Terracost: A Versatile and Scalable Approach to Computing Least-Cost Path Surfaces for Massive Grid-Based Terrains

**Terracost:** Hazel, Toma, Vahrenhold, Wickremesinghe

**I/O-Efficient Algorithms**

- Input data (grid) stored on disk
- I/O model (Carnegie and Ulmer, 1985)
- For a grid of size $M 	imes M$
  - 6 blocks of size $B$
  - $O(1)$ I/Os for each block!

**Outline**

- Background
- Shortest paths
- Related work
- Least-cost path surfaces
- Conclusions

**Least-cost path surfaces**

- Problem
  - Input: A cost surface on a grid and a set of sources
  - Output: A shortest path surface — each point represents the shortest distance to a source

- Applications
  - Distributed fire from different sources
  - Distance from rivers or roads
  - Cost of building pipelines or roads

- Basic data structures
  - Ridge: $\mathrm{Cost} = \max(x,y)$
  - Parent: $\mathrm{Parent}(x,y) \in \{x,y\}$

**Shortest Paths**

- Least-cost path can be computed by computing shortest paths between sources
- Plane $\rightarrow$ Sphere $\rightarrow$ Plane

**Dijkstra’s Algorithm**

- Efficient algorithm
- Shortest path on a grid
- Parallelization on a cluster

**Related Work**

- Efficient algorithms on a grid
- Plane $\rightarrow$ Plane $\rightarrow$ Plane

- Experimental analysis
  - Multiple sources
  - Multiple grid terrains
  - Related work

**Conclusions**

- Versatile: Interpolate between versions optimized for I/O or CPU
- Experimental analysis on real-life data
- Can handle bigger grids
- Can handle larger grids
Terracost: Hazel, Toma, Vahrenhold, Wickremesinghe

I/O-Efficient SSSP on Grids [ATV'01]

- Divide grid $G$ into subgrids of size $R^2$
- Compute $SP$ of each subgrid
- Determine the optimal tile size analytically
- Tiling in Terracost allows for parallelization
- Future Work
  - Determine the optimal tile size analytically
  - Find I/O-efficient SSSP/MSSP w/o increase of CPU-efficiency

Key Points
- Dijkstra's algorithm is very slow for large terrains
- GRASS users have complained it
- r.cost has same functionality
- Terracost lends itself to parallelization

Conclusions and Future Work

- But, to optimize CPU, we want a small $R$.
- So, to optimize I/Os, we want a large $R$.

Experimental Analysis

<table>
<thead>
<tr>
<th>Name</th>
<th>Number of processors</th>
<th>Time (sec)</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hazel</td>
<td>1.6 G</td>
<td>28.6</td>
<td>1.7 GB/s</td>
</tr>
<tr>
<td>Toma</td>
<td>8.0 G</td>
<td>32.5</td>
<td>1.2 GB/s</td>
</tr>
<tr>
<td>Vahrenhold</td>
<td>2.5 G</td>
<td>35.1</td>
<td>1.2 GB/s</td>
</tr>
<tr>
<td>Wickremesinghe</td>
<td>512 MB</td>
<td>38.2</td>
<td>1.2 GB/s</td>
</tr>
</tbody>
</table>

Future Work

- Find I/O-efficient SSSP/MSSP w/o increase of CPU-efficiency

CPU-I/O Tradeoff

- $P$: the size
- $O(I/O)$ complexity:
  - $O(N)$ parallelism on the boundary
  - $O(N)$ parallelism on the boundary
- $O(P)$ complexity:
  - $O(NP)$ parallelism on the boundary
- $O(P)$ complexity:
  - $O(NP)$ parallelism on the boundary
- $O(P)$ complexity:
  - $O(NP)$ parallelism on the boundary
- $O(P)$ complexity:
  - $O(NP)$ parallelism on the boundary

Terracost on Clusters

- Terracost lends itself to parallelization
- We parallelized the most CPU-intensive part
- But we can't ignore CPU-complexity completely
- $O$-bottleneck increases with number of sources for MSSP
- The CPU bottleneck allows for parallelization

Lower NE, Cost = Slope, 1GB RAM, Single source

Terracost runs on clusters

- GRASS users have complained it
- r.cost has same functionality
- Terracost lends itself to parallelization
- But, to optimize CPU, we want a small $R$.
- So, to optimize I/Os, we want a large $R$.