ACM/CS 114
Parallel algorithms for scientific applications

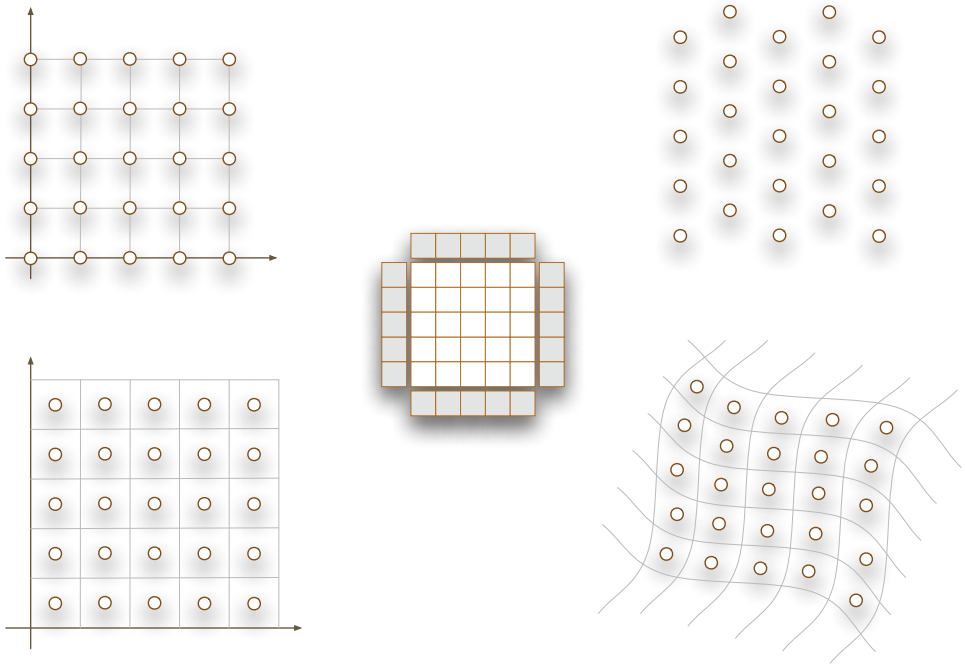
Michael A. G. Aïvázis

California Institute of Technology

Winter 2010

Overview

- lattices: logically Cartesian grids
  - flow, lattice dynamics, wave propagation, image processing
- suitable when it is possible to find an invertible map ϕ from the problem domain Ω to $\mathbb{Z}^d$
- data representation: multi-dimensional arrays
  - ϕ maps points in Ω to loop indices
  - with *guard cells* for enforcing boundary conditions

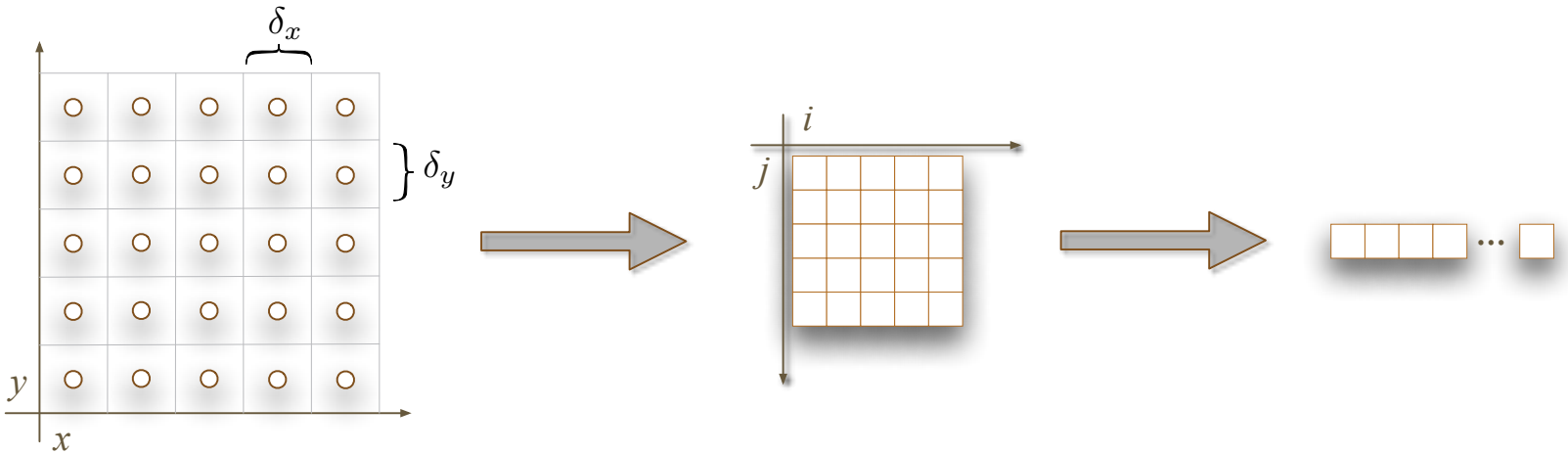


Advantages

- advantages: logically rectangular
  - indexing: easy traversal using loops
  - fixed stride: predictable memory layout
  - topology: finding neighbors is trivial
- disadvantages: many problems don't fit in simple boxes...
  - non-trivial geometries are hard to model
  - the representational simplicity disappears quickly as modeling complexity increases
    - domain feature resolution
    - complex initial and boundary conditions
    - adaptive refinement: allowing the properties of the solution to direct where computational resources are spent

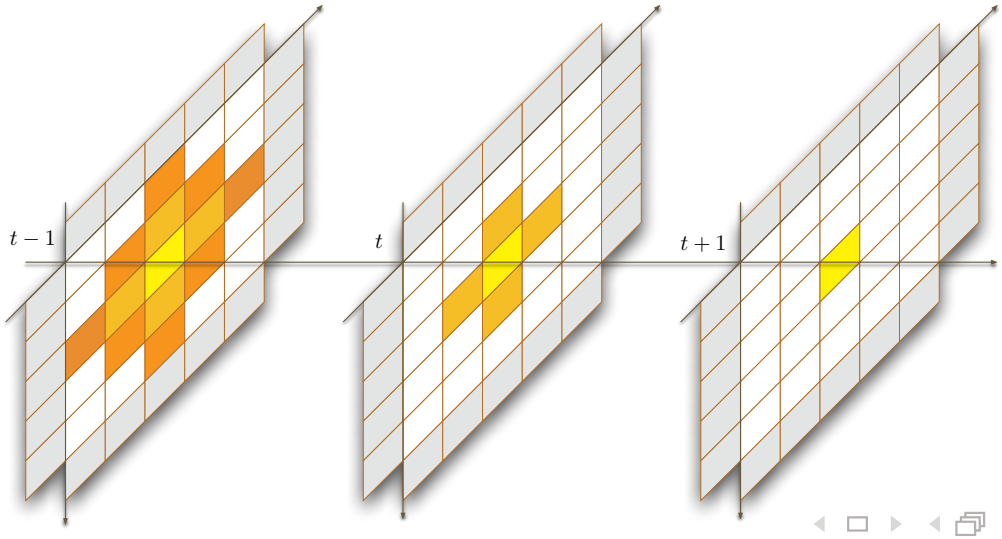
Data layout and performance

- in multi-tier memory architectures
  - good data locality enables efficient cache use
- multi-dimensional arrays: naïve implementations do not perform well
  - for large problem sizes
  - for complex physics updates that require keeping track of multiple fields
- representing scalar, vector and tensor fields
  - optimal layout is problem dependent
  - goal is to minimize cache misses while updating the fields
- the conventional mapping to arrays lays out the data in matrix form
  - not necessarily the most convenient convention
- take ownership of the indexing function



Updating the grid

- two broad categories of problems
  - steady state problems: iterates represent progress towards enforcing a spatial relationship dictated by the differential equation
  - time dependent problems: iterates represent the time evolution of the solution to the spatial problem
- two broad strategies:
  - *implicit* solvers cast the constraints among iterates as a large system of simultaneous equations
  - *explicit* solvers keep track of only a small number of the iterates and use them to advance the solution one step at a time
- the complexity of the update determines the *stencil*



Computing derivatives

- there are three different first order approximations
  - forward difference:

$$\partial \square\blacksquare\square = \frac{1}{\delta}\left(\square\square\blacksquare - \square\blacksquare\square\right) \tag{1}$$

  - backward difference:

$$\partial \square\blacksquare\square = \frac{1}{\delta}\left(\square\blacksquare\square - \blacksquare\square\square\right) \tag{2}$$

  - central difference:

$$\partial \square\blacksquare\square = \frac{1}{2\delta}\left(\square\square\blacksquare - \blacksquare\square\square\right) \tag{3}$$

  where δ is the uniform grid spacing
- forward and backward differences are most often used for *explicit* time integration
- central differences are used to compute spatial derivatives
- the second order central difference is given by

$$\partial \square\square\blacksquare\square\square = \frac{1}{12\delta}\left(-\square\square\square\blacksquare + 8\,\square\square\blacksquare\square - 8\,\square\blacksquare\square\square + \blacksquare\square\square\square\right) \tag{4}$$

Partial derivatives in two dimensions

- let δ_x and δ_y be the uniform grid spacing along each dimension
- then, the first order central difference approximations to the spatial derivatives are given by

$$\partial_x \; = \; \frac{1}{2\delta_x} \left(\; - \; \right) \tag{5}$$

$$\partial_y \; = \; \frac{1}{2\delta_y} \left(\; - \; \right) \tag{6}$$

- and the second order derivatives are given by

$$\partial_{xx} \; = \; \frac{1}{\delta_x^2} \left(\; - 2 \; + \; \right) \tag{7}$$

$$\partial_{yy} \; = \; \frac{1}{\delta_y^2} \left(\; - 2 \; + \; \right) \tag{8}$$

$$\partial_{xy} \; = \; \frac{1}{4\delta_x\delta_y} \left(\; - \; - \; + \; \right) \tag{9}$$

Solving a simple PDE on a uniform structured grid

- Laplace equation over some domain $\Omega \in \mathbb{R}^d$, subject to Dirichlet boundary conditions

$$\nabla^2 \phi = 0 \quad \text{with} \quad \phi(\partial\Omega) = f \tag{10}$$

- let the grid be uniform: $\delta_x = \delta_y$
- in two dimensions, using first order central differences, Eq. 10 becomes

$$(\partial_{xx} + \partial_{yy}) \; \square \quad = \quad 0 \tag{11}$$

$$4 \; \square \quad = \quad \square + \square + \square + \square \tag{12}$$

- and translates into the following constraint among grid elements

$$\square = \frac{1}{4} \; \square \tag{13}$$

using a shorthand for the sum of the neighboring cells

An example

- specifically,
  - let Ω be the unit box in two dimensions
  - and let ϕ satisfy the following boundary conditions

$$\begin{array}{rcccl} & & \phi(x,0) & = & \sin(\pi x) \qquad 0 \leq x \leq 1 \\ & & \phi(x,1) & = & e^{-\pi}\sin(\pi x) \qquad 0 \leq x \leq 1 \\ \phi(0,y) & = & \phi(1,y) & = & 0 \qquad 0 \leq y \leq 1 \end{array} \tag{14}$$

- the exact solution is given by

$$\phi(x,y) = e^{-\pi y}\sin(\pi x) \tag{15}$$

- we will solve this equation using the Jacobi iterative scheme:
  - make an initial guess for ϕ over a discretization of Ω
  - apply the boundary conditions
  - interpret Eq. 13 as an update step to compute the next iteration

$$\boxplus_{t} = \frac{1}{4}\boxplus_{t-1} \tag{16}$$

  - stop when a convergence criterion is met

Implementation strategy

- grid resolution:
  - ideally determined by analyzing the boundary conditions, since discrete sampling may wash out sharp features
  - for our simple example, this can be done as part of the solver initialization
  - we will use an $N \times N$ grid and let N be user specified so we can control the problem size

$$\delta_x = \delta_y = \frac{1}{N-1} \tag{17}$$

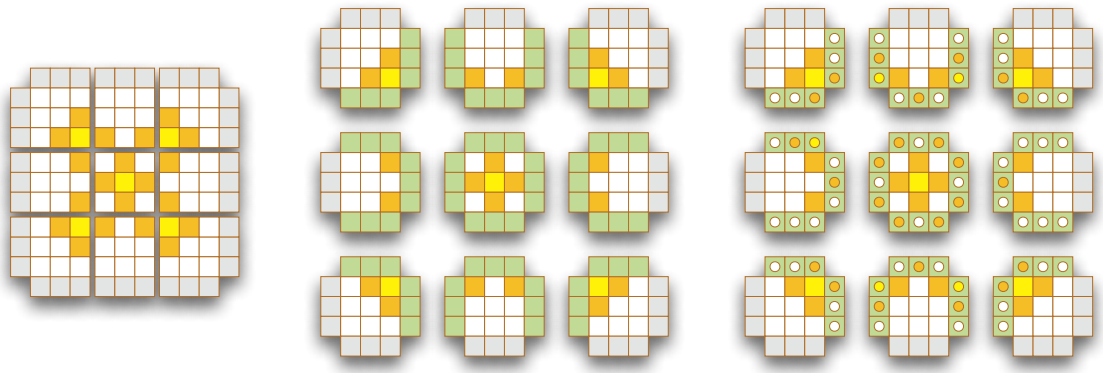
- data layout
  - investigate the effect of data locality by trying out various layouts
- setting up the update
  - we only need to keep track of two iterants
  - can be done in place; do you see how?
- convergence criterion
  - we will stop iterating when

$$\max_{\Omega} \left(\phi_t - \phi_{t-1}\right)) < \epsilon \tag{18}$$

  and let the user specify ϵ

Parallelization

- the finest grain of work is clearly the cell update based on the value of its four nearest neighbors
- the shared memory implementation requires
  - a scheme so that threads can update cells without the need for locks
  - while maximizing locality of data access
  - even the computation of the convergence criterion can be parallelized
- with `MPI`
  - must partition the mesh among processes
  - each process work on its own subgrid
  - communication is required every iteration
  - parallel convergence testing involves a collective operation



Sequential implementation - user interface

```
77  // main program
78  int main(int argc, char* argv[]) {
79     // default values for our user configurable settings
80     size_t N = 10;
81     double tolerance = 1.0e-6;
82     const char* filename = "laplace.csv";
83
84     // read the command line
85     int command;
86     while ((command = getopt(argc, argv, "N:e:o:")) != -1) {
87        switch (command) {
88        // get the convergence tolerance
89        case 'e':
90           tolerance = atof(optarg);
91           break;
92        // get the grid size
93        case 'N':
94           N = (size_t) atof(optarg);
95           break;
96        // get the name of the output file
97        case 'o':
98           filename = optarg;
99        }
100    }
```

Sequential implementation - driving the solver

```
102     // allocate space for the solution
103     Grid potential(N);
104
105     // initialize and apply our boundary conditions
106     initialize(potential);
107
108     // call the solver
109     laplace(potential, tolerance);
110
111     // open a stream to hold the answer
112     std::fstream output(filename, std::ios_base::out);
113
114     // build a visualizer and render the solution in our chosen format
115     Visualizer visualizer;
116     visualizer.csv(potential, output);
117
118     // all done
119     return 0;
120 }
```