Λογισμικό και Προγραμματισμός Συστημάτων Υψηλής Επίδοσης

2ο Σετ Ασκήσεων

Χειμερινό Εξάμηνο 2023-2024

Στοιχεία Ομάδας

Παπανικολάου Αικατερίνη	1064041	st1064041@ceid.upatras.gr
Πρέτσιος Κωνσταντίνος	1084666	st1084666@ceid.upatras.gr

Πληροφορίες

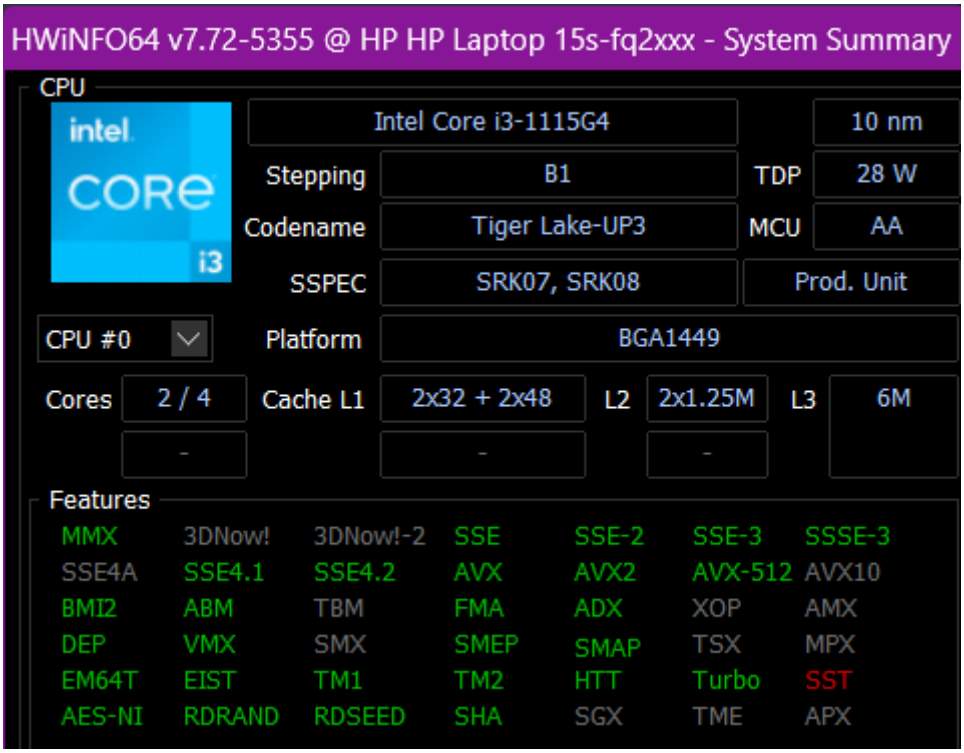
1. Τα πειράματά μας για το 1ο μέρος με SIMD Vectorization έγιναν σε υπολογιστή με τα παρακάτω χαρακτηριστικά:

OS:	Windows 11 Home
RAM:	8GB
Processor:	11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz 2.90 GHz

2. Τα πειράματά μας για το 2ο μέρος με CUDA έγιναν στο μηχάνημα krylov100 μέσω του λογαριασμού που μας παρείχατε.

Question 1: SIMD force calculation

a) Με τη βοήθεια του HWiNFO βρήκαμε ότι η CPU του μηχανήματός μας υποστηρίζει τα εξής όσον αφορά τα SSE/AVX:



Στον κώδικα του ερωτήματος γίνονται πράξεις σε δεδομένα τύπου float.

Αφού είναι διαθέσιμα τα **SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2** σημαίνει ότι υπάρχουν καταχωρητές των 128 bits. Σε αυτούς τους καταχωρητές μπορούν να χωρέσουν μέχρι 4 floats. Μπορούμε να περιμένουμε ο κώδικας με vectorization να τρέχει έως και 4 φορές πιο γρήγορα από ότι αν έτρεχε σειριακά.

Με τα **AVX, AVX2** υπάρχουν καταχωρητές των 256 bits. Σε αυτούς τους καταχωρητές μπορούν να χωρέσουν μέχρι 8 floats. Μπορούμε να περιμένουμε ο κώδικας με vectorization να τρέχει έως και 8 φορές πιο γρήγορα από ότι αν έτρεχε σειριακά.

Το **AVX-512** έφερε τους καταχωρητές των 512 bits. Σε αυτούς τους καταχωρητές μπορούν να χωρέσουν μέχρι 16 floats. Μπορούμε να περιμένουμε ο κώδικας με vectorization να τρέχει έως και 16 φορές πιο γρήγορα από ότι αν έτρεχε σειριακά.

b) Οι χρόνοι που πήραμε είναι:

```
katherine@LAPTOP-6P6F3R1F:/mnt/c/Users/Katherine/Desktop/CEID/7o Εξάμηνο/Λογισμικό και Προγραμματισμός Συστ/Ασκήσεις/hpc
_ex02_code/task1_simd$ ./force1d
Force acting at x_0=0.000000 : 324796.718750
Force acting at x_0=-0.100000 : 10106.603516
Force acting at x_0=-0.200000 : 895.442017
elapsed time: 838234.901428 mus
katherine@LAPTOP-6P6F3R1F:/mnt/c/Users/Katherine/Desktop/CEID/7o Εξάμηνο/Λογισμικό και Προγραμματισμός Συστ/Ασκήσεις/hpc
_ex02_code/task1_simd$ ./force1d_sse
Force acting at x_0=0.000000 : 324796.718750
Force acting at x_0=-0.100000 : 10106.603516
Force acting at x_0=-0.200000 : 895.442017
elapsed time: 231319.189072 mus
katherine@LAPTOP-6P6F3R1F:/mnt/c/Users/Katherine/Desktop/CEID/7o Εξάμηνο/Λογισμικό και Προγραμματισμός Συστ/Ασκήσεις/hpc
_ex02_code/task1_simd$ ./force1d_avx
Force acting at x_0=0.000000 : 324796.718750
Force acting at x_0=-0.100000 : 10106.603516
Force acting at x_0=-0.200000 : 895.442017
elapsed time: 225805.044174 mus
katherine@LAPTOP-6P6F3R1F:/mnt/c/Users/Katherine/Desktop/CEID/7o Εξάμηνο/Λογισμικό και Προγραμματισμός Συστ/Ασκήσεις/hpc
_ex02_code/task1_simd$
```

Βλέπουμε ότι οι vectorized κώδικες τελικά έτρεξαν 4 φορές πιο γρήγορα σε σύγκριση με τον σειριακό.

c) Ο κώδικας που φτιάξαμε εμείς βρίσκεται στο αρχείο force1d_vectorized.c

Δοκιμάσαμε να παραλληλοποιήσουμε το loop της συνάρτησης compute_force με AVX intrinsics. Αρχικά χρησιμοποιήσαμε τις _mm256_set1_ps για να ορίσουμε τις σταθερές ενώ με τη _mm256_loadu_ps φορτώσαμε τα δεδομένα από τον positions (τα οποία φροντίσαμε να είναι aligned για να φορτώνονται πιο γρήγορα). Έπειτα μέσα στο loop προσαρμόσαμε τις πράξεις και μετά από αυτό ορίσαμε έναν float πίνακα στον οποίο φορτώσαμε τα αποτελέσματα με την _mm256_storeu_ps. Η συνολική δύναμη γίνεται return μέσω της μεταβλητής force στην οποία απλά κάναμε ένα sum όλα τα στοιχεία του results.

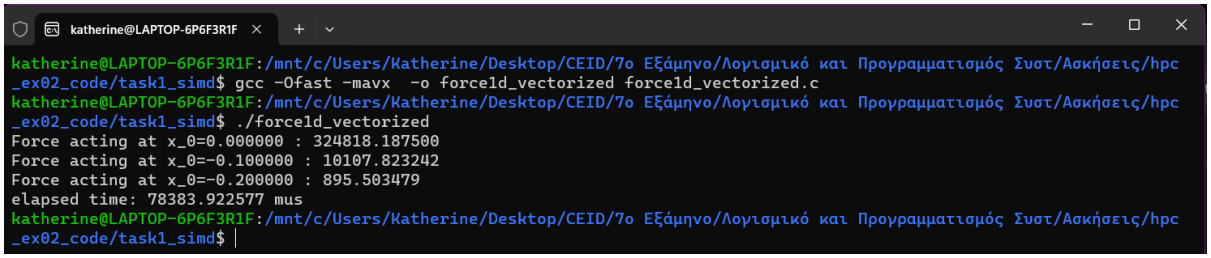
Ο χρόνος που πήραμε από το manual vectorization είναι:

```
katherine@LAPTOP-6P6F3R1F
katherine@LAPTOP-6P6F3R1F:/mnt/c/Users/Katherine/Desktop/CEID/7o Εξάμηνο/Λογισμικό και Προγραμματισμός Συστ/Ασκήσεις/hpc
_ex02_code/task1_simd$ gcc -o force1d_vectorized force1d_vectorized.c -mavx
katherine@LAPTOP-6P6F3R1F:/mnt/c/Users/Katherine/Desktop/CEID/7o Εξάμηνο/Λογισμικό και Προγραμματισμός Συστ/Ασκήσεις/hpc
_ex02_code/task1_simd$ ./force1d_vectorized
Force acting at x_0=0.000000 : 324813.218750
Force acting at x_0=-0.100000 : 10107.664062
Force acting at x_0=-0.200000 : 895.503418
elapsed time: 464849.948883 mus
katherine@LAPTOP-6P6F3R1F:/mnt/c/Users/Katherine/Desktop/CEID/7o Εξάμηνο/Λογισμικό και Προγραμματισμός Συστ/Ασκήσεις/hpc
_ex02_code/task1_simd$
```

Καταφέραμε δηλαδή να πάρουμε ο χρόνο εκτέλεσης περίπου 2 φορές γρηγορότερο από το σειριακό.

Ωστόσο με το flag -Ofast ο κώδικάς μας έτρεξε γρηγορότερα από τους βελτιωμένους κώδικες του compiler στο προηγούμενο ερώτημα. Το -Ofast προσπαθεί να βελτιώσει το χρόνο κάνοντας επιπλέον βελτιστοποιήσεις όπως dead code elimination, loop unrolling κλπ.

Παρακάτω παραθέτουμε το χρόνο του κώδικά μας κατά το compile με χρήση του -Ofast:


katherine@LAPTOP-6P6F3R1F
katherine@LAPTOP-6P6F3R1F:/mnt/c/Users/Katherine/Desktop/CEID/7ο Εξάμηνο/Λογισμικό και Προγραμματισμός Συστ/Ασκήσεις/hpc_ex02_code/task1_simd$ gcc -Ofast -mavx -o force1d_vectorized force1d_vectorized.c
katherine@LAPTOP-6P6F3R1F:/mnt/c/Users/Katherine/Desktop/CEID/7ο Εξάμηνο/Λογισμικό και Προγραμματισμός Συστ/Ασκήσεις/hpc_ex02_code/task1_simd$ ./force1d_vectorized
Force acting at x_0=0.000000 : 324818.187500
Force acting at x_0=-0.100000 : 10107.823242
Force acting at x_0=-0.200000 : 895.503479
elapsed time: 78383.922577 mus
katherine@LAPTOP-6P6F3R1F:/mnt/c/Users/Katherine/Desktop/CEID/7ο Εξάμηνο/Λογισμικό και Προγραμματισμός Συστ/Ασκήσεις/hpc_ex02_code/task1_simd$

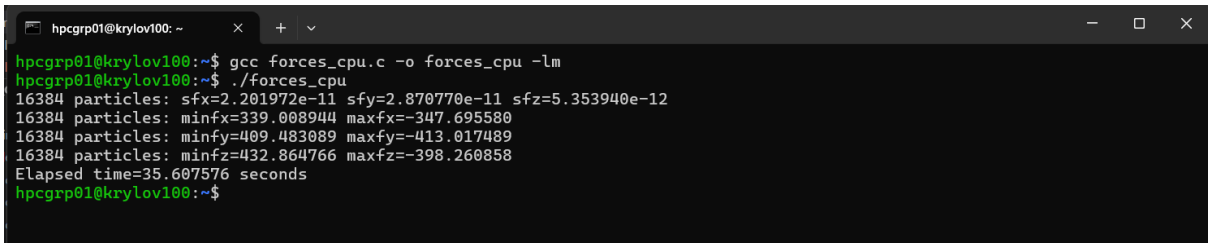
Question 2: CUDA

Ο κώδικάς μας που αξιοποιεί τη συμμετρία των δυνάμεων βρίσκεται στο αρχείο forces_gpu.c και ο παράλληλος κώδικας βρίσκεται στο αρχείο forces_gpu.cu .

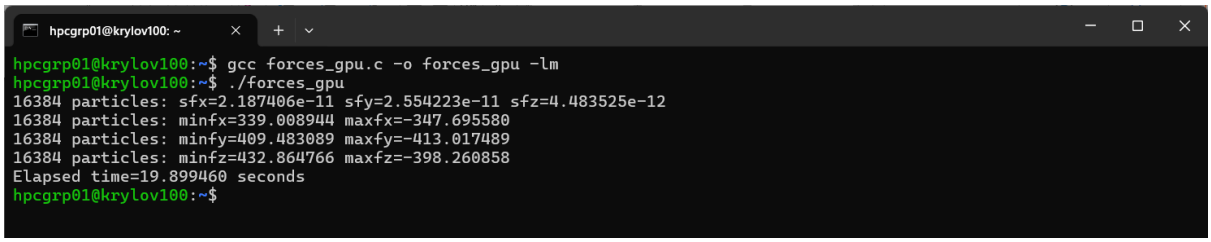
Για τη βελτιστοποίηση του κώδικα αξιοποιήσαμε τις συμμετρίες των δυνάμεων, μειώνοντας με αυτόν τον τρόπο σε μεγάλο βαθμό τις επαναλήψεις στη συνάρτηση computeGravitationalForces(). Αυτό επιτυγχάνεται αρχικά κάνοντας τις αρχικοποιήσεις των fx, fy και fz όλων των στοιχείων του πίνακα particles πριν γίνουν οι υπολογισμοί των δυνάμεων. Στη συνέχεια αλλάζουμε το εσωτερικό loop υπολογίζοντας σε κάθε επανάληψη, για κάθε ζεύγος σωματιδίων i και j, όχι μόνο τη δύναμη που ασκείται από το j στο i αλλά και τη δύναμη που ασκείται από το i στο j μειώνοντας έτσι και τους περιττούς επαναϋπολογισμούς των tmp και magnitude.

Όσον αφορά την παραλληλοποίηση με CUDA αρχικά ορίζουμε τη συνάρτηση computeGravitationalForces() ως kernel. Στη συνέχεια ορίζονται οι επαναλήψεις που θα αναλάβει κάθε thread με τη χρήση των μεταβλητών index και stride, οι οποίες αποτελούν τη θέση του πίνακα particles από την οποία θα ξεκινήσει κάθε thread και το βήμα με το οποίο θα εκτελέσει το loop το thread αντίστοιχα. Επίσης όταν θέλουμε να τροποποιήσουμε την τιμή fx, fy ή fz ενός σωματιδίου χρησιμοποιούμε τη συνάρτηση atomicAdd() ώστε να διασφαλίσουμε ότι δεν θα έχουμε race conditions. Στη συνάρτηση main() δεσμεύουμε χώρο στη unified μνήμη για τον πίνακα particles ώστε να είναι προσβάσιμος και από την GPU και καλούμε τη συνάρτηση computeGravitationalForces() με μέγεθος block 256 threads και αριθμό από blocks που υπολογίζεται ανάλογα με το μέγεθος n του πίνακα particles.

Ό χρόνος που έτρεχε ο σειριακός naive κώδικας είναι:



Ό χρόνος που έτρεχε ο σειριακός κώδικας που αξιοποιεί τη συμμετρία των δυνάμεων είναι:



Ό χρόνος που έτρεχε παράλληλος κώδικας με CUDA στο Krylov100 είναι: