**Міністерство освіти і науки України**

**Черкаський національний університет імені Богдана Хмельницького**

**Інститут фізики, математики та комп'ютерно-інформаційних систем**

**Кафедра прикладної математики та інформатики**

**З В І Т**

**з лабораторної роботи**

студента 4-го курсу спеціальності «Прикладна математика»

***Годованюк Матвія Ігоровича***

**Черкаси – 2016 р.**

pi.c

#include <stdio.h>

#define N 2000000000

#define vl 1024

int main(void) {

double pi = 0.0f;

long long i;

#pragma acc parallel vector\_length(vl)

#pragma acc loop reduction(+:pi)

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

double t= (double)((i+0.5)/N);

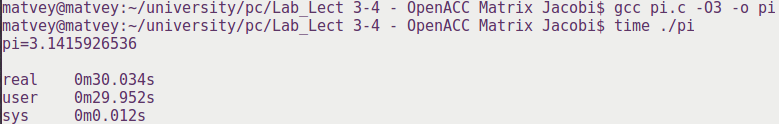
pi +=4.0/(1.0+t\*t);

}

printf("pi=%11.10f\n",pi/N);

return 0;

}



#include <stdio.h>

#include <omp.h>

#include <math.h>

#include <string.h>

int main(int argc, char\*\* argv)

{

int n = 4096;

int m = 4096;

int iter\_max = 1000;

const float pi = 2.0f \* asinf(1.0f);

const float tol = 1.0e-5f;

float error = 1.0f;

float \*restrict A = (float\*)malloc(sizeof(float)\*n\*m);

float \*restrict Anew = (float\*)malloc(sizeof(float)\*n\*m);

float \*restrict y0 = (float\*)malloc(sizeof(float)\*n);

memset(A, 0, n \* m \* sizeof(float));

// set boundary conditions

for (int i = 0; i < m; i++)

{

A[0\*m+i] = 0.f;

A[(n-1)\*m+i] = 0.f;

}

for (int j = 0; j < n; j++)

{

y0[j] = sinf(pi \* j / (n-1));

A[j\*m+0] = y0[j];

A[j\*m+m-1] = y0[j]\*expf(-pi);

}

printf("Jacobi relaxation Calculation: %d x %d mesh\n", n, m);

int iter = 0;

#pragma omp parallel for shared(Anew)

for (int i = 1; i < m; i++)

{

Anew[0\*m+i] = 0.f;

Anew[(n-1)\*m+i] = 0.f;

}

#pragma omp parallel for shared(Anew)

for (int j = 1; j < n; j++)

{

Anew[j\*m+0] = y0[j];

Anew[j\*m+m-1] = y0[j]\*expf(-pi);

}

while ( error > tol && iter < iter\_max ) {

error = 0.f;

#pragma acc kernels

{

#pragma acc loop independent

for( int j = 1; j < n-1; j++) {

for( int i = 1; i < m-1; i++ ) {

Anew[j\*m+i] = 0.25f \* ( A[j\*m+i+1] + A[j\*m+i-1]

+ A[(j-1)\*m+i] + A[(j+1)\*m+i]);

error = fmaxf( error, fabsf(Anew[j\*m+i]-A[j\*m+i]));

}

}

#pragma acc loop independent

for( int j = 1; j < n-1; j++) {

for( int i = 1; i < m-1; i++ ) {

A[j\*m+i] = Anew[j\*m+i];

}

}

}

if(iter % 100 == 0) printf("%5d, %0.6f\n", iter, error);

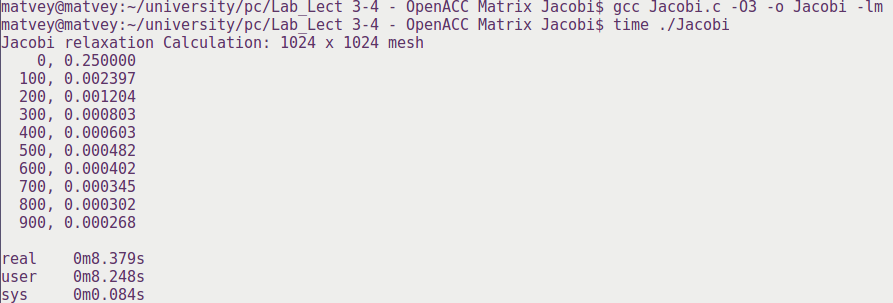
iter++;

}

double runtime = GetTimer();

printf(" total: %f s\n", runtime / 1000.f);

}



**mtx.c**

#include <stdio.h>

#include <stdlib.h>

#include <omp.h>

double \*\*dmatrix(int, int);

void free\_dmatrix(double \*\*);

int main(int argc, char \*argv[]) {

double \*\*a, \*\*b, \*\*c, sum;

long i, j, k, N;

double t\_start, t\_end;

N = 1500;

a = dmatrix(N, N);

b = dmatrix(N, N);

c = dmatrix(N, N);

for (i=0; i < N; i++)

for (j=0; j < N; j++) {

a[i][j] = (double)(i+j);

b[i][j] = (double)(i-j);

}

t\_start = omp\_get\_wtime();

//#pragma omp parallel for shared(a, b, c, N) private(i, j, k, sum)

#pragma acc data copyin(a[0:N][0:N],b[0:N][0:N]) copy(c[0:N][0:N])

{

# pragma acc region

{

#pragma acc loop independent vector(16)

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

#pragma acc loop independent vector(16)

for (j = 0; j < N; j++) {

for (k = 0; k < N; k++) {

sum += a[i][k] \* b[k][j];

}

c[i][j] = sum;

}

}

}

}

t\_end = omp\_get\_wtime();

printf("%10ld %20.15lf\n", N, t\_end - t\_start);

free\_dmatrix(c);

free\_dmatrix(b);

free\_dmatrix(a);

return 0;

}

// allocates a double matrix with range [0..m-1][0..n-1]

double \*\*dmatrix(int m, int n) {

double \*\*a;

int i, j;

a = (double \*\*) malloc(m \* sizeof(double \*));

a[0] = (double \*) malloc(m\*n \* sizeof(double ));

for (i=1; i < m; i++)

a[i] = a[i-1] + n;

return a;

}

// frees a matrix allocated with dmatrix

void free\_dmatrix(double \*\*a) {

free(a[0]);

free(a);

}

