Performance Comparison and Classification of Concurrent Algorithms on CPU and GPU

1st Geetam Talluri
Btech. Electrical Engineering
Indian Institute of Technology Dharwad

2nd Jatin Lather
Btech. Engineering Physics
Indian Institute of Technology Dharwad

3rd Uppu Eshwar
Btech. Chemical Engineering
Indian Institute of Technology Hyderabad

***Abstract*—As technology advances, the need for rapid and precise data processing grows. Concurrent programming plays a vital role in achieving this goal, but choosing the right hardware platform is crucial for performance requirements. GPUs excel in intensive parallel computation, while CPUs are better for sequential and memory-intensive tasks. In this research, we meticulously evaluate the compatibility of specific algorithms with CPU and GPU architectures, taking into account factors such as thread divergence, dataset size, and computational demands. Using the NVIDIA Jetson Nano, we assess ten diverse algorithms. Our findings show GPU efficiency depends on workload, algorithm complexity, and parallelization potential. Consequently, the decision between GPU and CPU deployment is heavily influenced by the intrinsic characteristics of the algorithm in question.**

***Index Terms*—Performance Assessment, Parallel Computing, GPU computing, CPU vs GPU,**

I. Introduction

In the expanding field of high performance computing, the choice between central processing units (CPUs) and graphics processing units (GPUs) for algorithms is significant, as each has distinct advantages. CPUs generally offer high clock speeds, better cache hierarchy, and a broad range of instruction sets, while GPUs, designed for parallelism, excel at concurrent processing. This study delves into the fundamental question of which hardware is best suited to specific tasks, with a focus on concurrent algorithms, a domain closely associated with parallelism and essential to modern computing. While existing studies have made significant contributions, there is a noticeable gap in performance evaluations. Many studies focus on CPUs or GPUs, or limit their scope to specific algorithm types on these devices. This restricted perspective often overlooks the intricate interplay between algorithms and hardware. Our inquiry extends beyond algorithm selection, delving into core characteristics—memory operations, computational operations, and thread divergence. These metrics reveal distinct performance profiles shaped by concurrent algorithms and the hardware they run on. Importantly, these characteristics impact CPUs and GPUs differently. Thread divergence, in particular, poses unique challenges for GPUs compared to CPUs. In summary, our analysis goes beyond CPU versus GPU, offering valuable insights across the computing spectrum. Our research empowers a diverse array of stakeholders to optimize computational performance by understanding the interplay between algorithms and hardware, ultimately benefiting end-users reliant on efficient computing solutions.

II. Methodology

In our study, we selected some algorithms with diverse characteristics. For memory operations, we looked at how these algorithms access memory, calculated the total number of memory accesses they make when processing a dataset with 'n' elements. Similarly for computational operations We then categorized these algorithms into three groups: low, moderate, and high memory usage based on certain ranges (n to n^3 as low, n^3 to n^5 as moderate, and greater than n^5 as high). In the thread divergence analysis, we pinpointed decision points in our algorithms where threads might make varying choices. We assigned probabilities, calculated cumulative probabilities, and quantified thread divergence. Low values showed convergence (follow a similar path of execution within a program), while high values indicated divergence (threads frequently diverged). We categorized divergence as low, moderate, or high, enhancing our understanding of how threads behaved during GPU execution—whether they tended to follow a common path or often diverged. Lastly, we measured the speedup, which tells us how much faster an algorithm runs on a CPU compared to a GPU. We did this by comparing the execution time on the CPU (t_{cpu}) to the execution time on the GPU (t_{gpu}). Speedup is calculated as ($Speedup = t_{cpu}/t_{gpu}$).

III. Evaluation

A. Experimental Setup

This evaluation is based on the NVIDIA Jetson Nano development kit which uses a Quad-core ARM Cortex-A57 MP Core processor and NVIDIA Maxwell architecture GPU with 128 NVIDIA CUDA cores both share a memory of 4 GB 64-bit LPDDR4, 1600MHz (25.6 GB/s). Each algorithm has been developed in C and CUDA to compare the amount of time spent respectively on the CPU and GPU, We used the GNU **time** command, because we wanted to consider the overheads of memory transfer in GPU. We used CUDA 10.2 to compile algorithms and all algorithms were tested repetitively many times. For CPU execution, parallel implementation is done using pthreads and OpenMP.

B. Data set Description

The Dataset consists of random floating-point numbers uniformly distributed within the [0, 1] interval, creating a diverse range of data inputs that challenge algorithms under

Algorithms and tasks	Dataset Range	Computational operations	Memory operations	Thread Divergence
Bitonic sort	128 to 131072	Moderate	Moderate	Low
Finite Difference Method(FDM)	64 to 4096	High	High	Moderate
C-means clustering	100 to 100000	High	Moderate	Moderate
Embedded Zerotree wavelet(EZW)	128 to 4096	High	Moderate	Low
Run Length Encoding (RLE)	100 to 100000	Moderate	Moderate	Low
RSA encryption	100 to 100000	Moderate	Moderate	Low
Euler angles	100 to 100000	High	Low	Low
QR decomposition (QR)	10 to 10000	Moderate	Moderate	Moderate
Vector addition	100 to 100000	Low	Low	Low

TABLE I
CHARACTERISTICS OF EACH ALGORITHM AND TASK.

examination. Notably, we have taken the additional step of normalizing the dataset into six equal parts to ensure uniformity in our evaluations. The even distribution of data ensures that our assessments are fair and free from any unfair influences or biases. Importantly, our dataset closely resembles real-life situations, highlighting that our research findings can be useful in many computational environments

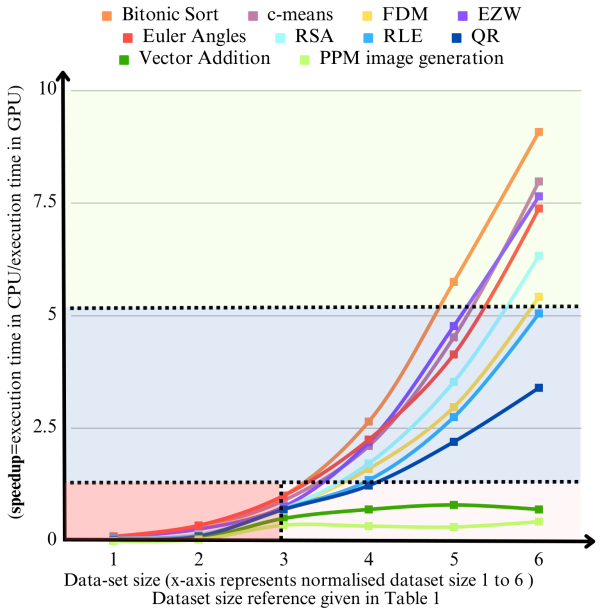

Fig. 1. Graph depicting each algorithm's speedup curve with respect to datset size.

C. Results

In this subsection, we present the results of our extensive performance evaluation, explaining the intricate relationship between concurrent algorithms and their hardware platforms, displaying the algorithmic behavior under varying conditions.

*1) **Speedup Analysis**:* Figure 1 illustrates the speedup achieved by various algorithms as a function of dataset size. There is a noticeable transition, highlighted in dark red, where the CPU performs significantly better for smaller datasets due to the inherent overhead of data transfer to and from the GPU. Conversely, the green region in the graph represents algorithms with optimal speedup characteristics, driven by factors such as low thread divergence, increased computational operations, and reduced memory-intensive tasks.

*2) **Algorithm-specific Insights**:* The Bitonic Sort algorithm performed consistently well on both CPU and GPU due to its low thread divergence, making it suitable for GPU parallel processing. The Finite Difference Method (FDM) displayed significant computational and memory operations, with the GPU's parallelism yielding enhanced speedups for larger dataset. Algorithms like C-means Clustering, Embedded Zerotree Wavelet (EZW), Run Length Encoding (RLE), and RSA Encryption performed efficiently on the CPU but benefited from GPU speedups with larger datasets. Euler Angles and QR Decomposition posed unique challenges, with the former leaning towards GPU optimization and the latter necessitating synchronization mechanisms on the GPU. For vector addition with minimal computational tasks, execution times were similar or sometimes faster on the CPU. This was because GPUs incurred higher memory access costs. Hence, some computational tasks are necessary to effectively utilize GPUs even in parallel processing.

IV. CONCLUSIONS

The decision to employ GPU or CPU resources should be based on carefully analysing algorithm characteristics, as discussed. As observed from the evaluation, If the algorithm primarily consists of low computational operations, the memory access patterns are not complex and have moderate or high thread divergence, where threads have significantly different execution paths, CPUs may be more suitable. On the other hand, When the algorithm requires intensive Computational Operations and exhibit low thread divergence, GPUs can work efficiently.

REFERENCES

[1] Syberfeldt, A. and Ekblom, T. (2017) "A comparative evaluation of the GPU vs the CPU for parallelization of evolutionary algorithms through multiple independent runs", International Journal of Computer Science and Information Technology, 9(3), pp. 01–14.

[2] H. MohammedFadhil and M. Issam Younis, "Parallelizing RSA algorithm on Multicore CPU and GPU," International Journal of Computer Applications, vol. 87, no. 6, pp. 15–22, 2014.

[3] J. Sanders and E. Kandrot, "CUDA by example: An Introduction to General-Purpose GPU programming". 2010.

[4] Pikle, N.K., Sathe, S.R. and Vyavhare, A.Y. GPGPU-based parallel computing applied in the FEM using the conjugate gradient algorithm: a review. Sādhanā 43, 111 (2018).