ECE445 Παράλληλοι και Δικτυακοί Υπολογισμοί
Χειμερινό Εξάμηνο 2022-2023
Εργασία 2

Ομάδα φοιτητών:
Ιωάννης Ρούμπος - 2980
Γεράσιμος Αγοράς - 2947

Άσκηση 1

a) _____________ Σειριακός αλγόριθμος_____________

X: Input array
Y: Output array
n: length of array
ω: n-οστή ρίζα της μονάδας

procedure fft(x, y, n, ω)
if n=1 **then**
    y[0] = x[0]
else
    for k=0 **to** $\frac{n}{2} - 1$
        p[k] = x[2k]
        s[k] = x[2k+1]
    end
    fft(p, q, $\frac{n}{2}$, ω^2)
    fft(s, t, $\frac{n}{2}$, ω^2)
    for k=0 **to** n-1
        y[k] = q[k mod $(\frac{n}{2})$] + ω^kt[k mod $(\frac{n}{2})$]
    end
end

Υπολογισμός κόστους:

$$\mathbf{T(n)} = \sum_{n=1}^{logn} \sum_{j=0}^{2^{s-1}} \frac{n}{2^s} = \sum_{n=1}^{logn} \frac{n}{2^s} * 2^{s-1} = \sum_{n=1}^{logn} \frac{n}{2} = \mathbf{O(nlogn)}$$

b) ___________ Παράλληλος αλγόριθμος___________

X: Input array
Y: Output array
n: length of array
ω: n-οστή ρίζα της μονάδας

procedure fft(x_myID, y_myID, n)
r = log(n)
MPI_COMM_RANK(&myID)
R_myID = x_myID

```
for m=0 to r-1
    S_myID = R_myID
    j = (b0 ... bm-1, 0, bm+1 ... br-1)
    k = (b0 ... bm-1, 1, bm+1 ... br-1)
    if myID = j
        MPI_SEND(SJ, k)
        MPI_RECV(SK, k)
    end
    if myID = k
        MPI_SEND(SJ, j)
        MPI_RECV(SK, j)
    end
    R_myID = SJ + SK x ω^(bm, bm-1 ... b0, 0 ... 0)
    MPI_BARRIER()
end
y = R_myID
otherID = (br-1 ... b2, b1, b0)
if myID ≠ otherID
    if myID < otherID
        MPI_SEND(y, otherID)
        MPI_SEND(y_myID, otherID)
    end else
        MPI_SEND(y_myID, otherID)
        MPI_SEND(y, otherID)
    end
end
```

Υπολογισμός κόστους:

$$T(n) = (t_c + t_s + t_w) \log(n)$$

Όπου:

t_c : κόστος πολλαπλασιασμού και πρόσθεσης
t_s: κόστος εκκίνησης των διεργασιών
t_w: κόστος μεταφοράς ανά λέξη

Για τον παράλληλο χρόνο εκτέλεσης έχουμε:

$T(p) = t_c*n\log(n)/p + t_s*\log(p) + t_w*n\log(p)/p = O(n\log(n))$ για $p \le n$

Καθώς αναλυτικότερα έχουμε:

- nlog(n) πράξεις οι οποίες χωρίζονται στις p διεργασίες, απ' όπου και προκύπτει ο παράγοντας του t_c
- το setup time είναι log(p) από τον αρχικό τύπο, λόγω του Divide And Conquer χαρακτήρα του αλγορίθμου
- ο παράγοντας του tw είναι n/p επαναπροσδιορισμοί των συνιστωσών για κάθε διεργασία για τα log(p) στάδια

c)

<u>Χρονοβελτίωση</u>: S = t_c*n*log(n)/$T_{(b)}$(p) = $\frac{p*n*\log(n)}{n*\log(n)+\frac{ts}{tc}*p*\log(p)+\frac{tw}{tc}*n*\log(p)}$

<u>Απόδοση</u>: E = $T_{(b)}(1)/T_{(b)}(p)$ = $\frac{1}{1+\frac{ts*p*\log(p)}{tc*n*\log(n)}+\frac{tw*\log(p)}{tc*\log(n)}}$

d)

$$dS/dp = \frac{\left(\frac{p*n*\log(n)}{n*\log(n)+\frac{ts}{tc}*p*\log(p)+\frac{tw}{tc}*n*\log(p)}\right)}{dp} =$$

$$\frac{n*\log(n)\left(n*\log(n)+\frac{ts}{tc}*p*\log(p)+\frac{tw}{tc}*n*\log(p)\right)-p*n*\log(n)\left(\frac{ts}{tc}*\log(p)+\frac{ts}{tc}+\frac{tw}{tc}*\frac{n}{p}\right)}{\left(n*\log(n)+\frac{ts}{tc}*p*\log(p)+\frac{tw}{tc}*n*\log(p)\right)^2} = 0$$

$n*\log(n)\left(n*\log(n)+\frac{ts}{tc}*p*\log(p)+\frac{tw}{tc}*n*\log(p)\right)-p*n*\log(n)\left(\frac{ts}{tc}*\log(p)+\frac{ts}{tc}+\frac{tw}{tc}*\frac{n}{p}\right) = 0 \Leftrightarrow$

(έστω n=256) $2048\left(2048+\frac{ts}{tc}*p*\log(p)+\frac{tw}{tc}*256*\log(p)\right)-p*2048\left(\frac{ts}{tc}*\log(p)+\frac{ts}{tc}+\frac{tw}{tc}*\frac{256}{p}\right) = 0 \Leftrightarrow$

$256*\frac{tw}{tc}*\log(p) = -2048+p*\frac{ts}{tc}+256 \Leftrightarrow \frac{tw}{tc}*\log(p) = -1792+p*\frac{1}{256}*\frac{ts}{tc}$

e)

$$dE/dp = \frac{\left(\frac{1}{1+\frac{ts*p*\log(p)}{tc*n*\log(n)}+\frac{tw*\log(p)}{tc*\log(n)}}\right)}{dp} = \frac{\frac{ts*\log(p)}{tc*n*\log(n)}+\frac{ts}{tc*n*\log(n)}+\frac{1}{p}*\frac{tw}{tc*\log(n)}}{\left(1+\frac{ts*p*\log(p)}{tc*n*\log(n)}+\frac{tw*\log(p)}{tc*\log(n)}\right)^2} = 0$$

$\frac{ts*\log(p)}{tc*n*\log(n)}+\frac{ts}{tc*n*\log(n)}+\frac{1}{p}*\frac{tw}{tc*\log(n)} = 0 \Leftrightarrow$ (έστω n=256) $\frac{ts*\log(p)}{tc*2048}+\frac{ts}{tc*2048}+\frac{1}{p}*\frac{tw}{tc*8} = 0$

Άσκηση 2

Εκτελούμε τον σειριακό αλγόριθμο του FFT, ο οποίος είναι βασισμένος στον Cooley-Tukey αλγόριθμο που φαίνεται στο αρχείο fft_serial.c με μεγέθη 2^{10}... 2^{20} και οι χρόνοι σε δευτερόλεπτα που παίρνουμε είναι:

K = 10	0.000604976	**K = 16**	0.0286929
K = 11	0.00123422	**K = 17**	0.0574807
K = 12	0.00292676	**K = 18**	0.121656
K = 13	0.00593098	**K = 19**	0.260288
K = 14	0.00644857	**K = 20**	0.584439
K = 15	0.0134876		

Άσκηση 3

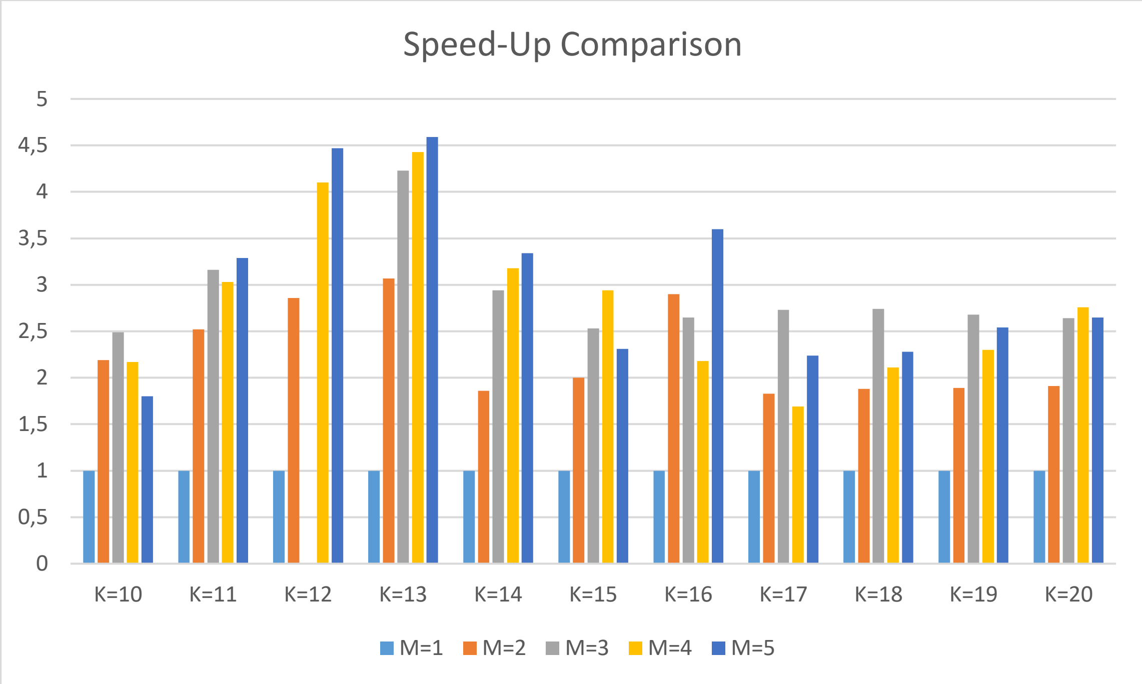
Προσθέτοντας OpenMP στον αλγόριθμο που χρησιμοποιήσαμε στην Άσκηση 2 (fft_parallel.c), και με μεγέθη τα ίδια που χρησιμοποιήσαμε παραπάνω παρατηρούμε ότι για αριθμό threads m = 6 παρουσιάζεται overhead στο μέγεθος k=10 (size = 2^{10}). Αυτό μας οδηγεί στο να πάρουμε μετρήσεις μέχρι και για αριθμό threads ίσο με 5.

Άσκηση 4

Οι μετρήσεις που παίρνουμε για τους διάφορους εκθέτης του μεγέθους k και τα διάφορα πλήθη νημάτων m φαίνονται στον παρακάτω πίνακα. Το speedup που φαίνεται σε κάθε περίπτωση είναι σε σύγκριση με το αντίστοιχο μέγεθος για την εκτέλεση σε 1 νήμα.

K vs M	1	2	3	4	5
10	0.000604976	0.000275948 (x2.19)	0.000242511 (x2.49)	0.00027853 (x2.17)	0.000335888 (x1.8)
11	0.00123422	0.00049004 (x2.52)	0.000390815 (x3.16)	0.000407605 (x3.03)	0.000375046 (x3.29)
12	0.00292676	0.00102307 (x2.86)	0.000719739 (x4.07)	0.000712662 (x4.1)	0.000654538 (x4.47)
13	0.00593098	0.00193173 (x3.07)	0.00140042 (x4.23)	0.00133922 (x4.43)	0.00129183 (x4.59)
14	0.00644857	0.00346731 (x1.86)	0.00218747 (x2.94)	0.00202571 (x3.18)	0.00192914 (x3.34)
15	0.0134876	0.00673193 (x2)	0.00532629 (x2.53)	0.00458149 (x2.94)	0.00582581 (x2.31)
16	0.0286929	0.0150378 (x1.9)	0.0108064 (x2.65)	0.0131791 (x2.18)	0.00795992 (x3.6)
17	0.0574807	0.0313205 (x1.83)	0.0210725 (x2.73)	0.0340494 (x1.69)	0.0256495 (x2.24)
18	0.121656	0.0646525 (x1.88)	0.0443356 (x2.74)	0.0575643 (x2.11)	0.0533176 (x2.28)
19	0.260288	0.137451 (x1.89)	0.0970999 (x2.68)	0.112899 (x2.3)	0.10246 (x2.54)
20	0.584439	0.305693 (x1.91)	0.221649 (x2.64)	0.211489 (x2.76)	0.220866 (x2.65)

Το καλύτερο speedup που παρατηρούμε είναι αυτό για μέγεθος = 2^{13} και threads = 5, με βελτίωση της τάξης του **459%**.

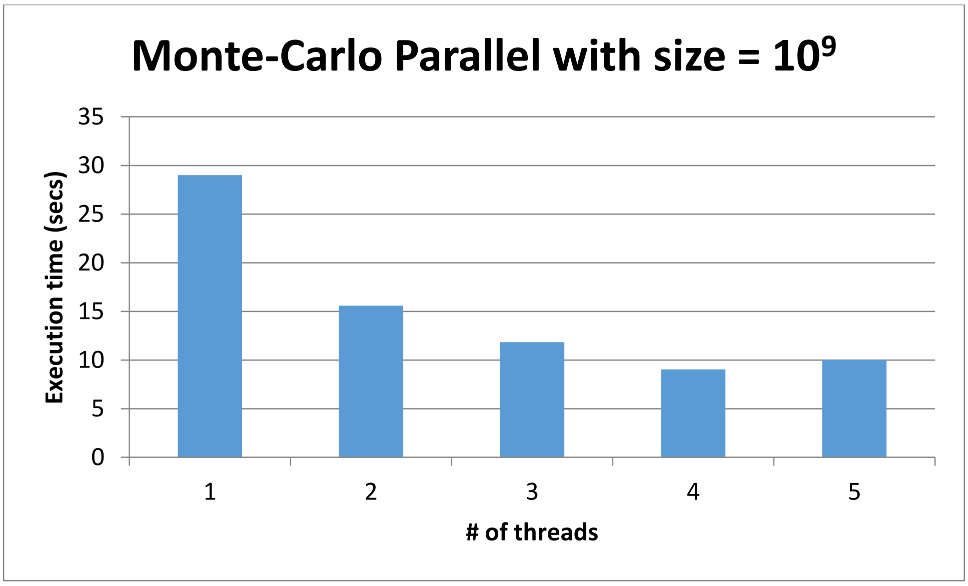


(Οι μετρήσεις αυτές βρίσκονται αναλυτικά στο excel εντός του zip, sheet "ex4")

Εκτελούμε τον σειριακό FFTW για μέγεθος = 2^{13} και ο χρόνος που παίρνουμε είναι ίσος με 0.000721484, καλύτερος δηλαδή από την καλύτερη υλοποίηση μας (0.00129183). Αυτό φανερώνει πως ο παράλληλος αλγόριθμος του FFT χρήζει μεγαλύτερης βελτίωσης.
(Για το compile του fftw.c, χρησιμοποιούμε τα flags: **gcc -Wall -g fftw.c -o fftw -lfftw3 -lm**)

Άσκηση 5

Παρακάτω έχουμε τον χρόνο εκτέλεσης για διάφορους αριθμούς threads στον αλγόριθμο Monte-Carlo. Παρατηρούμε ότι για το ενδεικτικό μέγεθος = 10^9 έχουμε speed-up μέχρι και για αριθμό threads ίσο με 4, ενώ όταν πάμε στα 5 εμφανίζεται overhead και το execution time χειροτερεύει.



Στο αρχείο excel που συμπεριλαμβάνεται στο zip υπάρχουν χρόνοι και για άλλα μεγέθη (sheet “ex5”).

ΠΑΡΑΡΤΗΜΑ

Περιγραφή μηχανήματος

- Αριθμός πυρήνων: 8
- Μεγέθη κρυφής μνήμης: L1d cache: 128KiB
  L1i cache: 128KiB
  L2 cache: 1MiB
  L3 cache: 6MiB

Κώδικας Άσκησης 2 (π.χ.)

Ακολουθεί ο κώδικας για το ερώτημα 2. Όνομα Αρχείου «fft_serial.c»

```
/*********************************
 * Ergasia 2 – Askhsh 2
 * Roumpos Ioannis  - 2980
 * Agoras Gerasimos - 2947
 *********************************/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>
#include <string.h>
#include <time.h>
#include <omp.h>

#define pi 3.14159265358979323846

unsigned int bitReverse(unsigned int x, int size)
{
    int n = 0;
    for (int i = 0; i < log2(size); i++)
    {
        n <<= 1;
        n |= (x & 1);
        x >>= 1;
    }
    return n;
}

void fft(complex double* input, complex double* output, int
size)
{
    int k,j,i;
```

```
    complex double expTable[size/2];

    // bit reversal of the given array
    for (i = 0; i < size; ++i) {
        int rev = bitReverse(i, size);
        output[i] = input[rev];
    }

    // Trigonometric Table
    for(k = 0; k < size / 2; k++){
        expTable[k] = cexp(-2*pi*k/size*I);
    }

    int n,halfsize,tablestep;
    double complex temp;

    // Cooley-Tukey decimation-in-time radix-2 FFT
    for(n = 2; n <= size; n *= 2){
        halfsize = n/2;
        tablestep = size / n;

        for(i = 0; i < size; i += n){
            for(j = i, k = 0; j < i + halfsize; j++, k +=
tablestep){
                temp = output[j + halfsize] * expTable[k];
                output[j + halfsize] = output[j] - temp;
                output[j] += temp;
            }
        }

        if(n == size)   // Prevent overflow in size *= 2
            break;
    }
}



int main(int argc,char* argv[]){
    struct timespec  tv1, tv2;

    if(argc != 2){
        printf("Give correct number of arguments!!");
        return -1;
    }

    int n = atoi(argv[1]);
    if((ceil(log2(n)) != floor(log2(n)))){
        printf("Size must be a power of 2!!");
        return -1;
    }

    double complex* input = malloc(n*sizeof(double complex));
    if (input == NULL){
        printf("Error in malloc!");
        return -1;
    }
```

```
    double complex* output = malloc(n*sizeof(double complex));
    if (output == NULL){
        printf("Error in malloc!");
        return -1;
    }

    printf("Input array\n");
    for(int i=0; i<n; i++){
        input[i] = (i+1) + 0*I;
        printf("%lf + 
%lf*i\n",creal(input[i]),cimag(input[i]));
    }

    clock_gettime(CLOCK_MONOTONIC_RAW, &tv1);
    fft(input,output,n);
    clock_gettime(CLOCK_MONOTONIC_RAW, &tv2);


    printf("\nOutput array\n");
    for(int i=0; i<n; i++){
        printf("%lf + 
%lf*i\n",creal(output[i]),cimag(output[i]));
    }

    printf ("Total time = %10g seconds\n",
                 (double) (tv2.tv_nsec - tv1.tv_nsec) / 
1000000000.0 +
                 (double) (tv2.tv_sec - tv1.tv_sec));

    free(input);
    free(output);
    return 0;
}
```

Κώδικας Άσκησης 3 (π.χ.)

Ακολουθεί ο κώδικας για το ερώτημα 3. Όνομα Αρχείου «fft_parallel.c»

```
/*********************************
 * Ergasia 2 – Askhsh 3
 * Roumpos Ioannis  - 2980
 * Agoras Gerasimos - 2947
 *********************************/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>
#include <string.h>
#include <time.h>
#include <omp.h>
```

```
#define pi 3.14159265358979323846

unsigned int bitReverse(unsigned int x, int size)
{
    int n = 0;
    for (int i = 0; i < log2(size); i++)
    {
        n <<= 1;
        n |= (x & 1);
        x >>= 1;
    }
    return n;
}

void fft(complex double* input, complex double* output, int
size)
{
    int k,j,i;
    complex double *expTable = malloc(size/2*sizeof(double
complex));
    if(expTable == NULL)
        return;

    // bit reversal of the given array
    #pragma omp parallel for shared(output) private(i)
    for (i = 0; i < size; ++i) {
        int rev = bitReverse(i, size);
        output[i] = input[rev];
    }

    #pragma omp parallel for shared(expTable,size) private(k)
    for(k = 0; k < size / 2; k++){
        expTable[k] = cexp(-2*pi*k/size*I);
    }

    int n,halfsize,tablestep;
    double complex temp;

    #pragma omp parallel
private(n,halfsize,tablestep,i,j,k,temp) shared(output)
    {
    for(n = 2; n <= size; n *= 2){
        halfsize = n/2;
        tablestep = size / n;

    #pragma omp for
        for(i = 0; i < size; i += n){
            for(j = i, k = 0; j < i + halfsize; j++, k +=
tablestep){
                temp = output[j + halfsize] * expTable[k];
                output[j + halfsize] = output[j] - temp;
                output[j] += temp;
            }
        }

        if(n == size)
```

```
            break;
    }
    }
    free(expTable);
}


int main(int argc,char* argv[]){
    struct timespec  tv1, tv2;

    if(argc != 2){
        printf("Give correct number of arguments!!");
        return -1;
    }

    int n = atoi(argv[1]);
    n = pow(2,n);

    double complex* input = malloc(n*sizeof(double complex));
    if (input == NULL){
        printf("Error in malloc!");
        return -1;
    }
    double complex* output = malloc(n*sizeof(double complex));
    if (output == NULL){
        printf("Error in malloc!");
        return -1;
    }

    /*printf("Input array\n");
    for(int i=0; i<n; i++){
        input[i] = (i+1) + 0*I;
        printf("%lf +
%lf*i\n",creal(input[i]),cimag(input[i]));
    }*/

    clock_gettime(CLOCK_MONOTONIC_RAW, &tv1);
    fft(input,output,n);
    clock_gettime(CLOCK_MONOTONIC_RAW, &tv2);


    /*printf("\nOutput array\n");
    for(int i=0; i<n; i++){
        printf("%lf +
%lf*i\n",creal(output[i]),cimag(output[i]));
    }*/
    printf("N = %.2lf ",log2(n));
    printf ("Total time = %10g seconds\n",
                (double) (tv2.tv_nsec - tv1.tv_nsec) /
1000000000.0 +
                (double) (tv2.tv_sec - tv1.tv_sec));

    free(input);
    free(output);
    return 0;
}
```

Κώδικας Άσκησης 4 (π.χ.)

Ακολουθεί ο κώδικας για το ερώτημα 4. Όνομα Αρχείου «fftw.c»

```
/********************************
 * Ergasia 2 – Askhsh 4
 * Roumpos Ioannis  - 2980
 * Agoras Gerasimos - 2947
 ********************************/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <fftw3.h>

int main(int argc,char* argv[]){
    struct timespec tv1,tv2;
    int N = atoi(argv[1]);
    N = pow(2,N);

    fftw_complex *in,*out;
    fftw_plan p;

    in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) *
N);
    for(int i=0; i < N; i++){
        in[i][0] = i+1;
        in[i][1] = 0;
    }

    out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) *
N);
    clock_gettime(CLOCK_MONOTONIC_RAW, &tv1);
    p = fftw_plan_dft_1d(N, in, out, FFTW_FORWARD,
FFTW_ESTIMATE);

    fftw_execute(p); /* repeat as needed */
    clock_gettime(CLOCK_MONOTONIC_RAW, &tv2);
    fftw_destroy_plan(p);

    printf ("Total time = %10g seconds\n",
                (double) (tv2.tv_nsec - tv1.tv_nsec) /
1000000000.0 +
                (double) (tv2.tv_sec - tv1.tv_sec));

    fftw_free(in); fftw_free(out);
    return 0;
}
```

Κώδικας Άσκησης 5 (π.χ.)

Ακολουθεί ο κώδικας για το ερώτημα 5. Όνομα Αρχείου «mc_parallel.c»

```
/*********************************
 * Ergasia 2 – Askhsh 5
 * Roumpos Ioannis  - 2980
 * Agoras Gerasimos - 2947
 *********************************/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>
#include <time.h>
inline double f(double x)
{
	return sin(cos(x));
}

// WolframAlpha: integral sin(cos(x)) from 0 to 1  =  0.738643
//                                       0.73864299803689018
//
	0.738642998036890183800090290585216041748020942244764851 8
714116299

int main(int argc, char *argv[])
{
	double a = 0.0;
	double b = 1.0;
	unsigned long n = 24e8;
	long tseed = time(0);

	if (argc == 2) {
		tseed = atol(argv[1]);
	}
	else if (argc == 3) {
		n = atol(argv[1]);
		tseed = atol(argv[2]);
	}

	const double h = (b-a)/n;
	const double ref = 0.73864299803689018;
	double res = 0;
	double t0, t1;
	unsigned long i;

    t0 = omp_get_wtime();

#pragma omp parallel
	{
	double local_res = 0;
	double xi;
```

```
    unsigned short buffer[3];
    buffer[0] = 0;
    buffer[1] = 0;
    buffer[2] = tseed+omp_get_thread_num();

#pragma omp for
    for (i = 0; i < n; i++) {
           xi = erand48(buffer);
        local_res += f(xi);
      }

#pragma omp atomic
      res += local_res;

}

    res *= h;
    t1 = omp_get_wtime();

      printf("Result=%.16f Error=%e Rel.Error=%e Time=%lf
seconds\n", res, fabs(res-ref), fabs(res-ref)/ref, t1-t0);

      return 0;
}
```