Saint Petersburg Electrotechnical University LETI

Faculty of Computer Science and Technology, Department of Computer Science and Engineering

Master of Computer Science and Knowledge discovery

Course name: Parallel Computing

Student Name: Omar Taha Mohammed

Professor: Natalya V. Razmochaeva

Table of Contents

Task Description ***3***

Answer ***4***

Output ***5***

Result Table ***6***

Result Chart ***6***

Task Description

Calculate PI value in parallel in C++

#include <unistd.h>

What we have

int filedes[2];	// The array pipefd is used to return two file descriptors referring to the ends of the pipe
	// pipefd[0] refers to the read end of the pipe.
	// pipefd[1] refers to the write end of the pipe.

int pipe(int filedes[2]);	// creating pipe using filedes // 0 if successful, –1 if successful.
close(file_desr)	
write(FD, char* msg, N)	
read(FD, char* msg, K)	
and all function from HW #1	

Theoretical bases
Inter-process communication is carried out in the following ways:

- Environment variables
- Signals
- Channels (**pipes**)
  - Named
  - **Unnamed**
- Sockets

Channel (pipe) is a data stream between two or more processes that has an interface similar to reading or writing to a file.
Channels have two limitations:

1. Historically, they are **simplex** (that is, data can be transmitted over them in only one direction);
2. Channels can only be used to organize interaction between processes that have a common ancestor. Typically, a channel is created by the parent process, which then calls the fork () function, after which this channel can be used to communicate between the parent and child processes.

Answer

1- Using OpenMP for multithreading
2- Using a critical section to remove impact of false sharing
3- Loop Reduction using openMP
4- Barriers, for loops and the nowait clause
5- Send Data Among Processes using fork,pipe,Write, and read… .

```cpp
//
//  main.cpp
//  ParallelComputing
//
//  Created by Omar Mohammed on 5/31/20.
//  Copyright © 2020 Omar Mohammed. All rights reserved.
//

#include <iostream>
#include <omp.h>
#include <unistd.h>
#include <limits>
#include <stdio.h>
int main (int argc, char *argv[])
{
    long num_steps = 100000000;
    int padNumber=8;
    int ThreadNumber =2;
       double step;
       double n;
       int fd[2];
       pid_t pid;
       char line[255];
     double x;
       int i, threadNumbers; double pi, sum[ThreadNumber][padNumber];
       step = 1.0/(double) num_steps;
       omp_set_num_threads(ThreadNumber);


       if (pipe(fd) < 0) {
           printf("Error while call function pipe\n");
           return 1;
       }


       if ((pid = fork()) < 0) {
           printf("Error while call function fork\n");
           return 1;
       } else if (pid > 0) {
           close(fd[0]);
           #pragma omp parallel
               { int i, id,nthrds;


                   id = omp_get_thread_num();
                   nthrds = omp_get_num_threads();
```

```
47
48                  if (id == 0) threadNumbers = nthrds;
49                  for (i=0;i< num_steps; i+=nthrds)
50
51                  {
52
53                      x = (i+0.5)*step;
54                      sum[id][0] += 4.0/(1.0+x*x);
55
56                  }
57              }
58
59          for(i=0, pi=0.0;i<threadNumbers;i++)pi += sum[i][0] * step;
60          std::string pitosend=std::to_string(pi);
61          write(fd[1],pitosend.c_str(),1000);
62      } else {
63          close(fd[1]);
64          n = read(fd[0], line, 255);
65          std::cout.precision(n);
66          printf("Message coming from another process PI value is= %s \n", line);
67      }
68
69
70
71      return 0;
72  }
73
74
```

Output

```
Message coming from another process PI value is= 3.141587
```

Result Table

Threads	1st SPMD	1st SPMD padded	SPMD critical
1	1.86	1.86	1.87
2	1.03	1.01	1
3	1.08	0.69	0.68
4	0.97	0.53	0.53

Result Chart