



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ


Συστήματα Παράλληλης Επεξεργασίας

Αναφορά 1ης Εργαστηριακής Άσκησης

**Ραπτόπουλος Πέτρος (el19145)
Κόγιος Δημήτριος (el19220)
Καπετανάκης Αναστάσιος (el19048)**

1. Συνδεόμαστε στο cluster του εργαστηρίου και λύνουμε πιθανά προβλήματα σύνδεσης:

```
parlab10@scirouter: ~  
(base) petrosrpto@petrosrptoAssistant:~$ ssh parlab10@orion.cslab.ece.ntua.gr  
The authenticity of host 'orion.cslab.ece.ntua.gr (147.102.3.236)' can't be established.  
ED25519 key fingerprint is SHA256:bA9q2rUKW1LvYf8uC1UaZTgLB0DJIYW+Nzvp7zrrsbdI.  
This key is not known by any other names  
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes  
Warning: Permanently added 'orion.cslab.ece.ntua.gr' (ED25519) to the list of known hosts.  
parlab10@orion.cslab.ece.ntua.gr's password:  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Wed Oct 11 12:19:49 2023 from 37.6.86.151  
parlab10@orion:~$ ssh scirouter.cslab.ece.ntua.gr  
Linux scirouter 2.6.26-2-amd64 #1 SMP Sun Mar 4 21:48:06 UTC 2012 x86_64  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
  
=====
```



```
=====
```

If you have any problems please contact admins@cs-lab.ece.ntua.gr

```
=====
```

Last login: Wed Oct 11 12:22:00 2023 from orion.cslab.ece.ntua.gr
parlab10@scirouter:~\$

2. Γινόμαστε οικείοι με την μεταγλώττιση και την υποβολή jobs στα cluster queues. Για παράδειγμα μεταγλωττίζουμε και υποβάλλουμε την δοθείσα σειριακή εκδοχή του Game Of Love:

```
parlab10@scirouter:~/ex1/serial$ ls
gameOfLife.c  Makefile  make_on_queue.sh  run_on_queue.sh
parlab10@scirouter:~/ex1/serial$ qsub -q parlab make_on_queue.sh
516044.localhost
parlab10@scirouter:~/ex1/serial$ ls
gameOfLife  gameOfLife.c  Makefile  make_gameOfLife.err  make_gameOfLife.out  make_on_queue.sh  run_on_queue.sh
parlab10@scirouter:~/ex1/serial$ cat make_gameOfLife.out
gcc -O3 -fopenmp -o gameOfLife gameOfLife.c
parlab10@scirouter:~/ex1/serial$ cat make_gameOfLife.err
parlab10@scirouter:~/ex1/serial$

parlab10@scirouter:~/ex1/serial$ qsub -q parlab run_on_queue.sh
516046.localhost
parlab10@scirouter:~/ex1/serial$ qstat -f 516046.localhost
qstat: Unknown Job Id 516046.localhost
parlab10@scirouter:~/ex1/serial$ ls
gameOfLife  Makefile  make_gameOfLife.out  run_gameOfLife.err  run_on_queue.sh
gameOfLife.c  make_gameOfLife.err  make_on_queue.sh  run_gameOfLife.out
parlab10@scirouter:~/ex1/serial$ cat run_gameOfLife.out
GameOfLife: Size 64 Steps 1000 Time 0.020344
parlab10@scirouter:~/ex1/serial$ cat run_gameOfLife.err
parlab10@scirouter:~/ex1/serial$
```

Τα περιεχόμενα των Makefile, make_on_queue.sh και run_on_queue.sh είναι τα εξής:

```
parlab10@scirouter:~/ex1/serial$ cat Makefile
all: gameOfLife

gameOfLife: gameOfLife.c
        gcc -O3 -fopenmp -o gameOfLife gameOfLife.c

clean:
        rm gameOfLife
```

```
parlab10@scirouter:~/ex1/serial$ cat make_on_queue.sh
#!/bin/bash

## Give the Job a descriptive name
#PBS -N make_gameOfLife

## Output and error files
#PBS -o make_gameOfLife.out
#PBS -e make_gameOfLife.err

## How many machines should we get?
#PBS -l nodes=1:ppn=1

##How long should the job run for?
#PBS -l walltime=00:10:00

## Start
## Run make in the src folder (modify properly)

module load openmp
cd /home/parallel/parlab10/ex1/serial
make
```

```
parlab10@scirouter:~/ex1/serial$ cat run_on_queue.sh
#!/bin/bash

## Give the Job a descriptive name
#PBS -N run_gameOfLife

## Output and error files
#PBS -o run_gameOfLife.out
#PBS -e run_gameOfLife.err

## How many machines should we get?
#PBS -l nodes=1:ppn=8

##How long should the job run for?
#PBS -l walltime=00:10:00

## Start
## Run make in the src folder (modify properly)

module load openmp
cd /home/parallel/parlab10/ex1/serial
export OMP_NUM_THREADS=8
./gameOfLife 64 1000
```

3. Παρατηρούμε ότι η κατάσταση κάθε χρονικής στιγμής εξαρτάται από αυτή της προηγούμενης χρονικής στιγμής και συνεπώς δεν μπορούμε να παραλληλοποιήσουμε την εργασία ως προς τον χρόνο. Όμως η κατάσταση κάθε κελιού την χρονική στιγμή t εξαρτάται μόνο από την κατάσταση των γειτονικών του κελιών για τη στιγμή $t-1$. Υπό αυτό το πρίσμα μπορούμε να βρούμε τη καθολική κατάσταση της χρονικής στιγμής t παράλληλα:

Προσθέτουμε τη γραμμή `#pragma omp parallel for shared(N, previous, current) private(i, j, nbrs)`

πριν το δεύτερο εμφωλιασμένο βρόγχο ώστε να παραλληλοποιήσουμε τον δοθέντα κώδικα υποθέτοντας `shared` address space και χρησιμοποιώντας το OpenMP.

Με την εντολή αυτή ζητάμε να παραλληλοποιηθεί ο βρόγχος που ακολουθεί ανάμεσα σε `OMP_NUM_THREADS` (μεταβλητή περιβάλλοντος) νήματα. Δηλώνουμε ότι η μεταβλητή `N` καθώς και οι δείκτες `previous/current` μοιράζονται μεταξύ των νημάτων, ενώ οι μεταβλητές `i, j, nbrs` είναι ιδιωτικά για το εκάστοτε νήμα.

Άρα ο κώδικας γίνεται:

```
parlab10@scirouter:~/ex1/parallel$ cat gameOfLife.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 <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

#define FINALIZE "\
convert -delay 20 `ls -1 out*.pgm | sort -V` output.gif\
rm *pgm\
"

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;

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

```

        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 shared(N, previous, current) 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;

```



```

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);
}

```

Στο script run_on_queue.sh προσθέτουμε το παρακάτω:

```

for thr in 1 2 4 6 8
do
    export OMP_NUM_THREADS=$thr
    ./gameOfLife 64 1000
    ./gameOfLife 1024 1000
    ./gameOfLife 4096 1000
done

```

4. Λαμβάνουμε μετρήσεις επίδοσης (Συνολικό χρόνο εκτέλεσης σε sec) για έναν κλώνο για 1,2,4,6,8 πυρήνες και για μεγέθη πίνακα 64 x 64, 1024 x 1024, 4096 x 4096 έχοντας θέσει 1000 γενιές/time-steps για όλες τις περιπτώσεις.

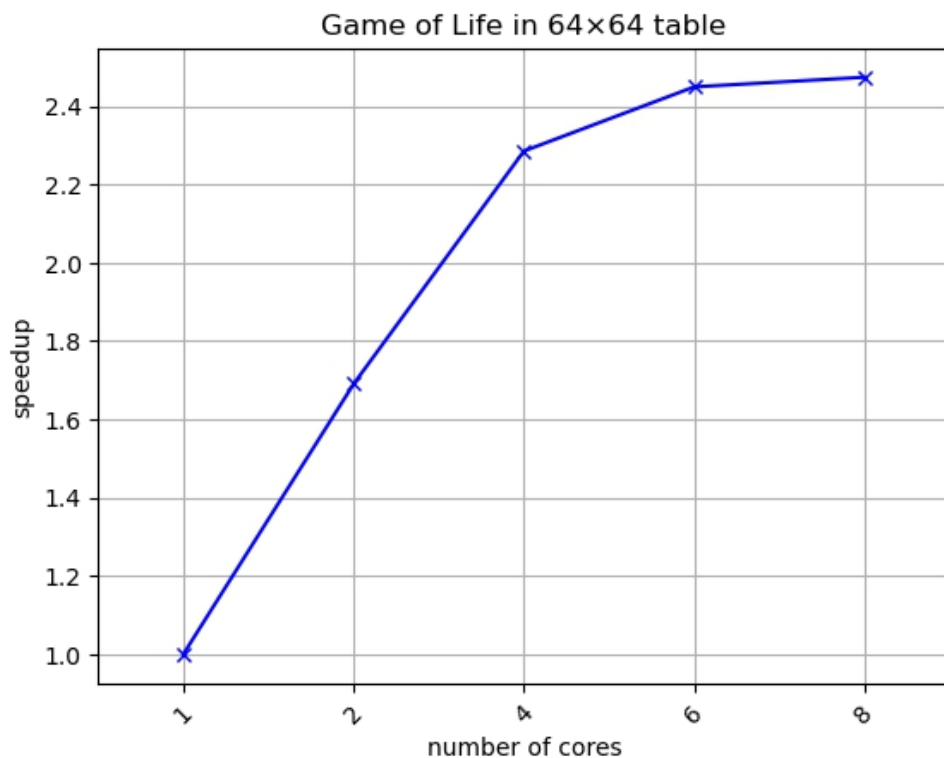
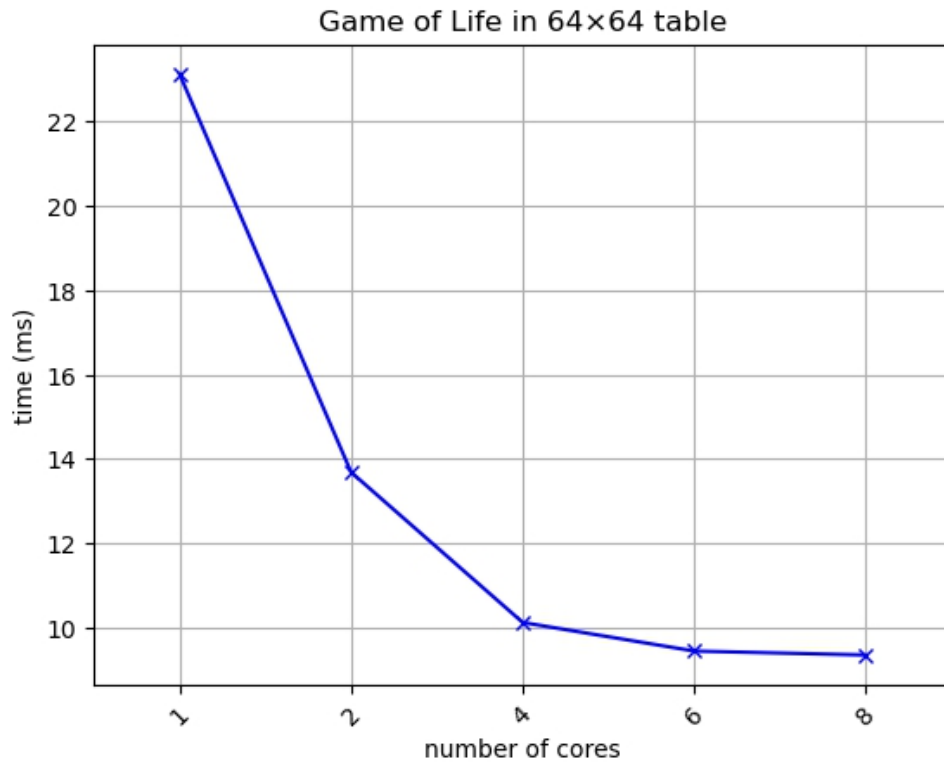
Cores/Table Size	64 x 64	1024 x 1024	4096 x 4096
1	0.023132	10.970660	175.946833
2	0.013687	5.457147	88.263236
4	0.010124	2.723553	44.536678
6	0.009446	1.830368	37.184969
8	0.009352	1.376726	36.517382

5. Προκειμένου να έχουμε εποπτική εικόνα της επίδρασης της αύξησης των cores στην επίδοση κατασκευάζουμε για κάθε μέγεθος ταμπλό διάγραμμα μεταβολής χρόνου και speedup συναρτήσεως του αριθμού των πυρήνων.

Σκοπός της παραλληλοποίησης είναι η μείωση του χρόνου εκτέλεσης, ιδανικά ανάλογα με τον αριθμό των πυρήνων. Ωστόσο αυτό δεν συμβαίνει πάντα λόγω:

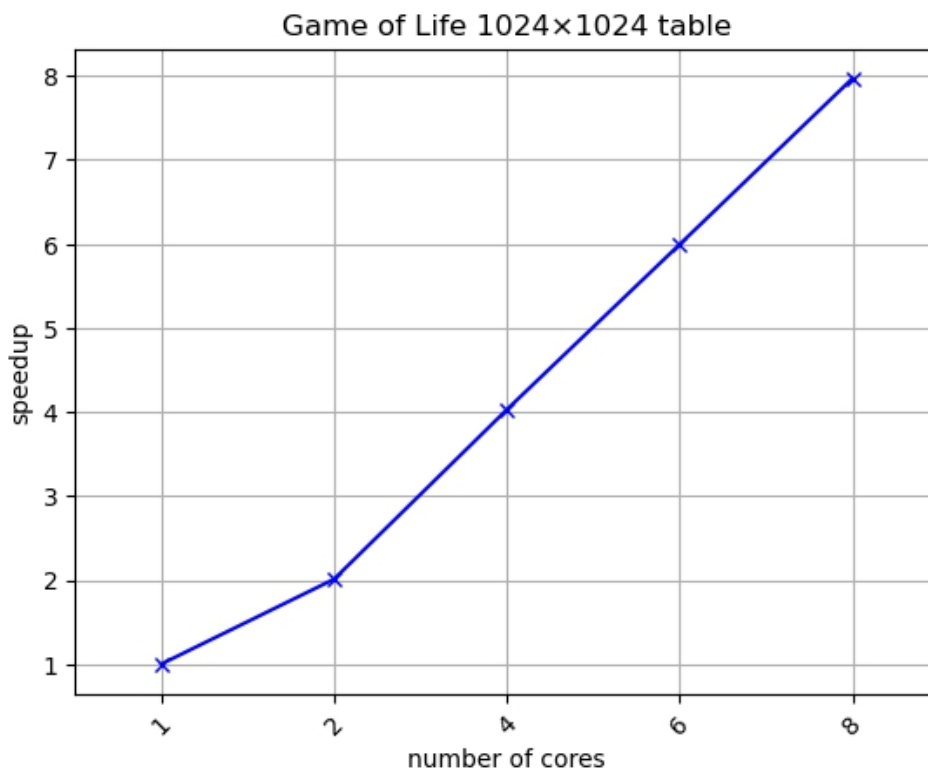
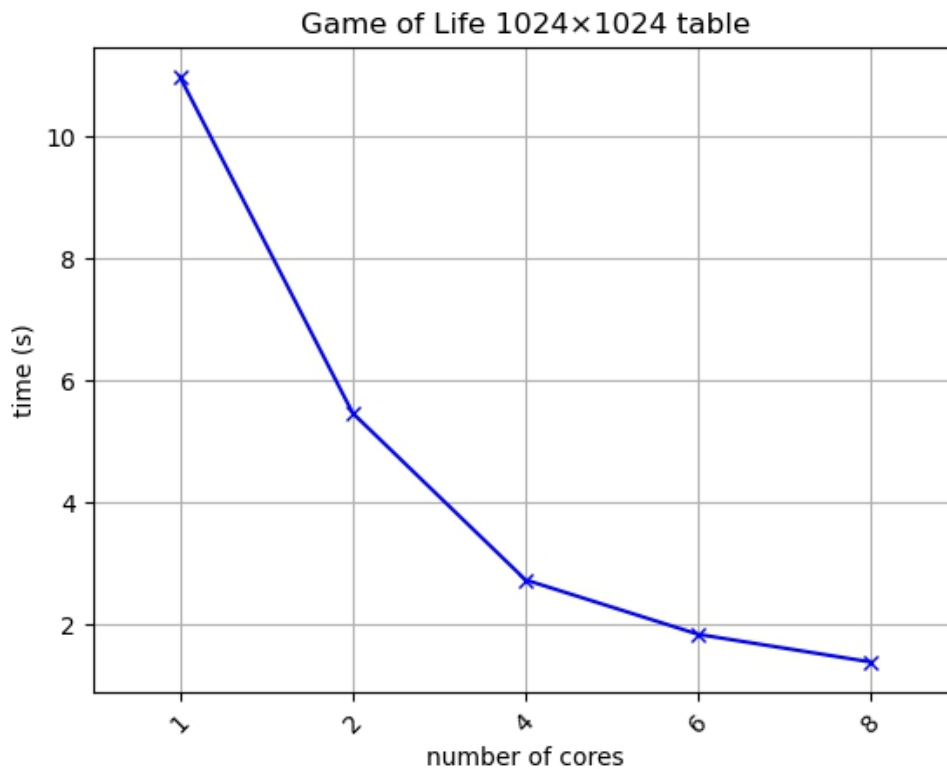
1. Κόστος δημιουργίας και τερματισμού νημάτων: Καθώς αυξάνεται ο αριθμός των νημάτων, το overhead που σχετίζεται με τη δημιουργία, διαχείριση και τερματισμό των νημάτων μπορεί να γίνει σημαντικό, ειδικά αν τα threads ζουν για λίγο.
2. Κόστος συγχρονισμού: Εάν ένα thread απαιτεί συχνό συγχρονισμό (χρησιμοποιώντας locks, semaphores...) το overhead αυξάνεται καθώς περισσότερα νήματα διαγωνίζονται για τους ίδιους πόρους.
3. Ανταγωνισμός πόρων: Πολλαπλά threads μπορεί να διαγωνίζονται για κοινούς πόρους, όπως μνήμη ή I/Os. Πάλι θα χρειαστεί να καταναλωθεί χρόνος στον συγχρονισμό τους.
4. Συνάφεια κρυφής μνήμης: Σε multicore συστήματα, κάθε πυρήνας έχει την δική του cache. Όταν ένας πυρήνας τροποποιεί τα δεδομένα στην δική του cache, το σύστημα πρέπει να ανανεώσει ή να κάνει invalidate τα αντίστοιχα entries στις υπόλοιπες caches ώστε να διατηρηθεί η συνάφεια cache. Αυτό προσθέτει περαιτέρω αναμονή αν τα threads επεξεργάζονται τα ίδια blocks.
5. Περιορισμό στο Bandwidth μνήμης: Καθώς αυξάνεται ο αριθμός των πυρήνων, η ζήτηση για πόρους από την μνήμη αυξάνεται οδηγώντας πολλές φορές σε κορεσμό του bandwidth, ειδικά αν όλοι οι πυρήνες προσπαθούν να διαβάσουν/γράψουν στην μνήμη ταυτόχρονα.
6. False Sharing: Περισσότερα νήματα σημαίνει μεγαλύτερη επεξεργασία δεδομένων. Πολλές φορές τα δεδομένα αυτά που χρησιμοποιούνται από διαφορετικά threads βρίσκονται στο ίδιο cache line προκαλώντας updates/invalidations και άρα επιπρόσθετες καθυστερήσεις.
7. Νόμος του Amdahl: Υπάρχουν όρια στο speedup από την παραλληλοποίηση. Το μέγιστο δυνατό speedup καθορίζεται από το ποσοστό του προγράμματος που είναι σειριακό και δεν μπορεί να παραλληλοποιηθεί. Καθώς προσθέτουμε πυρήνες, το σειριακό μέρος του προγράμματος κυριαρχεί στον χρόνο εκτέλεσης, περιορίζοντας το κατορθωτό speedup.

8. Ανισορροπίες φόρτου: Το workload μπορεί να μην γίνεται να κατανεμηθεί ομοιόμορφα μεταξύ των νημάτων, συνεπώς κάποια νήματα θα τελειώσουν την εργασία τους πολύ νωρίς και να παραμείνουν αδρανή ενώ άλλα δουλεύουν.



Παρατηρούμε ότι εν γένει με την αύξηση των πυρήνων έχουμε βελτίωση της επίδοσης του προγράμματος καθώς ελαττώνεται ο χρόνος εκτέλεσης. Ωστόσο η μείωση αυτή δεν είναι ανάλογη με τον αριθμό των πυρήνων που προσθέτουμε. Παρατηρούμε μεγάλη βελτίωση της επίδοσης από τον 1 πυρήνα στους 2, μικρότερη από τους 4 στους 6 και αμελητέα από τους 6 πυρήνες στους 8. Αφού το ταμπλό είναι σχετικά μικρό, υπάρχει περιορισμένη εργασία που μπορεί να παραλληλοποιηθεί και να κατανεμηθεί ανάμεσα σε περισσότερους “εργάτες”. Υπό αυτό το πλαίσιο, από ένα σημείο και μετά καταναλώνεται περισσότερος χρόνος για επικοινωνία και συγχρονισμό μεταξύ νημάτων παρά για ωφέλιμη εργασία.

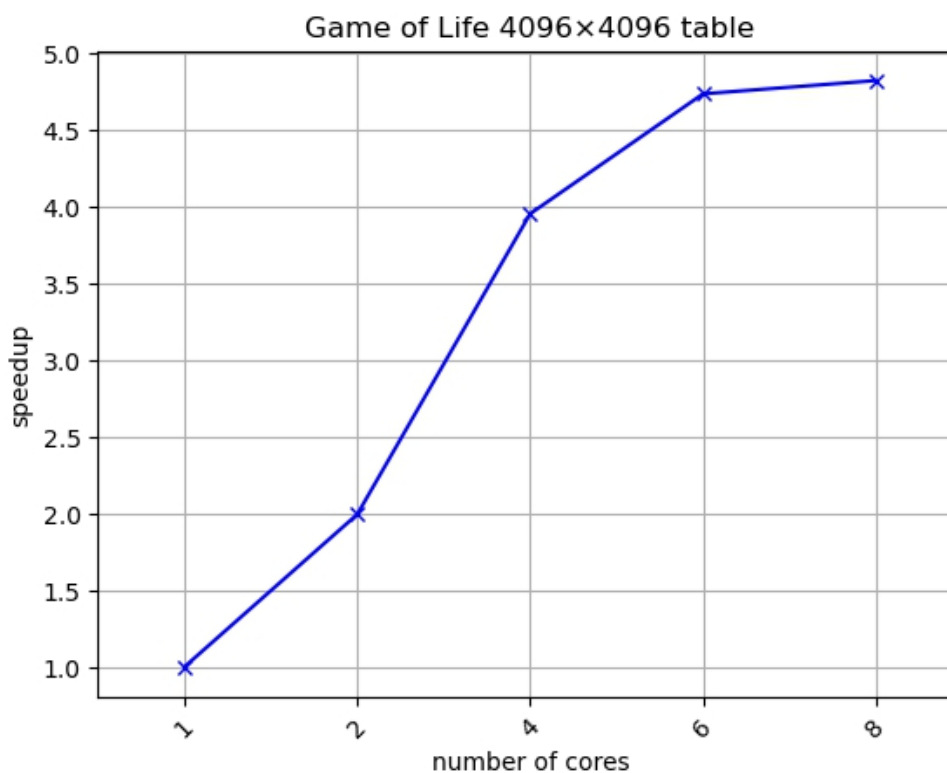
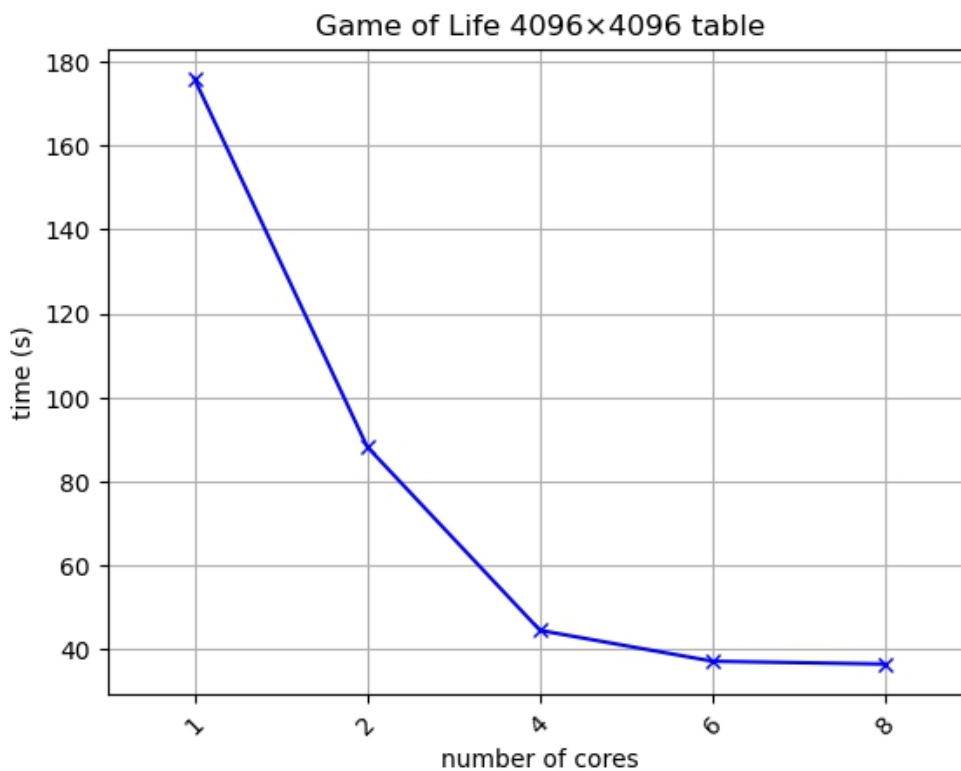
Η καμπύλη συνεπώς έρχεται σε “κορεσμό” και φαίνεται ότι περαιτέρω αύξηση του αριθμού των πυρήνων όχι μόνο δεν θα αυξήσει την απόδοση αλλά ίσως και να τη χειροτερεύσει.



Παρατηρούμε ότι εν γένει με την αύξηση των πυρήνων έχουμε βελτίωση της επίδοσης του προγράμματος καθώς ελαττώνεται ο χρόνος εκτέλεσης. Σε σύγκριση με πριν, πλέον η μείωση του χρόνου (και άρα το speedup) είναι ανάλογη με τον αριθμό των πυρήνων που προσθέτουμε. Παρατηρούμε μεγάλη (καθολική) μείωση του χρόνου εκτέλεσης από τον 1 πυρήνα στους 2, μικρότερη από τους 4 στους 6 και ακόμα μικρότερη από τους 6 πυρήνες στους 8 αλλά (όπως φαίνεται στο speedup όπου έχουμε 2πλάσιο για 2 πυρήνες, 4πλάσιο για 4 πυρήνες ...) η αναλογική μείωση (υποδιπλασιάζεται ο χρόνος εκτέλεσης) είναι η ίδια καθώς προσθέτουμε πυρήνες.

Συνεπώς το ταμπλό με μέγεθος 1024 x 1024 αποτελεί καλή περίπτωση παραλληλισμού. Δεν παρατηρείται “κορεσμός” και άρα προβλέπουμε ότι περαιτέρω αύξηση των πυρήνων θα οδηγήσει σε σημαντική μείωση του χρόνου εκτέλεσης. Αυτό συμβαίνει διότι στο εν λόγω ταμπλό ο χρόνος που χάνεται για την επικοινωνία των νημάτων σε σύγκριση με τον χρόνο που κερδίζουμε από την παραλληλοποίηση είναι μικρός.

Note: Το speedup ορίζεται ως ο λόγος του χρόνου που έκανε το σειριακό πρόγραμμα να εκτελεστεί προς τον χρόνο που έκανε το παράλληλο πρόγραμμα να εκτελεστεί.



Παρατηρούμε ότι εν γένει με την αύξηση των πυρήνων έχουμε βελτίωση της επίδοσης του προγράμματος καθώς ελαττώνεται ο χρόνος εκτέλεσης. Σε σύγκριση με πριν, πλέον η μείωση του χρόνου (και άρα το speedup) είναι ανάλογη με τον αριθμό των πυρήνων που προσθέτουμε μέχρι και τους 4 πυρήνες. Στη συνέχεια έχουμε μικρή αύξηση του speedup. Παρατηρούμε μεγάλη (καθολική) μείωση του χρόνου εκτέλεσης από τον 1 πυρήνα στους 2, μικρότερη από τους 4 στους 6 και αμελητέα από τους 6 πυρήνες στους 8.

Η καμπύλη συνεπώς έρχεται σε “κορεσμό” και φαίνεται ότι περαιτέρω αύξηση του αριθμού των πυρήνων όχι μόνο δεν θα αυξήσει την απόδοση αλλά ίσως και να τη χειροτερεύσει.

Υποθέτουμε ότι ο παραλληλισμός είναι χειρότερος από την περίπτωση μεγέθους 1024 x 1024 διότι το ταμπλό χώραγε στις caches των επεξεργαστών ενώ για το ταμπλό 4096 x 4096 έχουμε περισσότερες προσβάσεις στην γενική μνήμη.