Computação de Alto Desempenho

Felipe Schreiber Fernandes

May 5 2021

1 Introdução

A equação que o programe busca a solucionar é uma equação diferencial parcial elípitica:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0 \tag{1}$$

Ela modela uma gama de problemas nos quais conhecemos os valores de uma função na borda e queremos avaliar o seu interior. Para resolvê-la do ponto de vista numérico, uma etapa crucial é discretizá-la, que envolve dividir o interior da região com o auxílio de um grid. Assim, cada o valor em cada ponto do grid pode ser expresso pela influência dos pontos adjacentes (acima, abaixo, a esquerda e direita). Contudo, como não conhecemos o valor dos pontos interiores e muitos pontos interiores são vizinhos entre si, então precisamos chutar um valor inicial e a cada iteração atualizar com um valor mais preciso. A figura abaixo mostra isso. A condição de parada pode ser o número de iterações ou caso a máxima diferença, ou seja considerando todos os pontos do interior, entre atualizações sucessivas seja menor que um certo valor. Esse método é conhecido na literatura como o método de Gauss-Seidel.

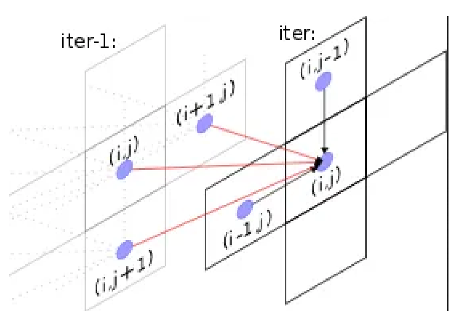

Figure 1: Atualização do valor da função no ponto interior pelo método iterativo de Gauss-Seidel

Como o programa pode levar muitas iterações, de acordo com tão pequeno for o erro desejado e número de pontos no grid, a perfilagem de código é de extrema importância. Assim, podemos analisar quais trechos de código são os pontos críticos do programa no consumo de tempo e recursos computacionais. Com esses insightss, abrimos espaço para melhorias no código. O programa foi feito em C++, logo a ferramenta utilizada para fins de perfilagem foi o gprof. O resto do documento segue a organização: Primeiro é listado o passo a passo para a execução do gprof, em seguida é apresentado o relatório do gprof para o código não otimizado. Após isso uma breve discussão a respeito dos pontos cruciais onde foram feitas as melhorias e, por fim, concluiremos esse trabalho com o relatório do programa otimizado.

2 Executando o gprof

Para executar a ferramenta, uma vez instalada no computador, basta:

1. Gere o executável passando a flag "-pg". Assim, para o nosso caso em que o código está todo contido em um único arquivo .cpp, laplace.cpp, a compilação fica: g++ -pg laplace_improved.cpp -o teste.

2. Execute o programa normalmente. Essa etapa gera um arquivo gmon.out.

3. Execute gprof da seguinte forma: "gprof _nome_executavel_compilado gmon.out". Caso queira salvar o resultado num arquivo, basta fazer "gprof _nome_executavel_compilado_ gmon.out > Log.txt"

3 Relatórios gprof

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total           
 time   seconds   seconds    calls  ms/call  ms/call  name    
 75.72      0.43     0.43      100     4.32     5.62  LaplaceSolver::timeStep(double)
 22.89      0.56     0.13 24800400     0.00     0.00  SQR(double const&)
  1.76      0.57     0.01        2     5.02     5.02  seconds()
  0.00      0.57     0.00     2000     0.00     0.00  BC(double, double)
  0.00      0.57     0.00        1     0.00     0.00  _GLOBAL__sub_I__ZN4GridC2Eii
  0.00      0.57     0.00        1     0.00     0.00  __static_initialization_and_destructio
  0.00      0.57     0.00        1     0.00     0.00  LaplaceSolver::initialize()
  0.00      0.57     0.00        1     0.00   562.10  LaplaceSolver::solve(int, double)
  0.00      0.57     0.00        1     0.00     0.00  LaplaceSolver::LaplaceSolver(Grid*)
  0.00      0.57     0.00        1     0.00     0.00  LaplaceSolver::~LaplaceSolver()
  0.00      0.57     0.00        1     0.00     0.00  Grid::setBCFunc(double (*)(double, dou
  0.00      0.57     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.
```



```
     Call graph (explanation follows)


granularity: each sample hit covers 2 byte(s) for 1.75% of 0.57 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    0.57                 main [1]
                0.00    0.56       1/1           LaplaceSolver::solve(int, double) [3]
                0.01    0.00       2/2           seconds() [5]
                0.00    0.00       1/1           Grid::Grid(int, int) [19]
```

```
                0.00    0.00       1/1           Grid::setBCFunc(double (*)(double, double))
                0.00    0.00       1/1           LaplaceSolver::LaplaceSolver(Grid*) [16]
                0.00    0.00       1/1           LaplaceSolver::~LaplaceSolver() [17]
-----------------------------------------------
                0.43    0.13     100/100         LaplaceSolver::solve(int, double) [3]
[2]     98.2    0.43    0.13     100         LaplaceSolver::timeStep(double) [2]
                0.13    0.00 24800400/24800400     SQR(double const&) [4]
-----------------------------------------------
                0.00    0.56       1/1           main [1]
[3]     98.2    0.00    0.56       1         LaplaceSolver::solve(int, double) [3]
                0.43    0.13     100/100         LaplaceSolver::timeStep(double) [2]
-----------------------------------------------
                0.13    0.00 24800400/24800400     LaplaceSolver::timeStep(double) [2]
[4]     22.8    0.13    0.00 24800400        SQR(double const&) [4]
-----------------------------------------------
                0.01    0.00       2/2           main [1]
[5]      1.8    0.01    0.00       2         seconds() [5]
-----------------------------------------------
                0.00    0.00    2000/2000        Grid::setBCFunc(double (*)(double, double))
[12]     0.0    0.00    0.00    2000         BC(double, double) [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(i
-----------------------------------------------
                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,
-----------------------------------------------
                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) [12]
-----------------------------------------------
                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.

Index by function name

```
[13] _GLOBAL__sub_I__ZN4GridC2Eii [5] seconds()        [16] LaplaceSolver::LaplaceSolver(
[12] BC(double, double)     [15] LaplaceSolver::initialize() [17] LaplaceSolver::~LaplaceS
 [4] SQR(double const&)      [3] LaplaceSolver::solve(int, double) [18] Grid::setBCFunc(do
[14] __static_initialization_and_destruction_0(int, int) [2] LaplaceSolver::timeStep(doubl
```

Nota-se do output gerado pelo gprof que os principais hotspots são a função timeStep e a SQR. Uma análise mais detalhada do código nos leva a ver que ele não obedece aos padrões de boa prática de programação, pois ele chama função no loop interno, por exemplo.

4 Alterações realizadas

Elas consistiram em, principalmente:

1. Transformar a matriz do grid em um vetor. Essa simples mudança por si só já causa um grande impacto pois a memória será alocada de for contígua. Assim, ao executarmos um laço de repetição duplo, como é o caso, o cache poderá armazenar mais endereços da matriz, reduzindo o tempo de acesso futuros.
2. Retirar a função SQR. Vemos que ela é chamada na ordem de milhões de vezes no programa, o que significa que pequenas otmizações podem causar economias enormes no tempo de execução. Uma possibilidade é aplicar a técnica de inlining. Ela consiste em implementar a função por completo nos lugares onde ela era chamada. Consequentemente isso reduz o número de chamadas/overhead de alocar e desalocar espaço na pilha de execução e na pilha de dados do programa.
3. Além dessas duas principais melhorias, outras também foram feitas:
   - Uso do memset, ao invés de um laço iterando sobre os elementos, para zerar o vetor (antiga matriz) alocado.
   - Operações repetitivas, como a subtração de uma constante pelo mesmo valor, dentro do looping foram armazenadas fora em uma variável e seu resultado foi chamado dentro do laço considerado.
   - Uma divisão e multiplicação por 0.5 é feita dentro do laço do duplo. Esse valor, $\frac{0.5}{(dx2+dy2)}$, foi calculado fora do loop e seu valor multiplicado pelos termos correspondentes no interior do laço.

5 Conclusão

Conforme relatório abaixo, constatamos uma redução significativa no tempo de execução. O tempo em ms/chamada da função timeStep que antes era 5.62 agora é 4.07. Por fim, vemos que algumas otimizações, como o blocking, não foram possíveis de serem implementadas por conta da dependência entre as atualizações: a cada nova iteração utilizamos o valor da iteração anterior para aproximar ainda mais ao valor correto. Contudo, se relaxássemos essa dependência, isto é, os valores poderem ser atualizados em ordens um pouco mais arbitrárias, obteríamos um ganho de tempo considerável com o paralelismo.

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 99.20      0.41     0.41      100     4.07     4.07  LaplaceSolver::timeStep(double)
  0.00      0.41     0.00     2000     0.00     0.00  BC(double, double)
  0.00      0.41     0.00        2     0.00     0.00  seconds()
  0.00      0.41     0.00        1     0.00     0.00  _GLOBAL__sub_I__ZN4GridC2Eii
  0.00      0.41     0.00        1     0.00     0.00  __static_initialization_and_destructio
  0.00      0.41     0.00        1     0.00     0.00  LaplaceSolver::initialize()
  0.00      0.41     0.00        1     0.00   406.70  LaplaceSolver::solve(int, double)
  0.00      0.41     0.00        1     0.00     0.00  LaplaceSolver::LaplaceSolver(Grid*)
  0.00      0.41     0.00        1     0.00     0.00  LaplaceSolver::~LaplaceSolver()
  0.00      0.41     0.00        1     0.00     0.00  Grid::setBCFunc(double (*)(double, dou
  0.00      0.41     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) 2012-2020 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 2.46% of 0.41 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    0.41                 main [1]
                0.00    0.41       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))
                0.00    0.00       1/1           LaplaceSolver::LaplaceSolver(Grid*) [15]
                0.00    0.00       1/1           LaplaceSolver::~LaplaceSolver() [16]
-----------------------------------------------
                0.41    0.00     100/100         LaplaceSolver::solve(int, double) [3]
[2]    100.0    0.41    0.00     100         LaplaceSolver::timeStep(double) [2]
-----------------------------------------------
                0.00    0.41       1/1           main [1]
[3]    100.0    0.00    0.41       1         LaplaceSolver::solve(int, double) [3]
                0.41    0.00     100/100         LaplaceSolver::timeStep(double) [2]
-----------------------------------------------
                0.00    0.00    2000/2000        Grid::setBCFunc(double (*)(double, double))
[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(i
-----------------------------------------------
                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,
-----------------------------------------------
                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.

Index by function name

```
  [12] _GLOBAL__sub_I__ZN4GridC2Eii [14] LaplaceSolver::initialize() [16] LaplaceSolver::~La
  [10] BC(double, double)      [3] LaplaceSolver::solve(int, double) [17] Grid::setBCFunc(do
  [13] __static_initialization_and_destruction_0(int, int) [2] LaplaceSolver::timeStep(doubl
  [11] seconds()              [15] LaplaceSolver::LaplaceSolver(Grid*)
```