Kennesaw State University

CS 7172 Parallel and Distributed Computing - Spring 2020

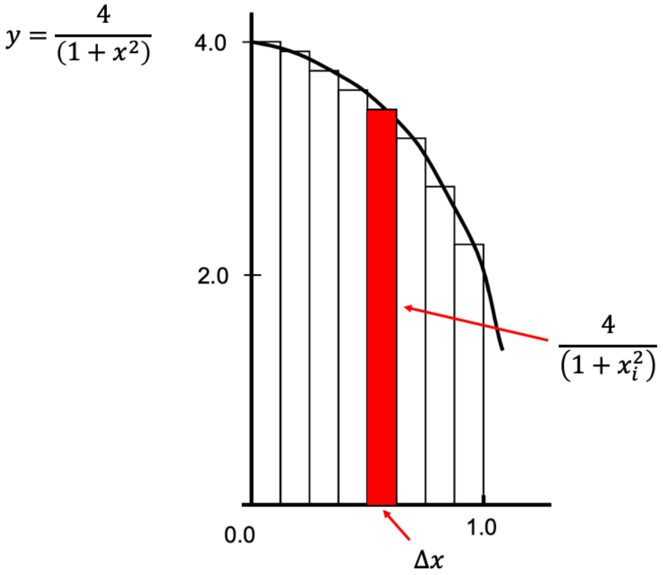
Project 3 – MPI

Instructor: Kun Suo
Points Possible: 100
Due date: check on the D2L

Task 1: calculate PI with MPI (50 points):

Mathematically, we know the following equation:

$$\int_0^1 \frac{4}{(1+x^2)}dx = \pi$$



We can approximate the value of π as a sum of rectangles:

$$\sum_{i=0}^{N} f(x_i)\Delta x \approx \pi$$

Where each rectangle has width Δx and height $F(x_i)$ at the middle of interval i.

The following code implements the above calculation of PI. We divide the area between 0 and 1 into 100000 small rectangles and the value of PI is approximately equal to the sum of all rectangles' size. However, the program executes in the sequential implementation.

https://github.com/kevinsuo/CS7172/blob/master/pi.c

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define NUMSTEPS 1000000

int main() {
        int i;
        double x, pi, sum = 0.0;
        struct timespec start, end;

        clock_gettime(CLOCK_MONOTONIC, &start);
        double step = 1.0/(double) NUMSTEPS;
        x = 0.5 * step;

        for (i=0;i<= NUMSTEPS; i++){
                x+=step;
                sum += 4.0/(1.0+x*x);
        }
        pi = step * sum;
        clock_gettime(CLOCK_MONOTONIC, &end);
        u_int64_t diff = 1000000000L * (end.tv_sec - start.tv_sec) + end.tv_nsec -
start.tv_nsec;

        printf("PI is %.20f\n",pi);
        printf("elapsed time = %llu nanoseconds\n", (long long unsigned int) diff);

        return 0;
}
```

Write a parallel program to calculate PI using MPI based on this sequential solution.

To compile the program with OpenMP, use:

```
$ mpicc -g program.c -o program.o
```

Please write a brief report introducing your implementation.
(Hint: MPI_Bcast and MPI_Reduce are required.)

Task 2 Soring in MPI (50 points):

https://github.com/kevinsuo/CS7172/blob/master/data.txt

The above link is a file which contains 1 million unsorted numbers.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

int data[1000000];

void swap(int* a, int* b)
{
      int t = *a; *a = *b; *b = t;
}

int partition (int arr[], int low, int high)
{
      int pivot = arr[high];       // pivot
      int i = (low - 1);                    // Index of smaller element

      for (int j = low; j <= high- 1; j++)
      {
               if (arr[j] < pivot)
               {
                         i++;    // increment index of smaller element
                         swap(&arr[i], &arr[j]);
               }
      }
      swap(&arr[i + 1], &arr[high]);
      return (i + 1);
}

void quickSort(int arr[], int low, int high)
{
      if (low < high)
      {
               int pi = partition(arr, low, high);
               quickSort(arr, low, pi - 1);
               quickSort(arr, pi + 1, high);
      }
}

void printArray(int arr[], int size)
{
      int i;
      for (i=0; i < size; i++)
               printf("%d\n", arr[i]);
}

int main()
{
      //read the unsorted array
      char str[100];
      int count = 0;
      struct timespec start, end;
               FILE* fp = fopen("data.txt", "r");
      while (fscanf(fp, "%s", str) != EOF) {
               data[count] = atoi(str);
               count++;
      }
      //quick sort the array
      clock_gettime(CLOCK_MONOTONIC, &start);
      quickSort(data, 0, count - 1);
      clock_gettime(CLOCK_MONOTONIC, &end);

      u_int64_t diff = 1000000000L * (end.tv_sec - start.tv_sec) + end.tv_nsec -
start.tv_nsec;
      printf("elapsed time = %llu nanoseconds\n", (long long unsigned int) diff);

//    printArray(data, count);

      fclose(fp);

      return 0;
}
```

---

https://github.com/kevinsuo/CS7172/blob/master/quicksort.c

The above source code file quicksort.c is a Divide and Conquer algorithm which sorts the above 1 million unsorted numbers. However, the program executes in the sequential implementation. Please write a parallel program quick sort using MPI based on this sequential solution. Compare the parallel program execution time with the sequential version and write a report with data and figures introducing your implementation.

(Hint: MPI_Send and MPI_Recv are required.)

Submitting Assignment

Submit your assignment zip file through D2L using the appropriate assignment link. For task 1 and 2, please submit the ***source code*** , ***screenshot of output*** and ***report***.