Project 1: Parallel implementation of search based on Fish School Behaviour

Note:

- The total mark for this project is 25. The due date is September 25, 2023, 11:59 pm.
- The project can be done either in a group of two, or individually. It is your responsibility to find a partner if you want to do the project in a group.
- Only one group member should submit the project.
- The submission should have two parts, a code file and a report in pdf format. You should zip these two files and submit the zip file.
- All submission is through cssubmit. I will create a submission folder soon.

Some clarifications:

- This project is very simple. It is set as a learning exercise in high performance computing, rather than for checking a right or wrong answer. In fact, this project does not have a right or wrong answer. The reason I have simplified the project to its present form, is due to the fact that this optimization strategy is not guaranteed to converge to a right answer. Evidence in the literature suggests that this convergence occurs in very high-dimensional space, and only sometime. The meaning of convergence here is that, all the fish collect to a common place in the lake. As such the implementation uses so much randomisation that no two executions of any implementation will give the same result.
- This project does not have any test case. Your program will initialize the fish school and simulate its behaviour. Hence, there is no way I can provide you with a test case.
- It is very easy to write a sequential code for this project. I have given some hints below. The parallelization of the code using OpenMP directives is also simple. The quality of a good project will be evaluated based on the experiments you do in parallelizing the code, and

how you analyze those experiments. Of course speedup (sequential time/parallel time) is an indication of a good implementation, however, what I am looking for is the analysis of different scenarions, and different approaches to parallelization. As such, I will check the correctness of the code, but that is not the main focus of this project.

- Apart from the aspects on high performance computing, quality of written communication is very important. I may also ask you to come and meet me so that I can verify that you have written your code and thought about the parallelization.

1 Introduction

Multi-dimensional optimization is a very important problem in computer science. Given some objective function involving several variables, the task is to find values of the variables for which the function attains a minimum or a maximum value. These problems are very hard (NP-hard) and some of the hardest problems known in computer science. Hence, many heuristic methods have been developed for solving these problems, that may not give the best solution, but try to find good solutions. Fish School Behaviour (FSB) is one such heuristic method.

Most of these heuristic algorithms provide opportunities for parallel implementation, and we will mainly focus on the parallelization of a simpler version of the FSB algorithm. Our focus here is not to solve the optimization problem, but to understand the process of parallelization of such optimization problems. This kind of parallelization is applicable for other optimization problems like *ant colony optimization* and *particle swarm optimization*.

2 The idea

Imagine a big lake and a school (a group of fish is called a school) of fish in the lake. There is food scattered in the lake, some places have more food, and some places have less food. Our aim is to direct a majority of the fish to the parts of the lake where there is more food. Each fish can eat and move - the details of these activities are below. So the idea is that if a fish can get some food, its weight increases and it moves. Though each fish moves randomly towards the direction where it thinks there is more food, we also

help the fish to choose the direction, depending on the collective experience of the school. This means, that if many of the fish have found food, the other fish who have not found food are oriented according to the direction where food is available.

We will imagine that the lake is represented by a 200×200 square. The center of this square is the point $(0, 0)$, and x coordinates are (positive) negative to the (right) left of the center, and similarly, the y coordinates are (positive) negative to (above) below the center. We will distribute the fish randomly within this square by generating random numbers between $(-100, 100)$ for x coordinates, and $(-100, 100)$ for y coordinates. Each fish has a coordinate (x, y). The distance of a fish with coordinates (x, y) is $\sqrt{(x^2 + y^2)}$ from the origin. We will then simulate the behaviour of the school. The simulation will progress in rounds, and each fish will take some actions in each round, and also we will orient all the fishes in each round. The aim is to divide the entire lake into smaller parts and the computation in each part will be taken care of by a thread (the number of parts will depend on the number of threads).

The actions of individual fish:

Eat: Each fish is born with a fixed weight w, and this weight can increase up to $2w$ (you can choose some suitable values for w). We introduce the objective function at this stage. The objective function is either maximized or minimized in an optimization problem. In our case, we use a very simple objective function $f = \sum_{i=1}^{N} \sqrt{(x_i^2 + y_i^2)}$. Though the x and y coordinates can be negative as well as positive, this sum is always positive and reaches a minimum when it is 0. The sum is over all the fish in the square. So this is quite a heavy computation if there is a large number of fishes and it requires to be done by each thread, and also threads must cooperate. The weight of a fish i at simulation step $(t + 1)$ is denoted by $w_i(t + 1)$ and the weight at simulation step t is denoted by $w_i(t)$.

$$w_i(t + 1) = w_i(t) + \frac{\delta(f_i)}{max(\delta(f_i))}$$

here, $\delta(f_i)$ is the change in the objective function after the i-th fish has randomly swam in the $(i + 1)$-th step. You have to replace the coordinates of the fish after it swims (see below) in this step in the equation

for f and the coordinates of the fish in the previous step. The difference of these two values of f is δf_i. Then you have to find the maximum such difference for all the fishes and find the new weight according to the equation above. There is no previous step when the simulation starts, so you can choose a suitable random value for $\frac{\delta(f_i)}{max(\delta(f_i))}$ in the first step.

Swim: A fish swims in a random direction if it can eat in the current round. The swimming is simulated by generating two random numbers between -0.1 and 0.1 and adding to the x and y coordinates of the fish.

Collective action:

The collective action is to orient all the fishes towards the barycentre of the school. This is a bit complex in actual fish school simulation, and we will avoid the complications and instead see how much parallelization we can get from multi-threaded programming. So we will simply do the following computation in each simulation step to calculate the *barycentre* of the fish school, but we will not actually orient the fish (this is required for solving the optimization problem).

$$Bari = \frac{\sum_{i=1}^{N}(\sqrt{x_i^2 + y_i^2}\ w_i(t))}{\sum_{i=1}^{N}\sqrt{x_i^2 + y_i^2}}$$

In other words, for every fish i, you have to calculate $(\sqrt{x_i^2 + y_i^2})w_i(t)$ and divide it by $\sum_{i=1}^{N}\sqrt{x_i^2 + y_i^2}$ to calculate the barycenter of the fish school. This computation will require the threads to cooperate in a way that we have discussed in the lectures.

3 Project deliverables

- Write a sequential C program for this simulation. The number of steps in the simulation should be specified as a constant with a `#define` at the top of the program, so that you can experiment with different number of simulation steps. You should also use a `#define` at the top of the program to indicate the number of fish in the school. This number should be possible to vary and test your program for different number of fish. You should time this sequential code.

- Write a multi-threaded code using OpenMP for this problem. You should experiment with different number of threads, different scheduling strategies, and any other OpenMP constructs that you think may be useful for better parallelization. Remember that not everything will give you speedup (the ratio of sequential time and parallel time), and also larger number of threads may not give better speedup, but what I am looking for is, you should do extensive experiments, plot graphs showing your experimental results, and explain your results.
- You should also experiment with the cache behaviour of Setonix, whether you find some performance variations while partitioning your array of fishes differently.
- I cannot suggest all experiments that you should conduct. It is up to you to design your experiment depending on your experience in running your program in Setonix.

4 Some C programming issues

You should not use static arrays for generating a large number of fish. If you have followed the lecture on computer architecture, you know that it is best to allocate a large amount of space required in your program in the heap, as all static arrays are allocated space in the stack, and usually space available in stack is much less compared to the space available in the heap. I expect you to experiment with very large numbers of fish that will be beyond the capacity of the stack provided by Setonix. There is another small difficulty in our problem. Each fish has an x and a y coordinate. Though one can use two different arrays, but it is very messy. The best way to store related data in a C program is to use a structure (it is a bit like a class in Java where you store many class variables in a class definition). Our structure for fish will look like this:

```
struct fish {
  float x;
  float y;
  //other protperties of fish if you need
```

```
};
```

You can then declare a fish:

```
struct fish fish1;
```

If you declare a fish like this, you can refer to its x and y coordinates using the 'dot' operator. For example, the x coordinate of `fish1` is `fish1.x`.

You can also declare an array of fish:

```
struct fish fishes[100]; //100 fish
```

You can again use the 'dot' operator to access the x coordinate, e.g., `fishes[1].x`.

However, as explained above, this is allocation in the stack, and is not ideal. To allocate in the heap, first declare:

```
struct fish *fishes;
```

This declares a pointer to the structure `fish`. An important thing to understand is that this is only a declaration of a pointer, and no space has been allocated yet. Now you can allocate a lot of fishes by:

```
fishes=(struct fish*) malloc(1000000*sizeof(struct fish));//allocate
                                                          //1000000 fishes
```

`sizeof` is an operator that calculates the size of a datatype in C. The datatype can be a fundamental datatype like `int`, but here it is a user-defined datatype `fish`. You want to allocate 1000000 fishes, so you multiply the `sizeof(fish))` by 1000000. `malloc` is a C library function that allocates space in the heap. In this case you are allocating space for 1000000 fishes in the heap.

You can now access any of the 1000000 fishes and their x and y coordinates. Here is a simple example program with two fishes:

```
#include<ctype.h>
#include<stdlib.h>
void main()
{
```

```
struct fish{
 int x;
 int y;
};

struct fish *fishes;
fishes = (struct fish*)malloc(2*sizeof (struct fish));

fishes->x=1;
fishes->y=1;

(fishes+1)->x=2;
(fishes+1)->y=2;

printf("x coordinate of first fish=%d\n",fishes->x);
printf("x coordinate of second fish=%d\n",(fishes+1)->x);

}
```

Since fishes is now a pointer (it stores the address) of the starting location for two fishes, we have to use the `->` operator. Also, `(fishes+1)` indicates the starting location of the second fish here. Please understand the use of these notations by writing small C programs.

Once you have allocated random coordinates to your fishes, you can partition the work among the threads by sorting. You can use any sorting algorithm you like, however, you should give references to any code that you have used.

5 Marking guide

- Correct sequential implementation with allocation in stack - 2 marks (+3 marks for allocation in heap).
- Correct parallel implementation - 5 marks
- Experiments with threads and fishes:
  - Fix a (large) number of fishes, and experiment with speedup for at least five different numbers of threads - 4 marks

  - Fix a number of threads, and experiment with speedup for at least five different numbers of fishes - 4 marks
  - A good analysis of your results is required for full marks.

- Experiments with difference thread scheduling methods and their analyses - 4 marks

- Any other experiments and quality of the report - 3 marks.