B160509

Performance Programming Coursework: Serial Optimization of a Molecual Dynamics Program

Abstract

Keywords: Scientific programming, serial optimization, molecular dynamics, Fortran

1. Introduction

This paper documents the serial performance optimization conducted for a molecular dynamics program written in the Fortran programming language. The program reads data from n molecules as its input and iterates a predefined amount of steps. Each step the data of the molecules (e.g. force, position in a three dimensional space, velocity) is updated. The program counts collisions between molecules and writes the data for each molecule of the simulation out at certain, also predefined, intervals. The original version of the program is not optimized for computational efficiency.

This paper describes, documents and discusses the process of optimizing the original version of the program serially. The program was optimized for the Cirrus supercomputer, a tier-2 UK national supercomputer of the engineering and physical sciences research council, which is hosted and maintained by the EPCC (EPCC, 2020a). The compilers used were Intel's Fortran compiler `ifort`, version 18.0.5 and 19.0.0 for the linux operating system (Intel, 2018, 2019). The conducted optimizations range from choosing the appropriate compiler flags over rewriting performance critical sections of the program to hardware specific optimizations, like leveraging vectorization and cache optimizations. Since raw performance benefits are not all that is important for writing well performing and good programs, the maintainability and portability of the program are also looked at and discussed.

This paper continues in Section 2 with describing the molecular dynamics program in detail. Section 3 describes Cirrus and how the correctness of the program is tested with a regression test suite. Also the benchmark suite used for assessing the performance benefits of an optimization is outlined. Section 4 lists, describes and discusses all optimizations tested with their performance benefits. Afterwards, the results are discussed in Section 5. At least a conclusion is given in Section 6.

2. Molecular Dynamics Program

This section describes what the molecular dynamics program, which is optimized, does. The program is a simple molecular dynamics program. It reads data from n particles from a file. In the following, subscripts $1 \leq i \leq n$ refer to a particle of the simulation. The data for a particle i is its mass m_i, the viscosity of a fluid i is part of vis_i, the position of the particle's center in a three dimensional space $\vec{p_i}$, which is relative to a large central mass and its velocity $\vec{v_i}$. The program iterates a predefined amount of iterations. Each iteration represents a time step in the particle simulation. The time is updated by a constant value Δ_t. Every step the position and the velocity of each particle is updated, based on the gravitational forces operating on each particle. The gravitational forces are the forces between each particle and the gravitational force coming from the central mass. The particles are part of a fluid, which means the viscosity of the fluid must also be taken into account. The last external force operating on a particle is wind $\vec{w}$, which is a constant vector over the whole simulation.

The gravitational forces are computed based on Newton's law of universal gravitation. It states, that any two physical objects attract each other with a force, which is proportional to the mass of both objects and inversely proportional to the squared distance between both (Feynman, 1963). The scalar gravitational force F between two objects 1 and 2 can be mathematically described as:

$$F_{1,2} = G\frac{m_1 m_2}{||\vec{p}_1 - \vec{p}_2||_2^2}, \tag{1}$$

where G is the gravitational constant, m_i the mass of object i and $||\vec{p}_1 - \vec{p}_2||_2$ the Euclidean or L_2 distance between the centers of both objects. The vector form of the gravitational force object 2 operates on object 1, which also accounts for the direction of the force, is given by:

$$\vec{F}_{1\leftarrow 2} = -F_{1,2}\frac{\vec{p}_1 - \vec{p}_2}{||\vec{p}_1 - \vec{p}_2||_2} = -G\frac{m_1 m_2(\vec{p}_1 - \vec{p}_2)}{||\vec{p}_1 - \vec{p}_2||_2^3}. \tag{2}$$

The gravitational force, which object 1 operates on object 2 is the additive inverse of $\vec{F}_{1\leftarrow 2}$: $\vec{F}_{2\leftarrow 1} = -\vec{F}_{1\leftarrow 2}$. If particles collide, the gravitational forces between the two object are negated. The program finds collisions, by checking if the distance of two particles is smaller than a threshold $\tau_{1,2}$, which is the sum of the radii of the two particles. For the program, all particles have a radius of $\frac{1}{2}$, so the threshold for a collision is 1. Now we can define a pairwise gravitational forces function for the particles of the simulation as:

$$f_{1\leftarrow 2} := \begin{cases} \vec{F}_{1\leftarrow 2} & \text{if } ||\vec{p}_1 - \vec{p}_2||_2 \geq \tau_{1,2} \\ -\vec{F}_{1\leftarrow 2} & \text{otherwise} \end{cases}. \tag{3}$$

Other than the pairwise gravitational forces between the particles, there is the gravitational force of the central mass $\vec{F}_{1\leftarrow central}$. The central mass lies at the origin of the three dimensional space, which means its position vector $\vec{p}_{central} = 0$.

The viscosity of the fluid is another force operating on a particle. It is simply the negative of the viscosity of the fluid multiplied by the velocity vector of particle $i : -vis_i\vec{v}_i$. Shear velocity of the fluid is not taken into account. Lastly, there is the wind force, which is simply the negated viscosity of the fluid particle i is part of multiplied by the wind vector: $-vis_i\vec{w}$. The overall force per iteration operating on particle i can now be described as:

$$\vec{F}_i = -vis_i(\vec{v}_i + \vec{w}) + \vec{F}_{i \leftarrow central} + \sum_{j \neq i}^{n} f_{i \leftarrow j}. \tag{4}$$

Based on $\vec{F}_i$ we can now update $\vec{p}_i$ and $\vec{v}_i$:

$$\vec{p}_i = \vec{p}_i + \Delta_t \vec{v}_i \tag{5}$$

$$\vec{v}_i = \vec{v}_i + \Delta_t \frac{\vec{F}_i}{m_i}. \tag{6}$$

The iterations of the program are broken down into supersteps. On completion of a superstep, the updated particles are exported to a file with the same format as the input file.

3. Setup

This section outlines the settings, under which the program was optimized for performance. Information about the used hardware is given. The way correctness of the program was tested is described. Lastly the settings and the criterion for the benchmark for the assessment of the optimizations are shown.

Like stated in Section 1, the program was optimized for the Cirrus supercomputer (EPCC, 2020a). Since we are running the program serially, we are not concerned with the amount of nodes or the interconnect, but will focus on a single compute node. A single compute node of Cirrus contains two 2.1 GHz, 18-core Intel Xeon E5-2695 processors (code name: Broadwell). The processor supports the AVX2 vector instruction set (Buxton, 2011). Each processor is connected to 128 Gigabyte of memory. Both processors are within a NUMA region, so 256 Gigabyte of memory are actually at ones disposal (EPCC, 2020b). The compute node offers three levels of cache:

1. 32 Kilobyte instruction and 32 Kilobyte of data cache (per core)
2. 256 Kilobyte (per core)
3. 45 Megabyte (shared)

Testing the correctness of the program is not as straight-forward as it seems at first glance. Like stated in Section 2, after each superstep, the updated particles are written out to file. Comparing the output of the optimized version of the program with the original one would be a sufficient test for correctness, if it were not for floating point rounding errors. These accumulate and after a certain amount of time, the numbers generated by the optimized version will be too different from the original ones.

In order to avoid getting different results, just because of floating point rounding errors, the original version of the code was augmented by a special test setting. This test setting differs from the normal program, because it reads the data it has written out after a superstep back in. That way, the next superstep will work with the floating point numbers that are crippled by writing them to file. The floating point numbers are written to file text based in exponential form with 16 digits, eight digits on the right side of the decimal point (see e.g. Shene, 2020, for formatting io in Fortran). Changing the output format to a more precise representation is not possible, because this would mean the files generated as output would not have the same format as the input file with the initial states of the particles. The optimized version of the program has the exact same test setting. This allows for effectively comparing the output files of both versions, because floating point rounding errors now only accumulate over a single superstep.

Once the discrepancy between the output values of the optimized version and the original one surpasses a predefined error level, the optimization is deemed to result in an incorrect version of the program. The predefined error level was set to be 0.05. If any output file contains a `NaN` value, the program is also deemed incorrect. The regression test suite was implemented with a Python script.

The program runs five supersteps. Each superstep encompasses 100 iterations. The input file which was used contained 4096 particles. Computation was done using double precision floats. The goal of the optimization process was to reduce the wall-clock time of the program to a minimum, while bearing in mind portability and more importantly maintainability. The program measures the time it needs for each superstep and its overall time, including the file output. In order to build the benchmark suite around the program, the timings are exported to another file when the program has finished the simulation. The program was tested by running it ten times on a compute node of Cirrus and taking the average from those ten runs as the performance measurement. Running it ten times is more than enough to get a stable average, because running the program as a job on a compute node of Cirrus means exclusive hardware access to that node. Only IO performance can be influenced by other users, because Cirrus uses Lustre for its file system which is shared (EPCC, 2020b). As will be discussed below, IO performance is actually negligible when it comes to performance optimization when compared to the computational effort of the simulation.

4. Optimizations

5. Discussion

6. Conclusion

References

Mark Buxton. Haswell New Instruction Descriptions Now Available!, 2011. URL https://software.intel.com/en-us/blogs/2011/06/13/haswell-new-instruction-descriptions-now-available/.

EPCC. Cirrus, 2020a. URL https://www.cirrus.ac.uk.

EPCC. Cirrus Hardware, 2020b. URL http://www.cirrus.ac.uk/about/hardware.html.

Richard P. (Richard Philipps) Feynman. *The Feynman lectures on physics*. Addison-Wesley Pub. Co., Reading, Mass. ; London, 1963. ISBN 0201020106.

Intel. Intel Fortran Compiler 18.0 for Linux, 2018. URL https://software.intel.com/en-us/articles/intel-fortran-compiler-180-for-linux-release-notes-for-intel-parallel-studio-xe-2018.

Intel. Intel Fortran Compiler 19.0 for Linux, 2019. URL https://software.intel.com/en-us/articles/intel-fortran-compiler-190-for-linux-release-notes-for-intel-parallel-studio-xe-2019#new_features.

C. K. Shene. Fortran Formats, 2020. URL https://pages.mtu.edu/~shene/COURSES/cs201/NOTES/chap05/format.html.