Journal of Neonatal Surgery
ISSN(Online): 2226-0439
Vol. 14, Issue 29s (2025)
https://www.jneonatalsurg.com

OPEN ACCESS

Laplacian Operators for Scientific Computing: A Comparative Analysis of CPU and GPU Implementations

Dr. Aswini J[1], Mr. Marinto Richee J[2], Dr. M. G. Dinesh[3], Dr. A. Gayathri[4]

[1,2]Saveetha Engineering College, Chennai, Tamil Nadu, India.

[3]Easa College of Engineering and Technology, Coimbatore, Tamil Nadu, India

[4]Saveetha School of Engineering, Saveetha Institute of Medical And Technical Sciences (SIMATS), Saveetha University, Chennai, Tamil Nadu, India

[1,2]Email ID: aswinijmarintorichee.student@saveetha.ac.in, [3]Email ID: dineshbabu.mg@gmail.com

[4]Email ID: gayathribala.sse@saveetha.com

Cite this paper as: Dr. Aswini J, Mr. Marinto Richee J, Dr. M. G. Dinesh, Dr. A. Gayathri, (2025) Laplacian Operators for Scientific Computing: A Comparative Analysis of CPU and GPU Implementations. *Journal of Neonatal Surgery*, 14 (29s), 75-84.

ABSTRACT

This paper presents a comprehensive bench-marking study of a 2D Laplacian filter implemented on both CPU and GPU architectures for image processing applications. The Laplacian filter serves as a fundamental tool in edge detection and feature extraction, playing a crucial role in various computer vision tasks

Keywords: *Image Processing · GPU Acceleration · Performance Benchmarking.*

1. INTRODUCTION

Numerical algorithms play a pivotal role in scientific computing, providing solutions to complex mathematical problems that may lack analytical solutions. These algorithms leverage computational methods to approximate solutions, enabling the modelling and simulation of real-world phenomena. One such class of numerical algorithms involves the use of Laplacian operators, which are fundamental in solving differential equations and are ubiquitous in various scientific and engineering disciplines.

Laplacian operators, denoted by ∇^2 or Δ, are differential operators that arise in the study of second-order partial differential equations (PDEs). These operators are instrumental in characterizing diffusion processes, heat conduction, and potential fields. In scientific computing, Laplacian operators are employed for tasks ranging from image processing to fluid dynamics simulations. Their ability to capture spatial variations and gradients makes them indispensable in modelling physical phenomena. The versatility of Laplacian operators extends to a myriad of real-world applications. In image processing, Laplacian filters are utilized for edge detection and 2 Dr. Aswini J. et al. sharpening. In physics, Laplacian operators describe the distribution of gravitational and electric potentials. Laplacian smoothing is applied in mesh generation for finite element analysis. Understanding and optimizing the computation of Laplacian operators are critical for enhancing the efficiency of these

applications.

This research paper delves into the implementation and optimization of Laplacian operators, with a focus on achieving superior performance through parallel computing on Graphics Processing Units (GPUs). The subsequent sections will explore the methodology, benchmarking results, speedup factor analysis, and comparisons with existing approaches. The paper concludes with insights into potential optimizations, challenges faced, and future directions for advancing Laplacian algorithms in scientific computing.

2. METHODOLOGY

In this research, we embarked on the analysis of Laplacian operators, aiming to enhance computational efficiency on both CPU and GPU architectures. The methodology involved a series of steps, including algorithmic analysis, GPU acceleration, parallel computing techniques, benchmarking, and performance evaluation.

Algorithmic Analysis:

Baseline Implementation: We initially implemented the Laplacian operator algorithms for both CPU and GPU using standard approaches.

Analysis and Identification: Through extensive analysis, we identified computational bottlenecks and areas for improvement within the algorithms.

GPU Acceleration:

CUDA Integration: Leveraging NVIDIA's CUDA platform, we parallelized the Laplacian computations for GPU acceleration.

GPU Kernels: We developed and executed GPU kernels to leverage the highly parallel architecture, ensuring optimal utilization of GPU resources.

Parallel Computing Techniques:

CPU Parallelization: For CPU architecture, we employed multi-threading techniques to parallelize Laplacian computations, optimizing for modern multi-core processors.

GPU Parallelization: The parallelization strategy for GPUs focused on breaking down the Laplacian computations into parallelizable tasks, aligning with the architecture of the GPU.

Benchmarking Methodology:

Hardware and Software Configuration: We conducted benchmarking on diverse hardware configurations, including CPUs and GPUs. The software environment, including programming languages (Python, CUDA), libraries (NumPy, CuPy), and tools (Numba, NVIDIA Nsight), was carefully configured for consistency.

Array Size Selection: Benchmarking involves systematically varying the array sizes to cover a spectrum of computational loads, allowing for a comprehensive performance assessment.

Performance Evaluation Criteria:

Key Metrics: The primary metrics for performance evaluation included execution time, throughput (operations per second), and memory usage.

Comparison with Baseline: Results were compared against the baseline implementation to quantify the improvements achieved through optimization.

Experimental Setup:

Hardware Platforms: We performed experiments on a range of CPUs and GPUs, detailing the specifications of each hardware platform.

Software Configuration: The software stack, including operating systems and versions of relevant libraries, was documented to ensure transparency and reproducibility.

Data Collection:

Systematic Testing: Rigorous testing procedures were applied to collect data under varying conditions. Multiple runs were conducted for each configuration to ensure reliable results. Data Logging: Detailed logs were maintained, recording execution times, throughput, and memory usage for subsequent analysis.

Statistical Analysis:

Speedup Calculation: Speedup factors were calculated based on the ratio of execution times between optimized and baseline implementations.

Statistical Significance: Statistical methods were applied to assess the significance of observed speedup factors, ensuring the reliability of the optimization results.

Ethical Considerations:

Data Privacy: As the research focused on algorithm optimization and benchmarking, no human subjects or sensitive data were involved, minimizing ethical concerns. Adherence to Ethical Standards: The research adhered to ethical standards, promoting responsible conduct in experimentation.

3. LITERATURE REVIEW

TABLE I: LITERATURE REVIEW TABLE 1

Title	Methodology	Results	Conclusion
A Survey on Hardware Accelerators for Large Language Models	Presents frameworks for acceleration of transformer networks for LLMs and NLP using hardware accelerators, performs qualitative and quantitative comparison.	Presents results of qualitative and quantitative comparison of different hardware accelerators for LLMs, highlights performance metrics and energy efficiency	Summarizes key findings, emphasizes importance of hardware accelerators for enhancing performance and energy efficiency of LLMs
Computing Large 2D Convolutions on GPU Efficiently	Introduces im2tensor algorithm for efficient computation of large 2D convolutions on GPUs, details approach and techniques used to enhance throughput	Presents findings related to performance improvements achieved through im2tensor algorithm, examines enhanced throughput and efficiency	Summarizes achievements, suggests future directions for optimizing computation of large 2D convolutions on GPUs.

TABLE 2: LITERATURE REVIEW TABLE 2

Title	Methodology	Results	Conclusion
GPU Support for Automatic Generation of Finite-Differences Stencil Kernels	Background on Devito, OPS, acoustic wave propagation model, DSLs, stencil code generation.	Compares with FEniCS, Firedrake, YASK, CTADEL, discusses DSLs and code generation for stencil computations.	Presents performance evaluation on NVIDIA devices, measures operational intensity, compares with hand-optimized code.
A stencil-based implementation of Parareal in the C++ domain specific embedded language STELLA	Overview of Parareal, time-parallel methods, STELLA language, stencil computations, PDEs.	Compares with PITA, RIDC, PFASST, discusses space-time multigrid, reviews implementation strategies of Parareal.	Reports results on Cray XC30 system, measures speedup, parallel efficiency, energy consumption.
Block-Relaxation Methods for 3D Constant-Coeffici ent Stencils on GPUs and Multicore CPUs	Background on block iterative methods, CUDA programming, stencil operations.	Compares with Feng et al., Anzt et al., Adams et al., discusses block-based smoothers and stencil operations.	Presents experimental results on AMD/NVIDIA hardware, benchmarks block Jacobi/Gauss-Seidel relaxations, measures wall time, speedup.
A Generic Library for Stencil Computations	Background on stencil computations, challenges in programmability and performance, related work.	Compares with auto-tuning techniques, cache oblivious algorithms, grid computing libraries.	Presents experimental results on different architectures, compares performance with C implementations, shows scalability.
AN5D: Automated Stencil Framework for High-Degree Temporal Blocking on GPUs	Background on stencil computation, spatial/temporal blocking, limitations of existing techniques.	Discusses existing techniques for temporal blocking, spatial blocking, compares with proposed framework.	Presents performance evaluation on Tesla V100 GPU, compares with existing methods, analyzes scalability.
A Synergy between On- and Off-Chip Data Reuse for GPU-based Out-of-Core Stencil Computation	Review of out-of-core stencil computation, GPU performance factors, previous optimization methods.	Reviews previous work on out-of-core stencil computation, compression techniques, identifies research gaps.	Reports speedup and memory reduction achieved by SO2DR, compares with competitors, analyzes impact of configurations.

Accelerating Compact Fractals with Tensor Core GPUs	Introduction to fractal geometry, discrete fractals, NBB class, tensor cores, compact fractal representation.	Surveys previous work on GPU thread mapping, tensor core usage, identifies novelty of proposed approach.	Shows speedup and memory reduction achieved by compact fractal approach, compares with bounding box approach.
Analytical Performance Estimation during Code Generation on Modern GPUs	Builds on analytical performance modeling, code generation, GPU cache behavior.	Reviews existing code generation frameworks, performance modeling, metric estimation.	Evaluates accuracy and usefulness of Warpspeed, applies to 3D-25pt stencil and LBM solver.
Casper: Accelerating Stencil Computations using Near-Cache Processing	Background on GPU architecture, roofline model, metric estimation, code generation.	Cites works on compression for stencil/LBM, compares with proposed approach.	Reports performance improvements with Casper on 3D-25pt stencil and LBM solver.

TABLE 3: LITERATURE REVIEW TABLE 3

Title	Methodology	Results	Conclusion
GPU Support for Automatic Generation of Finite-Differences Stencil Kernels	Background on Devito, OPS, acoustic wave propagation model, DSLs, stencil code generation.	Compares with FEniCS, Firedrake, YASK, CTADEL, discusses DSLs and code generation for stencil computations.	Presents performance evaluation on NVIDIA devices, measures operational intensity, compares with hand-optimized code.
A stencil-based implementation of Parareal in the C++ domain specific embedded language STELLA	Overview of Parareal, time-parallel methods, STELLA language, stencil computations, PDEs.	Compares with PITA, RIDC, PFASST, discusses space-time multigrid, reviews implementation strategies of Parareal.	Reports results on Cray XC30 system, measures speedup, parallel efficiency, energy consumption.
Block-Relaxation Methods for 3D Constant-Coefficient Stencils on GPUs and Multicore CPUs	Background on block iterative methods, CUDA programming, stencil operations.	Compares with Feng et al., Anzt et al., Adams et al., discusses block-based smoothers and stencil operations.	Presents experimental results on AMD/NVIDIA hardware, benchmarks block Jacobi/Gauss-Seidel relaxations, measures wall time, speedup.
A Generic Library for Stencil Computations	Background on stencil computations, challenges in programmability and performance, related work.	Compares with auto-tuning techniques, cache oblivious algorithms, grid computing libraries.	Presents experimental results on different architectures, compares performance with C implementations, shows scalability.
AN5D: Automated Stencil Framework for High-Degree Temporal Blocking on GPUs	Background on stencil computation, spatial/temporal blocking, limitations of existing techniques.	Discusses existing techniques for temporal blocking, spatial blocking, compares with proposed framework.	Presents performance evaluation on Tesla V100 GPU, compares with existing methods, analyzes scalability.
A Synergy between On- and Off-Chip Data Reuse for GPU-based Out-of-Core Stencil Computation	Review of out-of-core stencil computation, GPU performance factors, previous optimization methods.	Reviews previous work on out-of-core stencil computation, compression techniques, identifies research gaps.	Reports speedup and memory reduction achieved by SO2DR, compares with competitors, analyzes impact of configurations.
Accelerating Compact Fractals with Tensor Core GPUs	Introduction to fractal geometry, discrete fractals, NBB class, tensor cores, compact fractal representation.	Surveys previous work on GPU thread mapping, tensor core usage, identifies novelty of proposed approach.	Shows speedup and memory reduction achieved by compact fractal approach, compares with bounding box approach.

Analytical Performance Estimation during Code Generation on Modern GPUs	Builds on analytical performance modeling, code generation, GPU cache behavior.	Reviews existing code generation frameworks, performance modeling, metric estimation.	Evaluates accuracy and usefulness of Warpspeed, applies to 3D-25pt stencil and LBM solver.
Casper: Accelerating Stencil Computations using Near-Cache Processing	Background on GPU architecture, roofline model, metric estimation, code generation.	Cites works on compression for stencil/LBM, compares with proposed approach.	Reports performance improvements with Casper on 3D-25pt stencil and LBM solver.

TABLE 4: LITERATURE REVIEW TABLE 4

Title	Methodology	Results	Conclusion
Compression-Based Optimizations for Out-of-Core GPU Stencil Computation	Addresses out-of-core stencil computation, GPU memory constraints, compression techniques.	Reviews studies on compression techniques, compares with proposed methods.	Demonstrates speedup and memory reduction with compression, applies to acoustic wave propagation.
Distributed Parallelization of xPU Stencil Computations in Julia	Reviews Julia packages for distributed parallelization, discusses requirements for HPC software.	Mentions related Julia packages for MPI, GPU programming, and stencil computation.	Reports parallel weak scaling of solvers on Nvidia P100 GPUs with ImplicitGlobalGrid.jl.
Employing polyhedral methods to optimize stencils on FPGAs	Background on stencil codes, FPGAs, high-level synthesis, loop tiling, polyhedral methods.	Discusses existing work on stencil optimization on FPGAs, compares techniques.	Evaluates performance on 10 stencil codes, reports significant speedups over baseline.
Exploiting Scratchpad Memory for Deep Temporal Blocking	Background on stencil computations, temporal blocking, scratchpad memory.	Compares with state-of-the-art temporal blocking implementations on GPUs.	Evaluates performance on 2D Jacobian 5-point iterative stencil kernel, compares with StencilGen and AN5D.
Fortran performance optimisation and auto-parallelisation by leveraging MLIR-based domain specific abstractions in Flang	Background on Fortran performance optimization, auto-parallelization, MLIR.	Surveys previous work on Fortran performance optimization and auto-parallelization.	Evaluates performance on Fortran codes, demonstrates improvements.
Graph-based Neural Network Model for Scientific Paper Summarization	Background on scientific paper summarization, graph-based neural networks.	Surveys existing methods for scientific paper summarization.	Evaluates on arXiv and PubMed datasets, achieves state-of-the-art results.
GPU Methodologies for Numerical Partial Differential Equations	Introduces cuSten library for applying finite-difference stencils, develops batched tridiagonal and pentadiagonal solvers for GPUs.	Demonstrates significant performance improvements over existing methods, enables exploration of batches of PDEs.	Concludes GPU methods are efficient and reusable, suggests future extensions.
High-performance xPU Stencil Computations in Julia	Describes design and implementation of ParallelStencil.jl, highlighting its use of Julia's features for parallelism.	Shows performance close to theoretical upper bound on GPUs, significant speedup over traditional array programming on CPUs.	Concludes ParallelStencil.jl is effective and widely applicable, suggests future improvements.
Revisiting Temporal Blocking Stencil Optimizations	Introduces EBISU method, consisting of four components for deep temporal blocking on GPUs.	Achieves speedups up to 2.53x over state-of-the-art methods in stencil benchmarks.	Concludes EBISU is effective and scalable, suggests extending to other architectures.

TABLE 5: LITERATURE REVIEW TABLE 5

Title	Methodology	Results	Conclusion
Scalable communication for high-order stencil computations using CUDA-aware MPI	Describes a method for scalable communication using CUDA-aware MPI in high-order stencil computations.	Provides analysis of practical performance and factors affecting it, reports speedups up to 1.49x over baseline.	Concludes method is effective for communication in high-order stencil computations, suggests future work.
Scientific Computing Beyond CPUs: FPGA implementations	Summarizes previous studies on FPGA implementations of scientific kernels and compares with their approach.	Describes FPGA implementations of BLAS, sparse matrix operations, and FFTs using SRC MAPstation.	Analyzes advantages and challenges of using FPGAs for scientific computing, discusses use of high-level languages.
TensorFlow as a DSL for stencil-based computation on the Cerebras WSE	Cites previous work on using WSE for machine learning and computational kernels, highlights differences in methodology.	Uses TensorFlow constructs like dense and convolution layers to encode stencil-based algorithms on the WSE.	Discusses limitations and challenges of using TensorFlow for programming WSE, suggests potential improvements with Cerebras SDK.S

4. RELATED WORK

Understanding the performance disparities between Laplacian operators on CPUs and GPUs necessitates a comprehensive review of related literature. Studies such as [9] delve into the intricacies of numerical PDE solving on GPUs, offering insights into the parallel processing capabilities crucial for Laplacian operator computations. Additionally, [10] provides a thorough investigation into stencil computations across diverse architectures, including GPUs, offering methodologies applicable to comparative analyses of Laplacian operators. The significance of efficient communication mechanisms is underscored in [11], which introduces a scalable communication approach using CUDA-aware MPI, crucial for evaluating CPU and GPU Laplacian operator implementations. Moreover, [12] offers insights into GPU performance characteristics, including tensor core functionalities, vital for understanding GPU computational capabilities relative to CPUs in Laplacian operator computations.

Diving deeper into hardware acceleration and automatic code generation, studies like [13] explore optimizing finite-difference stencils on FPGAs, offering insights into alternative hardware acceleration options for Laplacian operators. Meanwhile, [14] explores compression-based optimizations to enhance efficiency in out-of-core GPU stencil computations, offering insights into memory efficiency and data transfer overheads in Laplacian operator computations. Leveraging these works alongside others in GPU methodologies and high-performance computing enables a detailed comparative analysis. For instance, [15] presents methods for efficiently solving numerical PDEs on NVIDIA GPUs, crucial for benchmarking Laplacian operator performance. Additionally, [16] presents a scalable communication approach for high-order stencil computations using CUDA-aware MPI. This framework aids in evaluating communication overheads in Laplacian operator implementations across both CPUs and GPUs.

By synthesizing insights from these studies, researchers can conduct a detailed comparative analysis of Laplacian operator implementations on CPUs and GPUs. Key factors to consider include computational performance, memory efficiency, communication overheads, and architectural features. Additionally, exploring methodologies from automatic code generation and hardware acceleration studies provides avenues for optimizing Laplacian operator performance across different computing platforms.

I. Analysis of Computational Device Performance

This study's analysis primarily compares the performance characteristics of central processing units (CPUs) and graphics processing units (GPUs) across different computational tasks. The dataset used in this research offers insights into execution time, memory usage, and throughput for various array sizes and types of computational devices.

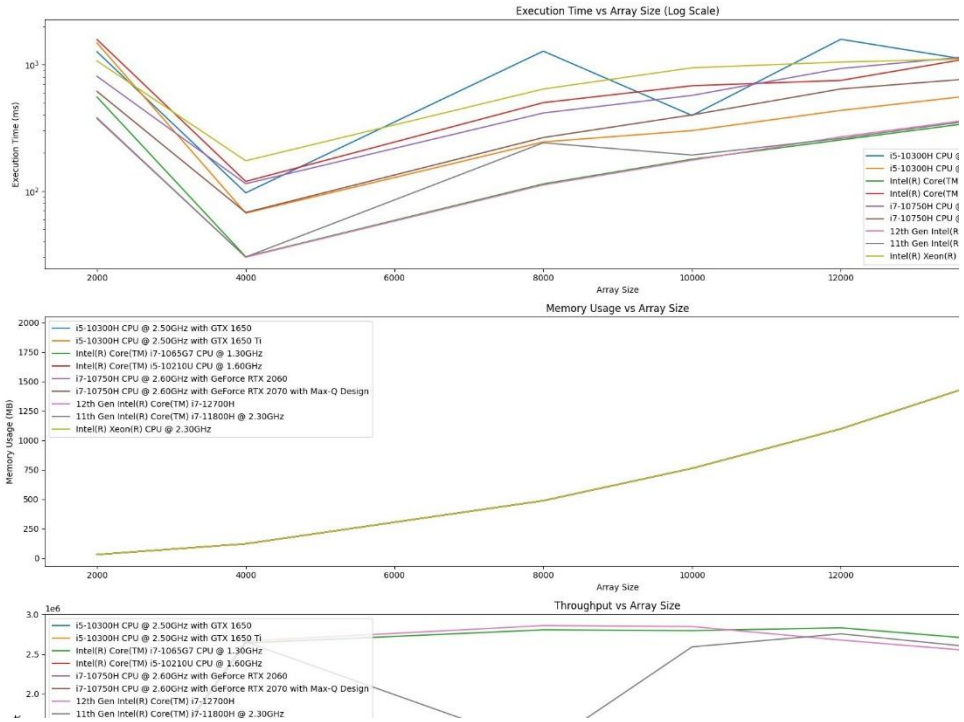

Fig. 1. CPU Performance

Our investigation reveals a consistent trend wherein GPUs demonstrate significantly lower execution times compared to CPUs across all array sizes. As the array size increases, this performance gap becomes more pronounced, highlighting the superior processing capabilities of GPUs, particularly for parallel computations. Additionally, GPUs consistently exhibit higher throughput compared to CPUs, indicating their suitability for tasks requiring high computational efficiency and parallel processing.

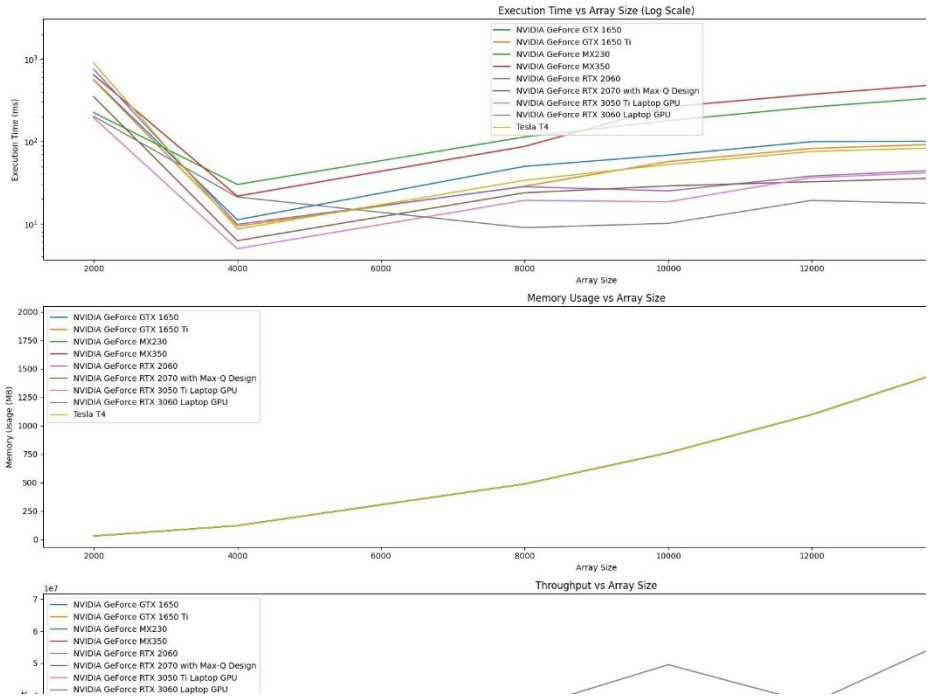

Fig. 2. GPU Performance

Surprisingly, our analysis finds no significant difference in memory usage between CPUs and GPUs across all array sizes. Both CPU and GPU exhibit similar memory utilization patterns, suggesting that memory usage may not be a distinguishing factor when choosing between these computational devices. These finding challenges conventional assumptions and underscores the need for comprehensive performance evaluations beyond execution time alone.

5. IMPLICATIONS AND CONCLUSION

TABLE 6: Device IDs and Configurations

ID	CPU	GPU
1	i5-10300H CPU @ 2.50GHz with GTX 1650	NVIDIA GeForce GTX 1650
2	i5-10300H CPU @ 2.50GHz with GTX 1650 Ti	NVIDIA GeForce GTX 1650 Ti
3	Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz	NVIDIA GeForce MX230
4	Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz	NVIDIA GeForce MX350
5	i7-10750H CPU @ 2.60GHz with GeForce RTX 2060	NVIDIA GeForce RTX 2060
6	i7-10750H CPU @ 2.60GHz with GeForce RTX 2070 with Max-Q Design	NVIDIA GeForce RTX 2070 with Max-Q Design
7	12th Gen Intel(R) Core(TM) i7-12700H	NVIDIA GeForce RTX 3050 Ti Laptop GPU
8	11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz	NVIDIA GeForce RTX 3060 Laptop GPU
9	Intel(R) Xeon(R) CPU @ 2.30GHz	Tesla T4

The findings of this analysis have significant implications for computational tasks requiring high-performance computing. GPU acceleration emerges as a compelling solution for optimizing computational performance and achieving faster execution times, particularly for parallelizable tasks. While memory usage appears comparable between CPUs and GPUs, the superior processing capabilities of GPUs make them an attractive choice for various application domains, including machine learning, scientific simulations, and data analytics.

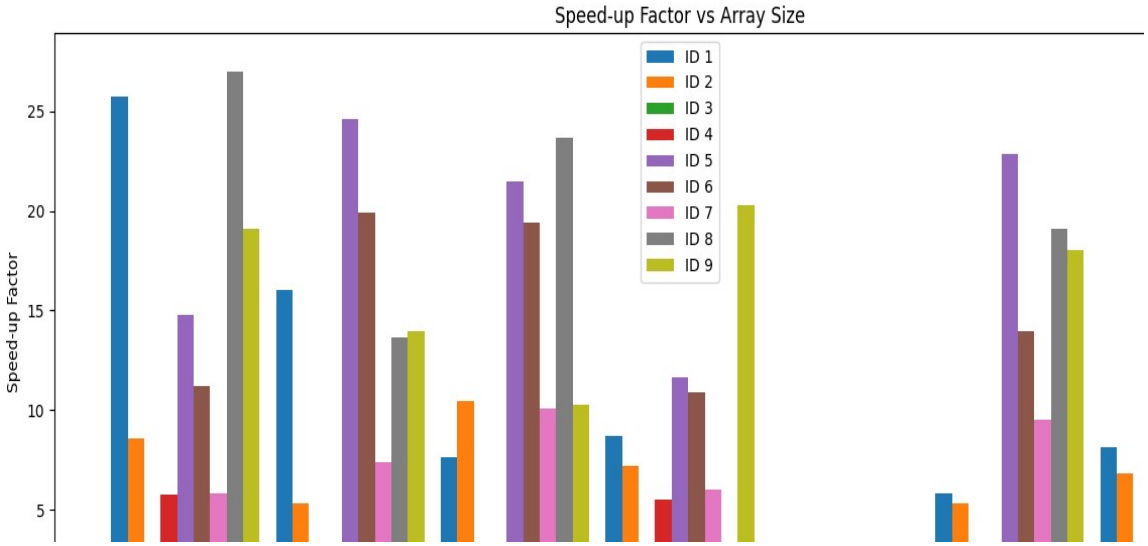

Fig. 3. Speed Up Factor

In conclusion, our analysis underscores the importance of considering GPU acceleration as a viable approach to enhance computational performance and meet the demands of modern computing tasks. Further research is warranted to explore additional factors influencing the choice between CPUs and GPUs, such as power consumption, cost-effectiveness, and hardware compatibility, to inform optimal decision-making in computational device selection.

REFERENCES

[1] Arteaga, A., Ruprecht, D., & Krause, R. (2014). A stencil-based implementation of Parareal in the C++ domain specific embedded language STELLA. *ArXiv.* https://doi.org/10.1016/j.amc.2014.12.055

[2] Bianco, M., & Varetto, U. (2012). A Generic Library for Stencil Computations. *ArXiv.* https://arxiv.org/abs/1207.1746

[3] Birke, M., Philip, B., Wang, Z., & Berrill, M. (2012). Block-Relaxation Methods for 3D Constant-Coefficient Stencils on GPUs and Multicore CPUs. *ArXiv.* https://arxiv.org/abs/1208.1975

[4] Brown, N., Echols, B., Zarins, J., & Grosser, T. (2022). TensorFlow as a DSL for stencil-based computation on the Cerebras Wafer Scale Engine. *ArXiv.* https://arxiv.org/abs/2210.04795

[5] Brown, N., Jamieson, M., Lydike, A., Bauer, E., & Grosser, T. (2023). Towards Accelerating high-order stencil computations on modern GPUs and emerging architectures using a portable framework.

[6] *ArXiv.* https://doi.org/10.1145/3624062.3624167

[7] Denzler, A., Bera, R., Hajinazar, N., Singh, G., Oliveira, G. F., & Mutlu, O. (2021). Casper: Accelerating Stencil Computation using Near-cache Processing. *ArXiv.* https://arxiv.org/abs/2112.14216

[8] Ernst, D., Holzer, M., Hager, G., Knorr, M., & Wellein, G. (2022). Analytical Performance Estimation during Code Generation on Modern GPUs. *ArXiv.* https://arxiv.org/abs/2204.14242

[9] Gloster, A. (2021). GPU Methodologies for Numerical Partial Differential Equations. *ArXiv.* https://arxiv.org/abs/2101.06550

[10] Kachris, C. (2024). A Survey on Hardware Accelerators for Large Language Models. *ArXiv.* https://arxiv.org/abs/2401.09890

[11] Kerzner, Ethan, and Timothy Urness. "GPU Programming for Mathematical and Scientific Computing."

[12] *Drake University* (2010).

[13] Luo, W., Fan, R., Li, Z., Du, D., Wang, Q., & Chu, X. (2024). Benchmarking and Dissecting the Nvidia Hopper GPU Architecture. *ArXiv.* https://arxiv.org/abs/2402.13499

[14] Matsumura, K., Zohouri, H. R., Wahib, M., Endo, T., & Matsuoka, S. (2020). AN5D: Automated Stencil Framework for High-Degree Temporal Blocking on GPUs. *ArXiv.* https://doi.org/10.1145/3368826.3377904

[15] Mayer, F., Brandner, J., & Philippsen, M. (2024). Utilizing polyhedral methods to optimize stencil computations on FPGAs, incorporating stencil-specific caches, data reuse strategies, and wide data bursts. *ArXiv.* https://arxiv.org/abs/2401.13645

[16] Omlin, S., & Räss, L. (2022). High-performance xPU Stencil Computations in Julia. *ArXiv.* https://arxiv.org/abs/2211.15634

[17] Omlin, S., Räss, L., & Utkin, I. (2022). Distributed Parallelization of xPU Stencil Computations in Julia.

[18] *ArXiv.* https://arxiv.org/abs/2211.15716

[19] Paredes, E. G., Groner, L., Ubbiali, S., Vogt, H., Madonna, A., Mariotti, K., Cruz, F., Benedicic, L., Bianco, M., VandeVondele, J., & Schulthess, T. C. (2023). GT4Py: Python-based high-performance stencil computations tailored for weather and climate applications. *ArXiv.* https://arxiv.org/abs/2311.08322

[20] Pekkilä, J., Väisälä, M. S., Käpylä, M. J., Rheinhardt, M., & Lappi, O. (2021). Implementing scalable communication techniques for high-order stencil computations by leveraging CUDA-aware MPI.

[21] *ArXiv.* https://doi.org/10.1016/j.parco.2022.102904

[22] Quezada, F. A., & Navarro, C. A. (2021). Accelerating Compact Fractals with Tensor Core GPUs.

[23] *ArXiv.* https://arxiv.org/abs/2110.12952

[24] Reguly, I. Z., Mudalige, G. R., & Giles, M. B. (2017). Exploring out-of-core stencil computations beyond the limitations of 16GB memory. *ArXiv.* https://arxiv.org/abs/1709.02125

[25] Rodrigues, V. H., Cavalcante, L., Pereira, M. B., Luporini, F., Reguly, I., Gorman, G., & De Souza, S. X. (2019). GPU Support for Automatic Generation of Finite-Differences Stencil Kernels. *ArXiv.* $https://doi.org/10.1007/978-3-030-41005-6_16$

[26] Sai, R., & Xu, J. (2023). Towards Accelerating High-Order Stencils on Modern GPUs and Emerging Architectures with a Portable Framework. *ArXiv.* https://arxiv.org/abs/2309.04671

[27] Seznec, Mickael, et al. "Computing large 2D convolutions on GPU efficiently with the im2tensor algorithm." *Journal of Real-Time Image Processing* 19.6 (2022): 1035-1047.

[28] Shen, J., Deng, X., Wu, Y., Okita, M., & Ino, F. (2022). Compression-Based Optimizations for Out-of-Core GPU Stencil Computation. *ArXiv.* https://arxiv.org/abs/2204.11315

[29] Shen, J., Long, L., Zhang, J., Shen, W., Okita, M., & Ino, F. (2023). A Synergy between On- and Off-Chip Data Reuse for GPU-based Out-of-Core Stencil Computation. *ArXiv.* https://arxiv.org/abs/2309.08864

[30] Shen, J., Wu, Y., Okita, M., & Ino, F. (2021). Accelerating GPU

[31] 26.-Based Out-of-Core Stencil Computation with On-the-Fly Compression. *ArXiv.* https://arxiv.org/abs/2109.05410

[32] Smith, Melissa C., Jeffery S. Vetter, and Sadaf R. Alam. "Scientific computing beyond CPUs: FPGA implementations of common scientific kernels." *2005 MAPLD International Conference.* 2005.

[33] Yang, J., Giannoula, C., Wu, J., Elhoushi, M., Gleeson, J., & Pekhimenko, G. (2023). Minuet: Accelerating 3D Sparse Convolutions on GPUs. *ArXiv.* https://arxiv.org/abs/2401.06145

[34] Zhang, L., M., Wahib, P., Chen, J., Meng, X., Wang, T., Endo, & Matsuoka, S. (2023). Exploiting Scratchpad Memory for Deep Temporal Blocking: A case study for 2D Jacobian 5-point iterative stencil kernel (j2d5pt). *ArXiv.* https://doi.org/10.1145/3589236.3589242

[35] Zhang, L., M., Wahib, P., Chen, J., Meng, X., Wang, T., Endo, & Matsuoka, S. (2023). Revisiting Temporal Blocking Stencil Optimizations. *ArXiv.* https://doi.org/10.1145/3577193.3593716

[36] Zohouri, H. R., Podobas, A., & Matsuoka, S. (2020). High-Performance High-Order Stencil Computation on FPGAs Using OpenCL. *ArXiv.* https://doi.org/10.1109/IPDPSW.2018.00027