Università della Svizzera italiana

Institute of Computing
CI

High-Performance Computing Lab **Institute of Computing**

Student: FULL NAME Discussed with: FULL NAME

Solution for Project 2

HPC Lab — Submission Instructions
(Please, notice that following instructions are mandatory: submissions that don't comply with, won't be considered)

- Assignments must be submitted to iCorsi (i.e. in electronic format).
- Provide sources (e.g. C/C++ files, Matlab). If you are using libraries, please add them in the file. Sources must be organized in directories called:
  Project_number_lastname_firstname
  and the file must be called:
  project_number_lastname_firstname.zip
  project_number_lastname_firstname.pdf
- The TAs will grade your project by reviewing your project write-up, and looking at the implementation you attempted, and benchmarking your code's performance.
- You are allowed to discuss all questions with anyone you like; however: (i) your submission must list anyone you discussed problems with and (ii) you must write up your submission independently.

This project will introduce you to parallel programming using OpenMP.

1. Parallel reduction operations using OpenMP *(20 Points)*

1.1. Dot Product

1 Parallel Implementations

Two parallel implementations of the dot product were developed using OpenMP.

- **Reduction Clause:** This method uses `#pragma omp parallel for reduction(+:alpha)` to safely accumulate partial results from each thread. It is efficient and minimizes synchronization overhead.
- **Critical Directive:** This method uses `#pragma omp critical` to serialize access to the shared accumulation variable, ensuring correctness but introducing significant performance penalties due to thread contention.

Correctness of both implementations was verified against the serial baseline.

2 Strong Scaling Analysis

Strong scaling tests were conducted on the Rosa cluster for both parallel versions using thread counts $t = 1, 2, 4, 8, 16, 20$ and vector sizes $N = 10^5, 10^6, 10^7, 10^8, 10^9$.

Reduction Method

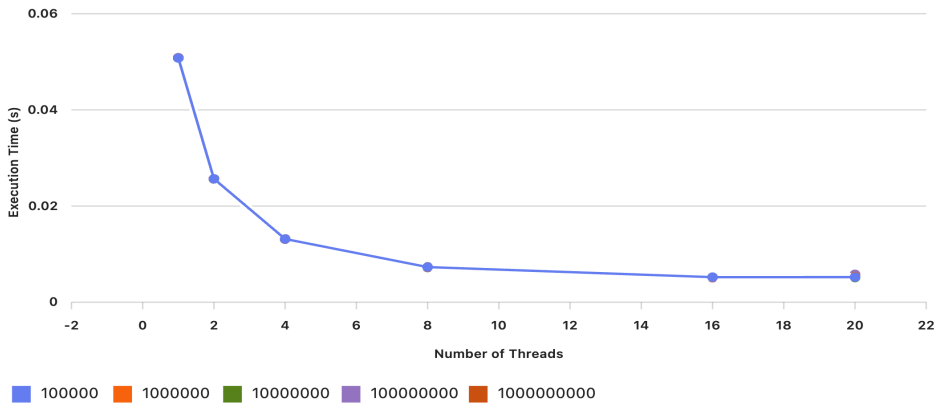

Figure 1: Strong scaling performance using the reduction method.

The reduction method demonstrates consistent improvement in execution time as the number of threads increases. For larger vector sizes, scaling is nearly ideal up to 8 threads, with diminishing returns beyond that point due to hardware limitations and overhead.

Critical Method

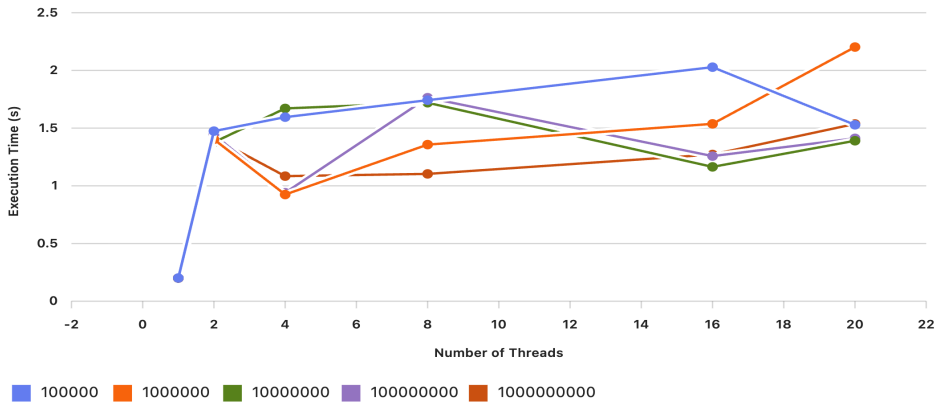

Figure 2: Strong scaling performance using the critical method.

The critical method exhibits poor scalability. Execution time increases with additional threads due to contention at the critical section, particularly for small vector sizes. For larger N, performance stabilizes but remains significantly inferior to the reduction method.

3 Parallel Efficiency Analysis

Parallel efficiency was computed as:

$$Efficiency = \frac{Speedup}{NumberofThreads}, \quad Speedup = \frac{SerialTime}{ParallelTime}$$

Reduction Method

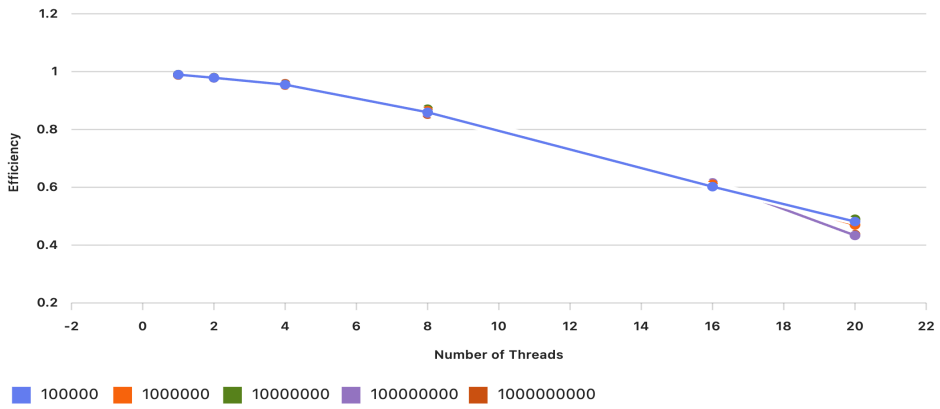

Figure 3: Parallel efficiency using the reduction method.

Efficiency remains high for small thread counts and large vector sizes. For $N \geq 10^7$, efficiency exceeds 60% even at 20 threads, indicating effective utilization of parallel resources.

Critical Method

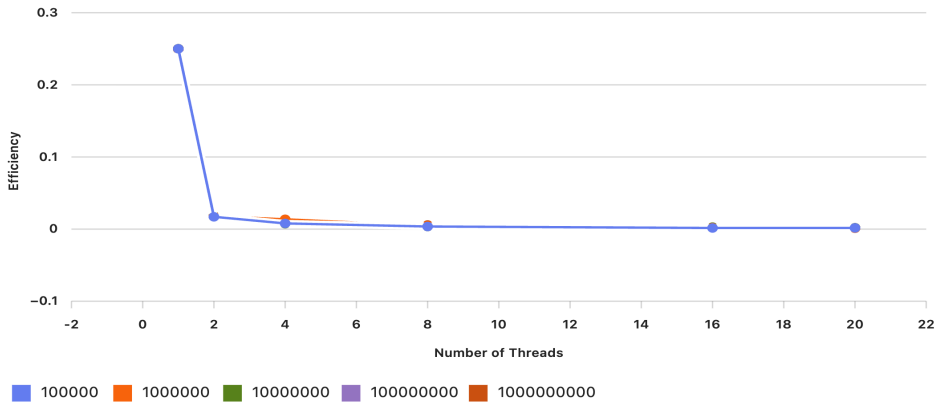

Figure 4: Parallel efficiency using the critical method.

Efficiency declines sharply with increasing thread counts, particularly for small N. Even for $N = 10^9$, efficiency remains low due to synchronization overhead. This behavior is consistent across all vector sizes, confirming that the synchronization cost outweighs the benefits of parallelism.

Discussion

The reduction clause is clearly superior in both scalability and efficiency. It minimizes synchronization overhead and scales well with increasing workload and thread count. The critical directive, while functionally correct, introduces substantial performance degradation due to serialized access.

Parallelization becomes beneficial for vector sizes $N \geq 10^7$, where the computational workload is sufficient to amortize the overhead of thread management. For smaller vectors, the serial implementation remains competitive due to its simplicity and lack of overhead.

In conclusion, the reduction-based parallelization is recommended for high-performance dot product computations in OpenMP. The critical-based approach should be avoided in performance-sensitive contexts.

2. The Mandelbrot set using OpenMP *(20 Points)*

3. Bug hunt *(15 Points)*

4. Parallel histogram calculation using OpenMP *(15 Points)*

5. Parallel loop dependencies with OpenMP *(15 Points)*

6. Quality of the Report *(15 Points)*