Structured Grid Calculations

- Numerical solution represented on a hierarchy of nested rectangular arrays.

- Three kinds of computational operations:
  - Local stencil operations on rectangles. Explicit methods, iterative solvers.
  - Irregular computation on boundaries.
  - Copying between rectangles.
Algorithmic Characteristics

- $O(1)$ flops per memory access (10 - 100’s).
- **Codimension** one irregularity.
- Multiphysics complexity: many different operators acting in sequence on the same collection of state variables (the operators may contain state for performance reasons). *Not OOP.*
- Irregular computation combined with irregular communication.
Simplest Case: Laplacian on a Rectangle

\[ \phi : [0, 1] \times [0, 1] \to \mathbb{R} \]

\[ \Delta \phi \equiv \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} \]

Discretize using finite differences.

\[ \phi^h : \Omega^h \to \mathbb{R}, \Omega^h = [0, \ldots, N] \times [0, \ldots, N] \]

\[ h = \frac{1}{N}, \phi^h_{i,j} \approx \phi(ih, jh) \]

\[ (\Delta^h \phi^h)_{i,j} \equiv \frac{1}{h^2} (\phi^h_{i+1,j} + \phi^h_{i-1,j} + \phi^h_{i,j+1} + \phi^h_{i,j-1} - 4\phi^h_{i,j}) \]

\[ \Delta^h \phi^h : \Omega^h_0 \to \mathbb{R}, \Omega^h_0 = [1, \ldots, N - 1] \times [1, \ldots, N - 1] \]

• Regular data access – lots of stride 1 access if you set it up correctly. Computing locations of data simple.

• Mathematical properties of these discretizations well-understood.
Arrays Over a Single Rectangular Grid.

C++ does not have multidimensional arrays with run-time specification of dimensions. Want to build a C++ class to provide us with that feature specifically for use in rectangular-grid discretizations of PDE.

- Make memory management automatic through suitable definition of constructors / destructors.
- Modularity / separation of concerns.
- Anticipate generalization to unions of rectangles.
- Enable implementations that are independent of dimension, in the sense that it appears as a compile-time parameter.
Representing Data on a Rectangle

template <class T, unsigned int C=1, ...>
    class BoxData
{

public:

    /// bunch of member functions here

private:

    Box m_box; // a separate class to represent range
            // of indices over which array is defined.
    T* m_rawPtr; // contiguous block of data.

}
Representing Data on a Rectangle

Class Box
{
public:
    /// bunch of member functions here
private:
    Point m_lowCorner; // low corner (e.g. (0,0)
    Point m_highCorner; // low corner (e.g. (N,N)
Representing Data on a Rectangle

Class Point
{
    public:
    /* a bunch of member functions here */

    private:

    int m_tuple[DIM]; // integer tuple.

    /* DIM is a compile-time constant, given by the dimensionality of the space. */
Why make a separate Classes (Point, Box) to represent the range of indices over which the array is defined?

- Facilitates writing dimension-independent code.
- Operations occur on subsets of the grid: applying the operator, boundary conditions. These subsets can be computed using member functions of Box (set calculus).
- Building block for defining data on unions of rectangles.
#include <stream>
int main(int argc, char* argv[]) {
Point lowCorner, highCorner;
cout << "input size in each direction (" << DIM <<" ints)" << endl;
for (int i = 0; i < DIM; i++)
{
    int length;
    cin >> length;
    highCorner[i] = size-1;
    lowCorner[i] = 0;
}

double h = 1./(hi[0]);
Box D(lowCorner, highCorner);
Box D0 = D.grow(-1);
BoxData<double> phi(D);
BoxData<double> LOfPhi(D0);

Grid size coming in from terminal input
Define mesh spacing.
Define $\Omega^h$
Define $\Omega^h_0$ by shrinking $\Omega^h$
Define $\phi^h$ as an array on $\Omega^h$
Define data holder for $\Delta^h \phi^h$
A walk through mdarrayMain.cpp

for (Point p = D.lowCorner(); !(p == D.highCorner()); D.increment(p)) {
    double val = 1.;
    for (int dir = 0; dir < DIM; dir++) {
        val = val * sin(2*M_PI*p[dir]*h);
    }
    phi(p) = val;
}
Iterators

BoxIterator bi(D0);
for (bi.begin(); !bi.Done(); ++bi)
{
    LOfPhi(*bi) = ...;
}
...
Class BoxIterator
{
    public:
    BoxIterator(const Box& a_box);
    void begin();
    bool done() const;
    void operator++();
    const Point& operator*();

    By storing information in the data members of the iterator can reduce the amount of integer computation used to index into LOfPhi. What about indexing into arrays with other boxes?
A walk through mdarrayMain.cpp

for (Point p=D.lowCorner();!(p==D.highCorner());D.increment(p))
for (auto pIter = D.begin(); !pIter.done(); ++pIter)
{
    double val = 1.;
    Point p = *pIter;
    for (int dir = 0; dir < DIM; dir++)
    {
        val = val*sin(2*M_PI*p[dir]*h);
    }
    phi(p) = val;
}

auto — infers the type from the rhs of the assignment. If ambiguous, compile-time error.

Box has a member function that returns the BoxIterator bi.begin(*this).
for (auto p = D0.begin(); p.done(); p++)
{
    LOfPhi(*p) = 0.;
    for (int dir = 0; dir < DIM; dir++)
    {
        LOfPhi(*p) += (phi(*p+Point::Basis(dir)) +
                      + phi(*p-Point::Basis(dir))
    }
    LOfPhi(*p) -= 2*DIM*phi(*p);
    LOfPhi(*p) /= h*h;
    double delta = LOfPhi(p) + 4*DIM*M_PI*M_PI*phi(p);
    error = max(abs(delta), error);
}

cout << "error= " << error << endl;
#include <iostream>
#include <fstream>

iostream provides you access to screen I/O. You just use it.

```cpp
ofstream oFile;
oFile.open(a_string);
for (int k = 0; k < a_dbx0.sizeOf(); k++)
{
    Point pt = a_dbx0.getPoint(k);
    double x = pt[0]*dx + 0.5*dx;
    oFile << x << " ";
    for (int icomp = 0; icomp < NUMCOMPS; icomp++)
    {
        oFile << a_U(pt,icomp) << " ";
    }
    oFile << endl;
}
oFile.close();
```
Clumsy constructions and missing pieces.

We need to recompute a lot of information, rather than just incrementing an integer. Inside of RectMDArray, indexing is done by computing

\[ \text{ind} = p[0] + (m\_box\text{.highCorner()[0]} - m\_box\text{.lowCorner()[0]})*p[1]; \]

\[ \text{return m\_data[ind];} \]

(actually, more complicated if dimension-independent), instead of incrementing by 1 along each row.

This is also a potential performance issue. What order do we traverse the points? Function call overheads? Missed opportunities for optimization?
Clumsy constructions and missing pieces.

Different centerings of Boxes:

Vertex (●), cell (□) and face (⊕ and ⊗) sites on a grid

Do you want to handle these by having them as an attribute of Box, or of BoxData?
Other design choices.

Public copy constructors, assignment operators: yes or no?

• Pros: richer set of operators:

```c++
BoxData<float> A, B, C;
...
A = B + C;
```

• Cons: for large objects, can lead to memory bloat that is not obvious or easily controlled by the user. We will return to this later.
Other design choices.

Strong construction vs. default + define.

// Strong construction
Box B;
Box C(lo, hi);
B = C;

// Default + define.
BoxData<foo> phi(B);
BoxData<foo> psi;
psi.define(C);

Making aggregates:
Box BArray[5];
BArray[0] = B;
BArray[1] = C;
...

Default and define:
BoxData<foo> Phi[5];
Phi[0].define(BArray[0]);
Phi[1].define(BArray[1]);

Strong construction: use an array of pointers and new:
BoxData<foo>* Phi[5];
Phi[0] = new BoxData(BArray[0]);
Phi[1] = new BoxData(BArray[1]);

But then you need to make sure you delete.
Beyond a single rectangle.

What if you wanted to compute on a domain line made up of several rectangles?

Can create aggregate data types for representing the region and the data.
Beyond a single rectangle.

Class BoxLayout

/* Wrapper for a collection of Boxes that form a disjoint covering of some region in space. */

Class BLIterator

/* Iterator over the Boxes in a BoxLayout. */

Class LevelData

/* Wrapper for a collection of BoxDatas, defined over a region given by a DBL. May include ghost cell storage. */
Beyond a single rectangle.

Applying the operator on a union of rectangles.

- Get the additional values you need for your stencil from adjacent BoxDatas.
- Iterate over the BoxDatas in a LevelData, applying each operator separately.
Beyond a single rectangle.

Design Issues:

• You need to find your neighbors to obtain ghost cell values. How do you do that quickly i.e. no worse than $O(\log N)$ per Box?

• Cost of construction and storage of DBL can be significant, if you store one for every LevelData object. How do you minimize that overhead (answer: sharing).

• What do you do about data that is not cell-centered? (answer: DBL consist of Boxes that are cell-centered, associated data overlaps.)
Compiling and running mdArrayMain.cpp

clang++ -DDIM=2 -std=c++11 -g mdArrayMain.cpp -o mdArrayTest.exe

This is doing three things.

• Running the C preprocessor on the .cpp files.
• Translating the .cpp files into object code (compiler).
• Assembling the functions defined in the .o files that are needed, starting with main, into an executable file (linker).

Each of this functions could have been invoked separately, by changing the inputs given to the compiler on the command line.

• You can run the C preprocessor by itself and save the generated output (You won’t have a reason to do that in this class).
• You can run the compiler to generate object files, using the –c flag (“separate compilation”).
  clang++ -c -g -I. -DDIM=2 mdArrayMain.cpp -o mdArrayMain.o
• You can link a collection of files that have already been compiled.
  clang++ -std=c++11 -o mdArrayTest2D.exe ./mdArrayMain.o
C++ Macro define

Where did DIM come from?

At the top of Point.H, I included the following statement.

#define DIM 2

Prior to compilation, the C preprocessor replaces every occurrence of DIM with 2 (or 3 or 4 or whatever you choose).

We can override this in the compile line.
The C proprocessor - #

# ... – indicates a preprocessor command. The preprocessor has limited text
   manipulation capabilities.
#include <iostream>
#include <cmath>
#include "Proto_Point.H"
#include "Proto_Box.H"
#include "Proto_BoxData.H"
using namespace std;
using namespace Proto;

int main(int argc, char* argv[]) {
  ...
}
#include...: take the contents of the file and insert here.
“...” – look for these files in the directories given by -I in the command line.
<...> - the system knows where to find these (C++ std files).
Proto_Point.H

#ifndef _PROTO_POINT_H_
#define _PROTO_POINT_H_
#define DIM 2
#include <iostream>
...
namespace Proto
{
    class Point
    {
        public:
            inline Point();
            inline Point(const int a_tuple[DIM]);
            inline Point(const array<int,DIM> a_tuple);
            inline Point(const Point& a_pt);
            ...
    }
    #endif
Proto_Box.H

ifndef _PROTO_BOX_H___
define _PROTO_BOX_H___
#include "Proto_Point.H"
#include <iostream>
#include <cmath>
...

Class Box
{
Public: ...
}
#endif

Both Proto_Box.H and Proto_Point.H are included in mdArrayMain.cpp. The include guards keep Point.H from being included twice (_PROTO_POINT_H___ is already defined the first time Proto_Point.H is included, so ifndef _PROTO_POINT_H___ is false by the time it gets to the include in Proto_Box.H.

What happens if you don’t do that?
Other C preprocessor commands

#define DIM 3
#define OPTIMIZE
 ifndef OPTIMIZE
 endif
 Also:
 ifndef FOO
 if DIM==3
 if DEBUG
 if 0  
 0 is false, 1 is true. Easy way to temporarily delete / undelete sections of code.
  Definitions can be overwritten in the compile line.
   clang++ -D DIM=2 -D DEBUG=FALSE -U OPTIMIZE -D FOO
  Can also use the preprocessor to define functions. Again, something you will not be doing in this class (the preprocessor knows nothing about types, legal expressions — it is just manipulating text).
  Newline ends a preprocessor command, but can string multiple lines together with \

Doxygen ([www.doxygen.org](http://www.doxygen.org))

- Takes comments in header files and turns them into html and latex documents.
- /// Short description.
- /** ... */ Extended description.
- Knows C++.
- Warning: defaults many not include our file naming conventions e.g. *.

  H for headers.
Doxygen Demo
Other features ...

Grouping related features.

/** @name Constructors */

///<@

///<@

///<@

In DoxyWizard:
Set working directory (include carriage return!)
expert interface:
Adding *.H (Input - FILE_PATTERNS)
Where to look: Input - INPUT
Location of html, latex outputs (HTML, LaTeX) (relative to working dir.)
Translate LaTex in headers: HTML – USE_MATHJAX
Include private data / members: Build – EXTRACT_PRIVATE.
“Make originated with a visit from Steve Johnson (author of yacc, etc.), storming into my office, cursing the Fates that had caused him to waste a morning debugging a correct program (bug had been fixed, file hadn't been compiled, \texttt{cc *.o} was therefore unaffected). As I had spent a part of the previous evening coping with the same disaster on a project I was working on, the idea of a tool to solve it came up. It began with an elaborate idea of a dependency analyzer, boiled down to something much simpler, and turned into \texttt{Make} that weekend. Use of tools that were still wet was part of the culture. Makefiles were text files, not magically encoded binaries, because that was the Unix ethos: printable, debuggable, understandable stuff.”

- Stuart Feldman, inventor of \texttt{Make}.