

**ΛΟΓΙΣΜΙΚΟ & ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΥΣΤΗΜΑΤΩΝ**

**ΥΨΗΛΗΣ ΕΠΙΔΟΣΗΣ**

**ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ 2014-2015**

**ΥΛΟΠΟΙΗΣΗ ΠΟΛΛΑΠΛΑΣΙΑΣΜΟΥ ΜΗΤΡΩΟΥ ΕΠΙ ΔΙΑΝΥΣΜΑ**

**ΣΕ ΠΕΡΙΒΑΛΛΟΝ CUDA**

**Μέλη ομάδας: Γώγουλου Ευαγγελία (ΑΜ: 5284)**

**Πατρώνη Σωτηρία (ΑΜ: 5399)**

Οι κώδικες των ακόλουθων ερωτημάτων εκτελέστηκαν στη κάρτα γραφικών του εργαστηρίου, η οποία έχει τα ακόλουθα χαρακτηριστικά:

```
hpc14-15-187@scgroup11:~  
CUDA Device #0  
Major revision number:      1  
Minor revision number:      3  
Name:                        Tesla C1060  
Total global memory:         4294770688  
Total shared memory per block: 16384  
Total registers per block:    16384  
Warp size:                    32  
Maximum memory pitch:        2147483647  
Maximum threads per block:    512  
Maximum dimension 0 of block: 512  
Maximum dimension 1 of block: 512  
Maximum dimension 2 of block: 64  
Maximum dimension 0 of grid:  65535  
Maximum dimension 1 of grid:  65535  
Maximum dimension 2 of grid:  1  
Clock rate:                   1296000  
Total constant memory:         65536  
Texture alignment:             256  
Concurrent copy and execution: Yes  
Number of multiprocessors:     30  
Kernel execution timeout:      No
```

### **ΕΡΩΤΗΜΑ 1:**

Στο ερώτημα αυτό έγινε υλοποίηση του πολλαπλασιασμού μητρώου με διάνυσμα με χρήση της συνάρτησης `cublasDgemv` της βιβλιοθήκης `cuBLAS`. Οι χρόνοι εκτέλεσης των υπολογισμών για διάφορα μεγέθη παρατίθενται στο τέλος της αναφοράς.

### **ΕΡΩΤΗΜΑ 2:**

Στο ερώτημα αυτό έγινε μία απλή υλοποίηση του πολλαπλασιασμού μητρώου επί διάνυσμα σε `CUDA`. Η υλοποίηση αυτή περιλαμβάνει:

- Δέσμευση καθολικής μνήμης για την αποθήκευση τόσο των τελεστών όσο και του αποτελέσματος.
- Δημιουργία ενός μονοδιάστατου πλέγματος, όπου κάθε μπλοκ έχει 512 threads.
- Μεταφορά των δεδομένων από τη CPU στη GPU.

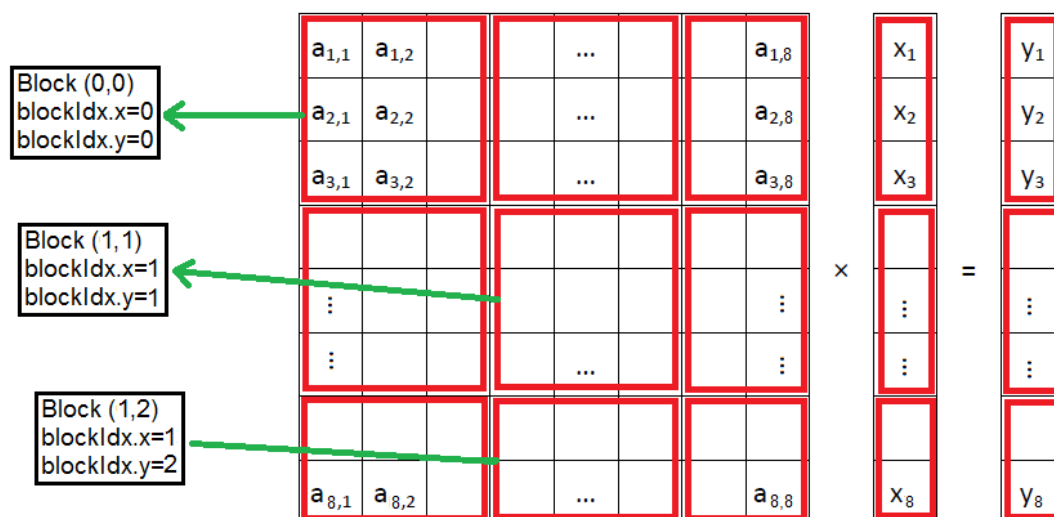
- Εκτέλεση υπολογισμών: κάθε thread υπολογίζει ένα εσωτερικό γινόμενο γραμμής του μητρώου επί το διάνυσμα, δηλαδή ένα στοιχείο του αποτελέσματος.
- Μεταφορά του αποτελέσματος από τη GPU στη CPU.

Οι χρόνοι εκτέλεσης των υπολογισμών για διάφορα μεγέθη παρατίθενται στο τέλος της αναφοράς. Όπως ήταν αναμενόμενο, η υλοποίηση αυτή δεν είναι καθόλου αποδοτική, ειδικά για μητρώα μεγάλου μεγέθους. Αυτό οφείλεται στο γεγονός ότι δεν γίνεται εκμετάλλευση ούτε της ιεραρχίας μνήμης της CUDA, ούτε των υπόλοιπων δυνατοτήτων της.

### **ΕΡΩΤΗΜΑ 3**

Κατά την εκκίνηση του πυρήνα δημιουργείται ένα δυσδιάστατο πλέγμα το οποίο αποτελείται από blocks μίας διάστασης με 512 threads το καθένα. Κατά κανόνα, κάθε block έχει διαστάσεις  $BLOCK\_HEIGHT=64$  και  $BLOCK\_WIDTH=512$ . Ο λόγος επιλογής των παραπάνω μεγεθών εξηγείται στη συνέχεια. Εξαίρεση των παραπάνω αποτελούν τα οριακά block, οι διαστάσεις των οποίων ενδέχεται να είναι μικρότερες από  $BLOCK\_HEIGHT$  και  $BLOCK\_WIDTH$ . Αυτό μπορεί να συμβεί στη περίπτωση που η πλοκαδοποίηση του μητρώου, σε αντιστοιχία με τα blocks των threads, δεν είναι ακριβής. Τότε, το ύψος και το πλάτος του block υπολογίζονται αντίστοιχα ως η διαφορά ύψους μητρώου μείον τη γραμμή εκκίνησης του τελευταίου block, ή αντίστοιχα πλάτος μητρώου μείον τη στήλη εκκίνησης του τελευταίου block. Εφόσον, λοιπόν, οι διαστάσεις του block ενδέχεται να διαφοροποιηθούν, είναι απαραίτητο να υπάρχουν μεταβλητές για την αποθήκευση των διαστάσεων του κάθε block. Οι μεταβλητές αυτές είναι οι `blockheight` και `blockwidth`, οι οποίες ορίζονται στο πυρήνα ως `shared`, ώστε να είναι κοινές για όλα τα threads του ίδιου block. Οι διαστάσεις του πλέγματος είναι  $r$  γραμμές και  $c$  στήλες, όπου  $r = \lceil \text{ύψος μητρώου} / BLOCK\_HEIGHT \rceil$  και  $c = \lceil \text{πλάτος μητρώου} / BLOCK\_WIDTH \rceil$ . Κάθε block του πλέγματος, εκτός των μεταβλητών `blockIdx.x`, `blockIdx.y`, χαρακτηρίζεται και από ένα μοναδικό ζεύγος τιμών `blockx`, `blocky`, ώστε να υπάρχει μία αντιστοίχιση με το block στοιχείων του μητρώου. Η τιμή της μεταβλητής `blockx` αντιστοιχεί στον αριθμό της πρώτης στήλης του

εκάστοτε block του μητρώου, τα στοιχεία της οποίας προσπελαύνει το block αυτό. Η τιμή του blocky υπολογίζεται με τον ίδιο τρόπο για τις γραμμές. Στο σημείο αυτό πρέπει να σημειωθεί ότι θεωρήθηκε ως δεδομένο ότι η προσπέλαση του μητρώου γίνεται κατά στήλες (Ο λόγος της θεώρησης αυτής θα εξηγηθεί παρακάτω). Με βάση τη θεώρηση αυτή, τα threads κάθε block του πλέγματος υπολογίζουν συνολικά blockwidth εσωτερικά γινόμενα, χρησιμοποιώντας το καθένα blockheight στοιχεία του διανύσματος. Η αντιστοίχιση κάθε μπλοκ των threads σε κάθε μπλοκ στοιχείων του μητρώου απεικονίζεται στο ακόλουθο σχήμα:



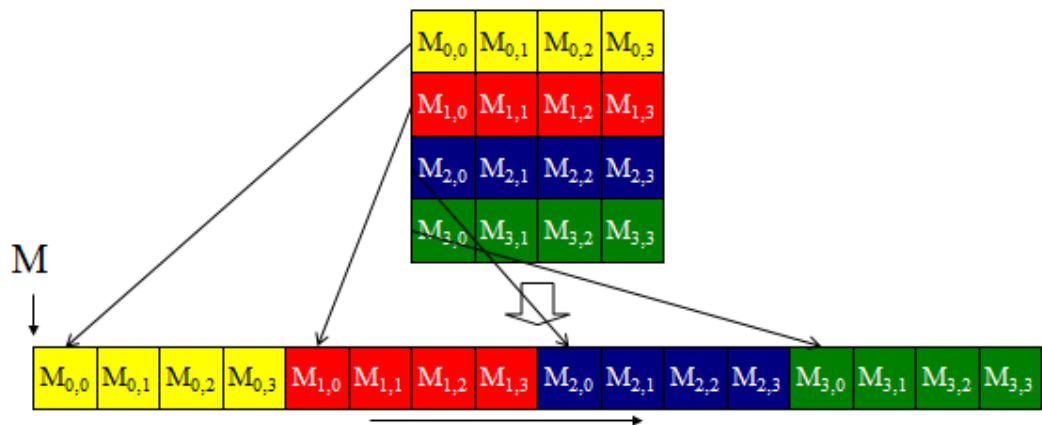
Κατά τη παραλληλοποίηση χρησιμοποιήθηκαν οι παρακάτω τεχνικές:

- Χρήση κοινής μνήμης  
Παρατηρήθηκε ότι για τον υπολογισμό του τελικού αποτελέσματος γίνεται επαναχρησιμοποίηση των στοιχείων του διανύσματος τόσες φορές, όσα και τα εσωτερικά γινόμενα που υπολογίζονται. Αντί, λοιπόν, να απαιτείται κάθε φορά προσπέλαση της καθολικής μνήμης της κάρτας γραφικών, τα ζητούμενα στοιχεία προσπελαύνονται από τη κοινή μνήμη, η προσπέλαση της οποίας είναι αρκετά πιο γρήγορη. Η φόρτωση των ζητούμενων στοιχείων στη κοινή μνήμη γίνεται κατά την εκκίνηση του πυρήνα. Ειδικότερα, φορτώνονται blockheight στοιχεία στη κοινή μνήμη από κάθε block του πλέγματος, ένα στοιχείο από κάθε thread. Το ποια στοιχεία του διανύσματος

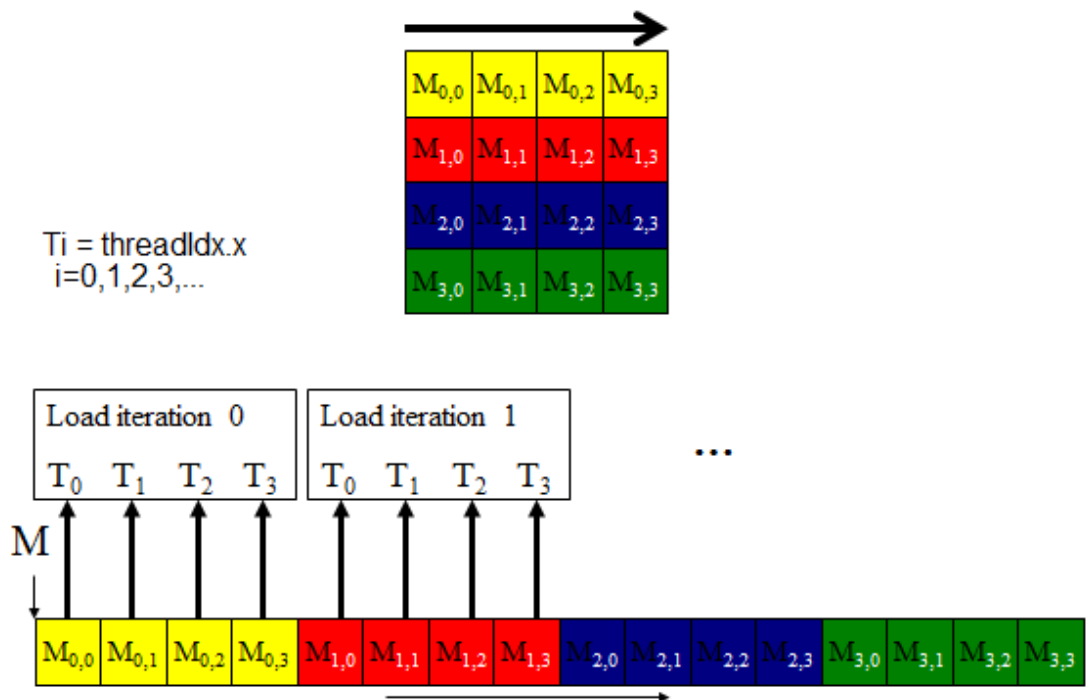
χρειάζεται κάθε block του πλέγματος προκύπτει εύκολα με βάση την ακόλουθη παρατήρηση : τα threads των blocks με  $\text{blockIdx.y} = n$  χρησιμοποιούν το  $n$ -οστό block στοιχείων του διανύσματος στον υπολογισμό των εσωτερικών γινομένων.

- Τεχνική συγκερασμού μνήμης (coalesced memory)

Όπως είναι γνωστό από τη θεωρία, ο χρόνος προσπέλασης της καθολικής μνήμης της κάρτας γραφικών είναι σχετικά μεγάλος. Παράλληλα, με κάθε αίτηση ανάγνωσης από τη μνήμη, δεν επιστρέφεται μόνο το ζητούμενο στοιχείο, αλλά και ένα σύνολο γειτονικών στοιχείων αυτού. Συνεπώς, για να μπορέσει ο πυρήνας να φτάσει το μέγιστο εύρος ζώνης της καθολικής μνήμης, θα πρέπει threads με διαδοχικό  $\text{threadIdx.x}$  να προσπελαίνουν διαδοχικά στοιχεία στη μνήμη. Με τον τρόπο αυτό, γίνεται συγκερασμός του υλικού και με μία αίτηση για προσπέλαση της καθολικής μνήμης επιστρέφονται πολλά διαδοχικά στοιχεία. Προκειμένου να εφαρμοστεί η τεχνική του συγκερασμού μνήμης στο συγκεκριμένο υπολογισμό, έγινε η θεώρηση της προσπέλασης του μητρώου κατά στήλες, ενώ η αποθήκευσή του στη μνήμη γίνεται κατά γραμμές. Συνεπώς, κάθε στοιχείο του τελικού αποτελέσματος προκύπτει ως εσωτερικό γινόμενο στήλης του μητρώου επί το διάνυσμα. . Με βάση αυτή τη θεώρηση, κάθε thread προσπελαύνει  $\text{blockheight}$  στοιχεία στήλης του μητρώου, ενώ και τα 512 threads κάθε block του πλέγματος προσπελαύνουν γειτονικά στοιχεία στη μνήμη σε κάθε βήμα υπολογισμού του εσωτερικού γινομένου. Στο παρακάτω σχήμα φαίνεται πως απεικονίζεται το μητρώο στη μνήμη:



Παρακάτω απεικονίζεται η προσπέλαση των threads με διαδοχικό threadIdx.x σε διαδοχικές θέσεις μνήμης:



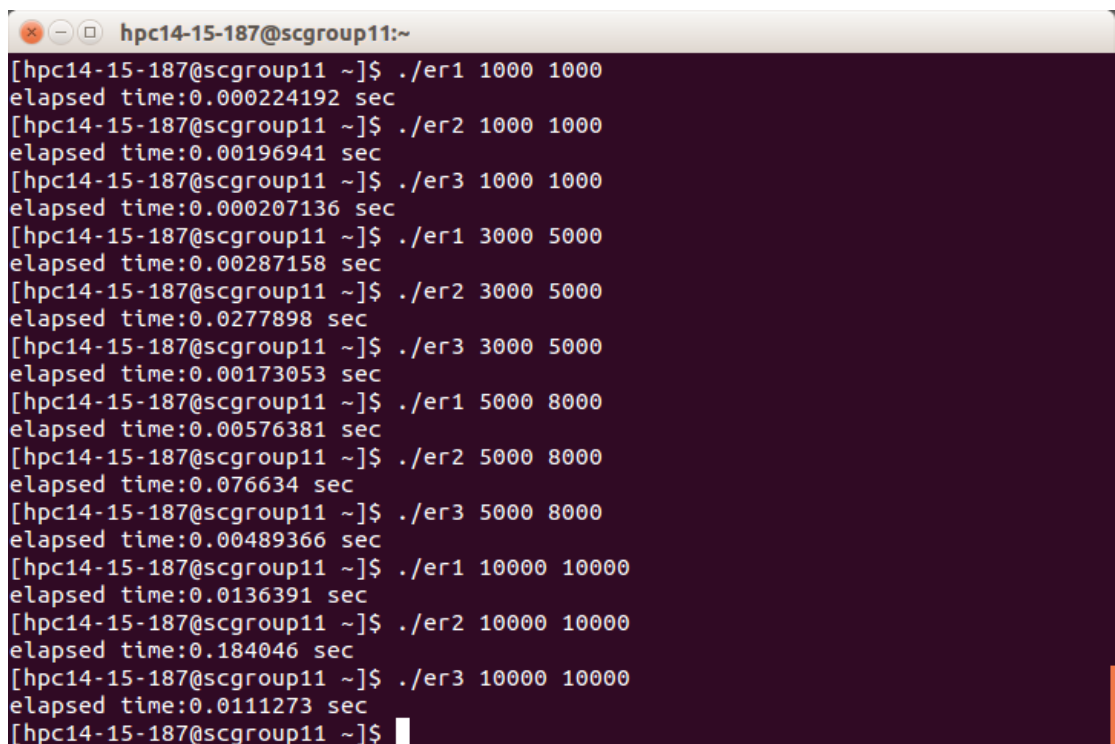
- Χρήση ατομικών εντολών

Όταν όλα τα threads έχουν ολοκληρώσει τους υπολογισμούς των μερικών εσωτερικών γινομένων, θα πρέπει να γίνει πρόσθεση των μερικών αποτελεσμάτων. Η πρόσθεση θα γίνει μεταξύ των τιμών που έχουν υπολογιστεί από threads τα οποία προσπελαίνουν στοιχεία ίδιας στήλης του μητρώου. Επειδή όμως τα threads εκτελούνται παράλληλα, θα πρέπει να υπάρχει κάποια μορφή συγχρονισμού. Έτσι θα πρέπει να αντιμετωπιστεί το

πρόβλημα της ταυτόχρονης προσπάθειας εγγραφής στην ίδια θέσης μνήμης από πολλά threads. Στη παρούσα υλοποίηση, επιλέχθηκε η χρήση της ατομικής εντολής πρόσθεσης. Επειδή η CUDA δεν παρέχει ατομική συνάρτηση πρόσθεσης αριθμών διπλής ακρίβειας, έγινε υλοποίηση τέτοιας συνάρτησης με το όνομα `atomicAdd` (Η υλοποίηση αυτή παρέχεται από την NVidia).

### Screenshots με χρόνους εκτέλεσης των ερωτημάτων

Παρακάτω απεικονίζονται οι χρόνοι εκτέλεσης των παραπάνω ερωτημάτων για διαφορετικές διαστάσεις μητρώων.



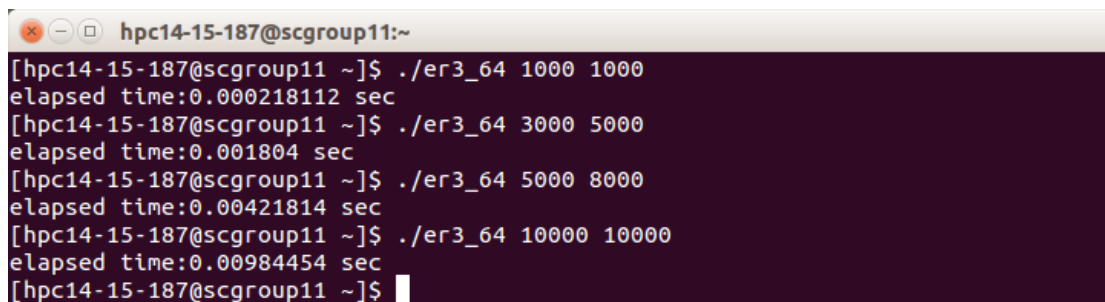
```
hpc14-15-187@scgroup11:~$ ./er1 1000 1000
elapsed time:0.000224192 sec
hpc14-15-187@scgroup11 ~]$ ./er2 1000 1000
elapsed time:0.00196941 sec
hpc14-15-187@scgroup11 ~]$ ./er3 1000 1000
elapsed time:0.000207136 sec
hpc14-15-187@scgroup11 ~]$ ./er1 3000 5000
elapsed time:0.00287158 sec
hpc14-15-187@scgroup11 ~]$ ./er2 3000 5000
elapsed time:0.0277898 sec
hpc14-15-187@scgroup11 ~]$ ./er3 3000 5000
elapsed time:0.00173053 sec
hpc14-15-187@scgroup11 ~]$ ./er1 5000 8000
elapsed time:0.00576381 sec
hpc14-15-187@scgroup11 ~]$ ./er2 5000 8000
elapsed time:0.076634 sec
hpc14-15-187@scgroup11 ~]$ ./er3 5000 8000
elapsed time:0.00489366 sec
hpc14-15-187@scgroup11 ~]$ ./er1 10000 10000
elapsed time:0.0136391 sec
hpc14-15-187@scgroup11 ~]$ ./er2 10000 10000
elapsed time:0.184046 sec
hpc14-15-187@scgroup11 ~]$ ./er3 10000 10000
elapsed time:0.0111273 sec
hpc14-15-187@scgroup11 ~]$
```

Αναλυτικά οι χρόνοι για κάθε ερώτημα και κάθε διάσταση απεικονίζονται και στον παρακάτω πίνακα:

Ερώτημα	Διαστάσεις μητρώου			
	1000x1000	3000x5000	5000x8000	10000x10000
1 <sup>ο</sup>	0.000224192 sec	0.00287158 sec	0.00576381 sec	0.0136391 sec
2 <sup>ο</sup>	0.00196941 sec	0.0277898 sec	0.076634 sec	0.184046 sec
3 <sup>ο</sup>	0.000207136 sec	0.00173053 sec	0.00489366 sec	0.0111273 sec

Όπως παρατηρείται από τον παραπάνω πίνακα, ο κώδικας του ερωτήματος 3 είναι ο ταχύτερος από όλους, και αυτό λόγω των παραπάνω τεχνικών παραλληλοποίησης που χρησιμοποιήθηκαν.

Επίσης, ο κώδικας του 3<sup>ου</sup> ερωτήματος τροποποιήθηκε με διαστάσεις block διαφορετικές από του κώδικα που εκτελέστηκε παραπάνω (από 128x512 σε 64x512). Η μόνη διαφορά είναι στη μεταβλητή BLOCK\_HEIGHT η οποία τέθηκε ίση με 64 (ενώ στον κώδικα που εκτελέστηκε παραπάνω είναι ίση με 128, αρχείο er3\_128.cu). Το αρχείο του κώδικα με BLOCK\_HEIGHT=64 ονομάζεται er3\_64.cu και παρακάτω απεικονίζονται τα αποτελέσματα της εκτέλεσής του:



```
hpc14-15-187@scgroup11:~  
[hpc14-15-187@scgroup11 ~]$ ./er3_64 1000 1000  
elapsed time:0.000218112 sec  
[hpc14-15-187@scgroup11 ~]$ ./er3_64 3000 5000  
elapsed time:0.001804 sec  
[hpc14-15-187@scgroup11 ~]$ ./er3_64 5000 8000  
elapsed time:0.00421814 sec  
[hpc14-15-187@scgroup11 ~]$ ./er3_64 10000 10000  
elapsed time:0.00984454 sec  
[hpc14-15-187@scgroup11 ~]$
```

Παρατηρούνται αμελητέες διαφορές συγκριτικά με μεγέθη block 128x512 αλλά σε μεγαλύτερα μεγέθη μητρώων η διαφορά γίνεται πιο αισθητή.