1886

ETU "LETI"

SAINT PETERSBURG ELECTROTECHNICAL UNIVERSITY

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: Sayed Moeid Heidari

Professor: Natalya V. Razmochaeva

Spring 2020

Table of Contents

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

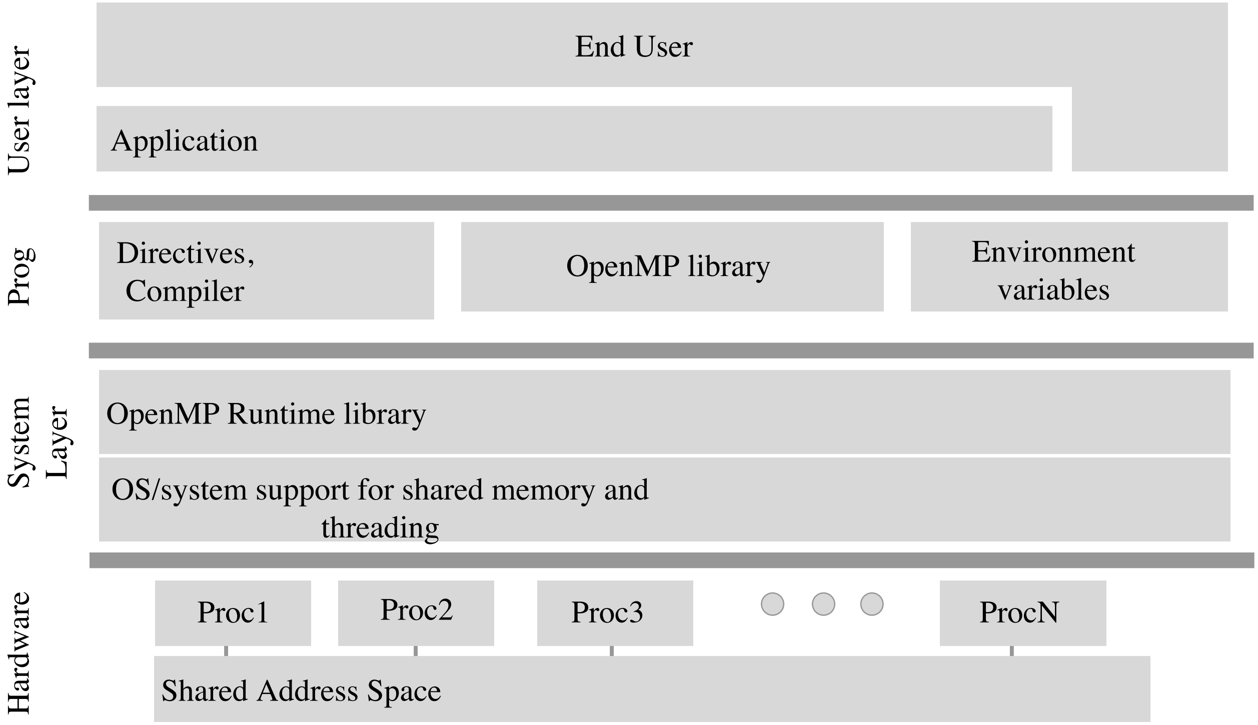
Channel (pipe) is a data stream between two or more processes that has an interface similar to reading or writing to a file.
Channels have two limitations:

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

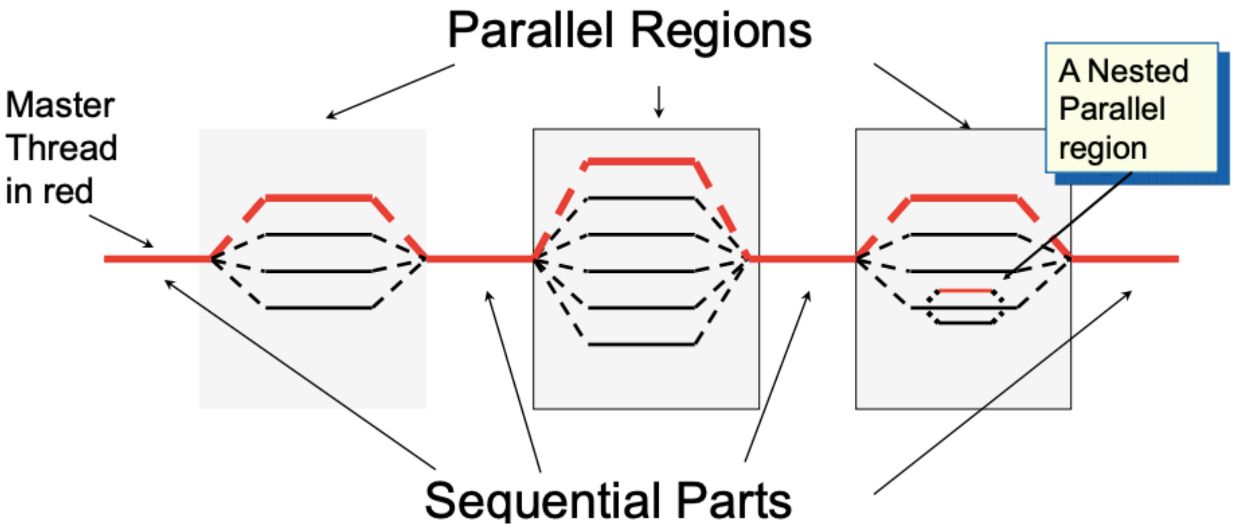
OpenMP: An API for Writing Multithreaded Applications

- A set of compiler directives and library routines for parallel application programmers
- Greatly simplifies writing multi-threaded (MT) programs in Fortran, C and C++
- Standardizes established SMP practice + vectorization and heterogeneous device programming

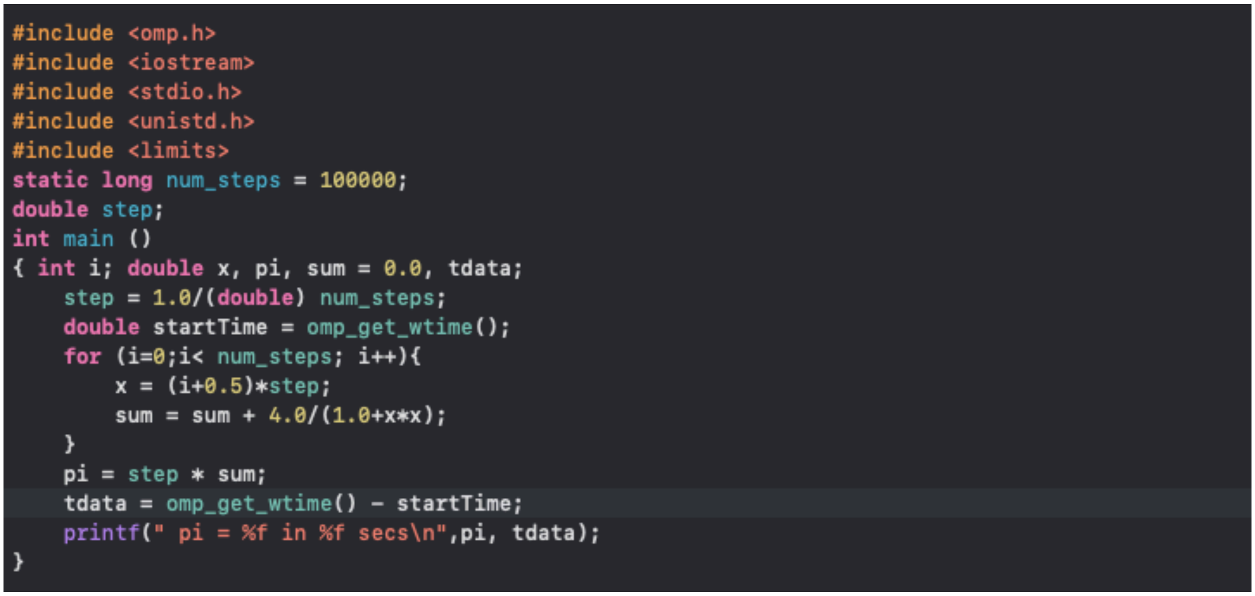


Fork-Join Parallelism:

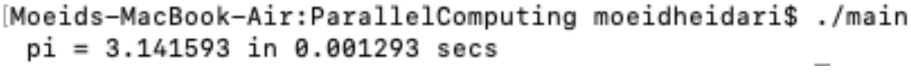
- Master thread spawns a team of threads as needed.
- Parallelism added incrementally until performance goals are met, i.e., the sequential program evolves into a parallel program



Pi Calculation Serial program, (Original Serial pi program with 100000000 steps ran in 1.83 seconds)



```
#include <omp.h>
#include <iostream>
#include <stdio.h>
#include <unistd.h>
#include <limits>
static long num_steps = 100000;
double step;
int main ()
{ int i; double x, pi, sum = 0.0, tdata;
    step = 1.0/(double) num_steps;
    double startTime = omp_get_wtime();
    for (i=0;i< num_steps; i++){
        x = (i+0.5)*step;
        sum = sum + 4.0/(1.0+x*x);
    }
    pi = step * sum;
    tdata = omp_get_wtime() - startTime;
    printf(" pi = %f in %f secs\n",pi, tdata);
}
```



```
[Moeids-MacBook-Air:ParallelComputing moeidheidari$ ./main
  pi = 3.141593 in 0.001293 secs
```

Compute N independent tasks on one processor:

Timeseq(1) = Tload + N*Ttask + Tconsume

Compute N independent tasks with P processors

Timepar(P) = Tload + (N/P)*Ttask + Tconsume

Speedup

the increasedperformance from running on Pprocessors

Speedup: $$S(P) = \frac{\mathrm{Time}_{seq}(1)}{\mathrm{Time}_{par}(p)}$$

Perfect Linear Speedup: $S(P) = P$

Super-linear Speedup: $S(P) > P$

maximum speedup you can expect from a parallel program

$$Time_{par}(P) = (serial\_fraction + \frac{parallel\_fraction}{P}) * Time_{seq}$$

If serial_fraction is alpha and parallel_fraction is (1- alpha) then the speedup is

$$S(P) = \frac{Time_{seq}}{Time_{par}(P)} = \frac{Time_{seq}}{(\alpha + \frac{1-\alpha}{P}) * Time_{seq}} = \frac{1}{\alpha + \frac{1-\alpha}{P}}$$

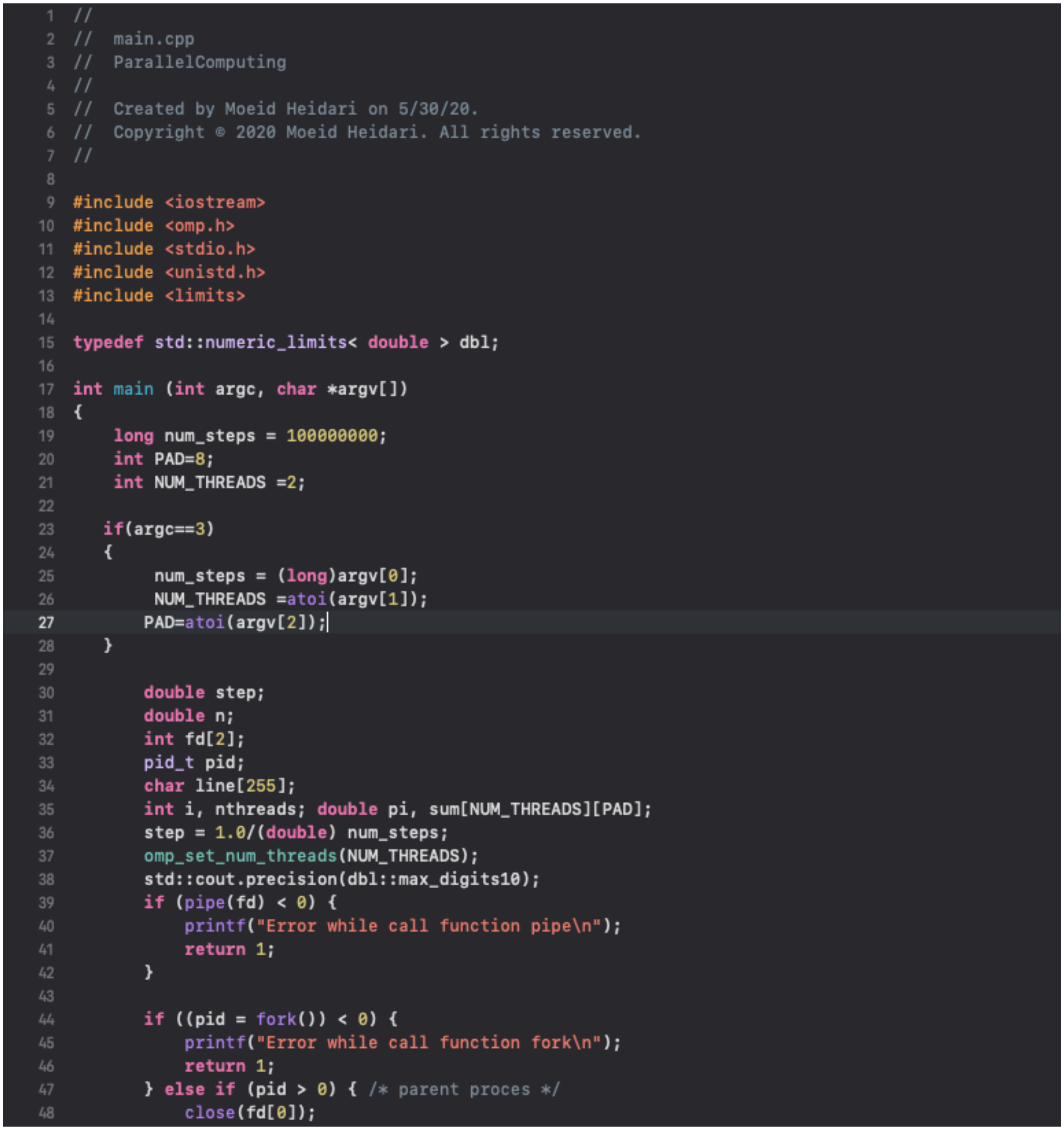
If there is infinite number of processors : $P \rightarrow \infty$

The maximum possible speed up is : $S= \frac{1}{a}$

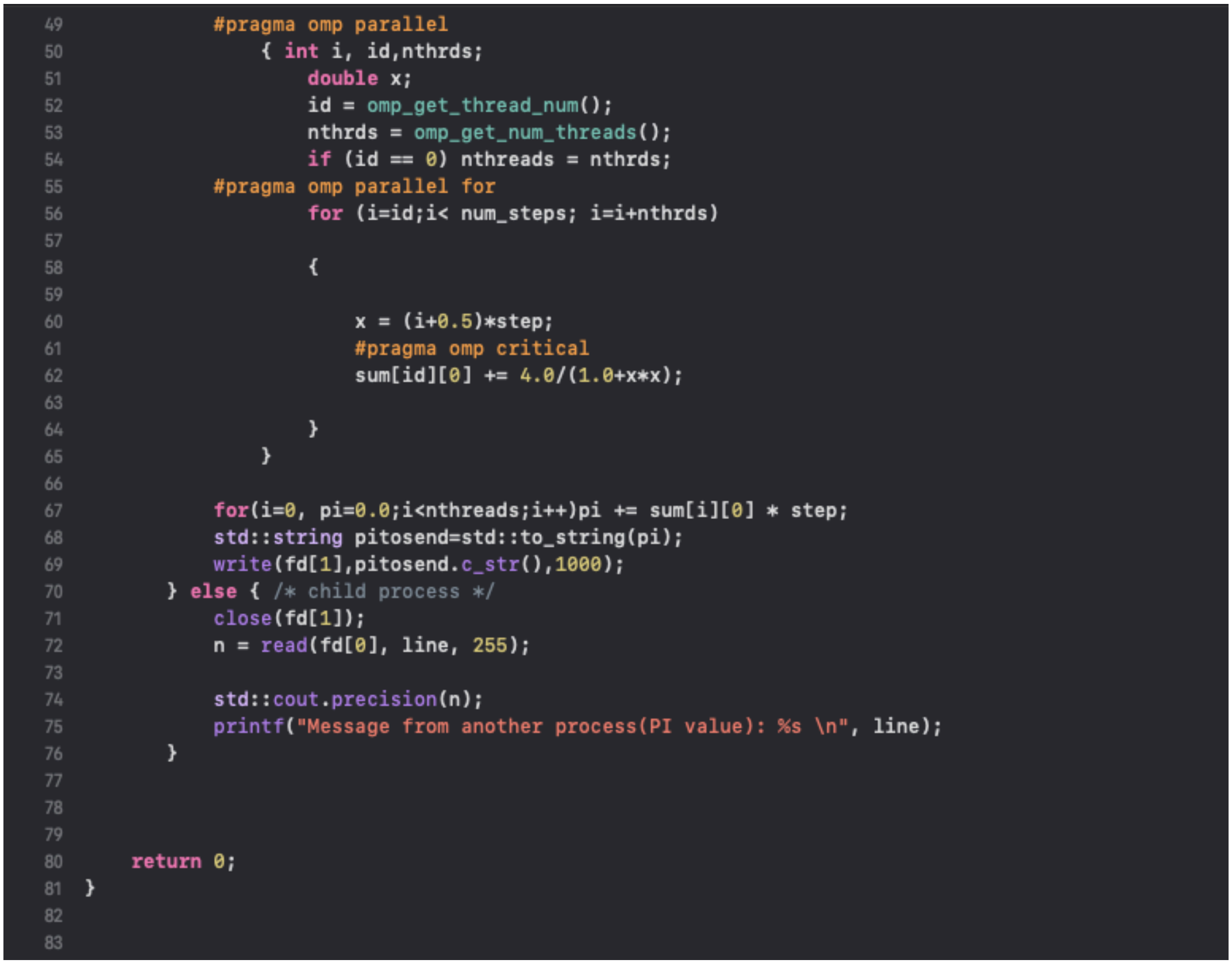
Features of this solution:

1- Using OpenMP for multithreading
2- Using a critical section to remove impact of false sharing
3- Combined parallel/worksharing construct
4- Loop Reduction using openMP
5- Barriers, for loops and the nowait clause
6- Send Data Among Processes using fork,pipe,Write, and read… .
7- Estimating PI value using Monte Carlo
8- Command line arguments (number of terms, number of threads, and number of pads)

Code

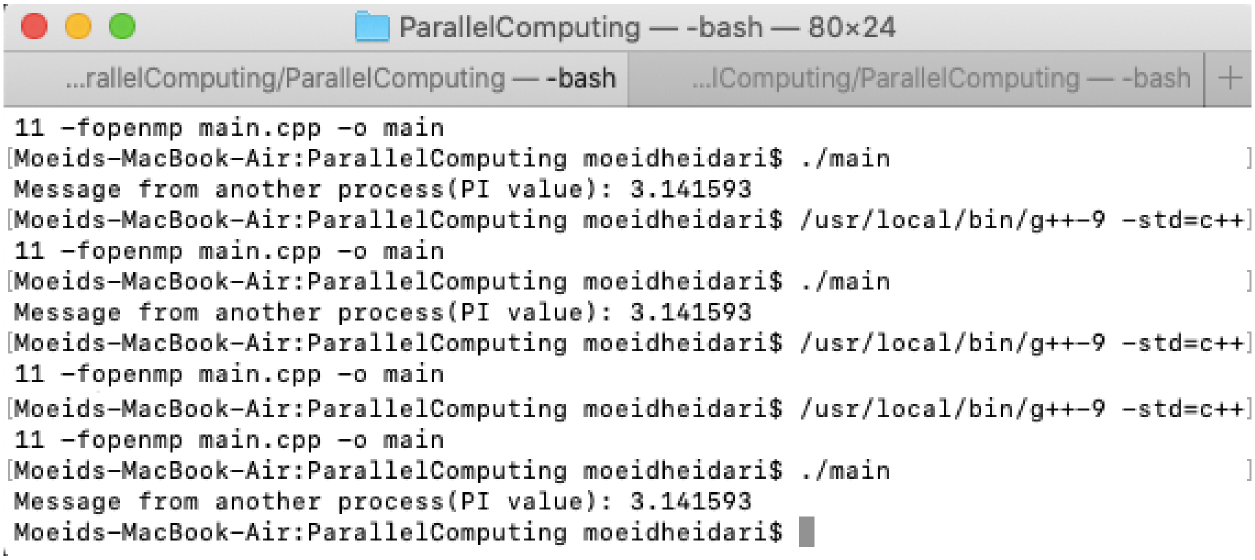


```
//
//  main.cpp
//  ParallelComputing
//
//  Created by Moeid Heidari on 5/30/20.
//  Copyright © 2020 Moeid Heidari. All rights reserved.
//

#include <iostream>
#include <omp.h>
#include <stdio.h>
#include <unistd.h>
#include <limits>

typedef std::numeric_limits< double > dbl;

int main (int argc, char *argv[])
{
    long num_steps = 100000000;
    int PAD=8;
    int NUM_THREADS =2;

   if(argc==3)
   {
        num_steps = (long)argv[0];
        NUM_THREADS =atoi(argv[1]);
       PAD=atoi(argv[2]);
   }

       double step;
       double n;
       int fd[2];
       pid_t pid;
       char line[255];
       int i, nthreads; double pi, sum[NUM_THREADS][PAD];
       step = 1.0/(double) num_steps;
       omp_set_num_threads(NUM_THREADS);
       std::cout.precision(dbl::max_digits10);
       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) { /* parent proces */
           close(fd[0]);
```



```
           #pragma omp parallel
               { int i, id,nthrds;
                   double x;
                   id = omp_get_thread_num();
                   nthrds = omp_get_num_threads();
                   if (id == 0) nthreads = nthrds;
           #pragma omp parallel for
                   for (i=id;i< num_steps; i=i+nthrds)

                   {

                       x = (i+0.5)*step;
                       #pragma omp critical
                       sum[id][0] += 4.0/(1.0+x*x);

                   }
               }

           for(i=0, pi=0.0;i<nthreads;i++)pi += sum[i][0] * step;
           std::string pitosend=std::to_string(pi);
           write(fd[1],pitosend.c_str(),1000);
       } else { /* child process */
           close(fd[1]);
           n = read(fd[0], line, 255);

           std::cout.precision(n);
           printf("Message from another process(PI value): %s \n", line);
       }



    return 0;
}


```

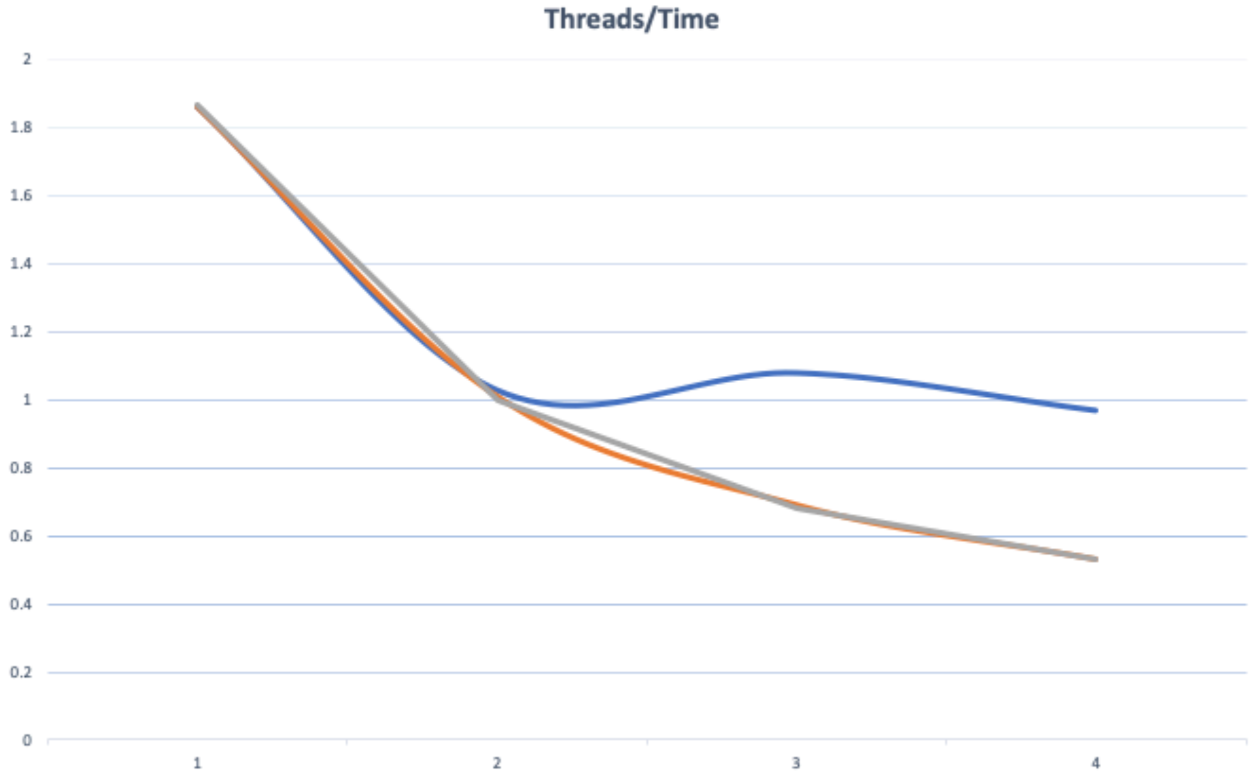
Output



ParallelComputing — -bash — 80×24
...rallelComputing/ParallelComputing — -bash
...lComputing/ParallelComputing — -bash

```
11 -fopenmp main.cpp -o main
[Moeids-MacBook-Air:ParallelComputing moeidheidari$ ./main                     ]
Message from another process(PI value): 3.141593
[Moeids-MacBook-Air:ParallelComputing moeidheidari$ /usr/local/bin/g++-9 -std=c++]
11 -fopenmp main.cpp -o main
[Moeids-MacBook-Air:ParallelComputing moeidheidari$ ./main                     ]
Message from another process(PI value): 3.141593
[Moeids-MacBook-Air:ParallelComputing moeidheidari$ /usr/local/bin/g++-9 -std=c++]
11 -fopenmp main.cpp -o main
[Moeids-MacBook-Air:ParallelComputing moeidheidari$ /usr/local/bin/g++-9 -std=c++]
11 -fopenmp main.cpp -o main
[Moeids-MacBook-Air:ParallelComputing moeidheidari$ ./main                     ]
Message from another process(PI value): 3.141593
Moeids-MacBook-Air:ParallelComputing moeidheidari$
```

Chart



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