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

NSF
NCAR NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
U.S. DEPARTMENT OF ENERGY Office of Science

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

$$\frac{y}{x} = 5x + 2x + 3xz$$

$$y = x^2(7 + 3z)$$

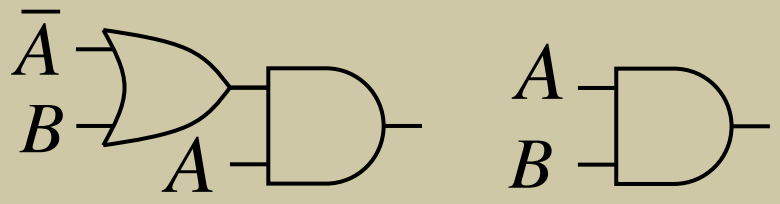
Current CLM and CESM bit-for-bit testing

$$A(\bar{A} + B)$$

$$A\bar{A} + AB$$

$$0 + AB$$

$$AB$$



Extract Constant

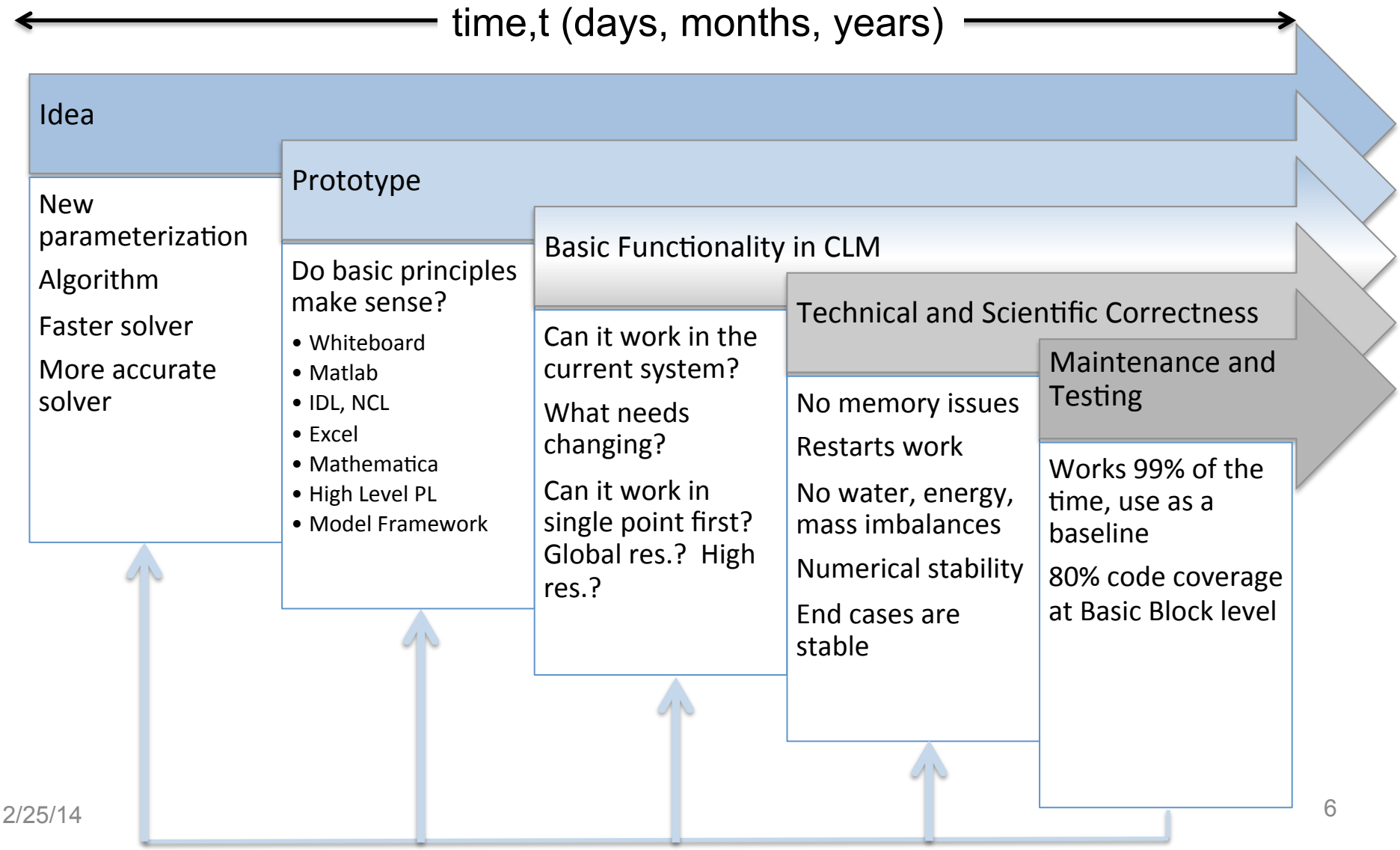
```
! before
x = y/2 * 299792458
! After

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

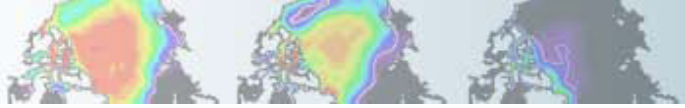
Extract Expression

```
! before
do j = 2,N
  …
  var(j) = 2*y/z * var(j-1)
  …
enddo
```

```
! after
! calculate x outside of loop
x = 2*y/z
do j = 2,N
   …
   var(j) = x * var(j-1)
   …
enddo
```

CLM development process





Refactor for:

Efficiency	Clarity	Organization	Robustness	Testing
Energy	Easy to read	Flexible	Compiler ports	Code coverage
Human resources	Easy to follow	Extensible	Hardware ports	Lines of code
Time to completion	Coding standards			
Scaling				

Memory access reordering for dynamic landunits

Bill Sacks

Old memory access

Grid cell	1	1	1	2	2	2
Landunit	1	2	3	1	2	3

New memory access

Grid cell	1	2	1	2	1	2
Landunit	1	1	2	2	3	3

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

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

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

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

Files				Functions				Blocks			
total	cvrd	uncvrd	cvrg%	total	cvrd	uncvrd	cvrg%	total	cvrd	uncvrd	cvrg%
274	213	61	77.74	2,479	1,120	1,359	45.18	156,731	65,824	90,907	42.00

- 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/lnd/clm

	F90	perl	xml	sh
clm4_0_00	117763	1451	7387	2599
clm4_5_00	241063	5300	22824	3085
clm4_5_64	229938	6107	22244	2747

../models

	F90	perl	xml	sh	C
clm4_0_00	275097	1667	12704	2599	8717
clm4_5_00	550534	5853	33998	12756	17449
clm4_5_64	540424	6662	34065	12418	18098

```
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

Basic Block – portion of code with:
1 entry point (label)
1 exit point (jump to target)
Used by compilers to create a Control Flow Graph (CFG) for
- dependency analysis
- dead code removal
- optimization
- register renaming
- etc…

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