618 Project Proposal

Title

A Comparative Study of OpenMP and MPI for Parallelizing the Grey Wolf Optimizer

Team members: Tong Jin and Yiding Chang

Project website: https://github.com/tongjin0521/15x18-final-project

Summary

The project aims to compare the performance of OpenMP and MPI in parallelizing the Grey Wolf Optimizer algorithm, a metaheuristic algorithm used in optimization problems.

Background

The Grey Wolf Optimizer (GWO) is a nature-inspired metaheuristic algorithm that imitates the hunting behavior of grey wolves to optimize problem-solving. The algorithm is computationally intensive and requires several iterations to achieve an optimal solution. GWO works on the principle that wolves move toward their prey by mimicking their hunting behavior. In this context, the position of each wolf signifies a potential solution to the optimization problem. The wolves hunt based on the alpha wolf's position, the beta wolf's position, and so on, which are updated at every iteration.

The GWO algorithm can be parallelized by employing parallelization on fitness evaluation functions and position update functions. These functions can be concurrently executed by multiple threads or processes on various computing resources, such as high-performance computing clusters or multi-core CPUs. Parallelizing these functions reduces the computation time, allowing the algorithm to converge toward an optimal solution faster.

Our project aims to compare the performance of two parallelization methods, OpenMP and MPI, in parallelizing the GWO algorithm. OpenMP is a shared memory parallelization technique that permits the parallel execution of code blocks within a single program on multi-core CPUs. In contrast, MPI is a message-passing parallelization technique that enables multiple processes to communicate and synchronize on different computing resources.

To determine the effectiveness of each parallelization method in accelerating the GWO algorithm, the project evaluates their performance on different resources. By comparing the results obtained from the two methods, it is possible to determine which method is more suitable for parallelizing the GWO algorithm. Moreover, it can provide insights into the best practices for parallelizing other nature-inspired metaheuristic algorithms.

Challenge

The challenge in parallelizing the GWO algorithm includes the lack of a serial implementation of the GWO algorithm in C++ available on the internet. This requires the development of a serial implementation of the algorithm before parallelizing it, which can add additional complexity and effort to the project.

The workload of the GWO algorithm involves the evaluation of the fitness function and the updating of the position of the wolves. Each iteration is also dependent on the previous one. The fitness function requires the evaluation of the objective function for each candidate solution, which can involve a significant amount of computation. The position update function involves the calculation of the new position of the alpha, beta, and delta wolves, which depends on the current position of all the wolves. This may lead to a high communication-to-computation ratio, which can limit the scalability of the parallelization.

The constraints in parallelizing the GWO algorithm include scalability and divergent execution. As mentioned above, there may be a high communication-to-computation ratio, which leads to the limit of stability. Also, the GWO algorithm involves different types of wolves, including the alpha, beta, and delta wolves, each with its own update rule. Parallelizing the algorithm among these wolves can be challenging, as the updates of the different types of wolves need to be coordinated and synchronized.

Resources

For this project, we plan to run our code on two different computing clusters for comparison: the Gates cluster and the PSC Bridges-2 RM machines. As for the code base, current open-source algorithm implementations are mostly in Python and utilize machine learning frameworks, which do not quite align with our needs. We plan to rewrite the algorithm in C++ based on Python implementations such as https://github.com/KaushalSahu/Grey-Wolf-Optimization. We also plan to refer to the original paper for detailed reference.

Goals and deliverables

Goals in case the work goes more slowly:

- Finish a sequential C++ implementation of the algorithm.
- Finish at least one implementation of the algorithm using either OpenMP or MPI.
- Calculate the speedup compared with the sequential baseline and analyze the results.

Goals that we plan to Achieve normally:

- Finish the implementation of the Grey Wolf Optimization algorithm using both OpenMP and MPI.
- Run both implementations on both the Gates cluster and the PSC Bridges-2 RM machines.
- Compare the speedups and the results of the two implementations and try to explain the results.

Goals that we hope to achieve if the project goes really well:

- Compare the speedup with a Python implementation using Big Data frameworks such as Spark.
- Try to further improve the performance of our implementations.

Planned demo:

We are planning on offering an interactive demo, which provides the opportunity for users to select an optimization target function and compare the speedups of all the programs under comparison. If for any reason the program should run too slowly for live demonstration purposes, we will display pre-run results of our chosen target functions.

Analysis purposes and plans:

In our analysis, we hope to learn about the performance difference between OpenMP and MPI in parallelizing the Grey Wolf Optimizer algorithm as an example. Using the comparison results, our analysis aims to answer the question of which parallelization

method is more suitable for parallelizing the GWO algorithm and provide insights into the best practices for parallelizing other nature-inspired metaheuristic algorithms.

Platform choice

For this project, we plan to run our code on the Gates cluster and the PSC Bridges-2 RM machines. We chose them because they are the most computationally powerful clusters we have access to, and the nature of the problem calls for clusters with high CPU computational power. We will be using C++ for the implementation and may use other scripting languages like Python to display our results.

Schedule

4.2-4.8: Start initial work on the project, including investigating the details of the algorithm and implementing a sequential C++ version.

4.9-4.15: Implement the OpenMP version of the algorithm.

4.16-4.22: Analyze the difference between the OpenMP version and the sequential version. Write the Milestone Report.

4.23-4.29: Implement the MPI version of the algorithm.

4.30-5.4: Do final analysis of all the implementations. Do the extra objective if time permits.