**ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ**

**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΤΕΛΙΚΗ ΑΝΑΦΟΡΑ για το μάθημα**

# ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ ΓΙΑ ΠΛΗΡΟΦΟΡΙΑΚΑ ΣΥΣΤΗΜΑΤΑ

**ΠΑΝΑΓΙΩΤΗΣ ΓΕΩΡΓΑΚΟΠΟΥΛΟΣ - 1115201600028**

**ΜΑΡΙΑ ΚΑΡΑΜΗΝΑ – 1115201600059**

**ΓΕΩΡΓΙΟΣ ΚΟΥΡΣΙΟΥΝΗΣ - 1115201600077**

## 1. Γενική περιγραφή λειτουργίας

### Α. Υλοποίηση πολυνηματισμού

Ο πολυνηματισμός έγινε με χρήση πολλών νημάτων (τα οποία αποφασίζει ο χρήστης) και ενός job scheduler όπως ζητήθηκε από την εκφώνηση. Ο scheduler κρατά μία ουρά από jobs (λίστα πινάκων με δείκτες σε jobs παρόμοια της λίστας αποτελεσμάτων του πρώτου μέρους O(1) append/pop) από την οποία τα threads μπορούν να παίρνουν το επόμενο job προς εκτέλεση και να το εκτελούν. Ένα job αποτελείται από τις παραμέτρους που χρειάζεται για να εκτελεστεί (void*) και δύο συναρτήσεις:

1) run: Η ρουτίνα κώδικα που θα εκτελέσει το νήμα (λεπτομέρειες παρακάτω).

2) destroy: χρησιμοποιείται για αποδέσμευση πόρων.

Στην υλοποίηση μας υπάρχει πλήρης παραλληλία (με εξαίρεση την φόρτωση των σχέσεων/ερωτημάτων και τα κατηγορήματα τύπου hidden self-join π.χ. 0.1=1.0&0.1=2.1&0.2=2.2 όπου η παραλληλοποίηση είναι δύσκολη και έχει μεγάλο κόστος).

Τα jobs είναι φτιαγμένα με τέτοιο τρόπο ώστε κάθε νήμα να δουλεύει ανεξάρτητα των άλλων και να μην περιμένει κάποιο άλλο να τελειώσει την δουλεία του ώστε να συνεχίσει, οι καθυστερήσεις να είναι ελάχιστες και μόνο δύο ειδών:

α) Ένα νήμα αν έχει τελειώσει μία δουλειά και η ουρά των jobs είναι κενή τότε περιμένει.

β) Ένα νήμα μπορεί να περιμένει για να αποκτήσει πρόσβαση σε μία κοινόχρηστη δομή που προστατεύεται με έναν mutex (οι mutexes χρησιμοποιούνται όσο λιγότερο γίνεται και οι δομές είναι αποδοτικές).

### Β. Εκτέλεση ερωτημάτων με την χρήση των jobs

Αρχικά μόλις ένα ερώτημα διαβαστεί από το stdin δημιουργείται ένα query_job το οποίο προστίθεται στην ουρά του scheduler. Όταν ένα νήμα την λάβει προς εκτέλεση κάνει τα εξής:

1) Αναλύει το ερώτημα

2) Ελέγχει ότι είναι έγκυρη ερώτηση και μπορεί να εκτελεστεί

3) Διατάσσει τα κατηγορήματα με την βέλτιστη σειρά

4) Δημιουργεί έναν πίνακα στον οποίο αποθηκεύεται αν χρειάζεται μία σχέση ταξινόμηση και μειώνει τις ταξινομήσεις αλλάζοντας θέσεις στα R/S

5) Αλλάζει την σειρά εκτέλεσης των κατηγορημάτων ώστε να μειωθεί η κατανάλωση μνήμης χωρίς να αυξηθεί ο χρόνος εκτέλεσης

6) Ξεκινά η εκτέλεση του ερωτήματος

Η εκτέλεση των κατηγορημάτων γίνεται με σειρά και διακρίνεται στις εξής περιπτώσεις:

i) Φίλτρο: χωρίζονται τα δεδομένα σε κομμάτια και δίνονται σε filter jobs οι οποίες τα φιλτράρουν, το thread σταματά το job που εκτελεί και συνεχίζει σε επόμενο. Τα δεδομένα μπορεί να είναι από σχέση ή από την ενδιάμεση δομή, και χωρισμός γίνεται ανάλογα με τον αριθμό στοιχείων/ κάδων λίστας. Μόλις ολοκληρωθεί το φιλτράρισμα το τελευταίο filter job ενώνει τα αποτελέσματα που δημιουργήθηκαν από τα άλλα jobs και ξαναβάζει το job του ερωτήματος στην ουρά ώστε να συνεχίσει από το σημείο που σταμάτησε.

ii) Join: Φτιάχνει δύο presort_jobs τα οποία δημιουργούν τα relations R/S και τα αντίγραφά τους (Project part 1) και το καθένα κάνει την ταξινόμηση δημιουργώντας ένα sort_job με παραμέτρους το 1ο byte και όρια ολόκληρη την σχέση, έπειτα τερματίζει. Κάθε sort job

δημιουργεί ένα ιστόγραμμα το οποίο χρησιμοποιεί για να φτιάξει και άλλα sort_jobs για κάθε κάδο (ο χωρισμός γίνεται όπως είχε περιγραφεί η ταξινόμηση στο part 1). Μόλις ταξινομηθούν και οι δύο σχέσεις R/S το τελευταίο sort_job βάζει στην ουρά ένα prejoin_job. Το prejoin job όταν εκτελεστεί χωρίζει τις σχέσεις σε κομμάτια τα οποία αναθέτει σε join_jobs τα οποία κάνουν το join, έπειτα τερματίζει.

Μόλις ολοκληρωθεί το join το τελευταίο join job ενώνει τα αποτελέσματα που δημιουργήθηκαν από τα άλλα jobs και ξαναβάζει το job του ερωτήματος στην ουρά ώστε να συνεχίσει από το σημείο που σταμάτησε.

Μόλις τα κατηγορήματα εκτελεστούν πρέπει να γίνουν οι προβολές, οπότε δημιουργούνται όσα projection jobs χρειάζονται και τα οποία υπολογίζουν τα αθροίσματα και μετά τα τοποθετούν στην projection list για να εμφανιστούν στον χρήστη με την σωστή σειρά. Το τελευταίο projection job απελευθερώνει τους πόρους του ερωτήματος.

## 2. Παραδοχές

1. Το Best Tree λειτουργεί με μέγιστο αριθμό πινάκων το 4 όπως έχει αναφερθεί.
2. Αν δύο πίνακες γίνονται join σε παραπάνω από ένα κατηγορήματα τότε το Best Tree λαμβάνει μόνο τη διακριτή τους σχέση και τα κατηγορήματα με ίδιους πίνακες τοποθετούνται σειριακά σύμφωνα με το αποτέλεσμα του Join Enumeration.

## 3. Σχεδιαστικές επιλογές & Δομές

1. Για την αποφυγή περιττών ταξινομήσεων κατά το στάδιο της εκτέλεσης query έχουμε κατασκευάσει ένα Bool array με τιμές 1 = το relation πρέπει να ταξινομηθεί και 0 = το relation δεν χρειάζεται να ταξινομηθεί.
2. Τα ενδιάμεσα αποτελέσματα αποθηκεύονται σε μια δομή middleman με τη μορφή πίνακα όπου κάθε θέση του πίνακα δείχνει σε μια λίστα απο rowIds. Η λίστα middle_list είναι μία απλά συνδεδεμένη λίστα με κεφαλή και δείκτη στον τελευταίο κάδο (ουρά). Κάθε κάδος της λίστας εκτός από τον buffer με μέγεθος 1MB και τον δείκτη στο επόμενο κάδο έχει και έναν δείκτη στην επόμενη κενή θέση του buffer ώστε σε συνδυασμό με τον δείκτη της ουράς η εισαγωγή να γίνεται σε O(1).
3. Ο χωρισμός σε join_jobs διαφέρει ελάχιστα από την εκφώνηση δεδομένου ότι ο χωρισμός των join_jobs ώστε να έχουν περίπου ίδιο αριθμό στοιχείων θεωρούμε ότι είναι πιο αποδοτικός από πλευράς χρόνου και κατανάλωσης μνήμης.
4. Ουρά από jobs. Λίστα όπου κάθε κόμβος είναι πίνακας με δείκτες σε jobs, το μέγεθος του πίνακα είναι σταθερό και ορισμένο με #define. Η εισαγωγή και εξαγωγή ενός job είναι O(1).
5. Λίστα προβολών. Είναι μία λίστα με ταξινομημένους κόμβους σύμφωνα με ένα id (σειρά της ερώτησης) η οποία όταν υπολογιστούν όλες οι προβολές ενός ερωτήματος τις εμφανίζει.
6. Δομή για τα ενδιάμεσα αποτελέσματα. Είναι ίδια με αυτή που χρησιμοποιήσαμε στο part 2 του project με την διαφορά πως το μέγεθος του κάδου από 1MB μειώθηκε στα 128KB και δεν είναι υποχρεωτικό κάθε κάδος να είναι γεμάτος (όταν ενώνονται τα αποτελέσματα από πολλά jobs).

## 4. Μεταγλώττιση και εκκίνηση προγράμματος

### Α. Μεταγγλώτιση

- Για τα queries και τα tests τους: ***make*** ή ***make all***
- Για τα queries μόνο: ***make notests***
- Για τα tests: ***make tests***
- Για διαγραφή των εκτελέσιμων και των αντικειμενικών (.o) αρχείων: ***make clean***

Χρησιμοποείται by default –O3 optimization.

Ο χρήστης την ώρα της μεταγλώττισης μπορεί να επιλέγει ποιες εργασίες θα γίνονται παράλληλα (λεπτομέρειες για τον τρόπο στην μεταγλώττιση του προγράμματος).

| **Flag** | **Make command** | **Λειτουργία** |
|---|---|---|
| -DSORTED_PROJECTIONS | sorted_projections=true | Εκτύπωση των αποτελεσμάτων με τη σωστή σειρά |
| -DSERIAL_EXECUTION | s_execution=true | Κάθε thread εκτελεί ένα ερώτημα |
| -DSERIAL_JOIN | s_join=true | Η ζεύξη των relation r s γίνεται από ένα thread |
| -DSERIAL_SORTING | s_sorting=true | Η ταξινόμηση μίας σχέσης γίνεται από ένα thread |
| -DSERIAL_PRESORTING | s_presorting=true | Η δημιουργία των relation R/S γίνεται από ένα thread (υποχρεωτικά συνδυάζεται με s_sort s_join) |
| -DSERIAL_FILTER | s_filter=true | Τα φίλτρα της μορφής 0.1 <>= uint64_t να εκτελούνται από ένα thread |
| -DSERIAL_SELFJOIN | s_selfjoin=true | Τα φίλτρα της μορφής 0.1=0.2 εκτελούνται από ένα thread |
| -DSERIAL_PROJECTIONS | s_projections=true | Οι προβολές εκτελούνταν από ένα thread |
| -DONE_QUERY_AT_A_TIME | one_query=true | Όλα τα threads εκτελούν το ίδιο ερώτημα |
| -DMAX_QUERIES_LIMIT | max_queries=true | Το πρόγραμμα παίρνει ένα παραπάνω όρισμα που είναι το πόσα queries το πολύ θα εκτελούνται ταυτόχρονα |
| -DONE_BATCH_AT_A_TIME | one_batch=true | Εκτελεί ένα batch ερωτημάτων την φορά |

### Β. Εκτέλεση

**Μέσω script:**

Εκτελέστε μέσα από τον φάκελο small (ή medium) δίνοντας για ορίσματα το path προς το εκτελέσιμο queries και τη λέξη "small" (ή "medium" αντίστοιχα). Το τελικό αποτέλεσμα είναι το output της συνάρτησης diff.

```
./script.sh ../queries αριθμός_threads small/medium
```

**Μέσω command:**

```
./queries
```

## 5. Μετρήσεις

Οι μετρήσεις εκτελέστηκαν αφενός στα μηχανήματα της σχολής (linux25) και αφετέρου σε έναν προσωπικό υπολογιστή μέλους της ομάδας.

Οι μετρήσεις αφορούν αποκλειστικά το dataset MEDIUM.

**Α. Αποτελέσματα μετρήσεων για το PART 2**

| Μηχάνημα | Συνολικός χρόνος (φόρτωση πινάκων & εκτέλεση queries) (sec) | Μέγιστη χρήση RAM (GB) |
|---|---|---|
| linux25 | 61.3 | 4.91 |
| B450 AORUS M | 55 | |

**Β. Αποτελέσματα μετρήσεων στον linux25 για το PART 3**

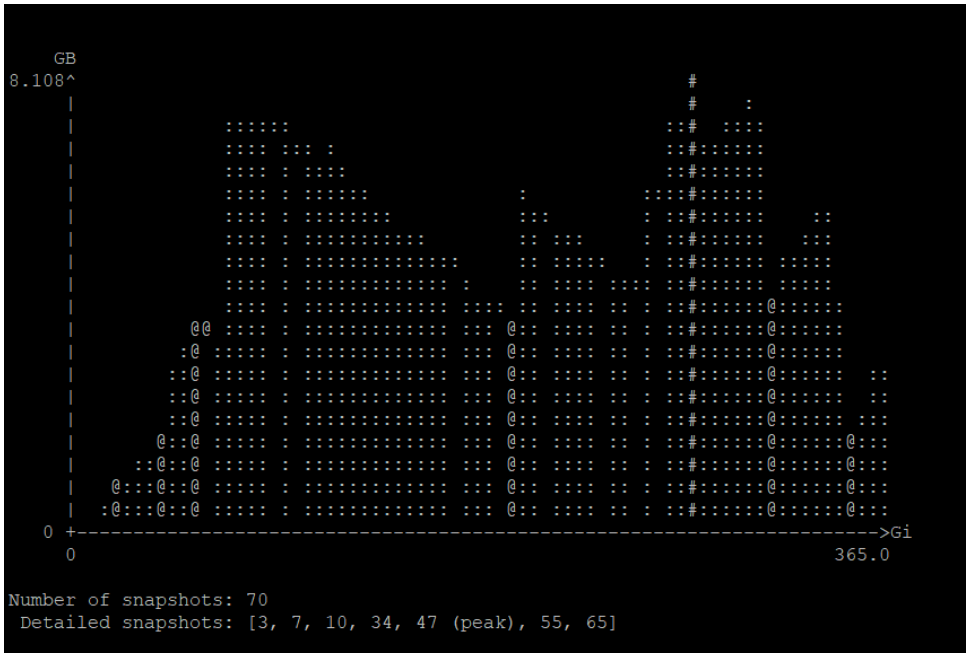
**i. Χαρακτηριστικά μηχανήματος**

```
>lscpu
Architecture:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Byte Order:              Little Endian
Address sizes:           39 bits physical, 48 bits virtual
CPU(s):                  4
On-line CPU(s) list:     0-3
Thread(s) per core:      2
Core(s) per socket:      2
Socket(s):               1
NUMA node(s):            1
Vendor ID:               GenuineIntel
CPU family:              6
Model:                   78
Model name:              Intel(R) Core(TM) i7-6500U CPU @ 2.50GHz
Stepping:                3
CPU MHz:                 2800.060
CPU max MHz:             3100.0000
CPU min MHz:             400.0000
BogoMIPS:                5199.98
Virtualization:          VT-x
L1d cache:               64 KiB
L1i cache:               64 KiB
L2 cache:                512 KiB
L3 cache:                4 MiB
```

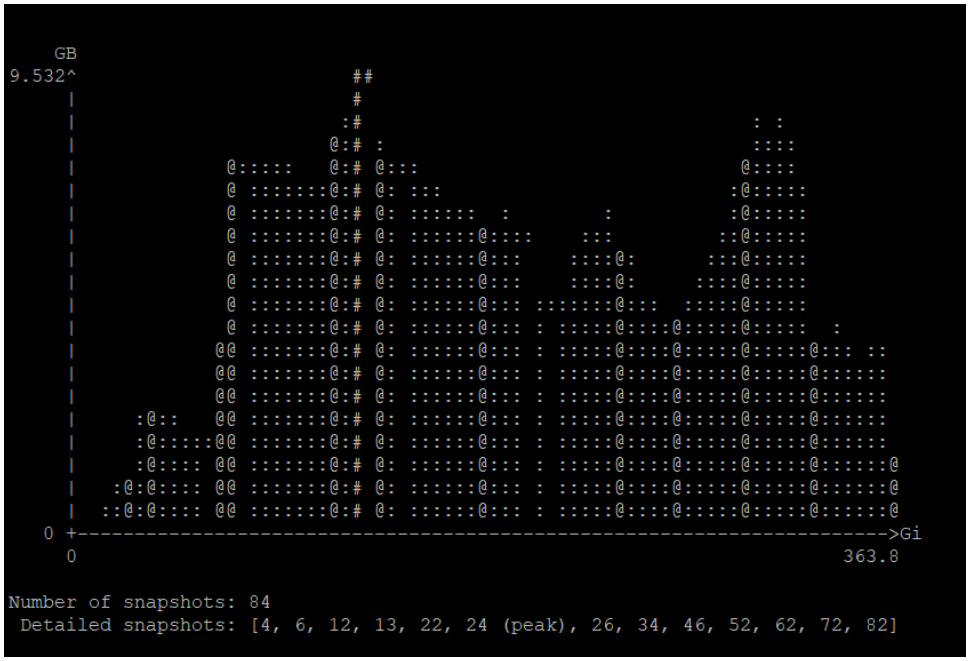
### ii. ΒΑΣΙΚΗ Εκτέλεση με πλήρη παραλληλία & ταξινομημένες προβολές (sorted_projections=true)

```
> make sorted_projections=true queries_fast
> cat /tmp/temp_medium_dataset/medium.txt | ./queries αριθμός_threads
```

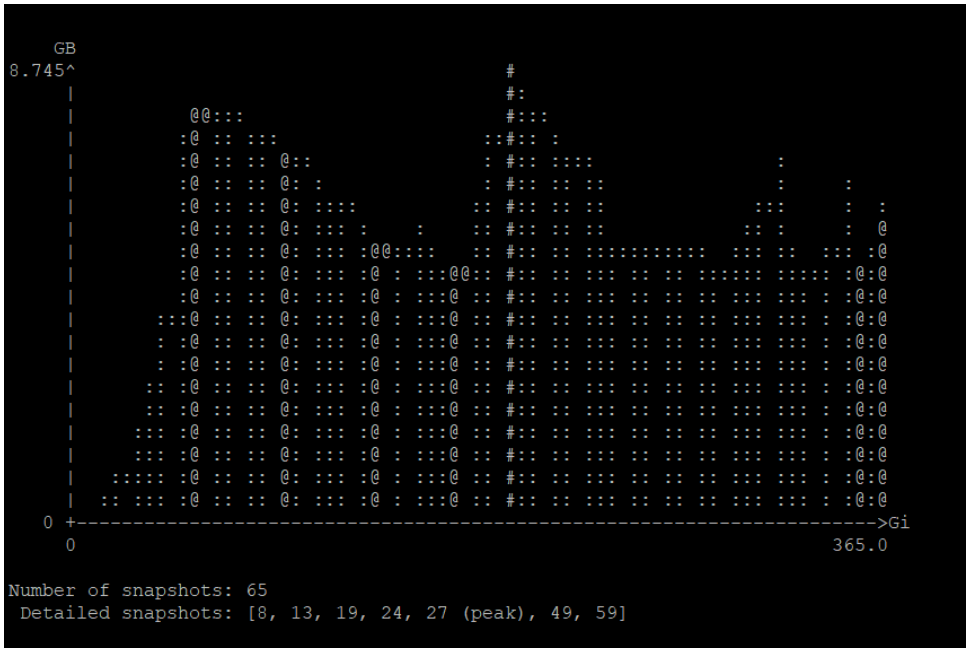
| Αριθμός Threads | Χρόνος φόρτωσης πινάκων (sec) | Χρόνος εκτέλεσης queries (sec) |
|---|---|---|
| 1 | 5.608659 | 61.650076 |
| 2 | 5.614574 | 34.318706 |
| **4** | **5.611118** | **21.290931** |
| 8 | 5.628218 | 22.141975 |



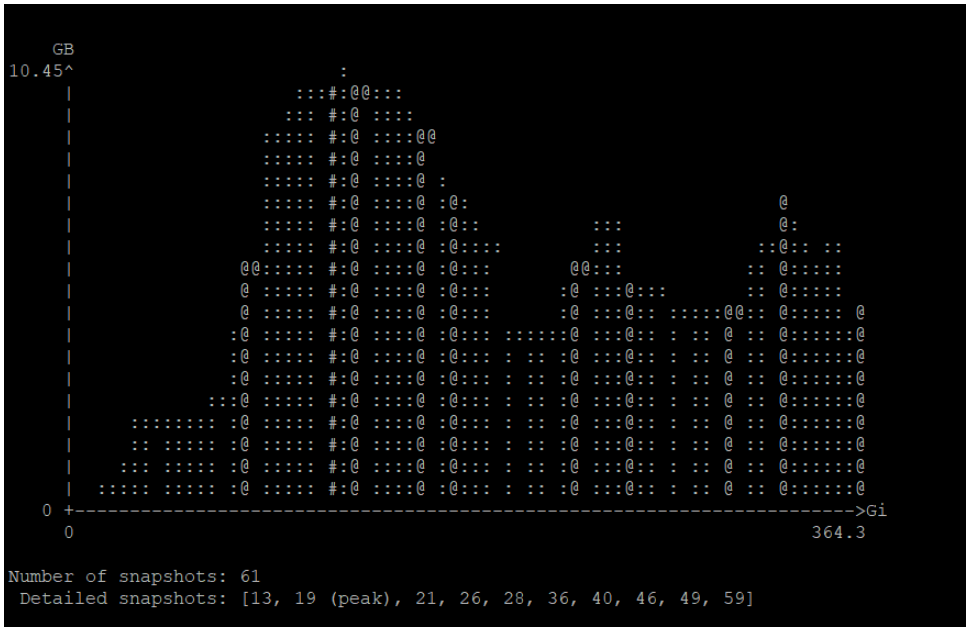

**Εικόνα 1. Διάγραμμα μνήμης για 1 thread**




**Εικόνα 2. Διάγραμμα μνήμης για 2 threads**




**Εικόνα 3. Διάγραμμα μνήμης για 4 threads**

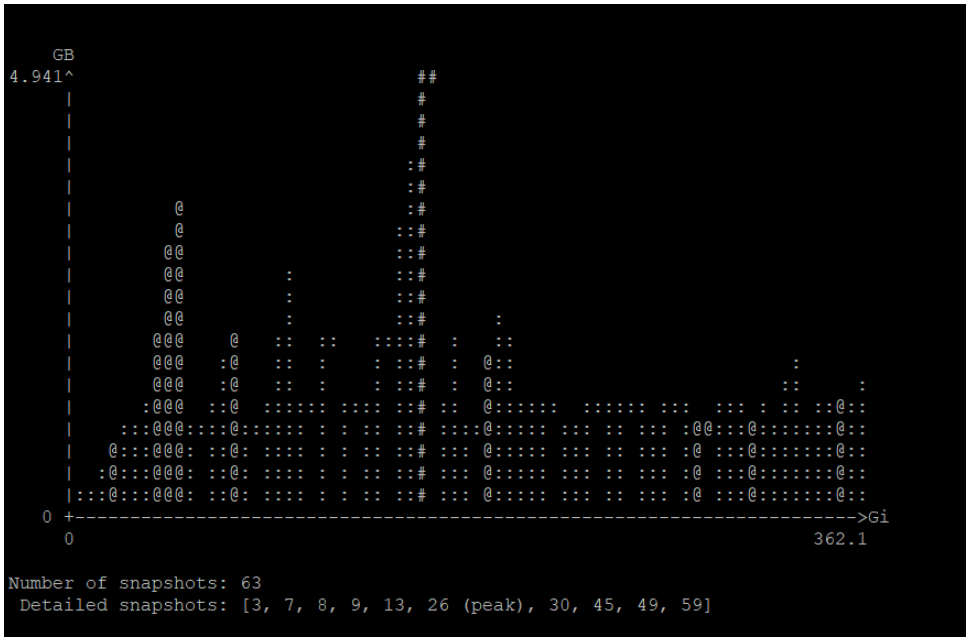



**Εικόνα 4. Διάγραμμα μνήμης για 8 threads**

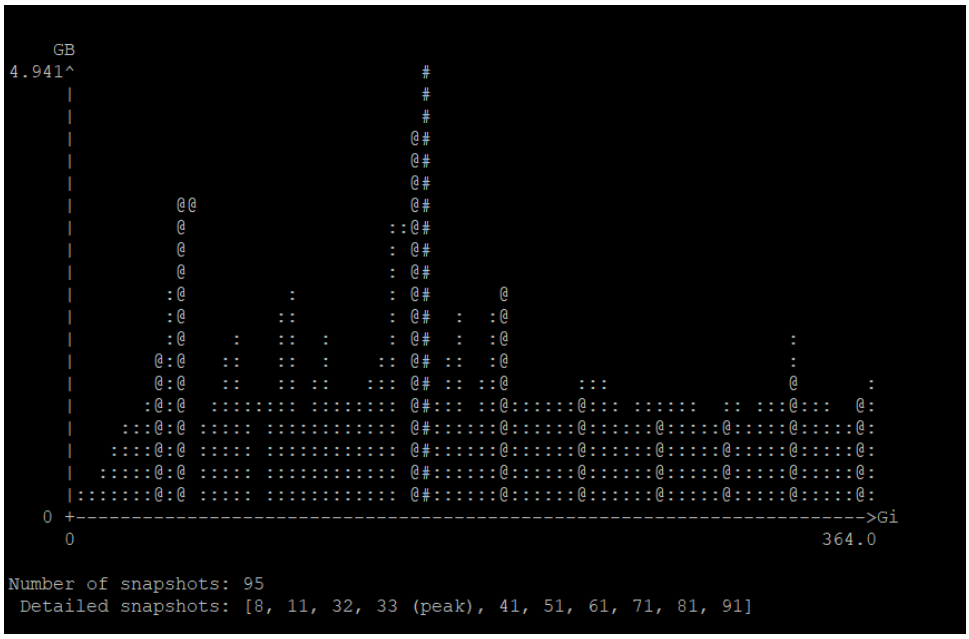
### iii. Όλα τα threads εκτελούν το ίδιο ερώτημα (one_query=true) & ταξινομημένες προβολές(sorted_projections=true)

```
> make sorted_projections=true one_query=true queries_fast
> cat /tmp/temp_medium_dataset/medium.txt | ./queries αριθμός_threads
```

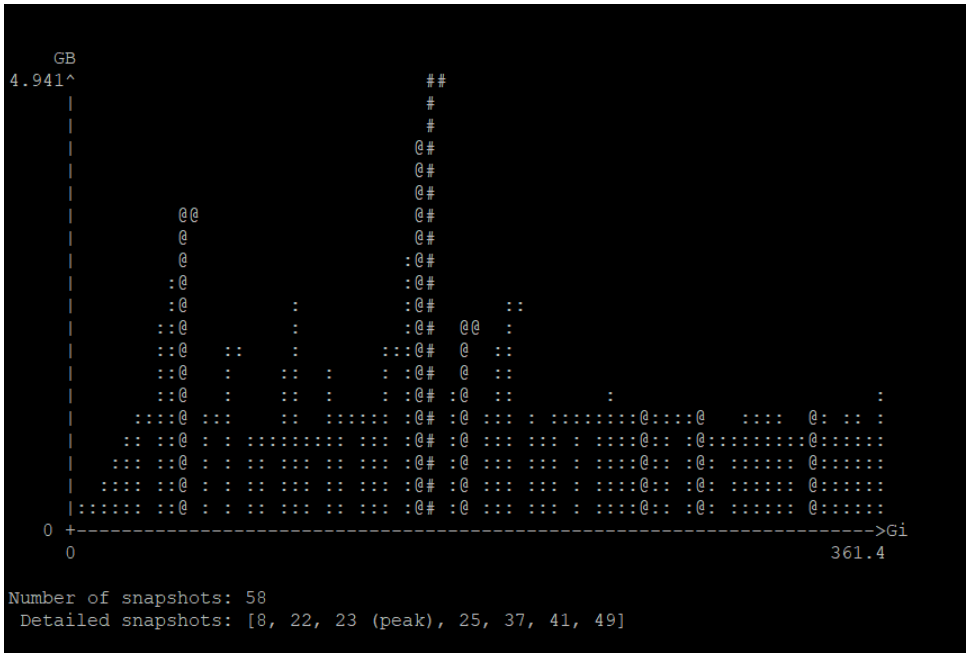
| Αριθμός Threads | Χρόνος φόρτωσης πινάκων (sec) | Χρόνος εκτέλεσης queries (sec) |
|---|---|---|
| 1 | 5.599503 | 64.325225 |
| 2 | 5.595457 | 50.462987 |
| **4** | **5.595775** | **44.894061** |
| 8 | 5.596730 | 44.943897 |



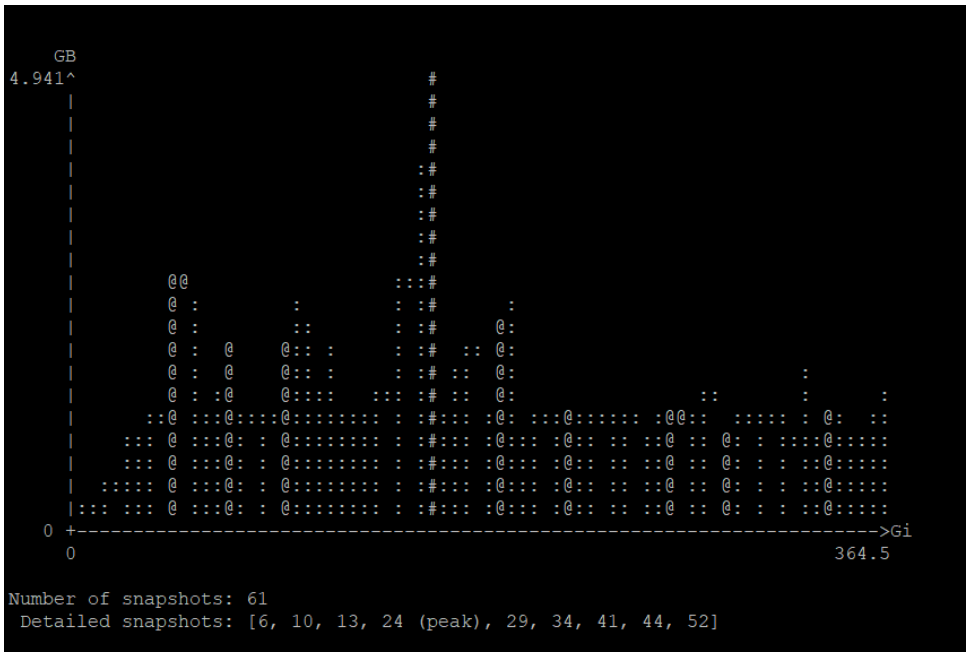

**Εικόνα 5. Διάγραμμα μνήμης για 1 thread**




**Εικόνα 6. Διάγραμμα μνήμης για 2 threads**




**Εικόνα 7. Διάγραμμα μνήμης για 4 threads**

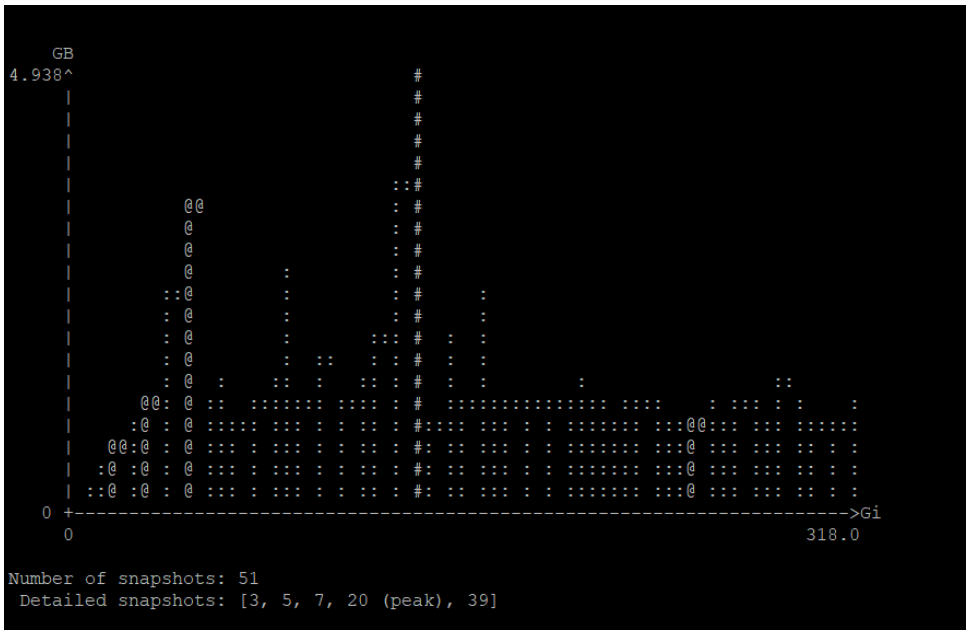



**Εικόνα 8. Διάγραμμα μνήμης για 8 threads**

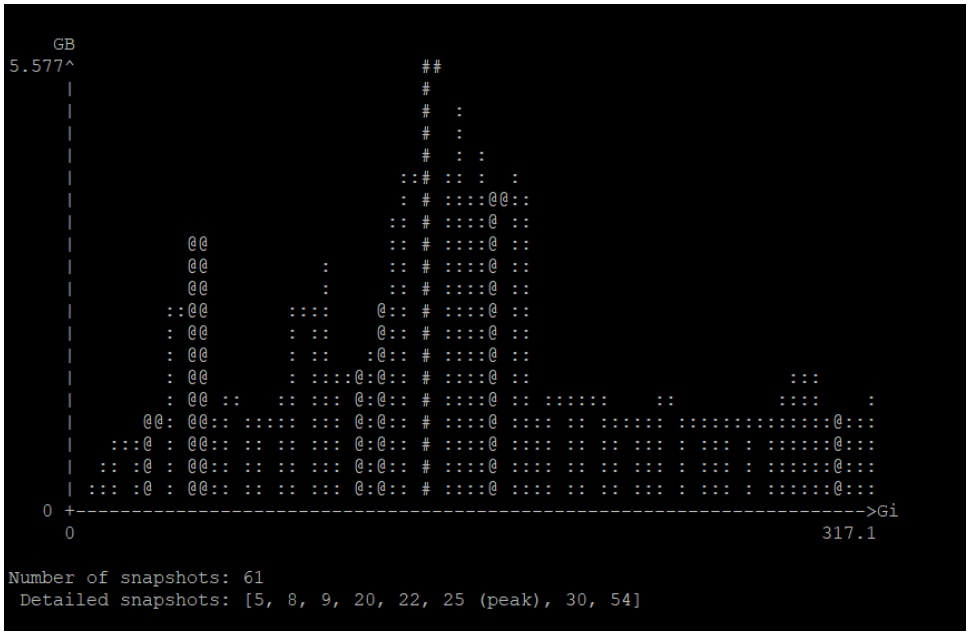
### iv. Κάθε thread εκτελεί ένα ερώτημα (s_execution=true) & ταξινομημένες προβολές(sorted_projections=true)

```
> make sorted_projections=true s_execution=true queries_fast
> cat /tmp/temp_medium_dataset/medium.txt | ./queries αριθμός_threads
```

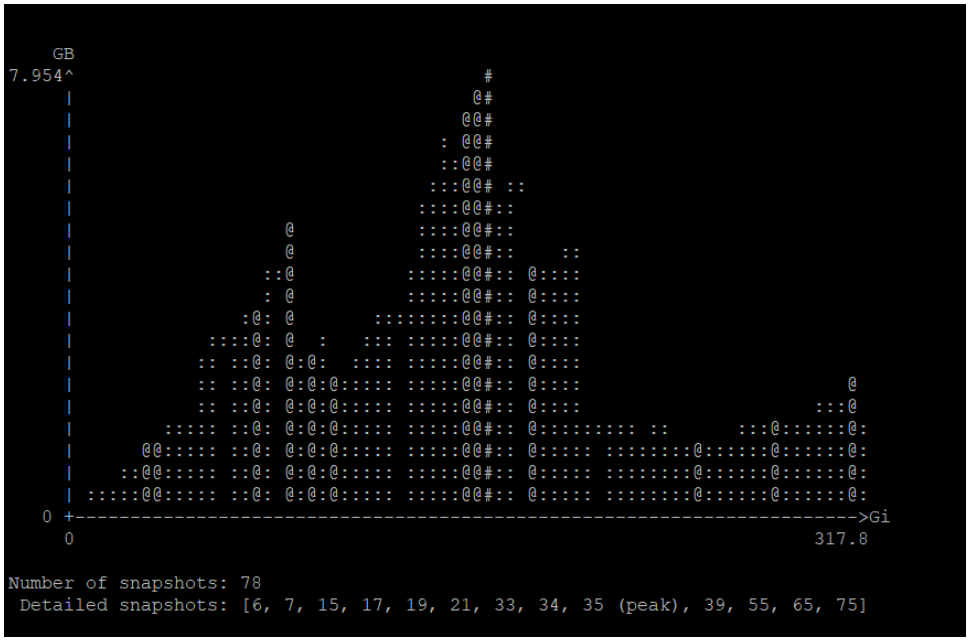
| Αριθμός Threads | Χρόνος φόρτωσης πινάκων (sec) | Χρόνος εκτέλεσης queries (sec) |
|---|---|---|
| 1 | 5.398737 | 55.457012 |
| 2 | 5.396011 | 29.555194 |
| **4** | **5.400537** | **17.406146** |
| 8 | 5.394617 | 17.626498 |



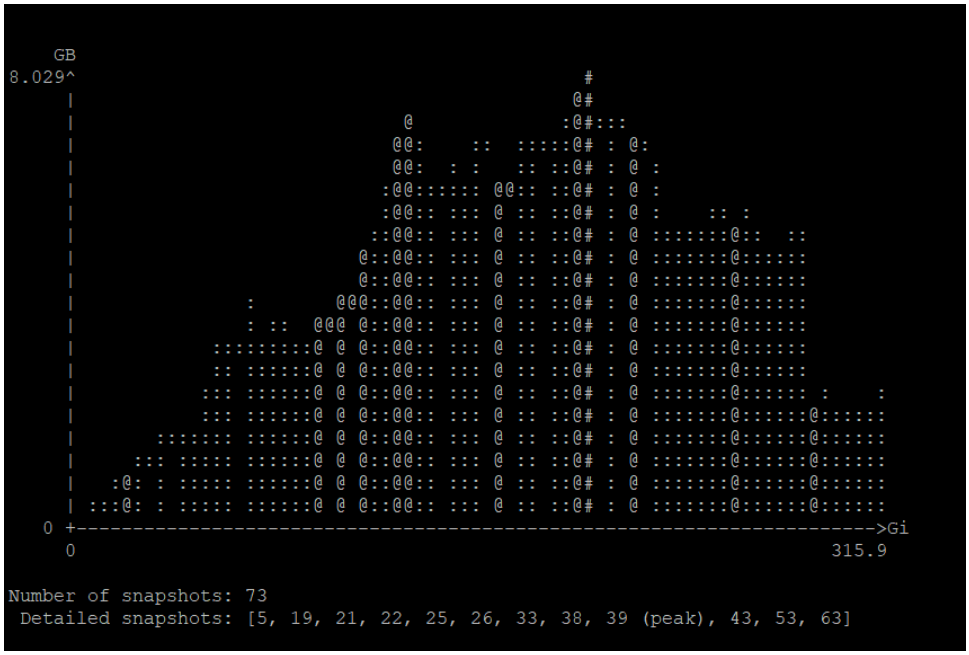

**Εικόνα 9. Διάγραμμα μνήμης για 1 thread**




**Εικόνα 10. Διάγραμμα μνήμης για 2 threads**




**Εικόνα 11. Διάγραμμα μνήμης για 4 threads**

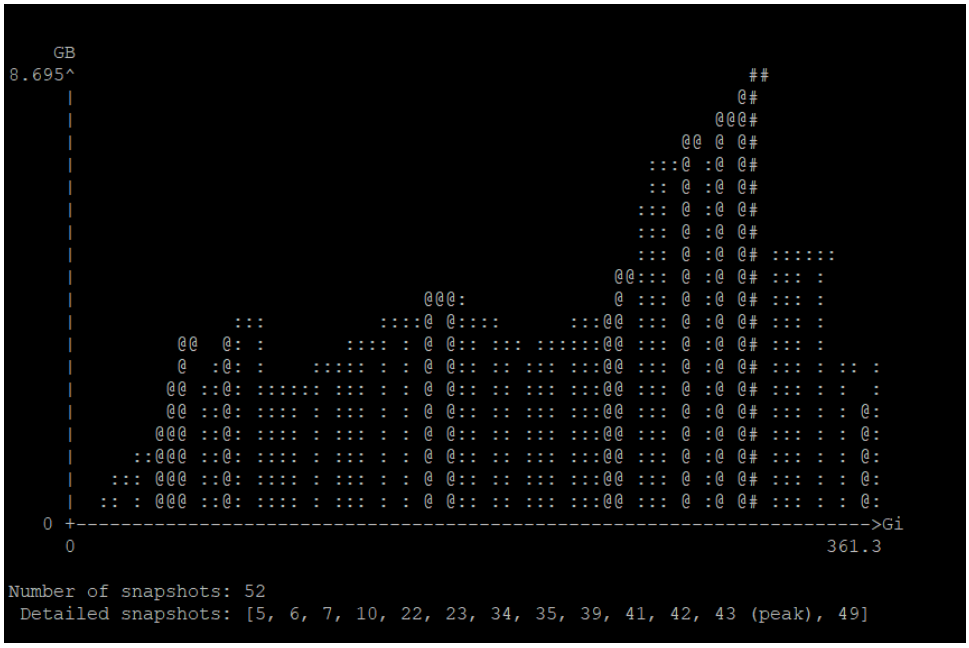



**Εικόνα 12. Διάγραμμα μνήμης για 8 threads**

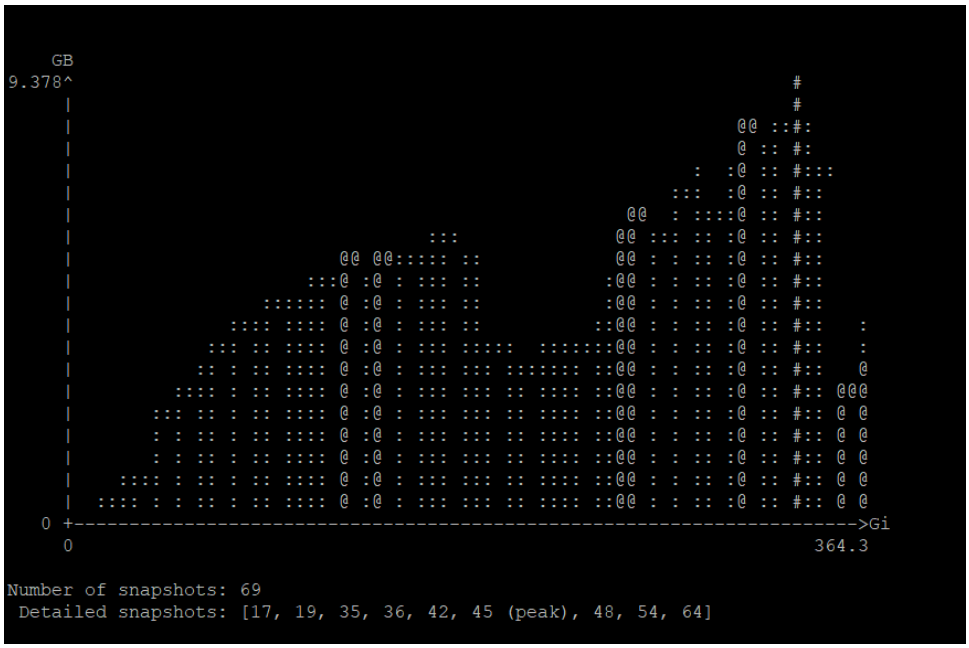
### v. Η ταξινόμηση μιας σχέσης γίνεται από ένα thread (s_sorting=true) & ταξινομημένες προβολές(sorted_projections=true)

```
> make sorted_projections=true s_sorting=true queries_fast
> cat /tmp/temp_medium_dataset/medium.txt | ./queries αριθμός_threads
```

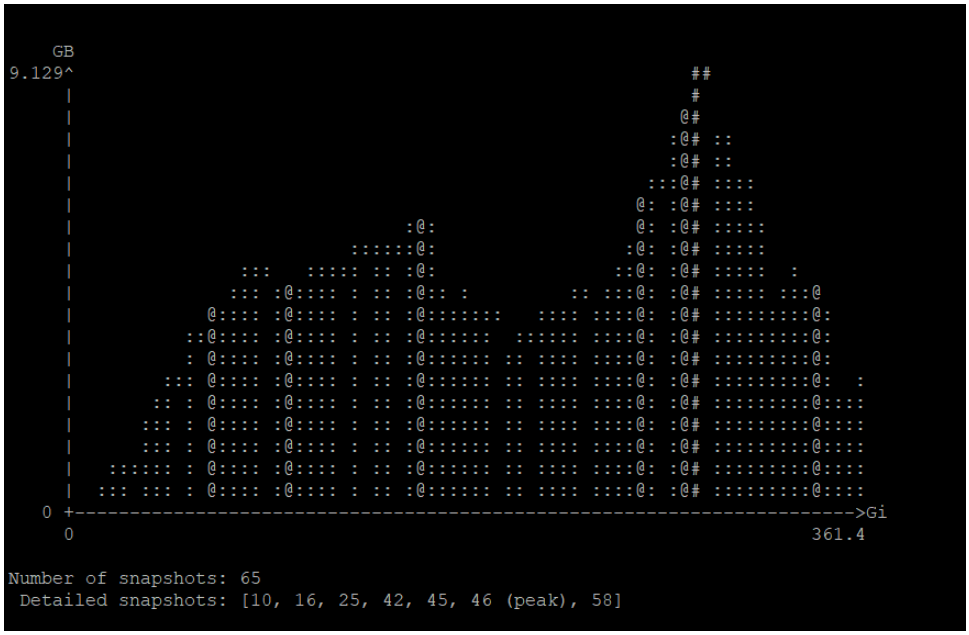
| Αριθμός Threads | Χρόνος φόρτωσης πινάκων (sec) | Χρόνος εκτέλεσης queries (sec) |
|---|---|---|
| 1 | 5.614153 | 60.790483 |
| 2 | 5.615919 | 33.340965 |
| **4** | **5.613277** | **19.575593** |
| 8 | 5.616220 | 20.980385 |



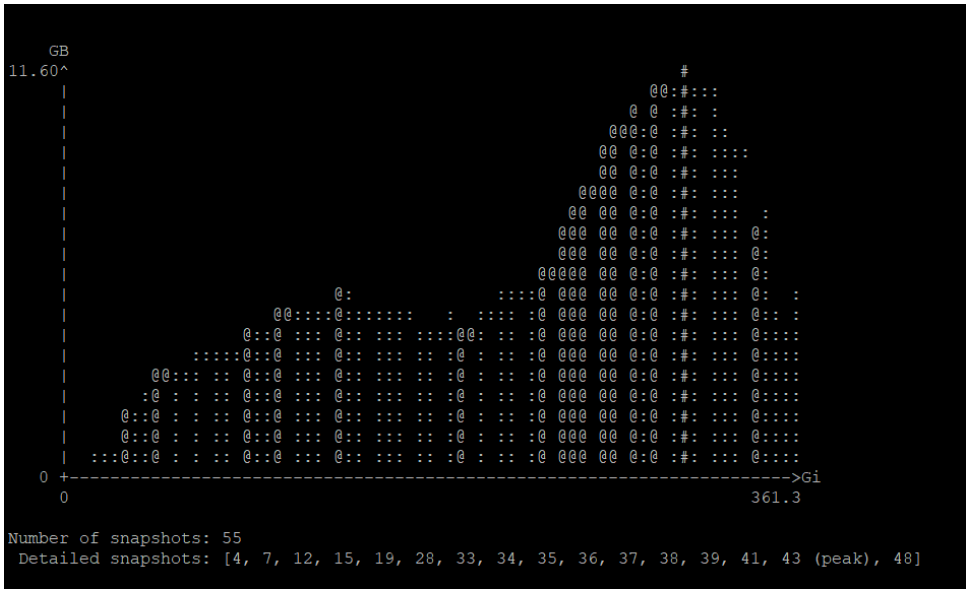

**Εικόνα 13. Διάγραμμα μνήμης για 1 thread**




**Εικόνα 14. Διάγραμμα μνήμης για 2 threads**




**Εικόνα 15. Διάγραμμα μνήμης για 4 threads**




**Εικόνα 16. Διάγραμμα μνήμης για 8 threads**

## Β. Αποτελέσματα μετρήσεων σε προσωπικό υπολογιστή B450 AORUS M

### i. Χαρακτηριστικά μηχανήματος

```
*-memory
   description: System Memory
   physical id: 9
   slot: System board or motherboard
   size: 32GiB
  *-bank:0
```

```
        description: DIMM DDR4 Synchronous Unbuffered (Unregistered) 3200 MHz (0,3 ns)
        product:
        vendor: Kingston
        physical id: 0
        serial:
        slot: DIMM 0
        size: 8GiB
        width: 64 bits
        clock: 3200MHz (0.3ns)
     *-bank:1
        description: DIMM DDR4 Synchronous Unbuffered (Unregistered) 3200 MHz (0,3 ns)
        product:
        vendor: Kingston
        physical id: 1
        serial:
        slot: DIMM 1
        size: 8GiB
        width: 64 bits
        clock: 3200MHz (0.3ns)
     *-bank:2
        description: DIMM DDR4 Synchronous Unbuffered (Unregistered) 3200 MHz (0,3 ns)
        product:
        vendor: Kingston
        physical id: 2
        serial:
        slot: DIMM 0
        size: 8GiB
        width: 64 bits
        clock: 3200MHz (0.3ns)
     *-bank:3
        description: DIMM DDR4 Synchronous Unbuffered (Unregistered) 3200 MHz (0,3 ns)
        product:
        vendor: Kingston
        physical id: 3
        serial:
        slot: DIMM 1
        size: 8GiB
        width: 64 bits
        clock: 3200MHz (0.3ns)
  *-cache:0
     description: L1 cache
     physical id: c
     slot: L1 - Cache
     size: 384KiB
     capacity: 384KiB
     clock: 1GHz (1.0ns)
     capabilities: pipeline-burst internal write-back unified
     configuration: level=1
  *-cache:1
     description: L2 cache
     physical id: d
```

```
      slot: L2 - Cache
      size: 3MiB
      capacity: 3MiB
      clock: 1GHz (1.0ns)
      capabilities: pipeline-burst internal write-back unified
      configuration: level=2
   *-cache:2
      description: L3 cache
      physical id: e
      slot: L3 - Cache
      size: 32MiB
      capacity: 32MiB
      clock: 1GHz (1.0ns)
      capabilities: pipeline-burst internal write-back unified
      configuration: level=3
   *-cpu
      description: CPU
      product: AMD Ryzen 5 3600 6-Core Processor
      vendor: Advanced Micro Devices [AMD]
      physical id: f
      bus info: cpu@0
      version: AMD Ryzen 5 3600 6-Core Processor
      serial: Unknown
      slot: AM4
      size: 3601MHz
      capacity: 4200MHz
      width: 64 bits
      clock: 100MHz
      configuration: cores=6 enabledcores=6 threads=12
```

### ii. ΒΑΣΙΚΗ Εκτέλεση με πλήρη παραλληλία & ταξινομημένες προβολές (sorted_projections=true)

```
> make sorted_projections=true queries_fast
> cat /tmp/temp_medium_dataset/medium.txt | ./queries αριθμός_threads
```

| Αριθμός Threads | Χρόνος φόρτωσης πινάκων (sec) | Χρόνος εκτέλεσης queries (sec) |
|---|---|---|
| 1 | 4.013499 | 53.158049 |
| 2 | 3.967026 | 28.231862 |
| 3 | 3.945958 | 20.039258 |
| 4 | 4.008930 | 16.191860 |
| 6 | 3.997297 | 13.212006 |
| 8 | 3.993070 | 12.335795 |
| **12** | **3.950410** | **11.396190** |
| 16 | 3.954679 | 11.737083 |
| 24 | 3.962276 | 11.512975 |

### iii. Όλα τα threads εκτελούν το ίδιο ερώτημα (one_query=true) & ταξινομημένες προβολές(sorted_projections=true)

```
> make sorted_projections=true s_execution=true queries_fast
> cat /tmp/temp_medium_dataset/medium.txt | ./queries αριθμός_threads
```

| Αριθμός Threads | Χρόνος φόρτωσης πινάκων (sec) | Χρόνος εκτέλεσης queries (sec) |
|---|---|---|
| 1 | 3.928364 | 47.466849 |
| 2 | 3.948091 | 24.893249 |
| 3 | 3.944387 | 17.477655 |
| 4 | 3.927595 | 13.961151 |
| 6 | 3.930287 | 10.673232 |
| 8 | 3.932697 | 9.878880 |
| **12** | **3.965038** | **8.987036** |
| 16 | 3.942931 | 9.736968 |
| 24 | 3.931237 | 10.112750 |

### iv. Κάθε thread εκτελεί ένα ερώτημα (s_execution=true) & ταξινομημένες προβολές(sorted_projections=true)

```
> make sorted_projections=true one_query=true queries_fast
> cat /tmp/temp_medium_dataset/medium.txt | ./queries αριθμός_threads
```

| Αριθμός Threads | Χρόνος φόρτωσης πινάκων (sec) | Χρόνος εκτέλεσης queries (sec) |
|---|---|---|
| 1 | 3.986762 | 54.682986 |
| 2 | 3.958781 | 41.672859 |
| 3 | 3.961692 | 38.563542 |
| 4 | 3.969754 | 36.892797 |
| 6 | 4.085157 | 35.179977 |
| 8 | 3.976101 | 34.938618 |
| **12** | **3.966161** | **34.351414** |
| 16 | 3.941654 | 34.610831 |
| 24 | 3.966126 | 34.660177 |

### v. Η ταξινόμηση μιας σχέσης γίνεται από ένα thread (s_sorting=true) & ταξινομημένες προβολές(sorted_projections=true)

```
> make sorted_projections=true s_sorting=true queries_fast
> cat /tmp/temp_medium_dataset/medium.txt | ./queries αριθμός_threads
```

| Αριθμός Threads | Χρόνος φόρτωσης πινάκων (sec) | Χρόνος εκτέλεσης queries (sec) |
|---|---|---|
| 1 | 3.959821 | 52.495572 |
| 2 | 3.974779 | 27.800552 |
| 3 | 3.967809 | 19.781864 |
| 4 | 3.995966 | 15.392042 |
| 6 | 3.962640 | 11.747725 |
| 8 | 4.001685 | 10.844093 |
| **12** | **3.978305** | **10.229269** |
| 16 | 3.999611 | 10.754502 |
| 24 | 3.966883 | 10.893566 |

### vi. Πλήρης παραλληλία με μέγιστο πλήθος ερωτημάτων 10 (max_queries=true) & ταξινομημένες προβολές(sorted_projections=true)

```
> make sorted_projections=true max_queries=true queries_fast
> cat /tmp/temp_medium_dataset/medium.txt | ./queries αριθμός_threads 10
```

| Μέγεθος dataset | Αριθμός Threads | Χρόνος φόρτωσης πινάκων (sec) | Χρόνος εκτέλεσης queries (sec) |
|---|---|---|---|
| medium | 12 | 3.992746 | 9.930044 |
| 2*medium | 12 | 3.957839 | 19.226326 |
| 4*medium | 12 | 3.972631 | 37.639145 |
| 8*medium | 12 | 3.957961 | 73.951724 |
| 16*medium | 12 | 3.958513 | 146.847615 |

## 6. Συμπεράσματα

Αρχικά βέλτιστες επιδόσεις σε κάθε υλοποίηση παρατηρούμε όταν χρησιμοποιούμε όσα threads έχει ο επεξεργαστής (4 για το linux25, 12 για τον 2ο υπολογιστή), αυτό είναι αναμενόμενο όπως συζητήθηκε και στο piazza αφού αν έχουμε περισσότερα των φυσικών υπάρχει ανταγωνισμός μεταξύ τους και χάνεται κάποιος χρόνος.

Από τα αποτελέσματα βλέπουμε πως η πιο γρήγορη υλοποίηση είναι αυτή που κάθε thread εκτελεί ένα ερώτημα από την αρχή του (ανάλυση, optimize) μέχρι τον υπολογισμό των προβολών του. Αυτό γίνεται επειδή:

1) Όλα τα ερωτήματα προς εκτέλεση του dataset ήταν γνωστά από την αρχή

2) Το πλήθος των thread των επεξεργαστών που δοκιμάσαμε είναι μικρότερο του πλήθους των ερωτημάτων.

Άρα σε πραγματικές συνθήκες όπου κάθε batch ερωτημάτων θα φτάνει με κάποια καθυστέρηση ή αν έχουμε μηχάνημα με πλήθος threads μεγαλύτερο των ερωτημάτων τότε η συγκεκριμένη υλοποίηση θα χάνει χρόνο αφού πολλά threads δεν θα έχουν ερώτημα να εκτελέσουν.

Η πλήρης παραλληλία είναι αρκετά αποδοτική και σε μεγαλύτερα δεδομένα θα απέδιδε ακόμα καλύτερα με μόνο μειονέκτημα την κατανάλωση RAM το οποίο λύνεται αν βάλουμε όριο στα ταυτόχρονα ερωτήματα που θα εκτελούνται όπως έχει αναφερθεί παραπάνω.