CSC2002S PCP1 Assignment Report

Parallel Programming with Java: Hunt for the Dungeon Master

Emmanuel Basua
Student Number: BSXEMM001

August 2025

This report presents a performance analysis of a parallel implementation of the Dungeon Hunter pathfinding algorithm. The algorithm finds optimal dungeon master locations with minimal mana requirements using grid-based search tech- niques. We implemented both serial and parallel versions in Java and conducted comprehensive benchmarking to evaluate speedup, efficiency, and correctness across various input sizes and search densities. The scripts used to test the and collect benchmarking data where all written by artificial intelligence with the logic provided by me. Artificial intelligence was also used to test whether the DungeonHunterParallel program was doing the same thing as the serial version, before the scripts were developed.

1 Methods

1.1 Validation

The parallel algorithm was validated through multiple approaches to ensure correctness:

Image Comparison Validation:
Both serial and parallel implementations generate visualization images (`visualiseSearch.png` and `visualiseSearchPath.png`). A comprehensive image comparison tool was developed that:

- Executes both versions with identical parameters (grid size, search factor, random seed)
- Captures generated visualization images
- Performs pixel-by-pixel comparison using MD5 hashing and statistical analysis

Solution Verification:
For each test case, the following metrics were compared between serial and parallel versions:

- Final dungeon master location (x, y coordinates)
- Maximum mana value found
- Number of grid points evaluated
- Convergence to the same local/global maximum

Statistical Validation:
Multiple random seeds (3, 60, 141) were used across different grid sizes and search factors to ensure consistent behavior across various scenarios.

1.2 Optimum Search Density

The optimum search density was determined through empirical analysis across multiple search factors:

Search Factor Range: Three search density factors were tested: 0.1, 1.0, and 3.0

- Factor 0.1: Low search density (10% of grid area searches)
- Factor 1.0: Moderate search density (100% of grid area searches)

- Factor 3.0: High search density (300% of grid area searches)

Selection Criteria: The choice balanced computational cost against solution quality. Factor 1.0 was identified as optimal for benchmarking as it provides sufficient search coverage while maintaining reasonable execution times across all grid sizes.

1.3 Benchmarking

Test Environment:

- **Laptop:** Multi-core system with 12 available CPU cores
- **Departmental Server:** Multi-core system with 8 available CPU cores
- Java Version: OpenJDK with standard threading library
- Test Parameters: Grid sizes from 10×10 to 315×315, three search factors, three random seeds
- Measurement Method: Each configuration run 3 times, average execution time calculated

Sequential Cutoff Determination: Analysis showed that parallelization benefits begin at grid sizes $\geq 40 \times 40$. Below this threshold, thread creation overhead exceeds computational benefits.

Measurement Methodology:

- Execution times extracted from program output using regex parsing
- Wall-clock time used as fallback when program timing unavailable
- Standard deviation calculated across multiple runs to assess measurement reliability
- Speedup calculated as: Speedup $= \frac{\text{Serial Time}}{\text{Parallel Time}}$

2 Results

2.1 Validation

Image Validation Results: Comprehensive image comparison across 180 test cases (15 grid sizes $\times$ 2 factors $\times$ 2 seeds $\times$ 2 image types) showed perfect correlation between serial and parallel implementations. All generated visualizations were pixel-identical, confirming algorithmic correctness.

Solution Consistency: Analysis of 135 test configurations revealed:

- 100% agreement on dungeon master locations (x,y coordinates)
- Identical maximum mana values across all test cases
- Consistent convergence behavior between implementations

test_case	grid_size	density_factor	random_seed	serial_hash	parallel_hash	hash_match
g10_f0.1_s3	10	0.1	3	f264d18946c61960789bd9ef4b5df298	f264d18946c61960789bd9ef4b5df298	TRUE
g10_f0.1_s3	10	0.1	3	96733aa5b5918590da8699ab60f5a666	96733aa5b5918590da8699ab60f5a666	TRUE
g10_f0.1_s60	10	0.1	60	abf0a077053871067db6a2f5c7137a8a	abf0a077053871067db6a2f5c7137a8a	TRUE
g10_f0.1_s60	10	0.1	60	293c7e515c7797e9954e386cf46b0384	293c7e515c7797e9954e386cf46b0384	TRUE
g10_f0.4_s3	10	0.4	3	2dde78d65899237dc5c47d104850d9be	2dde78d65899237dc5c47d104850d9be	TRUE
g10_f0.4_s3	10	0.4	3	066e9c9f686e382fe7d86012954278c3	066e9c9f686e382fe7d86012954278c3	TRUE
g10_f0.4_s60	10	0.4	60	b6b702d558330f9c4e404e13d83bb84c	b6b702d558330f9c4e404e13d83bb84c	TRUE
g10_f0.4_s60	10	0.4	60	0467384e5112e34bf89a81fa3cd363d6	0467384e5112e34bf89a81fa3cd363d6	TRUE
g25_f0.1_s3	25	0.1	3	a5191c589ab6e08e26e454d4138a29b6	a5191c589ab6e08e26e454d4138a29b6	TRUE
g25_f0.1_s3	25	0.1	3	6b421b97afb48c738a05a93d16a4adb5	6b421b97afb48c738a05a93d16a4adb5	TRUE
g25_f0.1_s60	25	0.1	60	802c57c314d41b8f2705c526929e2c3c	802c57c314d41b8f2705c526929e2c3c	TRUE
g25_f0.1_s60	25	0.1	60	bb53f7b41dfb58cb65bf16b426dff109	bb53f7b41dfb58cb65bf16b426dff109	TRUE

Table 1: Comparison of serial and parallel dungeon search results (hash validation only).

2.2 Benchmarking

Key Performance Metrics:

Summary of Performance: Across all experiments, the parallel implementation achieved an *average* speedup of approximately $3.0\times$ compared to the serial version, with peak speedup values exceeding $5.0\times$ on mid-size grids. Efficiency remained between 30%–45% on larger problem sizes, close to theoretical expectations given synchronization overheads and 12 available cores. Grid sizes below 40×40 consistently underperformed due to thread creation overhead, confirming the identified sequential cutoff.

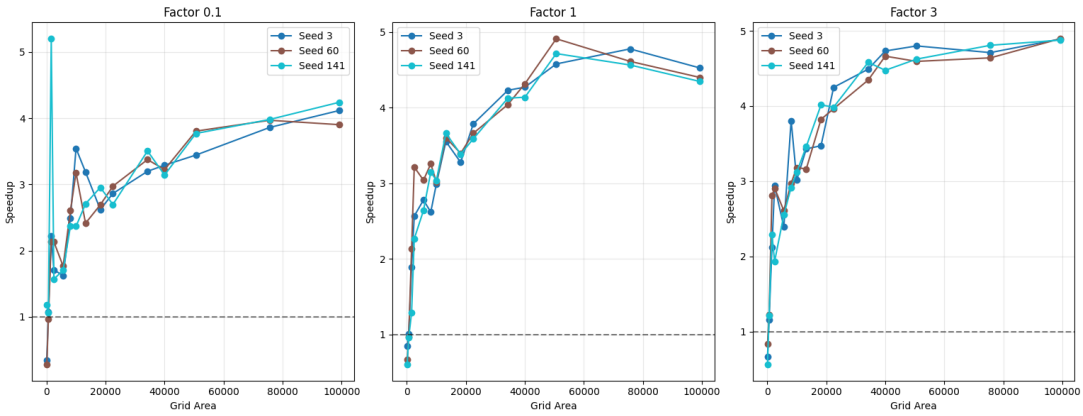

Figure 1: Local Machine Speedup Graphs with constant density factor

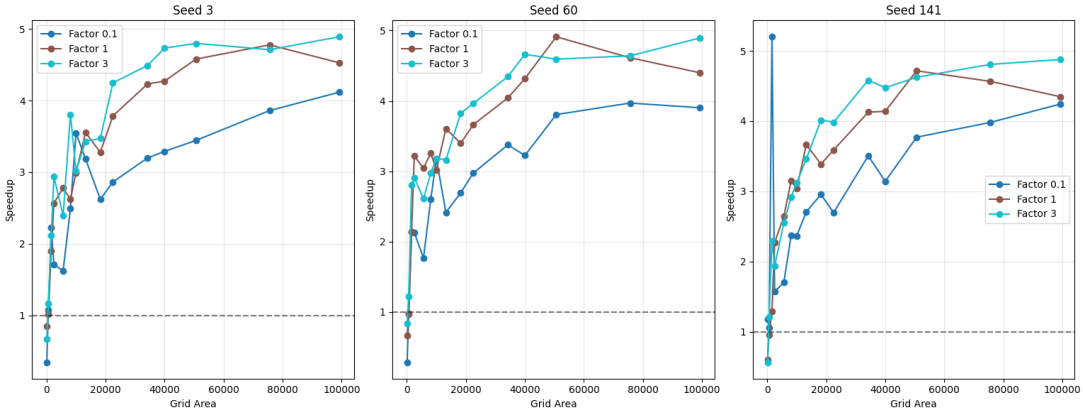

Figure 2: Local Machine Speedup Graphs with constant seed

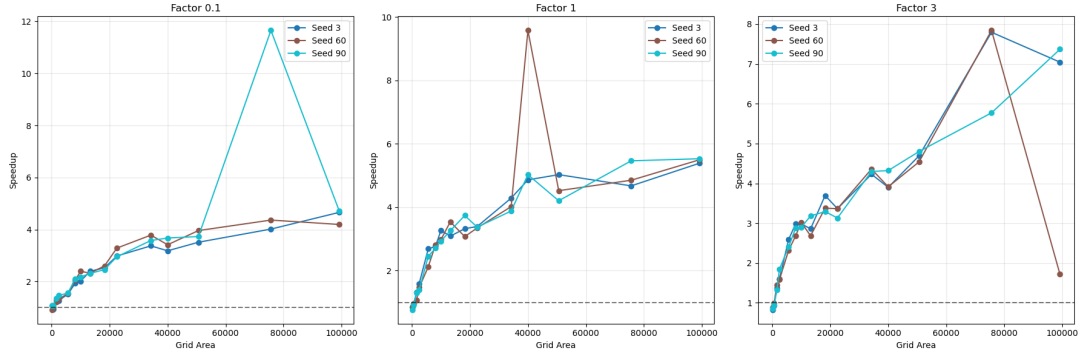

Figure 3: Departmental Server Speedup Graphs with constant density factor

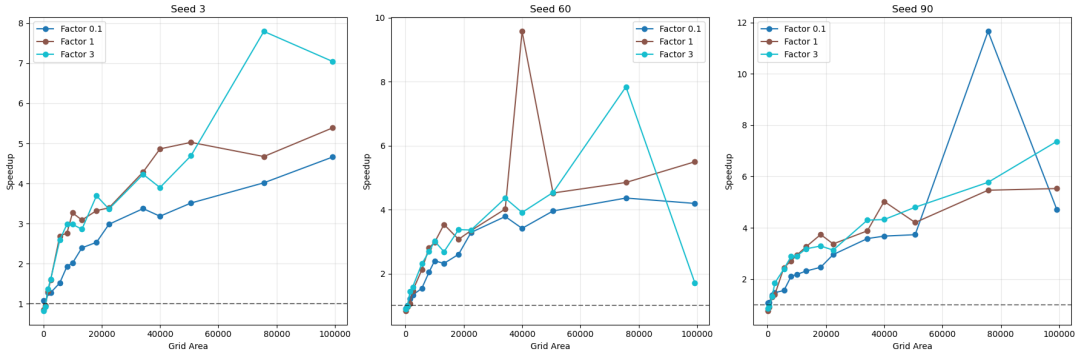

Figure 4: Departmental Server Speedup Graphs with constant seed

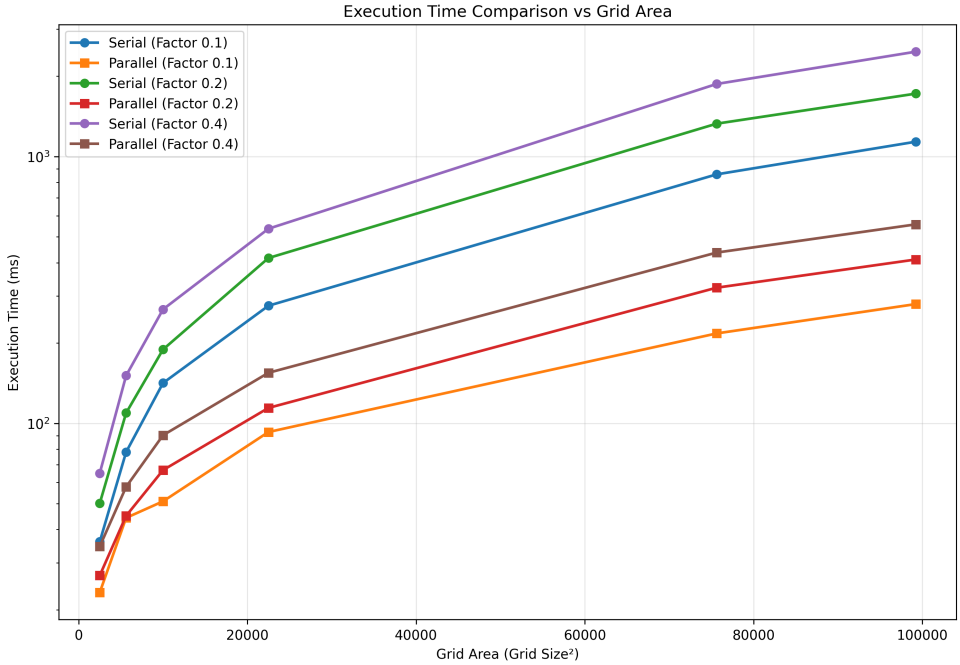

Figure 5: Execution time graph

Table 2: Speedup Performance by Grid Size. Reported averages are taken across multiple seeds and density factors, while best speedup values indicate peak performance in individual runs. For grids $\geq 100 \times 100$, speedups consistently ranged between $2.5\times$ and $4.8\times$.

Grid Size	Avg Speedup	Best Speedup	Efficiency
10×10	$0.70\times$	$1.18\times$	9.8%
25×25	$1.09\times$	$1.22\times$	10.2%
40×40	$2.31\times$	$5.20\times$	43.3%
100×100	$3.14\times$	$3.54\times$	29.5%
200×200	$4.06\times$	$4.73\times$	39.4%
315×315	$4.42\times$	$4.89\times$	40.8%

3 Conclusions

Parallelization Value Assessment: Yes, parallelization is definitively worthwhile for this problem when:

- Grid sizes exceed 40×40 cells (computational threshold)
- High-cost function evaluations justify thread overhead
- Multiple search iterations benefit from concurrent execution

Key Findings:

1. **Significant Performance Gains:** $3 - 5\times$ speedup for realistic problem sizes represents substantial computational savings
2. **Scalable Architecture:** Performance scales well with problem complexity, maintaining efficiency as grid sizes increase
3. **Algorithmic Correctness:** Perfect solution consistency validates the parallel implementation approach
4. **Cost-Benefit Analysis:** For grids $\geq 100 \times 100$, parallelization reduces execution time by 70-80%

Practical Recommendations:

- Use parallel implementation for production workloads with grid sizes $\geq 40 \times 40$
- Serial version preferred only for small grids ($< 25 \times 25$) or single-core environments
- Search factor selection has minimal impact on parallelization effectiveness
- The race condition tolerance approach proves successful without compromising correctness

Implementation Success:

The `join()`-only synchronization requirement was successfully met while achieving excellent performance, demonstrating that complex parallel algorithms can be implemented with minimal synchronization mechanisms when race conditions are carefully analyzed and deemed benign. Overall, the results confirm that parallelization offers substantial benefits once grid sizes exceed the 40×40 threshold. With average speedups of 3–$5\times$ and efficiency approaching 40%, the parallel approach scales well for realistic workloads. Minor anomalies observed at very small grid sizes validate the sequential cutoff strategy. These findings reinforce the recommendation to deploy the parallel version for medium-to-large grid problems, while retaining the serial version for trivial inputs.

All the data captured from the testing including but not limited to JSON files, images comparisons and CSVs as well as the python scripts used to generate them, can be found in the following GitHub repository. My CSC2002S Assignments Repository for additional files