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

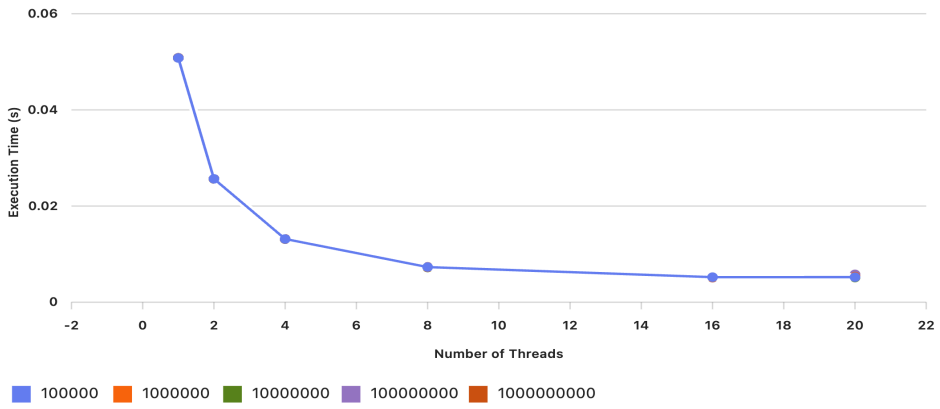

Figure 1: Strong scaling performance using the reduction method.

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

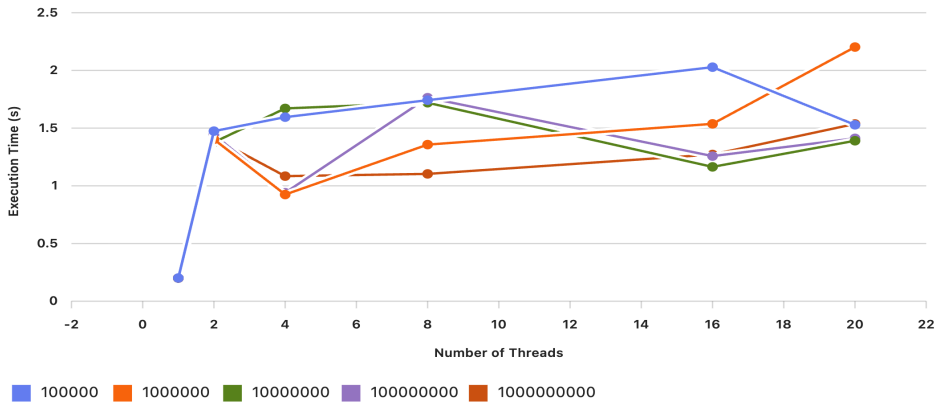

Figure 2: Strong scaling performance using the critical method.

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

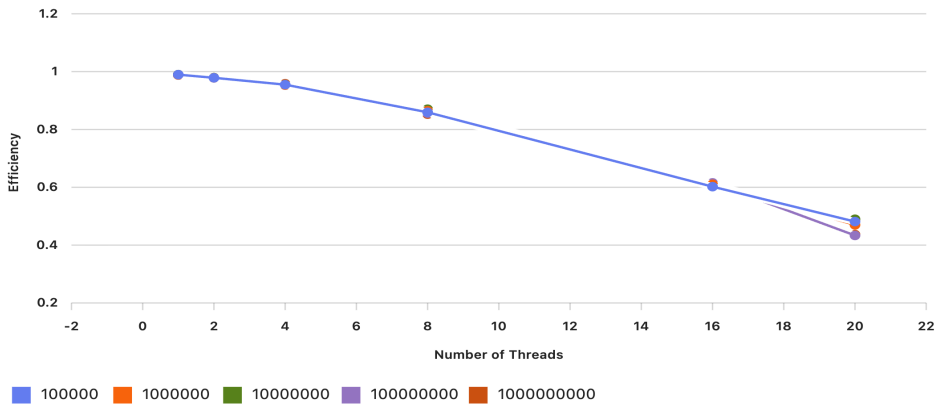

Figure 3: Parallel efficiency using the reduction method.

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

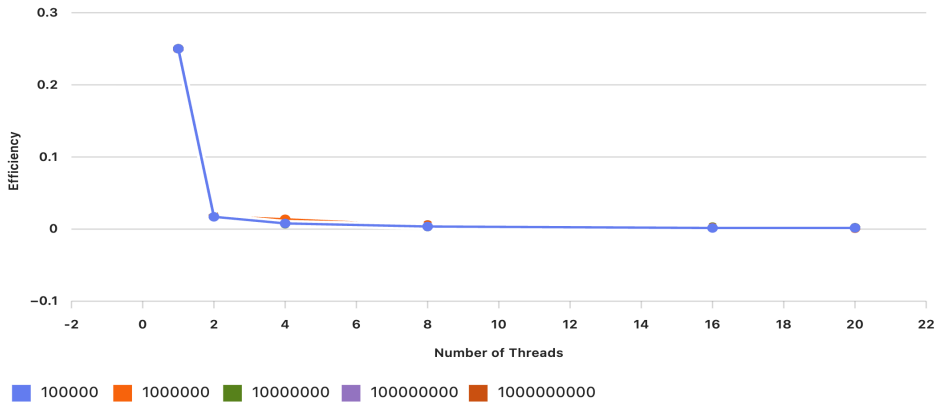

Figure 4: Parallel efficiency using the critical method.

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

OpenMP Overhead

The critical directive introduces significant overhead due to serialized access to the shared variable. In contrast, the reduction clause avoids this issue by efficiently combining partial results.

Thread Count vs. Workload

For small vector sizes ($N \leq 10^6$), parallelization offers limited benefit, as overhead outweighs performance gains. For larger sizes ($N \geq 10^7$), multi-threading becomes advantageous, particularly when using the reduction method.

Conclusion

The reduction method is preferred for parallel dot product computation. It demonstrates good scalability and maintains high efficiency for large workloads. The critical method should be avoided in performance-sensitive applications.

1.2 Approximating π

The value of π was approximated using the midpoint rule applied to the integral:

$$\pi = \int_0^1 \frac{4}{1+x^2}\, dx$$

A fixed number of subintervals, $N = 10^{10}$, was used to ensure a computationally intensive workload suitable for parallel analysis. A serial version was implemented using a standard loop, followed by a parallel version using OpenMP with the `#pragma omp parallel for reduction(+:sum)` directive. This approach was selected for its simplicity and efficiency, as it avoids locking overhead and ensures safe accumulation of partial sums across threads.

Execution times were recorded for both serial and parallel versions across thread counts $t = 1, 2, 4, 8$. Table **??** summarizes the results, including calculated speedup and parallel efficiency:

Table 1: Performance results for π approximation with $N = 10^{10}$

Threads	Serial Time (s)	Parallel Time (s)	Speedup	Efficiency
1	53.93037	53.93042	1.0000	1.0000
2	53.93950	26.97084	1.9999	1.0000
4	53.93071	13.48577	3.9991	0.9998
8	53.93037	6.743196	7.9977	0.9997

Speedup and Efficiency Analysis

Speedup quantifies how much faster the parallel version is compared to the serial one:

$$Speedup = \frac{SerialTime}{ParallelTime} = \frac{53.93037}{6.743196} \approx 7.9977$$

Using 8 threads made the program almost 8 times faster. Parallel efficiency measures how well the threads are utilized:

$$Efficiency = \frac{Speedup}{NumberofThreads} = \frac{7.9977}{8} \approx 0.9997$$

This corresponds to approximately 99.97% efficiency, which is excellent and indicates minimal overhead.

Discussion

The results demonstrate near-perfect linear scaling up to 8 threads, confirming strong scaling behavior where the problem size remains constant while the number of processing units increases. The OpenMP `reduction` clause provided an efficient and scalable solution for parallelization. The task is highly parallelizable and benefits significantly from multi-threading.

Figure **??** illustrates the speedup and efficiency trends on a log-scaled axis:

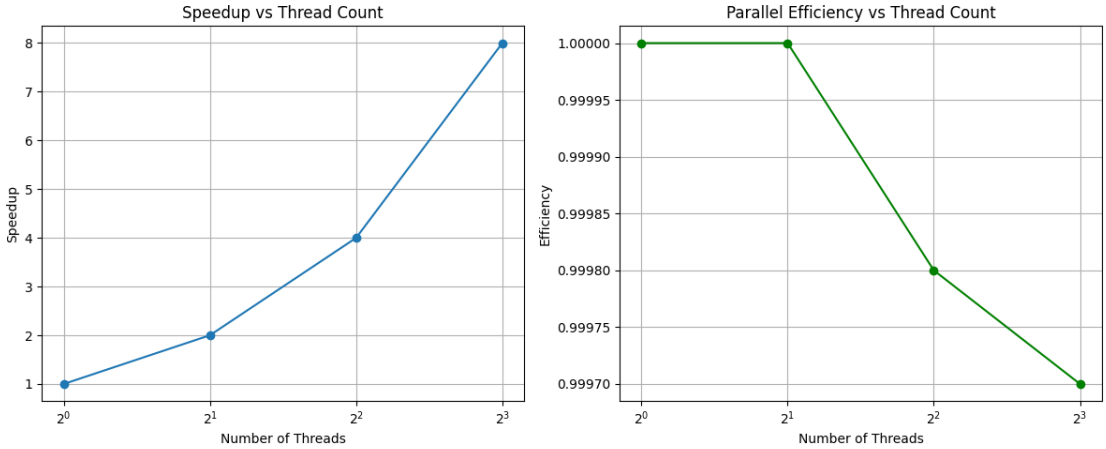

Figure 5: Speedup and Parallel Efficiency for π Approximation

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)*