05/11/2019

Συστήματα Παράλληλης Επεξεργασίας

Σειρά 1η - Ενδιάμεση Αναφορά

Ομάδα: parlab30
Τσαγκαράκης Στυλιανός - 03115180
Τσάκας Νικόλαος - 03115433

Άσκηση 1η

Ερώτημα 1.4

Μετά από δοκιμές, καταλήξαμε να παραλληλοποιούμε το game of life στο εξωτερικό (i) for loop κάθε γενιάς:

```
for ( t = 0 ; t < T ; t++ )
{
	#ifdef PARALLEL
	#pragma omp parallel for shared(current, previous) private(i,j) firstprivate(nbrs)
	#endif
	for ( i = 1 ; i < N-1 ; i++ )
	{
		for ( j = 1 ; j < N-1 ; j++ )
		{
			//Check neighbors and live or die
		}
}
```

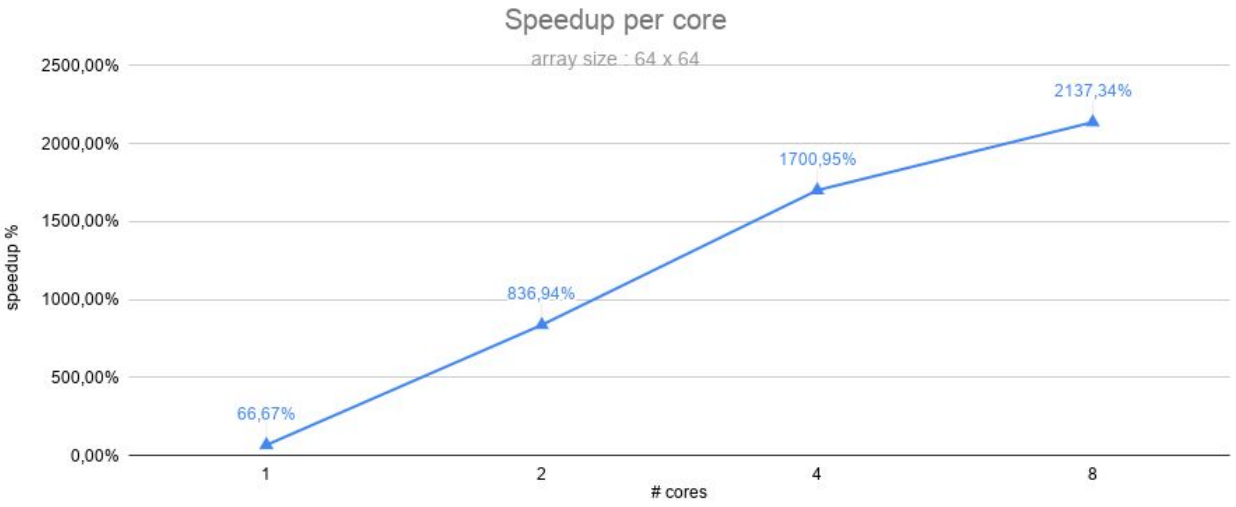
Η επιλογή δίνει συχνά καλύτερα αποτελέσματα καθώς η επεξεργασία μίας ολοκληρης στήλης του πίνακα από τον ίδιο πυρήνα κάνει καλύτερη εκμετάλλευση της τοπικότητας της μνήμης.

Επίσης παραλληλοποιήσαμε την αρχικοποίηση του πίνακα για ακόμα μεγαλύτερη επιτάχυνση του προγράμματος:

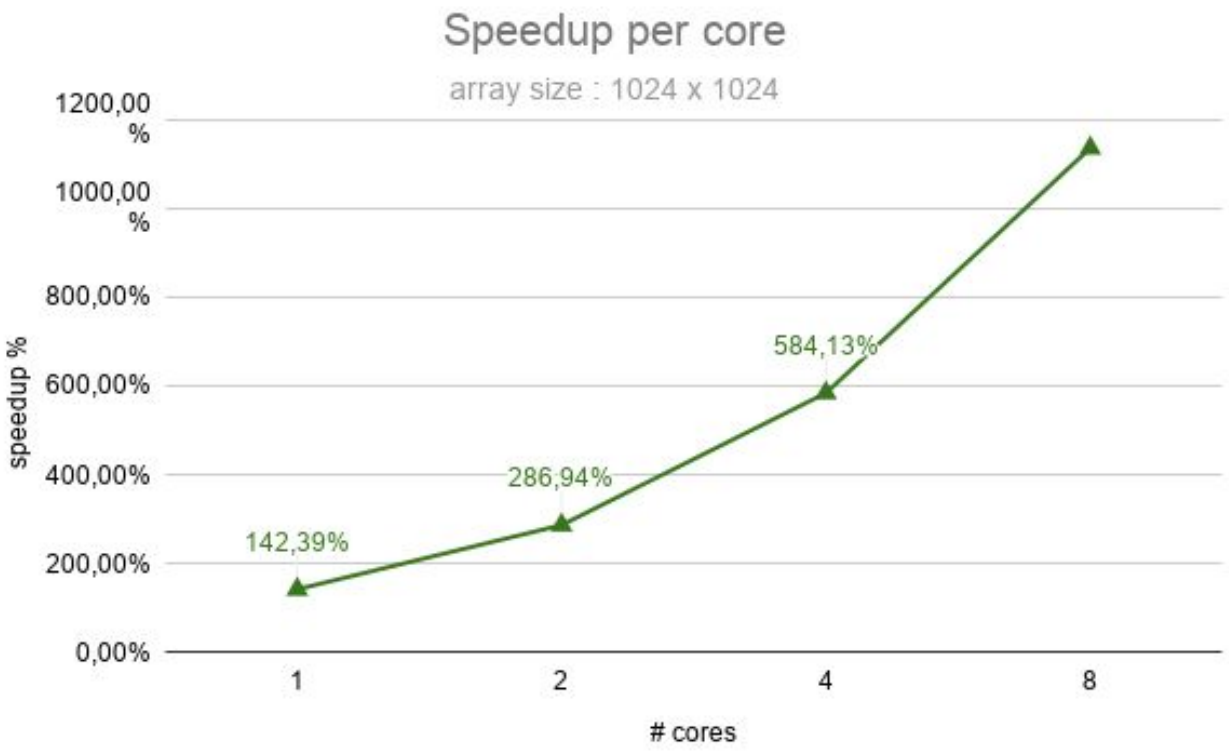
```
int ** allocate_array(int N) {
	/*
	 * malloc rows here
	 */
	#ifdef PARALLEL
	#pragma omp parallel private(i,j)
	#endif
	{
		for ( i = 0; i < N ; i++ )
			for ( j = 0; j < N ; j++ )
				array[i][j] = 0;
	}
	return array;
}
```

Η επιλογή αυτή δίνει καλύτερα αποτελέσματα γιατί ο πίνακας αρχικοποιείται με μια “data centric αντιμετώπιση”.

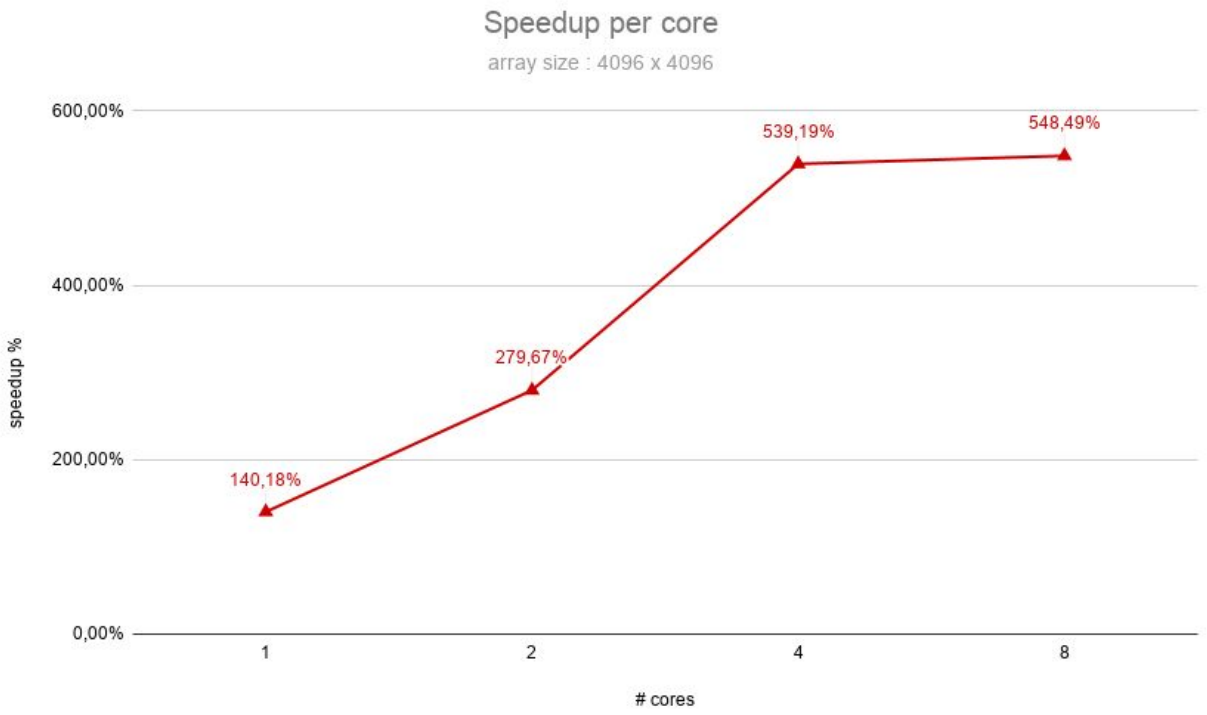
Τα γραφήματα που προέκυψαν από τις εκτελέσεις του script παρατίθενται παρακάτω:



Γράφημα 1

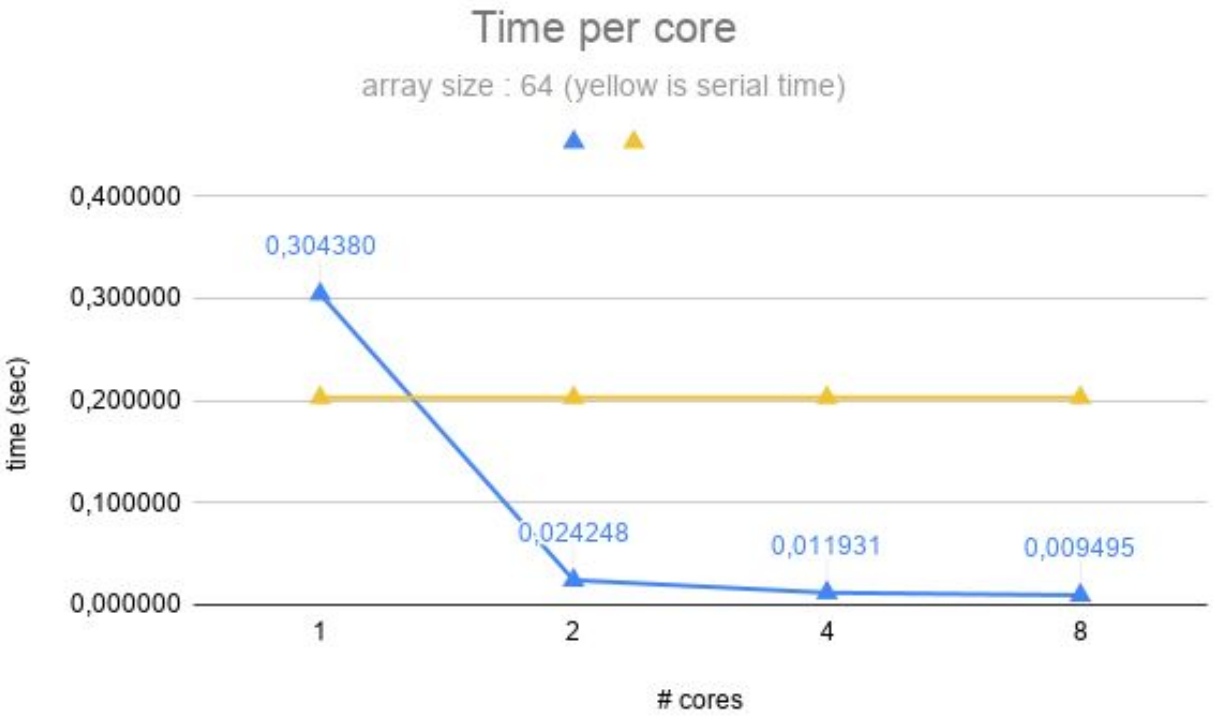


Γράφημα 2

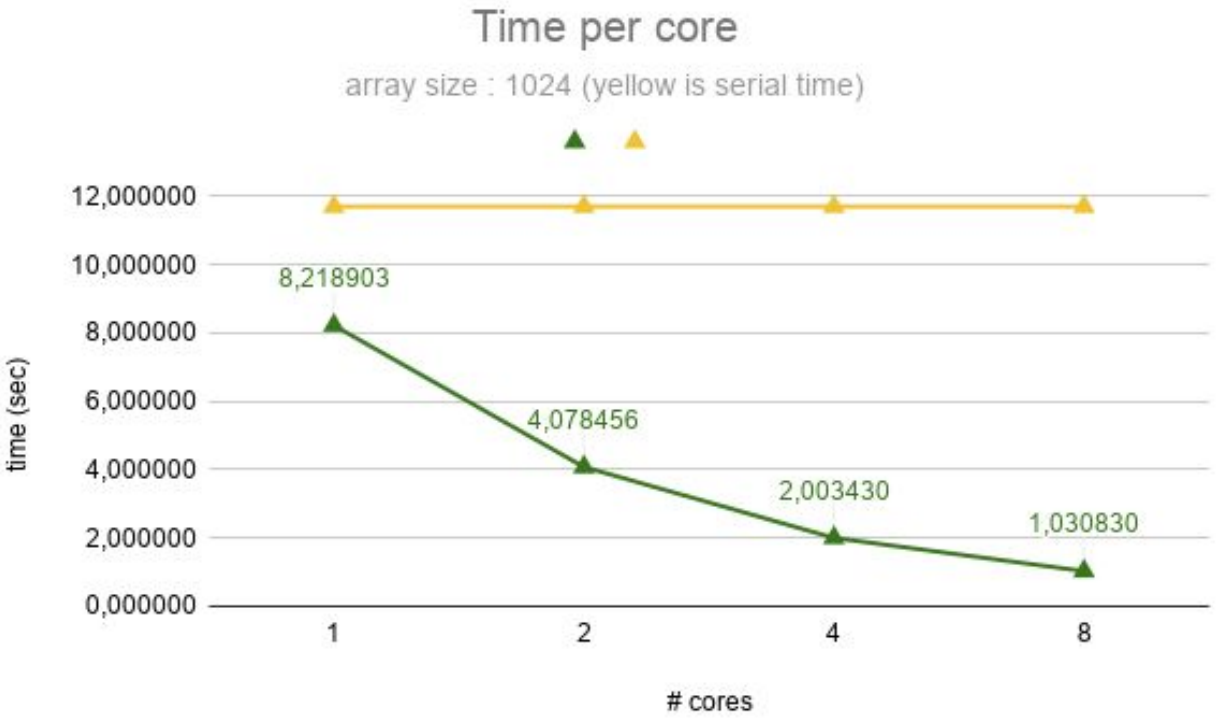


Γράφημα 3

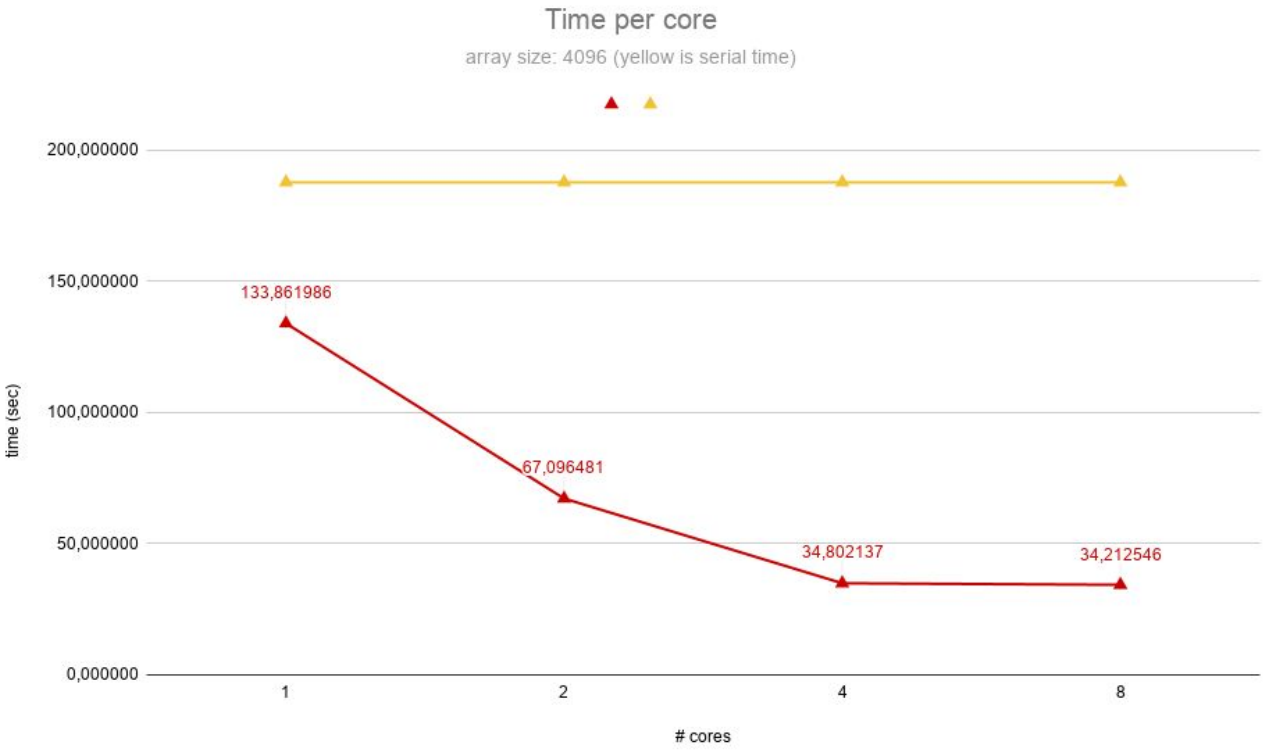
Γράφημα 4



Γράφημα 5



Γράφημα 6



Γράφημα 7

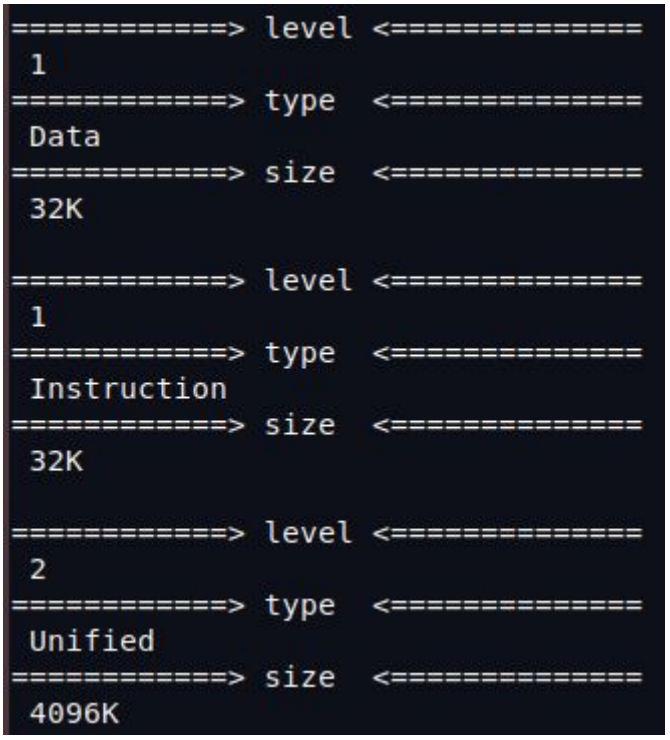
Παρατηρούμε τα εξής:

- Σε όλα τα γραφήματα (εκτός από Γράφημα 5, βλ. παρακάτω) πως υπάρχει μια ξεκάθαρη επιτάχυνση σε σχέση με κάθε προηγούμενη εκτέλεση με λιγότερους πυρήνες.
- Στα γραφήματα χρόνου πως υπάρχει περίπου η ίδια "εκθετική (φθίνουσα)" μορφή στα αντίστοιχα τμήματα (1-2, 2-4, 4-8). Αυτό μας δείχνει ότι άσχετα από το μέγεθος του πίνακα εισόδου, το speedup είναι κοινό σε κάθε διπλασιασμό πυρήνων.
- Στο Γράφημα 5 (array size : 64) παρατηρούμε ότι με στην εκτέλεση του παράλληλου προγράμματος με 1 πυρήνα έχουμε μια μικρή καθυστέρηση της τάξεως του 30%. Αυτό πιθανότατα συμβαίνει γιατί μέσα στον κώδικα της παράλληλης εκτέλεσης έχουμε κάποια directives (#pragma omp parallel for …) τα οποία πρέπει να τα εκτελέσει το thread που τρέχει. Στον κώδικα της σειριακής υλοποίησης δεν υπάρχει κάτι τέτοιο και η εκτέλεση συνεχίζεται ομαλά.
- Στο Γράφημα 7 (array size : 4096) υπάρχει μικρή επιτάχυνση μεταξύ των 4 πυρήνων και των 8 πυρήνων. Αυτό πιθανότατα οφείλεται στο μέγεθος της cache. Υποβάλλωντας ένα σχετικό script στην ουρά, βρίσκουμε το μέγεθος της cache του εκάστοτε cloneX που τρέχουμε.
  Το script περιέχει:

```
module load openmp

for d in /sys/devices/system/cpu/cpu0/cache/index*;
       do tail -c+1 $d/{level,type,size}
       echo
done
```

οπότε και βγαίνει το παρακάτω αποτέλεσμα:



Παρατηρούμε λοιπόν πως η cache έχει L1 = 32KB και L2 = 4096KB για δεδομένα.
Στις πρώτες 2 περιπτώσεις το μέγεθος πίνακα 64x64 και 1024x1024 συνολικά δηλαδή 64x64x4bytes = 16Kbytes και 1024x1024x4bytes = 4096KB. Αυτά ξεκάθαρα χωράνε στην cache οπότε είναι λογικό το σχεδόν διπλάσιο speedup.
Στην περίπτωση του πίνακα μεγέθους 4096x4096 έχουμε συνολικά δεδομένα 4096x4096x4bytes = 67108KB περίπου 16 φορές περισσότερα από την χωρητικότητα της cache. Συνεπώς φαίνεται πως εκεί υπάρχει μεγάλη καθυστέρηση και το speedup δεν είναι το αναμενόμενο.

Άσκηση 2η

Ερώτημα 2.4.1

α) Για την αρχική σειριακή παραλληλοποίηση του FW η στρατηγική μας είναι η ίδια με το game of life δηλαδή μία task centric προσέγγιση όπου κατανέμεται σε κάθε thread μία επανάληψη for loop. Θεωρούμε συμφέρον για καλύτερη εκμετάλλευση της κρυφής μνήμης όπως και στο game of life να παραλληλοποιήσουμε το i-for loop ώστε κάθε επεξεργαστής να διαχειρίζεται σε αποκλειστικότητα ένα συνεχόμενο μέρος του πίνακα.

β) Τόσο για την recursive όσο και για την tiled εκδοχή μία data centric προσέγγιση φαίνεται προφανής. Η στρατηγική μας είναι να κατανείμουμε διεργασίες που έχουν ορισμένη τοπικότητα στον πίνακα (στην αναδρομική περίπτωση τις 8 αναδρομικές κλήσεις και στην tiled τα for loops που τρέχουν τον FW στα tiles στα οποία είναι διαχωρισμένος ο πίνακας). Δεδομένου ότι η παράμετρος B είναι και στις δύο περιπτωσεις σεβαστά μικρότερη από το N, η παραλληλοποίηση των for loops του FW όταν τρέχει σε μικρές περιοχές φαίνεται να δημιουργεί overhead χωρίς ιδιαίτερο κέρδος και ιδιαίτερα αν συνδυαστεί με διαχωρισμό σε sections το hardware θα αναλώνεται σε πόρους χωρίς να εκτελεί ιδιαίτερο έργο.