4η Εργασία High Performance Computing

Κουκούλης Ιπποκράτης-Βασίλειος 2324

Ηλιάδης Ιωάννης 2308

Ερώτημα 1)

Για να βελτιστοποιήσουμε τις προσπελάσεις μνήμης προς την global memory, κάθε block πλέον έχει shared memory.

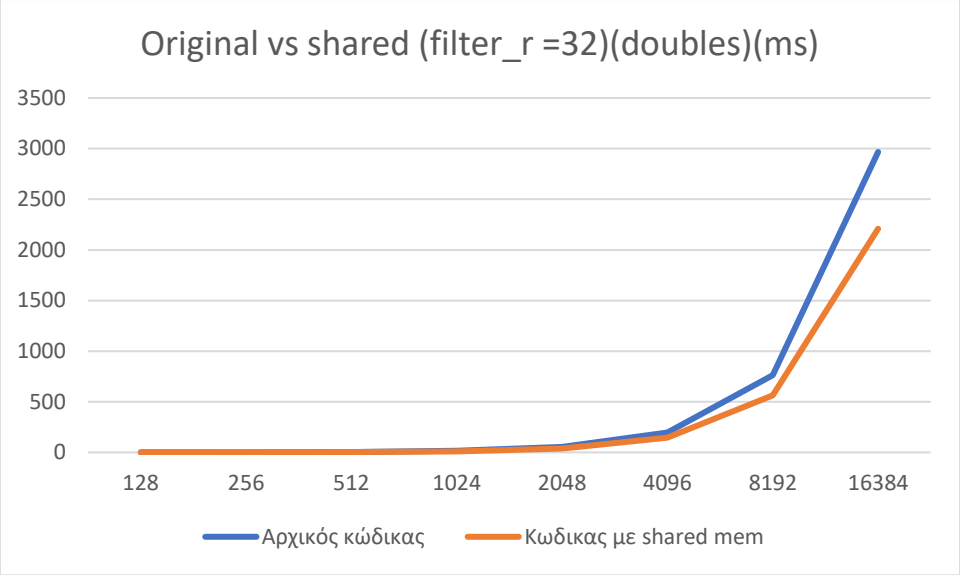
Συγκεκριμένα ο ConvolutionRowKernel αποθηκεύει στην shared memory ένα κομμάτι μιας γραμμής του πίνακα (ή δυνητικά μια ολόκληρη γραμμή αν μπορεί αν την χωρέσει) και υπολογίζει το convolution για τα στοιχεία αυτού του κομματιού. Το μέγεθος της cache είναι row_tile_size + 2*filter_radius οπού row_tile_size είναι και ο αριθμός των convolutions που θα κάνουμε.

Ο ConvolutionColumnKernel αποθηκεύει στην shared memory ένα ορθογώνιο block-κομμάτι του πίνακα και υπολογίζει το convolution αυτών των στοιχείων. Το μέγεθος της cache είναι (col_tile_size+2*filter_radius)*col_tile_size και συνολικά γίνονται col_tile_size^2 convolutions.

Στην 1η περίπτωση διαλέξαμε να οργανωθεί η shared memory με αυτόν τον τρόπο, διότι μπορούν να γίνουν περισσότεροι υπολογισμοί convolutions για τα δεδομένα που φέρνουμε στην shared cache από το να είχαμε μια cache που αποθηκεύει ένα ορθογώνιο block του πίνακα.

Στην 2η περίπτωση παρότι όσο μεγαλώνει η ακτίνα αναγκαζόμαστε να μικραίνουμε το col_tile_size και αρά γίνονται και λιγότερα convolutions για τα δεδομένα που φέρνουμε στην shared memory εν τέλει μας βολεύει αυτή η οργάνωση για να μπορούν να γίνονται coalesced reads εφόσον φέρνουμε συνεχόμενες στήλες από την global memory.

Το φίλτρο επίσης αποθηκεύεται σε constant μνήμη σταθερού μεγέθους(που θα είναι και η max ακτίνα που μπορεί να χωρέσει). Κάναμε πειράματα μέχρι ακτίνα 256 .Γενικώς παρατηρείται σημαντική βελτίωση στους χρόνους εκτέλεσης όπως φαίνεται και στο παρακάτω γράφημα και τα αποτελέσματα είναι αναλυτικά στο spreadsheet.



Εικόνα 1.Χρόνος εκτέλεσης-αρχικός κώδικας

Ερώτημα 2)

Παρατηρούμε ότι ο χρόνος εκτέλεσης των kernels αυξάνεται όπως ήταν αναμενόμενο με την αύξηση της ακτίνας του φίλτρου, καθώς όσο αυξάνεται η ακτίνα κάθε block του kernel υπολογίζονται ολοένα και λιγότερα στοιχεία του τελικού πίνακα καθώς πρέπει να φέρνουμε στην shared memory ολοένα και περισσότερά στοιχεία που απαιτούνται για τον υπολογισμό ενός και μόνο convolution.

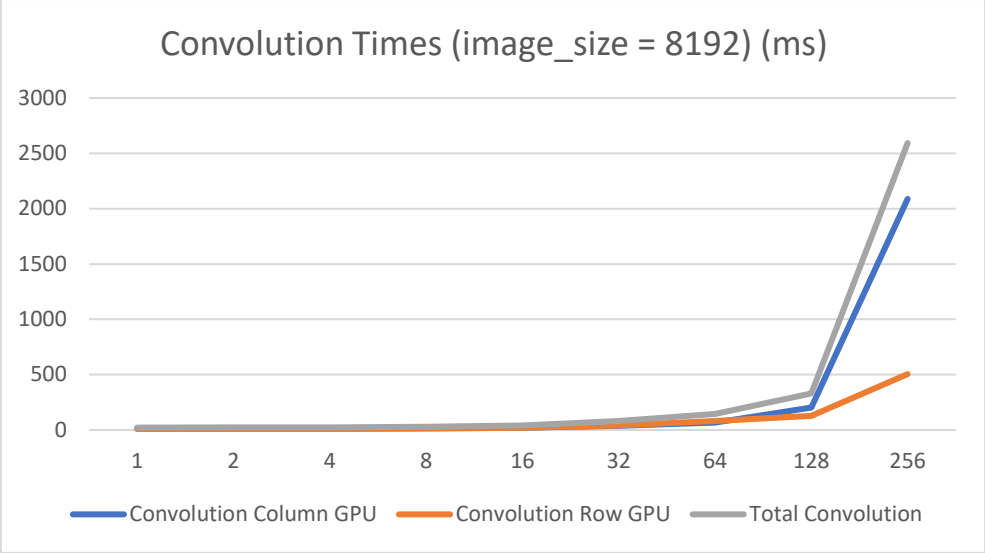
Επιπλέον επειδή η shared memory είναι περιορισμένη, όσο αυξάνεται η ακτίνα αντίστοιχα μειώνεται το tile_size που μπορεί να χωρέσει μέσα στην cache, καθώς το μέγεθος της cache εξαρτάται και από το filter_radius.

Αυτό γίνεται πιο εμφανές και στον ConvolutionColumnKernel οπού εκεί η αύξησή της ακτίνας οδηγεί εντέλει το tile_size να μειώνεται πολύ πιο γρήγορα και για αυτό γίνεται αρκετά πιο αργός από τον ConvolutionRowKernel στις μεγαλύτερες ακτίνες.

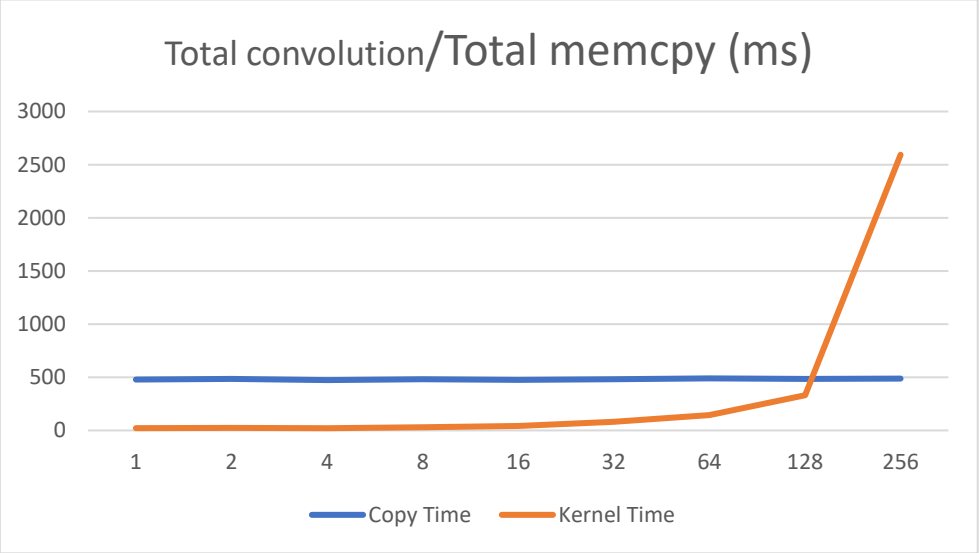
Η αύξηση της ακτίνας προφανώς οδηγεί στο να γίνονται περισσότερα global memory accesses καθώς τα δεδομένα που φέρνει κάθε block στην shared memory του γίνονται περισσότερο overlapped όσο αυξάνεται η ακτίνα και άρα κάθε block φέρνει για λογαριασμό του στην shared memory στοιχεία τα οποία έχουν ήδη φέρει άλλα block.

Ο χρόνος μεταφοράς δεδομένων από και προς την GPU παραμένει σταθερός για διάφορα μεγέθη ακτίνας όπως είναι αναμενόμενο.

Τα δεδομένα των μεταφορών και των χρόνων εκτέλεσης των kernels συλλέχθηκαν από το report του nvprof.



Εικόνα 2.Χρόνοι εκτέλεσης των kernels



Εικόνα 3.Σχέση χρόνου εκτέλεσης kernels/χρόνου μεταφορών

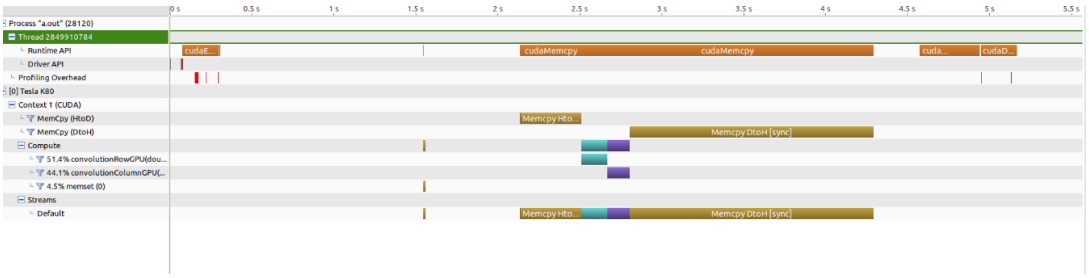
Ερώτημα 3-4)

Πλέον μεταφέρουμε στην gpu γραμμές/ταινίες του αρχικού πίνακα οι οποίες χωράνε μέσα στην gpu.

Ένας 16384X16384 πίνακας χωράει ολόκληρος (τα 3 cudamalloc που γίνονται για αυτόν δηλαδή χωράνε) στην μνήμη της gpu του artemis (~11.9 gb μνήμης).

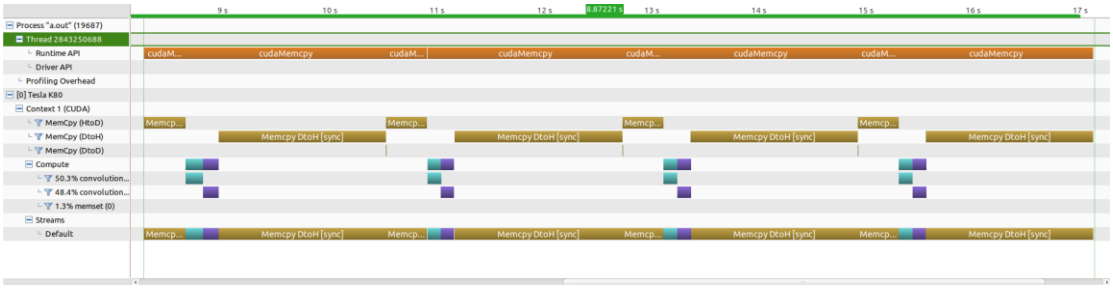
Αντίστοιχα, ένας 32768x32768 πίνακας για να χωρέσει στη μνήμη της gpu, σπάει σε 4 chunks των 8192 γραμμών και στη συνέχεια κάνουμε σταδιακά τους υπολογισμούς και τις μεταφορές.

Από τον visual profiler οι μεταφορές των δεδομένων για 16384



Εικόνα 4.Timeline 16384 ακτίνα 32 conv3.cu

Και για 32768:



Εικόνα 5.Timeline 32768 ακτίνα 32 conv3.cu

Για να επιτύχουμε overlaps μεταξύ των εκτελέσεων kernels και των μεταφορών μνήμης (ιδίως για τις μεταφορές device_to_host) χρειάστηκε να χρησιμοποιήσουμε το cudaMallocHost call και να χρησιμοποιήσουμε pinned memory για το output της gpu. Επειδή η gpu στο artemis υποστηρίζει hyper-q και έχει 2 cory engines στο device που τρέχουμε, μπορούμε να έχουμε και overlaps μεταξύ μεταφορών d_to_h και h_to_d.

Για να εκτελεστεί ένας convolutionColumnKernel στο i stream θα πρέπει να περιμένει να έχει τελειώσει ο convolutionRowKernel του i-1 stream(εκτός από το 1ο stream το οποίο φέρνει στο device περισσότερα δεδομένα από τα αλλά streams για να μπορέσει να εκτελέσει το convolutionColumnKernel οι υπόλοιποι περνούν τα αποτελέσματα από το i-1 stream με memcpy device_to_device) .

Με 8 streams έχουμε σημαντική βελτίωση στον χρόνο διεκπεραίωσης καθώς πλέον έχουμε 3-way overlaps αλλά και οι μεταφορές device_to_host είναι πολύ πιο γρήγορες εξαιτίας του ότι χρησιμοποιούμε pinned memory.

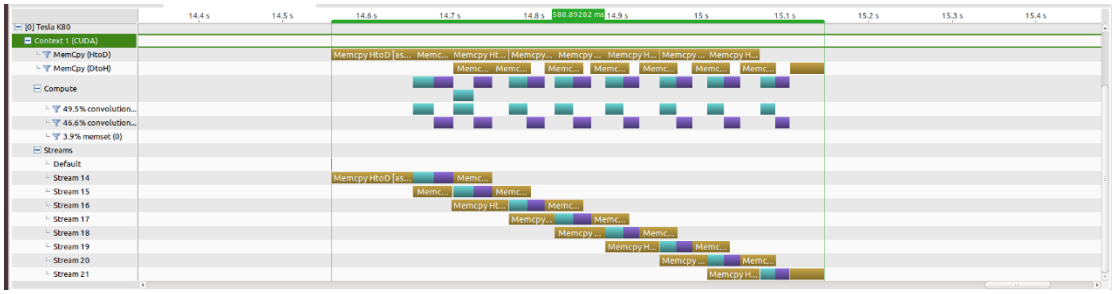
Βέβαια από τον nvprof παρατηρούμε πως τα cudaMallocHost και τα αντίστοιχη free calls έχουν πολύ μεγαλύτερους χρόνους εκτέλεσης από τα αντίστοιχα default api calls για διαχείριση μνήμης.

Και πάλι όμως συνδυαστικά το κόστος τους μαζί με τον χρόνο που χρειάστηκε για να ολοκληρωθούν οι μεταφορές και τα convolutions είναι μικρότερος από τον αρχικό χρόνο εκτέλεσης εξαιτίας των overlaps.

Τα host_to_device copies είναι γενικώς (στον προηγούμενο κώδικά) πιο γρηγορά από τα αντίστοιχα device_to_host copies. Αυτό συμβαίνει πιθανώς επειδή στην μνήμη του input έχουμε γράψει ήδη πριν ξεκινήσουμε τα memcopies και επομένως υπάρχει μια αρκετά καλή πιθανότητα να έχουμε αυτές τις σελίδες μνήμης διαθέσιμες (μπορεί να γίνουν όμως swapped out στον σκληρό δίσκο καθώς δεν είναι pinned). Επίσης τα device_to_host

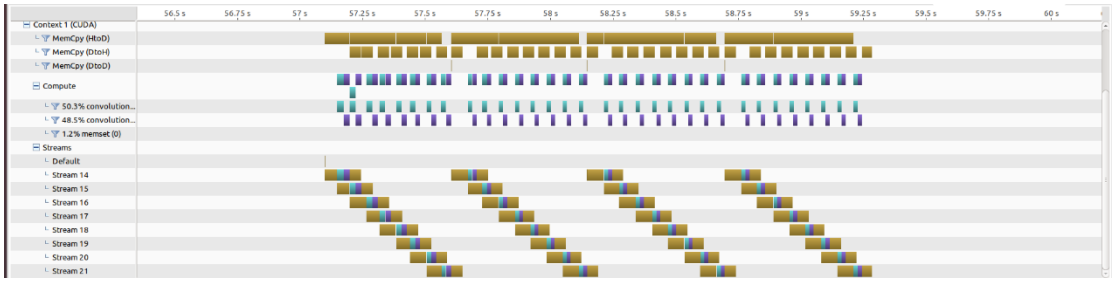
memcopies όπως παρατηρήσαμε σε pinned memory είναι πιο γρήγορα από τα αντίστοιχα host_to_device από pageable memory.

Τα αντίστοιχα timelines από τον nvvp για 16384*16384 με ακτινα 32:



Εικόνα 6.Timeline 16384 ακτίνα 32 conv4.cu

Και για 32786*32786 με ακτίνα 32:



Εικόνα 7.Timeline 32768 ακτίνα 32 conv4.cu

Και οι αντίστοιχές μετρήσεις χρόνων (και για τα malloc από τον nvprof) βρίσκονται στο excel.

Δοκιμάσαμε επίσης να κάνουμε mallochost το input και με αυτή την αλλαγή είχαμε ακόμα μικρότερους χρόνους μεταφορών host_to_device (αλλά όχι επιπλέον overlaps καθώς ο συγχρονισμός που αναφέραμε πάνω εμποδίζει να γίνει κάτι καλύτερο με τα overlaps) με αποτέλεσμα να έχουμε συνολικά ελαφρώς καλύτερους συνολικούς χρόνους εκτέλεσης μεταφορών και kernels. Ωστόσο δεν κρατήσαμε αυτή την αλλαγή καθώς παρατηρήσαμε μεγάλη αύξηση στον χρόνο εκτέλεσης των mallochost calls με αποτέλεσμα ο συνολικός χρόνος (αθροιστικά με τον χρόνο των malloc) να είναι χειρότερος σε σύγκριση με τον αντίστοιχο χρόνο του κώδικά χωρίς χρήση streams.

Βέβαια αν γίνονται συνεχώς feed από τον δίσκο δεδομένα από διαφορετικά αρχεία φωτογραφιών, τότε σε αυτό το σενάριο θα μας σύμφερε η $2^{η}$ υλοποίηση καθώς θα αποπληρώναμε το κόστος του extra malloc με περισσότερες μεταφορές.

Ερώτημα 5)

Πλέον μπορούμε επεξεργαστούμε μεγαλύτερες εικόνες (αρκεί βεβαία να χωράει έστω και μια γραμμή του πίνακα + 2*filter_r γραμμές + 2*filter_r στήλες στην μνήμη της gpu).

Οπότε αν ικανοποιείται αυτός ο περιορισμός, τότε περιοριζόμαστε από την μνήμη που δεσμεύουμε στην cpu. Στο artemis μπορέσαμε να εκτελέσουμε μέχρι 65536*65536 πίνακα με ακτίνα 32 καθώς τα mallocs που γίνονται για αυτό χωράνε στην μνήμη της cpu. Με μεγαλύτερο πίνακα ξεπερνάμε τα 128 gb ram του artemis και μας τελειώνει η διαθέσιμη μνήμη.