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).

Students should become aware of the case study specificities: i) the time profile of the several functions; dependency constraints among functions (e.g., call-graphs, and dependencies) and inside each function; data locality properties; and support for vector processing.

Introduction

The program to analyse and optimise one of the simplest fluid dynamics simulation code, a 3D version of the Jos Stam's stable fluid solver. The original code and description can be found in the repository: https://github.com/BlainMaguire/3dfluid. See the provided paper and illustrative interactive execution for a better understanding of the problem.

On this assignment phase students should perform a case study analysis and improve the execution time of a non-interactive version of the simulation code, supplied in the following git repository: https://github.com/jgbarbosa/3dfluid.

Suggested steps to follow:

- analyse the code (using the data given in the events.txt file in the repository) and get its profile, to identify the key blocks requiring modifications, which will lead to improved execution performance; we suggest the use of a call-graph with `gprof` (as in the lecture slides, see below);
- explore the **main goal** of this work assignment: to reduce the execution time; note that the simulation output must be maintained;
- estimate the execution performance improvement 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 three students.

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 group elements, e.g., pg43000_pg54000_pg76000. 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, or trash files/directories);
- a `Makefile` that generates an executable named `fluid_sim` in the base directory (please, follow the `Makefile` template given in the code repository).

It is suggested that each group should perform a private fork of the supplied git repository.

Before submission students must check if their project complies to the submission rules. A script will be latter provided for this purpose.

Submission deadline: 23:59, 21-Oct-24.

The defence of this assignment will be performed during the oral presentation of the WA-Phase 3 (in Jan'25).

Evaluation

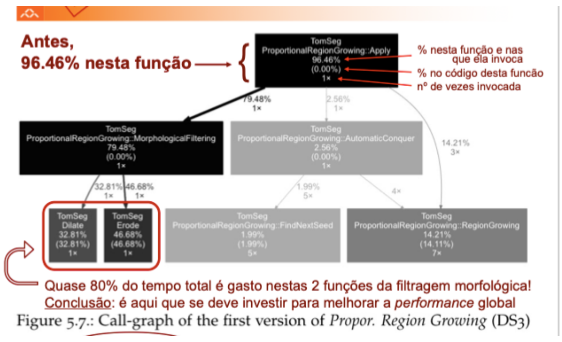
The evaluation of this work will consider:

(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%);**

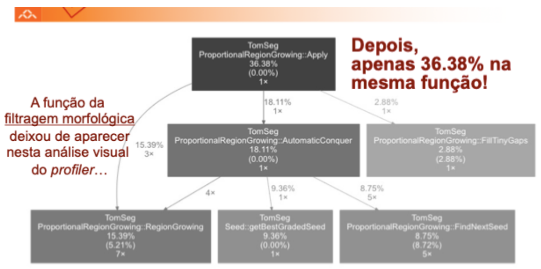
(ii) the implemented **optimisations** (locality, vectorisation, ...) **(20%);**

(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%).**

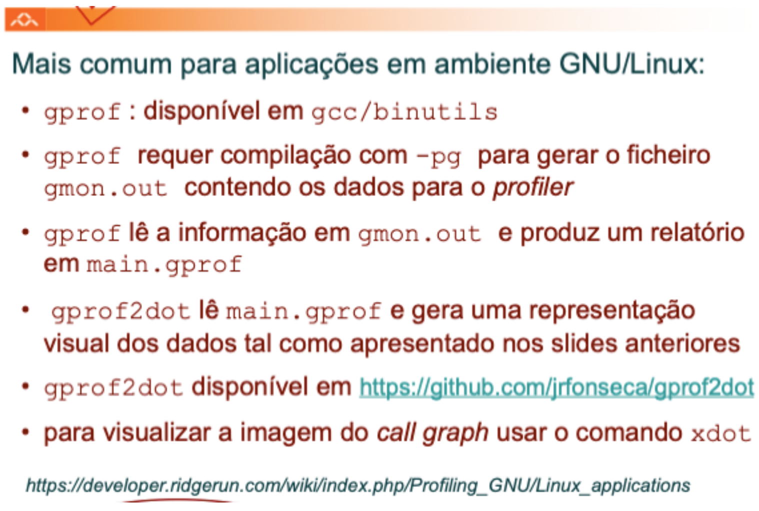
Call-graph in the lecture slides



Antes,
96.46% nesta função
TomSeg ProportionalRegionGrowing::Apply 96.46% (0.00%) 1×
% nesta função e nas que ela invoca
% no código desta funcão
nº de vezes invocada
79.48% 1×
2.56% 1×
14.21% 3×
TomSeg ProportionalRegionGrowing::MorphologicalFiltering 79.48% (0.00%) 1×
TomSeg ProportionalRegionGrowing::AutomaticConquer 2.56% (0.00%) 1×
32.81% 1×
46.68% 1×
1.99% 5×
4×
TomSeg Dilate 32.81% (32.81%) 1×
TomSeg Erode 46.68% (46.68%) 1×
TomSeg ProportionalRegionGrowing::FindNextSeed 1.99% (1.99%) 5×
TomSeg ProportionalRegionGrowing::RegionGrowing 14.21% (14.11%) 7×
Quase 80% do tempo total é gasto nestas 2 funções da filtragem morfológica!
Conclusão: é aqui que se deve investir para melhorar a performance global
Figure 5.7.: Call-graph of the first version of Propor. Region Growing (DS3)



TomSeg ProportionalRegionGrowing::Apply 36.38% (0.00%) 1×
Depois,
apenas 36.38% na mesma função!
A função da filtragem morfológica deixou de aparecer nesta análise visual do profiler…
18.11% 1×
2.88% 1×
15.39% 3×
TomSeg ProportionalRegionGrowing::AutomaticConquer 18.11% (0.00%) 1×
TomSeg ProportionalRegionGrowing::FillTinyGaps 2.88% (2.88%) 1×
4×
9.36% 1×
8.75% 5×
TomSeg ProportionalRegionGrowing::RegionGrowing 15.39% (5.21%) 7×
TomSeg Seed::getBestGradedSeed 9.36% (0.00%) 1×
TomSeg ProportionalRegionGrowing::FindNextSeed 8.75% (8.72%) 5×


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