Microarchitectural comparison and in-core modeling of state-of-the-art CPUs: Grace, Sapphire Rapids, and Genoa

Jan Laukemann
Erlangen National High Performance Computing Center
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany
jan.laukemann@fau.de

Georg Hager
Erlangen National High Performance Computing Center
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany
georg.hager@fau.de

Gerhard Wellein
Erlangen National High Performance Computing Center
Erlangen, Germany
gerhard.wellein@fau.de

***Abstract*—With Nvidia's release of the Grace Superchip, all three big semiconductor companies in HPC (AMD, Intel, Nvidia) are currently competing in the race for the best CPU. In this work we analyze the performance of these state-of-the-art CPUs and create an accurate in-core performance model for their microarchitectures Zen 4, Golden Cove, and Neoverse V2, extending the Open Source Architecture Code Analyzer (OSACA) tool and comparing it with LLVM-MCA. Starting from the peculiarities and up- and downsides of a single core, we extend our comparison by a variety of microbenchmarks and the capabilities of a full node. The "write-allocate (WA) evasion" feature, which can automatically reduce the memory traffic caused by write misses, receives special attention; we show that the Grace Superchip has a next-to-optimal implementation of WA evasion, and that the only way to avoid write allocates on Zen 4 is the explicit use of non-temporal stores.**

***Index Terms*—Intel Sapphire Rapids, NVIDIA Grace CPU Superchip, AMD Genoa, Golden Cove, Neoverse V2, Zen 4, in-core, performance analysis, performance modeling**

I. Introduction

A. Motivation and related work

The Grace Hopper Superchip as well as the Grace CPU Superchip (GCS) mark the first HPC and data center systems by Nvidia with their own CPU, based on Arm's Neoverse V2 design. One chip comprises 72 cores running at 3.4 GHz all within one ccNUMA domain. With this approach, Nvidia wants to catch up with the x86 competition and offer a full solution covering both accelerators (i.e., GPGPUs) and hosts (i.e., CPUs). In this work, we analyze the Nvidia Grace CPU Superchip, compare its performance to the state-of-the-art competitor x86 CPUs Intel Sapphire Rapids (SPR) and AMD Genoa, and provide an in-core performance model for all three microarchitectures for lower-bound runtime prediction that can be used as part of holistic performance models such as Roofline [1], leading to valuable insights into performance bottlenecks of these new CPU architectures.

Although previous work comparing the NVIDIA GCS with competitive x86 and Arm CPUs exists [2], [3], it focused mainly on application performance and ignored microarchitectural details.

While there is a wide range of tools for runtime prediction of machine code via static code analysis, we choose to use the Open Source Architecture Code Analyzer (OSACA) [4], [5] as it provides the user with the possibility of adding new microarchitectures into the existing framework relatively easily. Moreover, it stands out as it is not restricted to a single ISA (compared to uiCA [6] and Facile [7]), proved to provide superior accuracy (compared to llvm-mca [8]) for already existing performance models, focuses on performance prediction rather than the code quality (compared to the Code Quality Analyzer (CQA) [9]), and provides a white-box model for more insight for the user (compared to AI-based tools like Ithemal [10] and GRANITE [11]). In [12] a thorough analysis on write-allocate evasion for Intel Ice Lake and Sapphire Rapids CPUs was conducted. In this work we broaden the view to AMD Genoa and the NVIDIA Grace CPU Superchip.

B. Brief overview of the in-core port models

When thinking about the performance of a single CPU core, we assume what is widely known as the *port model*: Each instruction, optionally split into one or more *micro-ops* (μ-ops), gets assigned to and processed by *execution units* (EUs) and may even require multiple EUs (e.g., to load data and do an arithmetic computation on the loaded value). On the other hand, one EU might exist multiple times and can thus increase the instruction throughput via out-of-order (OoO) execution; e.g., two FMA units that can be accessed in parallel double the throughput for this instruction type. One or more EUs are grouped behind a *port* as seen by

979-8-3503-5554-3/24/$31.00 ©2024 IEEE
DOI 10.1109/SCW63240.2024.00181

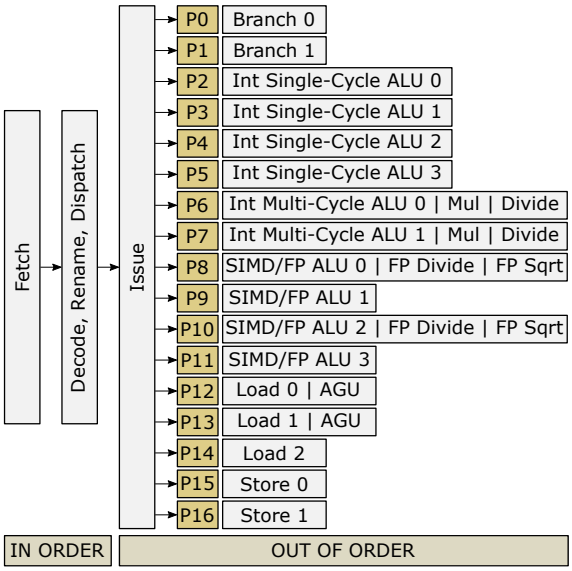

Fig. 1. Arm Neoverse V2 core block diagram and port model, compiled from Arm's Software Optimization Guide [13]. The port numbering ("`P[0-9]+`") is chosen by the authors and is not part of the official documentation.

the scheduler, i.e., for each port and each cycle, one μ-op can be issued (with a global maximum number of μ-ops issued per cycle). Figure 1 shows the port model of the Neoverse V2 microarchitecture, used in Nvidia's Grace CPU. While the width of the SVE registers is relatively small (128 bit), there is considerable instruction level parallelism (ILP) in available ports with similar execution units. As often seen in modern OoO-architectures, the execution of instructions including floating-point data is separated from the execution of integer data. For more information about the idea of port models, see [4].

C. Testbed and experimental methodology

All experiments were carried out on dedicated servers in our test cluster: A two-socket Nvidia Grace CPU Superchip, a two-socket Intel Xeon Platinum 8470, and a two-socket AMD EPYC 9684X system. The specific hardware features are listed in Table I. For compilation we used GCC 12.1, the oneAPI 2023.2 compiler framework and LLVM Clang 17.0.6 for the x86 machines and the Arm C Compiler 23.10 (based on LLVM 17) and GCC 13.2 for the Grace server. For cycle-accurate measurements we set the clock frequency to the corresponding base frequency using SLURM [14] if possible, i.e., 2.0 GHz for SPR and 2.55 GHz for Genoa. While Grace does not allow frequency fixing, we could not observe any frequency change running our benchmarks and validated the clock frequency of all runs with hardware performance counters using LIKWID [15] 5.3.0 [16][1]. This tool was also used for all other hardware performance counter measurements. To validate our in-core performance models, we used the OSACA [4], [5] tool in version 0.5.3 including our own extensions (which will be part of the next release) for supporting the microarchitectures in this paper. Furthermore, the LLVM Machine Code Analyzer (LLVM-MCA) [8] used for comparing the accuracy of our model.

[1]As there is no official release of the LIKWID software with Nvidia Grace support and only limited support for Genoa, we used the development versions from PR585 and PR618, respectively.

II. Architectural analysis

A. Hardware model and instruction performance

Since a port model visualization of all three microarchitectures as shown in Figure 1 would go beyond the scope of this work, we show the key aspects of the three cores in Table II.

While Golden Cove and Zen 4 have approximately the same number of ports (12 and 13, respectively), the Neoverse V2 stands out with its 17 ports, fully offloading any non-floating-point operations to other ports and providing a high ILP. As a downside, even though the core supports the SVE vector extension for width-agnostic vector registers, the maximum register width is 128 bit, which is only a fourth of Golden Cove's 512 bit registers. This leads to the expectation that the Golden Cove architecture can show its strength when executing highly vectorized code while the Neoverse V2 shines with code that is hard to vectorize and has many scalar instructions, as often seen in data center and AI workloads. The Zen 4 meets the two extremes in the middle with 256-bit registers and slightly more ILP than Golden Cove. Even though Zen 4 supports the AVX-512 extensions, their execution is split into 2×256 bit packets. While a comparison of the sustained peak memory bandwidths heavily depends on the built-in memory type and the number of DIMMs and would not represent a fair competition, we can compare them with the theoretical maximum and state that Genoa only achieves 81% of its theoretical memory bandwidth peak, while GCS and SPR reach 87% and 90%, respectively.

While there is some documentation on the microarchitectures' backends [13], [17], [18], the information often is incomplete or insufficient to build a useful performance model. Therefore, we write microbenchmarks with various benchmark tools [19], [20] for every interesting instruction to obtain its throughput, latency, and port occupation. For the latter, it is often necessary to interleave the instruction with known instructions to infer the potential ports of execution. While each model comprises hundreds of entries of individual combinations of assembly instructions and operands, we show the throughput and latency for some of the most important double-precision instructions in Table III. While the "VEC" rows refer to the common SIMD instructions in x86 and AArch64, such as `v[add, mul, fmadd213, div]pd` and `f[add, mul, mla, div]`, respectively, the "scalar" rows refer to scalar assembly instructions from recent ISA extensions working on vector registers, such as `v[add, mul, fmadd213, div]sd`, `f[add, mul, div]`, and `fmadd`. However, these numbers often fit the throughput and latency of similar or equivalent instructions from the respective ISA, e.g., when including a negate, a subtract instead of add, additional masking, or instruction aliases.

[2]GCS's 6 Int ports comprise 2 multi-cycle + 4 single-cycle ports.

TABLE I
COMPARISON OF THE CORE FEATURES OF THE GRACE CPU SUPERCHIP (GCS), THE INTEL XEON PLATINUM 8470 (SPR), AND THE AMD EPYC 9684X (GENOA). FOR ALL SERVERS, THE L1 AND L2 CACHE ARE EXCLUSIVE CACHES PER CORE, WHILE THE L3 IS SHARED WITHIN ONE CHIP.

	Nvidia Grace Superchip "GCS"	Intel Xeon Platinum 8470 "SPR"	AMD EPYC 9684X "Genoa"
Cores	72	52	96
Frequency (max/base)	3.4 GHz / 3.4 GHz	3.8 GHz / 2.0 GHz	3.7 GHz / 2.55 GHz
Theor. DP Peak	3.92 Tflop/s	6.32 Tflop/s	8.52 TFlop/s
Achiev. DP Peak	3.82 Tflop/s	3.49 Tflop/s	5.1 TFlop/s
TDP	250 W	350 W	400 W
Cache size (L1/L2/L3)	64 KB / 1 MB / 114 MB	48 KB / 2 MB / 105 MB	32 KB / 1 MB / 1152 MB
Main memory	240 GB LPDDR5X	512 GB DDR5	384 GB DDR5
ccNUMA domains	1	4 (SNC-mode)	4 (NPS=4)
Max. mem bandwidth (theor. / measured)	546 GB/s / 467 GB/s	307 GB/s / 273 GB/s	461 GB/s / 375 GB/s

TABLE II
COMPARISON OF THE IN-CORE FEATURES AND PORT MODELS FOR THE GCS, SPR, AND GENOA CORES.

	GCS (Neoverse V2)	SPR (Golden Cove)	Genoa (Zen 4)
Number of ports	17	12	13
SIMD-width	16 B	64 B	32 B
Int units	6^2	5	4
FP vector units	4	3	4
Loads/cy	3×128 B	2×512 B	2×256 B
Stores/cy	2×128 B	2×256 B	1×256 B

The Golden Cove architecture shows the highest throughput for all shown vector instructions due to its large register width. The Neoverse V2 can demonstrate its strength for scalar instructions due to the large ILP. A single Zen 4 core is slower or breaks even in terms of throughput for all instructions shown; however, a full chip comprises 96 cores while an SPR chip comes only with 52 cores. Therefore, if an application allows a high parallelism on the node level, e.g., through OpenMP, the overall throughput of the Genoa system might come out first, as shown in the artificial peak FLOP benchmark used in Table I. When looking at the latencies of the investigated instructions, one can clearly observe the superiority of the Neoverse V2 which shows a lower or even latency for every single instruction in Table II. Thus, arithmetic-heavy latency-bound codes such as iterative solvers using the Gauss-Seidel method [21] could benefit from running on a Grace CPU Superchip compared to the two competitors. Especially Intel seems to trade off their high throughput performance against a relatively high instruction latency, even though they managed to decrease the ADD latency by half compared to the predecessor Ice Lake microarchitecture.

B. Clock frequency throttling

Although wide registers can provide a good out-of-the-box speedup when using vectorization on a single core, it is a well-known problem for Intel to require the cores to throttle down for AVX-512-heavy code and when using multiple cores due to thermal constraints. Therefore we analyzed the sustained clock frequency for arithmetic-heavy codes while scaling across a socket on all systems (see Fig. 2). Each benchmark ran for several minutes and the clock frequency of all active cores

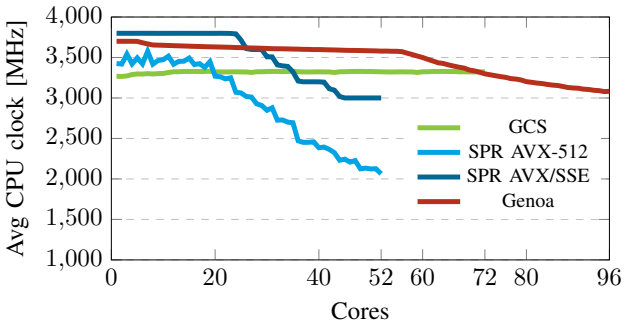

Fig. 2. Sustained CPU clock frequency for arithmetic-heavy code on GCS, SPR, and Genoa across one chip. If no ISA extension is specified, the architecture could sustain the same frequency for all supported ISA extensions.

was tracked using hardware performance counters. While SPR shows a different behavior right from the start for AVX-512-heavy code, the sustained frequency for the GCS and Genoa did not change across ISA extensions. Both SPR and Genoa eventually fall down to a frequency of 2.0 GHz and 3.1 GHz for AVX-512-heavy code, which results in 53% and 84% of their respective single-core turbo limit, even though SPR manages to sustain a frequency of 3.0 GHz for the case of AVX- or SSE-heavy code (78% of Turbo). The Nvidia GCS exhibits a constant frequency of 3.4 GHz (the base frequency) throughout the whole socket. Therefore, for highly parallel arithmetic-heavy code, one might see better performance on GCS compared to SPR despite even for throughput- or latency-bound code due to a 1.7× higher sustained clock frequency.

C. Performance modeling with OSACA

The individual measurements as shown in Table III can be incorporated with the specific port occupations into an in-core performance model that can be used for optimistic runtime prediction or as a building block for node-wide performance models (e.g., a more realistic horizontal ceiling in the Roofline Model [1] or the in-core component of the Execution-Cache-Memory (ECM) Model [22]). The Open Source Architecture Code Analyzer (OSACA) provides such an in-core model based on static analysis of assembly code without compiling or running it. The tool delivers three different predictions: a) an optimistic throughput analysis considering the port

TABLE III

THROUGHPUT AND LATENCY FOR SOME DOUBLE-PRECISION INSTRUCTIONS ON GCS, SPR, AND GENOA. IF MULTIPLE VALUES FOR ONE INSTRUCTION WERE APPLICABLE, E.G., DUE TO DIFFERENT PERFORMANCE FOR DIFFERENT VECTOR WIDTHS, THE BEST PERFORMANCE (I.E., HIGHEST THROUGHPUT, LOWEST LATENCY) WAS SELECTED. NOTE THAT THE THROUGHPUT OF THE "GATHER" AS A LOAD INSTRUCTION IS GIVEN IN "CACHE LINES PER CYCLE" WHILE THE REST IS EVALUATED IN DOUBLE PRECISION ELEMENTS PER CYCLE.

	GCS (Neoverse V2)	SPR (Golden Cove)	Genoa (Zen 4)	GCS (Neoverse V2)	SPR (Golden Cove)	Genoa (Zen 4)
Instruction	Throughput [DP elements / cy]			Latency [cy]		
gather [CL/cy]	¼	⅓	⅛	9	20	13
VEC ADD	8	16	8	2	2	3
VEC MUL	8	16	8	3	4	3
VEC FMA	8	16	8	4	4	4
VEC FP Div	0.67	0.5	0.8	7	14	13
Scalar ADD	4	2	2	2	2	3
Scalar MUL	4	2	2	3	4	3
Scalar FMA	4	2	2	4	5	4
Scalar Div	0.4	0.25	0.2	12	14	13

pressure on each individual port (i.e., the reciprocal throughput of all instructions executed on a port considering perfect scheduling), b) a loop-carried-dependency analysis, trying to detect dependency chains across loop iterations and showing the overall runtime (i.e., latency) of each one, and c) a critical path analysis, detecting the longest dependency chain within one loop, which insights on potential slowdowns due to no full overlap of dependency chains for a very long critical path or a small number of loop iterations. As a runtime prediction, we use the maximum number of cycles (i.e., slowest runtime) out of the throughput analysis and the loop-carried-dependency analysis as suggested in previous work on this tool [23]. For validation of our models we used 13 streaming microbenchmarks (Jacobi [2D 5-point|3D 27-point|3D 7-point|3D 11-point] stencil, ADD, COPY, Gauss-Seidel 2D 5-point stencil, π-computation by integration, INIT, Schönauer Triad, Sum reduction, STREAM Triad [24], UPDATE), compiled with different compilers (Armclang, GCC, oneAPI, and Clang) and different optimization flags (`-O1`, `-O2`, `-O3`, and `-Ofast`), resulting in 416 tests and 290 unique assembly representations.

Figure 3 (based on graphs in [25]) shows histograms of the relative prediction error (RPE) for the kernels with our models of the investigated microarchitectures incorporated into OSACA versus the LLVM performance models in LLVM-MCA. Each bucket marks a range of 10% relative error; bars right of the red dotted zero line indicate a prediction faster than the actual measurement while bars left of the line indicate a slower prediction. The bucket in the very left collects all predictions larger than -1.0 (i.e., off by more than a factor of 2). As we aim to provide a lower-bound estimate, we prefer to see all errors on the right of the zero line. Except for a few versions of the Gauss-Seidel kernel for the Neoverse V2, where OSACA (correctly) predicts a register dependency that the CPU can overcome by register renaming, and the π kernel for Zen 4, where our model assumes a lower throughput for the scalar divide than we measure, this is the case for all other tests (96%) with our performance model. There is one kernel predicted incorrectly by more than a factor of 2, and 37% (44%) are predicted accurately with a positive RPE of less than

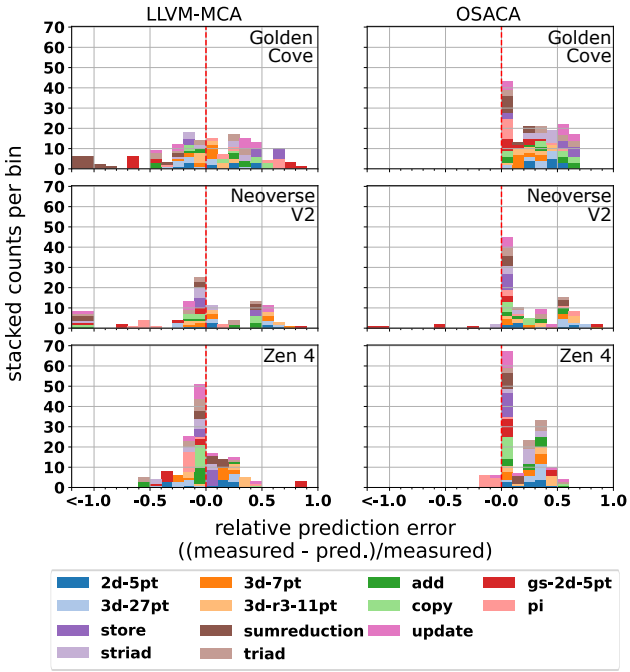

Fig. 3. Relative prediction error of 416 test blocks for LLVM-MCA and OSACA. Bars right of the red dotted line indicate a prediction faster than the actual measurement while bars left of the line indicate a slower prediction.

10% (20%). The LLVM-MCA model, however, predicts 75% of the test kernels slower than the actual measurements, with 14 measurements being off by more than a factor of 2. Only 10% (16%) are predicted correctly with positive a RPE of less than 10% (20%), although this value increases to 32% (48%) when considering the 10% (20%) bucket on the negative side of the zero line. The average RPE of only the under-predictions (i.e., right-hand-side errors) of our model in OSACA shows a smaller error for Golden Cove, V2, and Zen 4 with 24%, 30%, and 18% versus the LLVM model showing 38%, 34%, and 20%. When looking at the global (i.e., absolute) RPE, our model still performs better for Golden Cove (30% vs 35%) and V2 (26% vs 52%), and is slightly worse for Zen 4 than the

LLVM-based model (18% vs 16%).

D. Write-Allocate Evasion

Looking beyond the CPU core towards data movement in the memory hierarchy, one interesting feature that has entered x86 processors with the Intel Ice Lake generation is the automatic evasion of write-allocate (WA) transfers from memory. Write-allocate usually occurs in cache-based architectures when a standard store operation from a register to memory causes a write miss: Since the core can only communicate with its L1 cache, the cache line must be read from memory before it can be modified and then (later) written back. This extra data traffic can impact the performance and clutter the cache with data that may not be needed soon. *Cache line claim* and *non-temporal stores* are two ways to avoid write-allocates. Both can be supported by special instructions that claim a cache line in the cache without reading it first (available on some Arm CPUs) or write data to memory through a special write-combine buffer outside the normal cache hierarchy (available on Arm and x86 CPUs). Cache line claim can also be automatic if a core is able to detect that a cache line will be overwritten entirely. This feature has been supported for a long time by many Arm CPUs (including, e.g., the Marvell ThunderX2 and GCS) and by Intel server chips starting with the Ice Lake family, where Intel termed it *SpecI2M* [26].

In order to fathom the ability of the CPUs under investigation to employ automatic and explicit WA evasion, we run a simple store-only (array initialization) benchmark, measure the actual memory data traffic (which includes write-allocates), and divide it by the amount of stored data. With perfect WA evasion in place, this ratio should be equal to one. It should be equal to two if the full WA transfers apply. On Intel CPUs it was shown previously [12], [27] that the efficiency of SpecI2M depends crucially on the saturation of the memory interface: Only if a significant fraction of the maximum memory bandwidth is utilized will the WA evasion mechanism kick in. Figure 4 shows the results of the store benchmark with respect to the number of cores utilized for all three CPUs. In case of SPR and Genoa, we have added a variant with non-temporal (NT) stores for reference; ideally, NT stores should eliminate the WA transfers entirely.

The results show that only GCS is able to completely avoid WA transfers automatically in this simple scenario (solid green line). The SpecI2M mechanism in SPR can only reduce write-allocates by up to 25% and only kicks in when a large part of the 13 cores on a ccNUMA domain is utilized (solid blue line). The only way to WA evasion on Genoa is via non-temporal stores, which works perfectly, however (dotted dark red line vs. solid red line). Finally, on SPR even the non-temporal stores are not 100% effective, and there is a residual 10% of WA traffic except for very small core counts (dotted blue line).

III. Conclusion

Via a thorough in-core analysis of the Nvidia Grace CPU Superchip, the Intel Sapphire Rapids, and the AMD Genoa CPU, we showed peculiarities of their microarchitectures Neoverse V2, Golden Cove, and Zen 4, respectively, established an in-core performance model for each of them, and applied it to simple streaming kernels. We showed that the models, incorporated in the Open Source Architecture Code Analyzer (OSACA), yield more accurate lower bounds for in-core runtime than the existing LLVM-MCA model for a comprehensive set of microbenchmarks. Furthermore, we investigated the node-level capabilities of the HPC servers such as the sustained CPU clock frequencies and the memory bandwidth and focused on implicit and explicit Write-Allocate evasion techniques. In future work, we plan to continue these investigations by applying our in-core model to a node-wide performance model such as the Execution-Cache-Memory (ECM) model and study real-life applications on a larger scale.

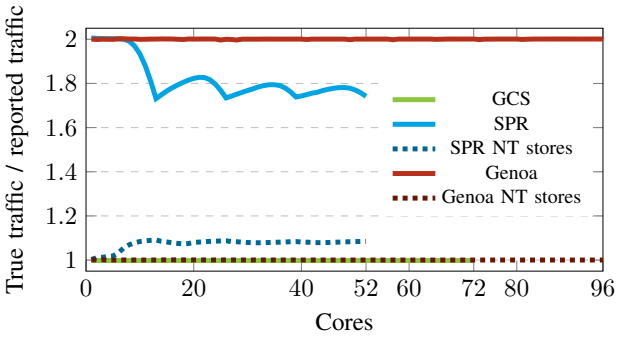

Fig. 4. Ratio of actual memory traffic to stored data volume vs. number of cores for a store-only benchmark loop (working set 40 GB). A value of 1.0 indicates perfect WA evasion, while a value uf 2.0 indicates full WA traffic. The variants labeled "NT stores" use non-temporal store instructions, while the others use standard stores.

Acknowledgment

This work was partly funded by BMBF through the DAREXA-F project.

References

[1] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, pp. 65–76, 2009, DOI: 10.1145/1498765.1498785.

[2] F. Banchelli, J. Vinyals-Ylla-Catala, J. Pocurull, M. Clascà, K. Peiro, F. Spiga, M. Garcia-Gasulla, and F. Mantovani, "NVIDIA Grace Superchip Early Evaluation for HPC Applications," in *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region Workshops*, ser. HPCAsia '24 Workshops. New York, NY, USA: Association for Computing Machinery, 2024, p. 45–54, DOI: 10.1145/3636480.3637284.

[3] N. A. Simakov, M. D. Jones, T. R. Furlani, E. Siegmann, and R. J. Harrison, "First Impressions of the NVIDIA Grace CPU Superchip and NVIDIA Grace Hopper Superchip for Scientific Workloads," in *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region Workshops*, ser. HPCAsia '24 Workshops. New York, NY, USA: Association for Computing Machinery, 2024, p. 36–44, DOI: 10.1145/3636480.3637097.

[4] J. Laukemann, J. Hammer, J. Hofmann, G. Hager, and G. Wellein, "Automated Instruction Stream Throughput Prediction for Intel and AMD Microarchitectures," in *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, 2018, pp. 121–131, DOI: 10.1109/PMBS.2018.8641578.

[5] J. Laukemann, J. Hammer, G. Hager, and G. Wellein, "Automatic Throughput and Critical Path Analysis of x86 and ARM Assembly Kernels," in *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, 2019, pp. 1–6, DOI: 10.1109/PMBS49563.2019.00006.

[6] A. Abel and J. Reineke, "uiCA: Accurate Throughput Prediction of Basic Blocks on Recent Intel Microarchitectures," in *ICS '22: 2022 International Conference on Supercomputing, Virtual Event, USA, June 27-30, 2022*, ser. ICS '22, L. Rauchwerger, K. Cameron, D. S. Nikolopoulos, and D. Pnevmatikatos, Eds. ACM, June 2022, pp. 1–12, DOI: 10.1145/3524059.3532396.

[7] A. Abel, S. Sharma, and J. Reineke, "Facile: Fast, Accurate, and Interpretable Basic-Block Throughput Prediction," in *2023 IEEE International Symposium on Workload Characterization (IISWC)*, 2023, pp. 87–99, DOI: 10.1109/IISWC59245.2023.00023.

[8] LLVM Compiler Infrastructure, "LLVM machine code analyzer," accessed 2024-09-01. [Online]. Available: https://llvm.org/docs/CommandGuide/llvm-mca.html

[9] A. S. Charif-Rubial, E. Oseret, J. Noudohouenou, W. Jalby, and G. Lartigue, "CQA: A code quality analyzer tool at binary level," in *2014 21st International Conference on High Performance Computing (HiPC)*, 2014, pp. 1–10, DOI: 10.1109/HiPC.2014.7116904.

[10] C. Mendis, A. Renda, S. Amarasinghe, and M. Carbin, "Ithemal: Accurate, Portable and Fast Basic Block Throughput Estimation using Deep Neural Networks," in *International Conference on machine learning*. PMLR, 2019, pp. 4505–4515. [Online]. Available: https://proceedings.mlr.press/v97/mendis19a.html

[11] O. Sykora, P. Phothilimthana, C. Mendis, and A. Yazdanbakhsh, "GRANITE: A Graph Neural Network Model for Basic Block Throughput Estimation," in *2022 IEEE International Symposium on Workload Characterization (IISWC)*. Los Alamitos, CA, USA: IEEE Computer Society, nov 2022, pp. 14–26, DOI: 10.1109/IISWC55918.2022.00012.

[12] J. Laukemann, T. Gruber, G. Hager, D. Oryspayev, and G. Wellein, "CloverLeaf on Intel Multi-Core CPUs: A Case Study in Write-Allocate Evasion," in *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2024, pp. 350–360, DOI: 10.1109/IPDPS57955.2024.00038.

[13] Arm Limited, "Arm Neoverse V2 Core Software Optimization Guide," Arm Limited, Tech. Rep., 2022, accessed 2024-09-01. [Online]. Available: https://developer.arm.com/documentation/109898/latest/

[14] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Job Scheduling Strategies for Parallel Processing*, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 44–60, DOI: 10.1007/10968987_3.

[15] T. Roehl, J. Treibig, G. Hager, and G. Wellein, "Overhead Analysis of Performance Counter Measurements," in *43rd International Conference on Parallel Processing Workshops (ICCPW)*, Sept 2014, pp. 176–185, DOI: 10.1109/ICPPW.2014.34.

[16] T. Gruber, J. Eitzinger, G. Hager, and G. Wellein, "Likwid," Nov. 2023. [Online]. Available: https://doi.org/10.5281/zenodo.10105559

[17] *Intel® 64 and IA-32 Architecture Optimization Reference Manual*, Intel Corporation, 1 2023. [Online]. Available: https://software.intel.com/en-us/download/intel-64-and-ia-32-architectures-optimization-reference-manual

[18] A. Abel and J. Reineke, "uops.info: Characterizing Latency, Throughput, and Port Usage of Instructions on Intel Microarchitectures," in *ASPLOS*, ser. ASPLOS '19. New York, NY, USA: ACM, 2019, pp. 673–686, DOI: 10.1145/3297858.3304062.

[19] J. Hammer, G. Hager, and G. Wellein, "OoO Instruction Benchmarking Framework on the Back of Dragons," 2018, SC18 ACM SRC Poster. [Online]. Available: https://sc18.supercomputing.org/proceedings/src_poster/src_poster_pages/spost115.html

[20] J. Hofmann, "ibench - Instruction Benchmarks," 2017. [Online]. Available: https://github.com/RRZE-HPC/ibench

[21] J. Hofmann, C. L. Alappat, G. Hager, D. Fey, and G. Wellein, "Bridging the Architecture Gap: Abstracting Performance-Relevant Properties of Modern Server Processors," *Supercomputing Frontiers and Innovations*, vol. 7, no. 2, p. 54–78, Jul. 2020, DOI: 10.14529/jsfi200204.

[22] H. Stengel, J. Treibig, G. Hager, and G. Wellein, "Quantifying Performance Bottlenecks of Stencil Computations Using the Execution-Cache-Memory Model," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, ser. ICS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 207–216, DOI: 10.1145/2751205.2751240.

[23] J. Laukemann and G. Hager, "Core-level performance engineering with the open-source architecture code analyzer (osaca) and the compiler explorer," in *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '23 Companion. New York, NY, USA: Association for Computing Machinery, 2023, p. 127–131, DOI: 10.1145/3578245.3583716.

[24] J. D. McCalpin, "Memory Bandwidth and Machine Balance in Current High Performance Computers," *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19–25, Dec. 1995. [Online]. Available: https://www.researchgate.net/publication/213876927_Memory_Bandwidth_and_Machine_Balance_in_Current_High_Performance_Computers

[25] Julian Hammer, "Design and Implementation of an Automated Performance Modeling Toolkit for Regular Loop Kernels," Ph.D. dissertation, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2023, DOI: 10.25593/opus4-fau-21514.

[26] I. E. Papazian, "New 3rd Gen Intel® Xeon® Scalable Processor (Codename: Ice Lake-SP)," in *Hot Chips Symposium*, 2020, pp. 1–22, DOI: 10.1109/HCS49909.2020.9220434.

[27] G. Hager. [Online]. Available: https://blogs.fau.de/hager/archives/8997

Appendix: Artifact Description

Artifact Description (AD)

I. Overview of Contributions and Artifacts

A. Paper's Main Contributions

- C_1 We propose three **in-core performance models** for the microarchitectures Neoverse V2, Golden Cove, and Zen 4 from the state-of-the-art servers NVIDIA Grace CPU Superchip, Intel Sapphire Rapids, and AMD Genoa, respectively, for the static code analyzer OSACA and compare the accuracy with LLVM-MCA.
- C_2 We investigate the architectural differences between the three servers in terms of **sustained clock frequency** for arithmetic-hot code.
- C_3 We investigate the architectural differences between the three servers in terms of **write-allocate evasion**.

B. Computational Artifacts

- A_1 https://github.com/RRZE-HPC/OSACA/releases/tag/v0.6.0
- A_2 https://github.com/RRZE-HPC/ibench
- A_3 https://doi.org/10.5281/zenodo.13770512

Artifact ID	Contributions Supported	Related Paper Elements
A_1	C_1	Tables 2–3
A_2	C_1	Tables 2–3
A_3	C_1, C_2, C_3	Figures 2–4

II. Artifact Identification

A. *Computational Artifact* A_1

Relation To Contributions

The OSACA tool is used to apply the created in-core performance models on the target code to analyze and contains the performance information for an assembly instruction for a specific architecture.

Expected Results

The results should be consistent with the numbers shown in Table 2 and 3.

Expected Reproduction Time (in Minutes)

The expected computational time of this artifact is 1 minute.

Artifact Setup (incl. Inputs)

Hardware: Since the tool in version 0.6.0 already contains all data for the relevant hardware models, no specific hardware is needed.

Software: To see the performance data, no further software than the git repository from the artifact link is enough.

Datasets / Inputs: No datasets or input data is needed.

Installation and Deployment: To install OSACA, you require Python in version 3.10 and the python packages networkx, pyparsing, ruamel.yaml, and Kerncraft $\geq$ v0.8.16.

Artifact Execution

Clone the repository via git and open the hardware models for the microarchitectures ("v2.yml" for Neoverse V2, "spr.yml" for Golden Cove, and "zen4.yml" for Zen 4, respectively) in `osaca/data/`. The information for Table II can be found in the "load_throughput", "store_throughput", "ports", and "port_model_scheme" fields of the database file. See Section II-C for the corresponding assembly instruction for Table 3.

Artifact Analysis (incl. Outputs)

No further output as the execution requires the manual lookup of the database files.

B. *Computational Artifact* A_2

Relation To Contributions

The ibench tool is used to measure the actual throughput and latency values shown in the OSACA database used for Tables 2 and 3.

Expected Results

The results should match the performance data in the OSACA database YAML files.

Expected Reproduction Time (in Minutes)

The expected computational time of this artifact is 1 minute.

Artifact Setup (incl. Inputs)

Hardware: The measurements require to be run on a NVIDIA Grace Superchip, Intel Sapphire Rapids, and AMD Genoa server.

Software: To compile the benchmarks, a modern compiler like GCC 13.2 is needed on all systems. On Sapphire Rapids and AMD Genoa, fixing the frequency is required for accurate measurements. Although this can be done in different ways, the authors of this work used the SLURM for this and further only show this option in the Artifact Description; please refer to the SLURM documentation for installation.

Datasets / Inputs: Most of the assembly instructions already exist in the ibench repository. To reproduce all instructions, the manual creation of further benchmarks is required. Please refer to the existing benchmarks in `src/<ISA-extension>` to create the corresponding benchmarks if needed.

Installation and Deployment: Clone the repository via git, adjust the "Makefile" to match your compiler and compile the benchmarks on the target platform with `make`.

Artifact Execution

T_1: After successful compilation, you can run each ISA extension (such as "AVX-512" or "SVE") with a fixed frequency using `srun --cpu-freq=<base-freq>-<base-freq>:performance ./ibench <ISA-extension> <base-freq>`, with `base-freq` being the base frequency of the server. Note that fixing the clock frequency is not needed for the NVIDIA Grace Superchip as it runs already with the frequency of 3.4 GHz.

Artifact Analysis (incl. Outputs)

T_1 will output the throughput and latency measurements of all compiled benchmarks of the given ISA extension.

C. *Computational Artifact* A_3

Relation To Contributions

The artifact includes all necessary scripts for creating the Figures 2–4 in the paper.

Expected Results

The results should be consistent with the graphs shown in Figures 2–4.

Expected Reproduction Time (in Minutes)

The expected computational time of this artifact is 5 hours on each platform.

Artifact Setup (incl. Inputs)

Hardware: The measurements require to be run on a NVIDIA Grace Superchip, Intel Sapphire Rapids, and AMD Genoa server. Make sure the Sapphire Rapids server is set in SNC mode and the Genoa server is set to NPS=4 mode.

Software: The artifact requires OSACA==0.6.0 (see requirements in Section II-A of the Artifact Description), as well the Python packages jupyter, kerncraft, pandas==2.2.1, numpy==1.26.4, and matplotlib==3.5.2. Furthermore, LLVM-MCA, the compilers GCC 12.1 and Arm C Compiler 23.10 for the NVIDIA Grace Superchip and GCC 13.2, oneAPI 2023.2, and LLVM Clang 17.0.6 for Intel Sapphire Rapids/AMD Genoa are needed. Finally, the Likwid toolsuite needs to be installed on all systems.

Datasets / Inputs: No further inputs are needed.

Installation and Deployment: Make sure all software requirements are installed. Please refer for this to the respective documentation of all mentioned packages mentioned in Section II-C. Copy the content of the artifact into the `validation/` directory of the OSACA repository.

Artifact Execution

The workflow consists of four tasks: T_1, T_2, T_3, T_4 with the sequence $T_3 \rightarrow T_4$ while the rest is independent.

T_1: For each of the three servers, adjust the `test_frequency.sh` file (i.e, set the right `ARCH` value) and run it on the target hardware. This creates three CSV files named `freqs_<ARCH>.csv` and one file named `freqs_spr_avx.csv`.

T_2: For each of the three servers, run the `test_store_ratio.sh` file on the target hardware. This creates three CSV files named `store_<ARCH>.csv` and another file with "_nt" suffix for Sapphire Rapids and Genoa.

T_3: On each of the three servers, run the `build_and_run_PMBS24.py` script, preferably with a fixed clock frequency to the base frequency of the servers in case of Sapphire Rapids and Genoa. You might have to manually add the number of bytes per assembly iteration if the script is not capable of retrieving this information automatically. This creates a result directory in the "build/" directory for each architecture.

T_4: After finishing T_3, you can open the `Analysis_PMBS24.ipynb` Jupyter notebook and go through the cells to post-process the created data from T_3.

Artifact Analysis (incl. Outputs)

Compare the CSV files from T_1 with Figure 2 in the paper, you should see the same behavior of declining frequencies for Intel Sapphire Rapids and AMD Genoa and a constant sustained frequency for NVIDIA Grace.

Compare the CSV files from T_2 with Figure 4 in the paper, you should see that the measured traffic to reported traffic ratio stays constant at around 2.0 for AMD Genoa, constant at around 1.0 for NVIDIA Grace and AMD Genoa with NT-stores, and declines with respect to the ccNUMA domains for Intel Sapphire Rapids, with a minimum around 1.4 to 1.6 (depending on the server model). The NT-stores version for Sapphire Rapids should match with Figure 4 and stay around 1.1.

The produced graph in the final part of the Jupyter Notebook of T_4 should match with Figure 3 in the paper.