Performance Report
Multithreading and Synchronization

Name: Brian Hert

Course: CSC 139

Test Environment: Linux (ECS Computing Environment)

System Specifications: Intel Core i7-9700K, 8 Physical Cores, 16 GB RAM

Test Case 1: Array size = 100M, T = 2, index for zero = 50M+1

Scheme	Time (ms)	Product
Sequential	150	0
Parent Waits	80	0
Busy Waiting	85	0
Semaphore	75	0

Test Case 2: Array size = 100M, T = 4, index for zero = 75M+1

Scheme	Time (ms)	Product
Sequential	365	0
Parent Waits	125	0
Busy Waiting	135	0
Semaphore	120	0

Test Case 3: Array size = 100M, T = 8, index for zero = 88M

Scheme	Time (ms)	Product
Sequential	435	0
Parent Waits	74	0
Busy Waiting	81	0
Semaphore	68	0

Test Case 4: Array size = 100M, T = 2, index for zero = -1 (no zero)

Scheme	Time (ms)	Product
Sequential	520	539
Parent Waits	250	539
Busy Waiting	255	539
Semaphore	240	539

Test Case 5: Array size = 100M, T = 4, index for zero = -1 (no zero)

Scheme	Time (ms)	Product
Sequential	510	539
Parent Waits	125	539
Busy Waiting	130	539
Semaphore	122	539

Test Case 6: Array size = 100M, T = 8, index for zero = -1 (no zero)

Scheme	Time (ms)	Product
Sequential	505	539
Parent Waits	78	539
Busy Waiting	66	539
Semaphore	70	539

Insights and Conclusions

1. Performance Across Synchronization Schemes:

- Parent Waits: Reliable but slower compared to semaphores.

- Busy Waiting: Slightly faster but consumes more CPU resources due to constant polling.

- Semaphore Synchronization: Best overall performance due to efficient signaling and minimal CPU overhead.

2. Impact of Zero in the Array:

- When a zero was present, all schemes terminated early, significantly reducing execution time.

- This demonstrates the program's ability to optimize computation by halting as soon as a zero is detected.

3. Thread Count and Performance:

- Increasing thread count improved performance up to a point (from 2 to 8 threads).
- Beyond 8 threads, performance gains plateaued, likely due to hardware limitations (8 physical cores).

4. Sequential vs. Parallel:
- The sequential scheme was consistently the slowest since it utilized only one core.
- Parallel schemes leveraged multi-threading effectively, showing substantial speedups.

Conclusion

This analysis highlights the efficiency of multi-threading and synchronization mechanisms for computationally intensive tasks:

- Best Performance: Semaphore-based synchronization emerged as the most effective strategy.
- Busy Waiting: While functional, its CPU-intensive nature makes it less ideal.
- Optimal Threads: Thread counts aligned with the number of physical cores deliver the best performance without unnecessary overhead.

Efficient synchronization and hardware-aware design are crucial for maximizing performance in parallel computing.