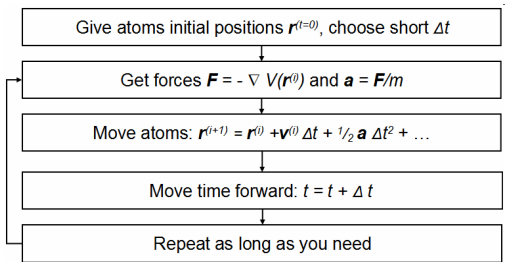
Work Assignment
Phase 1

Learning outcomes

This assignment phase aims to explore optimisation techniques applied to a (**single threaded**) program, using tools for code analysis/profiling and to improve and evaluate its final performance (execution time).

Introduction

The program to analyse and optimise is part of a simple molecular dynamics' simulation code applied to atoms of argon gas (original code version in *FoleyLab/MolecularDynamics: Simple Molecular Dynamics*).
The code follows a generic approach to simulate particle movements over time steps, using the Newton law:



The code uses Lennard Jones potential to describe the interactions among two particles (force/potential energy). The Verlet integration method is used to calculate the particles trajectories over time. Boundary conditions (i.e., particles moving outside the simulation space) are managed by elastic walls.

On this assignment phase students should improve the execution time of the simulation code (supplied in the course BB webpage), following these steps:

- analyse the code (with the given data) and get its profile, to identify the key blocks requiring code or data structures modifications, which will lead to improved execution performance; we suggest the use of a call-graph with `gprof` (as presented in lecture slides, see below);
- explore the **main goal** of this work assignment: to reduce the execution time for the given set of conditions (number of atoms, density, etc..); note that all simulation outputs should be maintained and that in the provided BB code some of the I/O in the original code was modified to comments and can be used for debug purposes, but it should not be included in the final submission;
- estimate the execution performance before applying any modification, since there are a wide range of possible optimisation techniques, many of them without any impact on performance;
- keep code legibility as much as possible, also a **key goal** in this work, since it is very relevant to have a clean code for upcoming assignment phases.

Groups, submission format and dates

The work assignment is to be performed by a group of two students (single student groups is an exception requiring a valid justification, and groups with three students are not allowed).

The work must be submitted through the e-learning platform, **compressed into a zip file** that, when unzipped, should generate a base directory whose name is the groups elements, e.g., a43000_pg54000). It should include:

- a 2-page PDF report (annexes excluded, these might be read by the evaluator) using the IEEE template (in https://www.ieee.org/conferences/publishing/templates.html);
- a subdirectory with all source code (please, do not submit executables);
- a `Makefile` that generates an executable in the base directory (see example in annex).

Submission deadline: 23:59, 23-Oct-23.

The defence of this assignment will be performed during the oral presentation of the WA-Phase 3 (in Jan'24).

Evaluation

The evaluation of this work will consider:

- **(i)** the **execution time** of the submitted source code with the given set of conditions, by automatically applying the `Makefile` and validating the expected outputs; failure leads to zero points **(50%);**
- **(ii)** the implemented **optimisations** (ILP improvements, memory hierarchy, data structures organization, vectorisation, ...) **(20%);**
- **(iii)** the **code legibility and report quality** (the English quality is not considered, but avoid automatic translation...); the report should include the code analysis/profile, an explanation/justification of the implemented optimisations, and the models and metrics that explain the results **(30%).**

Call-graph in the lecture slides

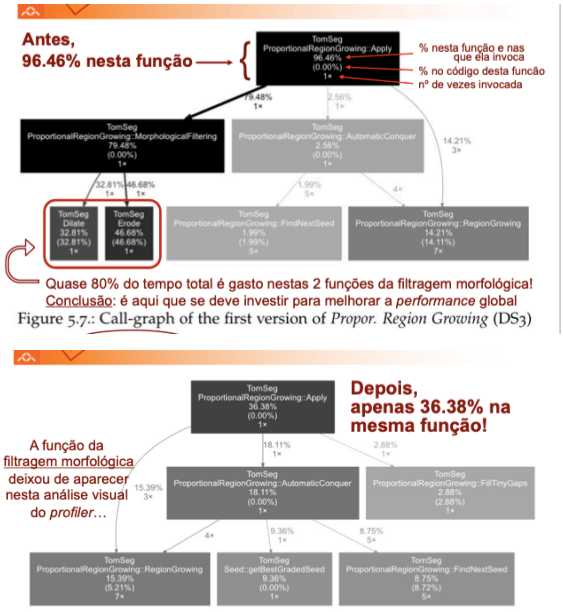


Figure 5.7.: Call-graph of the first version of *Propor. Region Growing* (DS3)

Mais comum para aplicações em ambiente GNU/Linux:

- `gprof` : disponível em `gcc/binutils`
- `gprof` requer compilação com `-pg` para gerar o ficheiro `gmon.out` contendo os dados para o *profiler*
- `gprof` lê a informação em `gmon.out` e produz um relatório em `main.gprof`
- `gprof2dot` lê `main.gprof` e gera uma representação visual dos dados tal como apresentado nos slides anteriores
- `gprof2dot` disponível em https://github.com/jrfonseca/gprof2dot
- para visualizar a imagem do *call graph* usar o comando `xdot`

https://developer.ridgerun.com/wiki/index.php/Profiling_GNU/Linux_applications

Annex - A simple *`Makefile`*

The work submission must include a `Makefile` that generates an executable named `MD.exe` on the base directory. All source files must be placed on a subdirectory (e.g., `src`).
In this example the program can be executed with `make run`. Students can fill `CFLAGS` with any flags supported by the `gcc 9.3.0` available on the SeARCH cluster. Example of a simple `Makefile`:

```
CC       = gcc
SRC      = src/
CFLAGS   = # none

.DEFAULT_GOAL = MD.exe

MD.exe: $(SRC)MD.cpp
        $(CC) $(CFLAGS) $(SRC)MD.cpp -lm -o MD.exe

clean:
        rm ./MD.exe

run:
        ./MD.exe < inputdata.txt
```