ACM/CS 114
Parallel algorithms for scientific applications

Michael A. G. Aïvázis

California Institute of Technology

Winter 2010

Sequential implementation - user interface

```
// main program
int main(int argc, char* argv[]) {
   // default values for our user configurable settings
   size_t N = 10;
   double tolerance = 1.0e-6;
   const char* filename = "laplace.csv";

   // read the command line
   int command;
   while ((command = getopt(argc, argv, "N:e:o:")) != -1) {
      switch (command) {
      // get the convergence tolerance
      case 'e':
         tolerance = atof(optarg);
         break;
      // get the grid size
      case 'N':
         N = (size_t) atof(optarg);
         break;
      // get the name of the output file
      case 'o':
         filename = optarg;
      }
   }
```

Sequential implementation - driving the solver

```
    // allocate space for the solution
    Grid potential(N);

    // initialize and apply our boundary conditions
    initialize(potential);

    // call the solver
    laplace(potential, tolerance);

    // open a stream to hold the answer
    std::fstream output(filename, std::ios_base::out);

    // build a visualizer and render the solution in our chosen format
    Visualizer visualizer;
    visualizer.csv(potential, output);

    // all done
    return 0;
}
```

Sequential implementation - the preamble

- back up to the beginning of the file

```
#include <getopt.h>
#include <cmath>
#include <cstdlib>
#include <fstream>
#include <iostream>

// forward declarations
class Grid;
class Visualizer;

// the solver; does nothing for the time being
void initialize(Grid & grid) {};
void laplace(Grid & grid, double tolerance){};
```

- we have separated out *visualization* in a different object to support different formats without disturbing the data representation
- `initialize` and `laplace` have trivial implementations for now
  - enables testing the scaffolding without worrying about the solver implementation just yet

Sequential implementation - the grid object stub

```
// the solution representation
class Grid {
   // interface: TBD
public:

   // meta methods
public:
   Grid(size_t size);
   ~Grid();

   // private data members: TBD
private:

   // disabled interface
   // grid will own dynamic memory, so don't let the compiler screw up
private:
   Grid(const Grid &);
   const Grid & operator= (const Grid &);
};

// the grid implementation
Grid::Grid(size_t size) {
}

Grid::~Grid() {
}
```

Sequential implementation - the visualizer stub

```
// the visualizer class
class Visualizer {
   // local type aliases
public:
   typedef std::ostream stream_t;

   // interface
public:
   void csv(const Grid & grid, stream_t & stream);

   // meta methods
public:
   inline Visualizer() {}
};

// the Visualizer class implementation
void Visualizer::csv(const Grid & grid, Visualizer::stream_t & stream) {
   return;
}
```

- the code now compiles and links
  - consistency check that the object collaborations are ok, for now
  - can be tested for command line option parsing

Fleshing out the initializer

```
// the grid initializer:
// clear the grid contents and apply our boundary conditions
void initialize(Grid & grid) {
   // ask the grid to clear its memory
   grid.clear(1.0);
   // apply the dirichlet conditions
   for (size_t cell=0; cell < grid.size(); cell++) {
      // evaluate sin(pi x)
      double sin = std::sin(cell * grid.delta() * pi);
      // along the x axis, at top and bottom
      grid(cell, 0) = sin;
      grid(cell, grid.size()-1) = sin * std::exp(-pi);
      // along the y axis, left and right
      grid(0, cell) = 0.0;
      grid(grid.size()-1, cell) = 0.0;
   }

   return;
}
```

- the grid knows its size, its spacing δ, and can initialize out its memory
- access to grid elements happens through an overloaded `operator()` so we can *encapsulate* the indexing function

The grid class declaration

```
// the solution representation
class Grid {
   // interface
public:
   // set all cells to the specified value
   void clear(double value=0.0);
   // the grid dimensions
   size_t size() const {return _size;}
   // the grid spacing
   double delta() const {return _delta;}
   // access to the cells
   double & operator()(size_t i, size_t j) {return _block[j*_size+i];}
   double operator()(size_t i, size_t j) const {return _block[j*_size+i];}
   // meta methods
public:
   Grid(size_t size);
   ~Grid();
   // data members
private:
   const size_t _size;
   const double _delta;
   double* _block;
   // disable these
private:
   Grid(const Grid &);
   const Grid & operator= (const Grid &);
};
```

The grid class implementation

```
// the grid implementation
// interface
void Grid::clear(double value) {
   for (size_t i=0; i < _size*_size; i++) {
      _block[i] = value;
   }

   return;
}

// constructor
Grid::Grid(size_t size) :
   _size(size),
   _delta((1.0 - 0.0)/(size-1)),
   _block(new double[size*size]) {
}

// destructor
Grid::~Grid() {
   delete [] _block;
}
```

Grid visualization

```
// the visualizer class
class Visualizer {
   // local type aliases
public:
   typedef std::ostream stream_t;
   // interface
public:
   void csv(const Grid & grid, stream_t & stream);
   // meta methods
public:
   inline Visualizer() {}
};

// the Visualizer class implementation
void Visualizer::csv(const Grid & grid, Visualizer::stream_t & stream) {
   for (size_t j=0; j < grid.size(); j++) {
      stream << j;
      for (size_t i=0; i < grid.size(); i++) {
         stream << "," << grid(i,j);
      }
      stream << std::endl;
   }

   return;
}
```

Printing out the initial grid

- we should be able to print out the initialized grid

```
#> mm laplace
#> laplace
#> cat laplace.csv
0,0,0.3827,0.7071,0.9239,1,0.9239,0.7071,0.3827,1.225e-16
1,0,1,1,1,1,1,1,1,0
2,0,1,1,1,1,1,1,1,0
3,0,1,1,1,1,1,1,1,0
4,0,1,1,1,1,1,1,1,0
5,0,1,1,1,1,1,1,1,0
6,0,1,1,1,1,1,1,1,0
7,0,1,1,1,1,1,1,1,0
8,0,0.01654,0.0306,0.03992,0.04321,0.0399,0.03056,0.01654,0
```

- notice that
  - the top line contains some recognizable values
  - the left and right borders are set to zero
  - the interior of the grid is painted with our initial guess
- still to do:
  - write the update
  - build a grid with the exact solution
  - build the error field (why?)