Update: CLM Refactoring Efforts

LMWG Meeting
Boulder, Colorado
Tuesday Feb. 25 2014

Stefan Muszala, NCAR
Benjamin Andre, LBL
Forrest Hoffman, ORNL
Erik Kluzek, NCAR
David Lawrence, NCAR
Bill Sacks, NCAR
Mariana Vertenstein, NCAR
Update: CLM Refactoring Efforts

- Refactoring and development process
- Recent CLM refactoring efforts
- Benefits of systematic and continued refactoring
Refactoring: The process of improving the internal structure of software while maintaining the same external behavior

\[ \frac{y}{x} = 5x + 2x + 3xz \]
\[ y = x^2(7 + 3z) \]

Current CLM and CESM bit-for-bit testing

\[ A(\overline{A} + B) \]
\[ A\overline{A} + AB \]
\[ 0 + AB \]
\[ AB \]
Extract Constant

! before
x = y/2 * 299792458
! After

! Speed of light in a vacuum (m/s)
real, parameter :: C = 299792458
...
x = y/2 * C
Extract Expression

! before
do j = 2,N
  ...  
  var(j) = \textcolor{red}{2*y/z} \times \textcolor{red}{var(j-1)}
  ...
enddo

! after
! calculate x outside of loop
x = \textcolor{red}{2*y/z}
do j = 2,N
  ...  
  var(j) = \textcolor{red}{x} \times \textcolor{red}{var(j-1)}
  ...
enddo
CLM development process

time, t (days, months, years)

Idea
- New parameterization
- Algorithm
- Faster solver
- More accurate solver

Prototype
- Do basic principles make sense?
  - Whiteboard
  - Matlab
  - IDL, NCL
  - Excel
  - Mathematica
  - High Level PL
  - Model Framework

Basic Functionality in CLM
- Can it work in the current system?
- What needs changing?
- Can it work in single point first?
- Global res.? High res.?

Technical and Scientific Correctness
- No memory issues
- Restarts work
- No water, energy, mass imbalances
- Numerical stability
- End cases are stable

Maintenance and Testing
- Works 99% of the time, use as a baseline
- 80% code coverage at Basic Block level

2/25/14
Refactor for:

**Efficiency**
- Energy
- Human resources
- Time to completion
- Scaling

**Clarity**
- Easy to read
- Easy to follow
- Coding standards

**Organization**
- Flexible
- Extensible

**Robustness**
- Compiler ports
- Hardware ports

**Testing**
- Code coverage
- Lines of code
Memory access reordering for dynamic landunits

*Bill Sacks*

<table>
<thead>
<tr>
<th></th>
<th>Old memory access</th>
<th>New memory access</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Grid cell</strong></td>
<td>1 2 1 2 2 2</td>
<td>1 2 1 2 1 2</td>
</tr>
<tr>
<td><strong>Landunit</strong></td>
<td>1 2 3 1 2 3</td>
<td>1 1 2 2 3 3</td>
</tr>
</tbody>
</table>

24% performance improvement
CLM initialization refactor

Mariana Vertenstein

Interpinic is now an optional part of online initialization

• Now possible to interpolate from any initial data set to desired output resolution
• Runs in parallel
• Eliminates the need for cold start (still an option if desired)

Module initialization refactor

• Each functionality (CN, CNDV, CH4) has own initialization routine
• Can set values up for cold-start
• Those values may then be overwritten with values from interpinic
PTCLM refactor

\textit{Erik Kluzek}

- US-UMB fix in scripts and DATM update
- Build-namelist aborts if inconsistency between CLM\_BLDNML\_OPTS and user\_nl\_clm
- Add CO2 streams as DATM option
- Deprecating pts. mode

build-namelist refactor

\textit{Ben Andre}

- Refactor to modularize
- Add unit-testing capability
- Addresses code-coverage of ‘scripts’ portion of CLM
### Other refactoring highlights in the last 6 months

<table>
<thead>
<tr>
<th>Category</th>
<th>Highlights</th>
</tr>
</thead>
</table>
| Remove preprocessor definitions               | • One binary to compile  
• Less time from compilation to simulation |
| Remove hard coded parameters                  | • Uncertainty quantification via parameter mods  
• Increased modularity                                |
| Modify array indexing                         | • OpenMP fix in CLM45  
• Easier to read                                                                 |
| Interface redesign (procedure and function)   | • Fewer pointers increase compiler optimization  
• Easier debugging  
• Unit testing                                            |
| Code Generation                               | • Leverage work in Pio  
• Now have ncdio.F90.in -> ncdio.F90                                                    |
Code Coverage – ICLM45CN – April 2013

- Tool to do differential coverage between an arbitrary N coverage runs
- Time to do the analysis initially, then analyze continually (quarterly, every Mth tag, etc…)
- Each component should be around 80% Basic Block coverage, then deal with coupled cases.
Benefits of refactoring – Lines of F90 are decreasing while model functionality is increasing

..models/Ind/clm

<table>
<thead>
<tr>
<th></th>
<th>F90</th>
<th>perl</th>
<th>xml</th>
<th>sh</th>
</tr>
</thead>
<tbody>
<tr>
<td>clm4_0_00</td>
<td>117763</td>
<td>1451</td>
<td>7387</td>
<td>2599</td>
</tr>
<tr>
<td>clm4_5_00</td>
<td>241063</td>
<td>5300</td>
<td>22824</td>
<td>3085</td>
</tr>
<tr>
<td>clm4_5_64</td>
<td>229938</td>
<td>6107</td>
<td>22244</td>
<td>2747</td>
</tr>
</tbody>
</table>

..models

<table>
<thead>
<tr>
<th></th>
<th>F90</th>
<th>perl</th>
<th>xml</th>
<th>sh</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>clm4_0_00</td>
<td>275097</td>
<td>1667</td>
<td>12704</td>
<td>2599</td>
<td>8717</td>
</tr>
<tr>
<td>clm4_5_00</td>
<td>550534</td>
<td>5853</td>
<td>33998</td>
<td>12756</td>
<td>17449</td>
</tr>
<tr>
<td>clm4_5_64</td>
<td>540424</td>
<td>6662</td>
<td>34065</td>
<td>12418</td>
<td>18098</td>
</tr>
</tbody>
</table>

find . -name "*.sh" -exec wc -l {} \; | awk '{total = total + $1}END{print total}'
- CLM tag planning

https://trello.com/b/yzLcXkAx/cesm-clm-and-rtm-development

- Start page for LMWG developers guidelines

http://www2.cesm.ucar.edu/working-groups/lmwg/developer-guidelines

- Coding conventions for everyday development

https://wiki.ucar.edu/display/ccsm/CLM+Coding+Conventions
Blank slide
Code Coverage:

Type of Structural (White Box) testing – tests internal structure (regardless of application) as opposed to behavioral (Black Box) testing, which tests functionality (tied to application)

What can it cover?
- functions
- statements
- decisions
- conditions
- multiple conditions
- decision/condition
- actual argument values

Why use it?
- Tells you how well your tests exercise your code base.
- Tells you if you need to adjust the number of tests
  - Indirect measure of test quality

Intel’s codecov tool covers the following
  1) lines of source
  2) basic blocks
  3) functions

Problems:
- How well does asm (assembly) translate back to source code?
  - if it’s critical, need to run code-coverage on the asm. (luckily, not relevant in our case).
- Figuring out differential coverage time-consuming