Parallel computing with IPython: an application to air pollution modeling

Josh Hemann, Rogue Wave Software, University of Colorado

Brian Granger, IPython Project, Cal Poly, San Luis Obispo
Outline

• IPython? Parallel Computing? I thought it was an interactive shell?
• An example application.
IPython Overview

• Goal: provide an efficient environment for exploratory and interactive scientific computing.
• Cross platform and open source (BSD).
• Two main components:
  o An enhanced interactive Python shell.
  o A framework for interactive parallel computing.
IPython's Parallel Framework

- Goal: provide a high level interface for executing Python code in parallel on everything: multicore CPUs, clusters, supercomputers and the cloud.
- Easy things should be easy, difficult things possible.
- Make parallel computing collaborative, interactive.
- A dynamic process model for fault tolerance and load balancing.
- Want to keep the benefit of traditional approaches:
  - Integrate with threads/MPI if desired.
  - Integrate with compiled, parallel C/C++/Fortran codes.
- Support different types of parallelism.
- Based on processes not threads (the GIL).
- Why parallel computing in IPython?
  - R(EEEEEE...)PL is the same as REPL if abstracted properly.
• Python code as strings
• Functions
• Python objects
Architecture details

- The IPython Engine is a Python interpreter that executes code received over a network.
- The Controller maintains a registry of the engines and a queue for code to be run on each engine. Handles load balancing.
- Dynamic and fault tolerant: Engines can come and go at any time.
- The Client is used in top-level code to submit tasks to the controller/engines.
- Client, Controller and Engines are fully asynchronous.
- Remote exception handling: exceptions on the engines are serialized and returned to the client.
- Everything is interactive, even on a supercomputer or the cloud.
MultiEngineClient and TaskClient

- **MultiEngineClient**
  - Provides direct, explicit access to each Engine.
  - Each Engine has an id.
  - Full integration with MPI (MPI rank == id).
  - No load balancing.
- **TaskClient**
  - No information about number of Engines or their identities.
  - Dynamic load balanced queue.
  - No MPI integration.
- **Extensible**
  - Possible to add new interfaces (Map/Reduce).
  - Not easy, but we hope to fix that.
Job Scheduler Support

To perform a parallel computation with IPython, you need to start 1 Controller and N Engines. IPython has an `ipcluster` command that completely automates this process. We have support for the following batch systems.

- PBS
- ssh
- mpiexe/mpirun
- SGE (coming soon)
- Microsoft HPC Server 2008 (coming soon)
Work in progress

- Much of our current work is being enabled by ØMQ/PyØMQ. See SciPy talk tomorrow and www.zeromq.org
- Massive refactoring of the IPython core to a two process model (frontend+kernel). This will enable the creation of long awaited GUI/Web frontends for IPython.
- Working heavily on performance and scalability of parallel computing framework.
- Simplifying the MultiEngineClient and TaskClient interfaces.
How is IPython used

???
The views expressed in this poster are those of the authors and do not necessarily reflect the views or policies of the U.S. Environmental Protection Agency.
The next ~10 minutes...

- Air pollution modeling - quick background
- Current software used for modeling
- Better and faster software with Python, PyIMSL
- Even faster software with IPython
- Likely parallelization pain points for newbies (like me)
Size Distribution of Airborne PM

- **ultrafine**
- **fine**
- **coarse**
- **combustion & atmospheric reactions**
- **mechanically generated**

Size (diameter in μm)

0.02  0.1  2.5  10
PM$_{2.5}$ induced health effects

Increases in ozone not associated with increases in death.

Increases in PM$_{2.5}$ are associated with increases in death.

Results from the 6-cities cohort study.

What are the sources of the pollution?

Receptor Models

\[ x_{ij} = \sum_{k=1}^{p} g_{ik} f_{jk} + e_{ij} \]

- \( x_{ij} \): species \( j \) measured in sample \( i \) (ambient measurements)
- \( g_{ik} \): mass fraction of species \( j \) from source \( k \) (source profile or fingerprint)
- \( f_{jk} \): mass from source \( k \) in sample \( i \) (source contribution)
- \( e_{ij} \): residual
- \( Q = \sum_{i=1}^{m} \sum_{j=1}^{n} \left( \frac{e_{ij}}{S_{ij}} \right)^2 \)

To solve, we find minimum of ...
Show a factor profile and contribution plot.
EPA Positive Matrix Factorization (PMF) 3.0 Model

Receptor models provide scientific support for current ambient air quality standards and for implementation of those standards by identifying and quantifying contributions for source apportionment. Ambient air quality data sets have been improving greatly due to more species being measured, species being stratified by particle size, and shorter durations of sampling. Receptor model algorithms have also improved greatly to take advantage of these higher quality data sets.

To ensure that receptor modeling tools are available for use in the development and implementation of air quality standards, the United States Environmental Protection Agency’s Office of Research and Development (ORD) has continued to develop a suite of receptor modeling tools that are freely distributed to the air quality management community.

EPA PMF is one of the receptor models that ORD has developed. The user provides a file of sample species concentrations and uncertainties which the model uses to calculate the number of source types, profiles, relative contributions, and a time series of contributions. The algorithms used in EPA PMF model to compute profiles and contributions have been peer reviewed by leading scientists in the air quality management community and have been certified to be scientifically robust. The PMF 3.0 Fundamentals & User Guide provides details on how PMF is implemented as well as references.

EPA PMF software program is a stand-alone product, meaning that it requires no other software and is self-contained. There is no need for a license and the program is available free of charge. The software is menu-driven, using Graphical User Interfaces, and therefore eases inputting of data, generating, evaluating, and exporting of results.

System Requirements:
The EPA PMF 3.0 model works on Windows XP and Windows Vista. The computer should have at least a 2.0 GHz processor and at least 1 GB of memory. In addition a 800x600 pixel display is recommended.

IMPORTANT: PLEASE REGISTER FIRST (link below) before using the software.

Registration Form

Comments or questions on EPA PMF 3.0 or problems with the comments that can be considered for future development.

Product available for download:

EPA PMF 3.0 Software (57.6 MB)
EPA PMF 3.0 Fundamentals & User Guide (PDF, 81 pages)
The U.S. EPA is currently working on EPA PMF 3.1. This version will refine the solution. The updated version will be posted to ensure that the most recent version of the model is used in NERL RM Support Team.

EPA PMF v3.0.2.2

File View Action Help

Input/Output Files
Analyze Input Data Model Execution Base Model Results Bootstrap Model Results Speak Model Results

Input Files
Model input data is in tab-delimited [*.tsv], comma-separated value [*.csv], or Excel workbook [*.xlsx] format. Species names in first row, units in second row (optional), and date/times in first column (optional).

Concentration Data File
Browse

Concentration data table with parameter names in the first row. Optionally, the second row may contain units and the first column may contain date/time.

Uncertainty Data File
Browse

Observation-based or equation-based uncertainty values for each sample. Must match concentration data format.

Missing Value Indicator
888
Exclude Entire Sample Replace Missing Values with Species Mean

Output Files
Output Folder
Browse

Specify a destination folder for all model run output files.

Output File Type
Tab Delimited Text (*.tsv) Comma Delimited Text (*.csv) Excel Workbook (*.xlsx)

Program Configuration
Configuration File
Browse

Enter or browse to a configuration file.
Source Apportionment Tool

### Base

<table>
<thead>
<tr>
<th>Day 1</th>
<th>Day 2</th>
<th>Day 3</th>
<th>Day 4</th>
<th>Day 5</th>
<th>Day 6</th>
<th>Day 7</th>
<th>Day 8</th>
<th>Day 9</th>
<th>Day 10</th>
</tr>
</thead>
</table>

Used EPA PMF3.0 to determine best “Base Case”, which includes number of factors.

#### Average Apportioned Mass

- **Factor 1**
  - **Factor 2**
- **Factor 1**
  - **Factor 2**
  - **Factor 3**
- **Factor 1**
  - **Factor 4**
  - **Factor 2**
  - **Factor 3**
Source Apportionment Tool

<table>
<thead>
<tr>
<th>Base</th>
<th>Day 1</th>
<th>Day 2</th>
<th>Day 3</th>
<th>Day 4</th>
<th>Day 5</th>
<th>Day 6</th>
<th>Day 7</th>
<th>Day 8</th>
<th>Day 9</th>
<th>Day 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bootstrap 1</td>
<td>Day 9</td>
<td>Day 10</td>
<td>Day 3</td>
<td>Day 1</td>
<td>Day 4</td>
<td>Day 7</td>
<td>Day 3</td>
<td>Day 6</td>
<td>Day 3</td>
<td>Day 7</td>
</tr>
<tr>
<td>Bootstrap 99</td>
<td>Day 3</td>
<td>Day 5</td>
<td>Day 3</td>
<td>Day 6</td>
<td>Day 7</td>
<td>Day 1</td>
<td>Day 5</td>
<td>Day 5</td>
<td>Day 1</td>
<td>Day 2</td>
</tr>
</tbody>
</table>

Created our own tool using Python and PyLMSL Studio

Critical to success of the bootstrap ...
Matching factor 4 from bootstrap 86 to a factor in the Base Case
1. Use Non-negative matrix factorization to factorize measurement matrix $X$ into $G$ (factor scores) and $F$ (factor scores). This will be the "base case" model.

2. Block bootstrap resample measurement days (rows) in $X$ to yield $X^*$

3. Factorize $X^*$ into $G^*, F^*$

4. Use neural network or naive Bayes classifier to match factors in $G^*$ and $F^*$ with base case $G$ and $F$ (i.e. sort the columns of $G^*$ and the rows of $F^*$ such that factor $i$ always corresponds to the same column/row index in any given $G/F$ matrix)

5. Repeat steps 2. through 4. 1,000 times

6. With pile of $F$ and $G$ matrices, compute descriptive statistics for each element, generate visualizations, etc
Factor 1 Profile Variability

Factor 1 Timeseries Variability

- 5th–95th Percentile Range
- Interquartile Range
- Basecase Contribution

Normalized Proportion

Factor Contribution, ng/m³

Time (Feb to Dec)
How long does this modeling take to run?

1,000 bootstrap replications on my dual-core laptop...

• EPA PMF 3.0
  ~ 1 hour and 45 minutes
  – Black box, only single core/processor actually used

• Python and PyIMSL Studio
  ~ 30 minutes
  – MKL and OpenMP-enabled analytics means I don’t have to do anything to use both of my cores, It Just Works

Can we make this faster?
from IPython.kernel import client

# Set up each Python session on the clients...
mec = client.MultiEngineClient(profile='DASH')
mec.execute('import os')
mec.execute('import shutil')
mec.execute('import socket')
mec.execute('import parallelBlock')
mec.execute('reload(parallelBlock)')
mec.execute('from parallelBlock import parallelBlock')

# Task farm-out the 6 analysis steps...
tc = client.TaskClient(profile='DASH')
numReps = 1000

# Python server side...
taskIDs = []
for rep in xrange(1, numReps):
    t = client.MapTask(parallelBlock, args=[rep])
    taskIDs.append(tc.run(t))
tc.barrier(taskIDs)

results_list = [tc.get_task_result(tid) for tid in taskIDs]

for task, result in enumerate(results_list):
    # Unpack results from each iteration and do analysis/visualization
A Solution Architecture

- EPA PMF 3.0
  ~ 1 hour and 45 minutes

- Python/PyIMSL
  Studio/IPython/MS HPC cluster, 6 engines, 48 cores
  ~4 minutes
  - Actual bootstrap simulation takes ~40 seconds, the rest is pre and post processing that only happens on head node

EPA PMF 3.0
- MATLAB-based pollution source apportionment tool
What make parallelizing hard...

There are complex aspects of my application that have nothing to do with cool mathematics...

- Existing application used for a couple of years, not written with parallelization in mind from the start
- Analytics are not just simple calls to pure Python
  - PyIMSL algorithms wrap ctypes objects that sometimes involve C structures (that may contain other complex types), not just simple data types
  - 3rd party Fortran 77 dll called
    - Does it's own file I/O, which is critically important to read, but for which I have little control of (with respect to file names and paths)
- Big time sink is in post-processing of results to set up data for visualization, a whole separate aspect not related to the core analysis
Gotchas...

- Portability of code
  - Not everyone has the newest IPython and the dependencies needed for the parallel extensions or to run on MS HPC Server. How can code be written to automatically take advantage of multiple cores/processors, but always work in the "degenerate" case?

- Pickle-abilitynessitude
  - If it can't be pickled it can't be passed between the main script and the engines
  - Send as little as possible between the engines
    - Implies having local code to import, data to read/write, and licenses on each engine, which means duplicated files, more involved system admin of nodes, etc...

- File I/O
  - Make sure files written out on a given engine are found by that same engine in subsequent work $\implies$ keeping certain analysis steps coupled

- Local file systems
  - `shutil.move` to force flush of 3rd party file output (race conditions?)
  - Windows registry hack needed if you want to cmd.exe to be able to use UNC paths
• Debugging and diagnostics
  o Sanity checking and introspection can be more involved

```python
def parallelBlock(rep):
  ini_file = 'pmf_bootstrap.ini'
  fh = open("NUL", "w")
  subprocess.Popen('pmf2wopt.exe %s %i' % (ini_file, rep), stdout=fh).communicate()
  hostname = socket.gethostname()
  try:
    #Analysis steps. Nothing to do if PMF did not
    #converge for this bootstrap replication...
    except:
      return (-id, hostname, [ ], [ ], [ ], [ ], [ ])
  except:
    return (id, hostname, v, w, x, y, z)
```
I'm happy to talk outside of this presentation!

josh.hemann@roguewave.com