A Parallelized Cartographic Modeling Language (CML)-Based Solution for Flux Footprint Modeling

Michael E. Hodgson, April Hiscox, Shaowen Wang, Babak Behzad, Sara Flecher, Kiumars Soltani, Yan Liu and Anand Padmanabhan
• What are the sources of observed air samples (e.g. ammonia)?
• In what proportions are the source contributions?
• What is the level of uncertainty of each source?
Location of Tower and Receptor
Field Measurements

• Continual Monitoring
  – Weekly 12 hour air concentrations of NH$_4$ (daytime and nighttime)
  – Wind speed and direction (10$_{hz}$ sampling)
  – Temperature and relative humidity
  – Net radiation
Measurements

**AQ Wind Vane:**
- Wind speed and direction
- Two heights

**Denuders:**
- NH₄ Concentrations
- Two heights

**3D Sonic Anemometers:**
- Turbulence Data
- Two heights

Photos taken by Sara Flecher
Modeling Problem

• 12 hour samples of NH$_4$ concentrations$\rightarrow$ where did the samples come from?
  
The NH$_4$ sample over a 12-hour period is a product of the atmospheric factors over this period of time.

• How much is from crop of interest (e.g. sugarcane) and how much is from other places (like cows)

• **Solution:** Design a model, implement, and process 10$_{hz}$ data over a 12-hour period.
Footprint

- Footprint models are used in the micrometeorological community to determine what the atmospheric sensor “sees”.

The footprint is analogous to the “field of view” (Schmid, 2002) of the sensor in airborne/satellite remote sensing. However, in remote sensing the contribution for the observation is ~instantaneous while many atmospheric observations (ammonia concentration) are for a long sample period (e.g. 12 hours).
The “Footprint”

- Footprint in **micrometeorological community**

The typical ‘footprint’ spread function in micrometeorology is often much less peaked than in remote sensing...

...and is influenced by the dynamic 3-d atmospheric environment.

- Footprint in **airborne/satellite remote sensing**

The ‘footprint’ in remote sensing is actually better represented as a Gaussian function than a rectangle.
One-Dimensional Flux Footprint Model
(Horst and Weil, 1992)

\[ \Phi = A \beta \left[ \frac{Z_m}{\beta \bar{Z}} \right] e^{-\left(\frac{Z_m}{b \beta \bar{Z}}\right)r} \]

- \( Z_m \) = measurement height of instrument
- \( \bar{Z} = f(\text{downwind distance}) \)

- **How would this look in a GIS-based script model?**

- **Could we implement for large data sets using parallel computing?**
What is the Cartographic Modeling Language (CML) Framework?

- Common language for spatial modeling in environmental applications
- Widely accepted in the GIS analyst community
- Widely implemented in the GIS Software Industry
- Common approach used in education.
CyberGIS HPC Solution?

Barriers to Entry

• **Software** (operators, models, UI/UE)
  Observation: The model I want to use does not exist in an HPC form.

• **Infrastructure**
  Observation: Gateway ‘provides’ infrastructure

• **Knowledge/Community**
  Observation: my knowledge from the early 1990s is outdated
A Parallel CML Framework?

• A CML modeling language compatible with parallel computing architecture
  (e.g. includes a function call for distributing spatial, temporal, other parts of the problem to multiple processors)

• A Parallel CML does not exist

• What would it look like?

• What is the required knowledge?

• How can we get there?
From CML to Parallel CML: Required Analyst Education

- Parallel Architecture Fundamentals
- Spatial Independence
- Temporal Independence

GIS Environment

CML Model in python

CML –like Model in x

Parallel CML Model

1 CPU
Exploring Implementation Options

• Different computational modalities studied
  – Multi-processing python library
  – OpenMP-based C implementation

• Different modalities enable parallel CML implementation to adapt to different available hardware resource and software environment

• Optimization and parallelization strategies identified and implemented
Initial Implementation

- Language: Python
- Local operations
  - +, -, *, /, exp, con, sin
- Focal operations
  - Distance
  - Direction
- Problem size
  - Raster: 600 x 600
  - Temporal:
    - 1 day of measurements at frequency of 0.1 second. 864,000 in total
- Initial results on performance
  - 100 measurements took 529.51 seconds. The whole dataset? 53 days
Optimization Techniques

• Identifying loop invariants
  – Euclidean distance and direction vector calculation independent of loop

• Summarizing vector operations
  – Vector operations can be expensive
  – We were able to summarize multiple vector operations into one
Parallelization Strategies

• Spatial dimension:
  – Parallelize focal direction and distance matrix calculation

• Temporal dimension
  – Temporal loop Parallelization

• Parallel CML operators/functions are exposed as library
  – User CML program can be executed without modification

• Providing basic principles for further development
  – Performance profiling
  – Identification of parallelisms
  – Integration with original CML scripting language
Python Parallelization

• Optimization on loop invariant
• Multiprocessing-based parallelization
  – Master-slave model with result management

Data size: 86,400 measurements
Python Parallelization with Load Balancing Strategy

- Static scheduling led to load imbalance as the number of cores increases.
- Dynamic scheduling was implemented as cycle-stealing to keep every core busy.

![Graph showing comparison between new and old code performance with data size of 864,000 measurements.]
Interpreted vs Compiled

• Observation on python parallelization effort
  – The python code has been optimized and parallelized, but is not good enough to process the whole dataset efficiently
  – Python is a scripting language

• C version development
  – Explore the difference on performance between interpreted and compiled language implementations
  – Optimization: reducing the number of temporary layers used as intermediate results
  – Parallelization: OpenMP
## Performance Comparison

<table>
<thead>
<tr>
<th>Number of cores</th>
<th>Python with Multiprocessing</th>
<th>C with OpenMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>17263</td>
<td>1134.169</td>
</tr>
<tr>
<td>32</td>
<td>9515</td>
<td>690.254</td>
</tr>
<tr>
<td>64</td>
<td>5937</td>
<td>493.950</td>
</tr>
<tr>
<td>128</td>
<td>4148</td>
<td>251.200</td>
</tr>
</tbody>
</table>

Data size: 864,000 measurements
Scalability of OpenMP Implementation

Data size: 864,000 measurements
## Performance Results

<table>
<thead>
<tr>
<th>Dataset Size</th>
<th>Original (Sequential)</th>
<th>With Optimizations (Sequential)</th>
<th>Multi-processing Python (Parallel - 128 cores)</th>
<th>OpenMP (Parallel - 128 cores)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>529.51</td>
<td>20.69</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>86,400</td>
<td>NA¹</td>
<td>13433.72</td>
<td>629.31</td>
<td>26.478</td>
</tr>
<tr>
<td>864,000</td>
<td>NA¹</td>
<td>NA¹</td>
<td>5549.19</td>
<td>251.200</td>
</tr>
</tbody>
</table>

Table: Comparing performance of parallel and optimization strategies (Time in seconds)

Notes:
¹ Execution times not available because model did not run to completion in reasonable time
Load balanced evenly among cores
Parallel CML Operators/Functions

• Parallelization of individual CML operators/functions
  – Focal distance
  – Focal direction
• Implemented in C using OpenMP
• Python binding
  – Python-C interface using SWIG
  – Parallel CML operators/functions as linked library
  – Transparent use of parallel CML operators/functions
    • No modification to CML program is needed
• Demo
Performance of Euclidean Distance Operator

![Graph showing computation time versus number of cores for different resolutions (8K x 8K, 32K x 32K, 64K x 64K).](image)

- **8K x 8K**
- **32K x 32K**
- **64K x 64K**
One-Dimensional Modeled Results
Future Work

• Micro-meteorology research
  – Integration of Ammonia Observation with Probability Surface
  – Linkage of Temporal Probability Surface with Land Cover Change

• Computational science research
  – Computational intensity analysis on CML operators/functions
  – User interface/user experience
    • Python
    • GUI-based workflow builder?
  – Parallelization of CML workflow execution
  – Development of strategies to exploit different implementations on hybrid supercomputer architectures
    • Shared-memory machines, clusters, GPU, data-oriented computing platform (Hadoop)