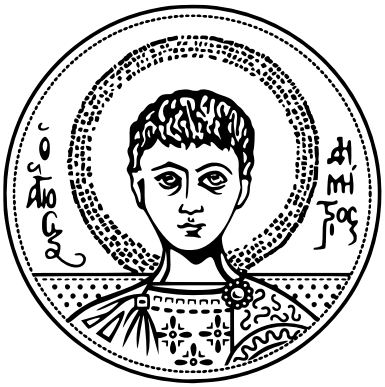

Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πολυτεχνική Σχολή
Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Αρχιτεκτονική Προηγμένων Υπολογιστών και Επιταχυντών 2024-2025
Αναφορά 2ου Εργαστηρίου

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

Διδάσκων: Παπαευσταθίου Ιωάννης

20 Δεκεμβρίου 2024

Εισαγωγή στο Vitis

Σε πρώτη φάση μελετήσαμε αναλυτικά το παράδειγμα της vadd, όπου πραγματοποιείται στον kernel η πρόσθεση των input διανυσμάτων. Συγκεκριμένα, από την πλευρά του host αρχικοποιούνται οι πίνακες που θα χρησιμοποιηθούν σαν input της συνάρτησης του kernel, και υπολογίζονται οι software τιμές της πρόσθεσης για τον τελικό έλεγχο της σωστής λειτουργίας του hardware. Έπειτα, αφού επιτευχθεί επικοινωνία με το fpga, δεσμεύει χώρο στην καθολική μνήμη για τους buffers μέσα από τους οποίους γίνεται η ανάγνωση και η εγγραφή από και προς τον kernel (μέσω του AXI4 interface). Στην συνέχεια αντιστοιχίζονται τα arguments της top συνάρτησης με τους κατάλληλους buffers και τα δεδομένα των δύο πρώτων πινάκων αντιγράφονται στην global memory. Τώρα ξεκινάει η λειτουργία του kernel, όπου αρχικά δίνεται ένα start signal μέσα από το AXI lite interface και γίνεται ανάγνωση των arguments από την global memory μέσα από το AXI4 interface. Η ανάγνωση αυτή γίνεται κατά bursts. Με την γραφή ή την ανάγνωση στην global μνήμη κατά bursts παίρνουμε με μία πρόσβαση πολλαπλά συνεχόμενα blocks δεδομένων, οπότε μειώνεται η καθυστέρηση αφού όσο γίνεται fetch ένα block, ένα άλλο υπόκειται επεξεργασίας, κρύβοντας έτσι το συνολικό latency. Τα δεδομένα που έχουν γίνει ήδη fetched προστίθενται καταλλήλως ώστε ο νέος πίνακας να είναι το άθροισμα των δύο πινάκων της εισόδου και πάλι κατά bursts γράφονται οι τιμές του στην global memory, όπου από εκεί τα λαμβάνει ο host. Τέλος, όταν σταλεί το σήμα από τον kernel ότι η διεργασία έχει τελειώσει, ο host συγκρίνει με τα δεδομένα που υπολόγισε με software τρόπο για να επιβεβαιώσει την σωστή λειτουργία του hardware.

Κώδικας του kernel

Μέσω του m_axi interface γίνεται η μεταφορά των δεδομένων από την global memory στο FPGA. Ορίστηκαν τα παρακάτω, σε αντιστοιχία με τον kernel του παραδείγματος:

```
#pragma HLS INTERFACE m_axi port = A offset = slave bundle = gmem
#pragma HLS INTERFACE m_axi port = B offset = slave bundle = gmem
#pragma HLS INTERFACE m_axi port = AB offset = slave bundle = gmem
#pragma HLS INTERFACE s_axilite port = A bundle = control
#pragma HLS INTERFACE s_axilite port = B bundle = control
#pragma HLS INTERFACE s_axilite port = AB bundle = control
#pragma HLS INTERFACE s_axilite port = size bundle = control
#pragma HLS INTERFACE s_axilite port = return bundle = control
```

Το m-axi (AXI4 interface) χρησιμοποιείται για την bulk μεταφορά των δεδομένων (ανάγνωση των δεδομένων εισόδων και εγγραφή πίσω στην καθολική μνήμη των αποτελεσμάτων), ενώ το s_axilite(AXI lite interface) για one-off σήματα ελέγχου και configuration του kernel (διευθύνσης, start/finish signals, size ως απλό int) καθώς έχει μικρότερο latency για μικρά σε όγκο δεδομένα.

Επίσης, έχουμε ορίσει τοπικά πίνακες προκειμένου να συγκρατούν τα δεδομένα των εισόδων, με στόχο να μειωθεί η καθυστέρηση που απαιτείται για την πρόσβαση στην global μνήμη. Ορίσαμε τους πίνακες αυτούς ως δισδιάστατους για μεγαλύτερη ευκολία στην ανάγνωση του κώδικα και τους κάνουμε partition, όπως και στο προηγούμενο εργαστήριο, προκειμένου να μπορέσουν οι υπολογισμοί να γίνουν παράλληλα χρησιμοποιώντας unroll. Στην συνέχεια, διαβάζουμε τους πίνακες A και B από την global μνήμη τους οποίους αποθηκεύουμε στους τοπικούς πίνακες. Χρησιμοποιούμε εδώ το pipeline pragma ούτως ώστε η διαδικασία της ανάγνωσης να είναι pipelined και να έχουμε μικρότερο latency. Επίσης κάνουμε clear τον τοπικό buffer(αρχικοποιούμε σε 0), καθώς χωρίς αυτό βγαίνουν σφάλματα στην εκτέλεση του hardware emulation. Στο unroll βάλαμε παράγοντα 16 ώστε να ισούται με την διάσταση των πινάκων και το τοποθετήσαμε στο $2^{ο}$ nested loop. Σκοπός είναι να γίνουν παράλληλα όλοι οι υπολογισμοί για την κάθε σειρά. Αφού γίνουν οι υπολογισμοί μεταφέρουμε τα δεδομένα από τον τοπικό πίνακα στην global μνήμη μέσω του port AB, πάλι αξιοποιώντας pipeline.

Κώδικας του host

Από την πλευρά του host ορίσαμε τα διανύσματα των γινομένων (δηλαδή τα source_sw_ results, source_hw_results) ως διανύσματα 64-bit integers. Ο σκοπός ήταν να μπορέσουμε να γράψουμε ολόκληρη την πληροφορία από τον πολλαπλασιασμό των πινάκων, όπου κάθε στοιχείο είναι 32-bit. Για να μην υπάρχει overflow, χρειαζόταν ένα άνω όριο για την τιμή κάθε στοιχείου των πινάκων. Λαμβάνοντας υπόψην μας ότι σε πολλαπλασιασμό 2 n-bit αριθμών χρειάζεται μεταβλητή 2n-bit για να είναι βέβαιο πως δεν θα χαθεί πληροφορία, όπως επίσης και για την πρόσθεση 2 n-bit αριθμών χρειάζεται n+1-bit μεταβλητή, και γνωρίζοντας ότι κάθε στοιχείο του νέου πίνακα περιέχει το άθροισμα 16 πολλαπλασιασμών, προκύπτει ότι το όριο είναι τα 2^{30}. Με την εισαγωγή του ορίου όμως (ψευδοτυχαίος αριθμός mod 2^{30}-1) παρατηρήθηκε αυξημένος χρόνος του device execution time, για αυτό και η ιδέα απορρίφθηκε. Αντ' αυτού, αφήσαμε την έξοδο ως έχει, υποθέτοντας ότι σε μια θεωρητική εφαρμογή, δεν μας ενδιέφεραν τιμές πάνω από το 2^{64}-1.

Αποτελέσματα

Τρέχοντας τον παραπάνω HLS κώδικα, ο βέλτιστος χρόνος εκτέλεσης (Kernel execution) έφτασε τα 0,009ms. 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	4	1024.000	25.000	0.004	0.002	0.002	9170.150

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