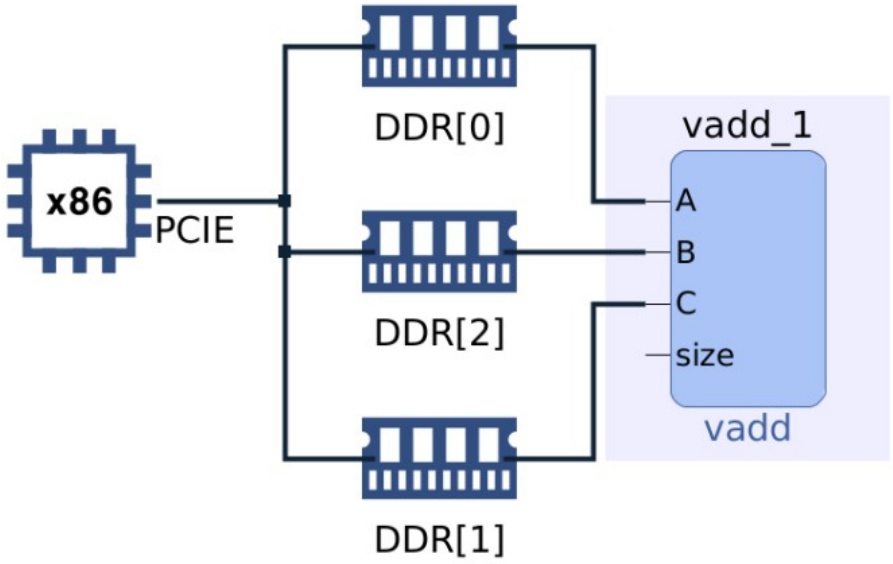
Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Αρχιτεκτονική Προηγμένων Υπολογιστών και Επιταχυντών 2024-2025
Αναφορά 3ου Εργαστηρίου



Ομάδα 3
Γαλάνη Δήμητρα, ΑΕΜ 10331
Σπανόπουλος Γρηγόριος, ΑΕΜ 10459

Διδάσκων: Παπαευσταθίου Ιωάννης

20 Δεκεμβρίου 2024

Περιγραφή κώδικα wide_vadd

Στον κώδικα υλοποιείται η πρόσθεση διανυσμάτων για επιτάχυνση FPGA. Το FPGA πρόκειται να επεξεργαστεί δύο διανύσματα εισόδου (in1 και in2) που είναι αποθηκευμένα στην καθολική μνήμη και γράφει το άθροισμά τους σε ένα διάνυσμα εξόδου (out), το οποίο μεταφέρεται και αυτό στην καθολική μνήμη. Σε κάθε iteration του εξωτερικού loop αποθηκεύονται στην τοπική μνήμη του πυρήνα 64 στοιχεία, όσο και το BUFFER_SIZE, των δύο εισόδων. Στην συνέχεια στο εσωτερικό loop add γίνεται η πρόσθεση των αριθμών, που περιέχει κάθε στοιχείο, της πρώτης εισόδου με τον αντίστοιχο ακέραιο της δεύτερης. Κάθε στοιχείο αποτελείται από 512 bits δηλαδή περιέχει 16 ακεραίους αριθμούς (που έχουν 32 bit). Οι δεκαέξι προσθέσεις γίνονται ταυτόχρονα χάρη στο unroll και με την κατάλληλη προσπέλαση των δεδομένων με την χρήση της συνάρτησης range. Στο τέλος μιας επανάληψης το αποτέλεσμα μεταφέρεται πίσω στην καθολική μνήμη, οπότε στο timeline παρατηρείται ότι μετά από δύο reads πραγματοποιείται ένα write στην καθολική μνήμη. Τα pragmas dataflow και stream είναι αυτά που επιτρέπουν την παράλληλη εκτέλεση της ανάγνωσης, της γραφής και του υπολογισμού της πρόσθεσης καθώς το ένα διαιρεί τον κώδικα σε ανεξάρτητα tasks και το άλλο υλοποιεί τους buffers ως FIFO ουρά, προκειμένου να μπορούν να ξεκινήσουν οι υπολογισμοί χωρίς να έχει τελειώσει το read.

Βελτιστοποίηση κώδικα πολλαπλασιασμού πινάκων

Σε αυτό το εργαστήριο βελτιστοποιείται η μεταφορά των δεδομένων καθώς οι αριθμοί αποθηκεύονται σε στοιχεία των 512 bit. Επιλέγουμε να χρησιμοποιήσουμε buffers με μήκος 16 αντί για 64. Οπότε οι buffers θα περιέχουν 16 στοιχεία, όπου κάθε στοιχείο είναι μεγέθους uint512_dt, δηλαδή 512 bit. Στα 512 bits θα υπάρχουν 16 ακέραιοι αριθμοί, 16x32=512, όποτε στους buffers a_local και b_local αποθηκεύονται εξ'ολοκλήρου οι πίνακες A, B, εφόσον θα έχουμε 16 elements που αποθηκεύουν 16 ακεραίους ο καθένας (16x16=64), δηλαδή ένα στοιχείο θα περιέχει μια γραμμή ή αντίστοιχα στήλη του πίνακα.

Αρχικά, φορτώνουμε όλα τα δεδομένα των πινάκων A και B από την global memory και έπειτα κάνοντας κατάλληλη προσπέλαση των δεδομένων κάθε element πραγματοποιούμε τον πολλαπλασιασμό των πινάκων. Τα αποτελέσματα τα αποθηκεύουμε σε έναν buffer που έχει το διπλάσιο μέγεθος των A και B προκειμένου να μην υπάρχει overflow κατά τον πολλαπλασιασμό. Οπότε κάθε element συγκρατεί 8 ακεραίους όπου ο κάθε ακέραιος έχει μέγεθος 64 bits.

Στην μεριά του host υπολογίζουμε τον transpose πίνακα του B και μεταφέρουμε αυτόν στην global memory, εφόσον κατά τον πολλαπλασιασμό πινάκων πολλαπλασιάζονται οι γραμμές του A με τις στήλες του B. Οπότε κάθε στοιχείο των 512 bit θα περιέχει μια στήλη του B πίνακα αντί για μια γραμμή.

Όσον αφορά τα pragmas, όπως και στα προηγούμενα εργαστήρια εισαγάγαμε unroll στις βρόχους, προκειμένου να εκτελούνται ταυτόχρονα και οι 16 επαναλήψεις μαζί. Στο πλαίσιο του εργαστηρίου για το συγκεκριμένο παράδειγμα όπου δεν έχουμε πολλά δεδομένα μια τέτοια υλοποίηση δεν μας παρουσίασε πρόβλημα, καθώς δεν υπερβαίνουμε τον αριθμό των διαθέσιμων resources. Ωστόσο μία τέτοια λύση δεν ενδείκνυται για scaling.

Αποτελέσματα πριν την εισαγωγή πολλαπλών banks της DDR μνήμης

Ακολουθούν εικόνες απευθείας από το vitis Analyzer:

Kernel	Enqueues	Total Time (ms)	Min Time (ms)	Avg Time (ms)	Max Time (ms)
vadd	1	0.009	0.009	0.009	0.009

Σχήμα 1: Kernel Execution Time

Top Data Transfer: Kernels from/to Global Memory

Compute Unit	Device	Number of Transfers	Avg Bytes per Transfer	Transfer Efficiency (%)	Total Data Transfer (MB)	Total Write (MB)	Total Read (MB)	Total Transfer Rate (MB/s)
vadd_1	xilinx_u200_gen3x16_xdma_2_202110_1-0	2	1024.000	25.000	0.002	0.002	0.000	11592.500

Σχήμα 2: Top Kernel Data Transfer

Data Transfer: Host from/to Global Memory

Context:Number of Devices	Transfer Type	Number of Buffer Transfers	Transfer Rate (MB/s)	Avg Bandwidth Utilization (%)	Avg Size (KB)	Total Time (ms)	Avg Time (ms)
context0:1	READ	1	0.100	N/A	2.048	N/A	N/A
context0:1	WRITE	1	0.050	N/A	2.048	N/A	N/A

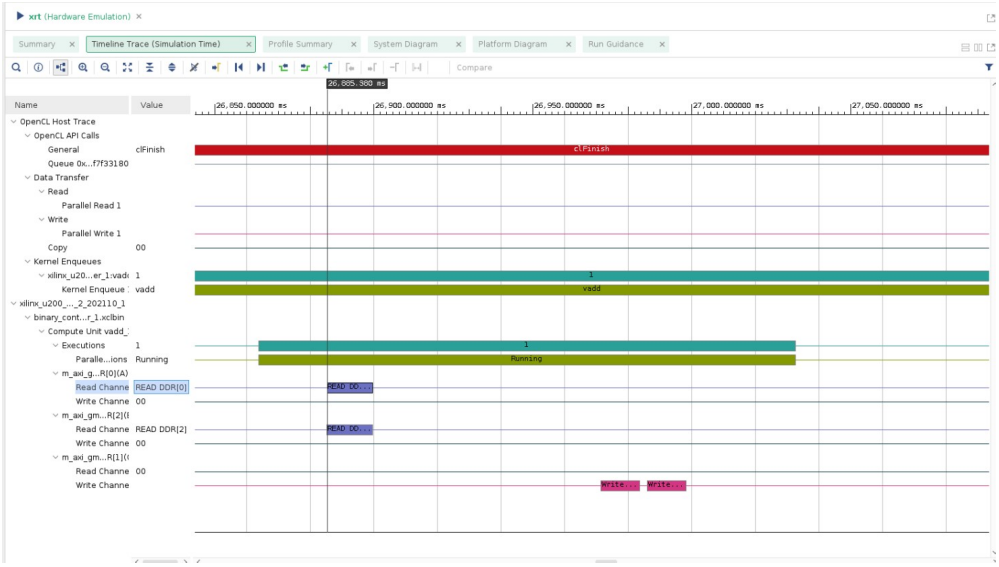
Σχήμα 3: Host Transfer

Εκμετάλλευση των banks της DDR μνήμης

Επίσης στο εργαλείο του vitis κάναμε τις κατάλληλες ρυθμίσεις προκειμένου να εκμεταλλευτούμε τα τρία DDR banks, ένα για κάθε πίνακα, προκειμένου να είναι επιτρεπτή η παράλληλη ανάγνωση των πινάκων A και B καθώς και η παράλληλη εγγραφή στον πίνακα C. Υπήρχε διαφορά μόνο στο Kernel Execution Time:

Kernel	Enqueues	Total Time (ms)	Min Time (ms)	Avg Time (ms)	Max Time (ms)
vadd	1	0.006	0.006	0.006	0.006

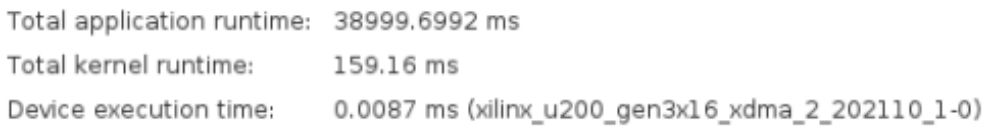
Σχήμα 4: Kernel Execution Time Using Multiple Memory Banks



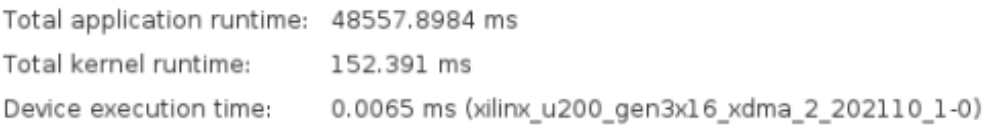
Σχήμα 5: Στο παραπάνω timeline φαίνεται ότι πράγματι τα δεδομένα εισάγονται παράλληλα από τα δύο banks.

Επιτάχυνση της υλοποίησης

Τέλος, υπολογίσαμε δύο επιταχύνσεις. Η μία είναι σε σχέση με το προηγούμενο εργαστήριο και η άλλη συγκρίνει τα execution times μεταξύ των δύο μεθόδων βελτιστοποίησης αυτού του εργαστηρίου. Για μεγαλύτερη ακρίβεια στον υπολογισμό της επιτάχυνσης, αντί να πάρουμε τα execution times από την καρτέλα Kernel & Compute Units→Kernel Execution, λάβαμε υπόψιν μας τις τιμές του Device execution time από το Summary. Παρακάτω παραθέτουμε τις τιμές αυτές που εμφανίζει το εργαλείο vitis καθώς και τις επιταχύνσεις που προκύπτουν.



Σχήμα 6: Device execution time πριν την εισαγωγή των πολλαπλών banks.



Σχήμα 7: Device execution time με πολλαπλά banks.

$$\alpha = \frac{execution\ time\ from\ lab\ 2}{execution\ time\ with\ multiple\ memory\ \ banks} = \frac{0.009}{0.0065} = 1.38 = 138\% \tag{1}$$

$$\alpha = \frac{execution\ time\ with\ one\ memory\ \ bank}{execution\ time\ with\ multiple\ memory\ \ banks} = \frac{0.0087}{0.0065} = 1.33 = 133\% \tag{2}$$

Γενικώς παρατηρούμε ότι δεν υπάρχει πολύ μεγάλη επιτάχυνση και στις δύο περιπτώσεις καθώς ο όγκος των δεδομένων είναι μικρός.