5η Εργασία High Performance Computing

Κουκούλης Ιπποκράτης-Βασίλειος 2324

Ηλιάδης Ιωάννης 2308

Στρατηγική παραλληλοποίησης

Βήμα 1)

Για αρχή μεταφέραμε τον κώδικα της CPU στην gpu μεταφέροντας τον υπολογισμό του ιστογράμματος σε έναν kernel στην gpu. Αυτό από μόνο του βελτίωσε αρκετά το χρόνο εκτέλεσης του υπολογισμού ιστογράμματος. Έπειτα δοκιμάσαμε να κάνουμε τις προσθήκες στο ιστόγραμμα ιεραρχικά. Συγκεκριμένα κάθε block έχει ένα δικό του histogram στην shared memory το οποίο υπολογίζει και προσθέτει στο τελικό/global histogram.Παρότι αυτό θεωρητικά θα έπρεπε να μειώσει τα conflicts που γίνονται για την πρόσβαση στα atomic operations, εν τέλει δεν φάνηκε να ωφελεί ιδιαίτερα τις εικόνες που επεξεργαστήκαμε και μάλιστα στις περισσότερες από αυτές είχαμε χειρότερο performance.

Ο λόγος για τον οποίο συμβαίνει αυτό, πιθανώς έχει να κάνει σχέση με την υλοποίηση των shared atomics στην αρχιτεκτονική Kepler που χρησιμοποιούν και οι 2 GPUs του mars και του artemis τα οποία στην πραγματικότητα είναι υλοποιημένα σε software.Πιο καινούριες αρχιτεκτονικές όπως η Maxwell, υλοποιούν τα shared atomics σε hardware και έτσι έχουν καλύτερο performance από τα global atomics, οπότε σε μια πιο καινούρια αρχιτεκτονική ο κώδικας με shared atomics θα είχε καλύτερο performance.Άρα εφόσον τα shared atomics δεν βελτίωσαν το performance (και σε αρκετές περιπτώσεις είχαν χειρότερο performance από τα global atomics) τα απορρίψαμε, τουλάχιστον για τις μετρήσεις στην gpu του artemis.

Έπειτα μεταφέραμε τον υπολογισμό του cdf στην gpu οπού εν τέλει δεν παρατηρήσαμε κάποιο κέρδος στο performance και μάλιστα σε αρκετές περιπτώσεις παρατηρήσαμε χειρότερο performance από αυτό της cpu. Για να υλοποιήσουμε το cdf χρησιμοποιήσαμε τον naive αλγόριθμο του prefix sum.

Την εκτέλεση του υπολογισμού του cdf και εν τέλει του lookup table την αναλαμβάνει ένα block 256 νημάτων. Γενικά αυτός ο υπολογισμός δεν εξαρτάται από το μέγεθος της εικόνας και άρα δεν περιμένουνε να έχουμε κάποιο speedup σε σχέση με την CPU όσο δοκιμάζουμε μεγαλύτερες εικόνες. Αυτό το κομμάτι το θεωρούμε ως ένα “bottleneck” καθώς ο υπολογισμός του στην gpu δεν είναι πιο γρήγορος από την cpu και ο kernel του equalization πρέπει να περιμένει αυτήν την διαδικασία να τελειώσει για να μπορέσει να εκτελεστεί .

Παρόλα αυτά κρατήσαμε αυτή την εκτέλεση στην gpu διότι το επιπρόσθετο overhead των extra memcopies για να μεταφέρουμε τον υπολογισμό του ιστογράμματος από την gpu στην cpu και το lookup table πίσω στην gpu, είναι σχεδόν ίδιο με το overhead της εκτέλεσης όλων αυτών των υπολογισμών στην gpu σύμφωνα με τις δοκιμές που κάναμε. Δοκιμάσαμε επιπλέον να κάνουμε και unroll το loop που υπολογίζει το cdf-prefix sum αλλά δεν παρατηρήσαμε καλύτερο performance.

Κάποιες ακόμη βελτιστοποιήσεις που δοκιμάσαμε ήταν να κάνουμε παράλληλή αναζήτηση στο histogram για να βρούμε το min αντί για ένα απλό loop που εκτελεί ένα thread, ωστόσο αυτό δεν απέφερε καλυτέρα αποτελέσματα σε κάθε σενάριο σε σύγκριση με την σειριακή αναζήτηση στην gpu οπότε το απορρίψαμε.

Δοκιμάσαμε επίσης να βάλουμε το lookup table σε shared μνήμη στον kernel που εκτελεί το equalization, για να εκμεταλλευτούμε το γεγονός ότι γίνονται πολλά accesses. Ωστόσο και αυτό δεν είχε τα επιθυμητά αποτελέσματα, καθώς δεν παρατηρήσαμε κάποια διαφορά στον χρόνο εκτέλεσης του kernel, και σε ορισμένες περιπτώσεις ήταν χειρότερος του αρχικού. Αυτό συμβαίνει πιθανώς επειδή το random access pattern που γίνεται στην shared μνήμη που βάλαμε οδήγησε σε αρκετά bank conflicts και εν τέλει τα requests στην μνήμη έγιναν serialized. Αυτό είχε ως αποτέλεσμα να μην έχει διαφορά το access latency στη shared μνήμη από τα αντίστοιχο στην global μνήμη, όποτε τελικά αφαιρέσαμε την shared μνήμη από τον kernel.

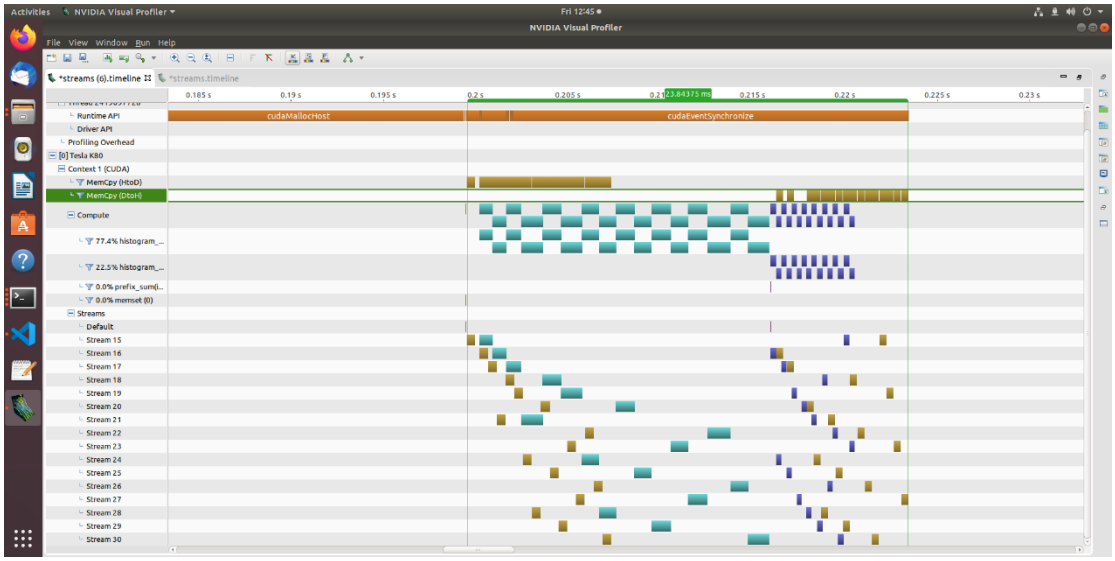
Γενικά και ο equalization kernel είχε καλύτερο χρόνο εκτέλεσης από το αντίστοιχο κομμάτι κώδικα που εκτελεί την ίδια διαδικασία στην gpu.

Επίσης δοκιμάσαμε διάφορα μεγέθη blocks στην γεωμετρία (128,256,512,1024) αλλά το performance παρέμενε το ίδιο οπότε τελικά χρησιμοποιούμε 1024 by default. Το occupancy με βάση και τα registers που δείχνει ότι χρησιμοποιούμε ο compiler (τα οποία είναι λίγα) και με βάση την shared μνήμη που χρησιμοποιείται μόνο στο block που υπολογίζει το cdf-lookup table, θα πρέπει να είναι υψηλό και σχετικά κοντά στο 100%.

Βήμα 2)

Για να μειώσουμε τον χρόνο εκτέλεσης των kernels και τον χρόνο μεταφορών των δεδομένων από και προς την gpu δοκιμάσαμε με την χρήση streams να κάνουμε overlap μεταξύ της μεταφοράς δεδομένων και της εκτέλεσης των kernels. Χρησιμοποιούμε pinned memory μέσω του cudaHostAlloc api call για την δέσμευση της μνήμης οπού επιστρέφει η gpu το τελικό αποτέλεσμα στον host ώστε να μπορούμε να κάνουμε overlap τις μεταφορές δεδομένων από το device προς το host με εκτελέσεις kernels και για να αυξήσουμε το throughput της κάθε μεταφοράς .

Μπορούμε να χρησιμοποιήσουμε cudaHostAlloc και για την μνήμη που βάζουμε το input αλλά αυτό δεν αποδίδει κάποιο ιδιαίτερο κέρδος καθώς ούτως η αλλιώς στην αρχή του προγράμματος γράφουμε σε αυτή την μνήμη οπότε έχουμε φέρει τις αντίστοιχες σελίδες στην μνήμη και υπάρχει καλή πιθανότητα αυτές οι σελίδες να είναι διαθέσιμες. Η cudaHostAlloc βέβαια έχει μεγαλύτερο κόστος συνολικά σε χρόνο από ότι η malloc όπως επιβεβαιώσαμε και από τις μετρήσεις στον χρόνο των malloc αλλά σε συνδυασμό με το κέρδος που περνούμε από τα overlap έχουμε εκτός από καλύτερους χρόνους εκτέλεσης kernel και μεταφορών και καλύτερούς χρόνους εκτέλεσης όλου του προγράμματος.



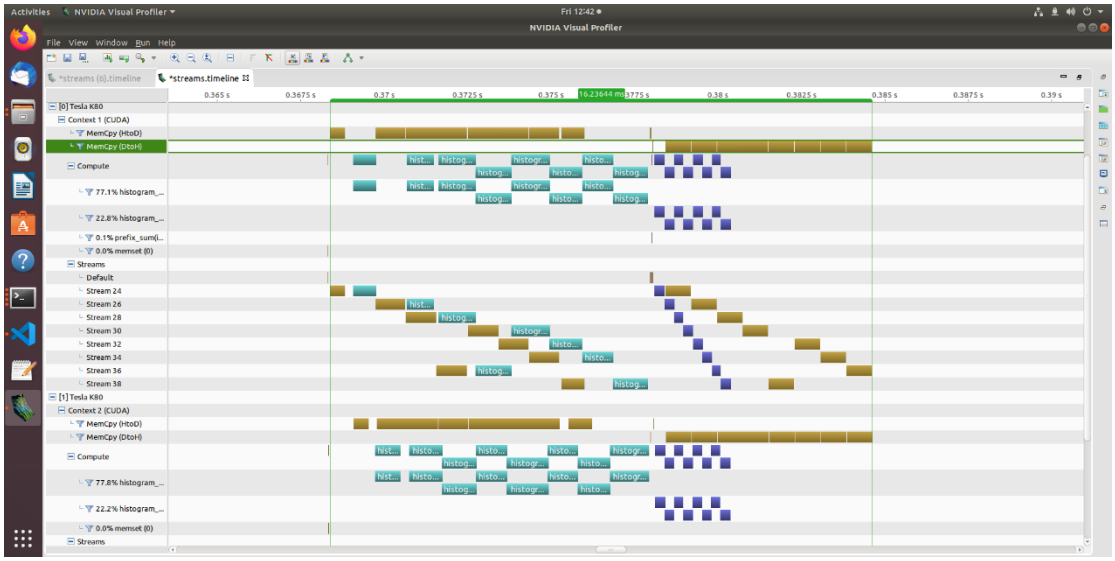
Εικόνα 1 Timeline για planet_surface.png 16 streams

Βήμα 3)

Στο τελικό βήμα δοκιμάσαμε να χρησιμοποιούμε και τις 2 gpus που έχει εσωτερικά η Tesla K80 για την εκτέλεση των kernel και την μεταφορά των δεδομένων. Να σημειώσουμε εδώ πώς αυτό γενικά έχει μια «ποινή» στον συνολικό χρόνο εκτέλεσης του προγράμματος καθώς το πρώτο cuda call (και συγκεκριμένα το πρώτο cudamalloc) που γίνεται για το device 1 έχει ένα εξτρά κόστος σε χρόνο, όπως έχει και το πρώτο cuda call που γίνεται στο device 0 για το initialization του context. Παρόλα αυτά, αυτό το overhead είναι σταθερό και άρα όσο δοκιμάζουμε μεγαλύτερες εικόνες το speedup από την χρήση των 2 gpus θα αυξάνεται και θα ξεπεράσει αυτό το overhead.

Γενικά και στα προηγούμενα βήματα ο χρόνος εκτέλεσης όλου του προγράμματος είναι χειρότερος από αυτόν της cpu για τις εικόνες που δόθηκαν στην εργασία καθώς δεν είναι αρκετά μεγάλες ώστε το speedup που έχουμε στον χρόνο εκτέλεσης των kernels και των μεταφορών να ξεπεράσει το overhead του context initialization της cuda. Με μεγαλύτερες εικόνες (της τάξης των 100 Mbytes) που δοκιμάσαμε η gpu σε όλα τα βήματα που αναφέραμε έχει καλυτέρους χρόνους εκτέλεσης όλου του προγράμματος από ότι η CPU.

Με την προσθήκη αυτή όπως φαίνεται και στο timeline παρακάτω πλέον έχουμε και overlaps μεταξύ εκτελέσεων kernel που εκτελούνται σε διαφορετικά devices που δεν είχαμε πριν αλλά και παράλληλες μεταφορές δεδομένων (αν και έχουν το μισό throughput από αυτές που παρατηρήσαμε με χρήση ενός device). Αυτό οδηγεί σε βελτίωση του συνολικού χρόνου εκτέλεσης kernels και μεταφορών εξαιτίας των επιπλέον overlaps εφόσον αξιοποιούμε 2 devices.

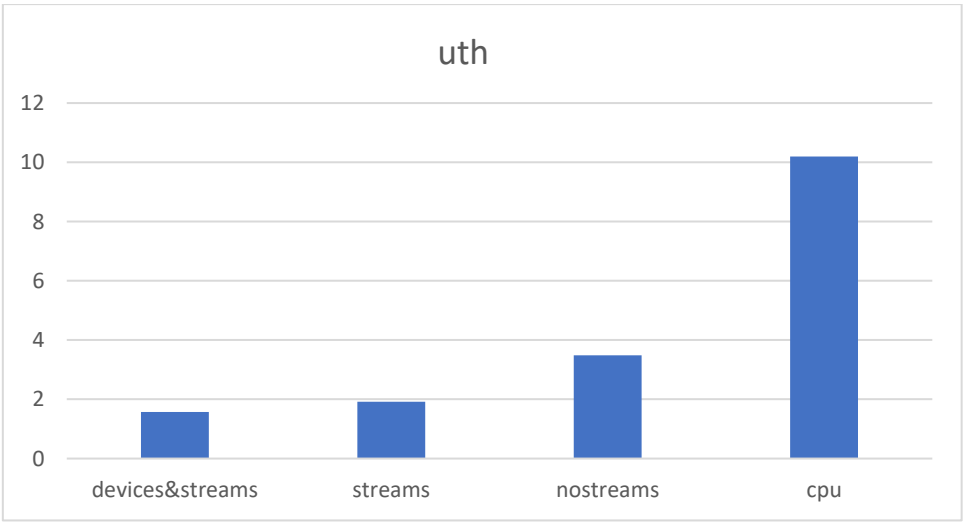


Εικόνα 2 Timeline τελικού κώδικα planet_surface.png 8+8 streams

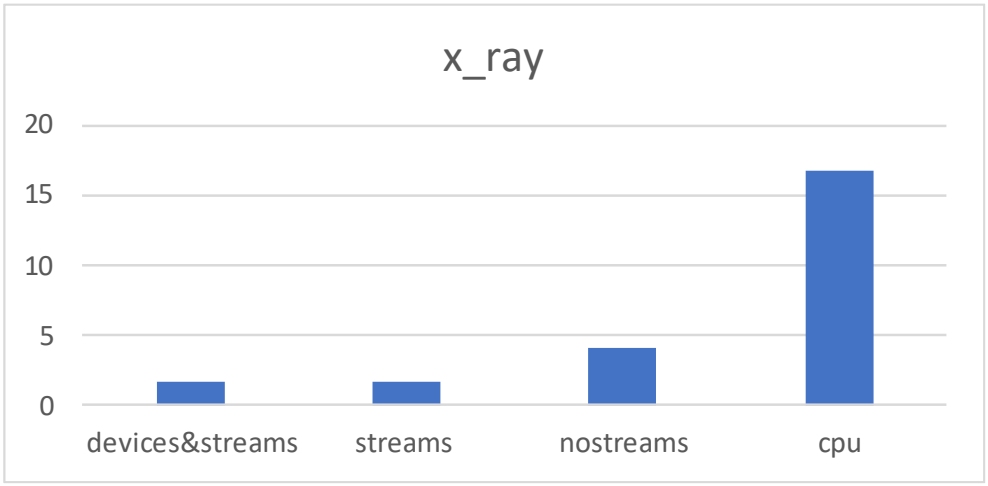
Ο κώδικας μπορεί να τρέξει και σε ένα device αντί για 2 που τρέχει by default θέτοντας το multi=1.

Παρακάτω ακολουθούν διαγράμματα με τους συνολικούς χρόνους εκτέλεσης των kernels και των μεταφορών δεδομένων από και προς την gpu για όλα τα βήματα σε σύγκριση με τους αρχικούς χρόνους εκτέλεσης της εξισορρόπησης ιστογράμματος στην cpu.

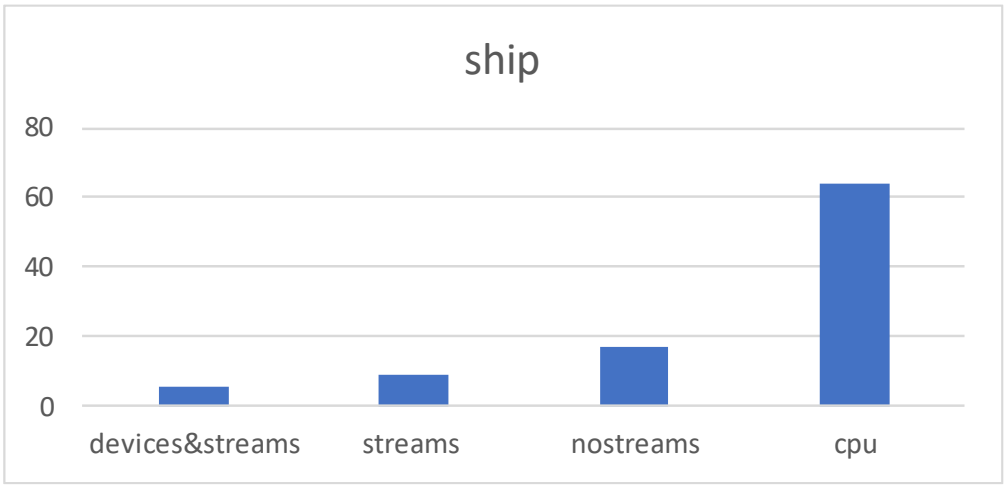
Όλοι οι χρόνοι είναι σε milliseconds, και οι μετρήσεις βρίσκονται αναλυτικά στο excel spreadsheet.



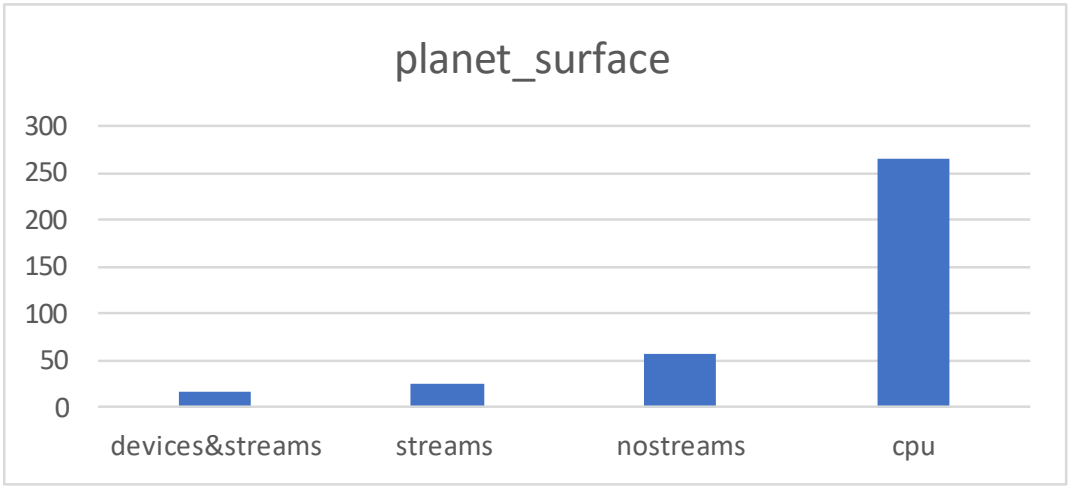
Εικόνα 3 Χρόνοι εκτέλεσης (ms) για uth



Εικόνα 4 Χρόνοι εκτέλεσης (ms) για x_ray



Εικόνα 5 Χρόνοι εκτέλεσης (ms) για ship



Εικόνα 6 Χρόνοι εκτέλεσης (ms) για planet_surface