Set 2 - SIMD and CUDA

Φοιτητής 1

Ονοματεπώνυμο: Αγγουρά Ρουμπίνη Μαρία

ΑΜ: 1084634

Έτος Σπουδών: 5ο

Φοιτητής 2

Ονοματεπώνυμο: Παυλόπουλος Ιάσονας

ΑΜ: 1084565

Έτος Σπουδών: 5ο

Επεξήγηση Υλοποίησης

Ερώτημα 1a

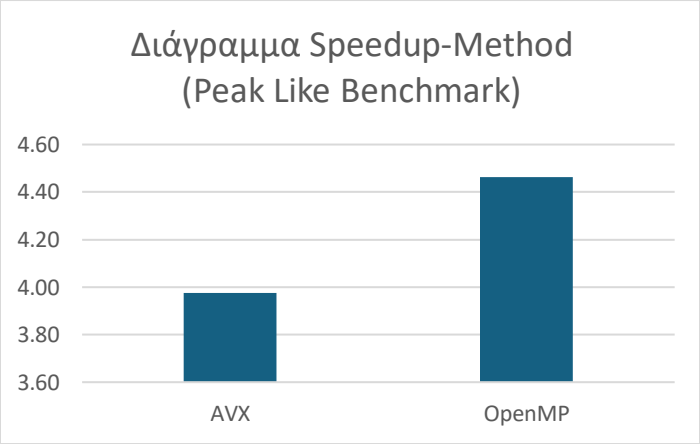
- Με την χρήση του flag -ftree-vectorize κατά το compilation, ενεργοποιείται το vectorization στο στάδιο της βελτιστοποίησης του δέντρου. Ο μεταγλωττιστής αναλύει βρόγχους ή άλλα μοτίβα κώδικα για να εντοπίσει ευκαιρίες για την ταυτόχρονη εκτέλεση πολλαπλών υπολογισμών. Αυτό εφαρμόζεται συνήθως σε βρόχους που εκτελούν αριθμητικές πράξεις.
- Για την παραλληλοποίηση του weno με την χρήση omp, χρησιμοποιούμε την εντολή #pragma omp parallel for, η οποία διαχωρίζει τις επαναλήψεις του βρόχου σε πολλαπλά νήματα, επιτρέποντας την ταυτόχρονη εκτέλεση υπολογισμών για διαφορετικούς δείκτες, καθώς επίσης και την παράμετρο simd η οποία εξασφαλίζει το vectorization, αξιοποιώντας τις εντολές υλικού SIMD για την επεξεργασία πολλαπλών σημείων δεδομένων σε μία λειτουργία εντός κάθε νήματος.
- Για την παραλληλοποίηση του weno με την χρήση SSE/AVX, χρησιμοποιούμε μεταβλητές/καταχωρητές που επεξεργάζονται τα δεδομένα σε κομμάτια των 8 floats.
  Η συνάρτηση _mm256_loadu_ps φορτώνει 8 αριθμούς κινητής υποδιαστολής από τη μνήμη σε καταχωρητές AVX.
  Οι καταχωρητές AVX εκτελούν αριθμητικές πράξεις και στις 8 κινητές μονάδες ταυτόχρονα με τη χρήση των _mm256_div_ps, _mm256_mul_ps και _mm256_add_ps.
  Τέλος, για όλα τα υπόλοιπα στοιχεία που δεν χωράνε στη διανυσματική επεξεργασία (αν το NENTRIES δεν είναι πολλαπλάσιο του 8), χρησιμοποιούμε τη μη βελτιστοποιημένη διαδικασία weno.h.

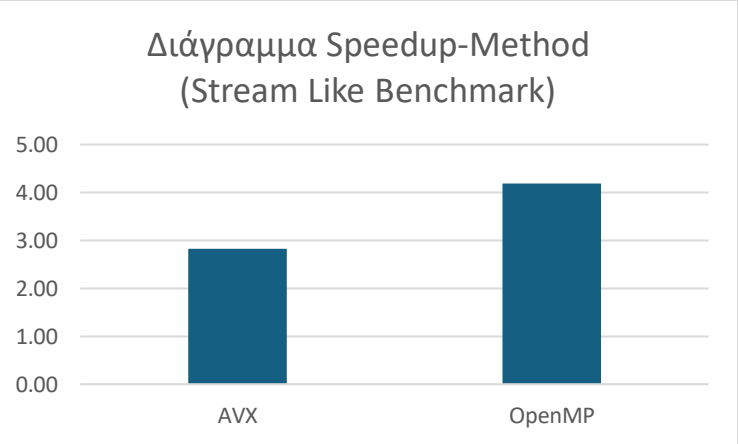
Ερώτημα 1b

Με την χρήση του timeval και της συνάρτησης check_error που προ-υπήρχε στο bench.c, μπορούμε να μετρήσουμε τον χρόνο εκτέλεσης καθώς και την ορθότητα των δεδομένων των τριών διαφορετικών υλοποιήσεων του weno.h.

Επίσης, πλέον (μέσω του Makefile), μπορούμε να πάρουμε δυναμικά το μέγεθος της L1 Data Cache (με την χρήση του lscpu) και να χρησιμοποιήσουμε το 50% για το Peak-Like Benchmark (ανεξαρτήτως επεξεργαστή), ενώ δίνεται η δυνατότητα να θέσουμε το μέγεθος της κύριας μνήμης που θέλουμε να διαθέσουμε για το Stream-Like benchmark (πάλι μέσω του Makefile).

Τέλος, τρέχοντας το bench.c, εξάγουμε τα παρακάτω αποτελέσματα σε μορφή διαγράμματος:





Αναλυτικότερα, βλέπουμε πως πετυχαίνουμε επιτάχυνση της διαδικασίας κατά περίπου 4 φορές με την χρήση AVX και κατά 4.5 φορές με την χρήση OpenMP, για το Peak Like Benchmark, αλλά περίπου κατά 3 φορές με την χρήση AVX και κατά 4 φορές με την χρήση OpenMP για το Stream Like Benchmark. Παρακάτω είναι αναλυτικά οι χρόνοι των 5 εκτελέσεων.

Stream Like					
Run Number	Serial	AVX	OpenMP	Serial-to-AVX	Serial-to-OMP
1	0.388	0.137	0.094		
2	0.389	0.138	0.089		
3	0.394	0.138	0.090		
4	0.391	0.139	0.087		
5	0.400	0.142	0.108		
AVG	0.392	0.139	0.094	2.83	4.19

Peak Like					
Run Number	Serial	AVX	OpenMP	Serial-to-AVX	Serial-to-OMP
1	0.000046	0.000016	0.001800		
2	0.000079	0.000016	0.000015		
3	0.000064	0.000016	0.000014		
4	0.000064	0.000016	0.000014		
5	0.000065	0.000016	0.000014		
AVG	0.000064	0.000016	0.000371	3.98	4.46
			0.000014		0.171243942

Παρατηρήσεις

1. Όπου κρίθηκε αναγκαίο, υπάρχουν τα απαραίτητα Comments στον κώδικα για την ευκολότερη ανάγνωση και κατανόησή του.
2. Τα παραπάνω αποτελέσματα των χρόνων εκτέλεσης και του Speedup, συλλέχθηκαν με τον υπολογισμό του μέσου όρου 5 εκτελέσεων της κάθε μεθόδου.
3. Κατά την εκτέλεση του 1^{ου} run του Peak Like Benchmark, παρατηρήσαμε μία αύξηση στον χρόνο εκτέλεσης του OpenMP weno.h, η οποία όμως δεν παρατηρείτε στα υπόλοιπα runs, επομένως τα διαγράμματα, δεν συμπεριλαμβάνουν αυτήν την μέτρηση για την εξαγωγή συμπερασμάτων.

Ερώτημα 2

Για την υλοποίηση του complex matrix multiplication, ακολουθήσαμε τα εξής βήματα:

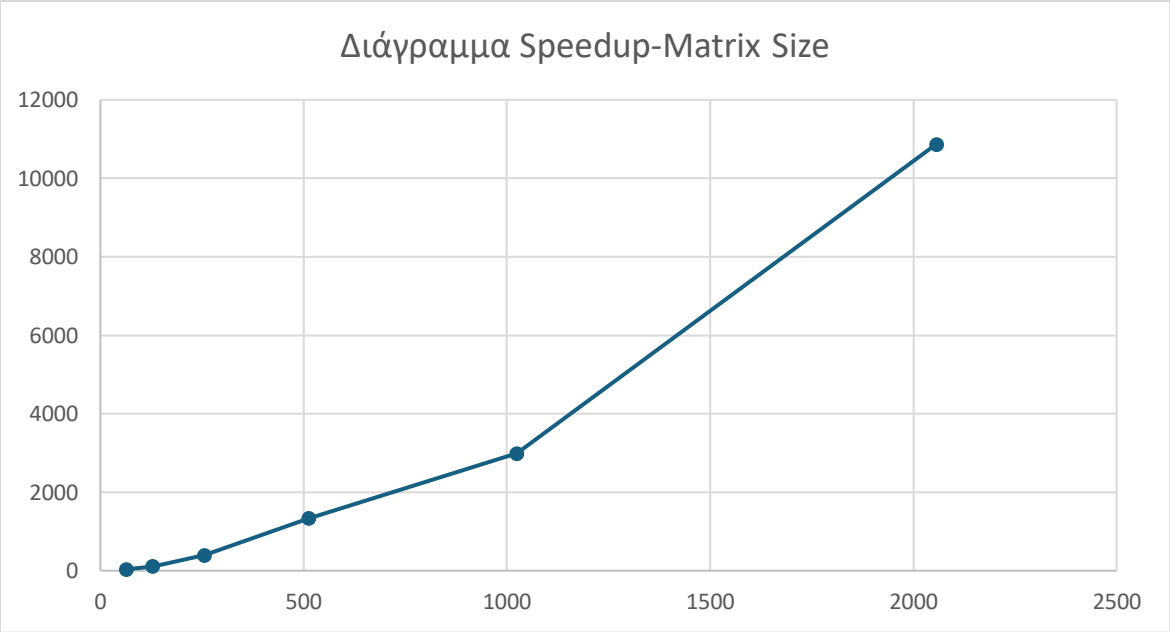
- Αρχικοποιούμε τις θέσεις μνήμης των μεταβλητών που θα χρειαστούμε στον host (CPU).
- Εισάγουμε τυχαίες τιμές (με την rand()) σε κάθε μία από τις μεταβλητές.
- Αρχικοποιούμε τις θέσεις μνήμης των μεταβλητών που θα χρειαστούμε και στο device (GPU).
- Θέτουμε την διάσταση του grid και των blocks (Συγκεκριμένα 16*16 = 256 threads/block).
- Αρχικοποιούμε τα Cuda events για την χρονομέτρηση του υπολογισμού και ξεκινάμε την χρονομέτρηση.
- Εκκινούμε τον complexMatrixMultiply kernel με τις παραμέτρους που θέσαμε παραπάνω.
    - Σε κάθε νήμα ανατίθεται μια μοναδική γραμμή και στήλη.
    - Πραγματοποιούνται οι πολλαπλασιασμοί όπως φαίνονται στην εκφώνηση με τα κατάλληλα offsets.
    - Αποθηκεύονται τα αποτελέσματα στις μεταβλητές e και f.
- Αντιγράφουμε τις μεταβλητές e και f από το Device (GPU) στο Host (CPU).
- Σταματάμε την χρονομέτρηση και υπολογίζουμε τον χρόνο.
- Απελευθερώνουμε τις θέσεις μνήμης στο device (GPU)

Επίσης, υλοποιήσαμε το complex matrix multiplication και στο host, με triple nested loop, το οποίο όπως θα δούμε παρακάτω προφανώς είναι πολύ αργό, αλλά χρησιμεύει για την σύγκριση και επαλήθευση των αποτελεσμάτων σε σχέση με την υλοποίηση στο device (και σε σχέση με τον χρόνο αλλά και με την ορθότητα των αποτελεσμάτων).

Για την χρονομέτρηση του complex matrix multiplication στο host χρησιμοποιήσαμε το timeval, όπως και στο προηγούμενο ερώτημα.

Για την επαλήθευση των αποτελεσμάτων και των 2 υλοποιήσεων, χρησιμοποιούμε την συνάρτηση verify_results η οποία παίρνει ως ορίσματα τους 4 πίνακες (2 για κάθε υλοποίηση), e και f, καθώς επίσης και ένα tolerance και συγκρίνει τα αποτελέσματα ένα προς ένα και αν υπάρχει κάποιο mismatch μας το εκτυπώνει.

Έχοντας λοιπόν τρέξει το complex_matrix_multiplication.cu, για διαφορετικά μεγέθη Matrix συλλέξαμε τα εξής δεδομένα σε μορφή διαγράμματος:



Παρατηρούμε, πως η επιτάχυνση του complex matrix multiplication, είναι εκθετική όσο αυξάνεται το matrix size, χρησιμοποιώντας την υλοποίηση στην GPU. Σίγουρα, η σύγκριση μεταξύ σειριακού σε CPU και παράλληλου κώδικα σε GPU δεν είναι δίκαιη, ωστόσο σε προηγούμενες ασκήσεις στις οποίες έγινε σύγκριση κώδικα σειριακού και παράλληλου σε CPU δεν παρατηρούμε επιτάχυνση τέτοιας τάξης.

Συγκεκριμένα, μπορείτε να δείτε τις μετρήσεις (οι χρόνοι σε ms) παρακάτω:

Matrix Size	CPU Time	GPU Time	Speedup
64	8.00	0.22	36
128	31.00	0.29	108
256	280.00	0.71	394
512	3447.00	2.59	1331
1024	32135.00	10.75	2989
2056	559305.00	51.47	10867

Παρατηρήσεις

1. Όπου κρίθηκε αναγκαίο, υπάρχουν τα απαραίτητα Comments στον κώδικα για την ευκολότερη ανάγνωση και κατανόησή του.
2. Τα παραπάνω αποτελέσματα των χρόνων εκτέλεσης και του Speedup, συλλέχθηκαν με τον υπολογισμό του μέσου όρου 5 εκτελέσεων της κάθε μεθόδου.
3. Μπορείτε να αλλάξετε το Matrix Size από το Makefile που δημιουργήσαμε για την εύκολη μεταγλώττιση και εκτέλεση του κώδικα.