Benchmarking the Evolution of Performance and Energy Efficiency Across Recent Generations of Intel Xeon Processors

Balázs Drávai
Faculty of Information Technology and Bionics
Pázmány Péter Catholic University
Budapest, Hungary
balazs.dravai@itk.ppke.hu

István Z Reguly
Faculty of Information Technology and Bionics
Pázmány Péter Catholic University
Budapest, Hungary
reguly.istvan@itk.ppke.hu

***Abstract*—In the span of 1.5 years, Intel has launched four families of Xeon Processors, with some novel architectural features; first the Sapphire Rapids generation which featured a version with on-package HBM, the Emerald Rapids generation, and then differentiated by releasing the performance-oriented Granite Rapids and the efficiency-oriented Sierra Forest families. In this work, we evaluate the performance and efficiency of CPUs from each of these generations and variants, with a particular focus on bandwidth-bound high performance computing (HPC) applications. We contrast runtime and energy consumption figures and track trends across generations.**

***Index Terms*—Benchmarking, Xeon, Performance, Energy, Cache, CFD**

I. Introduction

In recent years, Intel has released multiple generations of Xeon processors, each advancing performance and energy efficiency. This was enabled by moving from the Intel 7 to the Intel 3 process, and innovations such as chiplet designs or the integration of High Bandwidth Memory (HBM). This paper examines the performance and energy efficiency evolution of these processors, specifically focusing on Sapphire Rapids (SPR), Emerald Rapids (EMR), Granite Rapids (GNR), and Sierra Forest (SRF) product families. These families are differentiated based on their core counts, cache sizes, and energy efficiency, and with the latest generation, the separation of performance-oriented (P) and efficiency-oriented (E) families.

Memory bandwidth is a key limiter for many applications - newer generations support higher speed DDR modules, going from DDR5-4800 MHz in SPR, up to 8800 MHz with GNR, and increasing from 8 memory channels to 12 with GNR and SRF. At the same time the TDP for the highest core count models were at 350W for SPR and EMR, and this was increased to 500W with GNR.

Despite the advancements in GPU acceleration, many legacy codes remain CPU-only (such as nuclear security), making the evaluation of newer generation CPUs crucial. The absence of plans for further HBM-equipped CPUs raises questions about the performance of DDR-based solutions for these workloads. Our study provides an evaluation of the performance and energy efficiency of recent Intel Xeon processor generations. By examining a set of primarily bandwidth-bound applications, we shed light on the implications of these architectural innovations and inform the design of computing systems.

In this paper, we make the following contributions:

1) Establish performance baselines for the 5 Intel and 1 AMD systems to be used for architectural efficiency calculations.
2) Benchmark a variety of structured- and unstructured-mesh codes and contrast their performance in absolute terms (runtime) as well as relative (achieved architectural efficiency).
3) Measure the energy consumption of these applications running on the 6 bare-metal Intel systems, and study how energy efficiency improves across generations. For four, we report the energy consumption of the DRAM subsystem as well.

II. Systems and Baseline Performance

For this study, we evaluate the following systems - with Intel machines available in the Intel Developer Cloud. The full specifications are provided in Appendix A. Intel Xeon CPU MAX 9480 Processor with HBM (Sapphire Rapids, codenamed SPR+HBM), Intel Xeon Platinum 8480+ Processor (Sapphire Rapids, codenamed SPR+DDR), Intel Xeon Platinum 8592+ Processor (Emerald Rapids, codenamed EMR), Intel Xeon Platinum 6960P Processor (Granite Rapids, codenamed GNR), Intel Xeon 6740E Processor (Sierra Forest, codenamed SRF 96c), Intel Xeon (Unnamed) Processor (Sierra Forest, codenamed SRF 192c), AMD EPYC 9B14 (codenamed Genoa), available as a C3D virtual machine in Google Cloud (this VM has the access to RAPL energy counters disabled, therefore we could not collect energy measurements).

Table I shows the achieved bandwidth as measured by BabelStream Triad [1] with OpenMP, running on one or both sockets of each tested platform, noting the highest achieved bandwidth in Last Level Cache (LLC) and main memory (DDR or HBM), as well as the speedup of main memory bandwidth in comparison to SPR DDR. Subsequent generations of Xeon chips have larger Last Level Caches (LLC):

TABLE I
KEY FEATURES OF THE STUDIED PLATFORMS: NUMBER OF PHYSICAL CORES, LAST LEVEL CACHE SIZE, MEASURED MAXIMUM CACHE BANDWIDTH (GB/S), SUSTAINED BANDWIDTH (GB/S) AT 39 GB TOTAL ARRAY SIZE, AND RELATIVE SPEEDUP OF SUSTAINED BANDWIDTH OVER SPR DDR.

	SPR HBM	SPR DDR	EMR	GNR	SRF 96c	SRF 192c	Genoa
Cores	112	112	128	144	192	384	180
LLC (MB)	105	105	320	432	96	192	384
Cache BW	3481	4340	8149	7346	4139	7627	9534
DDR BW	1475	388	542	1150	396	667	529
Speedup	3.8	1.0	1.39	2.96	1.04	1.23	1.36

105 MB per socket for SPR, 320 MB for EMR, and 432 for GNR, but only 96 MB for SRF (96 core variant, 192 MB for the 192 core variant) - the tested Genoa CPU had 384 LLC per socket. Relative differences between maximum cache bandwidth and maximum DRAM bandwidth are also noteworthy: SPR, EMR and SRF (96 core) support 8 channels of DRAM per socket, and have a ratio of 11-15×, GNR, SRF 192 core, and Genoa have 12 channels per socket, yet GNR only has an 6.4× difference, and Genoa has a 18× difference. We also see a 2.96× increase in main memory bandwidth in the span of 1.5 years, going from SPR to GNR. SPR+HBM (the Intel Xeon MAX CPU) has a much higher bandwidth, but limited size HBM memory (64 GB per socket), and only a 2.4× difference between LLC and HBM bandwidth

The Intel machines were bare metal systems accessed in the Intel Developer Cloud, whereas the Genoa machine is a VM in Google Cloud. While VMs are known to have inefficiencies over bare-metal servers [2], these tend to be under 10% for most applications comparable to our benchmarks (bandwidth and compute intensive).

III. Applications and Parallelizations

We consider a number of primarily bandwidth-bound codes, which are implemented on top of two domain specific languages: OPS [3] (for structured meshes) and OP2 (for unstructured meshes) [4]. Prior work has shown that these implementations match or outperform the original hand-written implementations, and therefore the use of a DSL does not impede performance [5]–[8], but it does enable automatic parallelization with MPI as well as a variety of shared-memory parallel programming models.

Specifically we evaluate the following structured-mesh applications: CloverLeaf 2D/3D [9], OpenSBLI with Store All (SA) and Store None (SN) formulations [6], RTM, and Acoustic [10]. We study the following unstructured mesh applications: MG-CFD [11], Volna [12]. These codes are mainly bandwidth-bound, and to varying extents sensitive to latency [13]. Furthermore, we run the compute-intensive miniBUDE [14] proxy application. Details of benchmark applications and configurations are given in Appendix A.

In this work, we evaluate pure MPI parallelizations (both 1 process per physical core and 1 process per SMT logical core, where HyperThreading was enabled), as well as a hybrid MPI+OpenMP parallelization, where we use one process per NUMA region. For all computational loops in structured-mesh applications OpenMP's `collapse` is used for the whole loop nest, with the innermost loop instructed to vectorize using `omp simd`. For unstructured mesh codes we study auto-vectorizing implementations that explicitly pack and unpack vector registers, with the vector length adjusted for the platform (256 bit for SRF, 512 for the rest). In prior work, we studied SYCL parallelization on CPUs [13], and showed that it delivers lower performance compared to MPI/MPI+OpenMP - since this has not improved significantly since, we do not report performance with MPI+SYCL here.

IV. Results

We perform four runs of each benchmark configuration and average the results - each run reports both total wall time (excluding initial setup) as well as detailed per-subroutine timings with achieved effective bandwidth. Bandwidth is calculated as the sum of array sizes accessed by a given sweep across the grid (multiplied by 2 if read and written), divided by execution time, and it includes MPI communications. The variation in measured results were consistently below 5%. In the following results, we report the performance of the best variant (out of MPI, MPI+OpenMP, and SMT configurations), unless stated otherwise.

Figure 1 shows the wall times for different applications, and Figure 2 shows the hardware efficiency as calculated by the fraction of peak bandwidth (from Table I) and achieved effective bandwidth. miniBUDE is reported separately in Figure 2 showing fraction of peak FP32 compute achieved.

The first immediate observation is that the SPR HBM system closely competes with GNR on the structured-mesh benchmarks; these applications all have regular memory access patterns and low to moderate amounts of computational intensity, and therefore the primary factor determining performance should be the available bandwidth. As Figure 2 shows, only 45% of peak bandwidth is achieved on average, as shown in prior work [15] this is largely due to a shift towards latency limitations, as well as a lack of concurrency capable of saturating the available bandwidth [16]. Indeed, on average SPR HBM spends 26% of total runtime doing MPI communications, compared to 18-22% on the other Intel platforms.

The simplest application, CloverLeaf 2D exhibits the highest fraction of peak bandwidth achieved, and is the most consistent across different platforms too. Due to its simpler access patterns, it even has cross-loop data reuse, resulting in close to 100% utilization thanks to caching effects. CloverLeaf 2D also has a small MPI overhead; as little as 1.2% on EMR, up to 20% on Genoa (average at 7%). MG-CFD also exhibits excellent memory locality, especially at coarser multigrid levels, hence the high efficiency values. For both codes we can see a trend to higher efficiencies with larger cache sizes.

At the other end of the spectrum, RTM and Acoustic employ high-order (8^{th}) stencils, and especially the former has complex control flow and significant amounts of compute,

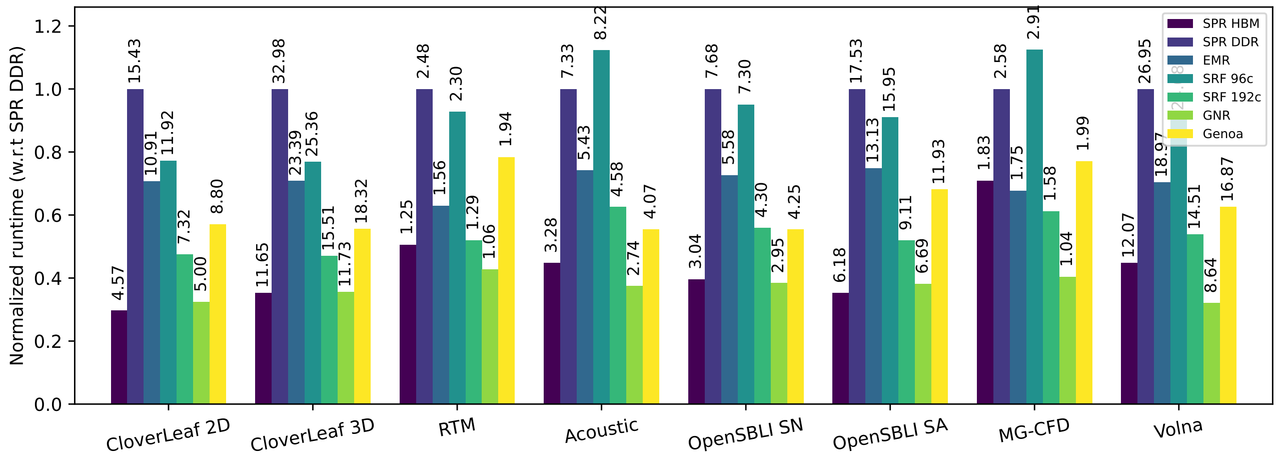


Fig. 1. Execution time of different benchmarks on the test systems, normalized to SPR DDR. Data labels are absolute runtimes.

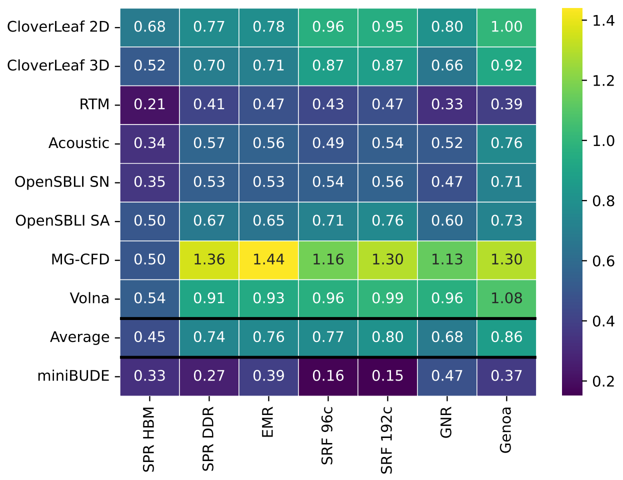


Fig. 2. Fraction of peak bandwidth achieved, and their average by platform. Fraction of peak FP32 compute shown separately for miniBUDE

therefore we observer lower bandwidth utilization. These applications are especially communication-intensive, with 34% and 44% of time spent in MPI respectively.

In-between, with the OpenSBLI applications, the Store All (SA) version achieves on average 65% bandwidth utilization, and spends only 10% of runtime in MPI, and the more computationally intensive Store None (SN) version achieves 53% of peak bandwidth and has a 25% MPI overhead - nevertheless SN is actually 2.3× faster than SA, showing that it is well worth trading computations for a reduction in memory movement. The Volna unstructured-mesh code does not have multigrid, yet achieves good cache locality in the indirect-access loops (of which there are fewer compared to MG-CFD).

The compute-intensive miniBUDE benchmark on the other hand is limited by compute throughput and control latency to a smaller some extent, we observe 27-47% efficiency, steadily increasing with subsequent compute-oriented generations. The Sierra Forest platforms on the other hand have significantly reduced efficiency: these chips are built with efficiency cores, with fewer and smaller (256-bit AVX2 instead of AVX-512) vectors, and presumably cannot run at high-frequencies on compute-intensive tasks unlike the performance-oriented Xeons.

A. Performance evolution across generations of Xeons

It is revealing to analyze the performance improvement of the subsequent generations of Xeons, given the architectural improvements over time. Designs moved from 4 chiplets with CPU cores in SPR to 2 chiplets with EMR, and to 3 with GNR, last level cache size increased from 105 MB per socket from SPR to 432 MB with GNR. The introduction of energy-efficient designs with SRF also significantly moved the architectural trade-offs.

Considering that most of our benchmarks are mainly bandwidth-limited, we calculate the ratio between the observed speedup over SPR DDR to the ratio of available bandwidth compared to SPR DDR: $(runtime_X/runtime_{SPR\ DDR})/(bandwidth_X/bandwidth_{SPR\ DDR})$. This gives us a sense of how "balanced" the architectural evolution is in terms of improvements in bandwidth and other factors (cache, compute, latency).

The most obvious case is that of the SPR HBM, which is architecturally the same as SPR DDR with the exception of the high-bandwidth memory, and therefore our benchmark applications experience a shift from being purely bandwidth limited to being more constrained by latency and cache behavior, only improving overall by 64% compared to the improvement in bandwidth.

The Emerald Rapids (EMR) platform saw a 15% improvement in core count, a 3× increase in cache size, a move from 2 chiplets to one, and a 40% improvement in available bandwidth. Overall these improvements have been balanced with applications improving overall within 2% of the bandwidth increase, meaning EMR is approximately 40% faster than SPR DDR on these benchmarks. The Granite Rapids (GNR) platform presents an interesting further step in this direction: we have another 12.5% increase in core count, a 1.35× increase in cache size, and 2.13× improvement in bandwidth compared to Emerald Rapids. This further massive lift in available bandwidth is slightly less balanced however, with 91% speedup compared to the improvement in bandwidth. Nevertheless, GNR in absolute terms is still on average 2.71× faster than SPR DDR and 1.93× faster than EMR.

The Sierra Forest family of CPUs represents a different set of architectural trade-offs with their energy-efficent design. The 96 core variant has 8 memory channels per socket, just like SPR and EMR, while the 192 core variant has 12 channels, similarly to GNR. While core counts are 1.3-3.4× higher, cache sizes are smaller, and the available bandwidth is 4% (96 core) or 70% (192 core) higher compared to SPR DDR. Yet SRF is on average 5% and 10% faster compared to the increase in bandwidth for the 96 core and 192 core variants respectively. Thanks to the higher amount of concurrency, it can more efficiently saturate the available bandwidth - and despite the higher core counts it does not suffer from significantly higher MPI overheads (19-23% on average, vs. 18-21% on SPR/EMR/GNR). The 192 core SRF platform represents a balanced improvement over the 96 core one, with a 1.65× increase in bandwidth, yielding a 1.73× improvement in the overall performance of our benchmarks (except miniBUDE, which has a 2x improvement, directly proportional to available cores).

B. *Energy efficiency*

Intel have built the Running Average Power Limit [17] counters and controls into their chips since the Sandy Bridge generation, which amongst other things has made it possible to accurately measure power consumption at the millisecond-scale. While energy counters may be available for the whole system, and components such as integrated graphics, main memory, and CPU package, on the studied Intel platforms we were able to measure the consumption of the CPU+memory system, and just the memory subsystem on EMR, SRF, and GNR. On Genoa these counters were not available due to security vulnerabilities in cloud VMs.

Figure 3 gives and overview of the reported energy consumption (CPU and memory together, and memory separately, for SPR HBM only on-package memory is included, external DRAM is not). The trends paint a similar picture to the execution times, since most of these CPUs have the same 2×350W TDP (SPR, EMR, SRF 192c), but notably the SRF 96c system has 2×250W, and the GNR has 2×500W (these TDP figures apply to the CPU only and exclude off-chip memory). This is also shown in Figure 4 that presents the average power (energy consumption divided by execution time), as well as the speedups in execution time compared to SPR DDR. We can see SPR HBM running on average 2.4× faster than SPR DDR, but at the same power, yielding a corresponding 2.4× improvement in energy efficiency. The EMR system on average consumes 20W less power than SPR DDR, but runs 1.42× faster, and therefore is 1.46× more energy efficient. The SRF 96 core system runs at only 445 Watts, yet achieves a 1.09× speedup, and therefore is 1.75× more energy efficient. The SRF 192 core system increases TDP, but also bandwidth and overall performance - it is 1.87× faster than SPR DDR, and is 1.88× more energy efficient. The GNR system further increases TDP and effective power to 1025 Watts on average - while being 2.72× faster than SPR DDR, it is also 1.88× more energy efficient. Power figures for the memory system are also included in Figure 4 where we were able to measure them; on both EMR and SRF 96c they account for 28% of total power - they both have 16 DIMMs installed, though the former has 5600 MHz modules while the latter 6400 MHz ones. The SRF 192 core system has 24 DDR5-6400MHz modules installed, and the memory system represents 40% of total power on these benchmarks. The GNR system has 24 DDR5-8800 MHz modules installed, accounting for 27% of total power.

The SRF and GNR systems nicely demonstrate the available trade-offs between energy efficiency and performance in relative terms to SPR DDR:

1) SRF 96 core: Slightly higher performance (1.09×) but at a 1.4× lower power
2) SRF 192 core: Significantly higher performance (1.87×), at the same power
3) GNR: Even higher performance (2.72×), but at 1.44× more power

V. Conclusions

This paper evaluated the performance and energy efficiency of Intel's recent Xeon-series processors released over the past two years, from the Sapphire Rapids generation, through Emerald Rapids, and to the recently differentiated energy-efficient Sierra Forest and performance-oriented Granite Rapids generations. We focus on a number of bandwidth-bound structured mesh and unstructured mesh applications, and compared results to an AMD EPYC Genoa-generation CPU.

For the traditional performance-oriented Xeons these generations brought a 3-4× improvement in cache size and available main memory bandwidth, but only a 38% increase in core counts. Overall, our applications' performance on Granite Rapids improved by 2.7× compared to Sapphire Rapids, and at the same time improved energy efficiency by 1.88×, despite the higher TDP of 500W per socket. The newly introduced energy-efficient Sierra Forest family of CPUs in contrast have much higher core counts (96-192 per socket), and are more energy efficient; delivering 9% more performance than Sapphire Rapids with 75% less energy at only 250W per socket (96 core version), or 87% higher performance at the same 350W per socket, resulting in 87% less energy consumed (192 core version).

Our work underlines the importance of architectural advancements in improving the performance and energy efficiency of HPC systems. The rapid evolution from Sapphire

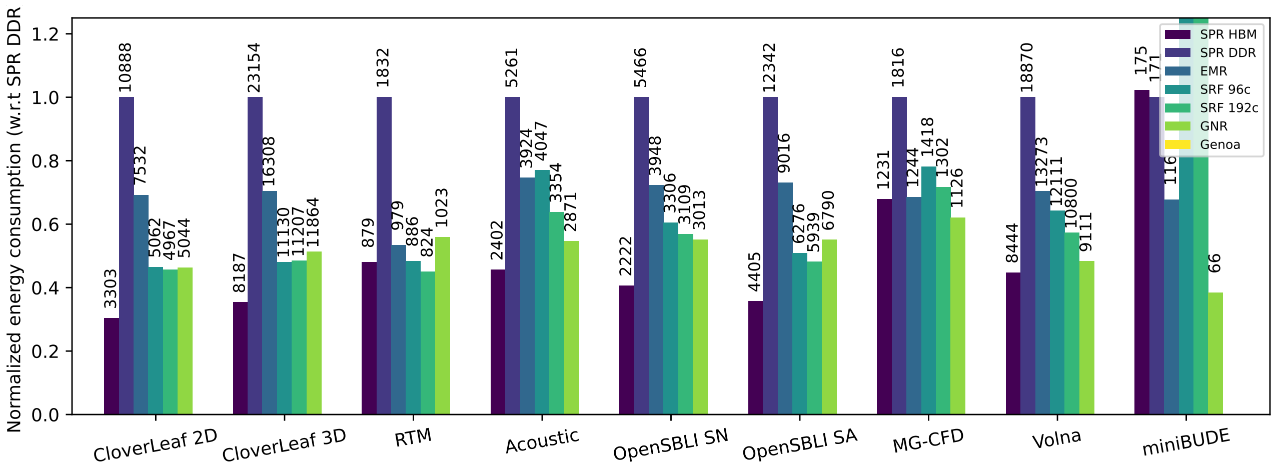
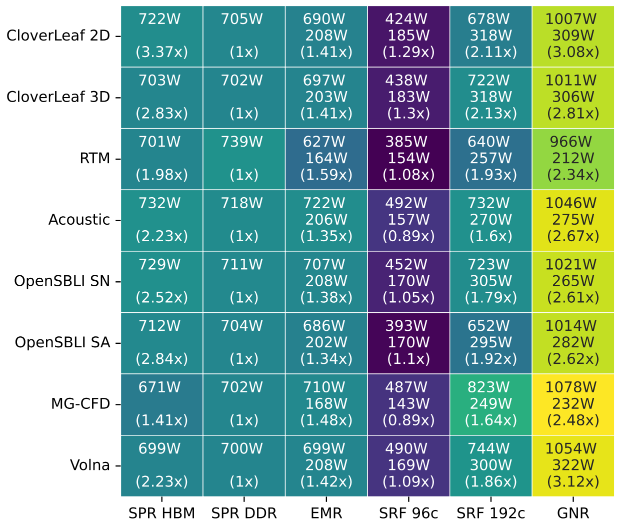


Fig. 3. Energy consumption of different benchmarks on different Intel CPU platforms, normalized to SPR DDR. Data labels are Joules.



	SPR HBM	SPR DDR	EMR	SRF 96c	SRF 192c	GNR
CloverLeaf 2D	722W (3.37x)	705W (1x)	690W 208W (1.41x)	424W 185W (1.29x)	678W 318W (2.11x)	1007W 309W (3.08x)
CloverLeaf 3D	703W (2.83x)	702W (1x)	697W 203W (1.41x)	438W 183W (1.3x)	722W 318W (2.13x)	1011W 306W (2.81x)
RTM	701W (1.98x)	739W (1x)	627W 164W (1.59x)	385W 154W (1.08x)	640W 257W (1.93x)	966W 212W (2.34x)
Acoustic	732W (2.23x)	718W (1x)	722W 206W (1.35x)	492W 157W (0.89x)	732W 270W (1.6x)	1046W 275W (2.67x)
OpenSBLI SN	729W (2.52x)	711W (1x)	707W 208W (1.38x)	452W 170W (1.05x)	723W 305W (1.79x)	1021W 265W (2.61x)
OpenSBLI SA	712W (2.84x)	704W (1x)	686W 202W (1.34x)	393W 170W (1.1x)	652W 295W (1.92x)	1014W 282W (2.62x)
MG-CFD	671W (1.41x)	702W (1x)	710W 168W (1.48x)	487W 143W (0.89x)	823W 249W (1.64x)	1078W 232W (2.48x)
Volna	699W (2.23x)	700W (1x)	699W 208W (1.42x)	490W 169W (1.09x)	744W 300W (1.86x)	1054W 322W (3.12x)

Fig. 4. Average power draw of the CPU and memory, the memory separately (where available), and speedup compared to SPR DDR

Rapids to Granite Rapids brought substantial benefits, particularly for bandwidth-bound applications. The introduction of the energy-efficient Sierra Forest family further demonstrates the potential for balancing performance and energy consumption in server CPUs. These findings provide valuable insights for optimizing HPC workloads and guiding the development of next-generation processors.

Acknowledgment

We are grateful for the support of the OneAPI Innovator program, and the advice and assistance of Rupak Roy, Omar Toral, Sri Ramkrishna at Intel in particular.

This research was supported by National Research, Development and Innovation Fund of Hungary (FK 145931), under the FK 23 funding scheme, as well as by Rolls-Royce plc., and by the UK EPSRC (EP/S005072/1 – Strategic Partnership in Computational Science for Advanced Simulation and Modelling of Engineering Systems – ASiMoV). This study was also partially funded by the National Research, the Development and Innovation Office in Hungary (RRF-2.3.1-21-2022-00004).

References

[1] T. Deakin, J. Price, M. Martineau, and S. M. Smith, "Evaluating attainable memory bandwidth of parallel programming models via babelstream," *International Journal of Computational Science and Engineering*, vol. 17, p. 247, 2018.

[2] S. Giallorenzo, J. Mauro, M. G. Poulsen, and F. Siroky, "Virtualization costs: Benchmarking containers and virtual machines against bare-metal," *SN Computer Science*, vol. 2, p. 404, 2021. [Online]. Available: https://doi.org/10.1007/s42979-021-00781-8

[3] I. Z. Reguly, G. R. Mudalige, M. B. Giles, D. Curran, and S. McIntosh-Smith, "The ops domain specific abstraction for multi-block structured grid computations," in *2014 Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing*. IEEE, 2014, pp. 58–67.

[4] I. Reguly, "Op2: An active library framework for solving unstructured mesh-based applications on multi-core and many-core architectures," in *2012 Innovative Parallel Computing (InPar)*. IEEE, 2012, pp. 1–12.

[5] I. Reguly, A. Mallinson, W. Gaudin, and J. Herdman, "Performance analysis of a high-level abstractions-based hydrocode on future computing systems," in *High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation: 5th International Workshop, PMBS 2014, New Orleans, LA, USA, November 16, 2014. Revised Selected Papers 5*. Springer, 2015, pp. 85–104.

[6] C. T. Jacobs, S. P. Jammy, and N. D. Sandham, "OpenSBLI: A framework for the automated derivation and parallel execution of finite difference solvers on a range of computer architectures," *Journal of Computational Science*, vol. 18, pp. 12–23, 2017.

[7] A. Owenson, "Towards the use of mini-applications in performance prediction and optimisation of production codes," Ph.D. dissertation, University of Warwick, 2020.

[8] G. R. Mudalige, I. Z. Reguly, A. Prabhakar, D. Amirante, L. Lapworth, and S. A. Jarvis, "Towards virtual certification of gas turbine engines with performance-portable simulations," in *2022 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2022, pp. 206–217.

[9] A. Mallinson, D. A. Beckingsale, W. Gaudin, J. Herdman, J. Levesque, and S. A. Jarvis, "Cloverleaf: Preparing hydrodynamics codes for exascale," *The Cray User Group*, vol. 2013, 2013.

[10] M. Louboutin, M. Lange, F. Luporini, N. Kukreja, P. A. Witte, F. J. Herrmann, P. Velesko, and G. J. Gorman, "Devito (v3. 1.0): an embedded domain-specific language for finite differences and geophysical exploration," *Geoscientific Model Development*, vol. 12, no. 3, pp. 1165–1187, 2019.

[11] A. Owenson, S. A. Wright, R. A. Bunt, Y. Ho, M. J. Street, and S. A. Jarvis, "An unstructured cfd mini-application for the performance prediction of a production cfd code," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 10, p. e5443, 2020.

[12] I. Z. Reguly, D. Giles, D. Gopinathan, L. Quivy, J. H. Beck, M. B. Giles, S. Guillas, and F. Dias, "The volna-op2 tsunami code (version 1.5)," *Geoscientific Model Development*, vol. 11, no. 11, pp. 4621–4635, 2018.

[13] I. Z. Reguly, "Evaluating the performance portability of sycl across cpus and gpus on bandwidth-bound applications," in *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, ser. SC-W '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 1038–1047. [Online]. Available: https://doi.org/10.1145/3624062.3624180

[14] A. Poenaru, W.-C. Lin, and S. McIntosh-Smith, "A performance analysis of modern parallel programming models using a compute-bound application," in *High Performance Computing*, B. L. Chamberlain, A.-L. Varbanescu, H. Ltaief, and P. Luszczek, Eds. Cham: Springer International Publishing, 2021, pp. 332–350.

[15] I. Z. Reguly, "Comparative evaluation of bandwidth-bound applications on the intel xeon cpu max series," in *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, ser. SC-W '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 1236–1244. [Online]. Available: https://doi.org/10.1145/3624062.3624195

[16] J. D. McCalpin, "Bandwidth limits in the Intel Xeon Max (Sapphire Rapids with HBM) Processors," in *ISC 2023 IXPUG Workshop*, 2023, pp. 1–4.

[17] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le, "Rapl: memory power estimation and capping," in *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 189–194. [Online]. Available: https://doi.org/10.1145/1840845.1840883

Appendix A
Test system specifications

Detailed specifications of the test systems:

1) Intel Xeon CPU MAX 9480 Processor with HBM (Sapphire Rapids, codenamed SPR+HBM), available in the Intel Developer Cloud. Two sockets, each with 56 cores, Hyperthreading on. 2x4 NUMA regions, with 2x64 GB HBM in HBM-only mode, with SNC4. Clock frequencies between 1.9 GHz (base frequency) - 2.6 GHz (all-core turbo), giving a theoretical 13.6-18.6 FP32 TFLOPS/s. Software: Intel OneAPI Base and HPC toolkits, 2024.1 (including Intel MPI). Benchmarked Jun 5, 2024.
2) Intel Xeon Platinum 8480+ Processor (Sapphire Rapids, codenamed SPR+DDR), available in the Intel Developer Cloud. Two sockets, each with 56 cores, Hyperthreading on. 16x32 GB DDR5-4800MHz RAM. Clock frequencies between 2.0 GHz (base frequency) - 3.0 GHz (all-core turbo), giving a theoretical 14.3-21.5 FP32 TFLOPS/s. Software: Ubuntu 22.04, Intel OneAPI Base and HPC toolkits, 2024.1 (including Intel MPI). Benchmarked June 13, 2024.
3) Intel Xeon Platinum 8592+ Processor (Emerald Rapids, codenamed EMR), available in the Intel Developer Cloud. Two sockets, each with 64 cores, Hyperthreading on. 16x64 GB DDR5-5600MHz RAM. Clock frequencies between 1.9 GHz (base frequency) - 2.9 GHz (all-core turbo), giving a theoretical 15.5-23.7 FP32 TFLOPS/s. Software: Ubuntu 22.04, Intel OneAPI Base and HPC toolkits, 2024.1 (including Intel MPI). Benchmarked June 15, 2024.
4) Intel Xeon Platinum 6960P Processor (Granite Rapids, codenamed GNR), available in the Intel Developer Cloud. Two sockets, each with 3 NUMA nodes and 72 cores total, Hyperthreading on. 24x32 GB DDR5-8800MHz RAM. Clock frequencies between 1.9 GHz (base frequency) - 2.9 GHz (all-core turbo), giving a theoretical 15.5-23.7 FP32 TFLOPS/s. Software: Centos 9 Stream, Intel OneAPI Base and HPC toolkits, 2024.1 (including Intel MPI). Benchmarked June 2, 2024.
5) Intel Xeon 6740E Processor (Sierra Forest, codenamed SRF 96c), available in the Intel Developer Cloud. Two sockets, each with 96 cores, no Hyperthreading. 16x64 GB DDR5-6400MHz RAM. Clock frequencies between 2.4 GHz (base frequency) - 3.2 GHz (all-core turbo), giving a theoretical 7.3-9.8 FP32 TFLOPS/s. Software: Centos 9 Stream, Intel OneAPI Base and HPC toolkits, 2024.1 (including Intel MPI). Benchmarked June 29, 2024.
6) Intel Xeon (Unnamed) Processor (Sierra Forest, codenamed SRF 192c), available in the Intel Developer Cloud. Two sockets, each with 192 cores, no Hyperthreading. 24x64 GB DDR5-6400MHz RAM. Clock frequencies between 2.4 GHz (base frequency) - 3.2 GHz (all-core turbo), giving a theoretical 14.6-19.6 FP32 TFLOPS/s. Software: Centos 9 Stream, Intel OneAPI Base and HPC toolkits, 2024.1 (including Intel MPI). Benchmarked Jul 18, 2024.
7) AMD EPYC 9B14 (codenamed Genoa), available as a C3D virtual machine in Google Cloud. Two sockets, each with 90 available cores, Hyperthreading off. 2x2 NUMA regions, with 24x64 GB DDR5 RAM, of which 1440 is available. Clock frequencies between 2.6 GHz (base frequency) - 3.6 GHz (turbo), giving a theoretical 19.9-27.5 FP32 TFLOPS/s. Software: Ubuntu 22.04, GCC 11.3. Benchmarked Jun 6, 2023.

Appendix B
Benchmark applications

The details of benchmark applications are as follows:

1) CloverLeaf 2D/3D [9] – developed by the UK mini-app consortium, CloverLeaf is a proxy for nuclear security codes, simulating shockwaves using a structured-mesh Eulerian hydrodynamics setup. The code is largely bandwidth-bound, with some operations on faces/edges that may be latency bound. Double precision, 7680^2(2D), 408^3(3D) problem size, 50 iterations.
2) OpenSBLI SA & SN [6] – a shock-capturing Navier-Stokes solver developed at the University of Southampton for studying aeroacoustic phenomena around aircraft. The code has 2 variants – Store All (SA), which

is bandwidth-bound, and Store None (SN), which recomputes derivatives on the fly, thereby trading compute for data movement, but is still mostly bandwidth bound. Double precision, 320^3 problem size, 20 time iterations.
3) RTM - computational geophysics proxy code, implementing the forward pass of a Reverse Time Migration algoritm. Uses an 8th order finite difference stencil. Due to the large stencils, it is sensitive to cache locality and vectorization, has large communications volume over MPI. Single precision, 320^3 problem size, 10 time iterations.
4) Acoustic – a proxy extracted from Devito [10], it is a structured-mesh high-order (8th) finite difference acoustic wave propagation solver. Bandwidth and cache locality bound, with large communications volume over MPI. Single precision, 1000^3 problem size, 30 time iterations.
5) MG-CFD [11] – a proxy for the Rolls-Royce CFD simulation code, implementing an unstructured mesh finite volume Euler equations solver with multigrid. Bound by latencies and indirect memory accesses. Double precision, NASA Rotor37 case with 8 million vertices, 25 iterations.
6) Volna [12] – unstructured mesh finite volume Nonlinear Shallow Water Equations solver. Also sensitive to indirect memory accesses as MG-CFD, but less so. Indian ocean case with 30 million vertices, 200 time iterations.
7) miniBUDE [14] – proxy molecular docking code representative of the University of Bristol's BUDE. Compute and latency bound. bm1 testcase, 30 iterations.

Appendix C
Artifact Description

A. Abstract

The paper performs the detailed benchmarking of a number of computational science applications and proxies on Intel(R) Xeon processors fromt he Sapphire Rapids, Emerald Rapids, Sierra Forest, and Granite Rapids families.

As such, the contributions of the paper rely entirely on computational results - specifically performance measurements. All measurements are reported automatically by the test framework, and are then parsed using scripts or by hand, and assembled into tables and figures shown in the paper. Most figures show runtimes and energy consumption, or achieved architectural efficiency, which are derived from the application logs.

The reproducibility of the results in the paper relies solely on the test framework available on GitHub (https://github.com/reguly/tests), which has been built to make it easy to run all experiments easily, producing application logs, which contain all the raw data for the results (runtimes, communication times, and bandwidth numbers).

B. Description

1) Check-list (artifact meta information):

- **Algorithm: Structured and unstructured mesh stencil computations**
- **Program: CloverLeaf, Acoustic, RTM, OpenSBLI, MG-CFD**
- **Compilation: Using standard CPU compilers, OneAPI and GCC**
- **Transformations: Code generation through the OPS and OP2 libraries**
- **Data set: built-in, or downloaded with the op2_get_data.sh script**
- **Run-time environment: See Appendix A of paper**
- **Hardware: See Appendix A of paper**
- **Execution: run_all.sh script**
- **Output: application logs for each application**
- **Publicly available?: yes**

2) How software can be obtained (if available): Source codes available at https://github.com/reguly/tests

3) Hardware dependencies: Described in Appendix A of the paper

4) Software dependencies: Fully contained in test repository. Benchmarked applicaitons are described in Appendix B of the paper

5) Datasets: Downloadable with the op2_get_data.sh script

C. Installation

1) `git clone --recursive https://github.com/reguly/tests.git`
2) To build dependent libraries, and download meshes, execute the `op2_dependencies.sh` and `op2_get_data.sh` scripts in the root directory
3) edit the `source_intel`, `source_gnu` files to match your environment, and choice of compilers
4) execute the `build_cpu.sh` script to build MPI and MPI+OpenMP versions of the applications

D. Experiment workflow

Edit and execute the `run_all.sh` script to match your compiler and application types to be tested (CPUTEST and/or TILING). Can use further command line parameters to run only a specific application

E. Evaluation and expected result

Results are placed in log files named `*_diag2`, these can be pulled into the current directory with e.g. `find ../../tests/ -name "*diag2" -exec mv {} ./ ;`. Use the `parse.py` file, passing the name of the directory containing these gathered files to extract timing data into tables.

F. Experiment customization

Experiments can be customized by editing the high-level configuration files in the root directory of the repository, including source_* files for different compilers, build_* files for building applications, and the runner_scripts/run_* scripts for running individual applications.