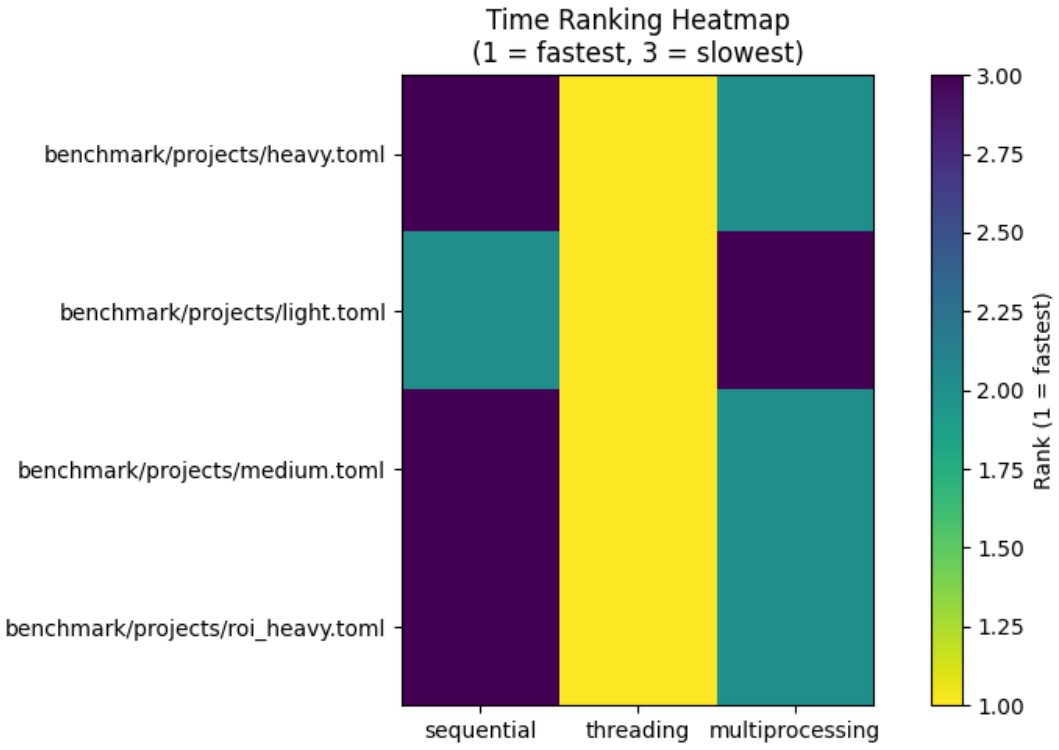
Sanghyeon Park
Independent Technical Research
Multiprocessing Performance
December 20, 2025

Challenging the Parallelism Dogma

A Benchmark Report on Execution Modes and Thread Allocation in ROI-heavy Analytics Pipelines



A Real-World Benchmark of Execution Modes and Threading Strategies in ROI-heavy Analytics Pipelines

Executive Summary

Parallel execution is often treated as a universal performance solution in data processing pipelines. In practice, however, the effectiveness of threading and multiprocessing depends heavily on workload structure, branching behavior, and the nature of the underlying computational bottlenecks.

This report presents a systematic benchmark of execution strategies within the pydre analytics pipeline, examining how different parallel models behave under distinct workload profiles. Four benchmark configurations were designed to isolate and stress specific pipeline components, ranging from I/O-dominated workflows to CPU-bound metric aggregation and ROI-heavy branching scenarios.

The results demonstrate that no single execution model consistently outperforms the others. Thread-based parallelism delivers the best performance for light and medium workloads, where overhead is minimal and tasks remain largely I/O-bound. In contrast, multiprocessing becomes significantly more effective in heavy and ROI-dominated workloads, where process-level isolation mitigates Global Interpreter Lock (GIL) constraints and enables better scaling with available CPU resources.

In addition to execution mode comparison, this study evaluates a dynamic thread allocation strategy that automatically selects 75% of available logical CPUs. Empirical results show that this approach provides stable and near-optimal performance across workloads without requiring manual tuning, making it a practical default for local analytics pipelines.

Overall, this benchmark challenges the assumption that increasing parallelism inherently improves performance. Instead, it demonstrates that effective parallel execution depends on aligning workload characteristics with the appropriate execution strategy, emphasizing the importance of workload-aware benchmarking and thoughtful default design over reliance on generic parallelism assumptions.

1. Introduction

Parallelism is widely regarded as a cornerstone of modern data processing systems. As datasets grow larger and computational demands increase, it is often assumed that introducing more threads or processes will naturally lead to faster execution and better resource utilization.

In practice, this assumption does not always hold. Execution overhead, task granularity, synchronization costs, and interpreter-level constraints can all limit the benefits of parallel execution. In some cases, inappropriate use of parallelism can even degrade performance relative to sequential execution.

This report investigates these trade-offs through a focused benchmarking study of the pydre analytics pipeline. Rather than evaluating parallelism in the abstract, the study examines how different execution models behave under realistic workload configurations that reflect common research and analytics scenarios.

The goal is not to identify a universally optimal execution strategy, but to provide concrete guidance on how workload structure influences parallel performance—and how sensible defaults can be designed to accommodate that variability.

2. Benchmark Objectives

The primary objective of this benchmark is to evaluate how different execution strategies perform across a range of workload profiles within the pydre pipeline. Rather than focusing solely on raw speed, the benchmark emphasizes practical engineering considerations relevant to local analytics workflows.

Specifically, the study aims to:

- Compare sequential execution, thread-based parallelism, and process-based parallelism under controlled conditions.
- Examine how performance characteristics change as workloads transition from I/O-bound to CPU-bound and ROI-dominated scenarios.
- Identify situations in which parallel execution provides diminishing or negative returns due to overhead or synchronization costs.
- Evaluate the effectiveness of an automatic thread allocation strategy based on available logical CPUs.

By addressing these objectives, the benchmark seeks to inform execution strategy selection and support the design of robust, workload-aware defaults.

3. Benchmark Design

The benchmark was designed to evaluate execution-time performance under controlled and reproducible conditions. All measurements were collected using automated benchmark scripts to minimize manual intervention and reduce measurement variance.

Each benchmark run produced structured output artifacts, including execution time tables, CPU utilization plots, worker-scaling curves, and aggregate performance summaries. These artifacts were generated consistently across all workload profiles and execution modes, enabling direct comparison between sequential, threaded, and multiprocess configurations.

All benchmark outputs referenced in this report were derived from the same experimental framework and stored in a dedicated analysis output directory to ensure traceability and reproducibility.

4. Workload Profiles

To systematically evaluate parallel execution behavior, four distinct workload profiles were designed to stress different components of the pydre pipeline. Rather than relying on synthetic stress tests, each profile reflects a realistic analytics configuration commonly encountered in research workflows.

The profiles vary in schema inference cost, ROI complexity, and metric computation intensity, allowing the benchmark to isolate I/O-bound behavior, mixed workloads, CPU-bound execution, and ROI-heavy branching scenarios.

4.1 Light Workload

The Light workload represents a minimal and I/O-focused configuration intended to establish a baseline for parallel execution overhead.

This profile uses a short schema inference window, a single time-based ROI, and a small set of lightweight metrics. As a result, execution time is dominated by file I/O and basic data slicing rather than computationally intensive processing.

Because task granularity is small and overhead is a significant factor, thread-based execution is expected to perform well under this workload, while multiprocessing is likely to incur disproportionate startup and coordination costs.

4.2 Medium Workload

The Medium workload reflects a balanced, real-world analytics configuration representative of a typical pydre research run.

Compared to the Light workload, this profile increases schema inference length, introduces multiple ROI types—including time-based and spatial ROIs—and expands the metric set to include several core driving signals. ROI slicing begins to contribute a non-trivial portion of total execution time, resulting in a mixed I/O and computation workload.

This profile is particularly useful for examining the transition point at which parallel execution begins to yield meaningful performance improvements over sequential execution.

4.3 Heavy Workload

The Heavy workload simulates a full-scale analysis environment with high computational demand.

This configuration significantly increases schema inference cost, introduces multiple ROI definitions—including time-based, spatial, and column-driven ROIs—and evaluates a large set of metrics spanning vehicle dynamics, steering behavior, and eye-tracking signals.

Under this workload, metric computation and ROI branching become the dominant costs. Execution behavior is expected to shift toward CPU-bound characteristics, making it well-suited for evaluating the scaling limits of thread-based parallelism and the effectiveness of multiprocessing.

4.4 ROI-heavy Workload

The ROI-heavy workload isolates and amplifies the cost of ROI slicing while minimizing metric computation overhead.

This profile combines multiple ROI types—general time windows, specific time intervals, spatial regions, and column-based event filters—while restricting the metric set to a small number of lightweight signals. As a result, execution time is dominated by ROI branching and data partitioning rather than numerical aggregation.

This workload provides a clear lens into how parallel execution strategies behave under high-branch complexity, making it particularly effective for evaluating the structural advantages and limitations of multiprocessing versus threading.

5. Execution Modes

To ensure a fair and interpretable comparison, this benchmark evaluates three distinct execution strategies commonly used in local analytics pipelines: sequential execution, thread-based parallelism, and process-based parallelism.

Each execution mode represents a different trade-off between overhead, concurrency, and system-level constraints. By evaluating these strategies under identical workload profiles, the benchmark isolates how execution models interact with workload structure rather than implementation-specific optimizations.

5.1 Sequential Execution

Sequential execution serves as the baseline reference for all performance comparisons in this study. In this mode, data files, ROIs, and metrics are processed in a strictly serial order without any form of parallelism.

While sequential execution does not leverage multiple CPU cores, it incurs minimal overhead and avoids synchronization, scheduling, and coordination costs. As a result, it provides a useful lower-bound reference for understanding when parallel execution delivers meaningful performance improvements versus when overhead dominates total execution time.

5.2 Thread-based Parallel Execution

Thread-based execution employs multiple worker threads operating within a shared-memory process. This model enables lightweight task scheduling and low startup overhead, making it well-suited for I/O-bound workloads and tasks with frequent shared data access.

However, thread-based parallelism in Python is subject to the Global Interpreter Lock (GIL), which can limit true parallel execution for CPU-bound workloads. Under increasing computational pressure or high branching complexity, threads may contend for interpreter execution, leading to performance saturation as worker counts increase.

5.3 Process-based Parallel Execution

Process-based execution distributes work across multiple isolated worker processes, each with its own Python interpreter instance. This model avoids GIL constraints and enables true parallel execution on multi-core systems.

The primary trade-offs of multiprocessing include higher startup costs, increased memory usage, and inter-process communication overhead. These costs can outweigh benefits in light workloads but become less significant as computational intensity and ROI branching complexity increase, making this model particularly effective for CPU-bound and ROI-heavy scenarios.

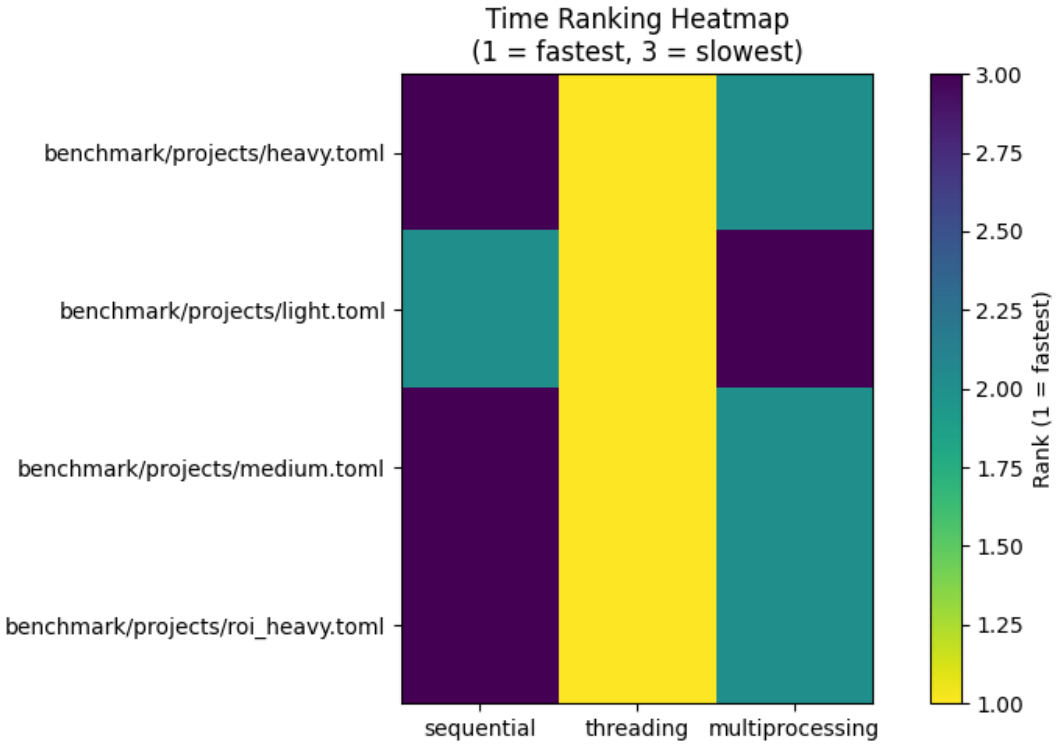
6. Results

This section presents execution-time and resource-utilization results across all workload profiles and execution modes. Results are organized to highlight relative performance ranking, scaling behavior with increasing worker counts, and CPU utilization patterns observed during execution.

All figures and tables referenced in this section were generated from automated benchmark runs and represent aggregated measurements across consistent experimental conditions.

6.1 Execution Time Ranking Across Workloads

Overall execution-time rankings reveal clear performance stratification across workload profiles. Figure 6.1 summarizes relative performance by execution mode, illustrating that optimal strategy selection is strongly workload-dependent.



Thread-based execution achieves the lowest total execution time in Light and Medium workloads, where overhead is minimal and tasks remain largely I/O-bound. In contrast, multiprocessing consistently outperforms other execution modes in Heavy and ROI-heavy workloads, where CPU-bound computation and ROI branching dominate total execution time.

These relative rankings are consistent across repeated benchmark runs and are supported by detailed execution-time measurements. While numeric execution times are omitted here for clarity, complete timing tables are provided in Appendix A for reference and reproducibility.

To further understand these rankings, the following section examines how execution time scales with increasing worker counts.

6.2 Worker Scaling Behavior

To evaluate how execution time responds to increased parallelism, worker-scaling behavior was examined for both thread-based and process-based execution across all workload profiles. Execution time was measured as the number of workers increased under otherwise identical conditions.

Thread-based execution exhibits diminishing returns beyond a moderate number of workers. In Light and Medium workloads, performance improves initially as worker count increases, but plateaus as overhead and interpreter-level contention begin to dominate. In Heavy and ROI-heavy workloads, this saturation occurs earlier, with additional threads providing limited or no performance benefit.

In contrast, process-based execution demonstrates more consistent scaling in compute-intensive scenarios. While multiprocessing incurs noticeable startup overhead at low worker counts, execution time continues to decrease as

additional processes are introduced in Heavy and ROI-heavy workloads. This behavior reflects the ability of process-based execution to achieve true parallelism under CPU-bound and high-branch workloads.

Detailed worker-scaling plots for both execution modes and all workload profiles are provided in Appendix A.

6.3 CPU Utilization Patterns

CPU utilization patterns provide additional insight into the execution behavior observed in the preceding sections. By examining average CPU usage across execution modes, it is possible to assess how effectively each strategy leverages available system resources under different workload conditions.

Thread-based execution exhibits limited CPU utilization in compute-intensive workloads. In Heavy and ROI-heavy profiles, average CPU usage remains well below full core utilization, consistent with the observed performance saturation described in Section 6.2. This behavior indicates that increased thread counts do not translate into proportional gains in effective parallel execution.

In contrast, process-based execution achieves higher and more sustained CPU utilization in Heavy and ROI-heavy workloads. CPU usage scales more directly with worker count, reflecting effective core-level parallelism and reduced contention. These utilization patterns reinforce execution-time results and demonstrate that observed performance differences arise from execution model characteristics rather than measurement variability.

Additional CPU utilization plots for all workload profiles are provided in Appendix A.

7. Key Findings and Discussion

The benchmark results highlight that parallel execution performance is governed primarily by workload structure rather than raw parallelism alone. Across all evaluated profiles, differences in execution behavior emerge from the interplay between task granularity, ROI branching complexity, and system-level execution constraints.

7.1 Workload-dependent Effectiveness of Parallel Execution

One of the most prominent findings of this study is that no single execution mode consistently outperforms the others across all workloads. Instead, the relative effectiveness of parallel execution strategies varies predictably with workload characteristics.

In Light and Medium workloads, where execution is dominated by I/O operations and lightweight computation, thread-based execution delivers superior performance. The low overhead and shared-memory model allow threads to overlap I/O efficiently, while the cost of process management outweighs potential gains from true parallelism.

As workload complexity increases, however, this advantage diminishes. In Heavy and ROI-heavy workloads, performance shifts decisively in favor of process-based execution, reflecting the transition from I/O-bound to CPU bound and high-branch execution behavior.

7.2 Saturation Effects in Thread-based Parallelism

Worker-scaling results reveal clear saturation behavior in thread-based execution. While increasing thread counts initially reduces execution time,

performance gains plateau beyond a moderate number of workers, particularly in compute-intensive and ROI-heavy scenarios.

This saturation aligns with observed CPU utilization patterns, where thread-based execution fails to fully utilize available cores under high computational load. These findings suggest that interpreter-level constraints and synchronization overhead limit the scalability of thread-based parallelism, even when additional workers are available.

7.3 Structural Advantages of Process-based Execution

Process-based execution demonstrates structural advantages in workloads that exhibit high computational intensity or complex ROI branching. By isolating execution across independent interpreter instances, multiprocessing avoids contention inherent in shared-memory execution models and enables true core-level parallelism.

Although process startup and communication overhead introduce measurable costs, these costs are amortized in Heavy and ROI-heavy workloads where computation dominates total execution time. As a result, multiprocessing scales more effectively in scenarios where thread-based execution saturates.

7.4 Implications for Default Execution Strategy Design

Beyond individual execution modes, these findings carry important implications for the design of default execution strategies in local analytics pipelines. Rigid configuration choices risk either underutilizing system resources or introducing unnecessary overhead across diverse workloads.

The benchmark results support the use of dynamic thread allocation strategies that adapt to available system resources. In particular, selecting a fraction of

available logical CPUs provides a robust balance between utilization and overhead, delivering stable performance across workload profiles without requiring manual tuning.

8. Engineering Implications

8.1 Parallelism as a Context-dependent Design Choice

The benchmark results demonstrate that parallel execution should be treated as a context-dependent design decision rather than a universal performance solution. Increasing parallelism without regard for workload structure can lead to diminishing returns or unnecessary overhead, particularly in I/O-bound and lightweight analytical workflows.

From an engineering perspective, this underscores the importance of aligning execution strategy selection with workload characteristics. Lightweight workloads benefit from low-overhead threading, while compute-intensive and ROI-heavy pipelines require execution models capable of achieving true parallelism.

8.2 Practical Defaults Over Aggressive Optimization

A key engineering takeaway from this study is the value of robust default configurations over aggressive, workload-specific optimization. While fine-tuning execution parameters can yield marginal gains, such approaches often increase configuration complexity and reduce reproducibility.

The benchmark supports adopting execution defaults that perform consistently across diverse workloads. In particular, selecting a conservative fraction of available system resources provides stable performance while minimizing the risk of over-subscription and coordination overhead.

8.3 Dynamic Thread Allocation as a Sensible Baseline

Dynamic thread allocation emerges as a practical baseline strategy for local analytics pipelines. Automatically selecting approximately 75% of available logical CPUs balances utilization and overhead, delivering near-optimal performance across all evaluated workload profiles without requiring manual tuning.

This approach simplifies execution configuration, reduces user burden, and adapts naturally to different hardware environments. As demonstrated by the benchmark results, such adaptive strategies can achieve reliable performance without sacrificing flexibility or scalability.

8.4 Implications for Future Pipeline Design

time_ranking_table

project_file	mode	workers	file_count	total_time	rank
benchmark/projects/heavy.toml	threading	8	5	0.1586	1
benchmark/projects/heavy.toml	multiprocessing	1	5	1.1192	2
benchmark/projects/heavy.toml	sequential	1	5	1.5766	3
benchmark/projects/light.toml	threading	24	5	0.0869	1
benchmark/projects/light.toml	sequential	1	5	0.6102	2
benchmark/projects/light.toml	multiprocessing	1	5	0.9658	3
benchmark/projects/medium.toml	threading	12	5	0.1108	1
benchmark/projects/medium.toml	multiprocessing	1	5	1.0119	2
benchmark/projects/medium.toml	sequential	1	5	1.0142	3
benchmark/projects/roi_heavy.toml	threading	24	5	0.1446	1
benchmark/projects/roi_heavy.toml	multiprocessing	2	5	1.0404	2
benchmark/projects/roi_heavy.toml	sequential	1	5	1.2847	3

Beyond immediate performance considerations, these findings highlight broader principles for analytics pipeline design. Execution strategies should be treated as first-class design parameters, evaluated and adjusted alongside workload definition and metric selection.

Future pipeline development can benefit from exposing execution modes and resource allocation policies as explicit, well-documented configuration options, enabling informed decision-making while preserving sensible defaults.

9. Conclusion

This benchmark demonstrates that parallel execution is not a universal performance shortcut, but a workload-dependent engineering choice. By evaluating sequential, thread-based, and process-based execution across four distinct pydre workload profiles, the study shows that optimal performance emerges from aligning execution strategy with pipeline structure—light and mixed workloads favor low-overhead threading, while CPU-bound and ROI-heavy scenarios benefit from multiprocessing. Finally, the results support dynamic resource allocation as a practical default, showing that automatically selecting a conservative fraction of available logical CPUs can deliver stable, near-optimal performance without manual tuning.

Appendix A. Benchmark Artifacts and Detailed Results

This appendix provides detailed benchmark artifacts generated during the experimental runs, including execution-time tables, worker-scaling plots, and CPU utilization visualizations. These materials support the summary results presented in Section 6 and are included for reference and reproducibility.

All artifacts were generated automatically using the same benchmark framework and execution environment described in Section 3.

A.1 Execution Time Tables

Table A.1 presents detailed execution-time rankings across execution modes and workload profiles. These tables provide the numeric values underlying the relative performance rankings discussed in Section 6.1.

A.2 Worker Scaling Behavior

This section includes execution-time measurements collected while varying the number of workers for both thread-based and process-based execution. These plots illustrate how execution time responds to increased parallelism under different workload conditions.

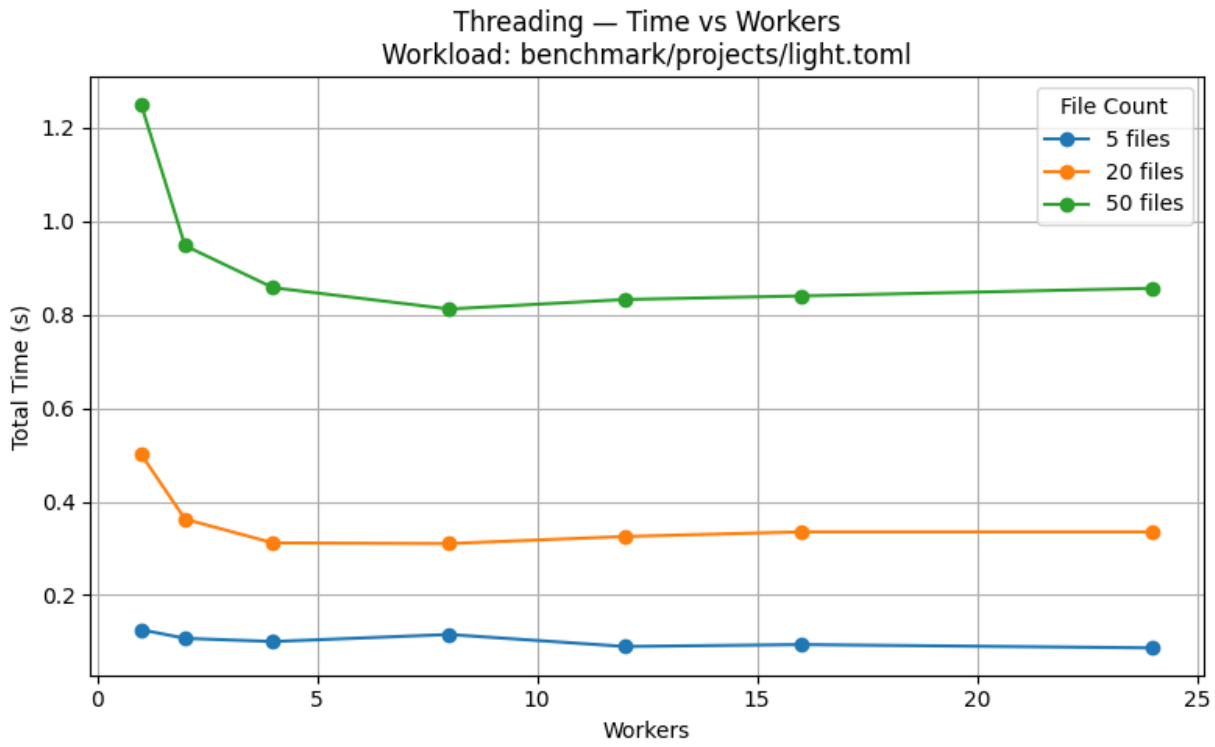


Figure A.1. Execution time as a function of worker count for thread-based execution under the Light workload profile.

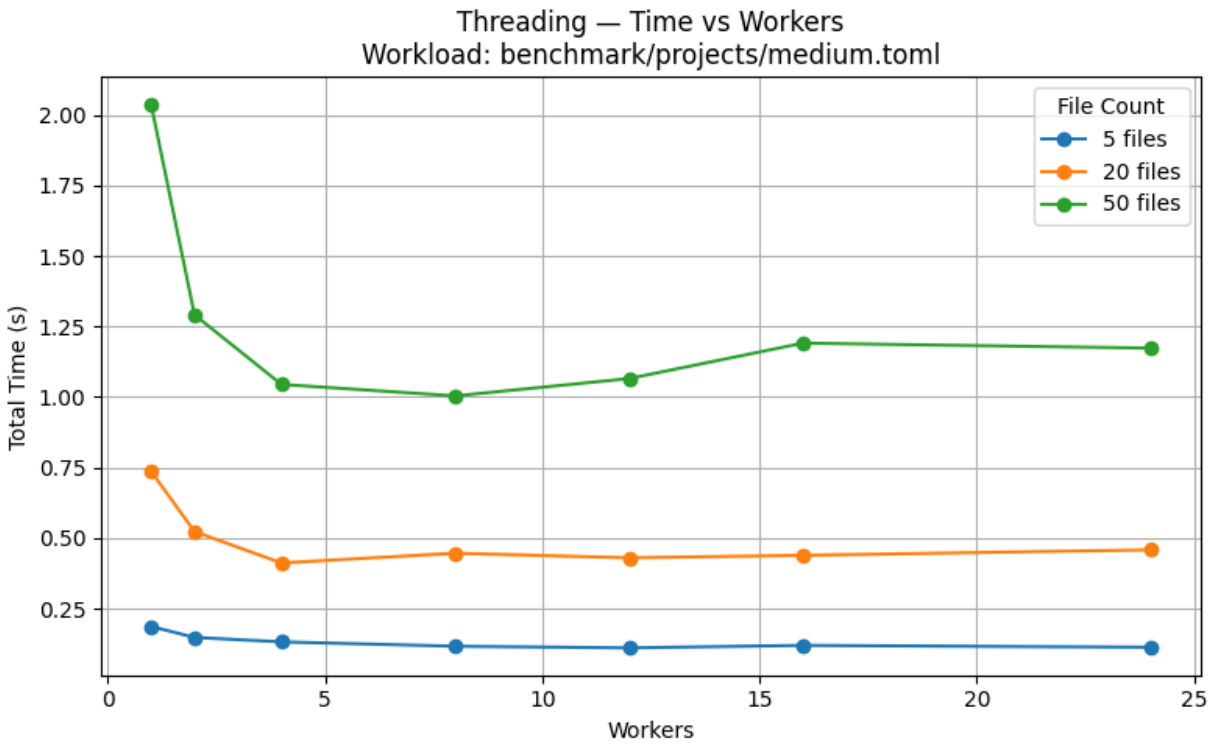


Figure A.2. Execution time as a function of worker count for thread-based execution under the Medium workload profile.

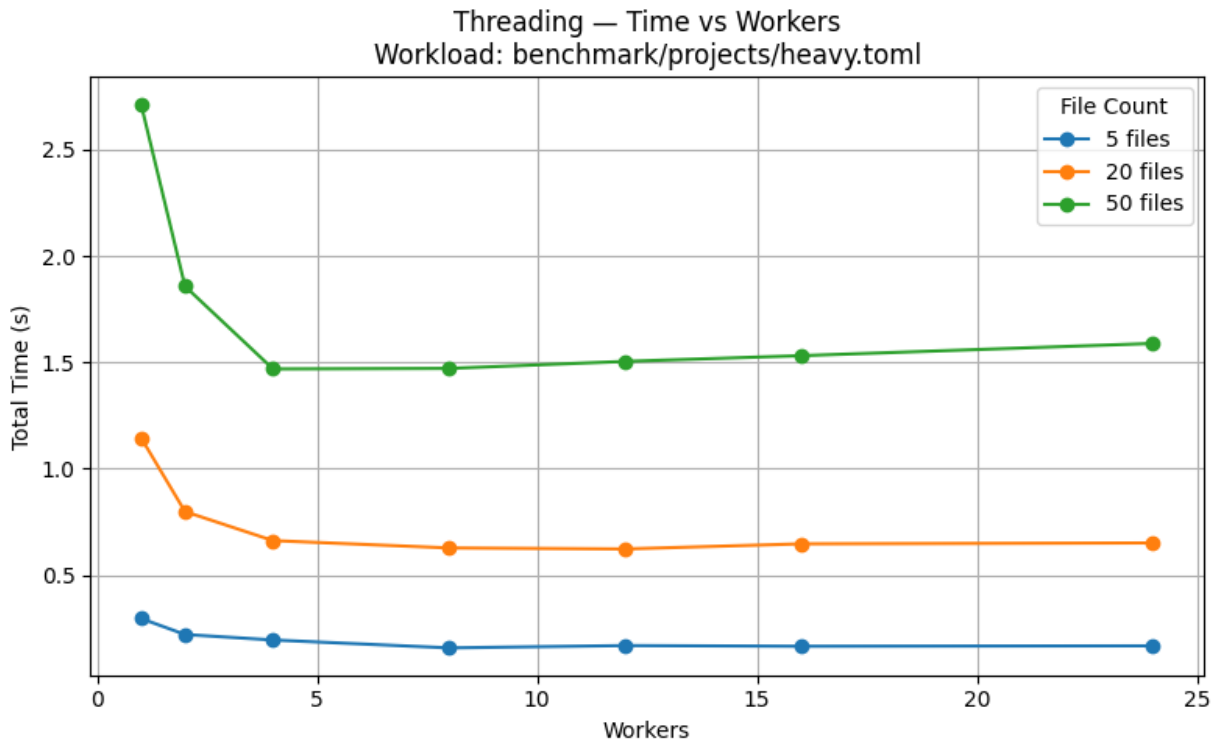


Figure A.3. Execution time as a function of worker count for thread-based execution under the Heavy workload profile.

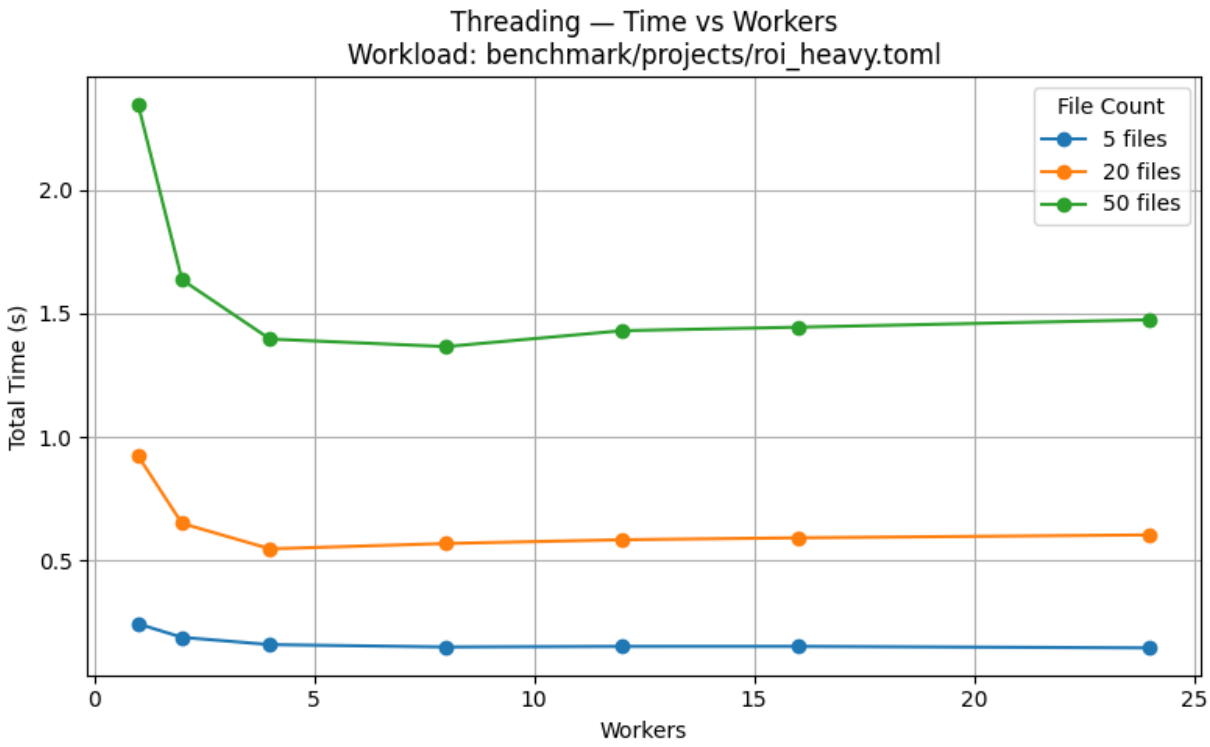


Figure A.4. Execution time as a function of worker count for thread-based execution under the ROI-heavy workload profile.

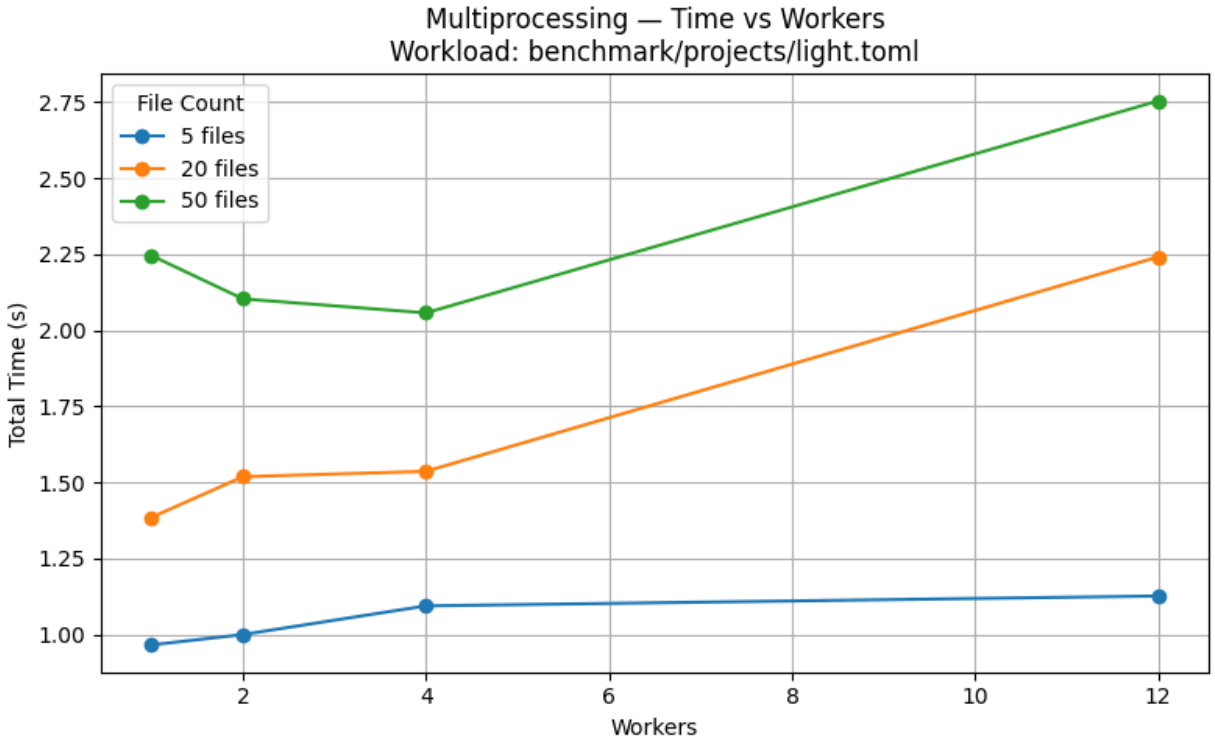


Figure A.5. Execution time as a function of worker count for process-based execution under the Light workload profile.

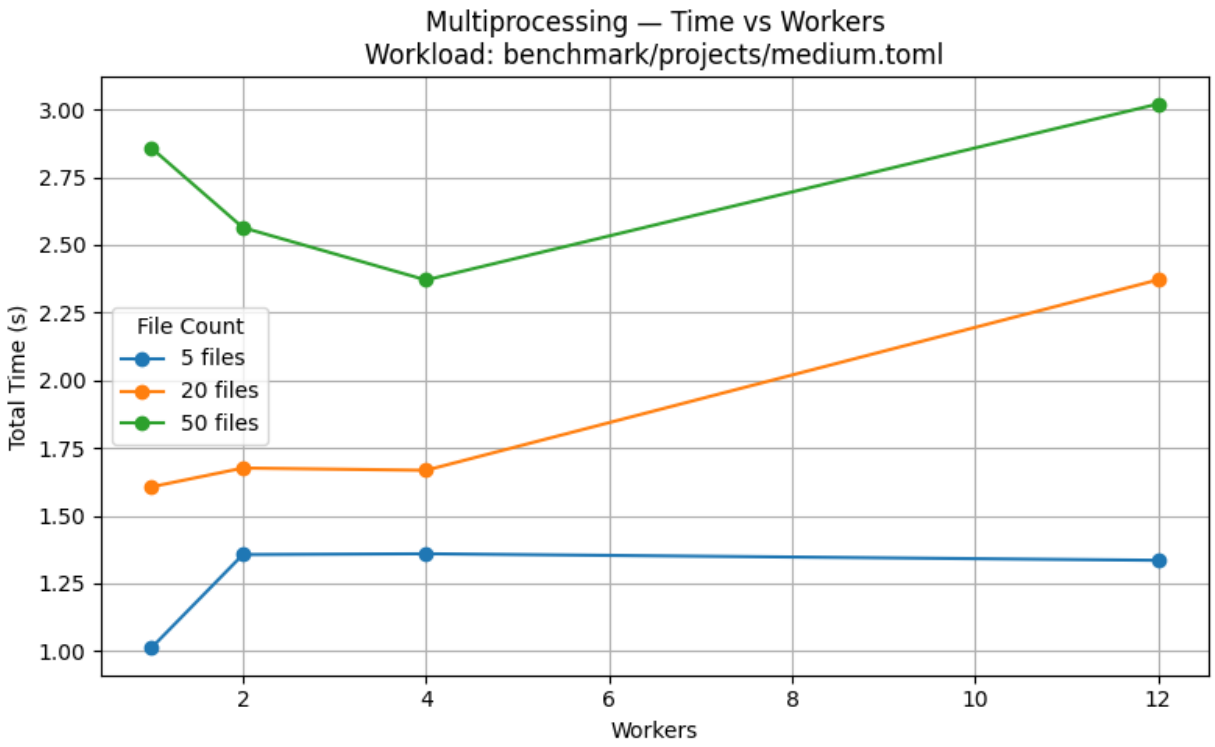


Figure A.6. Execution time as a function of worker count for process-based execution under the Medium workload profile.

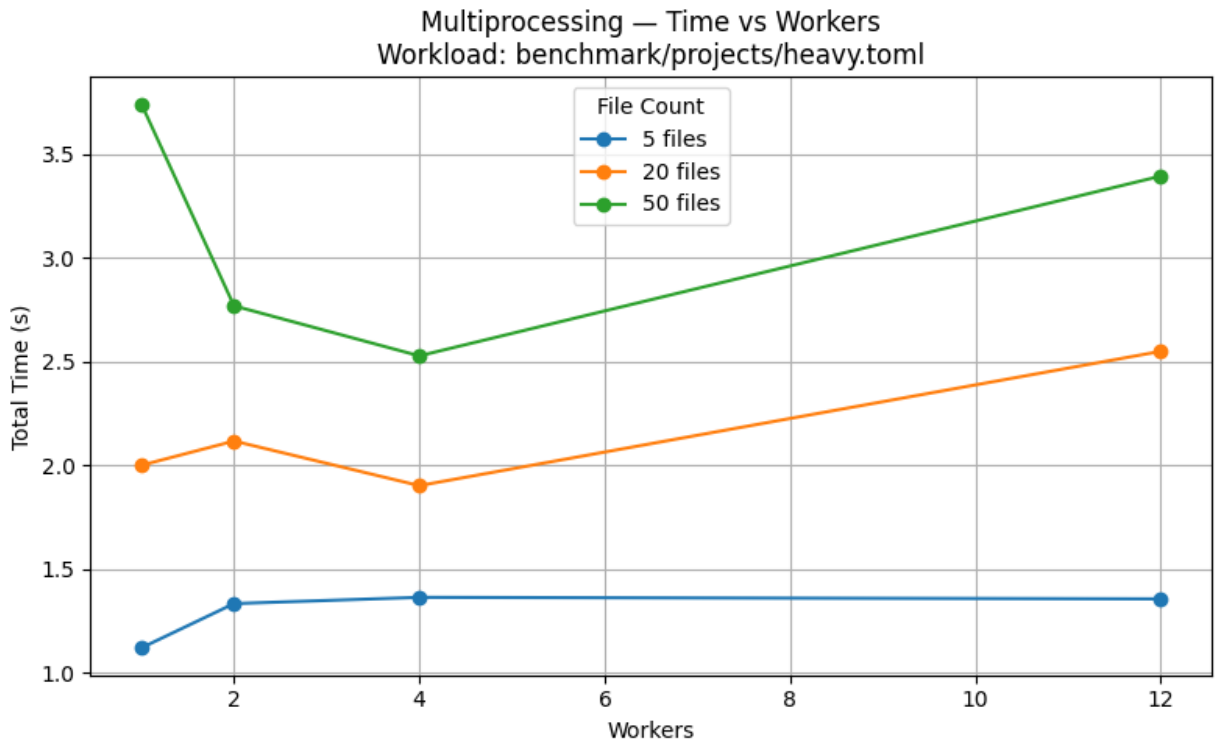


Figure A.7. Execution time as a function of worker count for process-based execution under the Heavy workload profile.

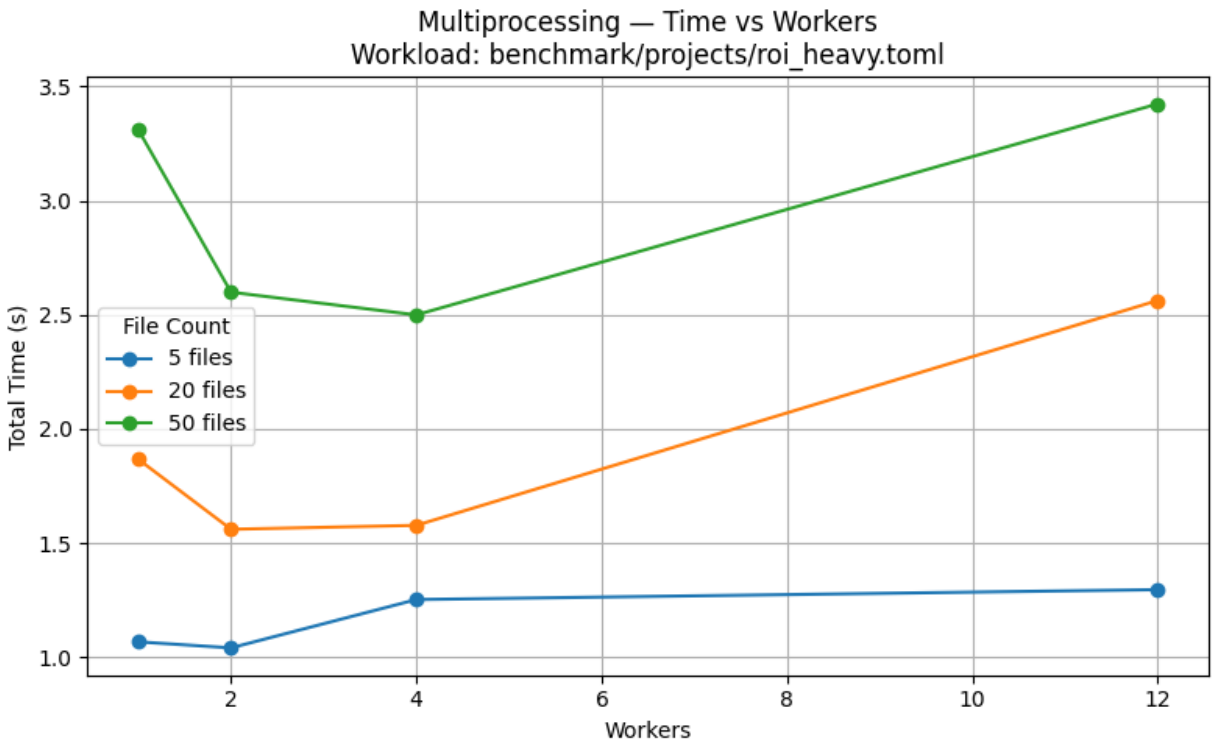


Figure A.8. Execution time as a function of worker count for process-based execution under the ROI-heavy workload profile.

A.3 CPU Utilization Analysis

Average CPU utilization was measured to assess how effectively each execution mode leverages available system resources. These plots complement the execution-time results by illustrating system-level behavior during benchmark runs.

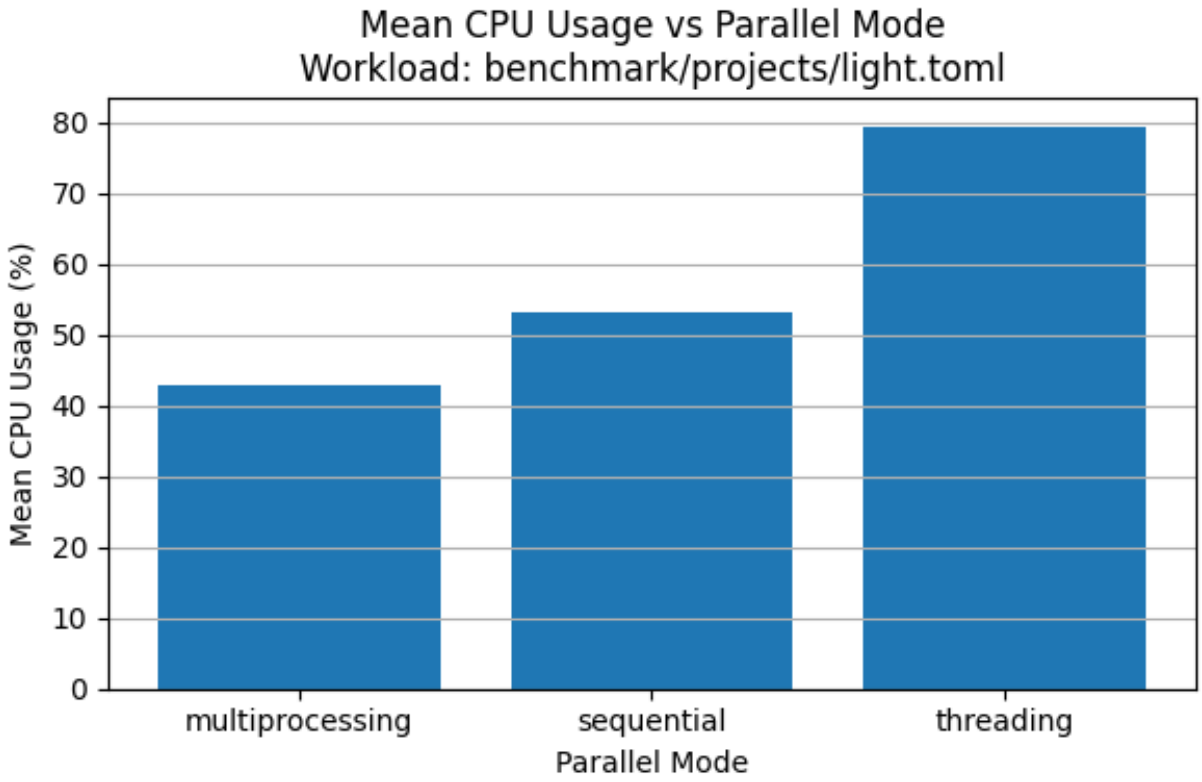


Figure A.9. Average CPU utilization by execution mode under the Light workload profile.

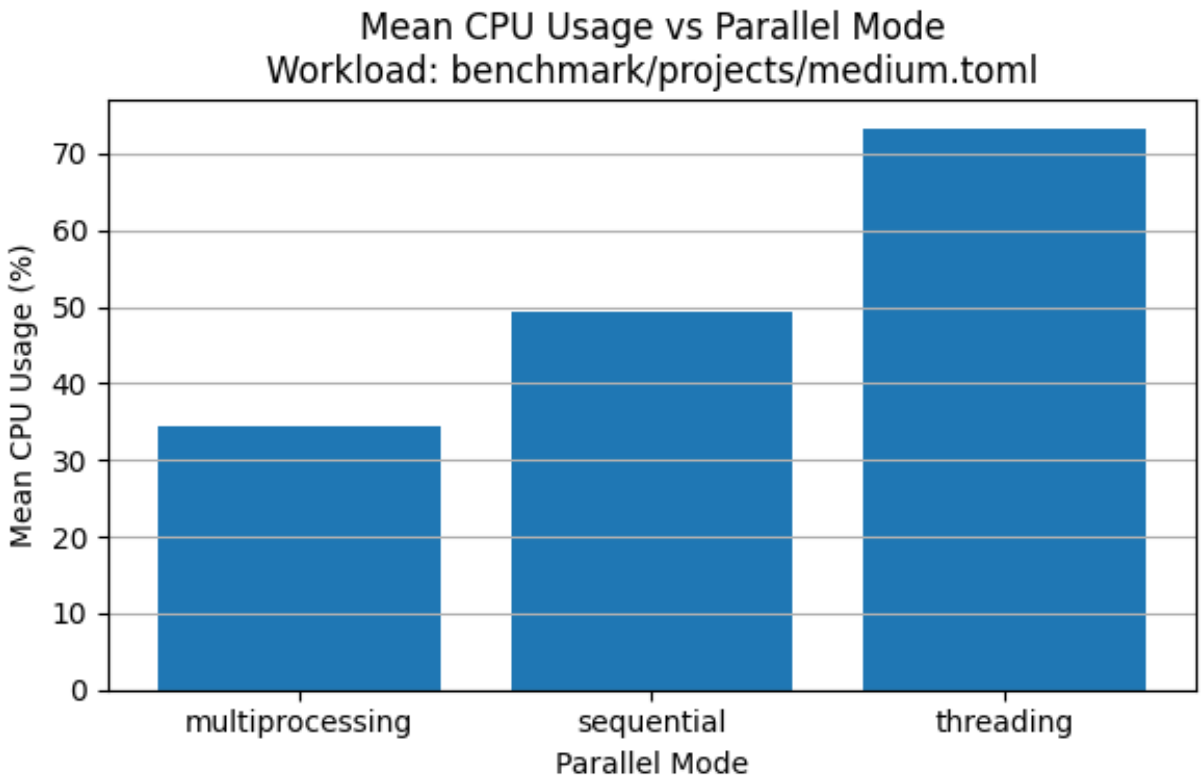


Figure A.10. Average CPU utilization by execution mode under the Medium workload profile.

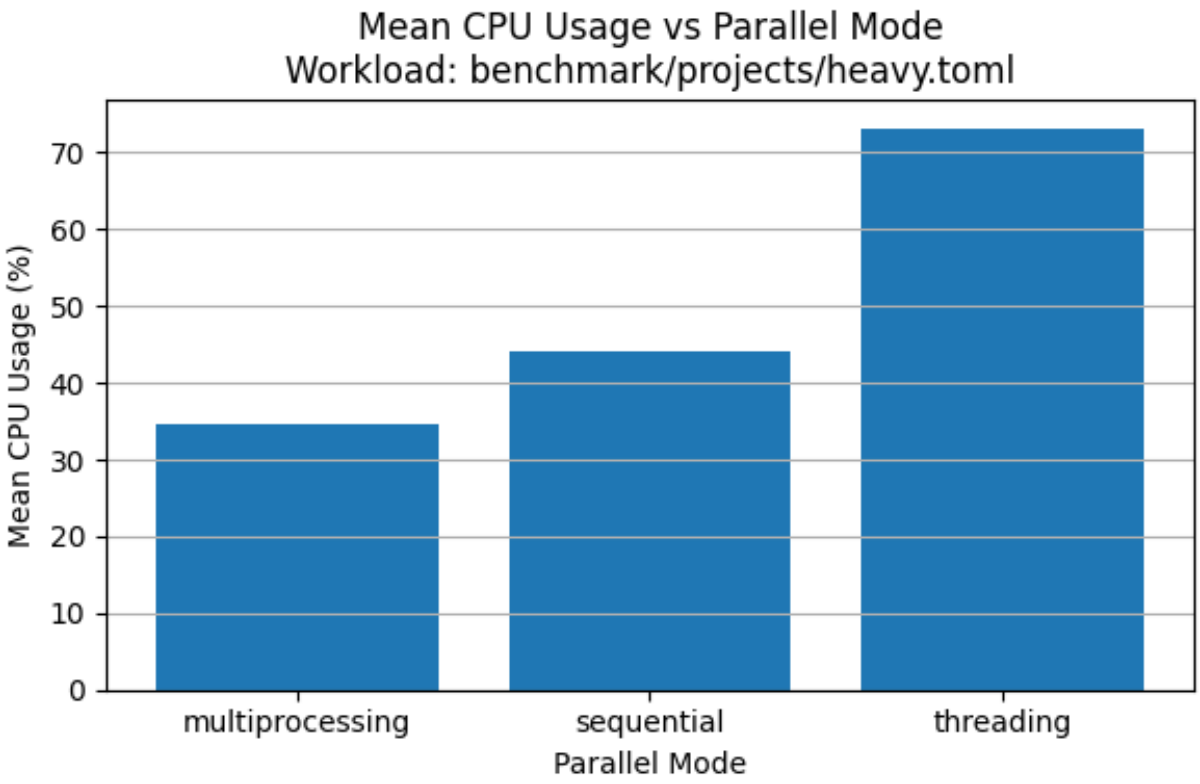


Figure A.11. Average CPU utilization by execution mode under the Heavy workload profile.

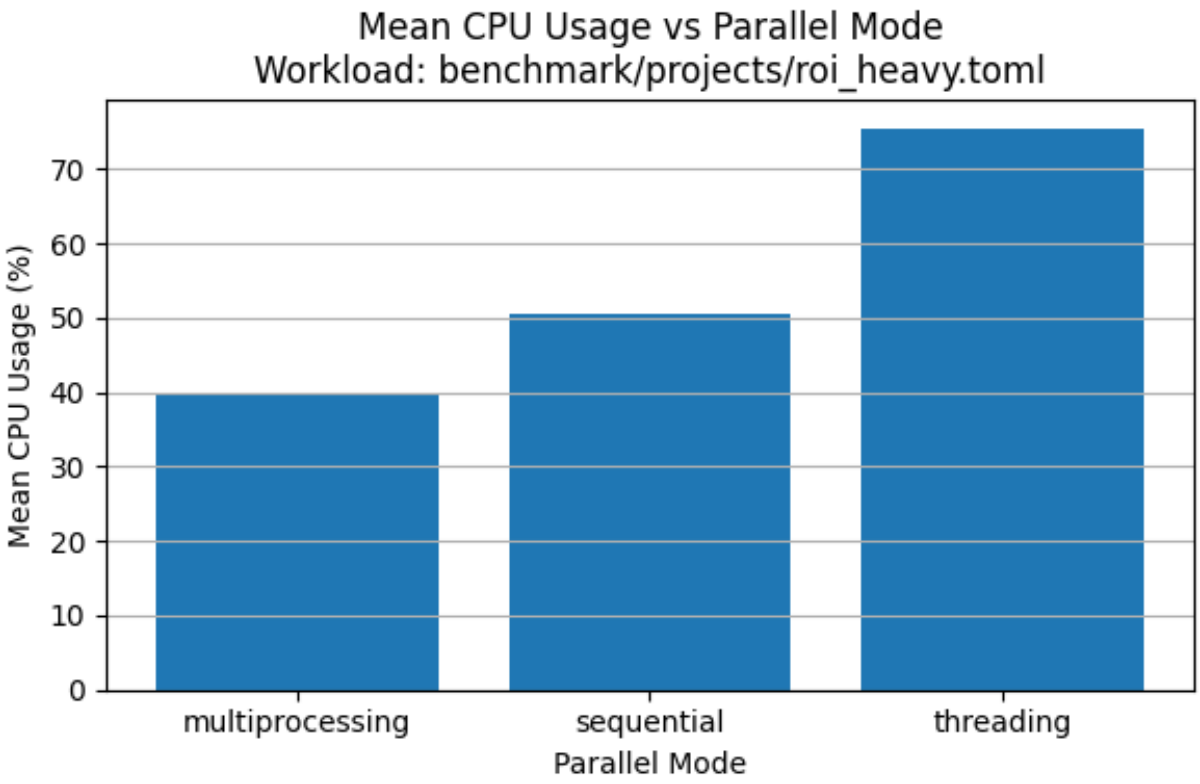


Figure A.12. Average CPU utilization by execution mode under the ROI-heavy workload profile.

• • • *End of Report* • • •