5η εργασία

Σταμούλος Αλέξανδρος 02954

Γαλάνης Αχιλλέας 02941

|  |
| --- |
| ./deviceQuery Starting...  CUDA Device Query (Runtime API) version (CUDART static linking)  Detected 2 CUDA Capable device(s)  Device 0: "Tesla K80"  CUDA Driver Version / Runtime Version 11.4 / 11.5  CUDA Capability Major/Minor version number: 3.7  Total amount of global memory: 11441 MBytes (11997020160 bytes)  (13) Multiprocessors, (192) CUDA Cores/MP: 2496 CUDA Cores  GPU Max Clock rate: 824 MHz (0.82 GHz)  Memory Clock rate: 2505 Mhz  Memory Bus Width: 384-bit  L2 Cache Size: 1572864 bytes  Maximum Texture Dimension Size (x,y,z) 1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)  Maximum Layered 1D Texture Size, (num) layers 1D=(16384), 2048 layers  Maximum Layered 2D Texture Size, (num) layers 2D=(16384, 16384), 2048 layers  Total amount of constant memory: 65536 bytes  Total amount of shared memory per block: 49152 bytes  Total number of registers available per block: 65536  Warp size: 32  Maximum number of threads per multiprocessor: 2048  Maximum number of threads per block: 1024  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)  Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)  Maximum memory pitch: 2147483647 bytes  Texture alignment: 512 bytes  Concurrent copy and kernel execution: Yes with 2 copy engine(s)  Run time limit on kernels: No  Integrated GPU sharing Host Memory: No  Support host page-locked memory mapping: Yes  Alignment requirement for Surfaces: Yes  Device has ECC support: Enabled  Device supports Unified Addressing (UVA): Yes  Device supports Compute Preemption: No  Supports Cooperative Kernel Launch: No  Supports MultiDevice Co-op Kernel Launch: No  Device PCI Domain ID / Bus ID / location ID: 0 / 6 / 0  Compute Mode:  < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >  Device 1: "Tesla K80"  CUDA Driver Version / Runtime Version 11.4 / 11.5  CUDA Capability Major/Minor version number: 3.7  Total amount of global memory: 11441 MBytes (11997020160 bytes)  (13) Multiprocessors, (192) CUDA Cores/MP: 2496 CUDA Cores  GPU Max Clock rate: 824 MHz (0.82 GHz)  Memory Clock rate: 2505 Mhz  Memory Bus Width: 384-bit  L2 Cache Size: 1572864 bytes  Maximum Texture Dimension Size (x,y,z) 1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)  Maximum Layered 1D Texture Size, (num) layers 1D=(16384), 2048 layers  Maximum Layered 2D Texture Size, (num) layers 2D=(16384, 16384), 2048 layers  Total amount of constant memory: 65536 bytes  Total amount of shared memory per block: 49152 bytes  Total number of registers available per block: 65536  Warp size: 32  Maximum number of threads per multiprocessor: 2048  Maximum number of threads per block: 1024  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)  Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)  Maximum memory pitch: 2147483647 bytes  Texture alignment: 512 bytes  Concurrent copy and kernel execution: Yes with 2 copy engine(s)  Run time limit on kernels: No  Integrated GPU sharing Host Memory: No  Support host page-locked memory mapping: Yes  Alignment requirement for Surfaces: Yes  Device has ECC support: Enabled  Device supports Unified Addressing (UVA): Yes  Device supports Compute Preemption: No  Supports Cooperative Kernel Launch: No  Supports MultiDevice Co-op Kernel Launch: No  Device PCI Domain ID / Bus ID / location ID: 0 / 7 / 0  Compute Mode:  < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >  > Peer access from Tesla K80 (GPU0) -> Tesla K80 (GPU1) : Yes  > Peer access from Tesla K80 (GPU1) -> Tesla K80 (GPU0) : Yes  deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 11.4, CUDA Runtime Version = 11.5, NumDevs = 2, Device0 = Tesla K80, Device1 = Tesla K80  Result = PASS |

Compile στην cpu: gcc -std=c99 -O3 -fopenmp -D\_DEFAULT\_SOURCE -o nbody nbody.c -lm

Compile στην gpu: nvcc -arch=sm\_37 -I. -ftz=true -Xcompiler -fopenmp -o nbody\_2 nbody\_2.cu

Openmp Version:

Ο κώδικας παραμένει ίδιος με την αρχική έκδοση με τη διαφορά ότι βάλαμε ένα openmp clause στο for:

|  |
| --- |
| #pragma omp parallel for  for (int i = 0; i < n; i++) {  float Fx = 0.0f; float Fy = 0.0f; float Fz = 0.0f;  for (int j = 0; j < n; j++) {  float dx = p[j].x - p[i].x;  float dy = p[j].y - p[i].y;  float dz = p[j].z - p[i].z;  float distSqr = dx\*dx + dy\*dy + dz\*dz + SOFTENING;  float invDist = 1.0f / sqrtf(distSqr);  float invDist3 = invDist \* invDist \* invDist;  Fx += dx \* invDist3; Fy += dy \* invDist3; Fz += dz \* invDist3;  } |

GPU version:

Κάνουμε τα απαραίτητα allocations για την gpu memory, μεταφέρουμε τα δεδομένα στην gpu, καλούμε τον kernel με block size ίσο με 256 (μετά από δοκιμές καταλήξαμε ότι αυτό είναι το optimal block size) και τέλος μεταφέρουμε τα δεδομένα πίσω στην cpu.Έξω από τα σημεία που μετράτε ο χρόνος τρέχουμε την υλοποίηση της cpu για να κάνουμε σύγκριση της ακρίβειας στο πρώτο iteration.

**1η βελτιστοποίηση:**

Μετατρέπουμε το float3 σε float4 στο struct body ώστε να έχουμε coalesced memory accesses.Οπότε κάνουμε και τις απαραίτητες αλλαγές στον κώδικά (π.χ. αντί για p[i].vx, p[i].vy, p[i].vz βάζουμε v[i].x, v[i].x, v[i].x) και βλέπουμε σημαντική μείωση του χρόνου.

**2η βελτιστοποίηση:**

Χρησιμοποιούμε την τεχνική του tiling για να έχουμε data reuse. Φορτώνουμε p bodies από το device στο shared memory. Το κάθε thread block υπολογίζει p διαδοχικές αλληλεπιδράσεις σωμάτων. Τα p threads εκτελούν τη συνάρτηση παράλληλα και το κάθε thread υπολογίζει την επιτάχυνση ως αποτέλεσμα της αλληλεπίδρασης του με p άλλα bodies.

Χρησιμοποιήσαμε το flush denormals to zero όπως συζητήθηκε στην τάξη και παρατηρήσαμε μια μικρή βελτίωση. Δοκιμάσαμε να κάνουμε loop unrolling άλλα δεν είχε αισθητή διαφορά.

|  |  |
| --- | --- |
| Serial | 84.636 s |
| OpenMP | 2.883 s |
| CUDA | 0.206 s |

ΟpenMP: **29.35** φορές γρηγορότερος από τον σειριακό.

CUDA: **325.52** φορές γρηγορότερος από τον σειριακό και **14** φορές γρηγορότερος από τον OpenMP.