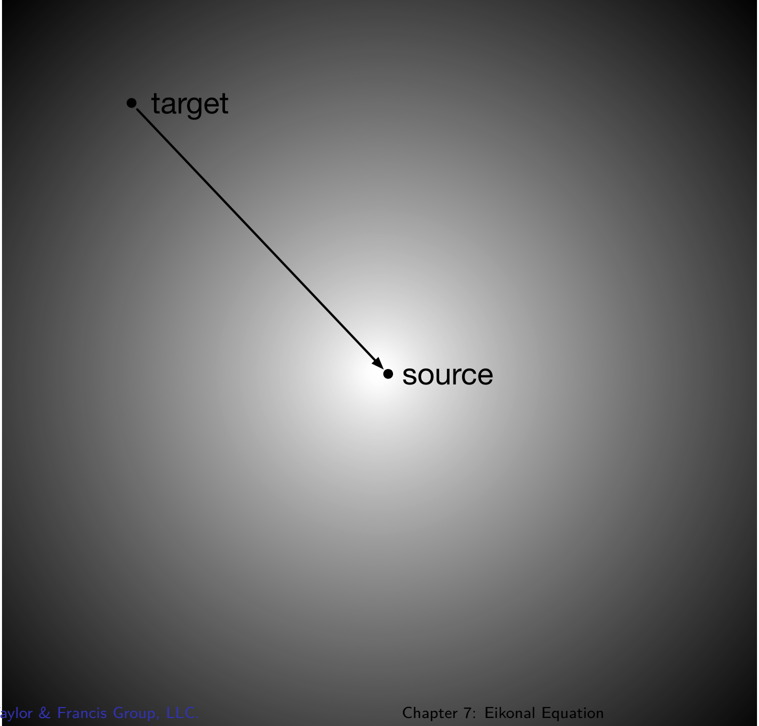
Chapter 7: Eikonal Equation

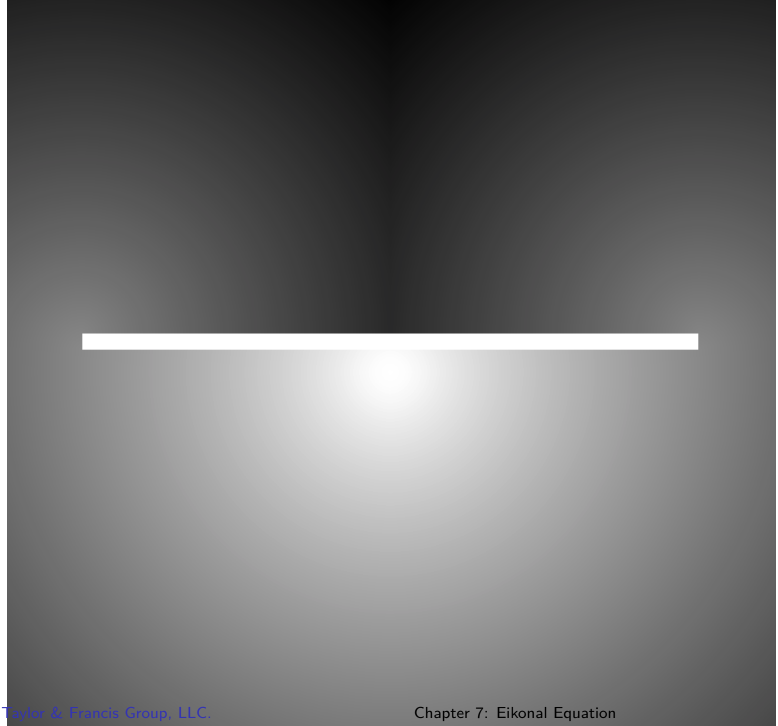
Elements of Parallel Computing

Eric Aubanel

Problem

- Find continuous shortest path in the plane from given source locations and a speed function.
- Moving fronts (wave propagation): stationary problem if front only passes through a give point once


target
source



Eikonal Equation

$$|\nabla U(\mathbf{x})| = \frac{1}{F(\mathbf{x})}, \ \mathbf{x} \in \Re^n,$$

- ∇: gradient, $|\ |$: Euclidean norm
- $F(\mathbf{x})$: positive speed function
- $U(\mathbf{x}) = 0$, $\mathbf{x} \ \in \Gamma \subset \Re^n$: initial position of the front:
- $U(\mathbf{x})$: the arrival time at position $\mathbf{x}$
- We'll stick to 2D

Discretization

- Discretize rectangular domain into $(n_i - 1) \times (n_j - 1)$ squares of length h
- Solve the Eikonal equation at corners of the squares
- $u_{i,j}$ is initialized to ∞, except to 0 at boundary (source of the front)
- Add another layer of points around the edge of the domain, with $u_{0,0..n_j+1} = u_{n_i+1,0..n_j+1} = u_{0..n_i+1,0} = u_{0..n_i+1,n_j+1} = \infty$

Example 7×7 Grid

Boundary at center point

100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100
100	100	100	100	**0**	100	100	100	100
100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100
100	100	100	100	100	100	100	100	100

Numerical Solution: Upwind finite difference

$$\frac{\partial U}{\partial x} \approx \frac{u_{i,j} - u^{imin}}{h}, \; u^{imin} = \min(u_{i-1,j}, u_{i+1,j})$$

Numerical Solution: Discretized Eikonal equation

$$\left((u_{i,j} - u^{imin})^+\right)^2 + \left((u_{i,j} - u^{jmin})^+\right)^2 = \frac{h^2}{f_{i,j}^2},$$

where $f_{i,j}$ is the speed function F evaluated at (i,j), and

$$(x)^+ = \begin{cases} x, & x > 0, \\ 0, & x \leq 0. \end{cases}$$

Solution of Discretized Equation

$$u_{i,j}^{\text{new}} =$$

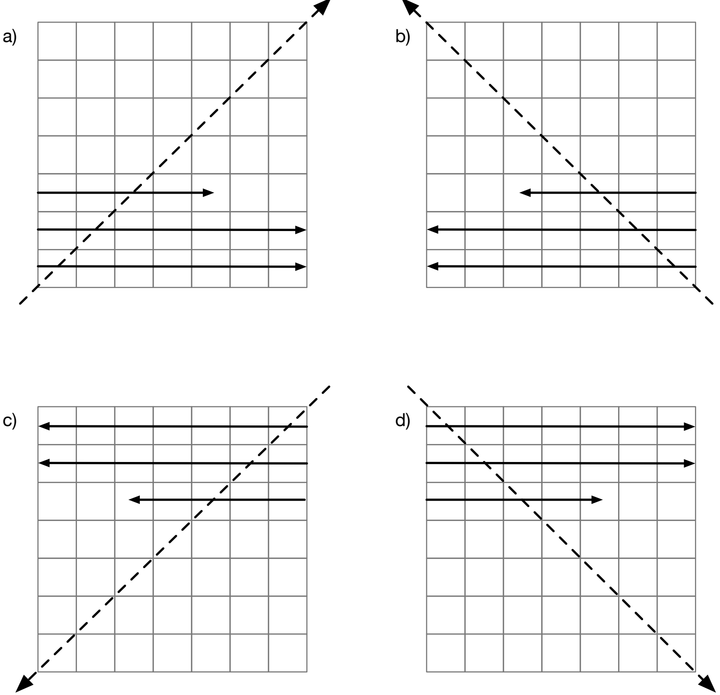
$$\begin{cases} \min(u^{imin}, u^{jmin}) + h/f_{i,j}, & |u^{imin} - u^{jmin}| \geq h/f_{i,j}, \\ \frac{u^{imin} + u^{jmin} + \sqrt{2h^2/f_{i,j}^2 - (u^{imin} - u^{jmin})^2}}{2}, & |u^{imin} - u^{jmin}| < h/f_{i,j}. \end{cases}$$

Fast Sweeping Method

Visit the interior points on the grid left to right, a row at a time, starting from the bottom row, replacing $u_{i,j}$ by $u_{i,j}^{\text{new}}$ at each point if $u_{i,j}^{\text{new}} < u_{i,j}$, using $F = 1$ everywhere and $h = 1$

First Sweep

100	100	**4**	**3**	**3.4422**	**4.0480**	**4.7551**
100	100	**3**	**2**	**2.5453**	**3.2524**	**4.0480**
100	100	**2**	**1**	**1.7071**	**2.5453**	**3.4422**
100	100	**1**	0	**1**	**2**	**3**
100	100	100	**1**	**2**	**3**	**4**
100	100	100	100	100	100	100
100	100	100	100	100	100	100


a)
b)
c)
d)

Second Sweep

4.7551	**4.0480**	**3.4422**	3	3.4422	4.0480	4.7551
4.0480	**3.2524**	**2.5453**	2	2.5453	3.2524	4.0480
3.4422	**2.5453**	**1.7071**	1	1.7071	2.5453	3.4422
3	**2**	1	0	1	2	3
3.7071	**2.7071**	**1.7071**	1	**1.7071**	**2.7071**	**3.7071**
5	**4**	**3**	**2**	**3**	**4**	5
100	100	100	100	100	100	100

Third Sweep

4.7551	4.0480	3.4422	3	3.4422	4.0480	4.7551
4.0480	3.2524	2.5453	2	2.5453	3.2524	4.0480
3.4422	2.5453	1.7071	1	1.7071	2.5453	3.4422
3	2	1	0	1	2	3
3.4422	**2.5453**	1.7071	1	1.7071	**2.5453**	**3.5453**
4.0480	**3.2524**	**2.5453**	2	**2.5453**	**3.4422**	**4.4422**
4.7551	**4.0480**	**3.4422**	**3**	**3.5453**	**4.4422**	**5.4422**

Fourth Sweep

4.7551	4.0480	3.4422	3	3.4422	4.0480	4.7551
4.0480	3.2524	2.5453	2	2.5453	3.2524	4.0480
3.4422	2.5453	1.7071	1	1.7071	2.5453	3.4422
3	2	1	0	1	2	3
3.4422	2.5453	1.7071	1	1.7071	2.5453	**3.4422**
4.0480	3.2524	2.5453	2	2.5453	**3.2524**	**4.0480**
4.7551	4.0480	3.4422	3	**3.4422**	**4.0480**	**4.7551**

2D Fast Sweeping Method

Input: Grid spacing h, $(n_i + 2) \times (n_j + 2)$ speed function matrix F, $(n_i + 2) \times (n_j + 2)$ solution matrix U initialized to large value everywhere except boundary (source of front), where it is initialized to 0.

Output: Solution matrix U

while *not converged* **do**

  `sweep(`U, n_i, 1, 1, n_j, F, h`) // Northeast`

  `sweep(`U, n_i, 1, n_j, 1, F, h`) // Northwest`

  `sweep(`U, 1, n_i, n_j, 1, F, h`) // Southwest`

  `sweep(`U, 1, n_i, 1, n_j, F, h`) // Southeast`

end

```
Procedure sweep(U, i_a, i_b, j_a, j_b, F, h)
    if i_b < i_a then step_i = -1 else step_i = 1
    if j_b < j_a then step_j = -1 else step_j = 1
    for i ← i_a to i_b step step_i do
        for j ← j_a to j_b step step_j do
            u_new ← solveQuadratic(U, i, j, F, h)
            if u_new < U[i, j] then U[i, j] ← u_new
        end
    end
end
```

```
Procedure solveQuadratic(U, i, j, F, h)
    // Don't update boundary points
    if U[i,j] ← 0 then
        return U[i,j]
    end
    a ←min(U[i − 1, j], U[i + 1, j])
    b ←min(U[i, j − 1], U[i, j + 1])
    if |a − b| ≥ h/F[i,j] then
        return min(a, b) +h/F[i,j]
    else
        return (a + b + sqrt(2 * (h/F[i,j])^2 − (a − b)^2)) /2
    end
end
```

$$\left(a + b + \sqrt{2 * (h/F[i,j])^2 - (a-b)^2}\right)/2$$

Fast Marching Method

- Like a continous Dijkstra algorithm
- Each point visited once
- Points labelled as KNOWN, BAND, or FAR
- Initially: all points FAR, except for boundary points, which are KNOWN
- BAND points: neighbours of known points
- At each iteration select BAND point with smallest value (becomes KNOWN), compute value of neighbours
- BAND points stored in an *indexed* priority queue

FMM step 1

$$
\begin{matrix}
100 & 100 & 100 & 100 & 100 & 100 & 100 \\
100 & 100 & 100 & 100 & 100 & 100 & 100 \\
100 & 100 & 100 & \mathbf{1} & 100 & 100 & 100 \\
100 & 100 & \mathbf{1} & 0 & \mathbf{1} & 100 & 100 \\
100 & 100 & 100 & \mathbf{1} & 100 & 100 & 100 \\
100 & 100 & 100 & 100 & 100 & 100 & 100 \\
100 & 100 & 100 & 100 & 100 & 100 & 100
\end{matrix}
$$

FMM Iteration 1

100	100	100	100	100	100
100	100	100	100	100	100
100	100	100	**1**	100	100
100	100	**1**	0	**1**	100
100	100	**2**	1	**2**	100
100	100	100	**2**	100	100
100	100	100	100	100	100

FMM Iteration 2

100	100	100	100	100	100
100	100	100	100	100	100
100	100	**2**	**1**	100	100
100	**2**	1	0	**1**	100
100	100	**1.7071**	1	**2**	100
100	100	100	**2**	100	100
100	100	100	100	100	100

FMM Iteration 10

100	100	100	100	100	100
100	100	**2.7071**	**2**	**2.7071**	100
100	**2.7071**	1.7071	1	1.7071	**2.5453**
100	**2**	1	0	1	2
100	**2.7071**	1.7071	1	1.7071	**2.5453**
100	100	**2.5453**	2	**2.5453**	100
100	100	100	**3**	100	100

FMM Iteration 11

100	100	100	100	100	100
100	100	**2.7071**	**2**	**2.7071**	100
100	**2.5453**	1.7071	1	1.7071	**2.5453**
3	2	1	0	1	2
100	**2.5453**	1.7071	1	1.7071	**2.5453**
100	100	**2.5453**	2	**2.5453**	100
100	100	100	**3**	100	100

Input: As in FSM, but also with desired width L of solution
Output: Solution matrix U
Data: $(n_i + 2) \times (n_j + 2)$ matrix G, with possible values KNOWN, BAND, FAR. Initialized to FAR everywhere except boundary, where it is initialized to KNOWN
Data: Indexed min priority queue `iQmin`

foreach *element* (i, j) *of* G *such that* $G[i, j] = \text{KNOWN}$ **do**
    `updateNeighbors(iQmin,` U, G, i, j, F, h`)`
end
while `iQmin` *is not empty* **do**
    $(l, m) \leftarrow$ `iQmin.minIndex() // retrieve index of element at front of queue`
    if $U[l, m] > L$ **then** break
    $G[l, m] \leftarrow \text{KNOWN}$
    `iQmin.delMin() // remove element at front of queue`
    `updateNeighbors(iQmin,` U, G, i, j, F, h`)`
end

Procedure `updateNeighbors(iQmin,` U, G, i, j, F, h)

  for $(l, m) \leftarrow (i+1, j), \ (i-1, j), \ (i, j+1), \ (i, j-1)$ **do**

    if $G[l, m] = \text{KNOWN} \vee (l, m)$ *outside domain* **then**

    continue

    $u_{\text{temp}} \leftarrow$ `solveQuadratic(`G, U, l, m, F, h`)`

    if $u_{temp} < U[l, m]$ **then**

      $U[l, m] \leftarrow u_{\text{temp}}$

      $G[l, m] \leftarrow \text{BAND}$

      if `iQmin.contains(`(l, m)`)` **then**

        `iQmin.change(`$(l, m), u_{temp}$`)// update element in queue`

      else

        `iQmin.insert(`$(l, m), u_{temp}$`)// insert element in queue`

      end

    end

  end

end

```
Procedure solveQuadratic(G, U, i, j, F, h)
    a ←selectMin(G, U, i + 1, j, i − 1, j)
    b ←selectMin(G, U, i, j + 1, i, j − 1)
    if a = −1 then
        return b + h/F[i, j]
    else if b = −1 then
        return a + h/F[i, j]
    else if |a − b| ≥ h/F[i, j] then
        return min(a, b) +h/F[i, j]
    else
        return (a + b + sqrt(2 ∗ (h/F[i, j])^2 − (a − b)^2)) /2
    end
end
```



```
Procedure selectMin(G, U, l, m, p, q)
    x ← −1
    if G[l, m] = KNOWN ∧ G[p, q] = KNOWN then
        x ← min(U[l, m], U[p, q])
    else if G[l, m] ≠ KNOWN ∧ G[p, q] = KNOWN then
        x ← U[p, q]
    else if G[l, m] = KNOWN ∧ G[p, q] ≠ KNOWN then
        x ← U[l, m]
    end
    return x
end
```

Analysis

- FSM: $O(n^2)$ for a $n \times n$ grid, number of iterations independent of problem size
- FMM: $O(n^2 \log n^2)$
- FMM usually faster in practice

Parallel Design Exploration

- Influence of geometry and speed function
- Many independent small problems or a few large problems?
- Strong scaling vs. weak scaling

Parallel FSM

Idea 1: do four sweeps in parallel

4.7551	**4.0480**	**3.4422**	**3**	**4**	100	100
4.0480	**3.2524**	**2.5453**	**2**	**3**	100	100
3.4422	**2.5453**	**1.7071**	**1**	**2**	100	100
3	**2**	**1**	0	**1**	100	100
4	**3**	**2**	**1**	100	100	100
100	100	100	100	100	100	100
100	100	100	100	100	100	100

100	100	100	100	100	100	100
100	100	100	100	100	100	100
4	**3**	**2**	**1**	100	100	100
3	**2**	**1**	0	**1**	100	100
3.4422	**2.5453**	**1.7071**	**1**	**2**	100	100
4.0480	**3.2524**	**2.5453**	**2**	**3**	100	100
4.7551	**4.0480**	**3.4422**	**3**	**4**	100	100

100	100	100	100	100	100	100
100	100	100	100	100	100	100
100	100	100	**1**	**2**	**3**	**4**
100	100	**1**	0	**1**	**2**	**3**
100	100	**2**	**1**	**1.7071**	**2.5453**	**3.4422**
100	100	**3**	**2**	**2.5453**	**3.2524**	**4.0480**
100	100	**4**	**3**	**3.4422**	**4.0480**	**4.7551**

Parallel FSM

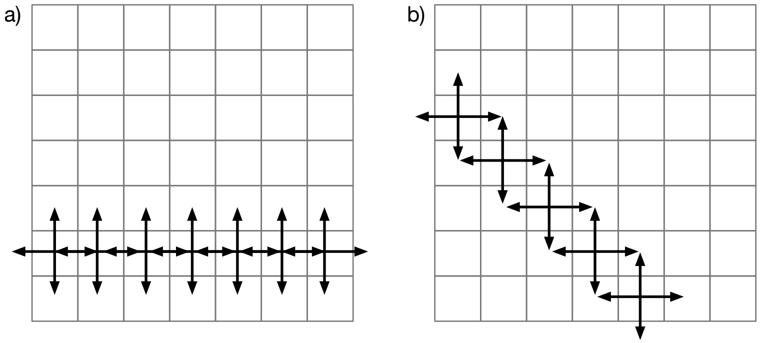
Idea 2: 2D domain decomposition

100	100	100	3	3.5453	4.2524	5.0480
100	100	100	2	2.7071	3.5453	4.4422
100	100	100	1	2	3	4
3	2	1	0	1	2	3
3.5453	2.7071	2	1	1.7071	2.5453	3.4422
4.2524	3.5453	3	2	2.5453	3.2524	4.0480
5.0480	4.4422	4	3	3.4422	4.0480	4.7551

4.7551	**4.0480**	**3.4422**	3	**3.4422**	**4.0480**	**4.7551**
4.0480	**3.2524**	**2.5453**	2	**2.5453**	**3.2524**	**4.0480**
3.4422	**2.5453**	**1.7071**	1	**1.7071**	**2.5453**	**3.4422**
3	2	1	0	1	2	3
3.4422	**2.5453**	**1.7071**	1	1.7071	2.5453	3.4422
4.0480	**3.2524**	**2.5453**	2	2.5453	3.2524	4.0480
4.7551	**4.0480**	**3.4422**	3	3.4422	4.0480	4.7551

Parallel FSM

Idea 3: Reordering sweeps



Parallel FMM: Fast Iterative Method

- Update band elements in parallel
- Update onto copy of grid

100	100	100	100	100	100	100
100	100	100	**2**	100	100	100
100	100	**1.7071**	1	**1.7071**	100	100
100	**2**	1	0	1	**2**	100
100	100	**1.7071**	1	**1.7071**	100	100
100	100	100	**2**	100	100	100
100	100	100	100	100	100	100

100	100	100	**3**	100	100	100
100	100	**2.5453**	2	**2.5453**	100	100
100	**2.5453**	1.7071	1	1.7071	**2.5453**	100
3	2	1	0	1	2	**3**
100	**2.5453**	1.7071	1	1.7071	**2.5453**	100
100	100	**2.5453**	2	**2.5453**	100	100
100	100	100	**3**	100	100	100