ECE445 Παράλληλοι και Δικτυακοί Υπολογισμοί
Χειμερινό Εξάμηνο 2022-2023
Εργασία 3

Ομάδα φοιτητών:
Ιωάννης Ρούμπος - 2980
Γεράσιμος Αγοράς - 2947

Άσκηση 1

- <u>Ερώτημα 1</u>

<u>COO Format Serial Algorithm:</u>

```
for (int i=0; i<nz; ++i) {
        y[row[i]] += val[i]*x[col[i]];
}
```

Υπολογισμός κόστους:

T(n) = $\boldsymbol{O}(\mathbf{nz})$, καθώς στο loop ξεκινάμε από το πρώτο στοιχείο και καταλήγουμε στο nz-1, οπότε συνολικά nz επαναλήψεις.

<u>CSR Format Serial Algorithm:</u>

```
for (int i = 0; i < n; ++i) {
        for (int j = i_ptr[i]; j < i_ptr[i+1];  ++j)
                y[i] += val[j]*x[col[j]];
}
```

Υπολογισμός κόστους:

Το εξωτερικό loop μας έχει n επαναλήψεις. Κάθε φορά στο εσωτερικό loop αρχίζουμε από το i_ptr(i) και φτάνουμε έως το i_ptr(i+1) – 1. Η χειρότερη περίπτωση που μπορεί να αντιμετωπίσουμε σ’ αυτό το loop είναι όλα τα στοιχεία NZ να βρίσκονται στην ίδια γραμμή, όπου κ’ έχουμε $\boldsymbol{\Theta}(\mathbf{n} * \mathbf{nz})$ συνολικά.

- <u>Ερώτημα 2</u>

<u>COO Format Parallel Algorithm:</u> (Υποθέτουμε ότι ο αριθμός των threads είναι ίσος με τον αριθμό των blocks)

```
for (int i=0; i < n; ++i) {
        if(id % i == 0) {
                if(id*(N/p) ≤ row[i] ≤ ((id+1)*(N/p) - 1))
                        y[row[i]] += val[i]*x[col[i]];
        }
}
```

Υπολογισμός κόστους:

T(n) = $\sum_{i=1}^{p} \sum_{j=1}^{n} \mathrm{a/p} = \sum_{i=1}^{p} \mathrm{n} * \mathrm{a/p}$ = n*a/p = **O(n/p)** , όπου a το σταθερό κόστος της πράξης y[row[i]] += val[i]*x[col[i]]

<u>Χρονοβελτίωση</u>: S = T(1)/T(p) = p

<u>Απόδοση</u>: E = 1/T(p) = $\frac{p}{\mathrm{a*n}}$

<u>CSR Format Parallel Algorithm:</u>

```
for (int i = 0; i < n; ++i)
        if(id % i == 0) {
                if(id*(N/p) ≤ i ≤ ((id+1)*(N/p) - 1)) {
                        for (int j = i_ptr[i]; j < i_ptr[i+1];  ++j)
                                y[i] += val[j]*x[col[j]];
                }
```

Υπολογισμός κόστους:

T(n) = $\sum_{i=1}^{p} \sum_{j=1}^{n} \sum_{k=i\_ptr_j}^{i\_ptr_{j+1}} \mathrm{a/p} = \sum_{i=1}^{p} \sum_{j=1}^{n} \mathrm{nz} * \mathrm{a/p} = \sum_{i=1}^{p} \mathrm{n} * \mathrm{nz} * \mathrm{a/p}$ = n*nz*a/p = **O(n*nz/p)**, όπου a το σταθερό κόστος της πράξης y[i] += val[j]*x[col[j]]

Χρονοβελτίωση: S = T(1)/T(p) = p
Απόδοση: E = 1/T(p) = $\frac{p}{a*n*nz}$

- Ερώτημα 3

COO Format Parallel Algorithm: (Υποθέτουμε ότι ο αριθμός των threads είναι ίσος με τον αριθμό των γραμμών)

```
for (int i=0; i < nz; ++i) {
    if(id % i == 0) {
        y[row[i]] += val[i]*x[col[i]];
    }
}
```

Υπολογισμός κόστους:

T(n) = $\sum_{i=1}^{nz} a/p$ = nz*a/p = **O(nz/p)** , όπου a το σταθερό κόστος της πράξης y[row[i]] += val[i]*x[col[i]]

CSR Format Parallel Algorithm:

```
for (int i = 0; i < n; ++i) {
    if(id % i == 0) {
        if(id*(N/p) ≤ i ≤ ((id+1)*(N/p) - 1)) {
            for (int j = i_ptr[i]; j < i_ptr[i+1];  ++j)
                y[i] += val[j]*x[col[j]];
        }
```

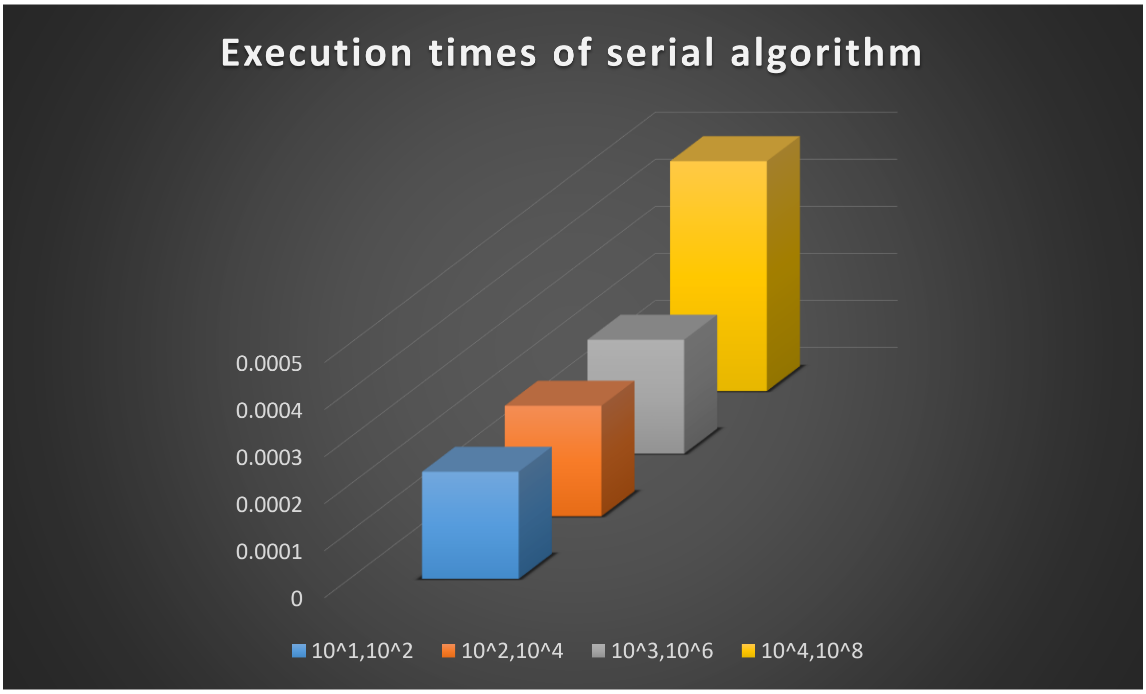
Υπολογισμός κόστους:

T(n) = $\sum_{i=1}^{p}\sum_{j=1}^{nz}\sum_{k=i\_ptr_j}^{i\_ptr_{j+1}} a/p = \sum_{i=1}^{p}\sum_{j=1}^{n} nz*a/p = \sum_{i=1}^{p} n*nz*a/p =$ n*nz*a = **O(n*nz)** , όπου a το σταθερό κόστος της πράξης y[i] += val[j]*x[col[j]]

Άσκηση 2

Εκτελούμε τον σειριακό αλγόριθμο για πολλαπλασιασμό CSR-sparse πίνακα με διάνυσμα σύμφωνα με τον αλγόριθμό που φαίνεται στην άσκηση 1 και είναι υλοποιημένος στο αρχείο serial_mul.c, και οι χρόνοι που παίρνουμε είναι οι εξής:

Πλήθος NZ	Μέγεθος πίνακα	Χρόνος εκτέλεσης
10	10^2	**0.000005866**
10^2	10^4	**0.000010209**
10^3	10^6	**0.000015473**
10^4	10^8	**0.000146976**

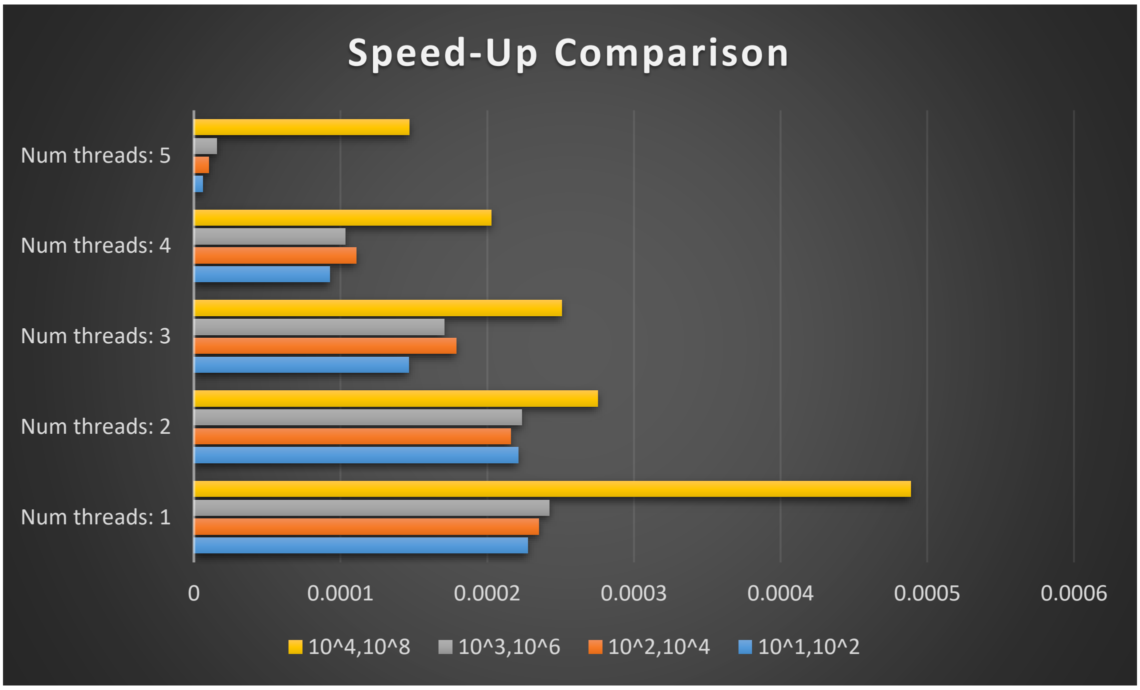


Άσκηση 3

Οι μετρήσεις που παίρνουμε για τα διάφορα μεγέθη του πίνακα και τα τον αριθμό των μη-μηδενικών κελιών φαίνονται παρακάτω για αριθμό νημάτων ίσο με 1,2,3,4 και 5 (parallel_mul.c). Τα speedup φαίνονται σε παρένθεση.

(NZ,N) vs p	1	2	3	4	5
(10^1,10^2)	0.000227618	0.00022103 (x1.03)	0.000146586 (x1.55)	0.000092505 (x2.46)	0.000005866 (x38.8)
(10^2,10^4)	0.000234993	0.000216179 (x1.08)	0.000178833 (x1.31)	0.000110756 (x2.12)	0.000010209 (x23.01)
(10^3,10^6)	0.000242295	0.000223502 (x1.08)	0.000170607 (x1.42)	0.000103059 (x2.35)	0.000015473 (x15.66)
(10^4,10^8)	0.000488788	0.000275242 (x1.77)	0.000250704 (x1.95)	0.000202796 (x2.41)	0.000146976 (x3.33)

Το καλύτερο speedup που παρατηρούμε είναι αυτό για μέγεθος = 10^2 και threads = 5, με βελτίωση της τάξης του **3880%**.



Παρατηρούμε πως για p = 5 υπάρχει ισχυρό speedup για όλα τα μεγέθη που έχουμε ελέγξει.

Άσκηση 4

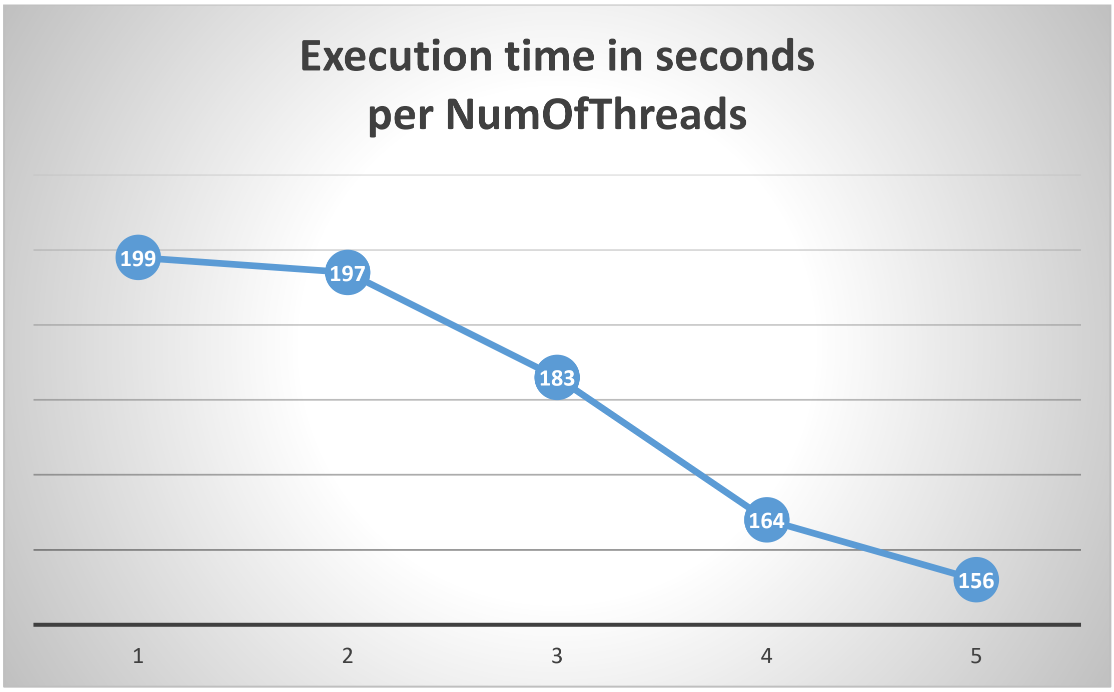
Εκτελούμε τη μέθοδο jacobi στο πίνακα που μας δίνεται σε μορφή CSR, αποθηκεύοντας τον όπως φαίνεται εντός του αρχείου “jacobi_openmp.c”.

Οι χρόνοι που φαίνονται παρακάτω είναι για μέγεθος πίνακα $n = 10^6$ και αριθμό επαναλήψεων ίσο με m = 1000, ώστε να φαίνεται η επίδραση του multithreading στην εκτέλεση και για μεγαλύτερη σύγκλιση στο αποτέλεσμα.

Σε κάθε επανάληψη εκτυπώνονται τα κατάλληλα σχόλια (iter = *, residual = *, difference = *), καθώς και στο τέλος ο χρόνος εκτέλεσης που εμφανίζεται παρακάτω στο διάγραμμα.

Ο πίνακας με τους χρόνους εκτέλεσης ανά multithreading execution φαίνεται παρακάτω.

NumofThreads	1	2	3	4	5
Computations Time (secs)	199	197	183	164	156



Στο αρχείο excel που συμπεριλαμβάνεται στο zip υπάρχουν χρόνοι (sheet “ex4”).

Παρατηρούμε ότι η επίδραση του multithreading στον κώδικα είναι ισχυρή, καθώς ο χρόνος που έχουμε για αριθμό νημάτων ίσο με 5 σε σχέση με την σειριακή υλοποίηση είναι <u>21.6%</u> χαμηλότερος.

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 «serial_mul.c»

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

int main(int argc,char**argv){
    if(argc != 3){
        printf("Give correct number of arguments!\n");
        printf("Correct form is: ./<name> <non zero elements>
<size of vector>\n");
        return -1;
    }

    struct timespec  tv1, tv2;
    int NZ = atoi(argv[1]);
    NZ = pow(10,NZ);
    int size_of_vector = atoi(argv[2]);
    size_of_vector = pow(10,size_of_vector);


    int *sparse_A = malloc(sizeof(int) * NZ);
    int *i_ptr = malloc(sizeof(int) * (size_of_vector + 1));
    int *j_index = malloc(sizeof(int) * size_of_vector);
    int *y = malloc(sizeof(int) * size_of_vector);
    int *x = malloc(sizeof(int) * size_of_vector);
    i_ptr[0] = 0;

    /*Initialize arrays*/
    for(int i=0; i < NZ; i++){
        sparse_A[i] = i + 1;
        j_index[i] = i%4;
    }
```

```
    /*Row offset*/
    for(int i=1; i < size_of_vector; i++){
        i_ptr[i] = i_ptr[i-1] +  NZ/size_of_vector;
    }
    i_ptr[size_of_vector] = NZ;

    /*Vector initialization*/
    for(int i=0; i < size_of_vector; i++){
        x[i] = i + 1;
    }

    clock_gettime(CLOCK_MONOTONIC_RAW, &tv1);
    /*Calculate sparse matrix vector multiplication*/
    for (int i=0; i < size_of_vector; i++){
        y[i] = 0;
        for (int j = i_ptr[i]; j < i_ptr[i+1]; j++){
            y[i] += sparse_A[j] * x[j_index[j]];
        }
    }
    clock_gettime(CLOCK_MONOTONIC_RAW, &tv2);
    printf ("Total time = %10g seconds\n",
                 (double) (tv2.tv_nsec - tv1.tv_nsec) /
1000000000.0 +
                 (double) (tv2.tv_sec - tv1.tv_sec));

    return 0;

}
```

Κώδικας Άσκησης 3 (π.χ.)

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

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

int main(int argc,char**argv){
    if(argc != 3){
        printf("Give correct number of arguments!\n");
        printf("Correct form is: ./<name> <non zero elements>
<size of vector>\n");
```

```
        return -1;
    }

     struct timespec  tv1, tv2;
    int NZ = atoi(argv[1]);
    NZ = pow(10,NZ);
    int size_of_vector = atoi(argv[2]);
    size_of_vector = pow(10,size_of_vector);


    int *sparse_A = malloc(sizeof(int) * NZ);
    int *i_ptr = malloc(sizeof(int) * (size_of_vector + 1));
    int *j_index = malloc(sizeof(int) * size_of_vector);
    int *y = malloc(sizeof(int) * size_of_vector);
    int *x = malloc(sizeof(int) * size_of_vector);
    i_ptr[0] = 0;

    /*Initialize arrays*/
    for(int i=0; i < NZ; i++){
        sparse_A[i] = i + 1;
        j_index[i] = i%4;
    }

    /*Row offset*/
    for(int i=1; i < size_of_vector; i++){
        i_ptr[i] = i_ptr[i-1] +  NZ/size_of_vector;
    }
    i_ptr[size_of_vector] = NZ;

    /*Vector initialization*/
    for(int i=0; i < size_of_vector; i++){
        x[i] = i + 1;
    }

    /*Start time*/
    clock_gettime(CLOCK_MONOTONIC_RAW, &tv1);
    /*Calculate sparse matrix vector multiplication*/
    #pragma omp parallel for schedule(guided)
    for (int i=0; i < size_of_vector; i++){
        y[i] = 0;
        for (int j = i_ptr[i]; j < i_ptr[i+1]; j++){
            y[i] += sparse_A[j] * x[j_index[j]];
        }
    }
    clock_gettime(CLOCK_MONOTONIC_RAW, &tv2);
    /*Stop time*/

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


    return 0;
}
```

Κώδικας Άσκησης 4 (π.χ.)

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

```
/*********************************
 * Ergasia 3 – Askhsh 4
 * Roumpos Ioannis  - 2980
 * Agoras Gerasimos - 2947
 *********************************/
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <math.h>
#include <time.h>

void jacobi(int n, int* IA, int* JA, double* A, double* b,
double* x, int max_iter) {
    int i, j, k;
    double sum, x_new, residual, difference;
    double* x_old = (double*) malloc(n * sizeof(double));
    double* Ax = (double*) malloc(n * sizeof(double));
    double* diff_mat = (double * ) malloc(n * sizeof(double));

      #pragma omp for schedule(dynamic) private(i)
    for (i = 0; i < n; i++) {
        x_old[i] = 0.0;
    }

    #pragma omp parallel private(i, j, k, sum, x_new)
    {
        for (i = 0; i < max_iter; i++) {
            #pragma omp for schedule(dynamic)
            for (j = 0; j < n; j++) {
                sum = b[j];
                for (k = IA[j]; k < IA[j+1]; k++) {
                    if (JA[k] != j) {
                        sum -= A[k] * x_old[JA[k]];
                    }
                }
                if(j == 0) {
                           x_new = sum / A[IA[j]];
                      }
                      else {
                           x_new = sum / A[IA[j]+1];
                      }
                x[j] = x_new;
            }
            #pragma omp barrier
            {
                // Compute residual
                      #pragma omp for schedule(dynamic)
                for (j = 0; j < n; j++) {
```

```
                    Ax[j] = b[j];
                    for (k = IA[j]; k < IA[j+1]; k++) {
                        Ax[j] -= A[k] * x[JA[k]];
                    }
                }
                residual = 0.0;
                #pragma omp for schedule(dynamic)
reduction(+:residual)
                for (j = 0; j < n; j++) {
                    residual += Ax[j] * Ax[j];
                }
                residual = sqrt(residual);

                     // Compute difference
                #pragma omp for schedule(dynamic)
                for(j = 0; j < n; j++){
                    diff_mat[j] = x_old[j] - x[j];
                }

                difference = 0.0;
                #pragma omp for schedule(dynamic)
reduction(+:difference)
                for (j = 0; j < n; j++) {
                    difference += diff_mat[j] * diff_mat[j];
                }
                difference = sqrt(difference);

                printf("Iter = %d, residual = %lf, difference
= %lf\n", i, residual, difference);

                #pragma omp for schedule(dynamic)
                for (j = 0; j < n; j++) {
                    x_old[j] = x[j];
                }
            }
        }
    }
    free(x_old);
    free(Ax);
    free(diff_mat);
}


int main() {
    int i;
    int n = 1000000;
    int max_iter = 1000;
    double* A_values = (double*) malloc((3*n-2)*
sizeof(double));
    double* b = (double*) malloc(n * sizeof(double));
    double* x = (double*) malloc(n * sizeof(double));
    int* i_ptr = (int*) malloc((n+1) * sizeof(int));
    int* j_index = (int*) malloc((3*n-2)* sizeof(int));
     struct timespec tv1,tv2;

     /*Init A Values*/
```

```
  A_values[0] = 2;
  A_values[1] = -1;
  A_values[3*n - 4] = -1;
  A_values[3*n - 3] = 2;

  /*Init i_ptr*/
  i_ptr[0] = 0;
  i_ptr[1] = 2;
  i_ptr[n] = 3*n - 2;
  //i_ptr[n] = 3*n - 1;

  /*Init j_index*/
  j_index[0] = 0;
  j_index[1] = 1;
  j_index[3*n - 4] = n-2;
  j_index[3*n - 3] = n-1;


// Initialize the matrix and right-hand side
# pragma omp parallel private ( i )
{
/*Init A values*/
#pragma omp for
for ( i = 2; i < 3*n - 4; i = i + 3){
    A_values[i] =  -1;
    A_values[i + 1] = 2;
    A_values[i + 2] = -1;
}

#pragma omp for
for( i = 2; i < n; i++){
    i_ptr[i] = 3*i - 1;
}

#pragma omp for
for( i = 2; i < 3*n - 4; i = i+3){
    j_index[i] = i/3;
    j_index[i + 1] = j_index[i] + 1;
    j_index[i + 2] = j_index[i] + 2;
}

/*
Set up the right hand side.
*/
# pragma omp for
    for ( i = 0; i < n; i++ )
    {
    b[i] = 0.0;
    }

    b[n-1] = ( double ) ( n + 1 );
/*
Initialize the solution estimate to 0.
Exact solution is (1,2,3,...,N).
*/
# pragma omp for
```

```
        for ( i = 0; i < n; i++ )
        {
        x[i] = 0.0;
        }

    }

    /*printf("A_vals: ");
    for( i =0; i < 3*n-2; i++){
    printf("%d ", (int)A_values[i]);
    }
    printf("\ni_ptr: ");
    for( i =0; i < n+1; i++){
    printf("%d ", i_ptr[i]);
    }
    printf("\nj_index: ");
    for( i =0; i < 3*n-2; i++){
    printf("%d ", j_index[i]);
    }
    printf("\nb: ");
    for( i =0; i < n; i++){
    printf("%f ", b[i]);
    }
    printf("\n");*/

    // Call the Jacobi method
    clock_gettime(CLOCK_MONOTONIC_RAW, &tv1);
    jacobi(n, i_ptr, j_index, A_values, b, x, max_iter);
     clock_gettime(CLOCK_MONOTONIC_RAW, &tv2);

    // Print the solution
    /*for (i = 0; i < n; i++) {
        printf("x[%d] = %f\n", i, x[i]);
    }*/

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

    free(A_values);
    free(b);
    free(x);
    free(i_ptr);
    free(j_index);
    return 0;
}
```