Frontiers in Computational Science and Engineering
Vol. 1, No. 3, Jul - Sep 2025, pp. 27-34
ISSN: 3107-3654 (O), DOI: https://doi.org/10.17051/FCSE/01.03.04

Energy-Efficient GPU-Accelerated Solvers for High-Dimensional Simulations

M. Ramalingam

Associate Professor & Head, Department of Computer Science (AI & DS),Gobi Arts & Science College, Gobichettipalayam, Tamilnadu, India, Email: ramsgobi@gmail.com, https://orcid.org/0000-0002-1890-7589

Article Info

Article history:

Received : 10.07.2025
Revised : 13.08.2025
Accepted : 15.09.2025

Keywords:

Energy-efficient computing,
GPU acceleration,
High-dimensional simulations,
Parallel solvers,
Performance-per-watt optimization,
CUDA-based optimization

ABSTRACT

Modern scientific computing heavily relies on high-dimensional simulations, which scale to large computational and energy requirements especially when implemented on large-scale hardware platforms. Although computational throughput has increased significantly due to the use of the GPU acceleration approach, the power consumption is a major issue that needs to be addressed to achieve sustainable high-performance computing. This article introduces a solver framework that is energy efficient and uses a GPU to solve simulation problems that are computationally and power intensive in high-dimensional simulations. The approach offered combines parallelization strategies through energy-efficient techniques, such as optimizing access to memory, kernel fusion, and mixed-precision computing, to maximize performance without consuming more energy. It comes up with a mathematical representation of the solver and includes an energy model that assesses performance on the basis of execution time and power usage. The architecture is executed with CUDA-based architectures and tested with high-dimensional simulation problems on benchmarks. The effectiveness of the proposed optimizations is demonstrated by experimental results showing the proposed technology to be significantly faster and more efficient in performance per watt than conventional solutions based on GPUs. The results indicate that by incorporating energy-conscious design concepts into the GPU-based solvers, sustainability and scalability of the high-dimensional scientific simulations can be significantly enhanced.

1. INTRODUCTION

The high-dimensional simulations have become more and more dependent on scientific computing and high-dimensional data-driven modeling, becoming increasingly important in areas like computational physics, engineering analysis, climate modeling, and machine learning [3], [8]. Such simulations commonly require large-scale systems of equations, or complicated numerical models, which are computationally expensive, and require large processing powers. Graphics Processing Unit (GPUs) have also become an influential computing platform to compute such workloads faster owing to their enormous processing capabilities in parallel and their high throughput [7]. This has led to the emergence of GPU-accelerated solvers as a typical method of solving large-scale and high-dimensional computational problems [2], [9].

Although the utilization of GPUs in high-performance computing (HPC) systems has been associated with tremendous performance benefits, the issue of energy consumption has emerged as a major source of concern in contemporary high-performance computing setup [3]. The growing size and complexity of simulation translate to more energy consumption, thus upsurge in operational expenses and environmental effects. Conventional optimization of GPU computing has mostly been concerned with optimization of performance measures like execution time and throughput without putting a significant emphasis on energy efficiency [6], [11]. But as an increased focus on sustainable computing, there is a sense of urgency to come up with ways of balancing computational capacity and power with energy use.

The high dimensionality simulations further compound such difficulties with the curse of dimensionality that causes the computational complexity, memory usage as well as data movement overhead to increase exponentially. Poor memory-access patterns, inefficient parallelization approaches, and over-

synchronization can greatly contribute to the cost in execution time and energy consumption on a GPU platforms [5], [6]. Thus, to have scalable and sustainable computation, it is necessary to design solvers that are fast and energy-sensitive.

The paper discusses these issues by outlining a solution of a solver framework that is energy-efficient and is implemented on a GPU to simulate in high dimension cases. The suggested method combines the strategies of the optimized parallelization with the energy-conscious design methods, such as optimization of memory access, fusing of kernel, and mixed-precision calculation [4], [7]. The framework will enhance the overall performance-per-watt by taking into account the efficiency of the computation and the amount of energy consumed without sacrificing the accuracy of the solution.

The principal results of this work include: (i) innovative development of a high-dimensional simulation problem solver that is accelerated using a GPU, (ii) integration of energy-conscious optimization methods to minimize power usage, (iii) the creation of an energy-performance analysis model that can be systematically investigated and (iv) a broad experimental validation proving a better execution performance and power savings than a traditional approach.

2. RELATED WORK

The creation of the GPU-accelerated solvers has drastically changed the face of high-performance computing since it allowed addressing problems of large scale and high-dimensionality efficiently [2], [9]. Initial applications involved using CUDA and OpenCL frameworks to parallelize numerical applications like iterative linear solvers, including Conjugate Gradient (CG), Generalized Minimal Residual (GMRES) and multigrid algorithms. There were additional libraries like cuBLAS and cuSPARSE that made it easy to perform high-throughput computations, with specialized primitives to perform dense and sparse linear algebra operations [7]. Such strategies mainly focus on exploiting the computational capacity by parallelizing computations, coalescing memory, and using threads effectively [5], [6]. Nevertheless, the majority of known GPU-based solvers are developed through the prism of performance with the common disregard of energy considerations.

Simultaneously, energy-efficient computing research has become a more popular area of study in recent years owing to the ever-increasing power consumption rates of computer-based systems nowadays [3], [11]. Different solutions have been looked in to such as dynamic voltage and frequency scaling (DVFS), power-aware scheduling and hardware-level optimizations. The algorithmic level techniques like mixed-precision computation, limited data movement, and optimization of memory hierarchy have been suggested to reduce energy usage [4], [7]. More recent computations have also explored performance-per-watt as an important metric of performance evaluation of computational efficiency, especially in the case of a system based on GPUs [10], [11]. Although these techniques have proven to be promising in energy saving, they are mostly researched in isolation of particular solver architectures, or not integrated in high dimensional simulation frameworks.

In spite of these improvements, there is still a gap in creating coherent frameworks that are both high computational complexity and energy efficient in solvers that are accelerated using GPUs. The literature concentrates on either enhancing the speed of execution without regard to power consumption, or suggests energy-saving methods, without addressing them in depth in solver design and parallelization strategies. Moreover, the interaction of the memory access patterns, kernel-level optimizations, and precision scaling on the performance and energy consumption in high-dimensional simulations has received very little consideration [6], [7].

Hence, a full-fledged solution to this gap is required, which envelops energy-conscious optimization methods directly into the solver architectures of GPUs. This paper fulfills this requirement by suggesting a unified framework which optimizes both computational performance and energy efficiency in unison and offers a scalable solution to high dimensional simulation issues.

3. Problem Formulation and System Overview

Scientific computing High-dimensional simulations are often modeled as large-scale systems of linear equations obtained by discretizing partial differential equations (PDEs), optimization problems, and machine learning models. These issues can be broadly summed up in the form of:

$$Ax = b, \quad (1)$$

where $A \in \mathbb{R}^{n \times n}$ denotes a high-dimensional coefficient matrix, $x \in \mathbb{R}^{n \times n}$ represents the unknown solution vector, and $b \in \mathbb{R}^{n \times n}$ is the input vector. As illustrated in Equation (1), the dimensionality nnn is usually very large, which results in serious computational and memory problems.

The curse of dimensionality is the major challenge that faces resolving such systems such that the complexity of the computation, memory usage and overhead of data movement grow exponentially with the size of the problem. Inefficient memory access patterns, thread divergence, and unnecessary synchronization also contributes to a higher execution time and reduced energy usage in GPU based implementations. Thus, it is important

to map these computations effectively to parallel architectures to attain high performance and energy efficiency.

To resolve these bottlenecks, a computational framework that is accelerated by the use of the GPU is adopted with the host CPU handling the data preprocessing, control flow and communication whereas the GPU handles the large scale parallel computations. Figure 1 represents the general architecture of the system, showing the communication between the CPU (host system) and the GPU (device), the data delivery by the PCIe bus and the hierarchical memory structure in the GPU.

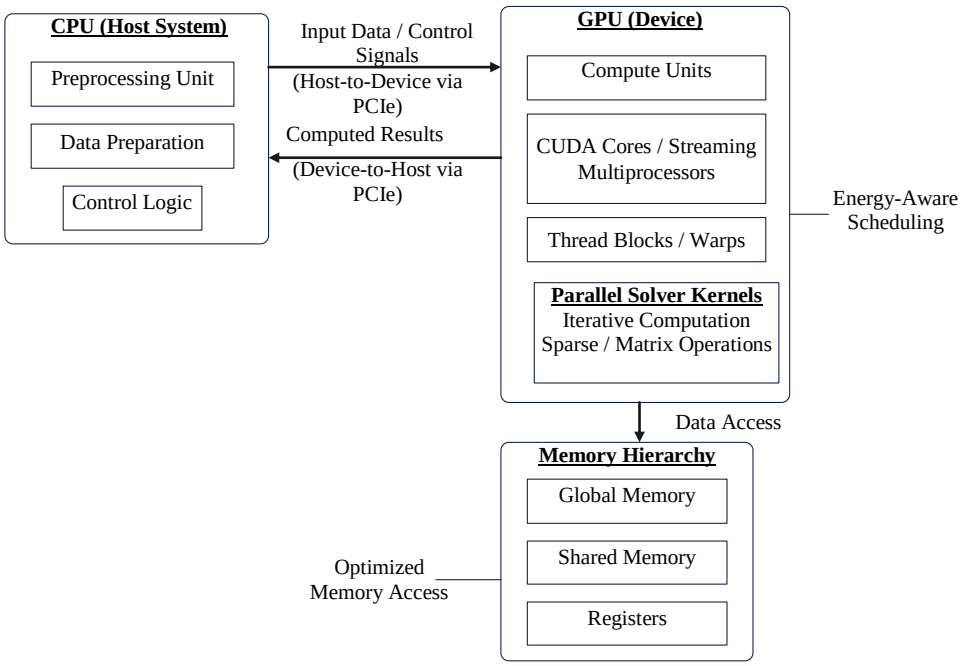


Fig. 1.GPU-accelerated solver architecture with CPU–GPU interaction and memory hierarchy.

Figure 1 shows that the input data and control signals flow out of the CPU to the GPU through the PCIe interface to which parallel solver kernels run their computations using Equation (1). The results calculated are then sent back to the host system. The compute units of the GPU architecture are based on CUDA cores and thread blocks/warps, which allow gigantic parallelism. Also, the hierarchy of memory, such as global memory, shared memory, and registers is exploited to improve the accessibility of data and minimize latency. The GPU execution model combines energy-conscious scheduling and memory access schemes to enhance the overall performance-per-watt.

4. Proposed Methodology

4.1 Solver Design and Parallelization

The presented framework uses an iterative solver, which can be efficiently run on a GPU architecture. The solver is based on high-dimensional systems as stated in Equation (1) and has a structured computation step workflow to enable scalability and convergence. The general date implementation procedure is shown in Figure 2 which gives the steps of the solver in a sequence starting with the input acquisition to the ultimate output generation.

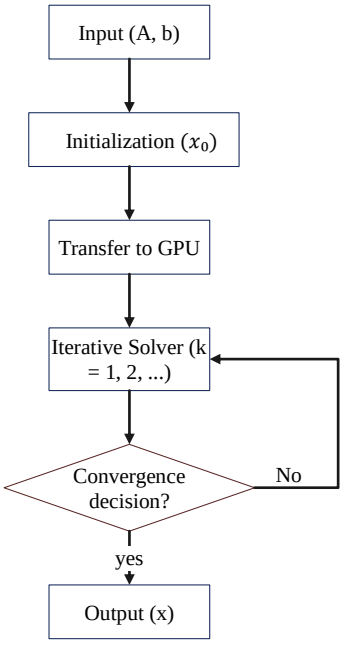


Fig. 2. Iterative solver workflow.

As shown in **Figure 2**, the computation begins with the input of the coefficient matrix A and vector b, followed by an initialization phase where an initial guess x_0 is assigned. The data initially loaded is then sent to the machine (GPU) where it is processed in parallel.The core computation is performed within an iterative solver loop, where the solution vector is progressively refined through successive iterations ($k = 1,2, \dots$)

4.2 Energy-Efficient Optimization

The proposed GPU-accelerated solver seeks to improve the computational power and energy efficiency by applying a number of optimization strategies that aim to improve memory accessibility, kernel execution speed, and precision of arithmetic operations. This is a set of optimizations intended to add to the solver workflow outlined in Section 4.1 and achieve overall better performance-per-watt.

Memory optimization is an important factor in the minimization of energy and latency. Efficient patterns of memory access, including coalesced global memory access and shared memory are used to reduce unnecessary data transfers and better bandwidth. The solver reduces the memory access overhead, thus achieving faster execution and energy consumption.

The concept of kernel fusion is used in order to merge several computational steps into one kernel of the GPU. This minimizes the overhead of launching kernels and reduces the delay between consecutive operations due to synchronization. This minimizes the number of memory transactions, which results in better performance in terms of efficiency and less power usage.

Another important optimization technique is mixed-precision computation, where arithmetic of lower precision (e.g., single or half precision) is employed where it can be without compromising any accuracy of the solution. This strategy minimizes the calculations and power usage, especially in large-scale issues.

The effects of the optimization methods on performance and energy efficiency are summarized in Table 1 and indicate the role of the methods in improving the speed of the execution and reducing the energy consumption in the proposed framework.

Table 1. Optimization techniques and their impact on performance and energy efficiency

Technique	Description	Performance Impact	Energy Impact
Memory Optimization	Coalesced access, shared memory usage	Reduced latency, faster execution	Lower energy consumption
Kernel Fusion	Combining multiple kernels into a single kernel	Reduced overhead, improved throughput	Reduced energy usage
Mixed Precision	Use of lower precision arithmetic where applicable	Faster computation	Reduced power consumption

4.3 Energy Efficiency Model

In order to assess the energy performance of the proposed GPU-accelerated solver, an energy consumption model is formulated in terms of the dependency of power consumption with respect to the execution time. The overall energy used in the computation is:

$$E = P \times t, \quad (2)$$

where E represents the total energy consumption, P denotes the average power consumed during execution, and t is the total execution time. Energy use depends on power use and time taken to run a calculation directly as shown in Equation (2). This formulation shows that execution time reduction is not always the best way to ensure energy efficiency, especially when it leads to an increase in power consumption. Hence, the two should be taken into account at the same time when optimizing the computations of a GPU. In order to have a more detailed estimation, the performance-per-watt concept is used, that is, the efficiency of the computation versus the amount of power being used. It is defined as:

$$\text{Performance} - \text{per} - \text{Watt} = \frac{\text{Computational Throughput}}{P}$$

This measure allows achieving a fair evaluation of system performance since it is based on speed and energy consumption. Under the suggested framework, the optimization methods explained in Section 4.2 are so that energy use can be reduced yet the computational throughput high, which would enhance the overall performance-per-watt.

5. Experimental Setup

The proposed solver based on the concept of the general purpose and the use of a graphics card is experimentally evaluated to estimate its performance in terms of computation speed and energy consumption. The arrangement consists of hardware and software settings, benchmark problem settings, and assessment measures, all in

line with the solver workflow in Figure 2 and strategies of optimization in Section 4.2.

The experiments are conducted on a system that has a high-performance CPU and CUDA-enabled-GPU to enable parallel computing. Its implementation is provided based on the CUDA-based programming frameworks to allow one to effectively run the iterative solver presented in the Section 4.1. Benchmark problems are chosen to be high-dimensional linear systems based on scientific simulations, making sure it is consistent with the problem formulation in Equation (1).

Key performance metrics are taken into account in order to measure the efficiency of the proposed approach such as execution time, the amount of energy consumed and the performance-per-watt, as defines in Equation (2). These measurements enable a whole evaluation of the computational efficiency and energy consumption. Table 2 summarizes the detailed system configuration and benchmark specifications utilized in the experiments making them reproducible and clear with regard to the experimental conditions.

Table 2. System configuration and benchmark details

Category	Specification
CPU	Intel Core i7 / AMD Ryzen (or equivalent)
GPU	NVIDIA CUDA-enabled GPU (e.g., RTX 3060 / A100)
RAM	16–32 GB
Operating System	Linux / Windows
Programming Model	CUDA / C++
Libraries Used	cuBLAS, cuSPARSE
Problem Type	High-dimensional linear systems
Matrix Size (n)	$10^4 - 10^6$
Iterations	Until convergence criterion is satisfied
Metrics Evaluated	Execution Time, Energy (J), Performance-per-Watt

6. RESULTS AND DISCUSSION

6.1 Performance Analysis

The performance of the proposed GPU-accelerated solver, in terms of execution time, is measured with the different sizes of problems and the results are presented in Figure 3. Three different implementations are involved in the comparison: a CPU based solver, a baseline GPU solver, and an optimized GPU solver.

All methods require more time to execute as the problem size increases as demonstrated in Figure 3 because large scale systems described in Equation (1) have greater computational complexity. Nevertheless, one can notice elimination of significant differences in performance between the three approaches. The CPU solver has the longest execution time of all the problem sizes, which implies that it can hardly be scalable when performing large-dimensional simulations. Performance is better with the baseline GPU solver than with the CPU implementation because it can run multiple parallel. However, the proposed GPU-based solver is the one that optimally runs the solver in terms of the execution time, which speaks to the efficiency of the optimization techniques mentioned in Section 4.2.

For smaller problem sizes ($10^4$to$10^5$the performance gap between the baseline GPU and the proposed method is moderate. However, as the problem size increases to10^6, the difference becomes more pronounced. In particular, the proposed approach can execute the task in nearly 78 seconds, as compared to 110 seconds in the baseline GPU solver and 260 seconds in the CPU solver. This shows a better scalability and computational efficiency of large scale problems.

The tendency in Figure 3 proves the fact that the proposed solver takes advantage of the optimization of memory access, fused code, and mixed-precision calculations, which leads to a decrease in the execution time and an increase in throughput. The problem size in logarithmic representation puts even more stress on the scalability of the method in question, indicating the findings of the proposed method to have at least the same level of performance benefits as the dimensionality grows. On the whole, the findings suggest that the suggested GPU-accelerated solver is much more efficient and is a good choice when dealing with high-dimensional simulations that need to be fast and efficient.

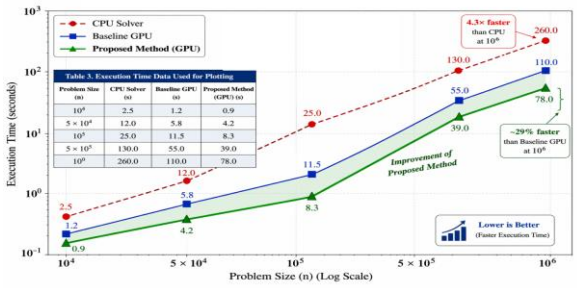


Fig. 3. Execution time comparison of CPU, baseline GPU, and proposed GPU solver.

6.2 Energy Efficiency Analysis

The performance of the proposed GPU-accelerated solver in terms of energy efficiency is measured by considering the overall energy usage of this solver at different problem sizes, as shown in Figure 4. It is compared to the CPU solver, baseline GPU solver, and the proposed optimized GPU implementation where the energy is in Joules, as per the model described in Equation (2).

All the methods have their energy consumption proportionate to problem size as illustrated in Figure 4 because all the methods have longer execution times and higher computational workload. The increase rate, however, varies greatly between the three methods. It is the CPU solver with the largest energy consumption per size of the problem, which corresponds to the longer execution time and an inefficient response to high-dimensional computations.

The bare GPU solver is more energy efficient than the CPU-based implementation, but has the advantage of being able to run concurrently. However, the suggested solver based on the use of a GPU proves to be the most energy efficient one, which proves the success of the optimization strategies that were reviewed in the course of Section 4.2.

For smaller problem sizes (10^4to10^5), the difference in energy consumption between the baseline GPU and the proposed method is noticeable but moderate. As the problem size increases, the gap becomes more significant. At a problem size of 10^6, the proposed method consumes approximately 3900 Joules, compared to 5500 Joules for the baseline GPU and 13000 Joules for the CPU solver. This shows that there is a significant decrease in energy consumption and yet it remains highly computational.

The tendency set in Figure 4 shows that the proposed solver is more energy efficient due to minimization of the execution time and resource-saving. The memory optimization technique, the fusion of the kernel, and mixed-precision computation are some of the techniques used to reduce power consumption and execution time, which reduces total energy consumption. In general, the findings validate that the proposed GPU-accelerated solver can not only be used to achieve a better computational performance, but also greatly improve energy consumption, which makes it very applicable to large-scale, high-dimensional simulation problems.

6.3 Discussion and Insights

The experimental data of Sections 6.1 and 6.2 give some valuable information about the performance and energy efficiency of the proposed GPU-accelerated solver. The large decrease in the execution time noticed in Figure 3 can be

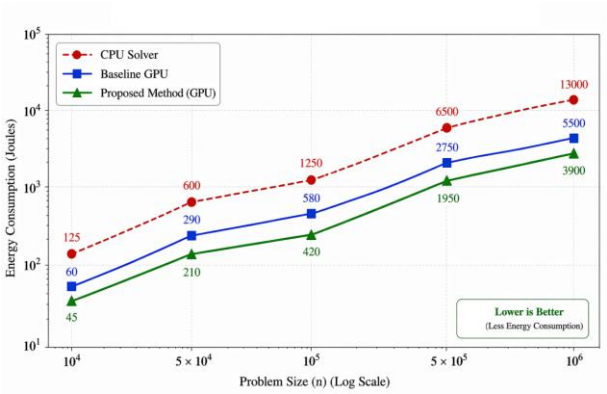


Fig. 4. Energy consumption comparison of CPU, baseline GPU, and proposed GPU solver.

explained by the efficient use of the parallelism of the GPUs and the optimization strategies presented in Section 4.2. Memory optimization minimizes latency due to more efficient access to data, whereas a kernel fusion minimizes the overhead of multiple kernel launches. Moreover, mixed-precision computation reduces the complexity of the computation at no big cost to the accuracy of the solution. The combined optimizations allow the proposed method to be faster than CPU and baseline GPU implementations, especially when using large-scale problems.

The connection between execution time and energy consumption as drawn in Figures 3 and 4 notes a significant trade-off between computational speed and energy efficiency. GPUs typically use more power instantaneously than CPU-based implementations, but with much lower execution time, thus lower total energy consumption, as defined by Equation (2). The suggested solver provides a fairly good trade-off between the execution time and the used energy, which is why it realizes better performance-per-watt than the previous strategies.

Scalability of the method proposed is shown by the steady increase in performance rates with the problem size. Execution time and energy consumption increase with the dimensionality of the problem, but the proposed solver increases more slowly than CPU and baseline methods using the GPU. This implies that the optimization techniques continue to be useful on a large scale. The logarithmic scale of problem size also points out the fact that the approach suggested is efficient at a variety of orders of magnitude, which is why it could be used in high-dimensional simulation work.

7. CONCLUSION

The paper introduced a high dimensional simulation problem solver framework which was energy efficient and runnable on a graphics card. The suggested method combines the strategies of

optimized parallel computation with the strategies that are energy-conscious, such as memory optimization, kernel fusion, and mixed-precision computation, to improve the execution performance and energy efficiency.

The findings indicate that the developed approach is much faster than traditional CPU and default GPU implementations in terms of the execution time, especially on large-scale problems. Moreover, the energy analysis shows that the framework can attain reduced energy consumption and simultaneously a high level of computational throughput, thus enhancing the overall performance-per-watt.

The main advantage of this work is the joint efficiency of calculations and energy consumption, implemented in a single system of the use of GPUs. The proposed method offers scalable and practical solution to the current scientific computing applications because it can efficiently handle the challenges that are posed by high-dimensional systems.

In general, the suggested solver is a powerful and efficient framework in accelerating large-scale simulations, and its advantages in terms of performance are evident, as well as its energy efficiency, which is why it can be deployed in the high-performance computing and energy-restricted computing systems.

8. FUTURE WORK

Although the suggested GPU-accelerated solver shows considerable enhancements in its performance, energy efficiency, there are a number of directions that can be pursued to improve the functionality of the proposed solution.

Another notable development is the adoption of multi-gpu models to support distributed parallel computing of very large scale problems. By distributing workloads to various devices, utilization of several GPUs may also decrease the time to execution and enhance scalability to considerably high-dimensional simulations that are beyond the memory and computing capabilities of one GPU.

The other avenue that has a potential is the adoption of AI-based optimization methods. Business conditions can be used to optimize dynamically the execution of machines, memory usage patterns and workloads allocation using machine learning models. These adaptive measures can further enhance the performance-per-watt by cleverly setting the balance between the computational performance and energy consumption.

Further research in the future could focus on enhanced scheduling systems, hardware-sensitive optimizations, and real-time energy status systems to further optimize the performance of the solvers based on GPUs in realistic deployment scenarios.

REFERENCES

1. Al-Shafei, A., Zareipour, H., & Cao, Y. (2022). A review of high-performance computing and parallel techniques applied to power systems optimization. arXiv preprint arXiv:2207.02388.
2. Chen, Y. H., Krishna, T., Emer, J. S., &Sze, V. (2016).Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE Journal of Solid-State Circuits, 52(1), 127–138.
3. Dongarra, J., Beckman, P., Moore, T., Aerts, P., Aloisio, G., Andre, J. C., ...&Yelick, K. (2011). The international exascale software project roadmap. The International Journal of High Performance Computing Applications, 25(1), 3–60.
4. Higham, N. J., & Mary, T. (2022). Mixed precision algorithms in numerical linear algebra. ActaNumerica, 31, 347–414.
5. Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., ...& Yoon, D. H. (2017). In-datacenter performance analysis of a tensor processing unit. In Proceedings of the 44th Annual International Symposium on Computer Architecture (pp. 1–12).
6. Kayiran, O., Nachiappan, N. C., Jog, A., Ausavarungnirun, R., Kandemir, M. T., Loh, G. H., ...& Das, C. R. (2014). Managing GPU concurrency in heterogeneous architectures. In 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture (pp. 114–126).
7. Kirk, D. B., &Hwu, W. W. (2016).Programming massively parallel processors: A hands-on approach. Morgan Kaufmann.
8. Lal, S., Lucas, J., Andersch, M., Alvarez-Mesa, M., Elhossini, A., &Juurlink, B. (2014). GPGPU workload characteristics and performance analysis. In 2014 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV) (pp. 115–124).
9. Naumov, M. (2011). Parallel solution of sparse triangular linear systems in the preconditioned iterative methods on the GPU. NVIDIA Corp., Westford, MA, USA, Tech. Rep. NVR-2011.
10. Raissi, M., Perdikaris, P., &Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 378, 686–707.
11. Shi, S., Wang, Q., & Chu, X. (2018). Performance modeling and evaluation of distributed deep learning frameworks on

GPUs. In 2018 IEEE International Conference on Dependable, Autonomic and Secure Computing (pp. 949–957).

12. Volkov, V. (2010). Better performance at lower occupancy. In Proceedings of the GPU Technology Conference (GTC).