

Computação de Alto Desempenho COC472 - Trabalho 2

Bruno Dantas de Paiva
DRE: 118048097

May 6, 2021

1 Questão 1

A natureza do código consiste em aproximar a equação de laplace para então resolvê-la utilizando o método da relaxação, que é tipicamente utilizado para obter a solução numérica de equações elípticas. A perfilagem torna-se importante para obter mais informações a respeito do comportamento do código no sistema, deste modo nós podemos entender melhor como o código está performando no sistema, assim, podendo identificar alguns hotstops e modifica-los caso necessário.

2 Como usar o gprof

Para perfilar seu código em C/C++ é necessário seguir os seguintes passos:

- Compile seu código em C/C++ utilizando a flag -pg. Ex: `g++ -pg test.cpp -o test`
- Execute o seu código para que ele gere um arquivo .out.
- Execute o gprof com o executavel e o arquivo.out como parâmetros. Ex: `gprof test gmon.out`

3 Relatório do gprof

Abaixo segue o relatório gerado pelo gprof:

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
84.93	0.22	0.22	100	2.21	2.61	LaplaceSolver::timeStep(double)
15.44	0.26	0.04	24800400	0.00	0.00	SQR(double const&)
0.00	0.26	0.00	2000	0.00	0.00	BC(double, double)
0.00	0.26	0.00	2	0.00	0.00	seconds()
0.00	0.26	0.00	1	0.00	0.00	_GLOBAL__sub_I_ZN4GridC2Eii
0.00	0.26	0.00	1	0.00	0.00	__static_initialization_and_destruction_0(int,
0.00	0.26	0.00	1	0.00	0.00	LaplaceSolver::initialize()
0.00	0.26	0.00	1	0.00	260.97	LaplaceSolver::solve(int, double)
0.00	0.26	0.00	1	0.00	0.00	LaplaceSolver::LaplaceSolver(Grid*)
0.00	0.26	0.00	1	0.00	0.00	LaplaceSolver::~~LaplaceSolver()
0.00	0.26	0.00	1	0.00	0.00	Grid::setBCFunc(double (*)(double, double))
0.00	0.26	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) 2012-2021 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 2 byte(s) for 3.83% of 0.26 seconds

index	% time	self	children	called	name
<spontaneous>					
[1]	100.0	0.00	0.26		main [1]
		0.00	0.26	1/1	LaplaceSolver::solve(int, double) [3]
		0.00	0.00	2/2	seconds() [12]
		0.00	0.00	1/1	Grid::Grid(int, int) [19]
		0.00	0.00	1/1	Grid::setBCFunc(double (*)(double, double)) [18]
		0.00	0.00	1/1	LaplaceSolver::LaplaceSolver(Grid*) [16]
		0.00	0.00	1/1	LaplaceSolver::~~LaplaceSolver() [17]

		0.22	0.04	100/100	LaplaceSolver::solve(int, double) [3]
[2]	100.0	0.22	0.04	100	LaplaceSolver::timeStep(double) [2]
		0.04	0.00	24800400/24800400	SQR(double const&) [4]

		0.00	0.26	1/1	main [1]
[3]	100.0	0.00	0.26	1	LaplaceSolver::solve(int, double) [3]
		0.22	0.04	100/100	LaplaceSolver::timeStep(double) [2]

		0.04	0.00	24800400/24800400	LaplaceSolver::timeStep(double) [2]
[4]	15.4	0.04	0.00	24800400	SQR(double const&) [4]

		0.00	0.00	2000/2000	Grid::setBCFunc(double (*)(double, double)) [18]
[11]	0.0	0.00	0.00	2000	BC(double, double) [11]

		0.00	0.00	2/2	main [1]
[12]	0.0	0.00	0.00	2	seconds() [12]

		0.00	0.00	1/1	__libc_csu_init [24]
[13]	0.0	0.00	0.00	1	_GLOBAL__sub_I_ZN4GridC2Eii [13]
		0.00	0.00	1/1	__static_initialization_and_destruction_0(int, int)

		0.00	0.00	1/1	_GLOBAL__sub_I_ZN4GridC2Eii [13]
[14]	0.0	0.00	0.00	1	__static_initialization_and_destruction_0(int, int) [14]

		0.00	0.00	1/1	LaplaceSolver::LaplaceSolver(Grid*) [16]
[15]	0.0	0.00	0.00	1	LaplaceSolver::initialize() [15]

		0.00	0.00	1/1	main [1]
[16]	0.0	0.00	0.00	1	LaplaceSolver::LaplaceSolver(Grid*) [16]
		0.00	0.00	1/1	LaplaceSolver::initialize() [15]

		0.00	0.00	1/1	main [1]

[17]	0.0	0.00	0.00	1	LaplaceSolver::~LaplaceSolver() [17]

		0.00	0.00	1/1	main [1]
[18]	0.0	0.00	0.00	1	Grid::setBCFunc(double (*)(double, double)) [18]
		0.00	0.00	2000/2000	BC(double, double) [11]

		0.00	0.00	1/1	main [1]
[19]	0.0	0.00	0.00	1	Grid::Grid(int, int) [19]

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) 2012-2021 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

[13] _GLOBAL__sub_I_ZN4GridC2Eii (laplace.cxx) [12] seconds() [16] LaplaceSolver::LaplaceSolver(Gr
[11] BC(double, double) [15] LaplaceSolver::initialize() [17] LaplaceSolver::~~LaplaceSolver()

```
[4] SQR(double const&) [3] LaplaceSolver::solve(int, double) [18] Grid::setBCFunc(double (*)(double))
[14] __static_initialization_and_destruction_0(int, int) (laplace.cxx) [2] LaplaceSolver::timeStep()
```

Como é possível observar no Flat profile, o tempo de execução do código foi de 0.28s, onde os hotspots são: a função `timeStep` da classe `LaplaceSolver` (primeira linha do Flat profile) e a função `SQR` (segunda linha do Flat profile).

Quanto a questão de seguir as boas práticas, podemos dizer que alguns cálculos poderiam ser armazenados previamente em registradores e algumas funções poderiam ser removidas, deste modo, o código teria um tempo de execução menor.

4 Alterações Realizadas

Dentre as alterações possíveis, duas foram escolhidas.

4.1 Remoção da função SQR

A função `SQR`, como é possível olhar no relatório do `gprof`, possui um número de chamadas maior que 10 milhões, deste modo, é possível pensar em uma alternativa para esta solução.

Observando melhor, é possível analisar que esta função poderia ser simplesmente removida, pois na verdade ela é uma simples multiplicação. Então, é possível utilizar a técnica `In-lining` para escrever explicitamente esta função onde ela é chamada, deste modo evitando múltiplas chamadas desnecessárias desta função.

4.2 Alteração na função timeStep

A função `timeStep`, como é possível olhar no relatório do `gprof`, possui um número de chamadas menor que alguns outros métodos, contudo ela ocupa uma maior porcentagem do tempo deste código deste modo, é possível pensar em uma forma de alterar o código, visando essa diminuição do tempo de execução. Observando um pouco melhor a função, vemos que o cálculo do numerador da linha 131 - 133, é realizado de uma forma não otimizada, pois ele executa uma multiplicação por 0.5 a cada iteração da matriz e também a divisão por uma constante.

Observando que o denominador não é sendo alterado conforme o loop, foram criadas algumas variáveis, gerando uma função `timeStep` Refatorada.

5 Conclusão

Para observar as alterações realizadas, o código foi compilado novamente e gerado um novo arquivo do gprof, como é possível observar abaixo:

Flat profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
100.38	0.20	0.20	100	2.01	2.01	LaplaceSolver::timeStep(double)
0.00	0.20	0.00	2000	0.00	0.00	BC(double, double)
0.00	0.20	0.00	2	0.00	0.00	seconds()
0.00	0.20	0.00	1	0.00	0.00	_GLOBAL__sub_I_ZN4GridC2Eii
0.00	0.20	0.00	1	0.00	0.00	__static_initialization_and_destruction_0(int,
0.00	0.20	0.00	1	0.00	0.00	LaplaceSolver::initialize()
0.00	0.20	0.00	1	0.00	200.76	LaplaceSolver::solve(int, double)
0.00	0.20	0.00	1	0.00	0.00	LaplaceSolver::LaplaceSolver(Grid*)
0.00	0.20	0.00	1	0.00	0.00	LaplaceSolver::~~LaplaceSolver()
0.00	0.20	0.00	1	0.00	0.00	Grid::setBCFunc(double (*)(double, double))
0.00	0.20	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) 2012-2021 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 2 byte(s) for 4.98% of 0.20 seconds

index	% time	self	children	called	name
<spontaneous>					
[1]	100.0	0.00	0.20		main [1]
		0.00	0.20	1/1	LaplaceSolver::solve(int, double) [3]
		0.00	0.00	2/2	seconds() [11]
		0.00	0.00	1/1	Grid::Grid(int, int) [18]
		0.00	0.00	1/1	Grid::setBCFunc(double (*)(double, double)) [17]
		0.00	0.00	1/1	LaplaceSolver::LaplaceSolver(Grid*) [15]
		0.00	0.00	1/1	LaplaceSolver::~~LaplaceSolver() [16]

		0.20	0.00	100/100	LaplaceSolver::solve(int, double) [3]
[2]	100.0	0.20	0.00	100	LaplaceSolver::timeStep(double) [2]

		0.00	0.20	1/1	main [1]
[3]	100.0	0.00	0.20	1	LaplaceSolver::solve(int, double) [3]
		0.20	0.00	100/100	LaplaceSolver::timeStep(double) [2]

		0.00	0.00	2000/2000	Grid::setBCFunc(double (*)(double, double)) [17]
[10]	0.0	0.00	0.00	2000	BC(double, double) [10]

		0.00	0.00	2/2	main [1]
[11]	0.0	0.00	0.00	2	seconds() [11]

		0.00	0.00	1/1	__libc_csu_init [23]
[12]	0.0	0.00	0.00	1	_GLOBAL__sub_I_ZN4GridC2Eii [12]
		0.00	0.00	1/1	__static_initialization_and_destruction_0(int, int)

		0.00	0.00	1/1	_GLOBAL__sub_I_ZN4GridC2Eii [12]
[13]	0.0	0.00	0.00	1	__static_initialization_and_destruction_0(int, int) [13]

		0.00	0.00	1/1	LaplaceSolver::LaplaceSolver(Grid*) [15]
[14]	0.0	0.00	0.00	1	LaplaceSolver::initialize() [14]

		0.00	0.00	1/1	main [1]
[15]	0.0	0.00	0.00	1	LaplaceSolver::LaplaceSolver(Grid*) [15]
		0.00	0.00	1/1	LaplaceSolver::initialize() [14]

		0.00	0.00	1/1	main [1]
[16]	0.0	0.00	0.00	1	LaplaceSolver::~~LaplaceSolver() [16]

		0.00	0.00	1/1	main [1]
[17]	0.0	0.00	0.00	1	Grid::setBCFunc(double (*)(double, double)) [17]

		0.00	0.00	2000/2000	BC(double, double) [10]

		0.00	0.00	1/1	main [1]
[18]	0.0	0.00	0.00	1	Grid::Grid(int, int) [18]

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) 2012-2021 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

[12] _GLOBAL__sub_I_ZN4GridC2Eii (laplace.cxx) [14] LaplaceSolver::initialize() [16] LaplaceSolver
[10] BC(double, double) [3] LaplaceSolver::solve(int, double) [17] Grid::setBCFunc(double (*)(
[13] __static_initialization_and_destruction_0(int, int) (laplace.cxx) [2] LaplaceSolver::timeStep(
[11] seconds() [15] LaplaceSolver::LaplaceSolver(Grid*)

Com este novo arquivo, é possível observar que as alterações realizadas tiveram um resultado satisfatório, onde foi reduzido o tempo total de execução do código e o número de chamadas da função SQR foram reduzidos a 0, devido a remoção deste trecho de código e a alteração da função timeStep.

Além disso, foi possível observar a importância do gprof como profilador, devido ao grande número de informações fornecida a respeito de cada código, facilitando encontrar os hotspots presentes do programa.

6 Códigos

6.1 SQR

```
inline Real SQR(const Real &x)
{
    return (x * x);
}
```

6.2 timeStep

```
Real LaplaceSolver ::timeStep(const Real dt)
{
    Real dx2 = g->dx * g->dx;
    Real dy2 = g->dy * g->dy;
    Real tmp;
    Real err = 0.0;
    int nx = g->nx;
    int ny = g->ny;
    Real **u = g->u;

    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 / (dx2 + dy2);
            err += SQR(u[i][j] - tmp);
        }
    }
    return sqrt(err);
}
```

6.3 timeStep Refatorada

```
Real LaplaceSolver ::timeStep(const Real dt)
{
    Real dx2 = g->dx * g->dx;
    Real dy2 = g->dy * g->dy;
    Real summation = dx2 + dy2;
    Real multiplier = 1 / (summation + summation);
    Real tmp;
    Real partial_result;
    Real err = 0.0;
    int nx = g->nx;
    int ny = g->ny;
    Real **u = g->u;

    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) *
                       multiplier;
            partial_result = u[i][j] - tmp;
            err += partial_result * partial_result;
        }
    }
    return sqrt(err);
}
```