**GARVIL JAIN 2022UCP1302 LAB5**

#include <stdio.h>

#include <omp.h>

double computeA() {

return 10.0;

}

double computeB() {

return 20.0;

}

int main() {

double res = 0.0;

double sharedVar = 0.0;

#pragma omp parallel

{

#pragma omp single

{

sharedVar = 5.0;

printf("Shared variable initialized by thread %d\n", omp\_get\_thread\_num());

}

#pragma omp sections reduction(+:res)

{

#pragma omp section

{

double resA = computeA() \* sharedVar;

printf("Section 1 computed by thread %d: %f\n", omp\_get\_thread\_num(), resA);

res += resA;

}

#pragma omp section

{

double resB = computeB() \* sharedVar;

printf("Section 2 computed by thread %d: %f\n", omp\_get\_thread\_num(), resB);

res += resB;

}

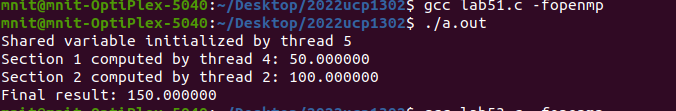
}

}

printf("Final result: %f\n", res);

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#include<time.h>

#define SIZE 1000000

double computeA() {

return 10.0;

}

double computeB() {

return 20.0;

}

double privateVar;

#pragma omp threadprivate(privateVar)

int main() {

srand(time(NULL));

int vec1[SIZE], vec2[SIZE];

long long dot\_product\_parallel = 0, dot\_product\_serial = 0;

double sharedVar = 0.0, section\_result = 0.0;

for (int i = 0; i < SIZE; i++) {

vec1[i] = rand() % 100 + 1;

vec2[i] = rand() % 100 + 1;

}

double start\_time\_serial = omp\_get\_wtime();

for (int i = 0; i < SIZE; i++) {

dot\_product\_serial += vec1[i] \* vec2[i];

}

double end\_time\_serial = omp\_get\_wtime();

double start\_time\_parallel = omp\_get\_wtime();

#pragma omp parallel reduction(+:dot\_product\_parallel, section\_result)

{

privateVar = omp\_get\_thread\_num() + 1.0;

#pragma omp single

{

sharedVar = 5.0;

printf("Shared variable set by thread %d\n", omp\_get\_thread\_num());

}

#pragma omp for

for (int i = 0; i < SIZE; i++) {

dot\_product\_parallel += vec1[i] \* vec2[i];

}

#pragma omp sections

{

#pragma omp section

{

double resA = computeA() \* privateVar \* sharedVar;

printf("Section computed by thread %d: %f\n", omp\_get\_thread\_num(), resA);

section\_result += resA;

}

#pragma omp section

{

double resB = computeB() \* privateVar \* sharedVar;

printf("Section computed by thread %d: %f\n", omp\_get\_thread\_num(), resB);

section\_result += resB;

}

}

}

double end\_time\_parallel = omp\_get\_wtime();

printf("Dot Product (Serial): %lld\n", dot\_product\_serial);

printf("Dot Product (Parallel): %lld\n", dot\_product\_parallel);

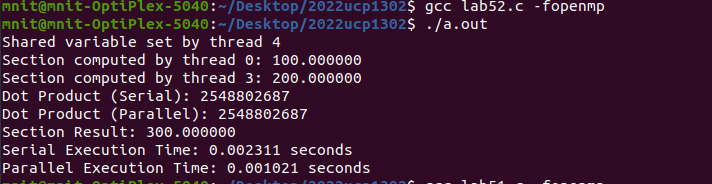
printf("Section Result: %f\n", section\_result);

printf("Serial Execution Time: %f seconds\n", end\_time\_serial - start\_time\_serial);

printf("Parallel Execution Time: %f seconds\n", end\_time\_parallel - start\_time\_parallel);

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

#include<time.h>

#define SIZE 100

int mat[SIZE][SIZE], vec[SIZE], result[SIZE];

int privateVar;

#pragma omp threadprivate(privateVar)

void initialize() {

srand(time(NULL));

for (int i = 0; i < SIZE; i++) {

vec[i] = rand() % 10;

for (int j = 0; j < SIZE; j++) {

mat[i][j] = rand() % 10;

}

}

}

void matrix\_vector\_multiplication() {

#pragma omp parallel

{

#pragma omp single

printf("Parallel computation started with %d threads.\n", omp\_get\_num\_threads());

#pragma omp for reduction(+:result[:SIZE])

for (int i = 0; i < SIZE; i++) {

result[i] = 0;

for (int j = 0; j < SIZE; j++) {

result[i] += mat[i][j] \* vec[j];

}

}

}

}

void compute\_sum\_and\_max() {

int sum = 0, max\_val = 0;

#pragma omp parallel sections

{

#pragma omp section

{

for (int i = 0; i < SIZE; i++) {

sum += result[i];

}

printf("Sum of all elements in result[]: %d\n", sum);

}

#pragma omp section

{

max\_val = result[0];

for (int i = 1; i < SIZE; i++) {

if (result[i] > max\_val) {

max\_val = result[i];

}

}

printf("Maximum value in result[]: %d\n", max\_val);

}

}

}

int main() {

srand(time(NULL)); // Seed for reproducibility

initialize();

matrix\_vector\_multiplication();

compute\_sum\_and\_max();

return 0;

}

