

# CAD-Trabalho 2 - gprof para a perfilagem

Amanda Lucio

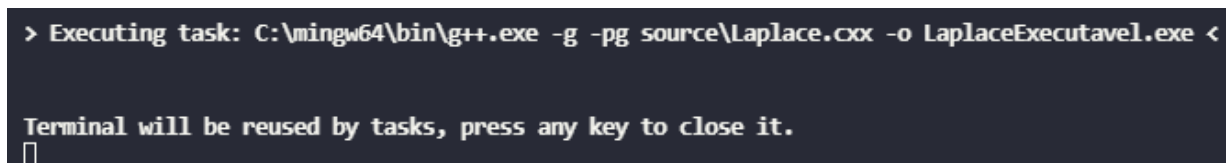
August 2021

## 1 Introdução

A equação de Laplace descreve uma série fenômenos físicos. No algoritmo temos o uso da discretização. À partir do domínio realiza-se a escolha de um conjunto de pontos e aproximamos as derivadas presentes na equação para uma equação de diferenças finitas. A perfilagem permite encontrar os hotspots do código (pontos de maior consumo de tempo de cpu), modificando-os e facilitando a melhoria de desempenho do programa. O Gprof é o perfilador do GNU. Ele permite obter o tempo das sub-rotinas e gerar gráficos de chamadas.

## 2 Passos necessários com gprof

1. Compile utilizando a flag `-pg`.

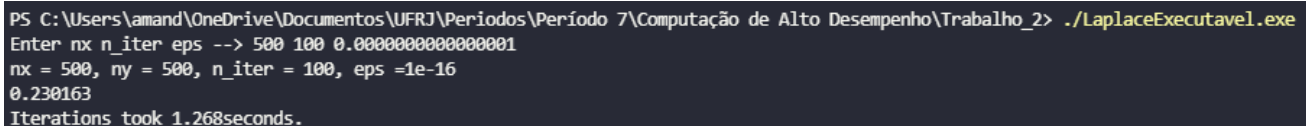


```
> Executing task: C:\mingw64\bin\g++.exe -g -pg source\Laplace.cxx -o LaplaceExecutavel.exe <

Terminal will be reused by tasks, press any key to close it.
█
```

Figure 1: Configurações do computador utilizado

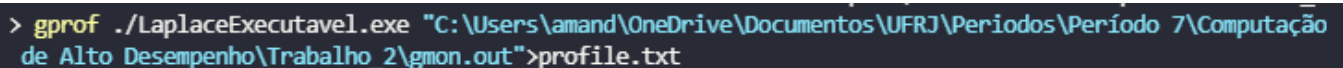
2. Rode o executável, para que um arquivo `gmon.out` seja gerado



```
PS C:\Users\amand\OneDrive\Documentos\UFRJ\Periodos\Período 7\Computação de Alto Desempenho\Trabalho_2> ./LaplaceExecutavel.exe
Enter nx n_iter eps --> 500 100 0.0000000000000001
nx = 500, ny = 500, n_iter = 100, eps =1e-16
0.230163
Iterations took 1.268seconds.
```

Figure 2: Imagem da task de execução

3. Execute o gprof utilizando como parâmetros o executável e o arquivo.out. Sendo possível redirecionar o arquivo .out para um txt.



```
> gprof ./LaplaceExecutavel.exe "C:\Users\amand\OneDrive\Documentos\UFRJ\Periodos\Período 7\Computação de Alto Desempenho\Trabalho_2\gmon.out">profile.txt
```

Figure 3: Imagem da execução do gprof

## 3 Relatório Gerado

O relatório gerado pode ser encontrado no Apêndice (Apêndice 7.1). Por meio do relatório identificamos na sessão Flat Profile que o tempo de execução do código foi de 0.80s. Além disso os hotspots são as funções `timeStep` e `SQR`. O projeto poderia ser melhor estruturado em relação a boas práticas de HPC através da armazenagem de cálculos em registradores previamente e por meio da eliminação de algumas funções.

## 4 Alterações

1. Expressões constantes em laços - Utilização da variável sumdx dy:

```
sumdx dy = dx2 + dy2

for (int i=1; i<nx-1; ++i) {
    for (int j=1; j<ny-1; ++j) {
        tmp = u[i][j];
        u[i][j] = ((u[i-1][j] + u[i+1][j])*dy2 +
                    (u[i][j-1] + u[i][j+1])*dx2)*0.5/(sumdx dy);
        err += SQR(u[i][j] - tmp);
    }
}
return sqrt(err);
```

Figure 4: Criação de variável sumdx dy para expressão constante

2. In-lining - Retirada da função SQL

```
for (int i=1; i<nx-1; ++i) {
    for (int j=1; j<ny-1; ++j) {
        tmp = u[i][j];
        u[i][j] = ((u[i-1][j] + u[i+1][j])*dy2 +
                    (u[i][j-1] + u[i][j+1])*dx2)*0.5/(sumdx dy);
        err += (u[i][j] - tmp)*(u[i][j] - tmp);
    }
}
return sqrt(err);
```

Figure 5: Substituição da função SQR

## 5 Conclusão

O relatório gerado pode ser encontrado no Apêndice (Apêndice 7.2).

## 6 Anexos

### 6.1 Relatório Inicial gprof

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
81.25	0.65	0.65	100	6.50	6.60	LaplaceSolver::timeStep(double)
8.75	0.72	0.07				__fentry__
8.75	0.79	0.07				_mcount_private
1.25	0.80	0.01	24800400	0.00	0.00	SQR(double const&)
0.00	0.80	0.00	2000	0.00	0.00	BC(double, double)
0.00	0.80	0.00	2	0.00	0.00	seconds()

0.00	0.80	0.00	1	0.00	0.00	LaplaceSolver::initialize()
0.00	0.80	0.00	1	0.00	660.00	LaplaceSolver::solve(int, double)
0.00	0.80	0.00	1	0.00	0.00	LaplaceSolver::LaplaceSolver(Grid*)
0.00	0.80	0.00	1	0.00	0.00	LaplaceSolver::~~LaplaceSolver()
0.00	0.80	0.00	1	0.00	0.00	Grid::setBCFunc(double (*)(double, double))
0.00	0.80	0.00	1	0.00	0.00	Grid::Grid(int, int)

%            the percentage of the total running time of the  
time            program used by this function.

cumulative a running sum of the number of seconds accounted  
seconds    for by this function and those listed above it.

self        the number of seconds accounted for by this  
seconds    function alone. This is the major sort for this  
            listing.

calls       the number of times this function was invoked, if  
            this function is profiled, else blank.

self        the average number of milliseconds spent in this  
ms/call    function per call, if this function is profiled,  
            else blank.

total       the average number of milliseconds spent in this  
ms/call    function and its descendents per call, if this  
            function is profiled, else blank.

name        the name of the function. This is the minor sort  
            for this listing. The index shows the location of  
            the function in the gprof listing. If the index is  
            in parenthesis it shows where it would appear in  
            the gprof listing if it were to be printed.

Copyright (C) 2012-2018 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,  
are permitted in any medium without royalty provided the copyright  
notice and this notice are preserved.

Call graph (explanation follows)

granularity: each sample hit covers 4 byte(s) for 1.25% of 0.80 seconds

index	% time	self	children	called	name
					<spontaneous>
[1]	82.5	0.00	0.66		main [1]
		0.00	0.66	1/1	LaplaceSolver::solve(int, double) [3]
		0.00	0.00	2/2	seconds() [76]
		0.00	0.00	1/1	Grid::Grid(int, int) [81]
		0.00	0.00	1/1	Grid::setBCFunc(double (*)(double, double)) [80]
		0.00	0.00	1/1	LaplaceSolver::LaplaceSolver(Grid*) [78]
		0.00	0.00	1/1	LaplaceSolver::~~LaplaceSolver() [79]

-----

		0.65	0.01	100/100	LaplaceSolver::solve(int, double) [3]
[2]	82.5	0.65	0.01	100	LaplaceSolver::timeStep(double) [2]
		0.01	0.00	24800400/24800400	SQR(double const&) [6]
-----					
		0.00	0.66	1/1	main [1]
[3]	82.5	0.00	0.66	1	LaplaceSolver::solve(int, double) [3]
		0.65	0.01	100/100	LaplaceSolver::timeStep(double) [2]
-----					
					<spontaneous>
[4]	8.8	0.07	0.00		__fentry__ [4]
-----					
					<spontaneous>
[5]	8.8	0.07	0.00		_mcount_private [5]
-----					
		0.01	0.00	24800400/24800400	LaplaceSolver::timeStep(double) [2]
[6]	1.3	0.01	0.00	24800400	SQR(double const&) [6]
-----					
		0.00	0.00	2000/2000	Grid::setBCFunc(double (*)(double, double)) [80]
[75]	0.0	0.00	0.00	2000	BC(double, double) [75]
-----					
		0.00	0.00	2/2	main [1]
[76]	0.0	0.00	0.00	2	seconds() [76]
-----					
		0.00	0.00	1/1	LaplaceSolver::LaplaceSolver(Grid*) [78]
[77]	0.0	0.00	0.00	1	LaplaceSolver::initialize() [77]
-----					
		0.00	0.00	1/1	main [1]
[78]	0.0	0.00	0.00	1	LaplaceSolver::LaplaceSolver(Grid*) [78]
		0.00	0.00	1/1	LaplaceSolver::initialize() [77]
-----					
		0.00	0.00	1/1	main [1]
[79]	0.0	0.00	0.00	1	LaplaceSolver::~LaplaceSolver() [79]
-----					
		0.00	0.00	1/1	main [1]
[80]	0.0	0.00	0.00	1	Grid::setBCFunc(double (*)(double, double)) [80]
		0.00	0.00	2000/2000	BC(double, double) [75]
-----					
		0.00	0.00	1/1	main [1]
[81]	0.0	0.00	0.00	1	Grid::Grid(int, int) [81]
-----					

This table describes the call tree of the program, and was sorted by the total amount of time spent in each function and its children.

Each entry in this table consists of several lines. The line with the index number at the left hand margin lists the current function. The lines above it list the functions that called this function, and the lines below it list the functions this one called.

This line lists:

index A unique number given to each element of the table.

Index numbers are sorted numerically.

The index number is printed next to every function name so it is easier to look up where the function is in the table.

% time This is the percentage of the 'total' time that was spent in this function and its children. Note that due to different viewpoints, functions excluded by options, etc,

these numbers will NOT add up to 100%.

self This is the total amount of time spent in this function.

children This is the total amount of time propagated into this function by its children.

called This is the number of times the function was called. If the function called itself recursively, the number only includes non-recursive calls, and is followed by a '+' and the number of recursive calls.

name The name of the current function. The index number is printed after it. If the function is a member of a cycle, the cycle number is printed between the function's name and the index number.

For the function's parents, the fields have the following meanings:

self This is the amount of time that was propagated directly from the function into this parent.

children This is the amount of time that was propagated from the function's children into this parent.

called This is the number of times this parent called the function '/' the total number of times the function was called. Recursive calls to the function are not included in the number after the '/'.

name This is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between the name and the index number.

If the parents of the function cannot be determined, the word '<spontaneous>' is printed in the 'name' field, and all the other fields are blank.

For the function's children, the fields have the following meanings:

self This is the amount of time that was propagated directly from the child into the function.

children This is the amount of time that was propagated from the child's children to the function.

called This is the number of times the function called this child '/' the total number of times the child was called. Recursive calls by the child are not listed in the number after the '/'.

name This is the name of the child. The child's index number is printed after it. If the child is a member of a cycle, the cycle number is printed between the name and the index number.

If there are any cycles (circles) in the call graph, there is an entry for the cycle-as-a-whole. This entry shows who called the cycle (as parents) and the members of the cycle (as children.) The '+' recursive calls entry shows the number of function calls that were internal to the cycle, and the calls entry for each member shows, for that member, how many times it was called from other members of the cycle.

Copyright (C) 2012-2018 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification, are permitted in any medium without royalty provided the copyright notice and this notice are preserved.

Index by function name

```
[75] BC(double, double)      [3] LaplaceSolver::solve(int, double) [80] Grid::setBCFunc(double (*)(double, double))
[6]  SQR(double const&)      [2] LaplaceSolver::timeStep(double) [81] Grid::Grid(int, int)
[76] seconds()               [78] LaplaceSolver::LaplaceSolver(Grid*) [4]  __fentry__
[77] LaplaceSolver::initialize() [79] LaplaceSolver::~LaplaceSolver() [5]  _mcount_private
```

## 6.2 Relatório Final gprof

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
100.00	0.32	0.32	100	3.20	3.20	LaplaceSolver::timeStep(double)
0.00	0.32	0.00	2000	0.00	0.00	BC(double, double)
0.00	0.32	0.00	2	0.00	0.00	seconds()
0.00	0.32	0.00	1	0.00	0.00	LaplaceSolver::initialize()
0.00	0.32	0.00	1	0.00	320.00	LaplaceSolver::solve(int, double)
0.00	0.32	0.00	1	0.00	0.00	LaplaceSolver::LaplaceSolver(Grid*)
0.00	0.32	0.00	1	0.00	0.00	LaplaceSolver::~LaplaceSolver()
0.00	0.32	0.00	1	0.00	0.00	Grid::setBCFunc(double (*)(double, double))
0.00	0.32	0.00	1	0.00	0.00	Grid::Grid(int, int)

%  
time      the percentage of the total running time of the  
          program used by this function.

cumulative a running sum of the number of seconds accounted  
seconds    for by this function and those listed above it.

self       the number of seconds accounted for by this  
seconds    function alone. This is the major sort for this  
          listing.

calls      the number of times this function was invoked, if  
          this function is profiled, else blank.

self       the average number of milliseconds spent in this  
ms/call    function per call, if this function is profiled,  
          else blank.

total        the average number of milliseconds spent in this  
ms/call     function and its descendents per call, if this  
            function is profiled, else blank.

name        the name of the function. This is the minor sort  
            for this listing. The index shows the location of  
            the function in the gprof listing. If the index is  
            in parenthesis it shows where it would appear in  
            the gprof listing if it were to be printed.

Copyright (C) 2012-2018 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification,  
are permitted in any medium without royalty provided the copyright  
notice and this notice are preserved.

Call graph (explanation follows)

granularity: each sample hit covers 4 byte(s) for 3.13% of 0.32 seconds

index	% time	self	children	called	name
					<spontaneous>
[1]	100.0	0.00	0.32		main [1]
		0.00	0.32	1/1	LaplaceSolver::solve(int, double) [3]
		0.00	0.00	2/2	seconds() [73]
		0.00	0.00	1/1	Grid::Grid(int, int) [78]
		0.00	0.00	1/1	Grid::setBCFunc(double (*)(double, double)) [77]
		0.00	0.00	1/1	LaplaceSolver::LaplaceSolver(Grid*) [75]
		0.00	0.00	1/1	LaplaceSolver::~LaplaceSolver() [76]
-----					
		0.32	0.00	100/100	LaplaceSolver::solve(int, double) [3]
[2]	100.0	0.32	0.00	100	LaplaceSolver::timeStep(double) [2]
-----					
		0.00	0.32	1/1	main [1]
[3]	100.0	0.00	0.32	1	LaplaceSolver::solve(int, double) [3]
		0.32	0.00	100/100	LaplaceSolver::timeStep(double) [2]
-----					
		0.00	0.00	2000/2000	Grid::setBCFunc(double (*)(double, double)) [77]
[72]	0.0	0.00	0.00	2000	BC(double, double) [72]
-----					
		0.00	0.00	2/2	main [1]
[73]	0.0	0.00	0.00	2	seconds() [73]
-----					
		0.00	0.00	1/1	LaplaceSolver::LaplaceSolver(Grid*) [75]
[74]	0.0	0.00	0.00	1	LaplaceSolver::initialize() [74]
-----					
		0.00	0.00	1/1	main [1]
[75]	0.0	0.00	0.00	1	LaplaceSolver::LaplaceSolver(Grid*) [75]
		0.00	0.00	1/1	LaplaceSolver::initialize() [74]
-----					
		0.00	0.00	1/1	main [1]
[76]	0.0	0.00	0.00	1	LaplaceSolver::~LaplaceSolver() [76]
-----					
		0.00	0.00	1/1	main [1]

[77]	0.0	0.00	0.00	1	Grid::setBCFunc(double (*)(double, double)) [77]
		0.00	0.00	2000/2000	BC(double, double) [72]
-----					
		0.00	0.00	1/1	main [1]
[78]	0.0	0.00	0.00	1	Grid::Grid(int, int) [78]
-----					

This table describes the call tree of the program, and was sorted by the total amount of time spent in each function and its children.

Each entry in this table consists of several lines. The line with the index number at the left hand margin lists the current function. The lines above it list the functions that called this function, and the lines below it list the functions this one called.

This line lists:

index A unique number given to each element of the table.

Index numbers are sorted numerically.

The index number is printed next to every function name so

it is easier to look up where the function is in the table.

% time This is the percentage of the 'total' time that was spent in this function and its children. Note that due to different viewpoints, functions excluded by options, etc, these numbers will NOT add up to 100%.

self This is the total amount of time spent in this function.

children This is the total amount of time propagated into this function by its children.

called This is the number of times the function was called.

If the function called itself recursively, the number only includes non-recursive calls, and is followed by a '+' and the number of recursive calls.

name The name of the current function. The index number is printed after it. If the function is a member of a cycle, the cycle number is printed between the function's name and the index number.

For the function's parents, the fields have the following meanings:

self This is the amount of time that was propagated directly from the function into this parent.

children This is the amount of time that was propagated from the function's children into this parent.

called This is the number of times this parent called the function '/' the total number of times the function was called. Recursive calls to the function are not included in the number after the '/'.

name This is the name of the parent. The parent's index number is printed after it. If the parent is a member of a cycle, the cycle number is printed between



the name and the index number.

If the parents of the function cannot be determined, the word '<spontaneous>' is printed in the 'name' field, and all the other fields are blank.

For the function's children, the fields have the following meanings:

self This is the amount of time that was propagated directly from the child into the function.

children This is the amount of time that was propagated from the child's children to the function.

called This is the number of times the function called this child '/' the total number of times the child was called. Recursive calls by the child are not listed in the number after the '/'.

name This is the name of the child. The child's index number is printed after it. If the child is a member of a cycle, the cycle number is printed between the name and the index number.

If there are any cycles (circles) in the call graph, there is an entry for the cycle-as-a-whole. This entry shows who called the cycle (as parents) and the members of the cycle (as children.) The '+' recursive calls entry shows the number of function calls that were internal to the cycle, and the calls entry for each member shows, for that member, how many times it was called from other members of the cycle.

Copyright (C) 2012-2018 Free Software Foundation, Inc.

Copying and distribution of this file, with or without modification, are permitted in any medium without royalty provided the copyright notice and this notice are preserved.

Index by function name

[72] BC(double, double) [3] LaplaceSolver::solve(int, double) [76] LaplaceSolver::~~LaplaceSolver()  
[73] seconds() [2] LaplaceSolver::timeStep(double) [77] Grid::setBCFunc(double (\*)(double,  
[74] LaplaceSolver::initialize() [75] LaplaceSolver::LaplaceSolver(Grid\*) [78] Grid::Grid(int, int)