Exercise Introduction

Before attempting the exercises in this document please ensure that you have read and understood the key topics covered in tutorial.

Contents

Task 1: Task (10 min)

- Theoretical analysis,
  1. Compute the computing power of the present computing system as defined in "Parallel Computing: Basic Principles" by Oswald Haan in this course on Wednesday (19.04.2023).
  2. Find out the number of respective operations in the following functions:
     a) Skeleton Code for Function 1 (language C):

```c
static inline double f(double x) {
    return 4.0/(1.0+x*x);
}
```

     b) Skeleton Code for Function 2 (language C):

```c
double Trap(
  double left_endpt /*in */,
  double right_endpt /*in */,
  uint64_t trap_count /*in */,
  double base_len /*in */) {

  double estimate, x;
  uint64_t i;
  estimate = ( f(left_endpt) + f(right_endpt) )/2.0;
  for (i = 1; i <= trap_count-1; i++) {
    x = left_endpt + i*base_len;
    estimate += f(x);
  }
  estimate = estimate*base_len;
  return estimate;
} /* Trap */
```

Task 2: Task (35 min)

- Do the scaling benchmark for the provided sample code that uses MPI (as presented for strong and week scaling benchmark).

```c
// This code example belongs to the Parallel Computing course exercises of Gottingen University.

#include <stdio.h>
#include <mpi.h>
#include <stdlib.h>
#include <stdint.h>

static inline double f(double x) {
    return 4.0/(1.0+x*x);
}

double Trap(
  double left_endpt /*in */,
  double right_endpt /*in */,
  uint64_t trap_count /*in */,
  double base_len /*in */) {

  double estimate, x;
  uint64_t i;
  estimate = ( f(left_endpt) + f(right_endpt) )/2.0;
  for (i = 1; i <= trap_count-1; i++) {
    x = left_endpt + i*base_len;
    estimate += f(x);
  }
  estimate = estimate*base_len;
  return estimate;
} /* Trap */

int main(int argc, char * const* argv) {
  uint64_t n = atoi(argv[1]) * 1000llu;
  uint64_t local_n;
  int my_rank, comm_sz;
  double a = 0.0, b = 3.0, h, local_a, local_b;
  double local_int, total_int;
  int source;

  MPI_Init(NULL, NULL);
  MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);
  MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);

  h = (b-a)/n; /* h is the same for all processes */
  local_n = n/comm_sz; /* So is the number of trapezoids */
  local_a = a + my_rank*local_n*h;
  local_b = local_a + local_n*h;
  local_int = Trap(local_a, local_b, local_n, h);
  if (my_rank != 0) {
    MPI_Send(&local_int, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
  } else {
    total_int = local_int;
    for (source = 1; source < comm_sz; source++) {
      MPI_Recv(&local_int, 1, MPI_DOUBLE, source, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
      total_int += local_int;
    }
  }

  if (my_rank == 0) {
    printf("With n = %lld trapezoids, our estimate\n", (long long) n);
    printf("of the integral from %f to %f = %.15e\n", a, b, total_int);
  }

  MPI_Finalize();
```

```
62   return 0;
63 } /* main */
```

- List of estimations to do:
  1. Ideal (theoretical) total number of operations,
  2. Ideal number of processors required for the execution of calculated operations,
  3. Ideal (calculated/estimated) total time for the full execution,
  4. Actual total number of operations on execution of the code.
  5. Actual number of processors involved in during the execution.
  6. Actual wall-clock time taken for the full execution.
  7. Repeat the steps to see how often you get the same result.

Task 3: Task (20 min)

- Document and complete your benchmarking report.
- Include all the details you argue relevant on your critical thinking.
- For convenience you could also store your records in a CSV file, with the file name "hpctrainingNN.csv".

Hints

- You could also create and include graphs and chart plots as felt necessary.
- Here "NN" is your user code. You could also share your findings during the session for the feedback.