PRÁCTICA 12

ALGORITMO PARALELOS PARTE I

## Alumno: Robles Martínez Héctor

## Número de Cuenta UNAM: 317356548

## Semestre: Tercer semestre de Ingeniería en Computación (2016)

## Semestre Actual: 2021-1

## Profesor: Jorge A. Solano G.

Objetivo:

Conocer y aprender a utilizar algunos patrones paralelos para el acceso a datos y distribu-ción de tareas que le servirán en el diseño e implementación de un algoritmo paralelo encomputadoras con arquitectura de memoria compartida.

Instrucciones:

•Realizar ejemplos del funcionamiento de directivas de OpenMP en lenguaje C. talescomo los constructores critical, section, single, master, barrier y las cláusulas reduc-tion y nowait que permitirán implementar algún patrón paralelo.

•Realizar programas que resuelvan un problema de forma paralela utilizando distintasdirectivas OpenMP.

Resultados obtenidos:

INTEGRAL:

SERIE:

#include <stdio.h>

#include <omp.h>

static long numSteps=10000000;

double step;

int main(){

double start = omp\_get\_wtime();

int i;

double x, pi, sum = 0.0;

step = 1.0/(double)numSteps;

for(i = 0; i < numSteps; i++){

x = (i+0.5)\*step;

sum = sum + 4.0/(1.0+x\*x);

}

pi = step \* sum;

double end = omp\_get\_wtime();

double time = end - start;

printf("Time: %f\n", time);

printf("Pi = %f\n", pi);

return 0;

}

SALIDA:

Time: 0.094558

Pi = 3.141593

PARALELA:

#include <stdio.h>

#include <omp.h>

#define NUM\_THREADS 2

static long numSteps=10000000;

double step;

int main(){

double start = omp\_get\_wtime();

int i = 0;

int thNums = omp\_get\_num\_threads();

double x, pi, suma[thNums], sum = 0.0;

step = 1.0/(double)numSteps;

for(i = 0; i< thNums; i++){

suma[i] = 0;

}

#pragma opm parallel

{

for(i = omp\_get\_thread\_num(); i <= (int)numSteps/thNums; i++){

x = (i+0.5)\*step;

suma[omp\_get\_thread\_num()] = suma[omp\_get\_thread\_num()] + 4.0/(1.0+x\*x);

}

}

for(i = 0; i < thNums; i++){

sum += suma[i];

}

pi = step \* sum;

double end = omp\_get\_wtime();

double time = end - start;

printf("Time: %f\n", time);

printf("Pi = %f\n", pi);

return 0;

}

SALIDA:

Time: 0.095112

Pi = 3.141593

PARALELA CON *UN CONSTRUCTOR*:

#include <stdio.h>

#include <omp.h>

#define NUM\_THREADS 12

static long numSteps=10000000;

double step;

int main(){

double start = omp\_get\_wtime();

int i = 0;

int thNums = omp\_get\_num\_threads();

double x, pi, sum = 0.0;

step = 1.0/(double)numSteps;

#pragma opm parallel

{

int i, id, nthrds;

double x, sum = 0.0;

id = omp\_get\_thread\_num();

nthrds = omp\_get\_num\_threads();

for(i = id; i <= numSteps; i+=nthrds){

x = (i+0.5)\*step;

sum += 4.0/(1.0+x\*x);

}

sum \*= step;

#pragma omp atomic

pi += sum;

}

double end = omp\_get\_wtime();

double time = end - start;

printf("Time: %f\n", time);

printf("Pi = %f\n", pi);

return 0;

}

SALIDA:

Time: 0.091873

Pi = 3.141593

BARRIER:

#include <stdio.h>

#include <omp.h>

#include <unistd.h>

#define NUM\_THREADS 4

int big\_call1(int id){

sleep(id);

printf("Big Call1 %d\n", id);

return id;

}

int big\_call2(int id){

sleep(id);

printf("Big Call2 %d\n", id);

return id;

}

int main(){

omp\_set\_num\_threads(NUM\_THREADS);

int A[NUM\_THREADS], B[NUM\_THREADS], id;

#pragma omp parallel

{

id = omp\_get\_thread\_num();

A[id] = big\_call1(id);

#pragma omp barrier

B[id] = big\_call2(id);

}

}

SALIDA:

Big Call1 0

Big Call1 1

Big Call1 2

Big Call1 3

Big Call2 3

Big Call2 3

Big Call2 3

Big Call2 3

CRITICAL:

#include <stdio.h>

#include <omp.h>

float bigJob(int i){

printf("i: %d\n", i);

return (float) i;

}

float consume(float b){

printf("b: %f\n", b);

return (float)b;

}

int main(){

int niters = 10;

float res = 0.0;

#pragma omp parallel

{

float B;

int i, id, nthrds;

id = omp\_get\_thread\_num();

nthrds = omp\_get\_num\_threads();

for(i = id; i<niters; i+=nthrds){

B=bigJob(i);

#pragma omp critical

res += consume(B);

}

printf("res= %f\n", res);

}

return 0;

}

SALIDA:

i: 0

b: 0.000000

res= 0.000000

i: 7

b: 7.000000

res= 7.000000

i: 9

b: 9.000000

i: 4

b: 4.000000

res= 20.000000

i: 3

b: 3.000000

res= 23.000000

i: 5

b: 5.000000

res= 28.000000

i: 1

b: 1.000000

res= 29.000000

i: 6

b: 6.000000

res= 35.000000

i: 2

b: 2.000000

res= 37.000000

res= 0.000000

i: 8

b: 8.000000

res= 45.000000

res= 0.000000

res= 16.000000

ATOMIC:

#include <stdio.h>

#include <omp.h>

double getNumber(int id){

return (double)(id%10);

}

double funcBig(double b){

if(b != (double)0){

return (double)(10%((int)b));

}else{

return 1;

}

}

int main(){

double A = 0, C = 0;

#pragma omp parallel

{

double tmp, B;

int id;

id = omp\_get\_thread\_num();

B = getNumber(id);

tmp = funcBig(B);

C += tmp;

printf("tmp: %f\n", tmp);

#pragma omp atomic

A+=tmp;

}

printf("A: %f\n", A);

printf("C: %f\n", C);

return 0;

}

SALIDA:

tmp: 1.000000

tmp: 1.000000

tmp: 2.000000

tmp: 3.000000

tmp: 1.000000

tmp: 4.000000

tmp: 0.000000

tmp: 0.000000

tmp: 0.000000

tmp: 2.000000

tmp: 0.000000

tmp: 1.000000

A: 15.000000

C: 7.000000

Conclusiones:

Esta práctica me sirvió para comprender cómo se pueden implementar los hilos y procesos paralelos en diferentes problemas y escenarios en la resolución de problemas.