HPC Project Report 2023

Michele Montesi
Matricola: 0000974934
E-Mail: michele.montesi3@studio.unibo.it

30 settembre 2023

Introduction

This research presents the development of two parallelized versions of the `sph.c` software, implemented using the *OpenMP* and *MPI* libraries, respectively. The objective of the research is to evaluate the advantages of using multiprocessor programming.

To evaluate the software's performance, a Python script has been created that runs the program with an increasing number of threads for *OpenMP* and processes for *MPI*, starting from 1 and reaching 12, and with a varying number of particles from 1400 to 5700. The script records relevant data in an Excel spreadsheet, from which the average will be calculated. This information will then be used to evaluate the performance.

The software has been tested on the `ISI-Raptor` server.

OpenMP Version

Implementation

In this implementation of the software, done with *OpenMP*, sections of code that require significant computation have been parallelized using the `#pragma omp parallel for` directive.

Where necessary, reductions have been made to optimize the performance of the parallelized software.

Performance

From Figure 1, it can be observed that as the number of threads increases, the speedup significantly improves, as indicated by the increasing curve. However, after reaching 8 threads, a saturation phase can be observed, where adding more threads no longer leads to a significant performance improvement. At this point, the use of additional threads may even degrade performance.

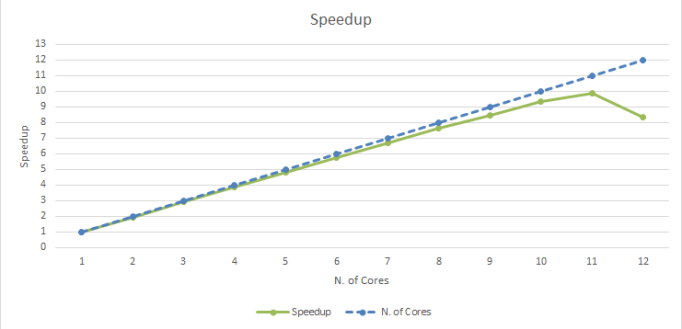

Figura 1: Speedup of the OpenMP implementation

From the graph in Figure 2, it can be noted that the Strong Scaling Efficiency reaches its maximum peak at around 6 threads, corresponding to an efficiency of about 90

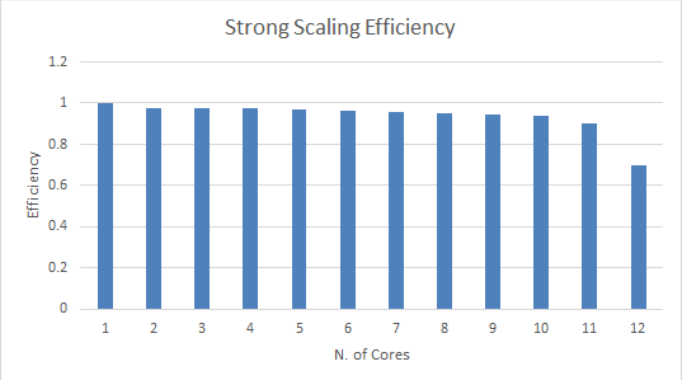

Figura 2: Strong scaling efficiency of the OpenMP implementation

In Figure 3, it can be observed that the Weak Scaling Efficiency remains constant as the problem size increases. This indicates that the software efficiently handles problems of varying sizes by distributing the workload among the available CPUs without a significant decrease in performance. This constancy is an indicator of good software scalability. At 12 threads, weak efficiency decreases, indicating that the addition of further threads does not lead to better performance.

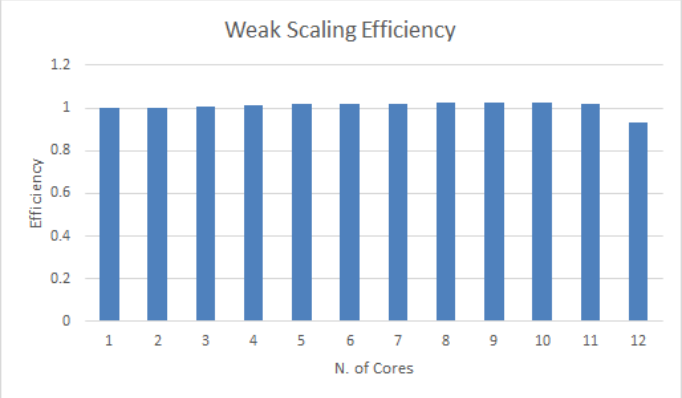

Figura 3: Weak scaling efficiency of the OpenMP implementation

MPI Version

Implementation

For the implementation of the software with MPI, a contiguous MPI_Datatype has been created to encapsulate the data structure during message exchange. To distribute particles among all processors, an MPI_Scatterv is performed from process 0. After compute_density_pressure and integrate, an MPI_Allgatherv is called to collect the processed data and redistribute it to all processors again, while compute_forces is not included as it only acts locally. Finally, after calculating the average velocity of the local particles, an MPI_Reduce with a summation operator is executed. This result is divided by all processors, obtaining the overall average velocity.

Performance

From Figure 4, it can be observed that the software's performance increases almost linearly as the number of processing units used increases. However, the performance gain in terms of speedup decreases as new processing units are added due to factors such as increased communication time between them, the need for synchronization between processes, and the presence of parts of the software that cannot be parallelized. Moreover, it can be noticed that speedup starts to saturate around 8 cores.

Figura 4: Speedup of the MPI implementation

From the graph in Figure 5, it is evident that the Strong Scaling Efficiency reaches its peak at around 6 cores, achieving an efficiency of approximately 90

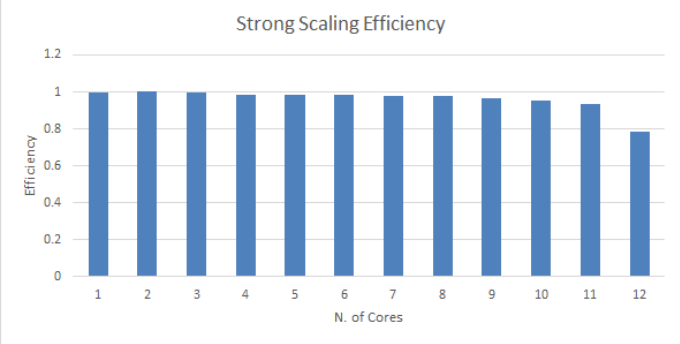

Figura 5: Strong scaling efficiency of the MPI implementation

The graph of Weak Scaling Efficiency (Figure 6) shows that the efficiency of the software remains constant as the problem size varies. This suggests that the software efficiently handles problems of varying sizes by distributing the workload among the available CPUs without a significant decrease in performance. This is an indicator of good software scalability. However, the addition of additional cores beyond 12 results in decreased weak efficiency, indicating that the use of additional resources does not translate into better performance.

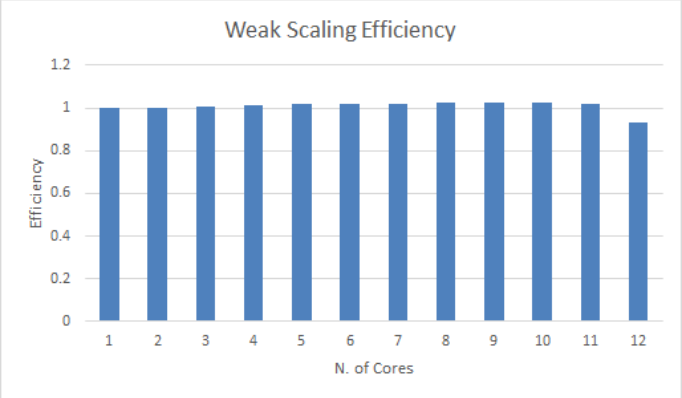

Figura 6: Weak scaling efficiency of the MPI implementation

Conclusions

The implementation of the software with OpenMP has led to significant performance improvements, as indicated by the speedup graph as a function of the number of threads. It was observed that increasing the number of threads significantly improved performance up to a certain point, after which the positive effect saturated, indicating that using additional threads could even degrade performance. Strong scaling efficiency was rather high for cases with a low number of threads but decreased as the number of threads increased. Weak scaling efficiency was good and consistent as the number of processors varied.

The implementation with MPI resulted in a significant reduction in execution times compared to the sequential version of the software, as indicated by the speedup graph as a function of the number of processors. It was observed that strong scaling efficiency was very high for all cases analyzed, indicating good scalability of the software. Weak scaling efficiency was good and consistent as the number of processors varied.

In conclusion, the implementation of the software with OpenMP and MPI led to significant performance improvements compared to the sequential version. The OpenMP approach is particularly effective for problems of limited size and with a small number of threads, while the MPI approach is more efficient for larger problems and with a high number of processors.