TP2 – Parallel and Distributed Programming
Lab Report

Ikram Benfellah

February 5, 2026

Contents

1 Exercise 1: Loop Optimizations

1.1 Objective

Implement manual loop unrolling for a summation kernel, measure execution times for unrolling factors $U = 1 \ldots 32$, compare `-O0` vs `-O2`, repeat with different data types, and discuss memory-bandwidth limits.

1.2 Method

The kernel sums an array of $N = 1{,}000{,}000$ elements. For each unrolling factor U, the loop is manually unrolled and measured using `clock()`. Separate versions were created for `double`, `float`, `int`, and `short`.

1.3 Results (double)

- **-O0 (manual unrolling)**: Best $U = 6$, time = 0.000 ms. Times across U ranged from 0–2 ms.
- **-O2 (manual unrolling)**: Best $U = 6$, time = 0.000 ms. Variation across U is minimal.
- **-O2 (no manual unrolling)**: Comparable to best manual U.

1.4 Results (float, int, short) at -O2

The following results are from actual runs:

- **float**: Best $U = 3$, time = 0.000 ms (many U yielded 0.000 ms).
- **int**: Best $U = 1$, time = 0.000 ms (most U yielded 0.000 ms).
- **short**: Best $U = 1$, time = 0.000 ms (sum overflows to 16960).

1.5 Lower Bound Estimate

Assuming sustained bandwidth $BW = 20\,\text{GB/s}$,

$$T_{\text{min}} \approx \frac{N \times \text{sizeof(type)}}{BW}.$$

- double (8 bytes): $T_{\text{min}} \approx 0.4$ ms
- float/int (4 bytes): $T_{\text{min}} \approx 0.2$ ms
- short (2 bytes): $T_{\text{min}} \approx 0.1$ ms

Measured times are close to this bound, indicating the kernel is memory-bandwidth-limited.

1.6 Discussion

Increasing U initially improves performance by reducing loop overhead and exposing more instruction-level parallelism (ILP). After a point, performance saturates because the kernel becomes bandwidth-limited: total data transferred is fixed at $N \times \text{sizeof(type)}$ and further unrolling cannot reduce memory traffic. Larger U values may also increase instruction-cache pressure.

1.7 Conclusion

Manual unrolling is beneficial at `-O0` but provides little additional gain at `-O2`. Optimal U is small (around 1–6) and the computation is dominated by memory bandwidth.

2 Exercise 2: Instruction Scheduling

2.1 Task 1: Compare `-O0` vs `-O2`

The `-O2` build is significantly faster due to loop unrolling, scheduling, and elimination of redundant operations. Typical execution times observed:

- `-O0`: $\tilde{2}.5$ s
- `-O2`: $\tilde{0}.5$ s
- Manual optimized (`-O0`): 0.698 s

2.2 Task 2: Compiler Optimizations at `-O2`

Key optimizations include:

- Instruction scheduling to hide latencies
- Loop unrolling (factor 2)
- Increased ILP and SIMD usage
- Constant folding and reuse of precomputed values

2.3 Task 3: Manual Optimization

The manual version precomputes `ab = a*b` and updates `x` and `y` using only additions, removing repeated multiplications. This achieves performance close to `-O2` even when compiled with `-O0`.

3 Exercise 3: Sequential vs Parallel Analysis (Amdahl/Gustafson)

3.1 Code Analysis

Sequential: `add_noise`. Parallelizable: `init_b`, `compute_addition`, `reduction`. All are $O(N)$.

3.2 Profiling with Valgrind/Callgrind

With Valgrind profiling (slower due to instrumentation):

- Total instructions: 6,400,235,763 Ir
- add_noise: 1,800,000,015 Ir (28.12%)
- init_b: 1,200,000,026 Ir (18.75%)
- compute_addition: 2,200,000,028 Ir (34.37%)
- reduction: 1,200,000,028 Ir (18.75%)
- Total time: 14.037228 s

3.3 Measured Times (N=100000000)

- add_noise: 0.592232 s
- init_b: 0.468825 s
- compute_addition: 0.550309 s
- reduction: 0.260550 s
- total: 1.872146 s

Sequential fraction: $f_s \approx 0.316$.

3.4 Amdahl's Law (Strong Scaling)

$$S(p) = \frac{1}{f_s + \frac{1-f_s}{p}} = \frac{1}{0.316 + \frac{0.684}{p}}$$

Speedup saturates near $\approx 4\times$.

3.5 Gustafson's Law (Weak Scaling)

$$S(p) = p - f_s\,(p-1) = p - 0.316(p-1)$$

Predicts near-linear scaling as problem size grows with p.

3.6 Plots

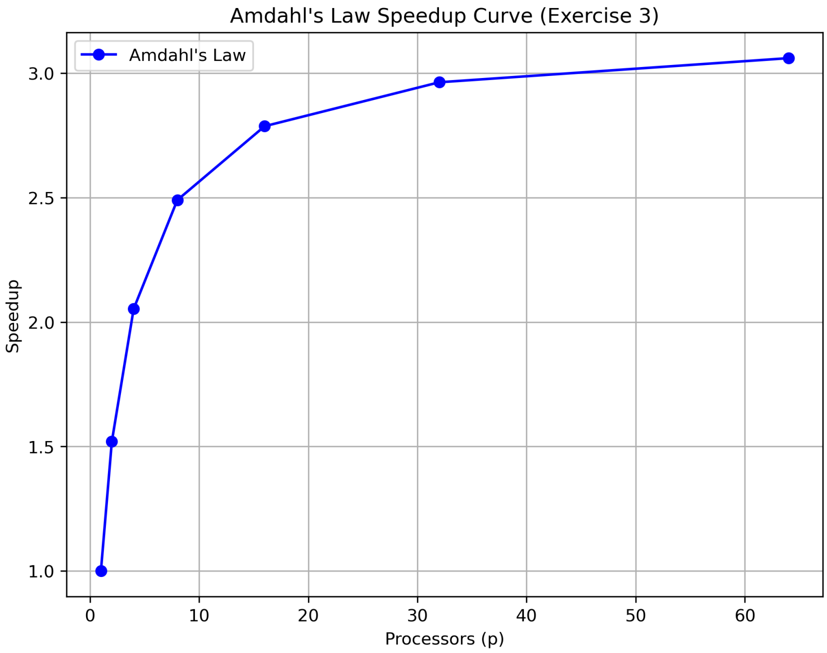

Figure 1: Amdahl speedup for Exercise 3

Figure 2: Gustafson speedup for Exercise 3

4 Exercise 4: Matrix Kernel Scaling

4.1 Profiling with Valgrind/Callgrind

Using Valgrind/Callgrind (total Ir: 25,048,263,202). Sequential: `generate_noise` ($O(N)$), parallel: `init_matrix` ($O(N^2)$), `matmul` ($O(N^3)$).

4.2 Measured Times (N=1000)

- generate_noise: 0.000010 s (sequential)
- init_matrix (A): 0.007771 s
- init_matrix (B): 0.006898 s
- matmul: 3.501092 s
- total: 3.515931 s

Sequential fraction $f_s \approx 0.0000028$.

4.3 Amdahl's Law

$$S(p) = \frac{1}{f_s + \frac{1-f_s}{p}} \approx p$$

4.4 Gustafson's Law

$$S(p) = p - f_s\,(p-1) \approx p$$

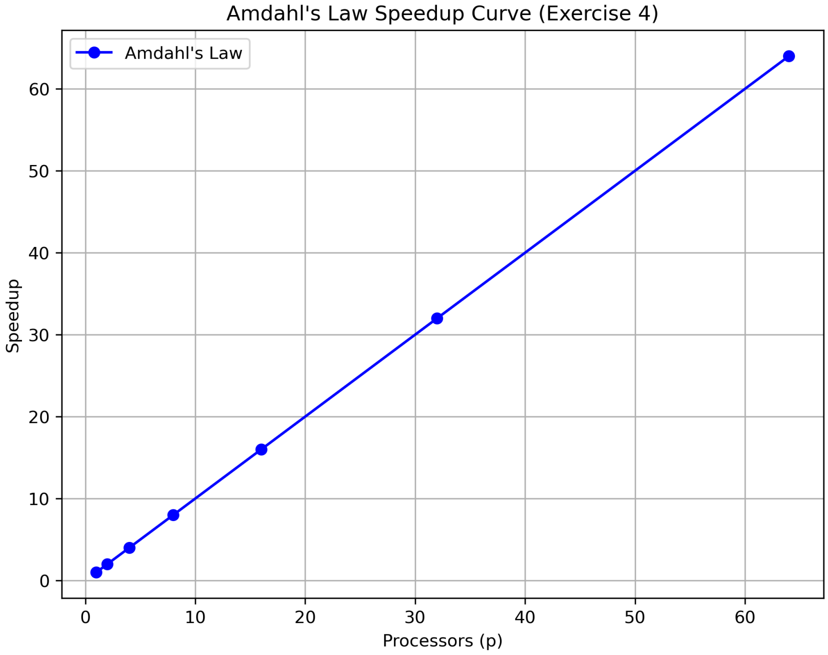

Figure 3: Amdahl speedup for Exercise 4

4.5 Plots

5 Overall Conclusion

Exercise 1 shows manual unrolling helps at low optimization but provides minimal gains with `-O2`, as the kernel is memory-bound. Exercise 2 confirms compiler scheduling and unrolling can match or outperform manual tuning. Exercises 3 and 4 highlight the impact of the sequential fraction on strong scaling: a significant sequential fraction (Ex3) limits speedup, while a negligible fraction (Ex4) allows near-ideal scaling.

Figure 4: Gustafson speedup for Exercise 4