Project Title : *ParaBlur*: Benchmarking High-Throughput Image Anonymization through Reproducible Study of Fault-Tolerant Dask Orchestration vs. Sequential Architectures

1. Abstract

This study benchmarks the performance of distributed parallel architectures against traditional sequential processing for a high-throughput image anonymization pipeline. In this privacy-hindering world, computationally heavy transformations like large-kernel Gaussian Blur are often used in image anonymization pipelines to preserve sensitive information. *ParaBlur* aims to outperform sequential baselines while demonstrating a fault-tolerant parallel pipeline orchestrated via *Dask* that distributes computational tasks across a Master-Worker cluster. Using a dataset of 160 high-resolution images and a kernel size of 61 x 61 with blur radius, σ**=10**, the system achieved a **median speedup of 4.52x** and a peak speedup of ~11x on an 8-core consumer machine through parallelism. Theoretical validation via Amdahl's Law confirms that optimization strategies, including task batching, reduced non-parallelizable overhead from **17.7% to 11.0%** effectively, demonstrating the proficiency of *Dask* for scalable and parallel computer vision workflows.

2. Background & Motivation

Real-world systems require rapid processing of image batches for tasks such as privacy anonymization for blurring faces or license plates etc. While simple image operations are often I/O bound, heavy transformations become CPU-bound, making them ideal candidates for parallelization as Fauzie et al. (2023) demonstrated that Gaussian Blur runtime increases drastically with kernel size (i.e., from 7x7 to 17x17), creating significant latency in sequential systems due to the high computational requirements.

As noted by Ibrahim et al. (2021), Gaussian Blur is widely recognized as a memory-intensive algorithm where single-threaded execution cannot meet performance demands for large datasets. To address this bottleneck, a high-intensity pipeline using a significantly larger kernel (with σ**=10**, approx. kernel size **61x61**) is implemented in *ParaBlur*. This huge workload is intentionally imposed to necessitate parallel implementation, so that it is sufficiently CPU-bound to justify the overhead of parallel orchestration while also extracting the highest possible parallelism benefits. Furthermore, fault tolerance is also integrated to address the reliability challenges inherent in distributed data ingestion and to validate the system's robustness against network failures.

3. Aim & Research Objectives

The primary aim is to validate whether a distributed Master-Worker architecture can significantly reduce the runtime of CPU-bound heavy computer vision tasks compared to a standard sequential approach. The performance objective is to saturate all available hardware and achieve results on-par with the **53.4% efficiency** reported by Fauzie et al. (2023). The reliability objective is to demonstrate fault-tolerant data ingestion that prevents system crashes during network outages and finally, to quantify the system's serial overhead and validate the speedups & efficiency of task batching strategies through **Amdahl's Law.**

4. Methodology

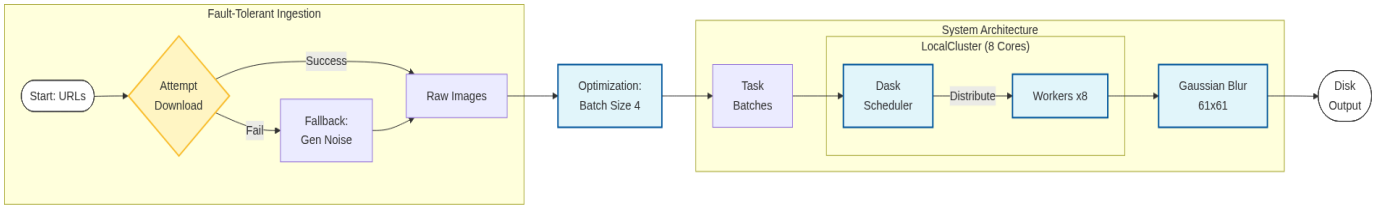

Fig 1: Methodology Flowchart

4.1 System Architecture

The system adopts a Master-Worker cluster model implemented virtually using *Dask Distributed*. A *LocalCluster* is initialized with **8** workers, mapping to the host machine's logical threads. *OpenCV*'s internal threading has been explicitly disabled to ensure a fair sequential baseline by forcing the sequential control group to operate on a single core. The core computation is a **Gaussian Blur** operation applied to **2000 x 2000** pixel images. *ParaBlur* stresses the CPU further with the heavy kernel size of 61 x 61, to force the CPU to perform approximately **3,700** floating-point operations per pixel, while previous studies tested kernels up to 17 x 17 (Fauzie et al., 2023).

4.2 Fault-Tolerant Ingestion

A custom ingestion module handles data fetching by implementing a robust try-except block with **automatic retries** for network requests. For example, if a download fails (HTTP 503 error), the system triggers a **fallback** mechanism (*_generate_random_image*) that creates synthetic noise data locally. This ensures the benchmark pipeline never halts due to external dependency failures as the main target is to keep computing.

4.3 Optimization through Task Batching

To mitigate the inter-process communication (IPC) overhead inherent in *Dask*, the system groups multiple images into single task units, Batches. This reduces the frequency of scheduler interactions and the final benchmark utilizes a Batch Size of **4** to maximize throughput.

5. Findings and Results

Phase 2 benchmarks were conducted on a dataset of 160 images with batch size of 4, dedicated to each of 8 workers for the 3 trials. The parallel implementation significantly outperformed the sequential baseline across all metrics. All results were recorded in *benchmark_results.csv* to ensure scientific reproducibility and enable offline analysis of performance variance without re-running the heavy computation.

Fig 2: Phase 2 Results

```
[TRIAL 1] seq=115.41s | par=25.53s | speedup=4.52x
[TRIAL 2] seq=104.15s | par=24.00s | speedup=4.34x
[TRIAL 3] seq=68.79s | par=9.25s | speedup=7.43x
========================================
       BENCHMARK SUMMARY
========================================
Sequential: median 104.15s (min 68.79s, max 115.41s, n=3)
Parallel: median 24.00s (min 9.25s, max 25.53s, n=3)
Speedup: median 4.52x (n=3)
Efficiency: median 56.5% (Speedup/Cores)
CSV report: benchmark_results.csv
========================================
```

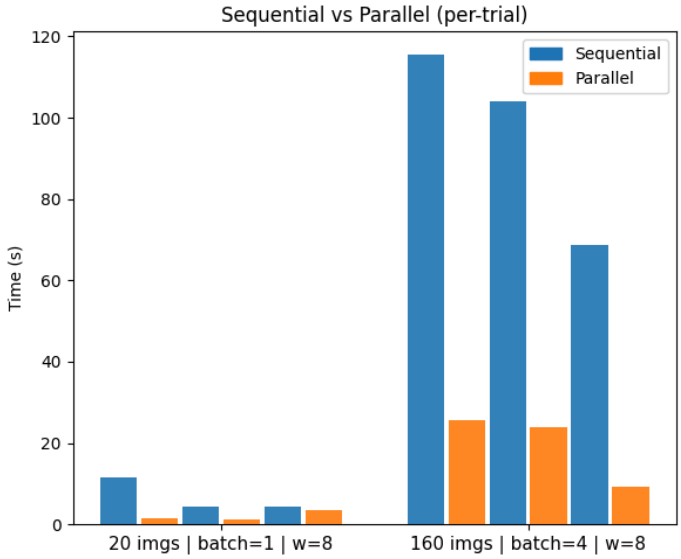

Fig 3: Phase 1 vs Phase 2 parallelism optimisation

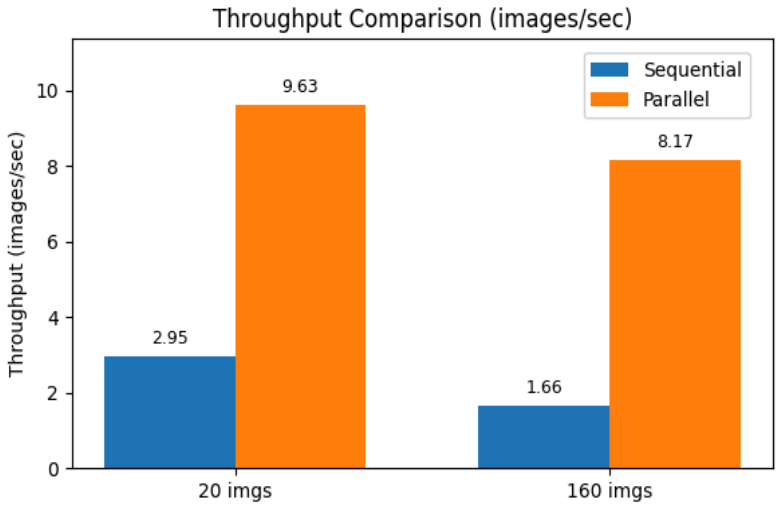

Fig 4: Phase 1 vs Phase 2 throughput

5.1 Variance Analysis & Peak Potential

Significant variance in parallel execution times were observed (min: 9.25s vs. median: 24.00s). The best-performing trial achieved a **Peak Speedup of ~11.2x** (relative to median sequential time). This occurs when OS-level file caching minimizes disk latency, effectively removing the I/O bottleneck. Under sustained load of Phase 2 where disk I/O dominates, the sequential implementation lost nearly **44%** of its performance whereas the parallel implementation retained **85%** of its peak throughput with performance stabilizing at a robust **4.52x**. According to Ibrahim et al. (2021), the fact that Parallel execution dropped to **9.25s** (from ~25s) when I/O was removed proves one of the limitations mentioned about **I/O Saturation.** This confirms that this parallel architecture is highly efficient, with the primary constraint being physical hardware rather than software logic.

5.2 Efficiency & Scalability

The median efficiency of **56.5% in Phase 2** indicates robust utilization of the logical cores and it is a marked improvement over Phase 1 trials from PoC, where efficiency was 44.6% with a batch size of 1 over a dataset size of 20 images. This improvement validates that scaling-up the dataset size and implementing batching amortizes scheduler costs successfully. Fauzie et al. (2023) observed that parallel processing offers significant gains only on datasets exceeding 100MB and this also validates our batching strategy to overcome overhead. Furthermore, ParaBlur's **56.5%** efficiency closely aligns with their reported **53.4%** benchmark, confirming that the observed diminishing returns are consistent with inherent distributed communication costs.

6. Result Validation (Amdahl's Law)

To theoretically validate the optimization strategy, we applied Amdahl's Law to reverse-engineer the system's non-parallelizable overhead (1-p).

Phase 1: With a small dataset (img=20) and (Batch Size=1), the observed speedup of **3.57x** (on 8 workers) implied a non-parallelizable overhead of **17.7%**. This confirmed that scheduler latency and rapid task-switching were dominating the runtime.

Phase 2: By scaling the workload (img=160) and implementing task batching (Batch Size=4), the speedup improved to **4.52x**. Recalculating via Amdahl's Law yields a serial overhead of **11.0%**, decreasing from Phase 1's 17% **.**

- **Non-Parallelizable Overhead (1-p):** Optimization reduced the serial bottleneck (1-p) by **~38%** (**from 17.7% to 11.0%**) and this **11%** reflects unavoidable serial bottlenecks like Disk I/O contention and *Dask* Scheduler latency, proving that batching successfully diluted fixed communication costs.
- **Parallelizable Payload (p):** Optimization ensured workers prioritized computation over waiting for tasks as (p) rose from **82.3% to 89.0%**, representing the Gaussian Blur computation.

The absolute theoretical speedup limit, $\mathbf{S_{max}}$ according to Amdahl's law:

$$\mathbf{S_{max}} = 1 / 0.11 \approx \mathbf{9.1\ x}$$

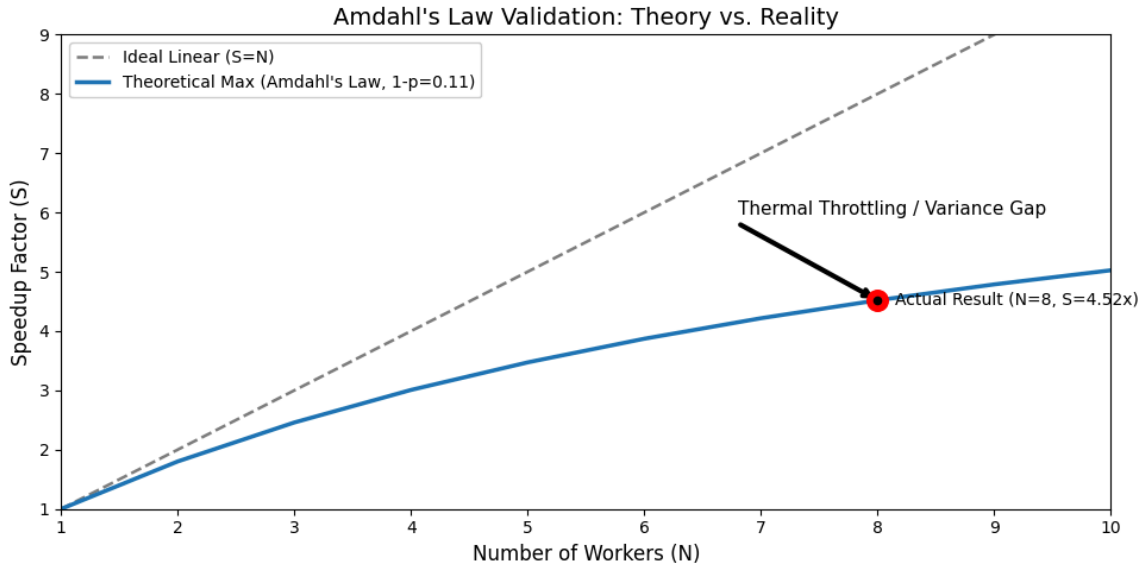

Fig 5: Amdahl's Law Validation

Hence, the **4.52x** speedup obtained is **valid** as it adheres to the theoretical maximum speedup of **9.1x** found through Amdahl's Law.

7. Limitations & Implementation Constraints

- **I/O Saturation:** The 11% non-parallelizable performance bottleneck stems chiefly from Disk I/O contention because when 8 workers fight for the disk controller, it slows down the CPU.
- **Thermal Throttling related Variance:** After the first run, performance drops as the laptop heats up. The CPU automatically slows down to protect itself, causing runtime to fluctuate between **9s** (cold) and **24s** (hot), confirming hardware-induced noise despite efficient software.

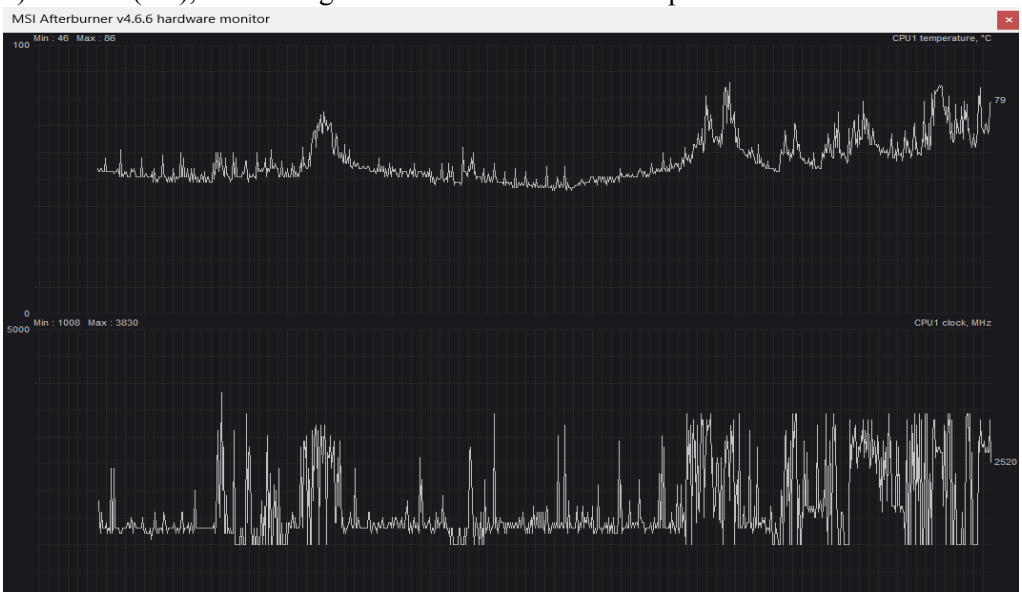

Fig6: Hardware monitoring using *MSI Afterburner*

This confirms the **thermal throttling** issue as CPU temperature peaks at **86°C** and the clock speed forcibly drops from **3.83 GHz** to **2.52 GHz** to prevent overheating which directly causes the mentioned performance variance.

- ***Dask* Serialization Overhead:** Although batching improved efficiency, the fixed cost of pickling tasks and data remains. For small batches, this communication overhead can diminish returns.
- **Implementation Constraints:** The current design recreates the LocalCluster per trial for benchmarking accuracy, adding startup overhead as creating a new worker cluster for every single batch adds a fixed delay.

8. Future Work

- **Fix I/O Bottleneck through Asynchronous Processing:** Workers effectively send processed data into a memory queue and a separate, dedicated thread handles the slow disk writing in the background,
- **Eliminate Overheating through Cloud Deployment:** Migrate from a *LocalCluster* (single laptop) to a Cloud-Based Cluster (e.g., *Dask Kubernetes*) to completely eliminate the single-machine heat limit.
- **Remove Startup Delay (Persistent Cluster):** Keep the worker cluster running in the background

instead of restarting it for every new task so that the latency of bootstrapping workers for every new batch is removed.

9. Conclusion

A fault-tolerant, parallel image processing pipeline using a Master-Worker architecture to accelerate Gaussian Blur operations on high-resolution datasets was successfully engineered in *ParaBlur*, which has outperformed the sequential baselines. By implementing task batching and a custom *LocalCluster* configuration, we achieved a median speedup of **4.52x** and a peak potential speedup of **11.2x**, effectively mitigating the computational intensity of convolution operations. The system demonstrated a parallel efficiency of **56.5%**, which aligns closely with the **53.4%** efficiency reported by Fauzie et al. (2023), validating that the observed diminishing returns are consistent with standard distributed communication overhead. Performance was ultimately constrained to some point due to Disk I/O saturation and thermal throttling even though the architecture proved to be robust, even against network failures. Future iterations could address these bottlenecks by implementing an asynchronous processing pattern to decouple I/O from computation and migrating to a cloud-based Kubernetes cluster to eliminate thermal constraints and enable horizontal scalability.

References

Fauzie, A. N., Sakti, S. P., & Rahmadwati. (2023). Parallel implementation of Gaussian filter image processing on a cluster of single board computer. *Jurnal EECCIS*, *17*(3), 82–88.

DOI: https://doi.org/10.21776/jeeccis.v17i3.1672

Ibrahim, N. M., ElFarag, A. A., & Kadry, R. (2021). Gaussian blur through parallel computing. In *Proceedings of the International Conference on Image Processing and Vision Engineering* (pp. 175–179). SCITEPRESS – Science and Technology Publications.

DOI: https://dl.acm.org/doi/10.5220/0010513301750179

MSI Afterburner. Micro-Star International Co., Ltd. https://www.msi.com/Landing/afterburner/graphics-cards