ECE445 Παράλληλοι και Δικτυακοί Υπολογισμοί
Χειμερινό Εξάμηνο 2022-2023
Εργασία 1

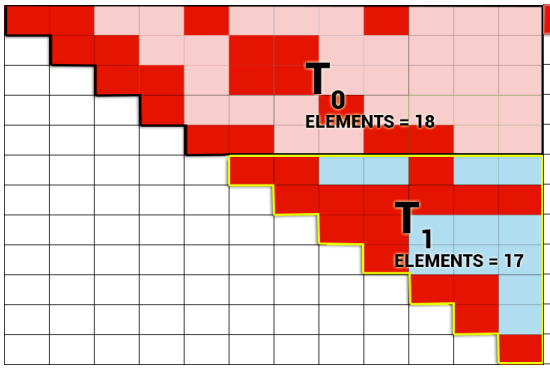
Ομάδα φοιτητών:
Ιωάννης Ρούμπος - 2980
Γεράσιμος Αγοράς - 2947

Άσκηση 1

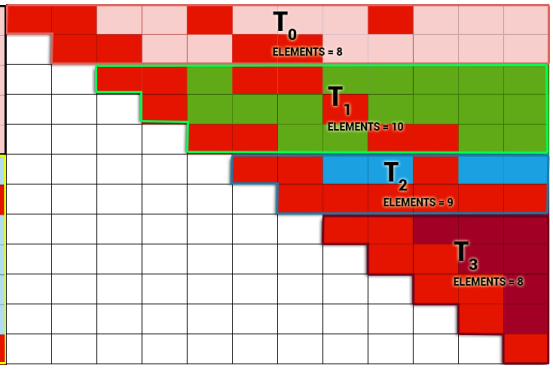
Τα δεδομένα που χρειάζονται και στις δύο περιπτώσεις (2 ή 4 επεξεργαστές) για τον καταμερισμό με τον προφανή τρόπο είναι και τα 12 στοιχεία του διανύσματος b, καθώς για αν υπολογιστεί το κάθε c[i,j] στοιχείο του τελικού διανύσματος χρησιμοποιούμε όλη την γραμμή i του A και όλο το διάνυσμα-στήλη b.

Για τον μη προφανή καταμερισμό:

- 2 επεξεργαστές: Παρατηρούμε ότι ο αρχικός πίνακας A είναι συμμετρικός ως προς την κύρια διαγώνιο του, οπότε μπορούμε να υπολογίσουμε είτε τον άνω, είτε τον κάτω τριγωνικό και να αξιοποιήσουμε δύο φορές τα ίδια δεδομένα του A (εικόνα 1).
- 4 επεξεργαστές: Χρησιμοποιώντας την ίδια λογική με αυτή των δύο επεξεργαστών, παίρνουμε τον άνω τριγωνικό πίνακα από τον A και τον χωρίζουμε σε τέσσερα blocks ίδιου αριθμού μη μηδενικών κελιών (εικόνα 2).



Εικόνα 1. Διαχωρισμός του άνω τριγωνικού πίνακα σε δύο threads



Εικόνα 2. Διαχωρισμός του άνω τριγωνικού πίνακα σε τέσσερα threads

Για τον καταμερισμό του πίνακα σε 2 blocks-στήλες:
Για τον υπολογισμό του κάθε κελιού του c, ο κάθε επεξεργαστής θα αποθηκεύει προσωρινά το άθροισμα των πολλαπλασιασμών μεταξύ της μισής γραμμής i και της μισής στήλης του b που του αναλογούν, και στο τέλος αυτής της διαδικασίας θα προσθέτει τα επιμέρους αποτελέσματα για τον υπολογισμό της τελικής τιμής του εκάστοτε κελιού.

Για τον καταμερισμό του πίνακα σε 4 blocks-στήλες:
Ακολουθώντας την ίδια λογική, πλέον θα έχουμε προσωρινά αποθηκευμένα 4 αποτελέσματα, τα οποία αθροίζοντας ‘τα, καταλήγουμε στην τελική τιμή ενός κελιού του c.

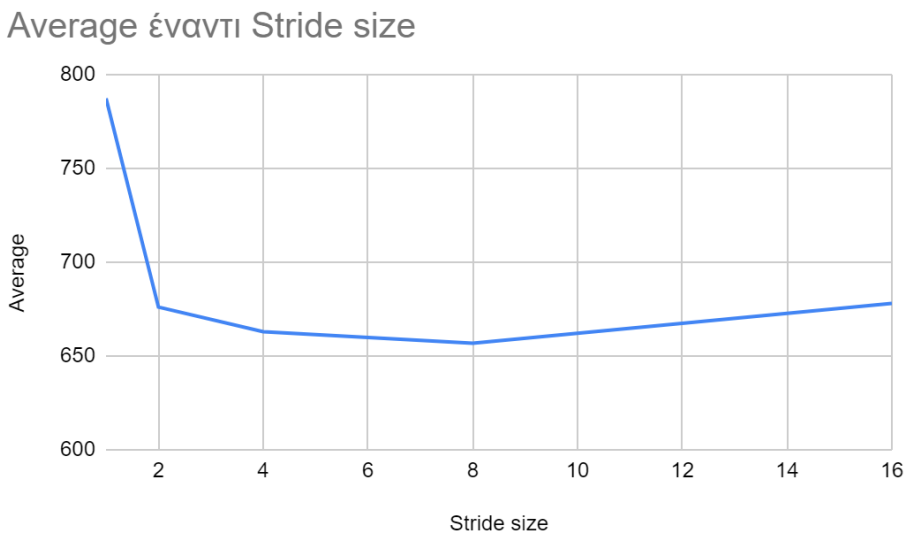
Για τις απαιτήσεις επικοινωνίας, κατά τον υπολογισμό των επιμέρους αποτελεσμάτων δεν υπάρχει εξάρτηση μεταξύ των επεξεργαστών, παρά μόνο κοινή χρήση του πίνακα b. Μοναδική επικοινωνία βρίσκουμε στο τέλος, όπου προστίθενται τα 4x12 αποτελέσματα των επεξεργαστών για να πάρει την τελική του μορφή το διάνυσμα c.

Άσκηση 2

Ο αριθμός των πράξεων στο πρόγραμμα είναι n προσθέσεις και n πολλαπλασιασμοί, όπου n το μέγεθος των διανυσμάτων.

Η βελτίωση που βλέπουμε στην αποδοτικότητα με την αύξηση του step οφείλεται στο caching του επεξεργαστή, καθώς σε κάθε επανάληψη του loop μειώνεται σημαντικά ο αριθμός των εντολών που αφορούν την διαχείριση του loop και οι αντίστοιχες επιβαρύνσεις.

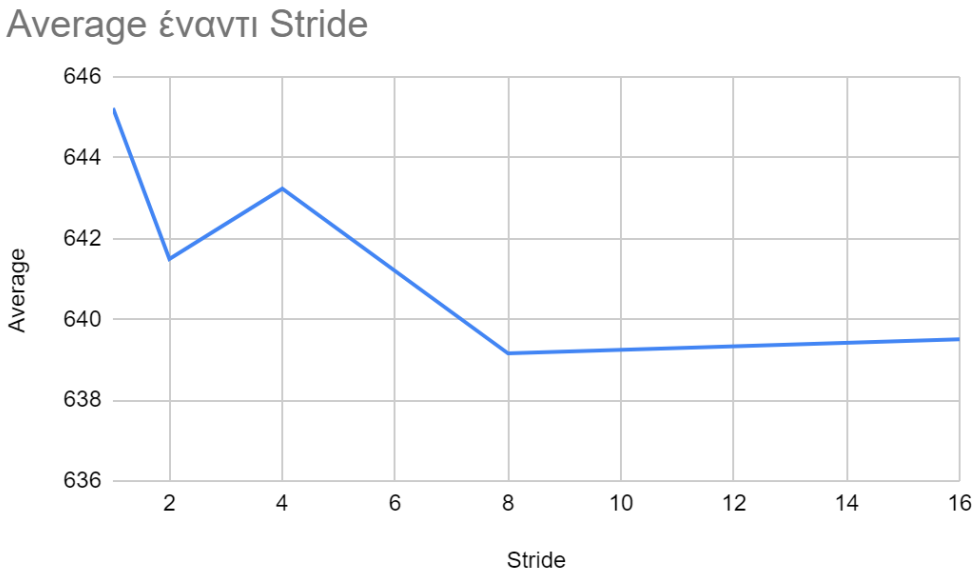
Από τα αποτελέσματα, παρατηρούμε ότι κρίσιμα k θεωρούνται τα 8 και 16, καθώς μέχρι το 8 υπάρχει βελτίωση στους χρόνους, όμως όταν χρησιμοποιούμε ως step το 16 οι χρόνοι χειροτερεύουν. Αυτό συμβαίνει λόγω των πολλαπλών προσπελάσεων στην μνήμη, κάτι το οποίο προσθέτει τέτοιο overhead που αναιρεί το loop unrolling.



Άσκηση 3

Τα FLOPS (Floating Point Operation Per Seconds) που έχουμε είναι $2n^3$.

Η βελτίωση της αποδοτικότητας οφείλεται στο loop unrolling κατά τον υπολογισμό των εσωτερικών βρόγχων.
Παρατηρούμε, επίσης, ότι για stride = 4 η απόδοση χειροτερεύει σε σχέση με stride = 2, κάτι το οποίο οφείλεται στις Cache μνήμες και τα cache misses τα οποία δημιουργούνται.

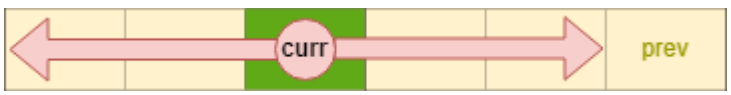


Άσκηση 4

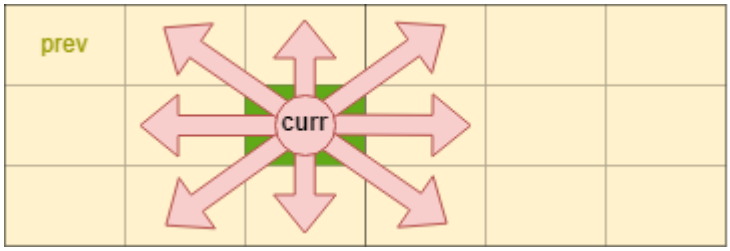
Περιγραφή μηχανήματος:

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

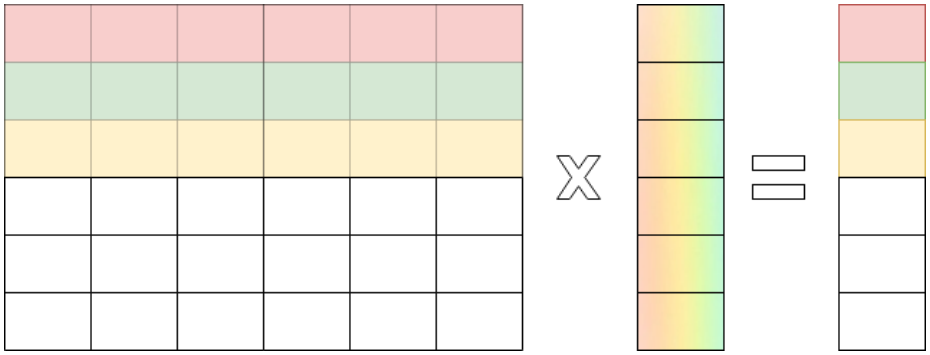
1. Stencil 1D
   - Ο υπολογισμός ενός κελιού του πίνακα curr (10000x1) βασίζεται στα γειτονικά κελιά (2 δεξιά, 2 αριστερά) και την τιμή του ίδιου που πήραμε στη προηγούμενη επανάληψη του αλγορίθμου.
   - Ο παραλληλισμός εφαρμόζεται στον υπολογισμό του κάθε κελιού, χωρίς να λαμβάνει υπόψη η κάθε διεργασία το που βρίσκονται οι υπόλοιπες, καθώς τα κελιά που υπολογίζονται στην ίδια επανάληψη του while είναι ανεξάρτητα μεταξύ τους.



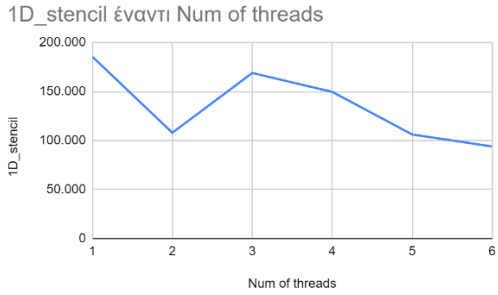
2. Stencil 2D
   - Ο υπολογισμός κελιού του πίνακα curr (3000x3000) βασίζεται στο 3x3 τετράγωνο με κέντρο το ίδιο το κελί, βάση των τιμών του πίνακα στη προηγούμενη επανάληψη του αλγορίθμου.
   - Όπως και στην προηγούμενη υλοποίηση, ο παραλληλισμός εφαρμόζεται μεταξύ των μη-εξαρτημένων κελιών της ίδια επανάληψης του while.

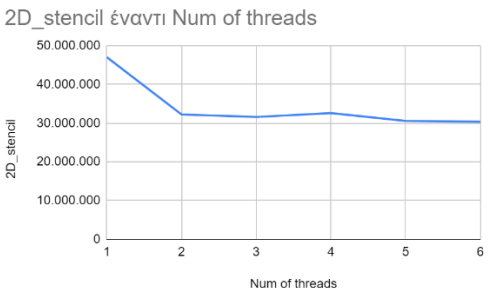


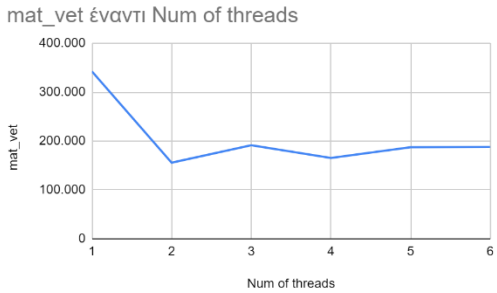
3. Γινόμενο πίνακα-διανύσματος
   - Ο αλγόριθμος είναι ο βασικός αλγόριθμος μεταξύ πίνακα και διανύσματος, ο οποίος βασίζεται στο άθροισμα των εσωτερικών γινομένων κάθε στοιχείου του διανύσματος-στήλης με το κελί της αντίστοιχης γραμμής του πίνακα.
   - Ο παραλληλισμός στην συγκεκριμένη περίπτωση υπάρχει στον υπολογισμό του τελικού αποτελέσματος-στοιχείου της παραπάνω διαδικασίας.
     Κάθε νήμα υπολογίζει και διαφορετικό στοιχείου του τελικού διανύσματος.



Παρακάτω βλέπουμε τις μετρήσεις και για τους τρεις αλγορίθμους της άσκησης 4.

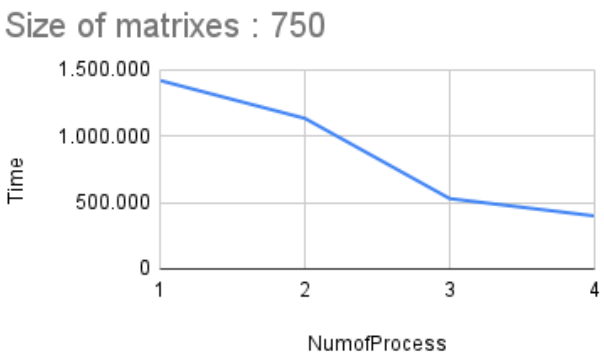


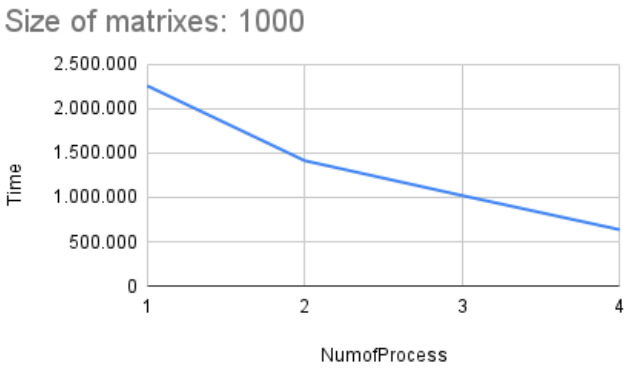




Άσκηση 5

Στην άσκηση αυτή πραγματοποιήσαμε πολλαπλασιασμό πίνακα 2D με διάνυσμα. Σύμφωνα με τις απαιτήσεις του μηχανήματος, οι οποίες είναι γνωστές από την προηγούμενη άσκηση, μπορέσαμε και τρέξαμε το πρόγραμμα μέχρι 4 επεξεργαστές. Κάναμε ένα πείραμα με πίνακα μεγέθους 1000x1000 και ένα ακόμα με μέγεθος 750x750. Στα παρακάτω διαγράμματα φαίνεται η σχέση αριθμός επεξεργαστών και χρόνος υπολογισμού. Όπως είναι αναμενόμενο, με την αύξηση του αριθμού των επεξεργαστών μειώνεται ο χρόνος εκτέλεσης.





Να σημειωθεί ότι ο χρόνος έχει υπολογιστεί σε μικρο-seconds(μs).

Όλες οι μετρήσεις των ασκήσεων βρίσκονται και στο αρχείο «Measurements_Assignment1_PDC» με τις αναλυτικές μετρήσεις.

ΠΑΡΑΡΤΗΜΑ

Κώδικας Άσκησης 4 (π.χ.)

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

```
/*********************************
 * Ergasia 1 – Askhsh 4
 * Ioannis Roumpos - 2980
 * Gerasimos Agoras - 2947
 *********************************/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "timer.h"

#define A(i,j)     A[(i)*M+j]
#define b(i)       b[i]
#define c(i)       c[i]

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

  int N  = 50;
  int M  = 40;

  double *A, *b, *c;

  int   size;
  int   i, j;

  /* Time */

  double time;

  if ( argc > 1 ) N  = atoi(argv[1]);
  if ( argc > 2 ) M  = atoi(argv[2]);

  printf("N=%d, M=%d\n", N, M);

  size = N * M * sizeof(double);
  A    = (double *)malloc(size);
  size = N * sizeof(double);
  c    = (double *)malloc(size);
  size = M * sizeof(double);
  b    = (double *)malloc(size);
```

```
  /* Initialize */

  for ( i=0 ; i < N ; i++ ) {
    for ( j=0 ; j < M ; j++ ) {
      A(i,j) = i + j;
      b(j)   = 1;
    }
  }

 /* Start Timer */
  initialize_timer ();
  start_timer();

  /* Compute */
  #pragma omp parallel for schedule(guided) private(j)
  for ( i=0 ; i < N ; i++ ) {
    c(i) = 0;
    for ( j=0; j < M ; j++ ) {
      c(i) += A(i,j) * b(j);
    }
  }

  /* stop timer */
  stop_timer();
  time=elapsed_time ();

  /* print results */

  for ( i=0 ; i < N ; i+= N/8 ) {
    printf("c[%d] = %lf\n", i, c(i));
  }

  printf("elapsed time = %lf\n", time);
  return 0;
}
```

Ακολουθεί ο κώδικας για το ερώτημα 4. Όνομα Αρχείου «stencil_1D.c»

```
/*********************************
 * Ergasia 1 – Askhsh 4
 * Ioannis Roumpos - 2980
 * Gerasimos Agoras - 2947
 *********************************/
#include <stdio.h>
#include <stdlib.h>
#include "timer.h"
#include <omp.h>

#define   INIT_VALUE      5000

void printResult(double *data, int size);

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

  int     N;
  int     t;
  int     MAX_ITERATION = 2000;
  double  *prev, *cur;
  double  error = INIT_VALUE;

  // Timer
  double  time;

  // temporary variables
  int     i,j;
  double  *temp;

  // Check commandline args.
  if ( argc > 1 ) {
    N = atoi(argv[1]);
  } else {
    printf("Usage : %s [N]\n", argv[0]);
    exit(1);
  }
  if ( argc > 2 ) {
    MAX_ITERATION = atoi(argv[2]);
  }

  // Memory allocation for data array.
  prev  = (double *) malloc( sizeof(double) * N);
  cur   = (double *) malloc( sizeof(double) * N);
  if ( prev == NULL || cur == NULL ) {
    printf("[ERROR] : Fail to allocate memory.\n");
```

```
    exit(1);
  }

  // Initialization

  for ( i=1 ; i < N-1 ; i++ ) {
      prev[i] = 0.0;
  }

    prev[0]  = prev[1]  = INIT_VALUE;
    prev[N-1]  = prev[N-2]  = INIT_VALUE;
    cur[0]  = cur[1]  = INIT_VALUE;
    cur[N-1]  = cur[N-2]  = INIT_VALUE;

  initialize_timer();
  start_timer();

  // Computation
  t = 0;


  while ( t < MAX_ITERATION) {

    // Computation
    #pragma omp parallel for
    for ( i=2 ; i < N-2 ; i++ ) {
        cur[i] = (prev[i-2]+prev[i-1]+prev[i]+prev[i+1]+prev[i+2])/5;
    }

    {
    temp = prev;
    prev = cur;
    cur  = temp;
    t++;
    }
  }

  stop_timer();
  time = elapsed_time();

  printResult(prev, N);

  printf("Data size : %d  , #iterations : %d , time : %lf sec\n", N, t, time);
}

void printResult(double *data, int size) {
  int i;
```

```
/* print a portion of the vector */
    printf("data[%d]: %lf \n",0, data[0]);
    printf("data[%d]: %lf \n",1, data[1]);
    printf("data[%d]: %lf \n",size/10, data[size/10]);
    printf("data[%d]: %lf \n",size/5, data[size/5]);
    printf("data[%d]: %lf \n",size/2, data[size/2]);

  return;
}
```

Ακολουθεί ο κώδικας για το ερώτημα 4. Όνομα Αρχείου «stencil_2D.c»

```
/*********************************
 * Ergasia 1 – Askhsh 4
 * Ioannis Roumpos - 2980
 * Gerasimos Agoras - 2947
 *********************************/
#include <stdio.h>
#include <stdlib.h>
#include "timer.h"
#include <math.h>
#define   INIT_VALUE      10000.0
#define   prev(i,j)       prev[(i)*N+(j)]
#define   cur(i,j)        cur[(i)*N+(j)]


void printMatrix(double *data, int size);

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

  int     N;
  int     t;
  int     MAX_ITERATION = 2000;
  double  *prev, *cur;

  //double  error = INIT_VALUE;

  // Timer
  double  time;

  // temporary variables
  int     i,j;
  double  *temp;

  // Check commandline args.
  if ( argc > 1 ) {
```

```
  N = atoi(argv[1]);
} else {
  printf("Usage : %s [N]\n", argv[0]);
  exit(1);
}
if ( argc > 2 ) {
  MAX_ITERATION = atoi(argv[2]);
}

// Memory allocation for data array.
prev  = (double *) malloc( sizeof(double) * N * N );
cur   = (double *) malloc( sizeof(double) * N * N );
if ( prev == NULL || cur == NULL ) {
  printf("[ERROR] : Fail to allocate memory.\n");
  exit(1);
}

// Initialization

for ( i=2 ; i < N-2 ; i++ ) {
  for ( j=2 ; j < N-2 ; j++ ) {
    prev(i,j) = 0.0;
  }
}

for ( i=0 ; i < N ; i++ ) {
  prev(i  , 0  ) = INIT_VALUE;  prev(i  , 1  ) = INIT_VALUE;
  prev(i  , N-1) = INIT_VALUE;  prev(i  , N-2) = INIT_VALUE;
  prev(0  , i  ) = INIT_VALUE;  prev(1  , i  ) = INIT_VALUE;
  prev(N-1, i  ) = INIT_VALUE;  prev(N-2, i  ) = INIT_VALUE;

  cur( i  , 0  ) = INIT_VALUE;  cur( i  , 1  ) = INIT_VALUE;
  cur( i  , N-1) = INIT_VALUE;  cur( i  , N-2) = INIT_VALUE;
  cur( 0  , i  ) = INIT_VALUE;  cur( 1  , i  ) = INIT_VALUE;
  cur( N-1, i  ) = INIT_VALUE;  cur( N-2, i  ) = INIT_VALUE;
}

initialize_timer();
start_timer();

// Computation
t = 0;


while ( t < MAX_ITERATION) {

  // Computation
  #pragma omp parallel for private(j) shared(cur)
```

```
    for ( i=2 ; i < N-2 ; i++ ) {
      for ( j=2 ; j < N-2 ; j++ ) {
        cur(i,j) = (prev(i-2,j)+prev(i-1,j)+prev(i+1,j)+prev(i+2,j)+
                    prev(i,j)+
                          prev(i,j-2)+prev(i,j-1)+prev(i,j+1)+prev(i,j+2)
                   )/9;
      }
    }

    temp = prev;
    prev = cur;
    cur  = temp;
    t++;

  }

  stop_timer();
  time = elapsed_time();

  printf("Data : %d by %d , Iterations : %d , Time : %lf sec\n", N, N, t, time);
  printf("Final data\n");
  printMatrix(prev, N);



}

void printMatrix(double *data, int size) {
  int i,j;

/* print a portion of the matrix */

  //#pragma omp parallel for private(j)
  for ( i= 0 ; i < 5 ; i++ ) {
    for ( j=0 ; j < 5 ; j++ ) {
      printf("%lf ", data[i*size+j]);
    }
    printf("\n");
  }

  return;
}
```

Κώδικας Άσκησης 5 (π.χ.)

Ακολουθεί ο κώδικας για το ερώτημα 6. Όνομα Αρχείου «ask5.c»

```
/**********************************
 * Ergasia 1 – Askhsh 5
 * Ioannis Roumpos - 2980
 * Gerasimos Agoras - 2947
 **********************************/
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>
#include <time.h>

#define N 100000

int main(int argc,char*argv[]){
   /*int N;
   N = atoi(argv[1]);
   if(argc > 2 || argc == 0){
      printf("./ask5 [N]\n");
      return -1;
   }*/

   int rank,size;
   int A[N],B[N][N],C[N],B_temp[N][N];
   int sum[N];
   struct timespec t_start = {0,0}, t_stop = {0,0};

   //Start mpi program
   MPI_Init(&argc,&argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   MPI_Comm_size(MPI_COMM_WORLD, &size);

   int root = 0;
   int thread_workload = N/size;
   /*Initialise matrix-vector*/
   if (rank == root) {
      for(int i=0; i<N; i++){
         for(int j=0; j<N; j++){
            B[i][j] = rand()%100;
         }
      }

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

```
            C[i] = rand()%100;
        }
    }

    /*Scatter matrix B row-wise*/
    MPI_Scatter(B, thread_workload, MPI_INT, B_temp, thread_workload, MPI_INT, 0,
MPI_COMM_WORLD);

    /*Broadcast vector C */
    MPI_Bcast(C, N, MPI_INT, root, MPI_COMM_WORLD);

    /*Calculate inner product and save in sum*/
    clock_gettime(CLOCK_MONOTONIC, &t_start);
    for(int i=0; i<N; i++){
        for(int j=0; j<N; j++)
            sum[i] += C[j]*B_temp[i][j];
    }
    clock_gettime(CLOCK_MONOTONIC, &t_stop);

    /*Gather the results*/
    MPI_Gather(&sum, thread_workload, MPI_INT, A, thread_workload, MPI_INT, 0,
MPI_COMM_WORLD);


/*printf("\n");
    if(rank == 0) {
        for(int i=0; i<N; i++){
            printf("A[%d]=%d\n", i,A[i]);
        }
    }*/

printf("Computation time is: %.6f\n",
          (((double)t_stop.tv_sec + 1.0e-9*t_stop.tv_nsec) -
          ((double)t_start.tv_sec + 1.0e-9*t_start.tv_nsec))*1000);


    MPI_Finalize();
}
```

.

.