this study, is the heat dissipation of CPU cores in face of the 3D-stacked cache. It has been reported that AMD’s Milan-X carefully stacks caches above areas of the chip that are not used for compute, i.e., mostly above caches [86]. Our assumption is that, by 2028, manufacturing technologies will have advanced enough to overcome this limitation.

VI. RELATED WORK

Stacked Memory and Caches: The size of LLC has increased for the last 25 years [87], a trend anticipated to continue into the future. Yet, 2D IC becomes hard to exploit for additional performance, despite recent attempts by IBM [88], [89]. However, 3D-stacking is becoming a promising alternative [14], as demonstrated by AMD’s 3D V-Cache [15] or Samsung’s proposed 3D SRAM stacking solution [90] based on 7 nm TSVs. Moreover, academics explored 3D-stacked DRAM cache [91], [92], but these incur much higher latency and power consumption [23], [26]. Non-Volatile Memory is considered as LLC alternative, yet it suffers similar latency issues [93]. However, what differs our work from the work of our peers is: (i) we focus on the impact of future caches, several magnitudes larger than those found today.

Performance modeling tools and methodologies: Computer architecture research is often based on simulators, such as the Structural Simulation Toolkit (SST) [79] or CODES [44], for efficiently evaluating and optimizing HPC architectures and applications. The gem5 simulator, by Binkert et al. [33], is widely used by academia and vendors for micro- and full-system architecture emulation and simulation. It supports validated models for x86 [81] and Arm [46]. We refer the interested reader to [www.gem5.org/publications/ for an comprehensive library of gem5-based research and derivative works. However, what differs our work from the work of our peers is: (ii): unlike prior work that utilizes (relatively) small kernels, our work operates on large-scale MPI/OpenMP-parallelized proxy-applications in order to quantify the impact of caches on realistic workloads. To our knowledge of reported research-driven gem5 simulations, this is the largest scale of cycle-accurate simulations conducted in terms of the aggregate number of instructions simulated ($6.08 \times 10^{13}$).

Other methodologies such as MUSA by Grass et al. [95] are closer to our MCA-based approach, since MUSA uses PIN which is the basis for Intel SDE (used in this study), but focus
on MPI analysis and multi-node workloads. We are not the first to utilize Machine Code Analyzers, see [43, 44] and derivative works such as [96–101]. However, what differs our work from the work of our peers is: (iii) instead of estimating accurate performance of existing system architectures, our MCA-based approach tries to gauge the upper-bound in obtainable performance, and exposes hidden bottlenecks better than the roofline approach, for common HPC applications.

VII. CONCLUSION

In this paper, we aspire to understand the performance implications of emerging SRAM-based die-stacking technologies on future HPC processors. We first designed a methodology to project the upper bound that an infinitely large cache would have on relevant HPC applications. We find that several well-known HPC applications and benchmarks have ample opportunities to exploit an increased cache capacity, with outliers reaching up to 20x speedup when given infinite cache.

We further expand our study by proposing a hypothetical processor (called LARC) in a 1.5 nm technology that could—given current technological roadmaps—be introduced in the year 2028. This hypothetical processor exploits the pinnacle of 3D stackable memories and would have nearly 6 GiB L2 cache memory; compared to our baseline A64FX5 CPU architecture with 32 MiB L2 cache. Next, we exercise a single LARC CMG using a plethora of HPC applications and benchmarks using the gem5 simulator and contrast the observed performance against the existing A64FX5 CMG. We find that the LARC CMG would (on average) be 1.9x faster than the corresponding A64FX5 CMG, albeit consuming $\frac{1}{3}$th of the area. When area-normalized to the real A64FX5 CMG (by assuming optimistic ideal scaling), we can expect to see an average boost of 9.77x for cache-sensitive HPC applications by the end of this decade.

Finally, we expect that the larger caches will motivate and facilitate algorithmic advances that in combination with the abundant cache can potentially yield an order of magnitude gain in performance. Instead of blindly expanding ALUs, we firmly believe that the combination of high-bandwidth, large, 3D-stacked caches, and algorithmic advances, is the path moving forward to next generation of high-performance processors.

REFERENCES

[1] G. E. Moore, “Progress in Digital Integrated Electronics,” International Electron Devices Meeting, IEEE, vol. 21, pp. 11–13, 1975.

[2] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, “Design of ion-implanted MOSFET’s with very small physical dimensions,” IEEE Journal of Solid-State Circuits, vol. 9, no. 5, pp. 256–268, 1974.

[3] J. Hruska, “The death of CPU scaling: From one core to many – and why we’re still stuck,” Feb. 2012. [Online]. Available: https://www.extremetech.com/computing/116561-the-death-of-cpu-scaling-from-one-core-to-many-and-why-were-still-stuck

[4] Gottlieb, Grishman, Kruskal, McAffuliffe, Rudolph, and Snir, “The NYU Ultracomputer—Designing a MIMD Shared Memory Parallel Computer,” IEEE Transactions on Computers, vol. C-32, no. 2, pp. 175–189, 1983.

[5] M. Sato, Y. Ishikawa, H. Tomita, Y. Kodama, T. Odajima, M. Tsujii, H. Yashiro, M. Aoki, N. Shida, I. Miyoshi, K. Hiriti, A. Furuya, A. Asato, K. Nakada, and T. Shimizu, “Co-Design for A64FX-5: Coarse Processor and Fugaku,” in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ser. SC ’20. IEEE Press, 2020, event-place: Atlanta, Georgia.

[6] D. Suggs, M. Subramony, and D. Bouvier, “The AMD “Zen 2” Processor,” IEEE Micro, vol. 40, no. 2, pp. 45–52, 2020.

[7] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, “GPGPU Computing,” Proceedings of the IEEE, vol. 96, no. 5, pp. 879–899, 2008.

[8] J. Nickolls and W. J. Dally, “The GPU Computing Era,” IEEE Micro, vol. 30, no. 2, pp. 56–69, 2010.

[9] T. N. Theis and H.-S. P. Wong, “The End of Moore’s Law: A New Beginning for Information Technology,” Computing in Science Engineering, vol. 19, no. 2, pp. 41–50, 2017.

[10] J. S. Vetter, E. P. DeBenedictics, and T. M. Conte, “Architectures for the Post-Moore Era,” IEEE Micro, vol. 37, no. 04, pp. 6–8, Jul. 2017.

[11] N. Hemsoth, “A Rogues Gallery of Post-Moore’s Law Options,” Aug. 2018. [Online]. Available; https://www.nextplatform.com/2018/08/27/a-rogues-gallery-of-post-moores-law-options/

[12] M. M. Shulaker, T. F. Wu, M. M. Sabry, H. Wei, H.-S. P. Wong, and S. Mitra, “Monolithic 3D Integration: A Path from Concept to Reality,” in Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition, ser. DATE ’15. San Jose, CA, USA: EDA Consortium, 2015, pp. 1197–1202, event-place: Grenoble, France.

[13] X. Hu, D. C. Stow, and Y. Xie, “Die Stacking Is Happening,” IEEE Micro, vol. 38, no. 1, pp. 22–28, 2018.

[14] M. Evers, L. Barnes, and M. Clark, “The AMD Next Generation Zen 3 Core,” IEEE Micro, pp. 1–1, 2022.

[15] M. A. Heroux, D. W. Dorrerler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, “Improving Performance via Mini-applications,” Sandia National Laboratories, Tech. Rep. SAND2009-5574, 2009.

[16] G. Bonshor, “AMD Releases Milan-X CPUs With 3D V-Cache: EPYC 7003 Up to 64 Cores and 768 MB L3 Cache,” Mar. 2022. [Online]. Available; https://www.anandtech.com/show/17332/amd-releases-milan-x-cpus-with-3d-vcache-epyc-7003

[17] S. A. McKee, “Reflections on the Memory Wall,” in Proceedings of the 1st Conference on Frontiers in Computing, ser. CF ’04. New York, NY, USA: Association for Computing Machinery, 2004, p. 162, event-place: ischia, Italy.

[18] J. D. McCalpin, “Memory Bandwidth and Machine Balance in Current High Performance Computers,” IEEE Technical Committee on Computer Architecture (TCCA) Newsletter, Dec. 1995.

[19] Z. Or-Bach, “A 1.00x Improvement in Computer Systems by Bridging the Processor-Memory Gap,” in 2017 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S), 2017, pp. 1–4.

[20] G. F. Oliveira, J. Gómez-Luna, L. Orosa, S. Ghose, N. Vijaykumar, I. Fernandez, M. Sadrosadati, and O. Mutlu, “DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks,” IEEE Access, vol. 9, pp. 134457–134502, 2021.

[21] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, “Near-Data Processing: Insights from a MICRO-46 Workshop,” IEEE Micro, vol. 34, no. 4, pp. 36–42, 2014.

[22] M. Evers, L. Barnes, and M. Clark, “The AMD Next Generation Zen 3 Core,” IEEE Micro, pp. 1–1, 2022.

[23] M. A. Heroux, D. W. Dorrerler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich, “Improving Performance via Mini-applications,” Sandia National Laboratories, Tech. Rep. SAND2009-5574, 2009.

[24] G. Bonshor, “AMD Releases Milan-X CPUs With 3D V-Cache: EPYC 7003 Up to 64 Cores and 768 MB L3 Cache,” Mar. 2022. [Online]. Available: https://www.anandtech.com/show/17332/amd-releases-milan-x-cpus-with-3d-vcache-epyc-7003

[25] S. A. McKee, “Reflections on the Memory Wall,” in Proceedings of the 1st Conference on Frontiers in Computing, ser. CF ’04. New York, NY, USA: Association for Computing Machinery, 2004, p. 162, event-place: Ischia, Italy.

[26] J. D. McCalpin, “Memory Bandwidth and Machine Balance in Current High Performance Computers,” IEEE Technical Committee on Computer Architecture (TCCA) Newsletter, Dec. 1995.

[27] Z. Or-Bach, “A 1.00x Improvement in Computer Systems by Bridging the Processor-Memory Gap,” in 2017 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S), 2017, pp. 1–4.

[28] G. F. Oliveira, J. Gómez-Luna, L. Orosa, S. Ghose, N. Vijaykumar, I. Fernandez, M. Sadrosadati, and O. Mutlu, “DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks,” IEEE Access, vol. 9, pp. 134457–134502, 2021.

[29] R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, “Near-Data Processing: Insights from a MICRO-46 Workshop,” IEEE Micro, vol. 34, no. 4, pp. 36–42, 2014.

[30] M. Evers, L. Barnes, and M. Clark, “The AMD Next Generation Zen 3 Core,” IEEE Micro, pp. 1–1, 2022.
[97] A. Renda, Y. Chen, C. Mendis, and M. Carbin, “DiffTune: Optimizing CPU Simulator Parameters with Learned Differentiable Surrogates,” in 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2020, pp. 442–455.

[98] C. Alappat, N. Meyer, J. Laukemann, T. Gruber, G. Hager, G. Wellein, and T. Wettig, “Execution-Cache-Memory modeling and performance tuning of sparse matrix-vector multiplication and Lattice quantum chromodynamics on A64FX,” Concurrency and Computation: Practice and Experience, vol. n/a, no. n/a, p. 30, Aug. 2021.

[99] C. Mendis, A. Renda, D. Amarasinghe, and M. Carbin, “Ithemal: Accurate, Portable and Fast Basic Block Throughput Estimation using Deep Neural Networks,” in Proceedings of the 36th International Conference on Machine Learning, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, Jun. 2019, pp. 4505–4515. [Online]. Available: https://proceedings.mlr.press/v97/mendis19a.html

[100] S. Corda, G. Singh, A. J. Awan, R. Jordans, and H. Corporaal, “Memory and Parallelism Analysis Using a Platform-Independent Approach,” in Proceedings of the 22nd International Workshop on Software and Compilers for Embedded Systems, ser. SCOPES ’19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 23–26.

[101] V. Caparrós Cabezas and P. Stanley-Marbell, “Parallelism and Data Movement Characterization of Contemporary Application Classes,” in Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures, ser. SPAA ’11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 95–104.