Code Optimization: Assignment N°6 Report

Duy Le - Cédric Léonard, 13/12/2021

Introduction

This third assignment concerns code vectorization. The experiment is about vectorizing the code implementing a N-body simulation with a classic $O(N^2)$ method. For this purpose, we will use the *Vector Class Library (VCL)*.

Realization

The first step was to change the function responsible for the distance computation between 2 particles: `dist`.

```
// basic implementation
double dist(double px, double py, double qx, double qy)
{ return sqrt (pow(px-qx, 2) + pow(py-qy, 2)); }

// Implementation using VCL
Vec4d distVCL(Vec4d px, Vec4d py, Vec4d qx, Vec4d qy)
{ return sqrt( (px-qx)*(px-qx) + (py-qy)*(py-qy) ); }
```

The code above shows the 2 different versions of this function. For the second one, using VCL, we have to work with vectors, symbolized with the type `Vec4d` here. This is with those types than the magic happens, indeed, depending on those types the compiler will use assembler instructions specific to the hardware architecture (when possible) and that will greatly optimize the code. Every architecture cannot use the same instructions, when talking about Single Instruction stream, Multiple Data stream (SIMD) the best instruction type we can use is determined by vector registers length. There are a lot of different SIMD Vector extensions (MMX, SSE2, AVX2, ...) but we considered only the basic: SSE (128-bits length), AVX (256-bits length) and AVX-512 (512-bits length).

The best one is AVX-512 as it allows to perform computation in parallel on 16 floats or 8 doubles. But as I wanted to test this program on my computer which only possesses AVX extension our code will use the `Vec4d` type which means our instructions will use vectors containing 4 doubles, or 256-bits.

The next step was to optimize the main function. The code was upgraded as shown below.

```
// Computes forces between bodies using VCL library
void ComputeForceVCL(int N, double *X, double *Y, double *mass, double *Fx,
double *Fy)
{
    // Minimal distance of two bodies of being in interaction
    const double minDist  = 0.0001;
    // Gravitational constant (should be e-10 but modified)
    const double G  = 6.67259e-7;
    // Let's use 256-bit vector with 4 doubles (AVX minimum instruction set)
```

```
 9      const int vecLen = 4; // 256-bit AVX, double data type
10      // Declare vectors
11      Vec4d Xvec, Yvec, Mvec, Fxvec, Fyvec;
12  
13      // For every bodies
14      for (int i = 0; i < N; i++)
15      {
16          // Take the x, y, and mass values from one particle Pi
17          Vec4d XvecI = X[i];      // Duplicate Xi to vector
18          Vec4d YvecI = Y[i];      // Duplicate Yi to vector
19          Vec4d MvecI = mass[i];   // Duplicate MASSi to vector
20  
21          // Set force to zero
22          Fx[i] = Fy[i] = 0.0;
23  
24          // For every possible vectors (nbbodies / veclen) [0 4 8 ... N]
25          for (int j = 0; j < N; j += vecLen)
26          {
27              // Load the vectors with other particles information
28              Xvec.load(&X[j]);
29              Yvec.load(&Y[j]);
30              Mvec.load(&mass[j]);
31              // Init force vectors
32              Fxvec = 0.0; Fyvec = 0,0;
33  
34              // Compute distance between points i and j, store result in a vector
35              Vec4d r_temp = distVCL(otherX, otherY, thisX, thisY);
36              // Check if r is 0.0 (particle pair), or too small. If that's the case replace it by 1.0
37              Vec4d r = select(r_temp > minDist, r_temp, 1.0);
38  
39              // Calculate forces from the other particles (if r too small forces are 0)
40              vecFx = select(r_temp > minDist, G * MvecI * Mvec * (Xvec-XvecI) / pow(r, 3), 0.0);
41              vecFy = select(r_temp > minDist, G * MvecI * Mvec * (Yvec-YvecI) / pow(r, 3), 0.0);
42  
43              // Add these forces together and store
44              Fx[i] += horizontal_add(Fxvec);
45              Fy[i] += horizontal_add(Fyvec);
46          }
47      }
48  }
```

As `distVCL`, the new function `ComputeForceVCL` uses the `Vec4d` type. As we use the double type that means each vector will contain 4 particles (`vecLen = 4`). After initializing every constant and vectors we enter in the outer loop, the first for loop that will browse every particle. In this loop we want to duplicate the information (X and Y position and its mass) of this particle in vectors. Thus, we will be able to perform the following computations in parallel. This duplication is realized lines 16 to 19, we also don't forget to initialize the forces to 0 as we will add each step the sub-forces of every other particle.

Then, we enter the inner loop, browsing the other particles, but with our vector length. This also means that $j \in \{0; 4; 8; ...; N\}$. In this inner loop, we will first load our vector with other particles information (done line 27 to 32). Then we compute the distance between each pair by calling the `distVCL` function. In the previous implementation, an `IF` statement prevented from computing the distance for the particle pair (a particle with itself) first because it doesn't make much sense, but also because it will result by a division by 0 in the force computation.

As it is complicated and too computationally complex to look for pair in our code we will simply check for a too small distance and when this is the case replace it by 1.0. This is realized line 37. Then, once again we check for that condition and finish computing both forces with the following formula:

$$F_{A \to B} = F_{B \to A} = G \times \frac{mass_A \times mass_B}{dist^2}$$

The formula above updated in the 2-D space this leads to lines 39 to 41. Finally, as the global force applied to one particle is the sum of every sub-forces from every other particle, we need to sum every forces we calculated and store them in `Fx` and `Fy` (line 43 to 45).

This concludes the code implementation. Unfortunately, we had too much struggle to set up VCL on Dione or my computer to test it. However, we compared this code algorithm with the previous one and have verified that it is correct.