
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών Ε.Μ.Π.

# ΣΥΣΤΗΜΑΤΑ ΠΑΡΑΛΛΗΛΗΣ ΕΠΕΞΕΡΓΑΣΙΑΣ

2016-2017

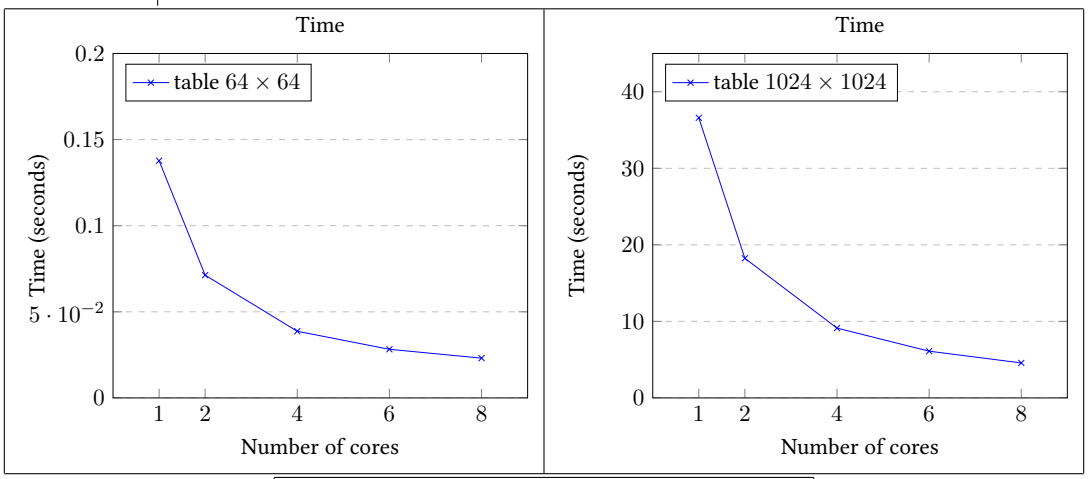
## Άσκηση 1
## Εισαγωγή στην Παραλληλοποίηση σε Αρχιτεκτονικές Κοινής Μνήμης

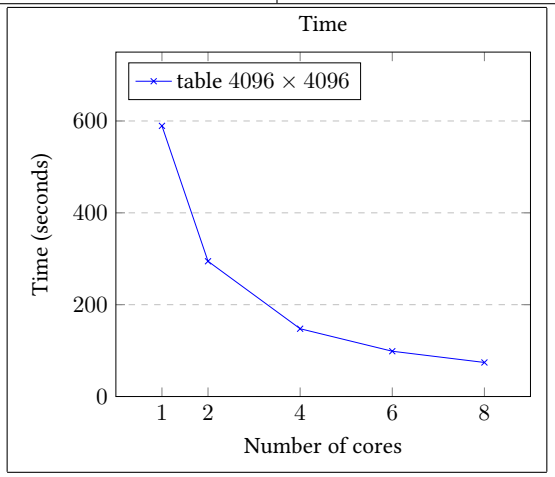
Βαρηά Χρυσούλα - 03112105
Κουτσανίτη Ειρήνη - 03112135

Ο τροποποιημένος κώδικας της άσκησης παρουσιάζεται στο τέλος της αναφοράς. Συγκεκριμένα οι αλλαγές που έγιναν αφορούν το for loop του υπολογισμού της μεταβλητής nbrs με χρήση των στοιχείων του πίνακα previous (δηλαδή η παραλληλοποίηση εμπλέκεται στο διαδοχικό υπολογισμό των τιμών του πίνακα current σε κάθε χρονική στιγμή t).

Στη συνέχεια παρουσιάζονται οι μετρήσεις, οι οποίες προέκυψαν από την εκτέλεση του αρχικού σειριακού προγράμματος και του αντίστοιχου παράλληλου, το οποίο αναπτύχθηκε με τη χρήση του OpenMP. Όλες οι μετρήσεις έγιναν για 1000 γενιές.

| | Σειριακό πρόγραμμα | 1 core | 2 cores | 4 cores | 6 cores | 8 cores |
|---|---|---|---|---|---|---|
| $64 \times 64$ | 0.112564 | 0.137741 | 0.071278 | 0.038715 | 0.028244 | 0.023109 |
| $1024 \times 1024$ | 30.795014 | 36.595844 | 18.242129 | 9.122369 | 6.107605 | 4.573550 |
| $4096 \times 4096$ | 496.127435 | 589.356565 | 294.553583 | 147.627172 | 98.632691 | 74.183505 |



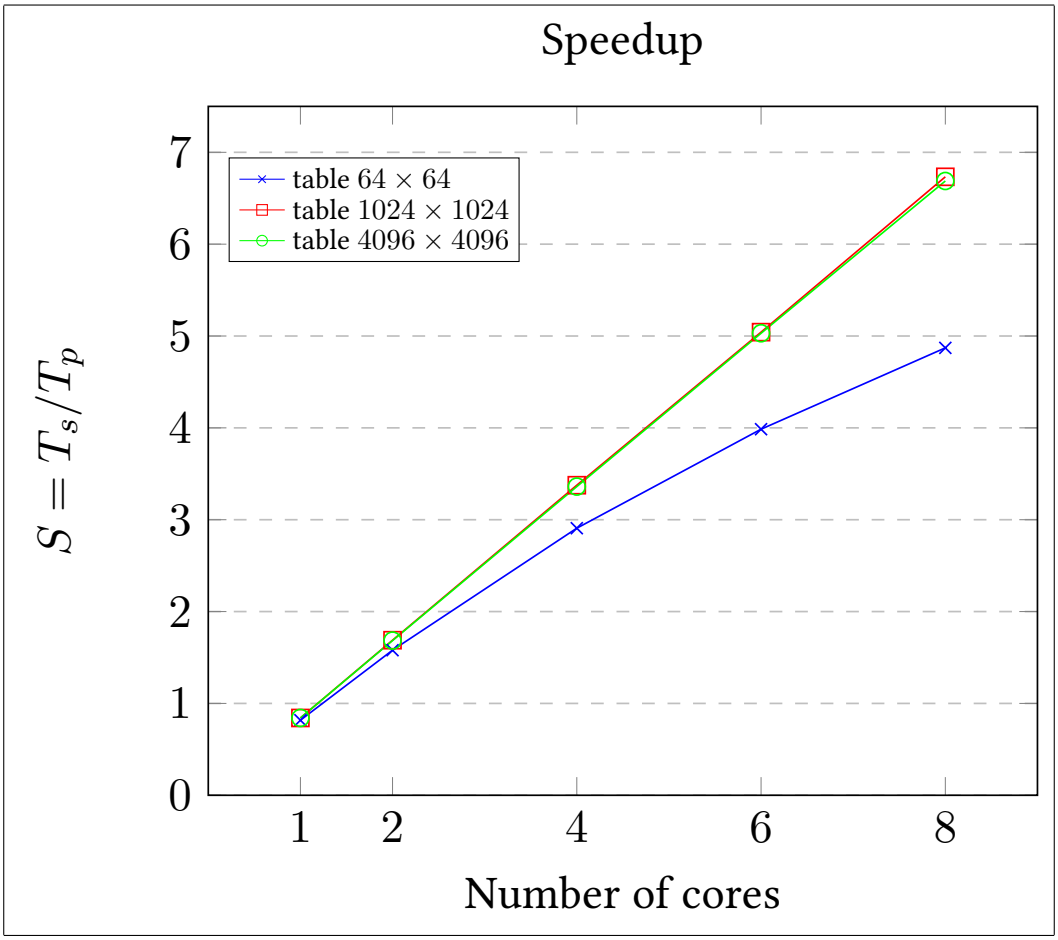

**Παρατηρήσεις**:

- Από τις παραπάνω γραφικές παραστάσεις προκύπτει ότι η αύξηση του πλήθους των επεξεργαστών οδηγεί σε μείωση του χρόνου εκτέλεσης του προγράμματος. Αυτό είναι αναμενόμενο, καθώς με την αξιοποίηση περισσότερων επεξεργαστών, άρα και τη χρήση περισσότερων νημάτων εκτέλεσης, περισσότεροι υπολογισμοί γίνονται παράλληλα, με αποτέλεσμα να μειώνεται ο συνολικός χρόνος εκτέλεσης.

- Επίσης παρατηρείται ότι με την αύξηση του μεγέθους του πίνακα, η μείωση του χρόνου εκτέλεσης είναι μεγαλύτερη. Αυτό οφείλεται στο γεγονός ότι αυξάνοντας το μέγεθος του πίνακα και περιορίζοντας το πλήθος των επεξεργαστών, η σειριοποίηση της επεξεργασίας των δεδομένων γίνεται εντονότερη, με αποτέλεσμα να αυξάνεται ο συνολικός χρόνος εκτέλεσης.

- Τέλος το παράλληλο πρόγραμμα μπορεί να έχει σημαντικό overhead όταν εκτελείται με χρήση ενός νήματος, όπως φαίνεται στο χρόνο του προγράμματος με είσοδο πίνακα διαστάσεων $4096 \times 4096$ (λόγω υιοθέτησης διαφορετικών τεχνικών βελτιστοποίησης κατά την μεταγλώττιση, αρχικοποίηση των δομών του νήματος κτλ.). Σε αυτή την περίπτωση υπάρχει καθυστέρηση περίπου ενάμιση λεπτό σε σχέση με το σειριακό πρόγραμμα.

Για την αξιολόγηση της απόδοσης στη συνέχεια φαίνονται δύο μετρικές, η επιτάχυνση και η αποδοτικότητα του παράλληλου προγράμματος.

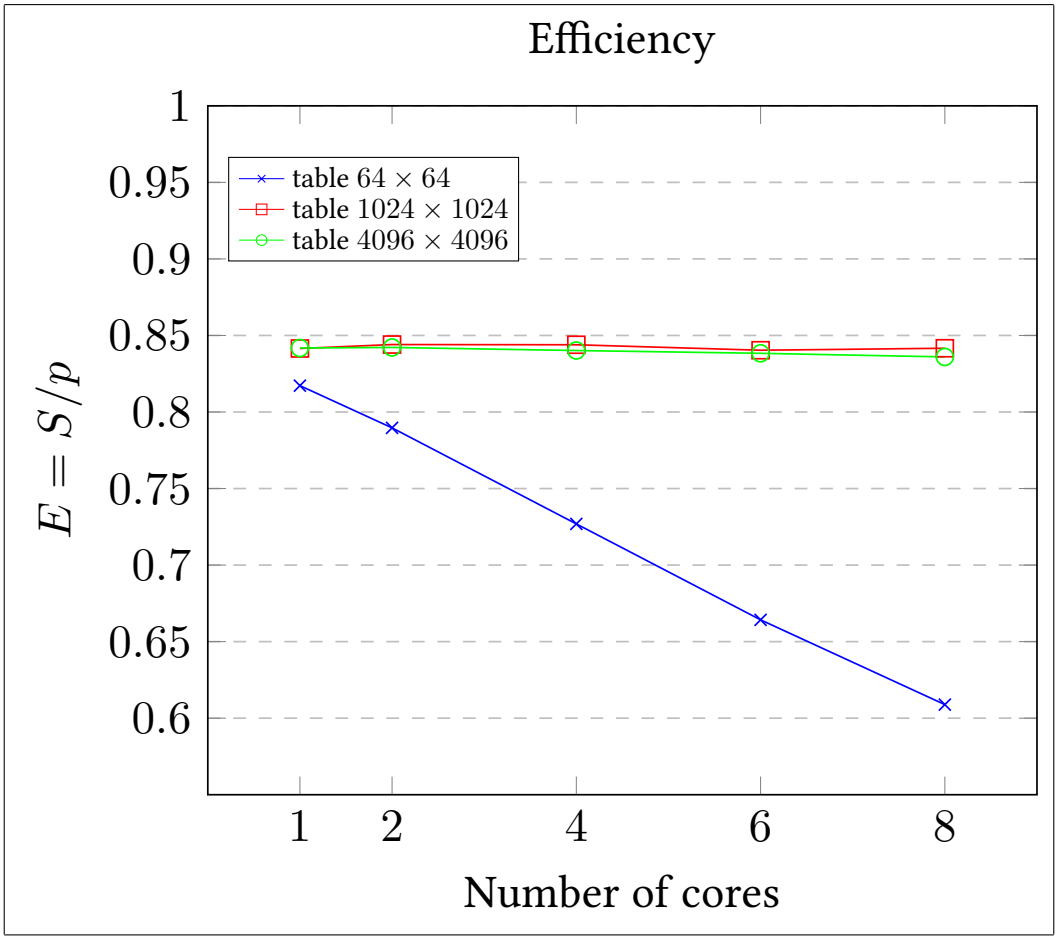



**Παρατηρήσεις**:

- Αρχικά παρατηρείται ότι στις περιπτώσεις του πίνακα με διαστάσεις $1024 \times 1024$ και $4096 \times 4096$ η επιτάχυνση S είναι σχεδόν γραμμική (linear speedup: $S \approx p$, όπου p είναι το πλήθος των επεξεργαστών). Στην περίπτωση πίνακα διαστάσεων $64 \times 64$, η επιτάχυνση είναι τυπική (typical speedup: S < p).

- Επίσης πρέπει να αναφερθεί ότι η προσθήκη περισσότερων επεξεργαστών (> 8) για την εκτέλεση του προγράμματος με πίνακα διαστάσεων $64 \times 64$, μπορεί να μην οδηγήσει σε μεγάλη μείωση του χρόνου εκτέλεσης. Σε ακραίες περιπτώσεις μάλιστα, το overhead της επικοινωνίας/συγχρονισμού μεταξύ των threads ίσως οδηγεί σε αύξηση του χρόνου εκτέλεσης, δηλαδή μείωση του speedup. Αντίθετα στις υπόλοιπες διαστάσεις, η αύξηση του πλήθους των επεξεργαστών, μπορεί να ευνοήσει την εκτέλεση και επιτάχυνση του προγράμματος.

- Από τα παραπάνω συμπεραίνουμε ότι το πλήθος των threads για την αποδοτικότερη εκτέλεση του προγράμματος εξαρτάται άμεσα από το είδος των δεδομένων εισόδου (δηλαδή αύξηση των νημάτων δε συνεπάγεται πάντα βελτίωση της απόδοσης).




**Παρατηρήσεις**:

- Σε αντίθεση με τις παρατηρήσεις των προηγούμενων διαγραμμάτων, από πλευράς αποδοτικότητας, η παραλληλοποίηση δε φαίνεται να είναι τόσο επιτυχημένη. Με άλλα λόγια το ποσοστό του χρόνου, στον οποίο κάθε επεξεργαστής κάνει χρήσιμη δουλειά δε βελτιώνεται με την αύξηση του πλήθους επεξεργαστών. Συγκεκριμένα στις περιπτώσεις των διαστάσεων $1024 \times 1024$ και $4096 \times 4096$ η αποδοτικότητα είναι σταθερή, ενώ στην περίπτωση διαστάσεων $64 \times 64$ μειώνεται. Αυτό μπορεί να οφείλεται τόσο στο κόστος επικοινωνίας/συγχρονισμού μεταξύ των νημάτων, όσο και στις περιορισμένες δυνατότητες μεταφοράς δεδομένων από τη μνήμη (memory wall), προκαλώντας αναμονή στις επεξεργαστικές μονάδες.

- Από τα παραπάνω συμπεραίνουμε ότι με την αύξηση των επεξεργαστών, ο χρόνος, στον οποίο αυτοί είναι ενεργοί κατά την εκτέλεση του προγράμματος, μπορεί να μειώνεται (δηλαδή η αξιοποίηση των διαθέσιμων πόρων δεν είναι αρκετά καλή). Επομένως θα πρέπει σε κάθε περίπτωση να λαμβάνεται υπόψη τόσο η γρήγορη εκτέλεση ενός προγράμματος όσο και η σωστή αξιοποίηση του υλικού, ανάλογα με τα δεδομένα εισόδου(δηλαδή να γίνονται κατάλληλα trade offs).

Κώδικας Game_Of_Life.c

```
/*****************************************************
 ************* Conway's game of life ******************
 *****************************************************

 Usage: ./exec ArraySize TimeSteps

 Compile with -DOUTPUT to print output in output.gif
 (You will need ImageMagick for that - Install with
  sudo apt-get install imagemagick)
 WARNING: Do not print output for large array sizes!
          or multiple time steps!
 *****************************************************/

#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

#define FINALIZE "\
convert -delay 20 out*.pgm output.gif\n\
rm *pgm\n\
"

int␣**␣allocate_array(int␣N);
void␣free_array(int␣**␣array,␣int␣N);
void␣init_random(int␣**␣array1,␣int␣**␣array2,␣int␣N);
void␣print_to_pgm(␣int␣**␣array,␣int␣N,␣int␣t␣);

int␣main␣(int␣argc,␣char␣*␣argv[])␣{
␣␣␣␣int␣N;␣␣␣␣␣␣␣␣␣␣␣␣␣␣//array␣dimensions
␣␣␣␣int␣T;␣␣␣␣␣␣␣␣␣␣␣␣␣␣//time␣steps
␣␣␣␣int␣**␣current,␣**␣previous;␣␣␣␣//arrays␣-␣one␣for␣current␣
   timestep,␣one␣for␣previous␣timestep
␣␣␣␣int␣**␣swap;␣␣␣␣␣␣␣␣␣␣␣␣//array␣pointer
␣␣␣␣int␣t,␣i,␣j,␣nbrs;␣␣␣␣␣␣//helper␣variables

␣␣␣␣double␣time;␣␣␣␣␣␣␣␣␣␣␣␣//variables␣for␣timing
␣␣␣␣struct␣timeval␣ts,tf;

␣␣␣␣int␣tid;
␣␣␣␣FILE␣*fp;

␣␣␣␣/*Read␣input␣arguments*/
␣␣␣␣if␣(␣argc␣!=␣3␣)␣{
␣␣␣␣␣␣␣␣fprintf(stderr,␣"Usage: ./exec ArraySize TimeSteps\n");
␣␣␣␣␣␣␣␣exit(-1);
␣␣␣␣}
␣␣␣␣else␣{
```

```
␣␣␣␣␣␣␣␣N␣=␣atoi(argv[1]);
␣␣␣␣␣␣␣␣T␣=␣atoi(argv[2]);
␣␣␣␣}

␣␣␣␣/*Allocate␣and␣initialize␣matrices*/
␣␣␣␣current␣=␣allocate_array(N);␣␣␣␣␣␣␣␣␣␣␣␣␣//allocate␣array␣for␣
    current␣time␣step
␣␣␣␣previous␣=␣allocate_array(N);␣␣␣␣␣␣␣␣␣␣␣␣␣//allocate␣array␣for␣
    previous␣time␣step

␣␣␣␣init_random(previous,␣current,␣N);␣␣//initialize␣previous␣array␣
    with␣pattern

␣␣␣␣#ifdef␣OUTPUT
␣␣␣␣print_to_pgm(previous,␣N,␣0);
␣␣␣␣#endif

␣␣␣␣/*Game␣of␣Life*/

␣␣␣␣gettimeofday(&ts,NULL);
␣␣␣␣for␣(␣t␣=␣0␣;␣t␣<␣T␣;␣t++␣)␣{
␣␣␣␣␣␣␣␣#pragma␣omp␣parallel␣for␣private(i,␣j,␣nbrs)
␣␣␣␣␣␣␣␣␣␣␣␣for␣(␣i␣=␣1␣;␣i␣<␣N-1␣;␣i++␣)␣{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣for␣(␣j␣=␣1␣;␣j␣<␣N-1␣;␣j++␣)␣{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣nbrs␣=␣previous[i+1][j+1]␣+␣previous[i+1][j]␣+␣
    previous[i+1][j-1]␣\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+␣previous[i][j-1]␣+␣previous[i][j+1]␣\
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣+␣previous[i-1][j-1]␣+␣previous[i-1][j]␣+␣
    previous[i-1][j+1];
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣if␣(␣nbrs␣==␣3␣||␣(␣previous[i][j]+nbrs␣==3␣)␣)
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣current[i][j]=1;
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣else
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣current[i][j]=0;
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣}
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣}

␣␣␣␣␣␣␣␣␣␣#ifdef␣OUTPUT
␣␣␣␣␣␣␣␣␣␣print_to_pgm(current,␣N,␣t+1);
␣␣␣␣␣␣␣␣␣␣#endif
␣␣␣␣␣␣␣␣␣␣//Swap␣current␣array␣with␣previous␣array
␣␣␣␣␣␣␣␣␣␣swap=current;
␣␣␣␣␣␣␣␣␣␣current=previous;
␣␣␣␣␣␣␣␣␣␣previous=swap;

␣␣␣␣␣␣}
␣␣␣␣␣␣gettimeofday(&tf,NULL);
␣␣␣␣␣␣time=(tf.tv_sec-ts.tv_sec)+(tf.tv_usec-ts.tv_usec)*0.000001;

␣␣␣␣␣␣free_array(current,␣N);
```

```
    free_array(previous, N);
    printf("GameOfLife: Size %d Steps %d Time %lf\n", N, T, time);
    #ifdef OUTPUT
    system(FINALIZE);
    #endif
}

int **allocate_array(int N) {
    int **array;
    int i,j;
    array = malloc(N * sizeof(int*));
    for ( i = 0; i < N; i++ )
        array[i] = malloc( N * sizeof(int));
    for ( i = 0; i < N; i++ )
        for ( j = 0; j < N; j++ )
            array[i][j] = 0;
    return array;
}

void free_array(int **array, int N) {
    int i;
    for ( i = 0; i < N; i++ ) {
        free(array[i]);
    }
    free(array);
}

void init_random(int **array1, int **array2, int N) {
    int i,pos,x,y;

    for ( i = 0; i < (N * N)/10 ; i++ ) {
        pos = rand() % ((N-2)*(N-2));
        array1[pos%(N-2)+1][pos/(N-2)+1] = 1;
        array2[pos%(N-2)+1][pos/(N-2)+1] = 1;

    }
}

void print_to_pgm(int **array, int N, int t) {
    int i,j;
    char *s = malloc(30*sizeof(char));
    sprintf(s,"out%d.pgm",t);
    FILE *f = fopen(s,"wb");
    fprintf(f, "P5\n%d %d 1\n", N,N);
    for ( i = 0; i < N; i++ )
        for ( j = 0; j < N; j++)
            if ( array[i][j]==1 )
                fputc(1,f);
            else
```

```
                fputc(0,f);
    fclose(f);
    free(s);
}
```