Scientific Programming

Assignment: OpenMP

Anand Kamble
7th December 2023

---

1 Introduction

This assignment involves the utilization of OpenMP to parallelize both the initial homework and the provided code. OpenMP, a widely used parallel programming model, streamlines the development of parallel applications by offering directives and library routines. By distributing computational work across multiple threads, we seek to enhance program performance and efficiency on multi-core architectures.

---

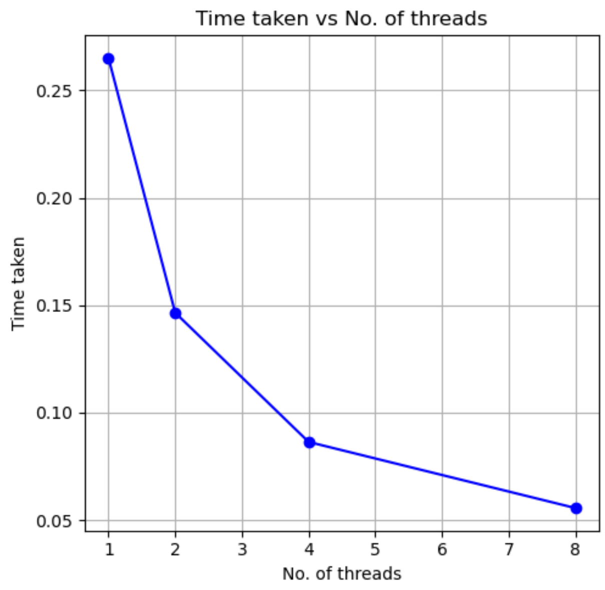
2 Problem 1

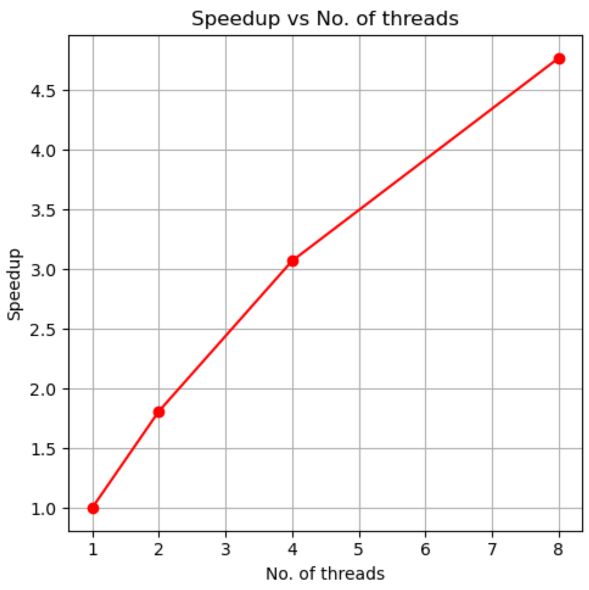
In this section, we focus on parallelizing the ‘simpsons_rule‘ and ‘trapezoidal_rule‘ functions from the initial homework. The targeted optimization involves parallelizing the for loop, responsible for cumulative additions to the variable ’result’. To achieve this parallelization, the OpenMP framework is employed, utilizing the reduction clause.[1] Specifically, the directive `\#pragma omp parallel for reduction(+ : result)` is implemented to distribute the workload efficiently among multiple threads and manage the summation operation concurrently.

To validate the our parallelization approach, we conducted timing experiments on the loops for varying numbers of threads. The table below presents the corresponding time taken for each thread configuration, along with the calculated speedup relative to the execution with a single thread.

Chosen value of N = 1,000,000,000

No. of Threads	Time Taken (s)	Speedup
1	6.094589	1.000000
2	3.135672	1.943631
4	1.686455	3.613848
8	1.000021	6.094471





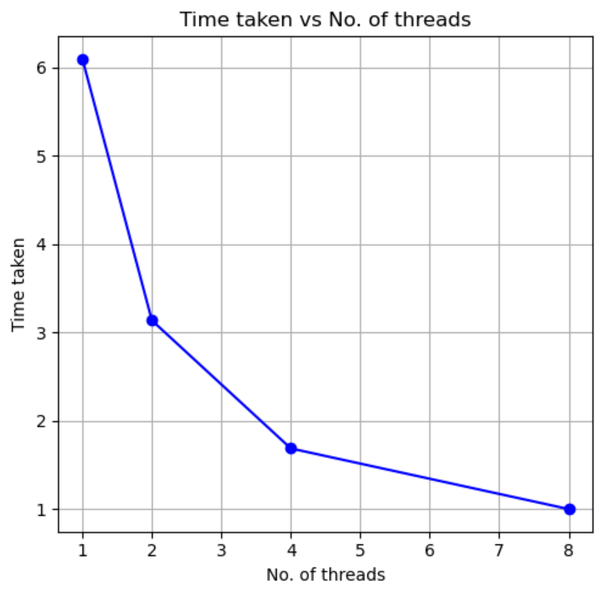
---

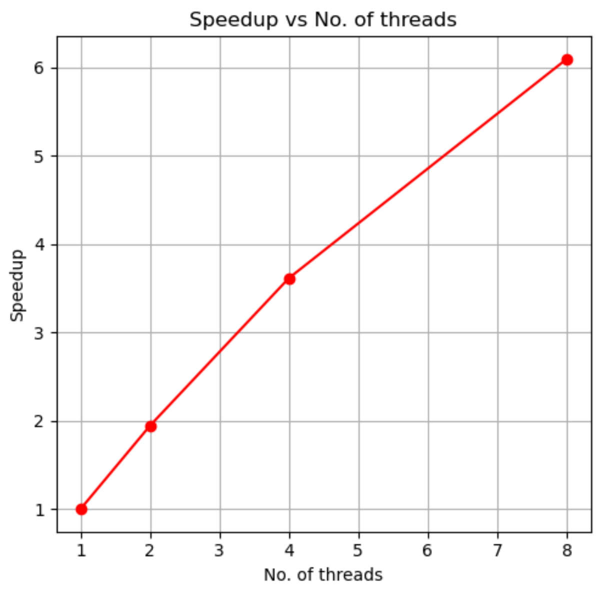
3 Problem 2

In this section, we focus on parallelizing the provided code, which involves the addition of values to elements of an array. Notably, the variable subject to reduction is an array, necessitating the specification of the array length in the OpenMP clause as follows: `#pragma omp parallel for private(i) shared(random_numbers) reduction(+ : num[ : 256])`.[2] This directive ensures that the parallelization process efficiently distributes the workload among multiple threads while maintaining the integrity of the array reduction operation.

After conducting tests with 1, 2, 4, and 8 threads, the following results were obtained:

No. of Threads	**Time Taken (s)**	**Speedup**
1	0.265110	1.000000
2	0.146537	1.809186
4	0.086372	3.069453
8	0.055592	4.769009





---

4 Execution

The code is organized into two folders, each corresponding to a specific problem. Inside each folder, you will find a make file responsible for compiling the program into an executable binary file named `a.out`. The compiled binary is located within the `bin` folder. To execute the program, use the command `./bin/a.out`.

---

5 Conclusion

In summary, the application of OpenMP for parallelizing our numerical integration tasks and array operations has demonstrated substantial performance improvements. The speedup observed across varying thread configurations in both Problem 1 and Problem 2 underscores the effectiveness of OpenMP in optimizing computational workloads.

The visual representations, including performance graphs, provide a clear picture of the efficiency gains achieved through parallelization. Our experiments confirm that OpenMP is a valuable tool for enhancing the computational efficiency of scientific programming tasks.

References

[1] OpenMP Architecture Review Board. Reduction clauses and directives. `https://www.openmp.org/spec-html/5.0/openmpsu107.html`, 2018. Accessed on December 8, 2023.

[2] dreamcrash (`https://stackoverflow.com/users/1366871/dreamcrash`). Openmp in c array reduction / parallelize the code. `https://stackoverflow.com/a/65872111/22647897`, 2021. Accessed on December 8, 2023.

[3] OpenAI. Chatgpt, 3.5. `https://chat.openai.com/share/af607756-dde1-491e-b084-2f4aed7bf6d9`, 2023. Accessed on December 8, 2023.