Parallelism in Dynamic Well-Spaced Point Sets

Umut A. Acar\textsuperscript{1}, Andrew Cotter\textsuperscript{2}
Benoît Hudson\textsuperscript{3}, Duru Türkoğlu\textsuperscript{4}

1. Max-Planck Institute for Software Systems
2. Toyota Technological Institute at Chicago
3. Autodesk, Inc.
4. University of Chicago
## Dynamic-parallel relationship

<table>
<thead>
<tr>
<th></th>
<th>Work</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sequential</strong></td>
<td>$n \cdot \log(n)$</td>
<td>$n \cdot \log(n)$</td>
</tr>
<tr>
<td><strong>Parallel</strong></td>
<td>$n \cdot \log(n)$</td>
<td>$\log(n)$</td>
</tr>
<tr>
<td><strong>Dynamic (k changes)</strong></td>
<td>$k \cdot \log(n)$</td>
<td>$k \cdot \log(n)$</td>
</tr>
<tr>
<td><strong>Dynamic &amp; parallel</strong></td>
<td>$k \cdot \log(n)$</td>
<td>$\log(n)$</td>
</tr>
</tbody>
</table>

- Parallel algorithms reduce *time*
- Dynamic algorithms reduce *work*
- We can get the best of both worlds!
Dynamic-parallel relationship

Computation graph
- Nodes are tasks
- Edges are dependencies
Dynamic-parallel relationship

Computation graph
- Nodes are tasks
- Edges are dependencies

Parallelism
- Execute independent operations in parallel
- Low depth results in good speedups
Dynamic-parallel relationship

Change propagation

- Re-execute affected nodes when input changes
  - This may change the graph structure
- For good performance:
  - Low depth
  - "Narrow" dependency chains
Dynamic-parallel relationship

Change propagation
- Ideally:
  - Small changes: little re-execution
Dynamic-parallel relationship

Change propagation

- Ideally:
  - Small changes: little re-execution
  - Large changes: much re-execution
Dynamic-parallel relationship

Dynamic & parallel algorithms
● Perform change propagation in parallel
● Parallel algorithms reduce time
● Dynamic algorithms reduce work
● Dynamic & parallel reduce both

![Diagram showing inputs and outputs with arrows connecting them, indicating a flow process.](image)
Dynamic-parallel relationship

Dynamic & parallel algorithms
- Small changes result in:
  - Large gains from dynamism
  - Small gains from parallelism
Dynamic-parallel relationship

Dynamic & parallel algorithms

- Small changes result in:
  - Large gains from dynamism
  - Small gains from parallelism

- Large changes result in:
  - Small gains from dynamism
  - Large gains from parallelism
Well-spaced point sets

- "Skinny" triangles
- Small angles

- No skinny triangles
- No small angles
Well-spaced point sets

Input point set
- Delaunay triangulation contains small angles
Well-spaced point sets

**Input point set**
- Delaunay triangulation contains small angles

**Procedure**
- Starting from input points
Well-spaced point sets

Input point set
- Delaunay triangulation contains small angles

Procedure
- Starting from input points
- Make point set well-spaced by inserting "Steiner points"
Well-spaced point sets

Input point set
- Delaunay triangulation contains small angles

Procedure
- Starting from input points
- Make point set well-spaced by inserting "Steiner points"

Output point set
- No small angles
Well-spaced point sets

Dynamic modifications
- Insertion and erasure of input points
- Result in *local* changes
- Maintain size-optimality of well-spaced superset
Well-spaced point sets

Dynamic modifications
- Insertion and erasure of input points
- Result in *local* changes
- Maintain size-optimality of well-spaced superset
Well-spaced point sets

Dynamic modifications
- Insertion and erasure of input points
- Result in *local* changes
- Maintain size-optimality of well-spaced superset
Well-spaced point sets

Well-spacedness

- For a point $v$, let $NN$ be the distance to $v$'s nearest neighbor
Well-spaced point sets

**Well-spacedness**

- For a point \( v \), let \( NN \) be the distance to \( v \)'s nearest neighbor.
- We call \( v \) "\( \rho \)-well-spaced" [Talmor '97] if its Voronoi cell is contained in a ball, centered on \( v \), of radius \( \rho \cdot NN \).
- \( \rho > 1 \) is a parameter.
Well-spaced point sets

Well-spacedness
- A point set is well-spaced if all of its points are well-spaced
- Well-spaced point sets result in high-quality triangulations
Our algorithm

Fill operation
- Find $NN$ and $\rho \cdot NN$
Our algorithm

**Fill operation**
- Find $NN$ and $\rho \cdot NN$
- Find $v$'s Voronoi cell
Our algorithm

Fill operation
- Find $NN$ and $\rho \cdot NN$
- Find $v$'s Voronoi cell
- Define the "picking region"
Our algorithm

**Fill operation**
- Find $NN$ and $\rho \cdot NN$
- Find $v$'s Voronoi cell
- Define the "picking region"
- Choose a Steiner point from the picking region
Our algorithm

Fill operation

- Find $NN$ and $\rho \cdot NN$
- Find $v$'s Voronoi cell
- Define the "picking region"
- Choose a Steiner point from the picking region
- Update the Voronoi cell
- Repeat if necessary
Our algorithm

Improvement to fill operation

- Restrict the picking region to a radius of $\beta \cdot NN$ for $\beta > \rho$
- This eliminates superfluous dependencies
Our algorithm

Scheduling

- The fill operation *directly* enforces well-spacedness
- Could repeatedly "fill" points until the point set is well-spaced
- Overall goals are to:
  - Insert a small number of Steiner points
  - Have a "good" computation graph
- To ensure these, fill operations must be scheduled carefully
Our algorithm

Ranks

- Define the "rank" of $v$ as the (integer) base-$\rho$ logarithm of $NN$
- Fill the points in rank order
  - When Steiner points are inserted, schedule them, and reschedule their neighbors
- Each point will be filled exactly once
- Output will be well-spaced and size-optimal
- The computation graph will not be "good"
  - Neither low depth nor narrowness
Our algorithm

**Coloring**
- Orders operations *within* a rank
- Repeating grid covers entire space
- Color of \( v \) is color of grid cell containing \( v \)
Our algorithm

Coloring
● Orders operations *within* a rank
● Repeating grid covers entire space
● Color of $v$ is color of grid cell containing $v$

Properties
● No grid cell contains multiple points
Our algorithm

Coloring
- Orders operations \textit{within} a rank
- Repeating grid covers entire space
- Color of \( v \) is color of grid cell containing \( v \)

Properties
- No grid cell contains multiple points
- Fill operations scheduled at the same color are independent
Our algorithm

Coloring parameters
- Number of colors
- Side-length of grid squares
- Both depend only on $\rho$ and $\beta$
- Number of colors is constant
Our algorithm

Scheduling by rank and color

- Order fill operations first by rank, then by color
  - Constant number of colors
  - Logarithmically many ranks
- Good dynamic and parallel performance
  - Parallel runtime is $\log(n)$
  - Work is $k \cdot \log(n)$, for $k$ dynamic changes
## Experiments

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$n$</th>
<th>$k$</th>
<th>1 core</th>
<th>2 core</th>
<th>4 core</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Zealand (2D)</td>
<td>18595</td>
<td>463</td>
<td>49×</td>
<td>74×</td>
<td>98×</td>
</tr>
<tr>
<td>Lake Superior (2D)</td>
<td>33487</td>
<td>296</td>
<td>102×</td>
<td>150×</td>
<td>189×</td>
</tr>
<tr>
<td>San Francisco (2D)</td>
<td>85910</td>
<td>322</td>
<td>196×</td>
<td>299×</td>
<td>408×</td>
</tr>
<tr>
<td>Bunny (3D)</td>
<td>35947</td>
<td>657</td>
<td>9.3×</td>
<td>16×</td>
<td>26×</td>
</tr>
<tr>
<td>Armadillo (3D)</td>
<td>172974</td>
<td>4242</td>
<td>14×</td>
<td>25×</td>
<td>42×</td>
</tr>
</tbody>
</table>

- Speedups are relative to recomputing from scratch
- $n$ is dataset size, $k$ is changeset size
Related work

Well-spaced point sets
- Chew '89 (first non-heuristic construction algorithm)
- Ruppert '95 (size-optimal in 2D)
- Har-Peled, Üngör '05 (efficient in 2D)
- Hudson, Miller, Phillips '06 (efficient in any D)
- Hudson, Miller, Phillips '07 (parallel version of above)

Direct influences
- Spielman, Teng, Üngör '02 (parallel)
- Acar, Cotter, Hudson, Türkoğlu '10 (dynamic)
Summary

Our findings

- An appealing connection between dynamic and parallel algorithms
- An efficient parallel change-propagation algorithm for the well-spaced point set problem
- Results demonstrating that dynamic and parallel algorithms interact well in practice
Summary

In our paper:
- Description of parallel dynamic algorithm targeting EREW PRAM model
- Proofs of size-optimality and performance guarantees
- Description of implementation on real-world multi-core hardware
- More detailed experimental evaluation
Thank you