Parallel Space-Time Kernel Density Estimation

**Erik Saule**†, Dinesh Panchananam†,
Alexander Hohl‡, Wenwu Tang‡, Eric Delmelle‡

† Dept. of Computer Science
‡Dept. of Geography and Earth Sciences
UNC Charlotte
Email: \{esaule,dpanchan,ahohl,wtang4,eric.delmelle\}@uncc.edu

Scheduling in Knoxville
May 26th, 2017
Outline

1. Space Time Kernel Density
2. Sequential Algorithms
3. Domain-Based Parallelism
4. Point-Based Parallelism
5. Conclusion
Space Time Kernel Density

What is it?
- Common way of visualizing events with time and place information
- Basically voxelize the space
- Give a value to each voxel that depends on the number of neighboring event to the voxel (with some kind of decay).
- Essentially a generalization of density maps (e.g., population density)

What is it useful for?
- Monitoring disease outbreak
- Political analysis
- Social media analysis
- Ornithology
Space-Time Kernel Density Estimate Formally

For a voxel \( x, y, t \)

\[
\hat{f}(x, y, t) = \frac{1}{nh_s^2 h_t} \sum_{i} \mathbf{i} | d_i < h_s, t_i < h_t | k_s \left( \frac{x-x_i}{h_s}, \frac{y-y_i}{h_s} \right) k_t \left( \frac{t-t_i}{h_t} \right)
\]

\[
k_s(u, v) = \frac{\pi}{2} (1 - u)^2 (1 - v)^2
\]

\[
k_t(w) = \frac{3}{4} (1 - w)^2
\]

\( h_s \) is the spatial bandwidth

\( h_t \) is the temporal bandwidth

\( n \) is the number of points (events)

Similar to computing sums of radial basis functions from physics.
Dengue Fever in Cali, Colombia

\[ h_s = 2500 \text{m}, \ h_t = 14 \text{days} \]

\[ h_s = 500 \text{m}, \ h_t = 7 \text{days} \]
Voxel Based Algorithm VB

Algorithm

for all voxels $s = (x, y, t)$ do

$sum = 0$

for all points $i$ at $x_i, y_i, t_i$ do

if $\sqrt{(x_i - x)^2 + (y_i - y)^2} < h_s$ and $|t_i - t| \leq h_t$ then

$sum + = k_s\left(\frac{x-x_i}{h_s}, \frac{y-y_i}{h_s}\right) k_t\left(\frac{t-t_i}{h_t}\right)$

$stkde[X][Y][T] = \frac{sum}{nh^2_s h_t}$

• $\theta(G_x G_y G_t n)$ distance tests
• $\theta(nh^2_s H_t)$ density values
• Complexity: $\theta(G_x G_y G_t n)$

But pleasingly parallel.
**Point Based Algorithm PB**

**Algorithm**

```
for all voxels \( s = (x, y, t) \) do
    \( \text{stkde}[X][Y][T] = 0 \)

for each points \( i \) at \( x_i, y_i, t_i \) do
    for \( X_i - H_s \leq X \leq X_i + H_s \) do
        for \( Y_i - H_s \leq Y \leq Y_i + H_s \) do
            for \( T_i - T_s \leq T \leq T_i + H_s \) do
                if \( \sqrt{(x_i - x)^2 + (y_i - y)^2} < h_s \) and \( |t_i - t| \leq h_t \) then
                    \( \text{stkde}[X][Y][T] += \frac{k_s \left( \frac{x-x_i}{h_s}, \frac{y-y_i}{h_s} \right) k_t \left( \frac{t-t_i}{h_t} \right)}{n h_s^2 h_t} \)
```

- \( \Theta(G_x G_y G_t) \) for memory initialization
- \( \Theta(nH_s^2 H_t) \) density computations
- Complexity: \( \Theta(G_x G_y G_t + nH_s^2 H_t) \)
- (Gain the \( \theta(G_x G_y G_t n) \) distance tests)
Exploiting Symmetries PB-SYM

For each point:
- Compute the $K_t$
- Compute the $K_s$
- Cross product

Complexity is the same, but saves computation in practice
### Experimental settings

<table>
<thead>
<tr>
<th>Instance</th>
<th>n</th>
<th>$G_x \times G_y \times G_t$</th>
<th>Size</th>
<th>$H_s$</th>
<th>$H_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dengue_Lr-Lb</td>
<td>11056</td>
<td>148x194x728</td>
<td>79MB</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Dengue_Lr-Hb</td>
<td>11056</td>
<td>148x194x728</td>
<td>79MB</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>Dengue_Hr-Lb</td>
<td>11056</td>
<td>294x386x728</td>
<td>315MB</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Dengue_Hr-Hb</td>
<td>11056</td>
<td>294x386x728</td>
<td>315MB</td>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>Dengue_Hr-VHb</td>
<td>11056</td>
<td>294x386x728</td>
<td>315MB</td>
<td>50</td>
<td>14</td>
</tr>
<tr>
<td>PollenUS_Lr-Lb</td>
<td>588189</td>
<td>131x61x84</td>
<td>2MB</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>PollenUS_Hr-Lb</td>
<td>588189</td>
<td>651x301x84</td>
<td>62MB</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>PollenUS_Hr-Mh</td>
<td>588189</td>
<td>651x301x84</td>
<td>62MB</td>
<td>25</td>
<td>7</td>
</tr>
<tr>
<td>PollenUS_Hr-Hb</td>
<td>588189</td>
<td>651x301x84</td>
<td>62MB</td>
<td>50</td>
<td>14</td>
</tr>
<tr>
<td>PollenUS_VHr-Lb</td>
<td>588189</td>
<td>6501x3001x84</td>
<td>6252MB</td>
<td>100</td>
<td>3</td>
</tr>
<tr>
<td>PollenUS_VHr-VLb</td>
<td>588189</td>
<td>6501x3001x84</td>
<td>6252MB</td>
<td>50</td>
<td>3</td>
</tr>
<tr>
<td>Flu_Lr-Lb</td>
<td>31478</td>
<td>117x308x851</td>
<td>117MB</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Flu_Lr-Hb</td>
<td>31478</td>
<td>117x308x851</td>
<td>117MB</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Flu_Mr-Lb</td>
<td>31478</td>
<td>233x615x1985</td>
<td>1085MB</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Flu_Mr-Hb</td>
<td>31478</td>
<td>233x615x1985</td>
<td>1085MB</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Flu_Hr-Lb</td>
<td>31478</td>
<td>581x1536x5951</td>
<td>20260MB</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Flu_Hr-Hb</td>
<td>31478</td>
<td>581x1536x5951</td>
<td>20260MB</td>
<td>10</td>
<td>21</td>
</tr>
<tr>
<td>eBird_Lr-Lb</td>
<td>291990435</td>
<td>357x721x2435</td>
<td>2391MB</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>eBird_Lr-Hb</td>
<td>291990435</td>
<td>357x721x2435</td>
<td>2391MB</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>eBird_Hr-Lb</td>
<td>291990435</td>
<td>1781x3601x2435</td>
<td>59570MB</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>eBird_Hr-Hb</td>
<td>291990435</td>
<td>1781x3601x2435</td>
<td>59570MB</td>
<td>30</td>
<td>5</td>
</tr>
</tbody>
</table>

**Shared memory machine:**
- 2 Intel Xeon E5-2667 v3 (2 times 8 cores)
- 128GB of DRAM
- G++ 5.3 (with OpenMP 4.0)
In practice

<table>
<thead>
<tr>
<th>Instance</th>
<th>Time (in seconds)</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>VB</td>
<td>VB-DEC</td>
</tr>
<tr>
<td>Dengue_Lr-Lb</td>
<td>219.163</td>
<td>2.283</td>
</tr>
<tr>
<td>Dengue_Lr-Hb</td>
<td>220.591</td>
<td>13.878</td>
</tr>
<tr>
<td>Dengue_Hr-Lb</td>
<td>866.445</td>
<td>9.522</td>
</tr>
<tr>
<td>Dengue_Hr-Hb</td>
<td>871.774</td>
<td>55.206</td>
</tr>
<tr>
<td>Dengue_Hr-VHb</td>
<td>1056.172</td>
<td>404.845</td>
</tr>
<tr>
<td>PollenUS_Lr-Lb</td>
<td>518.859</td>
<td>7.639</td>
</tr>
<tr>
<td>PollenUS_Hr-Lb</td>
<td>12721.001</td>
<td>189.337</td>
</tr>
<tr>
<td>PollenUS_Hr-Mb</td>
<td>17179.482</td>
<td>3126.947</td>
</tr>
<tr>
<td>PollenUS_Hr-Vh</td>
<td>2666.104</td>
<td>583.175</td>
</tr>
<tr>
<td>PollenUS_VHr-VLb</td>
<td>603.789</td>
<td>240.236</td>
</tr>
<tr>
<td>Flu_Lr-Lb</td>
<td>926.360</td>
<td>3.691</td>
</tr>
<tr>
<td>Flu_Lr-Hb</td>
<td>966.328</td>
<td>3.797</td>
</tr>
<tr>
<td>Flu_Mr-Lb</td>
<td>8591.165</td>
<td>30.355</td>
</tr>
<tr>
<td>Flu_Mr-Hb</td>
<td>8957.175</td>
<td>32.018</td>
</tr>
<tr>
<td>Flu_Hr-Lb</td>
<td>536.091</td>
<td>5.702</td>
</tr>
<tr>
<td>Flu_Hr-Hb</td>
<td>591.955</td>
<td>12.795</td>
</tr>
<tr>
<td>eBird_Lr-Lb</td>
<td>396.811</td>
<td>147.951</td>
</tr>
<tr>
<td>eBird_Lr-Hb</td>
<td>6969.187</td>
<td>1897.051</td>
</tr>
<tr>
<td>eBird_Hr-Lb</td>
<td>8373.273</td>
<td>3226.016</td>
</tr>
<tr>
<td>eBird_Hr-Hb</td>
<td>34577.745</td>
<td></td>
</tr>
</tbody>
</table>

Clearly, PB-SYM is the algorithm to make parallel.
Outline

1. Space Time Kernel Density
2. Sequential Algorithms
3. Domain-Based Parallelism
4. Point-Based Parallelism
5. Conclusion
Domain Replication PB-SYM-DR

Each worker:

- Initialize its own memory buffer
- Process some points in its own buffer (with load balancing)
- Participate in reducing the result
Why is DR bad? Some instances have low computation!

(and some run out of memory)
Decompose the domain in $K \times K$ subdomains

Each worker process different subdomains (load balanced on the subdomains)
Why is DD bad? Work overhead. Some cylinders are cut!

Does anyone know a cheap way to partition better? Some structures admit dynamic programming.
Outline

1. Space Time Kernel Density
2. Sequential Algorithms
3. Domain-Based Parallelism
4. Point-Based Parallelism
5. Conclusion
Point decomposition PB-SYM-PD

- Partition the points in a regular AxBxC grid such that each dimension is bigger than the bandwidth.
- For \((a, b, c) \in \{0, 1\}^3\)
  - Do in parallel grids \(2i + a, 2j + b, 2k + c, \forall i, j, k\)
Why is this bad? Too many dependencies?

Since all $2i, 2j, 2k$ are done before any $2i + 1, 2j, 2k$, there is a forced precedence of $0, 0, 0$ over $3, 0, 0$. But they are not dependent. This does coloring, instead of doing scheduling.

Building the graph from a coloring is simple (and easily expressed in OpenMP 4.0).
A better coloring with PD-SYM-PD-SCHED

We don’t need to color subdomains to minimize the number of colors. We need a coloring that minimizes the longest chain in the implied graph. Heuristic: greedily color subdomains in highest number of points first.

(If you don’t have a good eye, it is a bit better than before)
Why is it bad? Still too long critical path!

How hard is the coloring/edge-orientation problem to minimize critical path?
- NP-hard in general graph (harder than coloring)
- Trivial on chains
- Other?

How good is the heuristic?
I need to think more about this...

Erik Saule (UNC Charlotte)
One can replicate a subdomain and get perfect work parallelism. (at the expense of some memory initialization and reduction.) Does that sound like moldable DAG scheduling? (Actually a bit more complicated than that, but close enough.)

Heuristic: for all path longer than $\frac{n}{2P}$, add one copy to all tasks on the path.
Outline

1. Space Time Kernel Density
2. Sequential Algorithms
3. Domain-Based Parallelism
4. Point-Based Parallelism
5. Conclusion
Future Works

Other platforms:
- GPU (not quite sure how to approach it)
- KNL
- Distributed Memory

Some algorithmic problems:
- Better way to decompose for PB-SYM-DD
- Formally study the edge orientation problem for PB-SYM-DD-SCHED
- Look deeper into the moldable scheduling connection for PB-SYM-PD-SCHED-REP
- Model everything and derive analytical bounds on performance
Food for thoughts!

- Scheduling makes me think about parallel problems differently.
- Scheduling helps me understand performance.
- Maybe the real problem is to figure which instance of the problem to give to the scheduler.
- Maybe the scheduling algorithm is not always what is most important.
Thank you!

And thanks:

- Dan Janies for pointing out the Flu dataset
- Bora Uçar for pointing out the Gallai-Hasse-Roy-Vitaver theorem
- The US tax payer
  - Support from US NSF XSEDE Supercomputing Resource Allocation (SES170007) is acknowledged.
  - This material is based upon work supported by the National Science Foundation under Grant No. 1652442.

Want to know more?

Contact: esaule@uncc.edu
Visit: http://webpages.uncc.edu/~esaule