Solutions to Problem Set 8

Fundamentals of Simulation Methods 8 ECTS
Heidelberg University WiSe 20/21

Elias Olofsson
ub253@stud.uni-heidelberg.de

January 20, 2021

1 Tree code for gravity calculation (20 pts)

To implement a tree algorithm for faster force calculations of a N-body system, utilizing hierarchical multipole expansions, I started using the C-code template given in the file `tree.c`. For my completed and fully functional code, please see the attached C-code with the same filename.

Differing from the specific task given in this exercise sheet, I have here chosen to also implement a code that can use up to quadrupole moments for the force calculations, while the original task only required an implementation using monopole moments. Thus, all the following analysis have also been performed for the equivalent 'up to quadrupole' cases.

Starting out, I have for easier compilation included a `Makefile` which can easily compile the program, run memory tests and clean up binaries with a few simple commands. As an example on how to compile and run the program, please reference Fig.(1). Here in this figure, I also showcase an automated test script `autotest.sh` that I created for smoother data collection. To inspect the bash script, please see the attached files.

Using this automated test script, I can get execution times for the tree code and the direct summation methods, as well as relative error and average number of node-particle interactions per particle, and print the results to a text file for a large range of input parameters. Specifically, I let the script perform test with $N = \{5000, 10000, 20000, 40000\}$ particles and $\theta_c = \{0.2, 0.4, 0.8\}$ as opening angle thresholds, saving the results in the file `tree_test.txt`. This text file can readily be imported and processed in any software of choice.

Loading the data into the Jupyter Notebook `plotting.ipynb`, we can create a `pandas` dataframe which can relatively conveniently can be exported into Latex code. To observe the obtained data, please reference Tab.(1) or the Notebook itself.

Here in the table we can see a few things, namely how highly sensitive the tree code is on the opening angle threshold θ_c. In the lecture we theoretically derived that the average number of particle-node interactions per particle for a constant number of particles N scales with the opening angle threshold

Multipole:		Tree code t(s)		Direct t(s)	Relative error		Avg interactions	
N	θ_c	Mono	Quad	-	Mono	Quad	Mono	Quad
5000	0.2	0.605	1.067	1.780	2.79e-04	2.21e-05	1735.5	1735.5
	0.4	0.163	0.297	1.777	2.18e-03	3.84e-04	501.8	501.8
	0.8	0.034	0.064	1.783	1.49e-02	6.89e-03	110.0	110.0
10000	0.2	2.109	3.419	7.807	2.45e-04	2.01e-05	2403.9	2403.9
	0.4	0.486	0.952	8.345	1.81e-03	3.32e-04	621.9	621.9
	0.8	0.095	0.184	8.132	1.28e-02	6.11e-03	127.8	127.8
20000	0.2	5.445	9.589	31.585	2.25e-04	1.90e-05	3120.2	3120.2
	0.4	1.129	1.927	29.246	1.53e-03	2.94e-04	737.4	737.4
	0.8	0.226	0.420	29.623	1.06e-02	5.34e-03	144.2	144.2
40000	0.2	16.810	25.686	116.859	1.95e-04	1.59e-05	3976.4	3976.4
	0.4	3.273	4.933	118.655	1.30e-03	2.48e-04	867.0	867.0
	0.8	0.588	0.888	118.040	8.91e-03	4.82e-03	161.8	161.8

Table 1 – Table of results generated from the automated test script `autotest.sh`, showing execution time in seconds, relative error and average number of node-particle interactions per particle, for the three methods; tree code with/without quadrupole moment and direct summation.

more decisive in reality: how does the error scale with θ_c?

as $\mathcal{O}(\theta_c^{-3})$. And since the complexity of the full algorithm is proportional to the average number of interactions times the number of particles, we have the same dependency $\mathcal{O}(\theta_c^{-3})$ on the opening angle threshold for the full complexity, given a constant number of particles N. Without doing a deeper complexity analysis, we notice this highly sensitive dependence in the data for both the execution time and average number of interactions per particle, which seems to agree with the theory.

Furthermore, we can see how unwieldy the direct summation method quickly becomes for larger systems of particles. Going from $N = 5000$ to $N = 40000$ we have barely increased the system size by one order of magnitude, but the execution time has increased by almost two orders of magnitude, implying a complexity dependency of $\mathcal{O}(N^2)$ with number of particles N, which agrees with the theory on direct summation.

We can also observe the gain in accuracy by including the quadrupole terms into the force calculations for the tree code. While the inclusion of quadrupoles approximately increases the complexity by a factor of 2, the relative error drops by almost one order of magnitude. Additionally, my quadrupole implementation is far from perfectly optimized as well, and as such, the difference between only using monopoles and using up to quadrupoles should reasonably decrease if more thought and time is invested into the specifics of the implementation.

Continuing our analysis, we can graph the execution times for all three methods in a loglog-plot, as seen in Fig.(2).

you could / should have also looked into rel. error in dependency of θ_c [2] since this was asked in the sheet (more or less) – 1p and should scale with $\theta_c^{3.5}$

```
user@host:~/$ make
gcc tree.c -o tree -std=c99 -Wall -g -lm
user@host:~/$ ./tree
Usage:
         [-n] N [-t] theta [-q] [-m]
         where N is an integer, giving the numer of particles,
         and theta is the opening angle threshold.
         Use -q to estimate acceleration up to quadrupole moments.
         Use -m to output machine-readable table.

user@host:~/$ ./tree -n 5000 -t 0.8
Using only monopole moments.

Number of particles:       N     = 5000
Opening angle threshold:   theta = 0.8000

Force calculation with tree:         0.053036  sec
Calculation with direct summation:   1.79233   sec

Average relative error:                              0.0149259
Average particle-node interactions per particle:   109.968
user@host:~/$ ./tree -n 5000 -t 0.8 -q
Using monopole & quadrupole moments.

Number of particles:       N     = 5000
Opening angle threshold:   theta = 0.8000

Force calculation with tree:         0.084367  sec
Calculation with direct summation:   1.78976   sec

Average relative error:                              0.0068861
Average particle-node interactions per particle:   109.968
user@host:~/$ ./tree -n 5000 -t 0.8 -q -m
5000 0.800000 1  0.092532   1.83633 0.0068861 109.968
user@host:~/$ time sh autotest.sh
N = 5000, T = 0.2
N = 5000, T = 0.4
N = 5000, T = 0.8
N = 10000, T = 0.2
N = 10000, T = 0.4
N = 10000, T = 0.8
N = 20000, T = 0.2
N = 20000, T = 0.4
N = 20000, T = 0.8
N = 40000, T = 0.2
N = 40000, T = 0.4
N = 40000, T = 0.8
Test Complete.

real     17m8,259s
user     17m8,065s
sys      0m0,143s
```

Figure 1 – The printouts from a terminal compiling and running the tree code `tree.c`. First, the code is compiled using `make` and ran without arguments. A short user guide is printer showing available options. Then, in two consecutive runs, the tree code is ran using $N = 5000$ particles and opening angle threshold $\theta_c = 0.8$, with and without using quadrupole moments. After that, the machine-readable table option and the automatic test script `autotest.sh` is showcased.

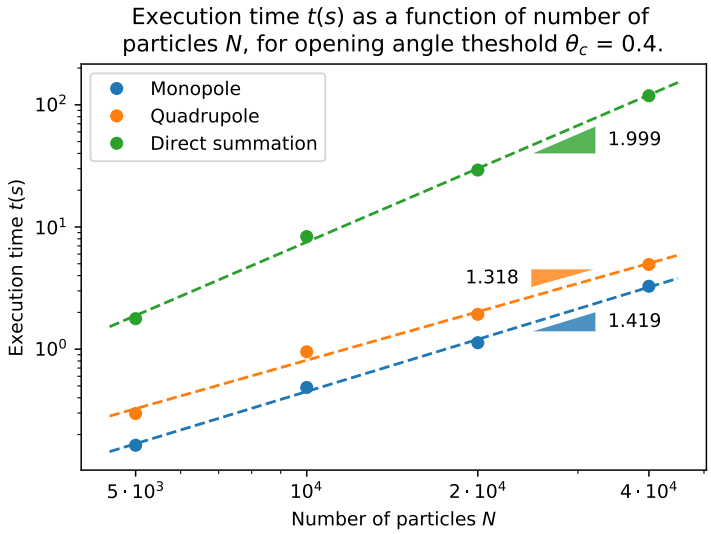

Figure 2 – Execution times $t(s)$ as a function of number of particles N, using a specific opening angle threshold $\theta_c = 0.4$. For each of the methods tree code with/without quadrupole moments and direct summation, a linear regression in loglog have been performed. The obtained slope in the loglog-frame for each of the methods are indicated in the figure.

Here we chose the specific opening angle threshold of $\theta_d = 0.4$ and performed linear regressions in the loglog-frame for each type. As seen in the figure, each of the slopes were estimated to $1.419, 1.318$ and 1.99 for tree code with monopoles, tree code with quadrupoles and direct summation, respectively. This gives a clear indication of the general complexity for each of the methods, since the slope k implies a complexity as $\mathcal{O}(N^k)$. Thus, direct summation is quadratic in complexity as predicted, while the tree codes are something in between linear and quadratic. From theory we expect these algorithms to scale as $\mathcal{O}(N \log(N))$, and thus the obtained results from the linear regressions seems reasonable.

Using these linear regressions, we can extrapolate and estimate the execution times for each of the algorithms for a system of $N = 10^{10}$ particles. For each of the tree codes we obtain an estimated time of $1.5 \cdot 10^8$ seconds and $6.5 \cdot 10^7$ seconds for monopole and quadrupole respectively, which corresponds to 4.7 and 2.1 years. For the direct summation, we obtain a staggering $7.5 \cdot 10^{12}$ seconds, which equals $2.4 \cdot 10^5$ years. Thus, pure direct summation is completely unfeasible for large systems of particles.

As a side note, even though we estimated a shorter execution time for the monopole tree code compared to the quadrupole variant, this will not occur in reality, since the complexity is inherently larger for the quadrupole code and will hence always result in a longer execution time. This highlights the problems of linearly extrapolating a non-linear dependency, and additionally doing so with only four data points in each regression.