Design of an Interactive GPU-Accelerated Visualization Tool for Teaching Parallel Algorithms

Tanjila B
Department of Computer Science and Engineering
G M Institute Of Technology, Davanagere, India
Email: ballaritanzeela26@gmail.com

Deepti H G
Department of Computer Science and Engineering
G M Institute Of Technology, Davanagere, India
Email: deeptihg.fet.scst@gmu.ac.in

***Abstract*—Parallel and Distributed Computing (PDC) is foundational in computer science education, yet students often struggle to develop intuition for complex parallel algorithmic concepts. We present an interactive GPU-accelerated web visualization tool designed to significantly enhance the teaching and learning of parallel algorithms. Built using WebGL and modern browser technologies, the tool provides visual, real-time demonstrations of two fundamental parallelism paradigms: data parallelism, exemplified through matrix multiplication, and task parallelism, illustrated by parallel merge sort. Its highly configurable interface enables in-depth exploration of algorithmic structure, thread utilization patterns, inter-thread communication, and observable performance metrics. This work contributes an accessible, browser-based framework that substantially aids educators in illustrating key PDC concepts, directly addressing documented gaps in existing pedagogical resources by fostering a more intuitive and engaging learning experience.**

***Index Terms*—Parallel Computing, GPU Acceleration, Visualization, Education, WebGL, Data Parallelism, Task Parallelism, Interactive Learning, Pedagogical Tool**

I. Introduction

The ubiquitous presence of multi-core processors and Graphics Processing Units (GPUs) has made parallel processing a cornerstone of modern computer science. From scientific simulations to AI advancements, the ability to design and optimize parallel algorithms is now an indispensable skill. Consequently, a strong understanding of Parallel and Distributed Computing (PDC) principles is a fundamental requirement in computer science curricula.

However, teaching and learning parallel algorithms present unique challenges. Concepts like concurrent execution, thread synchronization, race conditions, deadlocks, and efficient communication are abstract and complex. Traditional methods—static diagrams, pseudocode, or theoretical lectures—often fail to adequately represent the dynamic nature of parallel computations. This can lead to a superficial understanding, hindering problem-solving and engagement with advanced topics.

Interactive visualization offers a powerful solution by transforming abstract processes into dynamic, observable phenomena. This approach significantly enhances engagement, improves comprehension of algorithmic flow, illustrates real-time data transformations, and reveals computation states. For parallel computing, visualization uniquely allows "seeing" concurrency, observing independent and interdependent thread behaviors, highlighting synchronization points, and exposing performance bottlenecks. This experiential learning empowers students to actively explore, experiment, and directly observe the consequences of different parallelization strategies.

Recent advancements in web technologies, particularly WebGL (Web Graphics Library), provide direct, high-performance access to a computer's GPU via JavaScript within any modern web browser. This enables complex graphical rendering and significant computational offloading directly on the client-side, without specialized software. This ubiquity and accessibility make WebGL an ideal platform for sophisticated educational tools that can be deployed globally.

The presented tool is publicly available for demonstration and classroom use at: https://github.com/Tanjila-01/gpu-visualizer.

This paper introduces a novel, web-based, GPU-accelerated visualization tool designed to address these pedagogical challenges. Our tool provides dynamic, interactive demonstrations of two foundational parallelism paradigms:

- **Data Parallelism:** Illustrated through matrix multiplication, showcasing concurrent operations on distinct data elements.
- **Task Parallelism:** Demonstrated using parallel merge sort, highlighting recursive problem decomposition and crucial coordination among concurrent tasks.

The tool's highly configurable interface allows dynamic adjustment of parameters like thread count, input data size, and simulation speed. This enables in-depth exploration of algorithmic structure, thread utilization, synchronization mechanisms, and real-time performance metrics. By offering an accessible, browser-based framework, this work provides a potent resource for educators, enabling vivid illustration of key PDC concepts and fostering self-directed, experiential learning.

II. Objectives and Design Goals

The primary goal was to demystify parallel algorithms, transforming abstract concepts into tangible processes. Key objectives guided the design:

- **Comprehensive Paradigm Demonstration:** Visualize both data parallelism (matrix multiplication) and task parallelism (parallel merge sort) for comparative understanding.

- **Configurable Parameters:** Allow dynamic control over thread count, input data size, and animation speed to explore their impact on algorithmic behavior and performance.
- **Visualization of Core Concepts:** Visually represent independent operations, recursive decomposition, inter-thread synchronization, communication overhead, and workload distribution.
- **Real-Time Interactive Exploration:** Enable pausing, stepping through, and replaying simulations for meticulous analysis of complex interactions.
- **Comparative Performance Analysis:** Offer quantitative metrics, including simulated serial vs. parallel execution times and theoretical speedup (Amdahl's Law), to complement visual understanding.

The design prioritized a user-friendly interface integrated with robust GPU-accelerated rendering, delivering a powerful yet approachable educational resource.

III. Related Work

Literature on parallel visualization and performance spans algorithm design, architectural strategies, and educational interfaces. Recent studies present WebGL-based matrix optimization algorithms facilitating real-time construction and rendering of 3D visualization models, with an emphasis on computational efficiency and sparse matrix handling [8]. Classic work by Waheed and Rover investigated graphical performance visualization for parallel programs, laying the groundwork for modern debugging and analysis methods in visual computing [2]. GPU-enabled computation and geometric visualization have expanded into scientific domains, such as GPU-parallel Morse-Smale complexes for extracting features from vast topological datasets [3]. Image smoothing and filtering algorithms have been parallelized on both CPUs and GPUs, quantitatively evaluating thread efficiency and performance in high-performance computing scenarios [4]. Educational resources now offer comprehensive coverage of concurrency, architecture, and parallel methodology for both multicore CPUs and GPUs [5]. Analysis of Java APIs and task-level parallelism demonstrates efficacy and scalability on many-core systems [6]. Research on thread-level parallel sorting and mapping algorithms for multi-core computers has improved strategies for workload partitioning and multi-level cache utilization [7]. Prototype development for visual parallel programming platforms facilitates rapid deployment and visualization of parallel algorithms, messaging, and heterogeneous cluster workflows [1]. Our system synthesizes these lines of research, leveraging current GPU/browser technologies to deliver interactive, educational visualizations for parallel algorithm learning and experimentation in web environments.

IV. System Design and Implementation

The interactive visualization tool employs a modular architecture with three primary components: a visualization engine, a computation simulator, and a metrics panel. This modularity ensures clear separation of concerns, simplifying development and facilitating future expansions.

A. Visualization Engine

The visualization engine is the core graphical component, responsible for rendering dynamic algorithm behavior. It uses WebGL to leverage the client-side GPU, ensuring high-performance, real-time animation, and fluid interactivity. GPU shaders are extensively used to highlight critical aspects:

- **Computational Threads:** Represented by distinct colors, indicating active status, idleness, or specific sub-tasks.
- **Memory Dependencies:** Visual connectors or highlights show shared memory access, illustrating data movement, contention points, and cache interactions.
- **Intermediate Results:** Partial computations and data transformations are dynamically displayed, allowing students to trace data evolution.

B. Matrix Multiplication (Data Parallelism)

Matrix multiplication ($C = A \times B$) is a quintessential example of data parallelism due to its independent, element-wise computations, making it ideal for GPU acceleration. Each element $C[i][j]$ is calculated by the summation:

$$C[i][j] = \sum_{k=0}^{n-1} A[i][k] \times B[k][j]$$

where A is $m \times n$, B is $n \times p$, and C is $m \times p$.

In our tool, matrix multiplication is visualized as a grid where each output cell in C corresponds to an independent task. The visualization demonstrates that each $C[i][j]$ element can be calculated in parallel, requiring only specific rows from A and columns from B. As shown in Figure 1, each output cell

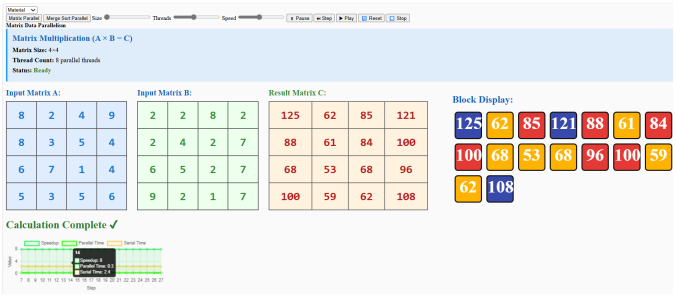

Fig. 1: Matrix Multiplication Visualization

is dynamically highlighted as a simulated "thread" processes it, illustrating:

- **Independent Operations:** Multiple $C[i][j]$ elements computed concurrently.
- **Minimal Synchronization:** Low overhead, primarily for initial data distribution and final result aggregation.
- **Workload Distribution:** Efficient distribution of computational workload, with threads handling subsets of output cells.

Users can configure matrix size and thread count to observe the impact on parallel execution, fostering an understanding of how data parallelism scales.

C. Parallel Merge Sort (Task Parallelism)

Parallel merge sort exemplifies task parallelism, where a problem is recursively decomposed into independent sub-problems (tasks) for concurrent execution. Its divide-and-conquer nature is well-suited for a hierarchical, tree-based visualization, clarifying task dependencies and recursive calls.

The tool displays parallel merge sort as a recursive binary tree. Each node represents a sub-problem—a segment of the array to be sorted. Processing each subtree is a distinct task assigned to an available thread. Figure 2 illustrates its dynamic

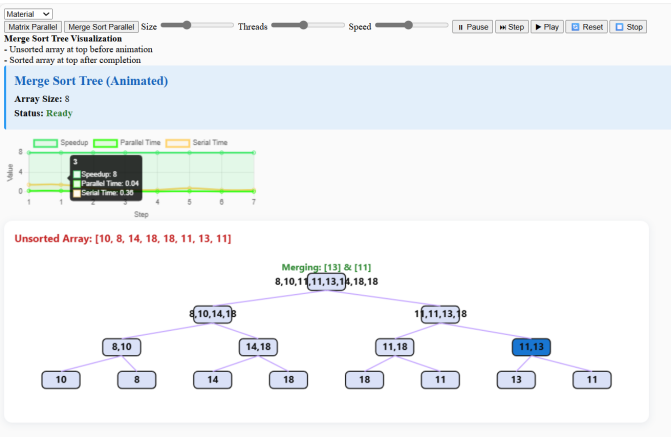

Fig. 2: Merge Sort Visualization

behavior, depicting:

- **Task Decomposition:** How a large sorting problem is broken down into smaller sub-problems.
- **Thread/Task Assignment:** Dynamic allocation of threads to process sub-problems.
- **Synchronization at Merge Points:** Visualization of synchronization barriers at each merge step, where threads wait for dependent sub-tasks to complete, demonstrating coordination needs.
- **Workload Distribution and Load Balancing:** Observation of task distribution and potential for imbalance with varying array sizes or limited threads.

Users can set array sizes and observe task decomposition, thread coordination, and dynamic task allocation, gaining intuition for task parallelism's benefits and challenges, including overhead and synchronization.

D. Metrics and Comparative Interface

The tool includes a dedicated side panel for quantitative performance metrics, offering an analytical perspective. It compares simulated serial and parallel execution times, along with theoretical speedup (Amdahl's Law). Figure 3 shows a visual comparison between theoretical maximum speedup and empirically observed speedup, which accounts for communication overhead, synchronization delays, and load imbalances. Key features of the metrics panel:

- **Runtime Comparison:** Numerical display of serial and parallel execution times is provided directly within the metrics panel. This enables students to observe and quantify the efficiency gains or bottlenecks of their parallel algorithm configurations immediately after each run.
- **Speedup Calculation:** The tool supports real-time calculation and visualization of speedup, comparing parallel execution to a simulated serial baseline, helping learners visualize effects of thread count, problem size, and scalability.
- **Resource Utilization Insights:** Highlights thread idle times during pauses and execution, making synchronization overhead and load imbalance visually apparent. By distinguishing idle from active threads, students can more easily diagnose performance losses.
- **Toggle between Visualizations:** Enables seamless switching between step-by-step algorithm animations and quantitative metric graphs. This dual-view approach helps users directly associate computational events with shifts in performance metrics.

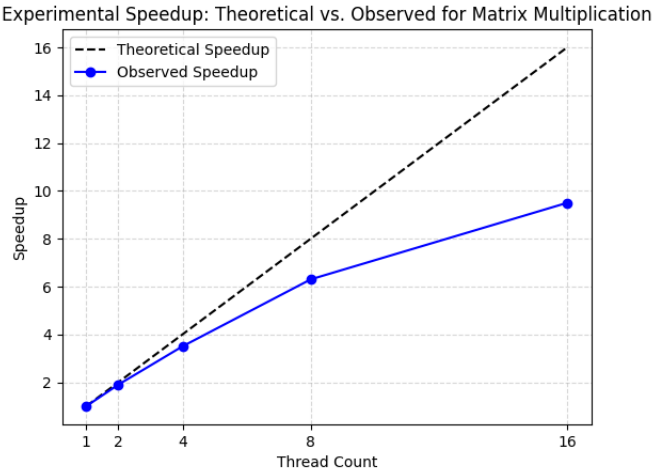

Fig. 3: Experimental Speedup: Theoretical (dashed) vs. Observed (solid) for matrix multiplication.

This integrated approach allows students to connect visual events (e.g., threads waiting) to their impact on performance metrics (e.g., reduced speedup), fostering a deeper understanding of parallel algorithm design and optimization.

V. Preliminary Classroom Usage

The tool was piloted in a senior-level Parallel and Distributed Computing lab with 17 students, who accessed it via the public GitHub repository. Informal observations and a short feedback form indicated that most students found the visualizations helpful for understanding thread assignment, synchronization behavior, and scaling effects. Students particularly appreciated the ability to vary parameters (e.g., thread count, input size) and observe performance differences in real time. Several commented that the merge-sort visualization made task-level coordination "easy to see" compared to static diagrams. While this preliminary usage was limited and not a controlled study, the positive reactions suggest the tool's practical value in teaching core parallel-algorithm concepts.

A more detailed evaluation with larger cohorts is planned for future work.

Table I concisely compares how data and task parallelism are characterized and visualized, serving as a quick reference.

TABLE I: Comparison: Data vs. task parallelism visualized.

Aspect	Matrix Multiplication	Merge Sort
Computation Structure	Independent, element-wise	Recursive, hierarchical
Thread Utilization	Concurrent across cells	Dynamic, per task subdivision
Synchronization	Minimal (final aggregation)	Frequent, at merge points
Performance Metrics	Runtime and theoretical speedup	Execution flow and task balance
Best Use Case	Dense numerical data	Divide-and-conquer problems
Key Concept	Data parallelism visualization	Task coordination and recursion

VI. Discussion and Educational Impact

The interactive GPU-accelerated visualization tool promises a profound educational impact by fostering visual intuition for parallel computing. Its contribution to pedagogical effectiveness includes:

- **Enhanced Conceptual Clarity:** Clearly distinguishes between independent data parallelism and coordinated task parallelism, making these fundamental paradigms concrete through visual observation of thread behavior.
- **Active Parameter Exploration:** Dynamic parameter adjustment enables active experimentation, allowing students to observe the impact of thread counts and input sizes on algorithmic behavior and performance, grounding theoretical concepts like Amdahl's Law in observable reality.
- **Unparalleled Accessibility and Usability:** Browser-based WebGL design ensures universal accessibility, eliminating complex installations and fostering widespread adoption across diverse environments.
- **Versatile Pedagogical Integration:** Flexible design allows seamless integration into lectures, laboratory assignments, self-study, and online learning modules, enriching the PDC learning ecosystem.

This compelling combination of interactivity, visual clarity, and quantitative insight positions the tool as a powerful educational asset, empowering educators and providing learners with profound visual and intuitive experiences of complex computational concepts.

VII. Future Work

Building upon the current implementation, several avenues for future enhancements are planned:

- **Expanded Algorithm Coverage:** Incorporate visualizations for MapReduce, prefix sum (scan), and intricate GPU kernel operations to cover a broader spectrum of parallel computing techniques.
- **Advanced Synchronization Visualization:** Feature dynamic visualizations of locks, barriers, semaphores, and atomic operations to illustrate their role in ensuring correctness and preventing race conditions.
- **GPU Profiling Backends Integration:** Add optional GPU profiling for live execution time and bandwidth demos on user hardware.
- **Guided Tutorials and Assessments:** Develop interactive tutorials, exercises, and quizzes for structured, objective learning.

These enhancements will significantly expand the tool's functionality, deepen its educational value, and solidify its position as an indispensable resource for parallel computing education.

VIII. Conclusion

We have presented the design and implementation of an interactive GPU-accelerated web-based visualization tool specifically engineered to enhance the teaching and learning of parallel algorithms. By illustrating both data parallelism (through matrix multiplication) and task parallelism (through parallel merge sort) in real time, the platform provides an accessible, practical, and highly engaging resource for educators and students alike. Its highly configurable interface, coupled with real-time performance metrics and a strong visual emphasis on thread behavior and synchronization support, effectively supports the development of an intuitive and profound understanding of complex parallel computing concepts. This work successfully establishes a robust and extensible foundation for future pedagogical tools and educational extensions in the rapidly evolving and critically important domain of parallel computing.

References

[1] X.-p. Liu, E.-z. Wang, L.-p. Zheng and X.-w. Wei, "Study on template for parallel computing in visual parallel programming platform," in *Proc. 1st Int. Symp. Pervasive Comput. Appl. (ISPCA)*, Urumqi, China, Aug. 2006, pp. 476–481, doi: 10.1109/SPCA.2006.297439.

[2] A. Waheed and D. T. Rover, "Performance visualization of parallel programs," in *Proc. IEEE Vis.*, San Jose, CA, USA, Oct. 1993, pp. 174–181, doi: 10.1109/VISUAL.1993.398866.

[3] V. Subhash, K. Pandey and V. Natarajan, "GPU parallel computation of Morse–Smale complexes," in *Proc. IEEE Vis. (VIS)*, Salt Lake City, UT, USA, Oct. 2020, pp. 36–40, doi: 10.1109/VIS47514.2020.00014.

[4] G. Sethumadhavan, S. A. Narayanasamy and A. Gopalakrishnan, "Performance evaluation of image smoothing on CPU and GPU using multithreading: An experimental approach in high performance computing," in *Proc. 2016 IEEE Int. Conf. Comput. Intell. Comput. Res. (ICCIC)*, Chennai, India, Dec. 2016, pp. 1–5, doi: 10.1109/ICCIC.2016.7919680.

[5] Y. Zamora and R. Robey, *Parallel and High Performance Computing*. Shelter Island, NY, USA: Manning, 2021.

[6] L. S. Nair, "An analytical study of performance towards task-level parallelism on many-core systems using Java API," in *Proc. 2021 6th Int. Conf. Commun. Electron. Syst. (ICCES)*, Coimbatore, India, Jun. 2021, pp. 1255–1259, doi: 10.1109/ICCES51350.2021.9489215.

[7] Z. Cheng, K. Qi, L. Jun and H. Yi-Ran, "Thread-level parallel algorithm for sorting integer sequence on multi-core computers," in *Proc. 4th Int. Symp. Parallel Archit., Algorithms Program. (PAAP)*, Tianjin, China, Dec. 2011, pp. 37–41, doi: 10.1109/PAAP.2011.57.

[8] Y. Liu, N. Xie, L. Qin and X. Zhang, "Matrix optimization algorithm for WebGL-based 3D visualization construction models," in *Proc. 2024 10th Int. Conf. Comput. Commun. (ICCC)*, Chengdu, China, Dec. 2024, pp. 834–839, doi: 10.1109/ICCC62609.2024.10942003.