Performance Analysis of Parallel Implementations

Ya-Wei Tsai

October, 2024

1 Project Description Recap

The project aims to develop and analyze three implementations of an image-processing task:

- **Implementation 1: Sequential Processing** – A baseline sequential version where images are processed one at a time without any parallelization.
- **Implementation 2: Parallel Processing of Multiple Images** – Each thread processes a separate image, minimizing inter-thread communication by assigning each thread an independent task. This implementation uses a custom TAS (Test-and-Set) lock to manage access to a queue implemented as a linked list.
- **Implementation 3: Parallel Slicing of Individual Images** – Each image is divided into slices, and threads work on each slice in parallel. This implementation also uses a queue implemented as a linked list but does not require a TAS lock for synchronization.

The goal of these implementations is to measure the speedup achieved with increasing thread count and compare measured performance with theoretical expectations based on Amdahl's law.

2 Experiment Instructions

To run the testing script, execute:

```
sbatch benchmark-proj1.sh
```

3 Theoretical Speedup Calculation

The theoretical speedup is estimated by Amdahl's law. We assume the parallel fraction P of the workload is 95%.

Using this estimate of P, we can calculate the theoretical speedup for N threads as:

$$\text{Speedup} = \frac{1}{(1-P) + \frac{P}{N}}$$

This formula gives us an approximation of the maximum possible speedup, based on the assumption that P accurately reflects the parallelizable fraction of the workload.

The speedup we might get in the limit, with an infinite number of threads, is calculated by the following formula.

$$\lim_{x \to \infty} \frac{1}{(1-P) + \frac{P}{N}} = \frac{1}{(1-P)} = 20$$

4 Original Data

The original data for the sequential and parallel execution times, along with the measured and theoretical speedup values, are presented in Table **??**. This data serves as the basis for the speedup calculations and performance analysis.

5 Graph Analysis (Interpretation of Results)

5.1 Graph 1: 'parfiles' (Multiple Images in Parallel)

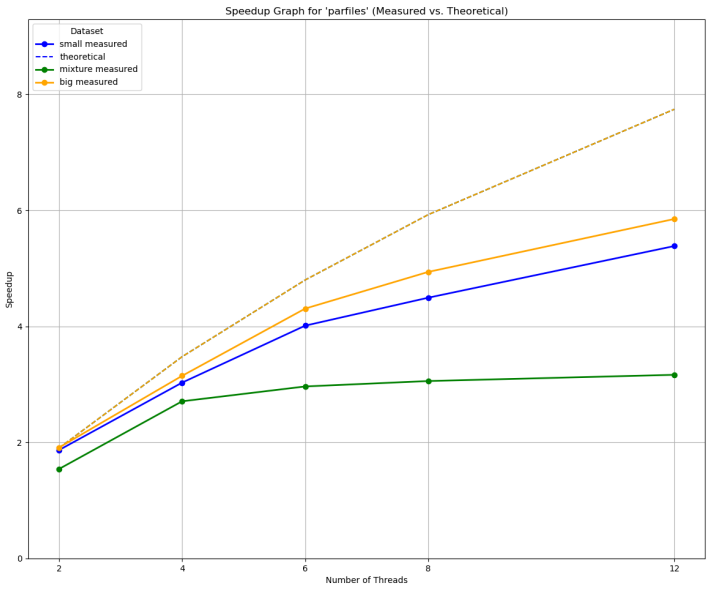

Figure 1: Speedup Graph for 'parfiles' (Multiple Images in Parallel)

- **Small Dataset:** The measured speedup increases as thread count increases, indicating that this dataset scales well with thread count but also has a gap that widens with more threads.
- **Mixture Dataset:** The measured speedup falls below the theoretical speedup, possibly due to inherent processing limitations or synchronization overhead.
- **Big Dataset:** Initial speedup aligns closely with theoretical predictions, but the gap widens with more threads, suggesting that resource contention limits scalability.

5.2 Graph 2: 'parslices' (Parallel Slicing of Each Image)

- **Small Dataset:** Speedup gain is minimal. Measured values are close to theoretical values, achieving a maximum speedup of approximately 1.75, indicating that this approach may not fully utilize additional threads.
- **Mixture Dataset:** Similar to the small dataset, the mixture dataset shows limited speedup, with minimal deviation from theoretical values.
- **Big Dataset:** Speedup maximum approaches 1.8, indicating that this approach may not effectively leverage more threads due to communication overhead.

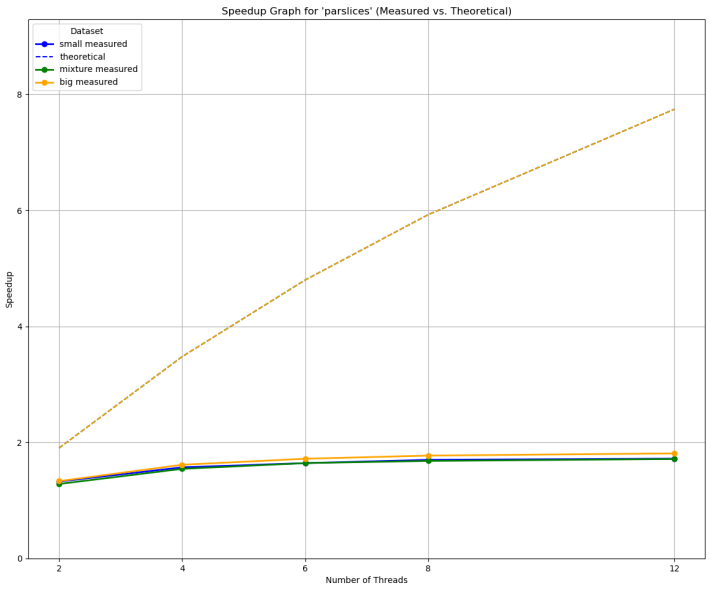

Figure 2: Speedup Graph for 'parslices' (Parallel Slicing of Each Image)

6 Performance Analysis Questions

6.1 Hotspots and Bottlenecks in Sequential Program

Hotspots are likely due to intensive image-processing tasks, such as applying effects that involve reading and writing large image data sections. Bottlenecks may arise in memory access patterns or repetitive computations, particularly for larger datasets.

6.2 Comparison of Parallel Implementations

The 'parfiles' implementation (Implementation 2) outperforms the 'parslices' approach (Implementation 3), particularly for small and big datasets. The 'parfiles' implementation scales better because each thread processes a separate image, minimizing inter-thread communication. Conversely, 'parslices' encounters significant overhead due to synchronization requirements when processing slices within a single image.

6.3 Impact of Problem Size on Performance

Problem size notably affects performance in the 'parfiles' implementation, where larger datasets achieve closer-to-theoretical speedups. In contrast, the 'parslices' approach shows limited performance gains regardless of data size, suggesting

that slice-based processing overhead diminishes scalability.

6.4 Closeness to Amdahl's Law

The 'parfiles' implementation approaches the speedup predictions from Amdahl's law more closely than the 'parslices' implementation. Deviations, particularly in 'parslices,' are likely due to synchronization costs and non-parallelizable overhead, which Amdahl's law does not fully account for in practical implementations.

6.5 Potential Performance Improvement Areas

The following optimizations could enhance performance:

- **Reducing Inter-thread Communication:** Using larger, overlapping slices to reduce the need for synchronization in 'parslices'.
- **Optimizing Memory Access:** Minimizing cache contention by refining memory access patterns for each Go routine.
- **Dynamic Load Balancing:** Assigning slices based on workload rather than static slicing could improve performance.
- **Optimized TAS Lock:** Improving the TAS lock mechanism in 'parfiles' could reduce contention on the queue, especially with high thread counts.

7 Conclusion

The analysis demonstrates that the 'parfiles' implementation (Implementation 2) generally achieves better performance across varying dataset sizes, aligning more closely with theoretical speedups. The 'parslices' approach (Implementation 3) faces practical limitations due to synchronization overhead and limited thread utilization. Future improvements could focus on reducing communication costs in 'parslices' and optimizing thread management in both implementations.

Table 1: Execution Times for Sequential and Parallel Implementations

Implementation	Dataset	Threads	Time (s)
Small Dataset			
Sequential	small	1	13.533167
Parfiles	small	2	7.238964
Parfiles	small	4	4.464302
Parfiles	small	6	3.372311
Parfiles	small	8	3.010968
Parfiles	small	12	2.513207
Parslices	small	2	10.201719
Parslices	small	4	8.609905
Parslices	small	6	8.232356
Parslices	small	8	7.958243
Parslices	small	12	7.863815
Mixture Dataset			
Sequential	mixture	1	79.917817
Parfiles	mixture	2	51.777809
Parfiles	mixture	4	29.492690
Parfiles	mixture	6	26.951831
Parfiles	mixture	8	26.129248
Parfiles	mixture	12	25.243248
Parslices	mixture	2	62.196268
Parslices	mixture	4	51.712936
Parslices	mixture	6	48.608128
Parslices	mixture	8	47.511146
Parslices	mixture	12	46.614016
Big Dataset			
Sequential	big	1	203.965315
Parfiles	big	2	106.621019
Parfiles	big	4	64.786197
Parfiles	big	6	47.356856
Parfiles	big	8	41.296116
Parfiles	big	12	34.857911
Parslices	big	2	152.873309
Parslices	big	4	126.234940
Parslices	big	6	118.614773
Parslices	big	8	114.997777
Parslices	big	12	112.589283