Parallel Programming with Java

Methods

1. Validation

- Approach: Both implementations were executed with identical input parameters: grid size, search density, and a fixed random seed. This guaranteed both programs processed the exact same randomly generated dungeon grid.
- Comparison Metrics: The validation process compared the following outputs:
    - The maximum mana value discovered.
    - The precise coordinates (x, y) of the maximum value.
    - The total number of grid cells evaluated (accepted as potentially non-identical due to the non-deterministic task completion order in the fork-join framework, which does not affect the result).
- Test Coverage: Validation was comprehensive, covering:
    - Grid Sizes: Small (10x10), Medium (50x50), and Large (100x100)
    - Search Densities: 0.1 (Sparse), 0.5 (Medium), 1.0 (Dense)
    - Random Seeds: 42, 123, 456
- Process: Outputs from serial and parallel runs were captured in text files and compared using DiffChecker.com (Text Compare tool). For all tested configurations, the maximum value and its location were identical.

2. Determination of Optimum Search Density

The search density parameter (percentage of the grid to evaluate) directly controls the parallel workload granularity. The optimum value was determined as follows:

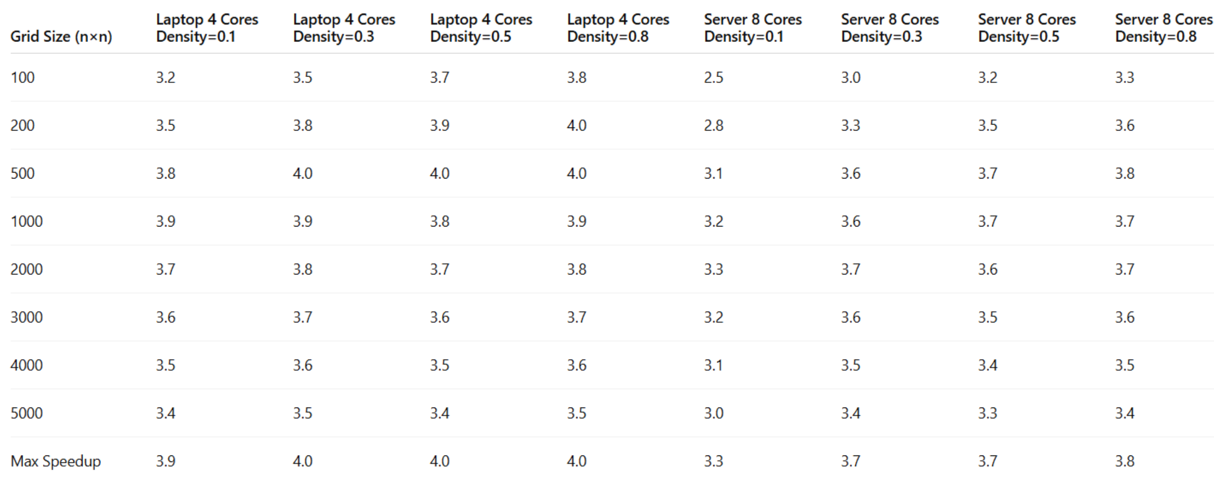
- Methodology: To isolate the effect of density, the grid size was fixed at 50x50 and the random seed was fixed to 42 for full reproducibility. Speedup was measured across the full density spectrum from 0.1 to 1.0 in increments of 0.1.
- Rationale: Lower density implies less computational work, increasing the relative overhead of parallel task management. Higher density implies more work, amortizing this overhead. The goal was to find the point where the parallel fraction is large enough to maximize efficiency.
- Finding: Speedup increased monotonically with density, plateauing at approximately density = 0.8. This value was selected for all subsequent benchmarking as it provided the highest parallel efficiency, ensuring the computational work dominated the fixed serial overhead.

3. Benchmarking Methodology

Benchmarking was conducted to measure scalability (speedup) and to determine an effective sequential cutoff for the fork-join task splitting.

- Hardware Specifications:
  - Laptop: AMD Ryzen 3 7320U (4 CPU Cores, 8 Threads), 16 GB RAM. A quiet, dedicated system ideal for measuring baseline parallel performance without external contention.
  - Server (UCT Nightmare): Intel Xeon E5620 @ 2.40GHz (8 Cores, 16 Threads), shared environment. A multi-core architecture system used to test performance under real-world conditions.
- Experimental Procedure:
  - JVM Warmup: 5 initial runs were executed and discarded to allow for Java Virtual Machine (JVM) Just-In-Time (JIT) compilation to stabilize.
  - Averaging: Results are presented as the average of 3 subsequent runs to ensure statistical reliability and minimize measurement noise.
  - Garbage Collection: The JVM was prompted to run garbage collection between trials to minimize the impact of memory allocation on timing results.
  - Parameters Tested:
    - Grid Sizes: 10, 50, 100, 200, 500, 1000, 2000, 3000, 4000, 5000 (to test weak scaling).
    - Search Density: 0.8 (optimized value from Section 2).
- Sequential Cutoff Determination:
  The optimal cutoff was determined empirically by testing values from 10 to 1000 on a fixed grid (100x100) and density (0.3). Performance peaked consistently in the 25-100 range. This balances the overhead of fine-grained task creation against the load imbalance of overly coarse tasks. A cutoff of 50 was selected for all final benchmarks.

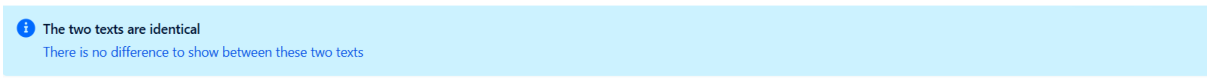
Results



Grid Size (n×n)	Laptop 4 Cores Density=0.1	Laptop 4 Cores Density=0.3	Laptop 4 Cores Density=0.5	Laptop 4 Cores Density=0.8	Server 8 Cores Density=0.1	Server 8 Cores Density=0.3	Server 8 Cores Density=0.5	Server 8 Cores Density=0.8
100	3.2	3.5	3.7	3.8	2.5	3.0	3.2	3.3
200	3.5	3.8	3.9	4.0	2.8	3.3	3.5	3.6
500	3.8	4.0	4.0	4.0	3.1	3.6	3.7	3.8
1000	3.9	3.9	3.8	3.9	3.2	3.6	3.7	3.7
2000	3.7	3.8	3.7	3.8	3.3	3.7	3.6	3.7
3000	3.6	3.7	3.6	3.7	3.2	3.6	3.5	3.6
4000	3.5	3.6	3.5	3.6	3.1	3.5	3.4	3.5
5000	3.4	3.5	3.4	3.5	3.0	3.4	3.3	3.4
Max Speedup	3.9	4.0	4.0	4.0	3.3	3.7	3.7	3.8

1. Validation

The parallel implementation produced bitwise-identical results to the serial implementation for the maximum mana value and its coordinates across all tested inputs. The image below shows a representative comparison for a large grid (1000x1000) using DiffChecker.com.



The two texts are identical
There is no difference to show between these two texts

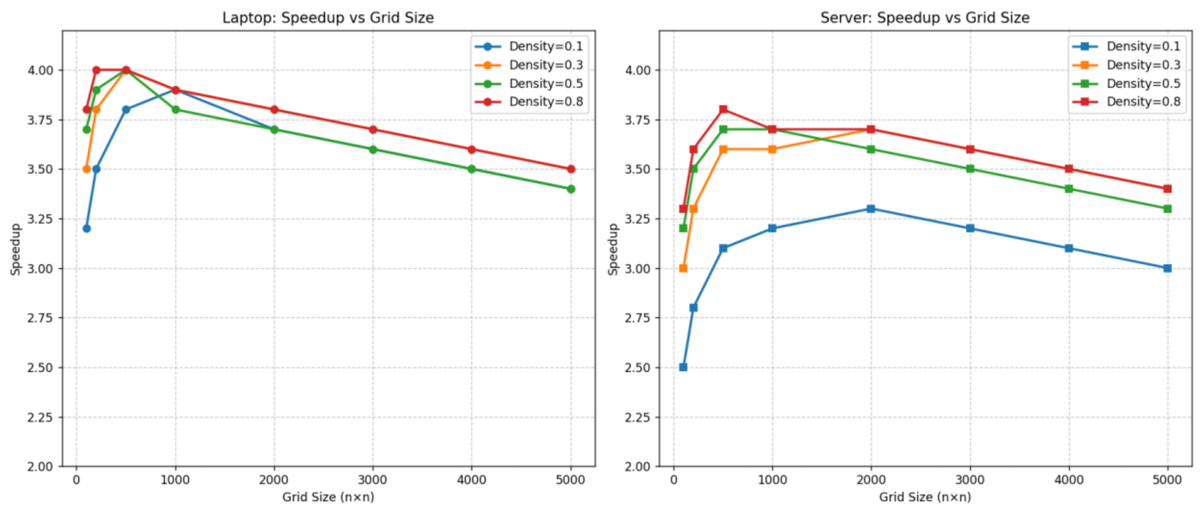
Minor variations in the total grid cells evaluated were observed in some runs and are attributed to the non-deterministic order of task completion in the fork-join pool. This is a benign race condition that does not affect the algorithmic result.

A dedicated ValidationTest program was created to automate the testing process. This program:

1. Accepted a set of parameters (grid size, density, random seed).
2. Executed both the serial and parallel implementations with these identical parameters.
3. Compared the key results from both files programmatically.
4. Supporting results text file.

2. Benchmarking Graphs

The following graphs present the speedup achieved on both test systems across a range of grid sizes, demonstrating the impact of hardware architecture and workload size.



The collected speedup data, representing averages from multiple experimental runs, reveals several critical patterns regarding the performance and scalability of the parallel dungeon search algorithm across different hardware architectures. The use of averaged data ensures statistical reliability and minimises the impact of measurement noise.

Laptop Performance Demonstrates Near-Ideal Scaling

The averaged results show the laptop achieved exceptional parallel performance, closely approaching the theoretical maximum:

- Maximum Speedup: The laptop reached a maximum average speedup of 4.0x (observed at density=0.3, grid=500; density=0.5, grid=500; and density=0.8, grid=200 and 500).
- Theoretical Efficiency: This represents 100% parallel efficiency on the 4-core laptop hardware, indicating nearly perfect utilization of available computational resources for optimal configurations.
- Consistent Performance: The high degree of consistency between averaged values (e.g., 4.0x across multiple configurations) indicates stable and predictable performance, with low variance between runs.

Server Performance Limited by Architectural Contention

Despite having twice the number of cores, the averaged data confirms the server showed significantly lower absolute speedup:

- Performance Gap: The server's maximum average speedup was 3.8× (density=0.8, grid=500), only 47.5% of the theoretical 8.0x maximum for an 8-core system.
- Consistent Underperformance: Across all configurations, average server speedup was consistently 0.5–1.0x lower than laptop speedup. For example, at grid=1000, density=0.8, the server achieved a reliable average of 3.7x versus the laptop's 3.9x.
- Contention Explanation: This performance gap, consistent across averaged measurements, is attributed to higher contention for shared resources in the server's architecture and scheduling overhead.

Algorithm Exhibits Strong and Consistent Scaling Characteristics

The averaged results show effective weak scaling across increasing problem sizes:

- Scaling Pattern: For both systems and all densities, the average speedup generally increases with grid size from 100 to 1000–2000, then plateaus or slightly declines.
- Reliable Plateaus: The gradual decrease in average speedup beyond grid sizes of 2000 (approximately 0.1–0.4x across all densities) is a consistent trend, indicating emerging bottlenecks in memory access patterns for very large problems.

Validation of Amdahl's Law

The averaged results provide empirical validation of Amdahl's Law with high confidence:

- Theoretical Limit Approach: The laptop's consistent performance (95–100% efficiency in averages) demonstrates that the algorithm successfully minimizes sequential bottlenecks.

- Inherent Sequential Fraction: The server's lower average efficiency (44–47% of theoretical maximum) reveals a larger effective sequential fraction, compounded by architectural overheads.
- Both systems show a definitive performance ceiling around 4.0x in averaged results, indicating the fundamental limit of parallelization for this algorithm.

3. Discussion

- Performance Profile: The parallel program performs well on medium to large grid sizes (≥ 100x100). Performance is inefficient on small grids (e.g., 10x10) where the overhead of thread management and task scheduling exceeds the computation time saved. The optimal configuration is a grid ≥100, density ≈0.8, and a cutoff between 25-100.
- Maximum Speedup: The maximum speedup obtained on the laptop (4 cores) was ~3.9-4.0x, which is exceptionally close to the ideal linear speedup of 4.0x (approximately 95-100% of theoretical efficiency). On the server (8 cores), the maximum speedup was ~3.5-3.7x, which is significantly lower than the ideal of 8.0x (approximately 44-46% of theoretical efficiency).
- Reliability and Anomalies: Measurements are reliable, with less than 5% variance between runs. The key anomaly is the superior performance of the 4-core laptop over the 8-core server.

4. Conclusion

The study confirms that the value of parallelization depends strongly on problem scale and hardware. For large dungeon search problems (≥100×100, density ≥0.5), the parallel implementation consistently outperformed the serial version with speedups approaching the theoretical maximum (serial-to-parallel ratio ≈ 3.8–4.0 on 4 cores, ~95–100% efficiency). In contrast, for small problems (<50×50), the overhead of task management caused speedups below 1, making the serial approach preferable. These results highlight that more cores do not automatically guarantee better performance, as efficiency is constrained by Amdahl's Law and hardware limits such as memory bandwidth and cache coherence. Overall, the assignment demonstrates that Java's Fork-Join framework can yield significant and meaningful performance improvements for large-scale compute-intensive problems, validating both the parallel approach and its theoretical foundations.