P. Hadjidoukas, E. Dermatas, E. Gallopoulos

Set 2 - SIMD and CUDA

Issued: December 8, 2023

Question 1: SIMD force calculation

```
/// computes the Lennard-Jones force for a particle at position x0
float compute_force(float *positions, float x0)
{
  // rm, eps: global constant variables
  float rm2 = rm * rm;
  float force = 0.0;
  for (size_t i=0; i<N; ++i) {
    float r = x0 - positions[i];
    float r2 = r * r; // r^2
    float s2 = rm2 / r2; // (rm/r)^2
    float s6 = s2*s2*s2; // (rm/r)^6
    force += 12 * eps * (s6*s6 - s6) / r;
  }
  return force;
}
```

The Lennard-Jones potential describes the potential energy of interaction between two non-bonding atoms or molecules based on their distance of separation. The code above, available in `task1_simd/force1d.c`, is a simple force calculation for a one-dimensional n-body problem. In this exercise, you should let the compiler optimize the compute_force loop, and try to achieve at least the same speedup with manual SIMD.

a) Find out which SSE/AVX version your machine and compiler support. Which speedup do you expect from it?

b) Compile the example code force1d.c with and without automatic vectorization and study the compiler's vectorization report to check that the loop is indeed vectorized in the former case. Comment on the relative performance of the two versions.

c) Now implement the loop with manual SSE or AVX intrinsics. Are you better than automatic vectorization? Comment on your results.

Optionally, you can apply any further optimizations to the code.

Question 2: CUDA

```c
/**
 * Compute the gravitational forces in the system of particles
 * Symmetry of the forces is NOT exploited
**/
void computeGravitationalForces(Particle_t *particles, int n)
{
  const double G = 6.67408e-11;

  for (int i=0; i<n; i++) {
    particles[i].fx = 0;
    particles[i].fy = 0;
    particles[i].fz = 0;

    for (int j=0; j<n; j++)
      if (i!=j) {
        double tmp = pow(particles[i].x-particles[j].x, 2.0) +
                     pow(particles[i].y-particles[j].y, 2.0) +
                     pow(particles[i].z-particles[j].z, 2.0);

        double magnitude = G * particles[i].m * particles[j].m / pow(tmp, 1.5);
        particles[i].fx += (particles[i].x-particles[j].x) * magnitude;
        particles[i].fy += (particles[i].y-particles[j].y) * magnitude;
        particles[i].fz += (particles[i].z-particles[j].z) * magnitude;
    }
  }
}
```

In this question, you will have to implement with CUDA a code that computes *anti*gravitational forces in the system of N planets in the 3D space. The force experienced by the i-th planet due to the j-th planet is given by the following expression:

$$\mathbf{F}_{ij} = G\frac{m_i m_j(\mathbf{r}_i - \mathbf{r}_j)}{\|\mathbf{r}_i - \mathbf{r}_j\|^3} \tag{1}$$

where $\mathbf{r} = (x, y, z)$ is the 3D coordinate vector, m is mass, $G = 6.67408 \times 10^{-11}$ is the gravitational constant and $\|\cdot\|$ denotes the length of a vector. Note the sign of the expression.

Therefore the total force on the i-th planet is

$$\mathbf{F}_i = \sum_{\substack{j=1\\j\neq i}}^{N} \mathbf{F}_{ij} = \sum_{\substack{j=1\\j\neq i}}^{N} G\frac{m_i m_j(\mathbf{r}_i - \mathbf{r}_j)}{\|\mathbf{r}_i - \mathbf{r}_j\|^3} \tag{2}$$

In `task2/forces_cpu.c` you will find the code that includes the above function and implements Eq. 2 for all the particles in a naive sequential way. The function accepts as an argument the array of N particles. Each particle, of type Particle_t, includes its coordinates, mass, and forces.

Your task is to implement a CUDA version of the function that computes the same forces. Modify the `Makefile` so that it compiles your code.