Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Ηλεκτρονικής

# Τίτλος διπλωματικής

Διπλωματική Εργασία
του
Κωνσταντίνος Χατζηαντωνίου

**Επιβλέπων:** Νικόλαος Πιτσιάνης
Καθηγητής Α.Π.Θ.

23 Οκτωβρίου 2020

## Περίληψη

Άδειο

**Abstract**

Empty

# Ευχαριστίες

Άδειο

# Τίτλος διπλωματικής

Όνομα Επίθετο
empty@auth.gr

23 Οκτωβρίου 2020

# Περιεχόμενα

Το πρώτο βήμα για να εδραιωθεί η Julia σαν καλή ή και καλύτερη επιλογή για προγραμματισμό σε CUDA, είναι να συγκριθεί με τη C ως προς την ταχύτητα και την ποιότητα κώδικα. Παρόλο που ήδη υπάρχουν μετρήσεις που πραγματοποιούν αύτη τη σύγκριση [ροντινια], το να γίνουν υλοποιήσεις από το μηδέν, βοηθάει στην απόκτηση βαθύτερης γνώσεις, εντοπισμό λεπτομερειών και πιο εύκολη αλλαγή του κώδικα για εκτενέτερη έρευνα.

## 0.1 Μελέτη 1 - Κοντινότερος Γείτονας σε πλέγμα

### 0.1.1 Περιγραφή Προβλήματος

Η εύρεση το κοντινότερου γείτονα σε πλέγμα (Grid KNN) είναι μια παραλλαγή του κανονικού αλγορίθμου εύρεσης κοντινότερων γειτόνων για τρισδιάστατα σημεία. Τα δεδομένα, τόσο εισόδου όσο και ερωτημάτων (queries) ταξινομούνται σε κύβους ίσου μεγέθους δημιουργώντας ένα πλέγμα. Η αναζήτηση του κοντινότερου γείτονα για ένα ερώτημα, γίνεται πρώτα στον κύβο που ανήκει και έπειτα στους γειτονικούς κύβους.

### 0.1.2 Περιγραφή Υλοποίσης

Υλοποιήθηκε μόνο η περίπτωση που αναζητάμε τον έναν κοντινότερο γείτονα. Υπάρχουν δύο προσεγγίσεις: είτε να γίνει η αναζήτηση σε όλους τους γειτονικούς κύβους, χώρις κάποιον έλεγχο (απλή εκδοχή), είτε να ελέγχεται αν κάποιος γείτονας είναι πιο κοντά από τις πλευρές των κύβων για να αποφευχθούν οι περιττές αναζητήσεις (εκδοχή με έλεγχο).

Για τις γλώσσες, οι υλοποιήσεις είναι πανομοιότυπες όσον αφορά την αλγοριθμική προσέγγισησ, με τη διαφορά να βρίσκεται στο συντακτικό. Η Julia, στην απλή εκδοχή της, έχει τρεις διαφορετικές συντακτικές προσεγγίσεις

- "jl_simple Ίδια με την υλοποίηση σε C. Οι τύποι των μεταβλητών δηλώνονται και η μόνη διαφορά είναι ότι χρησιμοποιείται τρισδιάστατη διευθυνσιοδότηση στους πίνακες.
- jl_view Η διαφόρα με παραπάνω είναι ότι χρησιμοποιείται η μακροεντολή @view για να μη χρησιμοποιείται μετατόπιση διεύθυνσης σε όλο τον κώδικα.
- jl_no_types Ίδια με την παραπάνω απλά χωρίς να δηλώνονται οι τύποι των μεταβλητών.

Για την εκδοχή με έλεγχο ακολουθείται ο ίδιος τρόπος ονομασίας.

### 0.1.3 Ποιότητα Κώδικα

**Δήλωση τύπων μεταβλητών** Η Julia δεν απαιτεί να δηλώνουμε τον τύπο των μεταβλητών. Αυτό έχει ως αποτέλεσμα λιγότερο ¨γεμάτο¨ κώδικα.

Listing 1: C παράδειγμα με δήλωση τύπων

```
int tid = threadIdx.x + threadIdx.y*blockDim.x;
int stride = blockDim.x*blockDim.y;
int start_points = intgr_points_per_block[p_bid];
int start_queries = intgr_queries_per_block[q_bid];
```

Listing 2: Julia παράδειγμα χωρίς δήλωση τύπων

```
tid = threadIdx().x + (threadIdx().y-1)*blockDim().x
stride = (blockDim().x)*(blockDim().y)
startPoints = IntPointsperblock[p_bid]
startQueries = IntQueriesperblock[q_bid]
```

Θα περιμέναμε ότι είναι πολύ σημαντικό για προγραμματισμό σε κάρτες γραφικών να δηλώνονται οι τύποι. 64-bit αριθμητικές εντολές μπορεί να είναι έως και 8 φορές πιο αργές από τις αντίστοιχες 32-bit.

**Προσπέλαση στοιχείων πίνακα** Η Julia μπορεί να χρησιμοποιήσει πολυδιάστατη διευθυνσιοδότηση σε πίνακες της κάρτας γραφικών, τόσο στους καθολικούς global όσο και στους στατικούς και δυναμικούς μεριζόμενους (shared). Αντίθετα η C χρησιμοποιεί μονοδιάστατη γραμμική διευθυνσιοδότηση σε όλους τους πίνακες, εκτός από τους στατικούς μεριζόμενους πίνακες, όπου μπορεί και πολυδιάστατη.

Χρησιμοποιώντας πολυδιάστατη διευθυνσιοδότηση, ο κώδικας είναι πιο ευανάγνωστος και λιγότερο ευπαθής σε λογικά λάθη.

Listing 3: C παράδειγμα προσπέλασης πίνακα

```
int q_index = q + tid + start_queries;
if(tid + q < total_queries){
    for(int d = 0; d < dimensions; d++){
        sh_queries[tid + d*stride]
                = queries[q_index + d*num_of_queries];
    }
    distance = distsances[q_index];
    neighbour = neighbours[q_index];
}
```

καρτιον

Listing 4: Julia παράδειγμα προσπέλασης πίνακα

```
qIndex = startQueries + q + tid
if tid + q <= totalQueries
    for d = 1:dimensions
       @inbounds SharedQueries[d,tid] = Queries[qIndex, d]
    end
    @inbounds dist = Distances[qIndex]
    @inbounds nb = Neighbours[qIndex]
end
```

**Δείκτης σε γραμμή ή στήλη** Η πολυδιάστατη διευθυνσιοδότηση στην Julia συμπληρώνεται από την μακροεντολή @view, που επιτρέπει τον ορισμό ενός τμήματος πίνακα. Το ανάλογο στην C είναι η χρήση δείκτη (pointer) για τον ορισμό της έναρξης μιας

γραμμής ή μιας στήλης ενός πίνακα. Οι δείκτες περιορίζονται στον ορισμό μονοδιάστατου τμήματο ένος πίνακα σε αντίθεση με τη μακροεντολή που μπορεί να ορίσει οποιονδήποτε υποπίνακα.

Listing 5: Julia παράδειγμα με μακροεντολή @view. Μόνο ο αριθμός του νήματος (thread id) και η μεταβλητή της επανάληψης χρησιμοποιούνται για την προσπέλαση πίνακα.

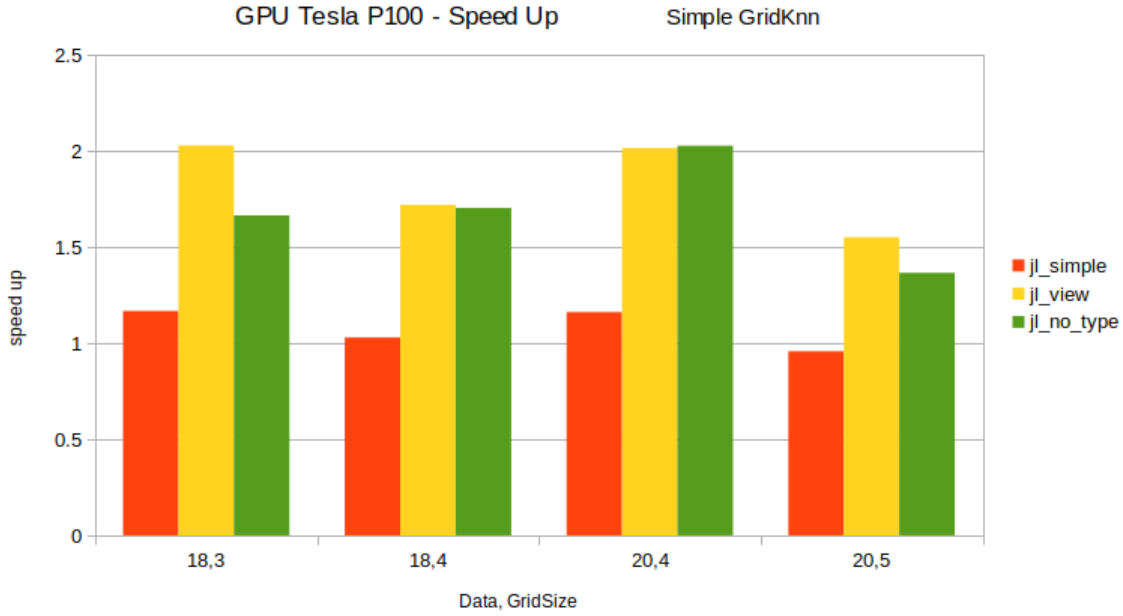
```
# Defining the views
@inbounds Queries = @view devQueries[
        (startQueries+1):(startQueries+totalQueries), :]
@inbounds query = @view SharedQueries[:, tid]
@inbounds Distances = @view devDistances[
        (startQueries+1):(startQueries+totalQueries)]
@inbounds Neighbours = @view devNeighbours[
        (startQueries+1):(startQueries+totalQueries)]


...
# Reading Queries from Global Memory
 if tid + q <= totalQueries
    for d = 1:dimensions
        @inbounds  query[d] = Queries[tid+q, d]
    end
    @inbounds dist = Distances[tid+q]
    @inbounds nb = Neighbours[tid+q]
end
```

Συνηθισμένη πρακτική για κώδικα σε κάρτες γραφικών, είναι ο διαχωρισμός των δεδομένων στα μπλοκ της κάρτας. Επομένως, αντί να χρησιμοποιούμε γραμμική διευθυνσιοδότηση και απόκλιση σε κάθε προσπέλαση πίνακα, μπορούμε να ορίσουμε τον υποπίνακα του κάθε μπλοκ με την μακροεντολή @view. Έτσι, γίνεται πιο κατανοητή η λογική του προγράμματος και δε χρειάζεται να σκεφτόμαστε κάθε φορά την περίπλοκη γραμμική διευθυνσιοδότηση.

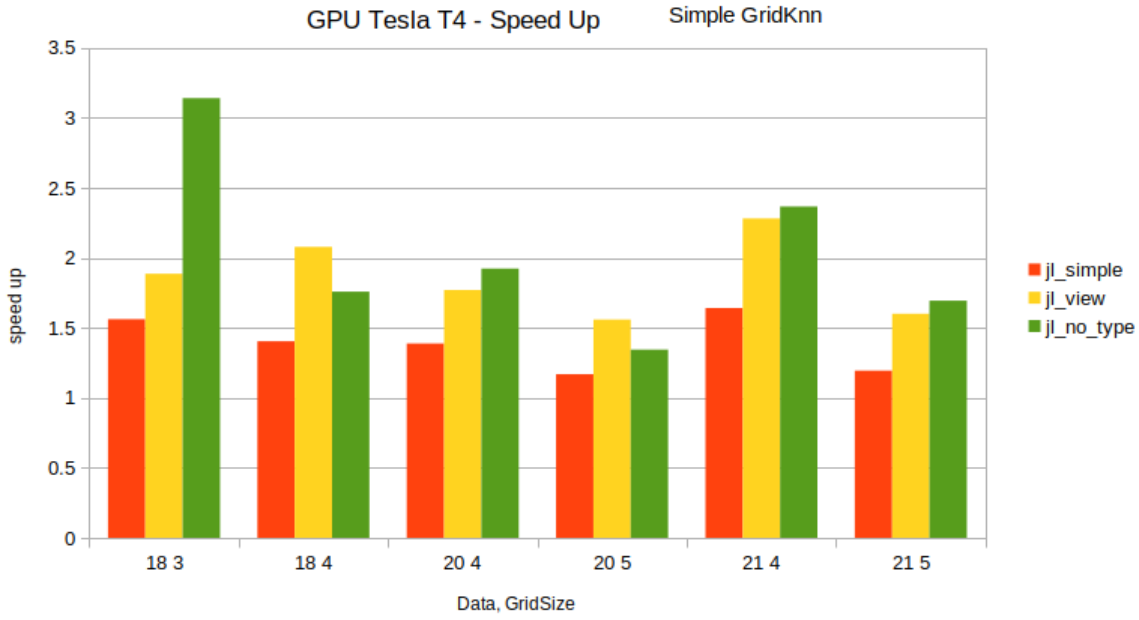
### 0.1.4 Επιδοση και Μετρικές

Ο χρόνος εκτέλεσης κάθε υλοποίησης μετρήθηκε στις κάρτες γραφικών NVIDIA Tesla P100 και T4 για $2^{18}$ τρισδιάστατα σημεία με $2^3$ και $2^4$ μέγεθος πλέγματος, και για $2^20$ τρισδιάστατα σημεία με $2^4$ και $2^5$ μέγεθος πλέγματος

Η επιτάχυνση στα παρακάτω γραφήματα ορίζεται ως $\frac{\text{χρόνος της C}}{\text{χρόνος της Julia}}$ για δεδομένο μέγεθος προβλήματος.



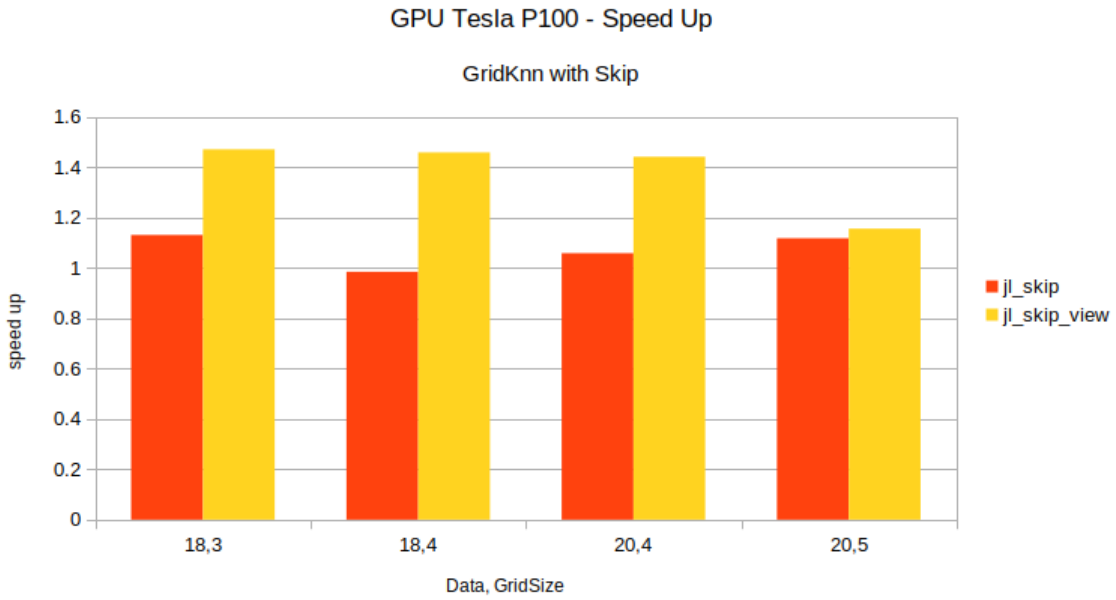

Σχήμα 1: Σύγκριση Julia με C για simple GridKnn



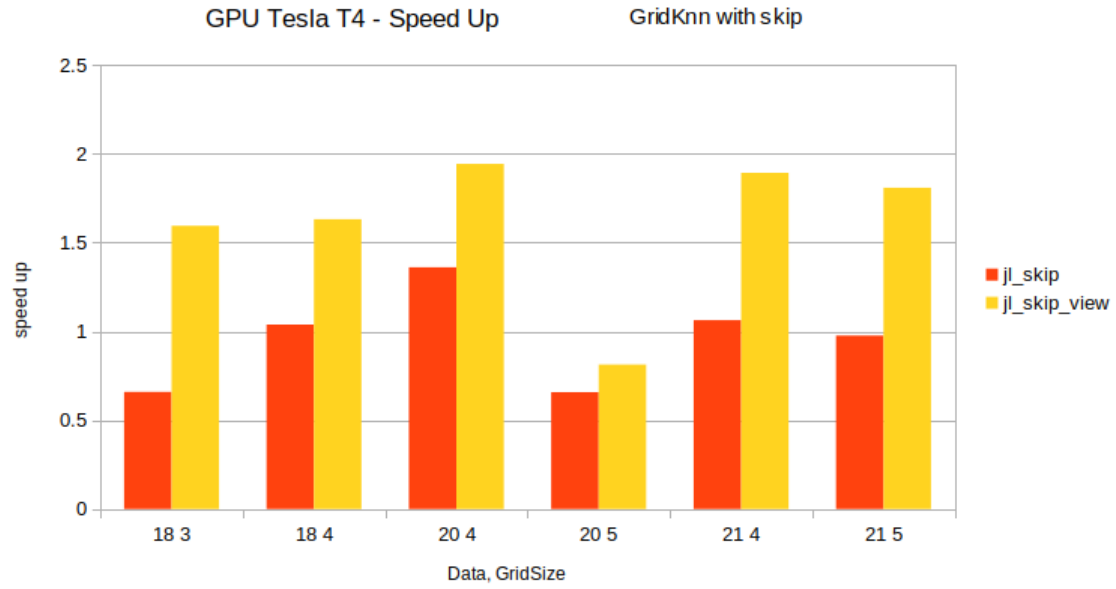

Σχήμα 2: Σύγκριση Julia με C για simple GridKnn

Σε πρώτη ματιά, το πρόγραμμα σε C είναι πάντα πιο αργό από τουλάχιστον ένα πρόγραμμα σε Julia. Τα περισσότερα από τα προγράμματα σε Julia είναι περίπου δύο φορές πιο γρήγορα, ενώ φτάνει έως και 3 φορές πιο γρήγορο.

Ομοίως στην εκδοχή με έλεγχο, η Julia είναι πιο γρήγορο, αλλά το χάσμα είναι μικρότερο.




Σχήμα 3: Σύγκριση Julia με C για GridKnn with Skip

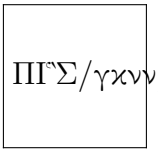



Σχήμα 4: Σύγκριση Julia με C για GridKnn with Skip

Είναι ενδιαφέρον πως μία δυναμική, υψηλού επιπέδου γλώσσα μπορεί να είναι σημαντικά πιο γρήγορη από μια στατική, χαμηλού επιπέδου, σε ένα αντικείμενο που σχετίζεται άμεσα με το hardware.Ακόμα πιο ενδιαφέρον είναι, πώς δυο φαινομενικά ίδιο κώδικες παράγουν διαφορετική PTX γλώσσα μηχανής. όπως φαίνεται από την επίδοση, αλλά και από τις μετρικές παρακάτω.

Οι 3 βασικές μετρικές, που γενικά χρησιμοποιούνται σαν γνώμονας για βελτίωση των προγραμμάτων, έχουν τις ίδιες τιμές ανεξάρτητα από τη γλώσσα που χρησιμοποιήθηκε. Οι μετρικές αυτές είναι *Global Memory Read Efficiency*, *Global Memory Write Efficiency*, *Share Memory Efficiency*. Οι μετρικές αυτές σχετίζονται με τον τρόπο που

γίνεται η προσπέλαση σε πίνακες. Για την καθολική (global) μνήμη, η ανάγνωση και αποθήκευση δεδομένων πρέπει να γίνεται ομαδικά (coalesced) από τα νήματα, ενώ στη μεριζόμενη (shared) μνήμη πρέπει να αποφεύγονται τα bank conflicts. Ήταν αναμενόμενο αυτές οι μετρικές να είναι ίδιες στις δύο γλώσσες.




Σχήμα 5: Ρεγιστερ Υσαγε φορ ΓριδΚνν ιμπλεμεντατιονς.

# Παράρτημα Α΄

# Ακρωνύμια και συντομογραφίες

**LAN** Local Area Network

# Bibliography