Lightweight AI for Efficient Resource Management in Heterogeneous-Core Architectures

Joshua Kim
The University of Texas at Austin

Chaojie Zhang
Microsoft Azure Research

Íñigo Goiri
Microsoft Azure Research

Christopher J. Rossbach
The University of Texas at Austin and Microsoft

Jovan Stojkovic
The University of Texas at Austin

Abstract

Datacenter applications increasingly exhibit fine-grained workload heterogeneity, driven by microservice architectures, datacenter "tax" operations, and diverse intra-service components. These factors lead to rapidly shifting execution phases with distinct and short-lived resource demands, which are difficult for static or hand-tuned resource management techniques to capture. In this paper, we present lightweight AI techniques for phase-aware resource management in heterogeneous-core server architectures. We characterize phase-level heterogeneity across representative datacenter workloads and show that simple threshold-based predictors struggle under fine-grained and noisy execution behavior. We then evaluate several low-cost machine learning approaches for online phase prediction and identify random forests as a promising design point, balancing accuracy, robustness, and inference overhead. We leverage the phase predictor to design a heterogeneous-core server architecture to improve Performance/Watt in datacenters while maintaining application transparency and ease of programming.

1 Introduction

Modern datacenter applications adopt a microservice architecture, decomposing monolithic logic into smaller, independently deployable services. This design improves modularity and manageability and is widely used by major IT companies [2, 8, 11, 24, 40–42]. As each microservice within an application displays unique compute, memory, and network requirements, they also demand different micro-architectural configurations for efficient execution. For performance, multiple microservices from the same application are often co-located on a single server.

Furthermore, microservice interconnections and increasing service complexity add intra-application heterogeneity. In particular, applications incur many "datacenter tax" operations (*e.g.*, compression and serialization) to connect microservices, each with distinct resource requirements. As services grow more complex, they also exhibit greater operational diversity within a single service.

For example, Figure 1 shows an Ad-serving workload [26] composed of 4 microservices: *Nginx* (front-end service), *Main Service* (application-specific computation), *Memcached* (in-memory caching), and *MySQL* (storage service). These services are connected by "tax" operations such as *(de)serialization* and *(de)compression*. In addition, within the *Main Service*, *PtrChase* first fetches data with memory-bound pointer chasing operations and then performs compute intensive operations on the data (*e.g.*, GEMM [32]).

Overall, these three sources (microservices, datacenter tax operations, and individual operations within a service) create a high-level of heterogeneity *within a single application*.

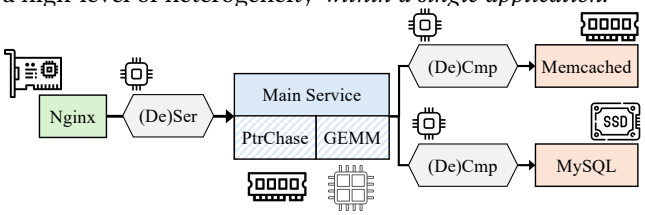

Figure 1. Ad-serving application composed of microservices, datacenter tax operations, and distinct operations within a service; each with distinct resource demands.

This workload heterogeneity is hard to analyze and exploit. First, microservice request rapidly shifts among μs-scale services with different resource needs, while other components (e.g., datacenter tax operations) occur at even finer time scales. We refer to these temporally stable intervals with distinct resource demands as *phases*: periods of relatively consistent behavior separated by abrupt transitions.

Second, the degree and mix of phases also vary widely across applications, which—together with their short lifetimes—makes static, hand-tuned analyses and optimizations ill-suited to capture the relevant trade-offs [6, 22, 44].

The heterogeneity matters because homogeneous, one-size-fits-all servers cannot be simultaneously well-matched to all phases: provisioning and tuning for compute-heavy phases wastes energy during stall-dominated phases, while tuning for efficiency can under-serve latency-critical compute phases. Heterogeneous servers could better match hardware resources to phase demands, but doing so requires a highly *dynamic* and *application-transparent* way to identify phases online and adapt execution accordingly.

To address this challenge, we propose a *lightweight AI–driven phase prediction* mechanism designed explicitly for

low overhead and rapid adaptation. Our system uses simple yet effective machine learning models, such as reinforcement learning and random forests, to infer execution phases from application agnostic hardware signals (e.g., cache misses, memory bandwidth, and system calls). To minimize latency and overhead, this predictor is implemented in hardware, enabling transparent, sub-millisecond phase awareness without requiring any changes to application code.

We then leverage this phase predictor to drive resource management in a *heterogeneous, chiplet-based server architecture* designed for next-generation datacenters. The server integrates multiple specialized chiplets, each optimized for a dominant class of execution phases—compute-intensive, memory-heavy, network-sensitive, or low-load. By dynamically classifying the current execution phase and steering threads to the chiplet best matched to its resource demands, the system aligns hardware capabilities with fine-grained workload heterogeneity, enabling both higher performance and improved resource efficiency without any code changes.

In summary, this work presents:

- Characterization of the phase-based execution heterogeneity in modern datacenter workloads.
- Analysis of the effectiveness of AI-driven approaches for phase prediction.
- A heterogeneous chiplet-based server design to dynamically match execution phases to best hardware.

2 Motivation

To understand the behavior of datacenter applications, we characterize the fine-grained phases in datacenter workloads running on standard datacenter hardware.

Experimental Setup. We run our analysis on an Intel Emerald Rapids server [16] with 28 2-way SMT cores at 3GHz. We use `cpu-freq-utils` to scale CPU frequency, and Intel's `CAT` [13] to control memory bandwidth and cache capacity. We use`perf` to collect metrics that capture utilization across key resource domains. We measure instructions per cycle (IPC), types of executed instructions, and frequency of branch misses to characterize compute phases, cache misses and memory bandwidth for memory phases, and I/O bandwidth and network system calls for network phases.

We use DCPerf [39], an open-source suite of applications that mimics Meta's services. This includes web services (*Mediawiki*, *Django*), object ranking (*Feedsim*), data caching (*TaoBench*), and CPU-based ML inference (*Adsim*).

To understand the fine-grained, varying behavior across these datacenter applications, Figure 2 inspects their IPC over time. High IPC indicates compute-intensive phases while low IPC indicates phases stalled by memory or network. First, we see ms-scale alternations between high- and low-IPC regions across workloads, even under stable input traffic. This repeated fluctuation in IPC points to rapid shifts in execution phases with unique resource demands. Second,

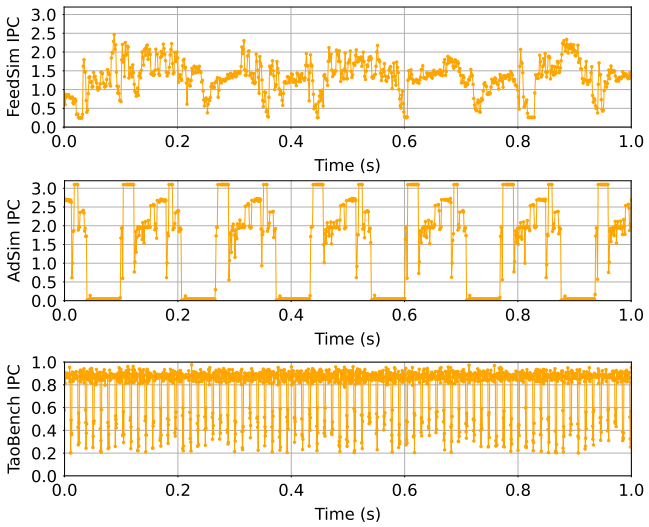

Figure 2. IPC over time for different DCPerf workloads: *FeedSim*, *AdSim*, and *TaoBench*.

we observe variation in both the magnitude of phase-level resource demand differences and the duration of recurring phase cycles. A higher level of disparity can be found in additional IPC over time graphs in the Appendix A.

This observation raises a question: *what drives such rapid and diverse phase behavior in datacenter workloads?* To this end, we characterize three sources of *phase heterogeneity*: *(1)* the microservice architecture, *(2)* the datacenter tax operations, and *(3)* internal sub-operations within a single service.

2.1 Heterogeneity in Microservices

Figure 3 focuses on execution classes across the microservices in a web-serving application [27]. *HHVM* service, the PHP execution server, exhibits high IPC and moderate cache activity, indicative of compute-intensive behavior. Memory services such as *Memcached* and *MySQL* show lower IPC but high LLC MPKI, reflecting their memory-bound nature. *Nginx*, the front-end web server, has moderate IPC but dominates in I/O bandwidth and the rate of networking system calls, making it representative of network-bound behavior.

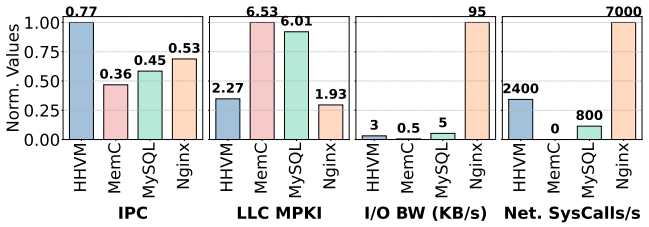

Figure 3. Normalized architectural metrics across different microservices within a web-serving application.

2.2 Heterogeneity in Datacenter Tax Operations

Datacenter tax operations: *(De)Compression*, *(De)Serialization*, *(De)Encryption*, and *Memcpy* follow similar heterogeneous

trends. *(De)Encryption* has the highest IPC with minimal branch misses, representative of its compute intensive behavior. *Memcpy* has the lowest IPC and significantly higher last-level cache (LLC) misses, in-line with its high memory traffic. *(De)Compression* and *(De)Serialization* fall between these extremes, displaying moderate IPC and LLC misses, reflecting mixed compute and memory bottlenecks.

2.3 Heterogeneity across Internal Components

We perform a sensitivity study of microarchitectural resources (*e.g.* core frequency, memory bandwidth, L2 capacity) across four representative intra-service operations. *Ranking* is a compute-intensive page-rank algorithm, *PtrChase* fetches data from a hash table via pointer chasing, *GEMM* aids in ML inference based Ad-serving, and *DeepCopy* performs a deep copy of output tensors from *GEMM*.

Figure 4 shows the scaling with core frequency across these components. Compute intensive *GEMM* and *Ranking* show high performance degradation at lower frequencies, while the other memory-centric operations are more lenient. When scaling memory bandwidth, *DeepCopy* suffers a 50% performance drop at 80% bandwidth capacity, while other operations are not as sensitive to this resource. Similarly, when scaling L2 capacity, only *PtrChase* suffers a 25% performance drop from 2MB to 1MB cache capacity, pointing to the memory-latency boundedness of pointer chasing.

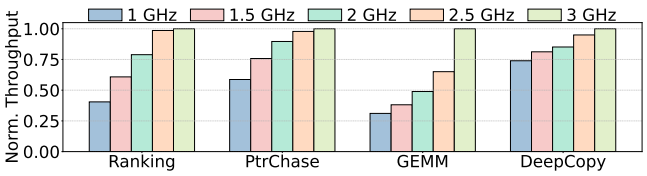

Figure 4. Scaling CPU frequency from 3 GHz to 1 GHz.

3 Lightweight AI for Phase Prediction

Our analysis shows that datacenter applications exhibit rapidly-shifting, fine-grained heterogeneous *phases*. Moreover, we see that this phase behavior is not uniform across applications. There is a high variation in how extreme resource-demand differences are across phases, and also in the length of the phases' repetitive execution cycles.

These challenges make it infeasible for profile-guided static analysis tools to accurately predict the phase of an executing thread. We confirm this limitation in a *Threshold-based* approach to phase prediction. This method creates a chain of conditionals on hand-optimized thresholds on profiling data points. For example, a task exhibiting high IPC with low networking calls and moderate cache activity is likely to be a compute-intensive task.

The accuracy of *Threshold-based* prediction was low, falling below 75% accuracy and even reaching 38% on fine-grained workloads. This indicates that there are nuanced behaviors that are hard to target statically by-hand.

Thus, we look to various AI-driven approaches to provide stronger reasoning ability. Unsupervised learning methods, namely clustering techniques such as K-Means, K-Medoids, HDBScan, and Hidden Markov Models (HMMs) group similar data points in feature space, capturing temporal or density-based patterns. Reinforcement learning (RL) models, including Multi-Armed Bandits and Contextual Bandits treat phase prediction as an action and accuracy as a reward to iteratively improve their phase selection. Random Forest, another machine learning approach, combines multiple pre-trained decision trees to produce a consensus prediction.

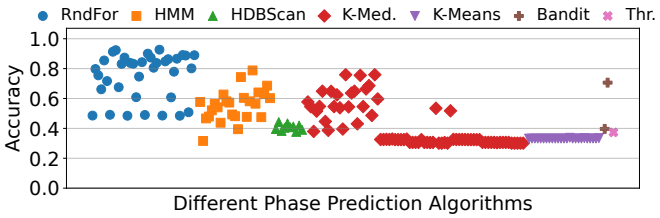

Figure 5. Accuracy of Different Phase Predictors.

Each of these approaches are provided with *application agnostic* features (*e.g.* IPC, cache, TLB, and branch miss rates), allowing the predictor to be entirely transparent to the application. When evaluating each approach, we consider three main requirements: *(1)* high accuracy in finding correlations between phases and features, *(2)* high tolerance against noise from a wide array of features, and *(3)* low cost prediction that does not stall execution of the workload.

Before proceeding, we note that requirement *(3)* disallows more complex AI approaches (*e.g.* transformer models), as it is infeasible to implement these methods efficiently.

Figure 5 shows the accuracy of the various approaches when applied on an Ad-serving applications [26] that displays fine-grained requests. We see that all approaches besides Random Forests show accuracies that fall below 80%. Clustering approaches suffer from noise across features that are important in one cluster but not to others. Multi-Armed Bandits is distribution-driven and thus fails to capture relationships between phases and features. Contextual Bandits also falls short to noise across features.

Hence, we select the *Random Forests* (RF) for our Predictor Module. To avoid software overheads (*e.g.*, interrupting the CPU), we implement a RF predictor in hardware, where prior work has shown it can be efficiently implemented [7].

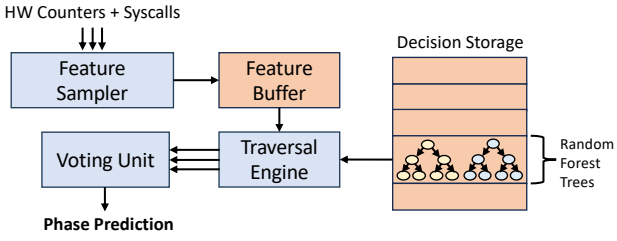

Figure 6. Microarchitecture of the Random Forest Predictor.

Figure 6 shows the microarchitecture of the Random Forest-based Predictor Module. The decision trees are trained offline using labeled data collected by sampling various microbenchmarks [28] and are stored in a compact SRAM structure called the *Decision Storage*. Each entry encodes a node, which compactly stores a feature ID (*e.g.* a label for each feature including IPC), threshold value, and the left and right child nodes. At runtime, the *feature sampler* collects data from counters and stores them in a *feature buffer*. At the end of every epoch ($100\mu s$), the *traversal engine* reads both the collected features and each pre-trained decision trees. Starting at the root node, it iteratively compares the selected feature value with the stored threshold, until it reaches a leaf. These results are aggregated by the *voting unit*, which outputs a single majority consensus prediction across the decision trees.

Overhead. Based on sensitivity studies, we use 15 decision trees of depth 5 that read from a collection of 15 features. Thus, we keep the storage requirement per predictor to less than $8KB$ (a conservative measurement) and cost of inference to 75 comparisons, or less than 100 cycles. Importantly, all of this computation happens off the application's critical path.

4 Heterogeneous Server Design

The Predictor Module allows the system to know the exact resource demands of the executing task. To exploit this, we introduce a heterogeneous-core server that provides the optimal execution environment for each phase. Figure 7 illustrates how we envision this proposal in hardware. The server is decomposed into a set of heterogeneous chiplets, each tuned to a specific execution class. A *compute-optimized* chiplet uses high-frequency, wide-issue cores. A *memory-optimized* chiplet pairs medium-sized cores with higher memory bandwidth (e.g., via additional or higher-performance memory channels [4]). A *network-optimized* chiplet combines medium-sized cores with integrated NICs to reduce networking latency and avoid PCIe overhead. Finally, an *efficiency-optimized* chiplet uses smaller, low-power cores to handle light-load phases with minimal energy overhead.

By matching core complexity and local resources to the computational intensity and bottlenecks of each execution class, the design delivers specialization and efficiency while still presenting a clean, general-purpose server abstraction. Moreover, as each core is now provisioned only with the resources it truly needs, rather than all possible capabilities, its area shrinks, enabling many more specialized cores to fit within the same silicon budget.

5 Proposed Evaluation and Related Work

Evaluation. We plan to evaluate our work with extensive simulation with the SST simulator [31] connected to DRAMSim3 [23]. We will use QEMU [35] to collect both user- and kernel-space instruction traces to provide to the simulator. We focus on two main metrics: tail latency and performance

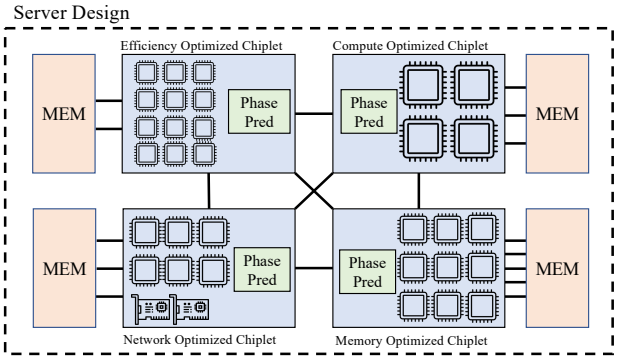

Figure 7. High-level architecture of proposed server

per watt, where performance is the maximum queries per second (QPS) rate for a given service-level objective (SLO). For the evaluation workload, we use DCPerf [39], an open-source benchmark suite that emulates production-grade datacenter workloads. Early results are shown in Appendix B.

Related Work. Prior work on phase detection has largely focused on traditional single- and multi-threaded applications, using instruction working sets, control-flow patterns, and event-counter behavior to identify coarse-grained phases and drive optimizations in caching, power management, and memory hierarchies [6, 25, 33, 34]). Separately, heterogeneous-core designs (e.g., ARM big.LITTLE) exploit perf-power tradeoffs by scheduling threads across fast and slow cores, via OS policies or hardware heuristics that operate at relatively coarse granularity and are not explicitly phase-aware [3, 9, 21, 30]. More recent datacenter-focused architectures and "datacenter tax" accelerators improve efficiency by specializing cores or offloading recurring functions, but they generally target particular bottlenecks rather than dynamically adapting to rapidly shifting execution behavior [1, 5, 10, 12, 14, 15, 17–20, 29, 36–38, 43]. In contrast, targets datacenter workloads with fine-grained, ms-scale phase changes, providing application-transparent, online in-hardware phase prediction and migration across heterogeneous chiplets to improve *Perf/Watt*, and is complementary to both specialized core designs and accelerator-based approaches.

6 Conclusion and Future Work

This paper introduced an AI-driven solution that aligns hardware specialization with fine-grained execution phases of datacenter workloads. Looking further, we envision a complete system that includes a benefit-aware migration runtime that takes in the predicted phases and load on a given resource to dynamically determine whether to move a phase to a new core. Moreover, the system can be extended to include more sources of hardware heterogeneity, by incorporating chiplets with a diverse set of accelerators (*e.g.* GPU, domain-specific accelerators for tax operations). We believe our work will pave the road for AI-driven heterogeneous resource management in the cloud, pushing the boundaries of performance and efficiency in modern datacenters.

References

[1] Bulent Abali, Bart Blaner, John Reilly, Matthias Klein, Ashutosh Mishra, Craig B. Agricola, Bedri Sendir, Alper Buyuktosunoglu, Christian Jacobi, William J. Starke, Haren Myneni, and Charlie Wang. 2020. Data Compression Accelerator on IBM POWER9 and z15 Processors : Industrial Product. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA'20)*.

[2] Amazon. 2025. What are Microservices? https://aws.amazon.com/microservices/.

[3] ARM. 2025. Processing Architecture for Power Efficiency and Performance. https://www.arm.com/technologies/big-little

[4] Evgeny Bolotin, David Nellans, Oreste Villa, Mike O'Connor, Alex Ramirez, and Stephen W. Keckler. 2015. Designing Efficient Heterogeneous Memory Architectures. *IEEE Micro* 35, 4 (2015), 60–68. doi:10.1109/MM.2015.72

[5] Junehyuk Boo, Yujin Chung, Eunjin Baek, Seongmin Na, Changsu Kim, and Jangwoo Kim. 2023. F4T: A Fast and Flexible FPGA-based Full-stack TCP Acceleration Framework. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA'23)*.

[6] A.S. Dhodapkar and J.E. Smith. 2003. Comparing program phase detection techniques. In *Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'03)*. doi:10.1109/MICRO.2003.1253197

[7] Furkan Eris, Marcia Louis, Kubra Eris, José Abellán, and Ajay Joshi. 2022. Puppeteer: A Random Forest Based Manager for Hardware Prefetchers Across the Memory Hierarchy. *ACM Trans. Archit. Code Optim.* 20, 1, Article 19 (Dec. 2022), 25 pages.

[8] Google Cloud. 2025. What is Microservices Architecture? https://cloud.google.com/learn/what-is-microservices-architecture.

[9] Vishal Gupta, Ripal Nathuji, and Karsten Schwan. 2011. An analysis of power reduction in datacenters using heterogeneous chip multiprocessors. *SIGMETRICS Perform. Eval. Rev.* 39, 3 (Dec. 2011). doi:10.1145/2160803.2160867

[10] Xiaokang Hu, Changzheng Wei, Jian Li, Brian Will, Ping Yu, Lu Gong, and Haibing Guan. 2019. QTLS: high-performance TLS asynchronous offload framework with Intel® QuickAssist technology. In *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming (PPoPP'19)*.

[11] Darby Huye, Yuri Shkuro, and Raja R. Sambasivan. 2023. Lifting the veil on Meta's microservice architecture: Analyses of topology and request workflows. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC'23)*.

[12] Stephen Ibanez, Alex Mallery, Serhat Arslan, Theo Jepsen, Muhammad Shahbaz, Changhoon Kim, and Nick McKeown. 2021. The nanoPU: A Nanosecond Network Stack for Datacenters. In *Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI'21)*.

[13] Intel. 2024. Intel RDT Software Package. https://github.com/intel/intel-cmt-cat/tree/master.

[14] Intel. 2024. Intel® QAT: Performance, Scale, and Efficiency. https://www.intel.com/content/www/us/en/products/docs/accelerator-engines/what-is-intel-qat.html.

[15] Intel. 2025. Intel® Dynamic Load Balancer. https://www.intel.com/content/www/us/en/download/686372/intel-dynamic-load-balancer.html.

[16] Intel. 2025. Intel® Xeon® Gold 5512U Processor. https://www.intel.com/content/www/us/en/products/sku/237565/intel-xeon-gold-5512u-processor-52-5m-cache-2-10-ghz/specifications.html.

[17] Jaeyoung Jang, Sung Jun Jung, Sunmin Jeong, Jun Heo, Hoon Shin, Tae Jun Ham, and Jae W. Lee. 2020. A Specialized Architecture for Object Serialization with Applications to Big Data Analytics. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA'20)*.

[18] Keon Jang, Sangjin Han, Seungyeop Han, Sue Moon, and KyoungSoo Park. 2011. SSLShader: Cheap SSL Acceleration with Commodity Processors. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI'11)*.

[19] Sagar Karandikar, Chris Leary, Chris Kennelly, Jerry Zhao, Dinesh Parimi, Borivoje Nikolic, Krste Asanovic, and Parthasarathy Ranganathan. 2021. A Hardware Accelerator for Protocol Buffers. In *Proceedings of the 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'21)*.

[20] Sagar Karandikar, Aniruddha N. Udipi, Junsun Choi, Joonho Whangbo, Jerry Zhao, Svilen Kanev, Edwin Lim, Jyrki Alakuijala, Vrishab Madduri, Yakun Sophia Shao, Borivoje Nikolic, Krste Asanovic, and Parthasarathy Ranganathan. 2023. CDPU: Co-designing Compression and Decompression Processing Units for Hyperscale Systems. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA'23)*.

[21] R. Kumar, D.M. Tullsen, P. Ranganathan, N.P. Jouppi, and K.I. Farkas. 2004. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In *Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA'04)*. doi:10.1109/ISCA.2004.1310764

[22] J. Lau, S. Schoenmackers, and B. Calder. 2005. Transition phase classification and prediction. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture (HPCA'05)*. doi:10.1109/HPCA.2005.39

[23] Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, and Bruce Jacob. 2020. DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator. *IEEE Computer Architecture Letters* 19, 2 (2020), 106–109. doi:10.1109/LCA.2020.2973991

[24] Shutian Luo, Huanle Xu, Chengzhi Lu, Kejiang Ye, Guoyao Xu, Liping Zhang, Yu Ding, Jian He, and Chengzhong Xu. 2021. Characterizing Microservice Dependency and Performance: Alibaba Trace Analysis. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC'21)*.

[25] Yangchun Luo, Venkatesan Packirisamy, Wei-Chung Hsu, and Antonia Zhai. 2010. Energy efficient speculative threads: dynamic thread allocation in Same-ISA heterogeneous multicore systems. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques (PACT'10)*. doi:10.1145/1854273.1854329

[26] Meta. 2025. Adsim. https://github.com/facebookresearch/DCPerf/tree/v2-beta/packages/adsim.

[27] Meta. 2025. Mediawiki. https://github.com/joshk411/DCPerf/tree/main/packages/mediawiki.

[28] Meta. 2025. WDLBench. https://github.com/facebookresearch/DCPerf/tree/main/packages/wdl_bench.

[29] Amirhossein Mirhosseini, Akshitha Sriraman, and Thomas F. Wenisch. 2019. Enhancing Server Efficiency in the Face of Killer Microseconds. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'19)*.

[30] Shaolei Ren, Yuxiong He, Sameh Elnikety, and Kathryn S. McKinley. 2013. Exploiting Processor Heterogeneity in Interactive Services. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC'13)*.

[31] Arun F. Rodrigues, K. Scott Hemmert, Brian W. Barrett, Chad D. Kersey, Ron A. Oldfield, M. Weston, R. Risen, J. Cook, Paul Rosenfeld, E. CooperBalls, and Bruce L. Jacob. 2011. The structural simulation toolkit. *SIGMETRICS Performance Evaluation Reviews* 38, 4 (2011).

[32] Amanzhol Salykov. 2024. Advanced Matrix Multiplication Optimization on Modern Multi-Core Processors . https://salykova.github.io/gemm-cpu.

[33] Xipeng Shen, Yutao Zhong, and Chen Ding. 2004. Locality phase prediction. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'04)*. doi:10.1145/1024393.1024414

[34] Timothy Sherwood, Suleyman Sair, and Brad Calder. 2003. Phase tracking and prediction. In *Proceedings of the 30th Annual International Symposium on Computer Architecture (ISCA'03)*. doi:10.1145/859618.859657

[35] Software Freedom Conservancy. 2025. QEMU: A generic and open source machine emulator and virtualizer. https://www.qemu.org/

[36] Jovan Stojkovic, Esha Choukse, Enrique Saurez, Íñigo Goiri, and Josep Torrellas. 2024. Mosaic: Harnessing the Micro-Architectural Resources of Servers in Serverless Environments. In *Proceedings of the 57th IEEE/ACM International Symposium on Microarchitecture (MICRO'24)*.

[37] Jovan Stojkovic, Chunao Liu, Muhammad Shahbaz, and Josep Torrellas. 2023. µManycore: A Cloud-Native CPU for Tail at Scale. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA'23)*.

[38] Jovan Stojkovic, Chunao Liu, Muhammad Shahbaz, and Josep Torrellas. 2025. HardHarvest: Hardware-Supported Core Harvesting for Microservices. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA'25)*.

[39] Wei Su, Abhishek Dhanotia, Carlos Torres, Jayneel Gandhi, Neha Gholkar, Shobhit Kanaujia, Maxim Naumov, Kalyan Subramanian, Valentin Andrei, Yifan Yuan, and Chunqiang Tang. 2025. DCPerf: An Open-Source, Battle-Tested Performance Benchmark Suite for Datacenter Workloads. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture (ISCA'25)*.

[40] Think Software. 2021. Microservices Architecture of Twitter Service. https://thinksoftware.medium.com/design-twitter-microservices-architecture-of-twitter-service-996ddd68e1ca.

[41] Uber. 2020. Introducing Domain-Oriented Microservice Architecture. https://www.uber.com/blog/microservice-architecture/.

[42] Ketan Varshneya. 2021. Understanding design of microservices architecture at Netflix. https://www.techaheadcorp.com/blog/design-of-microservices-architecture-at-netflix/.

[43] Adam Wolnikowski, Stephen Ibanez, Jonathan Stone, Changhoon Kim, Rajit Manohar, and Robert Soulé. 2021. Zerializer: towards zero-copy serialization. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS'21)*.

[44] Weihua Zhang, Jiaxin Li, Yi Li, and Haibo Chen. 2015. Multilevel Phase Analysis. *ACM Trans. Embed. Comput. Syst.* 14, 2, Article 31 (March 2015). doi:10.1145/2629594

A Additional Characterization Figures

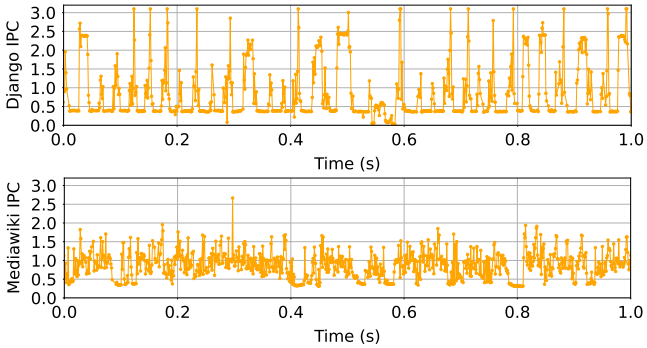

Figure 8. IPC over time for additional DCPerf workloads: *Django* and *Mediawiki*.

B Early Evaluation Results

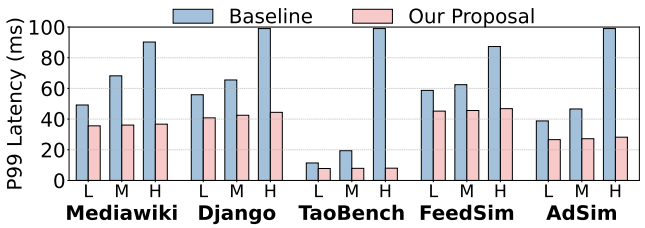

Figure 9. P99 tail latency across DCPerf applications at different loads (Low, Medium, and High, corresponding to the 25%, 50%, and 75% of the baseline server CPU utilization).