Συστήματα Παράλληλης Επεξεργασίας

Άσκηση 1
Εισαγωγή στην Παραλληλοποίηση σε Αρχιτεκτονικές Κοινής Μνήμης

Ομάδα	**parlab04**
Θεόδωρος Αράπης	el18028
Εμμανουήλ Βλάσσης	el18086
Παναγιώτης Παπαδέας	el18039

Αρχικά συνδεόμαστε στον orion.cslab.ece.ntua.gr, και από εκεί μπορούμε να συνδεθούμε στον εξυπηρετητή scirouter.cslab.ece.ntua.gr. Από τον scirouter στέλνουμε jobs για να εκτελεστούν στο cluster.

Δίνεται το αρχείο Game_Of_Life.c που υλοποιεί το παιχνίδι Game of Life και καλούμαστε να το παραλληλοποιήσουμε στο μοντέλο κοινού χώρου διευθύνσεων (shared address space) με το OpenMP.

Πρώτα συμπεριλαμβάνουμε στο αρχείο τη βιβλιοθήκη προσθέτοντας το αρχείο omp.h με το κατάλληλο include:

```
#include <omp.h>
```

Προκειμένου να παραλληλοποιήσουμε το πρόγραμμα προσθέσαμε μία οδηγία (directive):

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

Και την τοποθετούμε κάτω από το πρώτο for (η επανάληψη των χρονικών βημάτων) και πάνω από το δεύτερο for (η επανάληψη για κάθε γραμμή του πίνακα):

```
        /*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;
                        }
```

Η παραλληλοποίηση δεν μπορεί να γίνει από το πιο πάνω for καθώς υπάρχει εξάρτηση ενός loop από τα προηγούμενα (το ταμπλό χρειάζεται την κατάσταση της προηγούμενης χρονικής στιγμής για να ανανεωθεί).

Η γραμμή αυτή δίνει την οδηγία να μοιραστούν οι επαναλήψεις στα threads. Οι μεταβλητές/πίνακες μέσα στο *shared* είναι κοινές/κοινοί για όλα τα threads. Εδώ, κοινές είναι η μεταβλητή N (οι διαστάσεις του πίνακα), καθώς και οι πίνακες previous και current. Στον πίνακα previous γίνεται μόνο ανάγνωση, ενώ κάθε κελί του πίνακα current αλλάζει τιμή ακριβώς μια φορά (για συγκεκριμένο t που είναι εκτός του parallel for).

Οι μεταβλητές μέσα στο *private* δηλώνουν ότι χρειαζόμαστε νέα μεταβλητή για κάθε thread. Τα i και j αλλάζουν ανάλογα με το κελί με το οποίο ασχολείται το συγκεκριμένο thread, και το nbrs για κάθε επανάληψη αλλάζει ανάλογα με τα i και j. Έτσι, έχουμε ξεχωριστές μεταβλητές για κάθε thread και αποφεύγουμε race conditions.

Αρχικά υποβάλουμε το script make_on_queue.sh στον Torque που εκτελεί τη μεταγλώττιση του κώδικα:

```
#!/bin/bash

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

## Output and error files
#PBS -o make_omp_gol.out
#PBS -e make_omp_gol.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/parlab04/a1
make
```

Το script είναι πρακτικά ίδιο με αυτό που δόθηκε με την αλλαγή του directory στον φάκελο της ομάδας.

Στη συνέχεια υποβάλουμε στον Torque το script run_on_queue.sh που τρέχει το πρόγραμμα με διαφορετικά configurations:

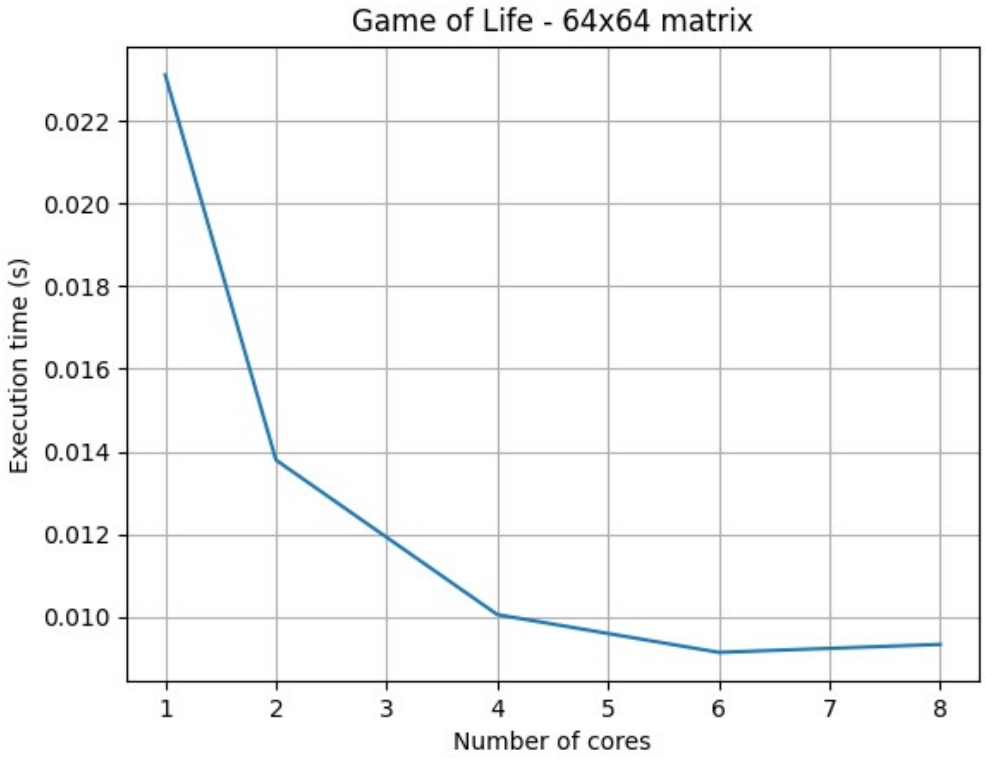
```
#!/bin/bash

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

## Output and error files
#PBS -o run_omp_gol.out
#PBS -e run_omp_gol.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/parlab04/a1
for thr in 1 2 4 6 8
do
        export OMP_NUM_THREADS=${thr}
        ./a1-openmp 64 1000
        ./a1-openmp 1024 1000
        ./a1-openmp 4096 1000
done
```

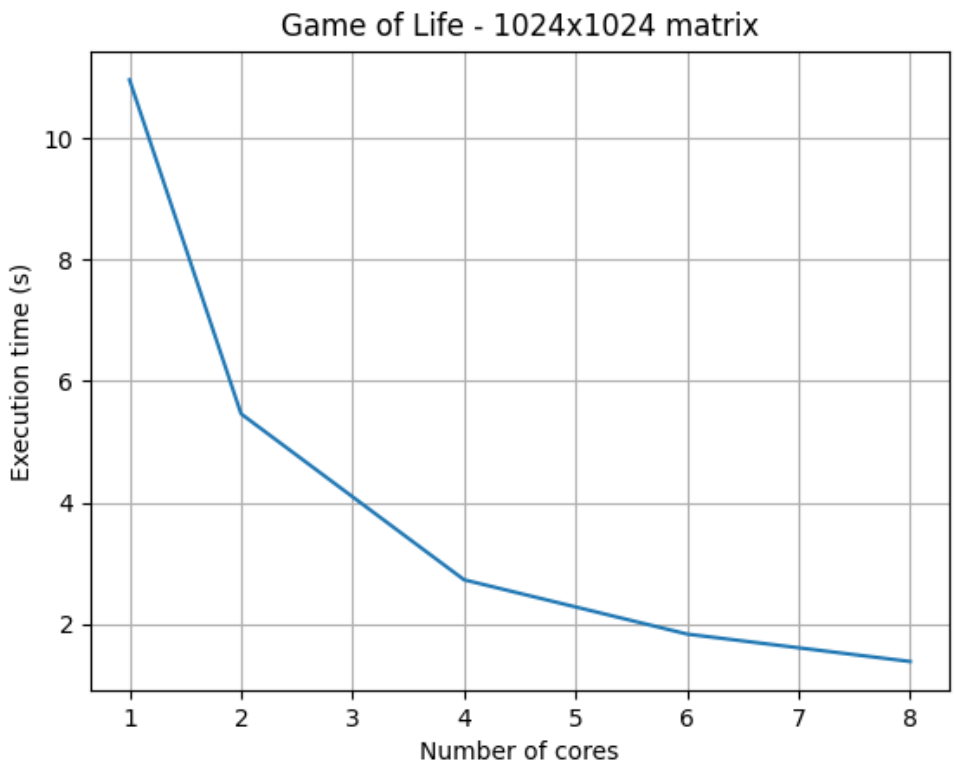
Το OpenMP χρησιμοποιεί το environment variable OMP_NUM_THREADS για να ορίσει τον αριθμό των threads που θα δημιουργήσει. Έτσι, χρησιμοποιώντας ένα απλό bash script τρέχουμε το πρόγραμμα για 1, 2, 4, 6, και 8 πυρήνες σε πίνακες 64x64, 1024x124, και 4096x4096. Ο αριθμός χρονικών βημάτων είναι πάντα 1000.

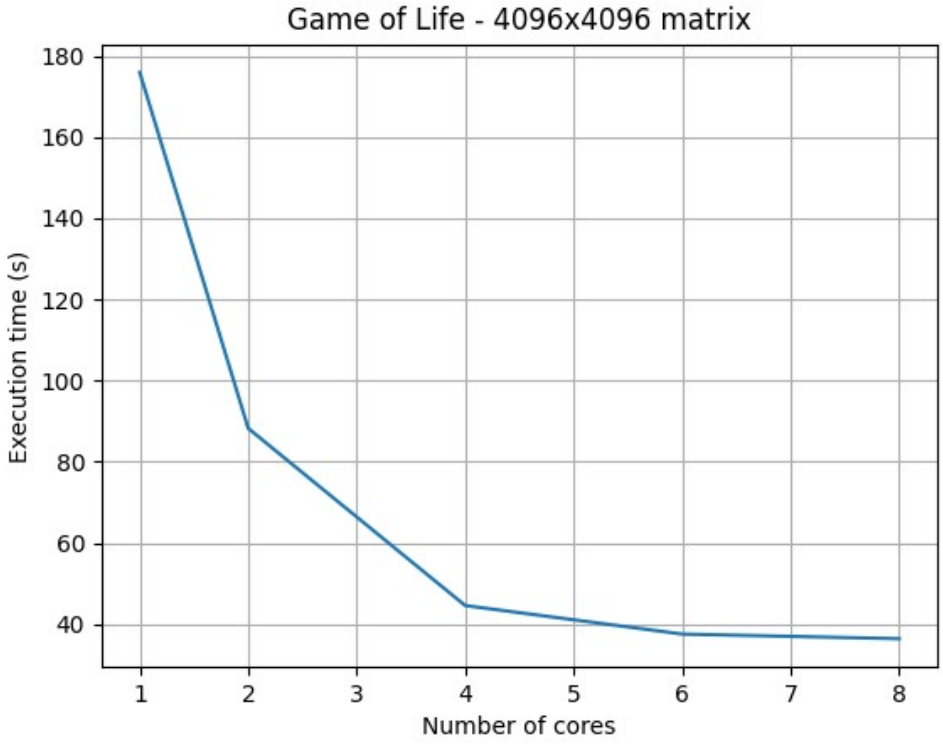
Αφού το job τελειώσει, η έξοδος (στο αρχείο run_omp_gol.out) είναι:

```
parlab04@scirouter:~/a1$ cat run_omp_gol.out
GameOfLife: Size 64 Steps 1000 Time 0.023117
GameOfLife: Size 1024 Steps 1000 Time 10.968734
GameOfLife: Size 4096 Steps 1000 Time 175.948043
GameOfLife: Size 64 Steps 1000 Time 0.013795
GameOfLife: Size 1024 Steps 1000 Time 5.461701
GameOfLife: Size 4096 Steps 1000 Time 88.226745
GameOfLife: Size 64 Steps 1000 Time 0.010052
GameOfLife: Size 1024 Steps 1000 Time 2.724696
GameOfLife: Size 4096 Steps 1000 Time 44.542239
GameOfLife: Size 64 Steps 1000 Time 0.009139
GameOfLife: Size 1024 Steps 1000 Time 1.829604
GameOfLife: Size 4096 Steps 1000 Time 37.515213
GameOfLife: Size 64 Steps 1000 Time 0.009332
GameOfLife: Size 1024 Steps 1000 Time 1.377511
GameOfLife: Size 4096 Steps 1000 Time 36.375931
```

Από τα αποτελέσματα αυτά φτιάχνουμε τα παρακάτω γραφήματα:




Game of Life - 1024x1024 matrix
Execution time (s)
10
8
6
4
2
1
2
3
4
5
6
7
8
Number of cores


Game of Life - 4096x4096 matrix
Execution time (s)
180
160
140
120
100
80
60
40
1
2
3
4
5
6
7
8
Number of cores

Παρατηρήσεις - Σχόλια

Αρχικά, παρατηρούμε ότι σε όλα τα ταμπλό με την αύξηση των πυρήνων που δίνουμε στο πρόγραμμα μειώνεται και ο χρόνος εκτέλεσης. Αυτό σημαίνει ότι η παραλληλοποίηση που προσθέσαμε και το OpenMP δουλεύουν όπως περιμέναμε. Όσον αφορά το μέγεθος του ταμπλό, τα αποτελέσματα που πήραμε είναι επίσης αναμενόμενα. Το ταμπλό 1024x1024 είναι 2^8 = 256 φορές μεγαλύτερο από το ταμπλό 64x64, και βλέπουμε ότι για έναν πυρήνα ο χρόνος εκτέλεσης είναι περίπου 256 φορές μεγαλύτερος. Αντίστοιχα, για έναν πυρήνα ο χρόνος εκτέλεσης του ταμπλό 4096x4096 είναι περίπου 16 φορές μεγαλύτερος από αυτόν του ταμπλό 1024x1024.

Για το ταμπλό 64x64, ο διπλασιασμός των πυρήνων από 1 σε 2 υποδιπλασίασε τον χρόνο εκτέλεσης. Όμως, η αύξηση των πυρήνων παραπέρα δεν φέρνει γραμμική επιτάχυνση. Από 2 σε 4 πυρήνες έχουμε επιτάχυνση 1.37, και από 4 σε 8 πυρήνες επιτάχυνση 1.07. Μάλιστα, από τους 6 στους 8 πυρήνες παρατηρούμε μια μικρή αύξηση του χρόνου εκτέλεσης.

Για το ταμπλό 1024x1024, καταφέρνουμε πρακτικά γραμμική επιτάχυνση σε όλες τις αυξήσεις πυρήνων: από τον 1 στους 2 πυρήνες (επιτάχυνση 2), από τους 2 στους 4 (επιτάχυνση 2), και από τους 4 στους 6 (επιτάχυνση 1.5), και από τους 6 στους 8 (επιτάχυνση 1.33).

Για το ταμπλό 4096x4096, υπάρχει γραμμική επιτάχυνση έως και τους 4 πυρήνες, από τους 4 στους 6 πυρήνες η επιτάχυνση μειώνεται, και η αύξηση στους 8 μειώνει ελάχιστα τον χρόνο εκτέλεσης.

Τα αποτελέσματα που περιγράψαμε παραπάνω είναι αρκετά αναμενόμενα. Είναι λογικό να μην υπάρχει γραμμική επιτάχυνση σε όλες τις αυξήσεις των πυρήνων. Πιθανές εξηγήσεις είναι ο χρόνος για την επικοινωνία μεταξύ των threads, συμφόρηση στον διάδρομο μνήμης με πολλά threads να ζητούν πρόσβαση από την ίδια θέση μνήμης (εδώ ίδια θέση μνήμης θα είναι ένα κελί στον πίνακα previous), ή το κόστος δημιουργίας των threads.

Ο λόγος που έχουμε χειρότερη επιτάχυνση στο μικρότερο ταμπλό είναι ότι είναι πιο δύσκολο ο χρόνος που κερδίζουμε από την παραλληλοποίηση να αντισταθμίσει το κόστος δημιουργίας των threads και της επικοινωνίας μεταξύ τους.