Writing a Custom RPython-based DSL Interpreter to Solve PDEs: Experience and Performance

Sean Fisk, Grand Valley State University
Advisor: Davide Del Vento, NCAR
Huh?

- **DSL:** domain-specific language
  - Often a programming language
- **PDE:** partial differential equation
  - Relates rate of change to continuous variables
  - Can be solved analytically or numerically
- **RPython**
  - A specialized version of Python used for writing interpreters
Goal

Design and build a simple programming language to implement finite difference using stencils.
Finite Difference

• A way to solve partial differential equations
• Frequently used in weather simulations
• Specify a grid, a transformation matrix (stencil), and time step
Stencils – A Simple Example

Matrix to transform (M)

\[
\begin{array}{cccc}
M_{0,0} & M_{0,1} & M_{0,2} & M_{0,3} & M_{0,4} \\
M_{1,0} & M_{1,1} & M_{1,2} & M_{1,3} & M_{1,4} \\
M_{2,0} & M_{2,1} & M_{2,2} & M_{2,3} & M_{2,4} \\
M_{3,0} & M_{3,1} & M_{3,2} & M_{3,3} & M_{3,4} \\
M_{4,0} & M_{4,1} & M_{4,2} & M_{4,3} & M_{4,4} \\
\end{array}
\]

Stencil (S)

\[
\begin{array}{ccc}
S_a & S_b & S_c \\
S_d & S_e & S_f \\
S_g & S_h & S_i \\
\end{array}
\]

Transformed Matrix

\[
M'_{1,1} = M_{1,1} + M_{0,0} \times S_a + ... + M_{1,1} \times S_e + ... + M_{2,2} \times S_i
\]

Other cells use a wrap-around.
# Stencils – End Result

<table>
<thead>
<tr>
<th>Matrix to transform</th>
<th>Stencil</th>
<th>Transformed matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>6       -9       10</td>
<td>1       1       1</td>
<td>17       -13      25</td>
</tr>
<tr>
<td>-9      -9       4</td>
<td>1       2       1</td>
<td>-17      -17      9</td>
</tr>
<tr>
<td>9       5        -6</td>
<td>1       1       1</td>
<td>24       16       -6</td>
</tr>
<tr>
<td>3       0        9</td>
<td></td>
<td>33       27       45</td>
</tr>
</tbody>
</table>
# Stencil Language

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>STO $R_x \ V$</td>
<td>Store a value in register $R_x$</td>
</tr>
<tr>
<td>ADD $R_x \ V$</td>
<td>Add value $V$ to register $R_x$</td>
</tr>
<tr>
<td>PR $R_x$</td>
<td>Print value in register $R_x$</td>
</tr>
<tr>
<td>CMX $M_x \ A \ B$</td>
<td>Create matrix $M_x$ with dimension $(A, B)$</td>
</tr>
<tr>
<td>SMX $M_x \ ...$</td>
<td>Set values of $M_x$ with $A \times B$ arguments</td>
</tr>
<tr>
<td>SMXF $M_x \ “filename”$</td>
<td>Set values of $M_x$ from file “filename”</td>
</tr>
<tr>
<td>PMX $M_x$</td>
<td>Print matrix $M_x$</td>
</tr>
<tr>
<td>PDE $M_x \ M_y$</td>
<td>Apply stencil $M_x$ to $M_y$ and store result in $M_y$</td>
</tr>
<tr>
<td>BNE $R_x \ V \ L$</td>
<td>Branch to relative location $L$ if $R_x \neq V$</td>
</tr>
</tbody>
</table>

- Registers use integers
- Matrices use real numbers (floats)
- Matrices must have odd dimensions
Sample Program

# Create matrices
CMX 0 3 3
SMXF 0 "sample-programs/matrices/stencil"
CMX 1 3 3
SMXF 1 "sample-programs/matrices/psi"

# Run main PDE loop
STO 0 0
ADD 0 1
PDE 0 1
BNE 0 12 -2
PMX 1
PyPy

- “Python in Python”
- Continuation of Psyco, a just-in-time compiler for Python
- Umbrella project
  - **RPython**: a toolchain for writing interpreted languages
  - **PyPy**: an implementation of Python in RPython
  - **Just-in-time compiler**: runtime optimization
RPython - Specification

- RPython is not Python!
- Restricted subset of Python
- Statically typed, but types are not specified
- UnionError
- Use of inheritance
RPython – What is it?

- Whatever the PyPy developers decide to implement.
RPython – JIT Compilation

- **JIT**: just-in-time compilation
- Dynamically re-compiles bytecode to machine code at runtime
- Can be faster than statically compiled code
- RPython generates JIT compilers
Donor Cell Algorithm

- A transport algorithm
- Can be implemented as repetitive application of a stencil
- A simplification of the MPDATA algorithm (no correction)
Performance – Comparison

Donor Cell Algorithm - Performance

- C++ (g++)
- Fortran (gfortran)
- Python (PyPy 1.9)
- Python (CPython 2.7.5)
- Translated RPython Stencil Language with JIT
- Translated RPython Stencil Language
Performance – Why is it slower?

- Slower than C++/Fortran
  - Has to interpret a language
- Slower than Python
  - Python is more optimized
  - Python uses NumPy (interfaces with C/Fortran)
Software Development

- Almost all code was developed using strict Test-Driven Development
- Stencil language has 142 individually-written tests
- Donor cell implementation runs 632 test cases against existing implementation
- Tools
  - pytest
  - mock
  - tox
  - flake8 (pep8 + pyflakes)
References

Future Ideas

• Further JIT compiler generation
• Parallelization
  – MPI
  – CUDA
• Other algorithms
  – Shallow water
Contact

• Find the project online
  - https://github.com/seanfisk/rpython-stencil-language

• Contact me
  - <sean@seanfisk.com>