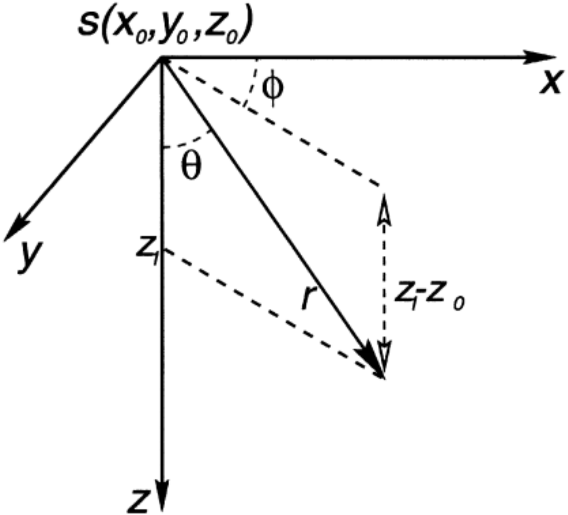
Parallel computing in solving eikonal equations

Qi Yingyu
Supervisor: Prof. Tong Ping

Nanyang Technological University, Singapore

Outline

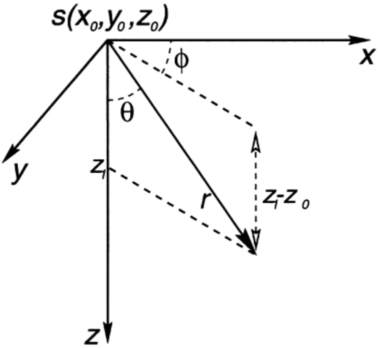
Spherical coordinate



Alkhalifah and Fomel 2001

Isotropic eikonal equation

$$\left(\frac{\partial t}{\partial r}\right)^2 + \frac{1}{r^2}\left(\frac{\partial t}{\partial \theta}\right)^2 + \frac{1}{r^2 \sin^2\theta}\left(\frac{\partial t}{\partial \phi}\right)^2 = \frac{1}{c^2}$$



Discretization

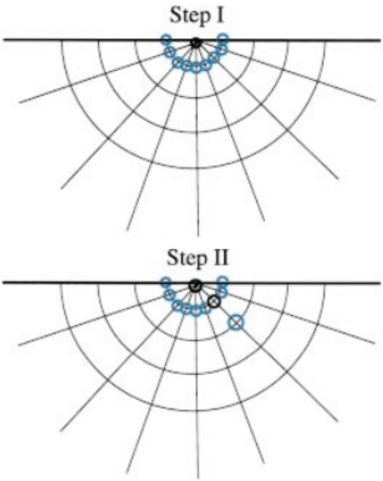
$$\max(D_{ijk}^{-r}t, -D_{ijk}^{+r}t, 0)^2 + \max(D_{ijk}^{-\theta}t, -D_{ijk}^{+\theta}t, 0)^2 + \max(D_{ijk}^{-\phi}t, -D_{ijk}^{+\phi}t, 0)^2 = \frac{1}{c_{ijk}^2}.$$

$$D_{ijk}^{-r}t = \frac{t_{i,j,k} - t_{i-1,j,k}}{\Delta r}, \quad D_{ijk}^{+r}t = \frac{t_{i+1,j,k} - t_{i,j,k}}{\Delta r};$$

$$D_{ijk}^{-\theta}t = \frac{t_{i,j,k} - t_{i,j-1,k}}{r\Delta\theta}, \quad D_{ijk}^{+\theta}t = \frac{t_{i,j+1,k} - t_{i,j,k}}{r\Delta\theta};$$

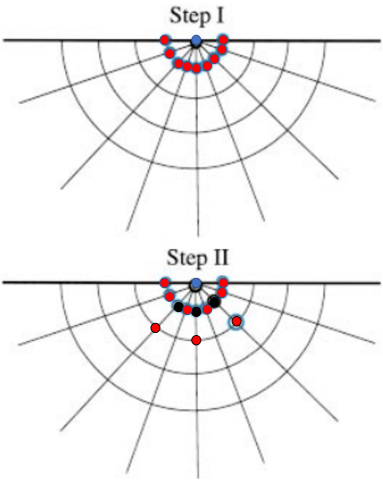
$$D_{ijk}^{-\phi}t = \frac{t_{i,j,k} - t_{i,j,k-1}}{r\sin\theta\Delta\phi}, \quad D_{ijk}^{+\phi}t = \frac{t_{i,j,k+1} - t_{i,j,k}}{r\sin\theta\Delta\phi}.$$

Fast marching method



Alkhalifah and Fomel 2001

Group marching method



Group Marching Method

Group marching method

Theorem

(Kim 2001) Let L be the close set, and define the set G as a subset of L,

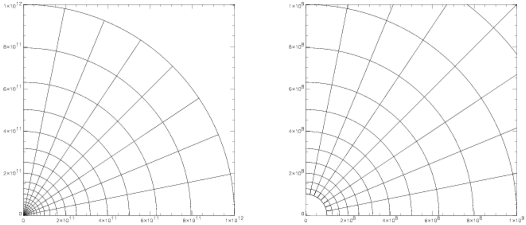
$$G = \{x \in L : \phi(x) \leq \phi_{L,min} + \frac{1}{\sqrt{2} f_{L,max}}\}. \tag{1}$$

where $\phi_{L,min} = \min\{\phi(x) : x \in L\}$ *and* $f_{L,max} = \max\{f(x) : x \in L\}$

Outline

Principle of Parallel Computing

1. the algorithm should not impose a particular update order
2. the algorithm should not use a separate, heterogeneous data structure for sorting, and
3. the algorithm should be able to simultaneously update multiple points



Zingale and Woosley

Fast Iterative Method

Algorithm 3.1: SolveQuadratic(a, b, c, f)

---

comment: Returns the value $u = U_{\mathbf{x}}$ that solves $g(U, \mathbf{x}) = 0$, where $a \leq b \leq c$

$u \leftarrow c + 1/f$
if $u \leq b$ **return** (u)
$u \leftarrow (b + c + \mathrm{sqrt}(-b^2 - c^2 + 2bc + 2/f^2))/2$
if $u \leq a$ **return** (u)
$u \leftarrow (2(a + b + c) + \mathrm{sqrt}(4(a + b + c)^2 - 12(a^2 + b^2 + c^2 - 1/f^2)))/6$
return (u)

Fast Iterative Method

Algorithm 3.2: UPDATE($\mathbf{X}$)

comment: 1. Initialization ($\mathbf{X}$: set of grid points, L : active list)

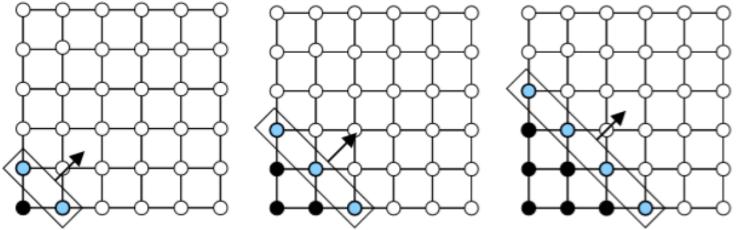
for each $\mathbf{x} \in \mathbf{X}$
- **do**
  - **if** $\mathbf{x}$ is source
  - **then** $U_{\mathbf{x}} \leftarrow 0$
  - **else** $U_{\mathbf{x}} \leftarrow \infty$

for each $\mathbf{x} \in \mathbf{X}$
- **do**
  - **if** any neighbor of $\mathbf{x}$ is source
  - **then** add $\mathbf{x}$ to L

comment: 2. Update points in L

while L is not empty
- **do**
  - **for each** $\mathbf{x} \in L$
  - **do**
    - $p \leftarrow U_{\mathbf{x}}$
    - $q \leftarrow g(U_{\mathbf{x}})$
    - $U_{\mathbf{x}} \leftarrow q$
    - **if** $|p - q| < \epsilon$
    - **then**
      - **for each** 1-neighbor $\mathbf{x}_{nb}$ of $\mathbf{x}$
      - **do**
        - **if** $\mathbf{x}_{nb}$ is not in L
        - **then**
          - $p \leftarrow U_{\mathbf{x}_{nb}}$
          - $q \leftarrow g(U_{\mathbf{x}_{nb}})$
          - **if** $p > q$
          - **then**
            - $U_{\mathbf{x}_{nb}} \leftarrow q$
            - add $\mathbf{x}_{nb}$ to L
      - remove $\mathbf{x}$ from L

Fast Iterative Method



(a) Initial stage (b) After first update (c) After second update

Fast Iterative Method

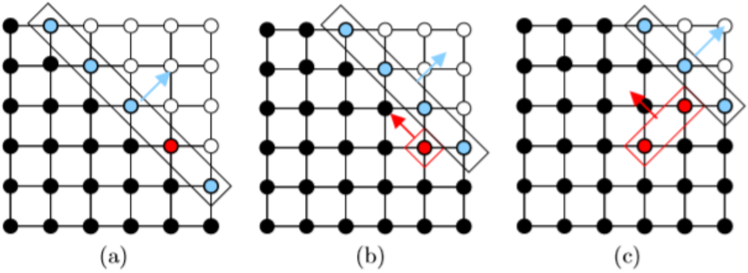


Figure 3: Schematic 2D example of the change of the characteristic direction.

Outline

Convergency of Fast Iterative Method

Lemma 3.1. *For strictly positive speed functions, the FIM algorithm appends every grid point to the active list at least once.*

Proof. For every non-source point, any path in the domain from that point to the boundary conditions has cost $< \infty$. As shown in the initialization step in pseudo code 0.3.2, all non-source points are initialized as ∞. Hence, the active list grows outward from the boundary condition in one-connected rings until it passes over the entire domain. $\square$

Convergency of Fast Iterative Method

Lemma 3.2. *FIM algorithm converges.*

Proof. For this we rely on monotonicity (decreasing) of the solution and boundedness (positive). From the pseudo code 0.3.2 we see that a point is added to the active list and its tentative solution is updated only when the new solution is smaller than the previous one. All updates are positive by construction. □

Convergency of Fast Iterative Method

Lemma 3.3. *The solution U at the completion of FIM algorithm with $\epsilon = 0$ (error threshold) is consistent with the corresponding Hamiltonian given in Equation 1.*

Proof. Each point in the domain is appended to the active list at least once. Each point $\mathbf{x}$ is finally removed from $\mathcal{L}$ only when $g(U, \mathbf{x}) = 0$ and the upwind neighbors (which impact this calculation) are also inactive. Any change in those neighbors causes $\mathbf{x}$ to be re-appended to the active list. Thus, when the active list is empty (the condition for completion), $g(U, \mathbf{x}) = 0$ for the entire domain. $\square$

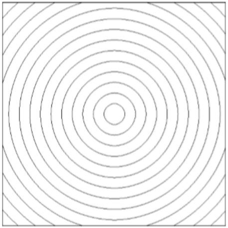
Convergency of Fast Iterative Method

Theorem 3.4. *FIM algorithm, for $\epsilon = 0$ gives an approximate solution to Equation 1 on the discrete grid.*

Proof. The proof of the theorem is given by the convergence and consistency of the solution, as given lemmas above. □

Outline

Homogeneous, Constant speed.

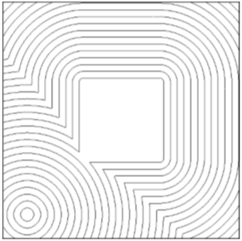


(a) Example 1

	2D				3D			
	256^2	512^2	1024^2	2048^2	32^3	64^3	128^3	256^3
FMM	0.141	0.563	2.516	11.547	0.094	0.922	10.812	129
GMM	0.062	0.312	1.328	6.079	0.062	0.469	4.469	39
FSM	0.063	0.266	1.484	5.968	0.062	0.5	5.532	44
FIM	0.078	0.313	1.282	5.516	0.047	0.406	3.578	30

Table 1: Running time on speed example 1

1-D Discontinuity

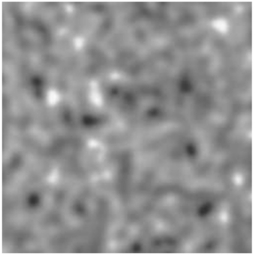


(b) Example 2

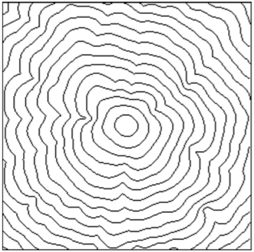
	2D				3D			
	256^2	512^2	1024^2	2048^2	32^3	64^3	128^3	256^3
FMM	0.141	0.641	3.813	31.82	0.093	0.937	11.20	165
GMM	14.79	67.67	304	1336	5.64	56.04	954	12916
FSM	0.063	0.266	1.484	5.937	0.093	0.922	11.57	109
FIM	0.141	0.672	4.032	31.61	0.062	0.531	6.609	50

Table 2: Running time on speed example 2

Inhomogeneous



(a) Example 3 speed map

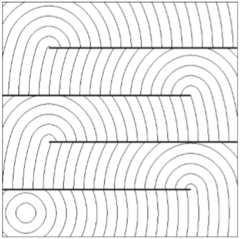


(b) Example 3 solution

	2D (256^2)	3D (256^3)
FMM	0.141	174
GMM	0.078	54
FSM	0.172	188
FIM	0.141	108

Table 3: Running time on speed example 3

Example 4

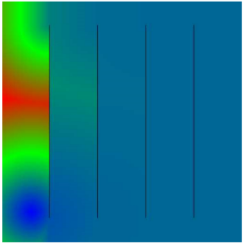


(a) Example 4

	2D				3D			
	256^2	512^2	1024^2	2048^2	32^3	64^3	128^3	256^3
FMM	0.109	0.469	2.078	9.141	0.078	0.812	9.656	128
GMM	0.062	0.281	1.203	4.985	0.046	0.422	4.015	39
FSM	0.125	0.516	2.937	11.79	0.14	1.188	13.57	111
FIM	0.078	0.297	1.172	4.703	0.046	0.359	3.156	29

Table 4: Running time on speed example 4

Example 5



(b) Example 5

	2D				3D			
	256^2	512^2	1024^2	2048^2	32^3	64^3	128^3	256^3
FMM	0.125	0.532	2.328	10.7	0.078	0.922	13.87	290
GMM	9.937	51.18	265.25	1217	3.687	35.79	376.68	6493
FSM	0.11	0.453	2.656	9.5	0.109	1.203	16.28	154
FIM	0.125	0.516	2.235	9.5	0.125	1.328	16.61	233

Table 5: Running time on speed example 5

Performance

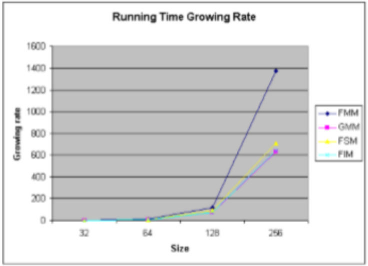


Figure 7: Comparison of CPU time increasing rate

	Example 1	Example 2	example 3	Example 4	Example 5
FMM	2.98	2.98	2.98	2.97	2.97
GMM	5.83	2.14	3.75	5.364	2.75
FSM	9	21	35	22	30
FIM	4.98	6.9	9.97	4.97	23.05

Table 6: Average number of solving quadratic equation per point on 256^3

Performance

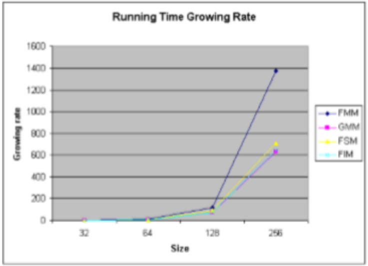


Figure 7: Comparison of CPU time increasing rate

	FMM	GMM	FSM	FIM
Large Speed Contrasts	+	−	+	+
Large Data Size	−	+	+	+
Varying Speed Values	−	+	−	−
Char. Dir. Changes	+	+	−	+
Parallelization	−	+	−	+

Table 7: Overall performance comparison on data categories

Thank you!