COC472 - Computação de Alto Desempenho

Trabalho 2

João Pedro de Lacerda

02/09/2021

1 Introdução

Este trabalho consiste na execução do código `./src/laplace.cpp` e na perfilagem do código.

O código é um algoritmo para resolução da Equação de Laplace utilizando o método das diferenças finitas.

Sobre a perfilagem, é um processo para entender o comportamento e performance de um código. Esse processo permite encontrar pontos onde podemos otimizar o nosso código, esses pontos são chamados de *hot spots*.

2 Utilizando gprof para perfilar

Para perfilar o código, podemos executar comando

```
$ make profile
```

ou manualmente

```
$ g++ -g -pg [SOURCE_FILE] -o [OUTPUT_FILE]
# Ao executar o código, um arquivo gmon.out será criado
$ [OUTPUT_FILE]
# O perfil então pode ser criado pelo gprof
$ gprof [BIN_FILE] gmon.out > profile.txt
```

3 Relatório inicial

3.1 Relatório

O relatório gerado pelo gprof foi o seguinte:

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 89.21      0.24     0.24      100     2.41     2.71  LaplaceSolver::timeStep(double)
 11.15      0.27     0.03 24800400     0.00     0.00  SQR(double const&)
  0.00      0.27     0.00     2000     0.00     0.00  BC(double, double)
  0.00      0.27     0.00        2     0.00     0.00  seconds()
  0.00      0.27     0.00        1     0.00     0.00  _GLOBAL__sub_I__ZN4GridC2Eii
  0.00      0.27     0.00        1     0.00     0.00  __static_initialization_and_destruct
  0.00      0.27     0.00        1     0.00     0.00  LaplaceSolver::initialize()
  0.00      0.27     0.00        1     0.00   270.98  LaplaceSolver::solve(int, double)
  0.00      0.27     0.00        1     0.00     0.00  LaplaceSolver::LaplaceSolver(Grid*)
  0.00      0.27     0.00        1     0.00     0.00  LaplaceSolver::~LaplaceSolver()
  0.00      0.27     0.00        1     0.00     0.00  Grid::setBCFunc(double (*)(double, d
  0.00      0.27     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) 2012-2021 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 3.69% of 0.27 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00    0.27                 main [1]
                0.00    0.27       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
                0.00    0.00       1/1           LaplaceSolver::LaplaceSolver(Grid*) [16]
                0.00    0.00       1/1           LaplaceSolver::~LaplaceSolver() [17]
-----------------------------------------------
                0.24    0.03     100/100         LaplaceSolver::solve(int, double) [3]
[2]    100.0    0.24    0.03     100         LaplaceSolver::timeStep(double) [2]
                0.03    0.00 24800400/24800400     SQR(double const&) [4]
-----------------------------------------------
                0.00    0.27       1/1           main [1]
[3]    100.0    0.00    0.27       1         LaplaceSolver::solve(int, double) [3]
                0.24    0.03     100/100         LaplaceSolver::timeStep(double) [2]
-----------------------------------------------
                0.03    0.00 24800400/24800400     LaplaceSolver::timeStep(double) [2]
[4]     11.1    0.03    0.00 24800400         SQR(double const&) [4]
-----------------------------------------------
                0.00    0.00    2000/2000        Grid::setBCFunc(double (*)(double, double
[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
-----------------------------------------------
                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)) [
                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.
```

```
Index by function name

  [13] _GLOBAL__sub_I__ZN4GridC2Eii (laplace.cpp) [12] seconds() [16] LaplaceSolver::Lapla
  [11] BC(double, double)     [15] LaplaceSolver::initialize() [17] LaplaceSolver::~Laplac
   [4] SQR(double const&)      [3] LaplaceSolver::solve(int, double) [18] Grid::setBCFunc(
  [14] __static_initialization_and_destruction_0(int, int) (laplace.cpp) [2] LaplaceSolver
```

3.2 Análise

Segundo o flat profile, o tempo de execução é de 0.29s e conseguimos perceber dois hotspots:

- função 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);
}
```

- função SQR

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

Analisando o código em `src/laplace.cpp`, é possível perceber que há espaço para melhorias de desempenho.

4 Otimizando o código

4.1 Função timeStep

Apesar de ser uma função com poucas chamadas, a função timeStep é responsável por boa parte do tempo de execução. Essa função pode ser otimizada na maneira que executa algumas operações dentro do loop, que são uma constantes dentro dos loops aninhados.

Essa função também é responsável pela chamada da função `SQR`, que será analisada no próximo item.

4.2 Função SQR

A função SQR, por ser uma simples multiplicação, pode facilmente ser removida. Ao invés de realizar as chamadas à função SQR, pode ser feita a multiplicação utilizando o operador `*`.

4.3 Resultado das otimizações

Após as alterações, a função `timeStamp` (que era a única que dependente da função `SQR`) ficou assim:

```
Real LaplaceSolver ::timeStep(const Real dt) {
  Real dx2 = g->dx * g->dx;
  Real dy2 = g->dy * g->dy;
  Real tmp;
  Real err = 0.0;
  Real sum = dx2 + dy2;
  Real mult = 1 / sum;
  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) *
mult;
      Real diff = u[i][j] - tmp;
      err += diff * diff;
    }
  }
  return sqrt(err);
}
```

5 Conclusão

O relatório gerado pelo gprof após as otimizações é o que segue:

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
100.37      0.19     0.19      100     1.91     1.91  LaplaceSolver::timeStep(double)
  0.00      0.19     0.00     2000     0.00     0.00  BC(double, double)
  0.00      0.19     0.00        2     0.00     0.00  seconds()
  0.00      0.19     0.00        1     0.00     0.00  _GLOBAL__sub_I__ZN4GridC2Eii
  0.00      0.19     0.00        1     0.00     0.00  __static_initialization_and_destruct
  0.00      0.19     0.00        1     0.00     0.00  LaplaceSolver::initialize()
  0.00      0.19     0.00        1     0.00   190.71  LaplaceSolver::solve(int, double)
  0.00      0.19     0.00        1     0.00     0.00  LaplaceSolver::LaplaceSolver(Grid*)
  0.00      0.19     0.00        1     0.00     0.00  LaplaceSolver::~LaplaceSolver()
  0.00      0.19     0.00        1     0.00     0.00  Grid::setBCFunc(double (*)(double, d
  0.00      0.19     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) 2012-2021 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 5.24% of 0.19 seconds

index % time    self  children    called     name
                                                 <spontaneous>
```

```
[1]    100.0    0.00    0.19                  main [1]
                0.00    0.19       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]
-----------------------------------------------
                0.19    0.00     100/100          LaplaceSolver::solve(int, double) [3]
[2]    100.0    0.19    0.00     100          LaplaceSolver::timeStep(double) [2]
-----------------------------------------------
                0.00    0.19       1/1            main [1]
[3]    100.0    0.00    0.19       1          LaplaceSolver::solve(int, double) [3]
                0.19    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
-----------------------------------------------
                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)) [
                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 (refactor_laplace.cpp) [14] LaplaceSolver::initialize(
  [10] BC(double, double)      [3] LaplaceSolver::solve(int, double) [17] Grid::setBCFunc(
  [13] __static_initialization_and_destruction_0(int, int) (refactor_laplace.cpp) [2] Lapl
  [11] seconds()              [15] LaplaceSolver::LaplaceSolver(Grid*)
```

Podemos notar que houve uma melhora no número de chamadas de funções, vista a remoção da função SQR, e uma redução de 0.05s no tempo gasto pela função `timeStep`.

Apesar de parecer uma redução pequena, é importante lembrar que quanto maior a escala dos problemas, mais importante são essas otimizações.