Core Model Proposal #371: Miscellaneous Bug Fixes

Product: Global Change Assessment Model (GCAM)

Institution: Joint Global Change Research Institute (JGCRI)

Authors: Maridee Weber, Page Kyle, Pralit Patel

Reviewers: Ellie Lochner, Matthew Binsted

Date committed: December 19, 2022

IR document number: PNNL-34048

Related sector: various

Type of development: bugfix (C++ code, gcamdata)

Purpose: This Core Model Proposal includes a range of miscellaneous bug fixes and features, including: reassigning gcamusa modules to be emissions modules; fixing some spelling errors; adding a "fixed" final demand; eliminating member variables that were either duplicative or unnecessarily marked STATE; minor solution bug fixes; and eliminating variability in iteration counts from run-to-run even when GCAM Parallel is enabled. Note, while there are many changes aimed at reducing memory usage or improving performance, the benefits in this regard are relatively minor.

Description of Changes

Reassign GCAM-USA modules

There were three GCAM-USA modules introduced in pull request 226 / core model proposal 254 that broke the capability to disable gcamusa modules within gcamdata. These three GCAM-USA modules dealt with emissions, and have been reassigned as follows:

original module name	new module name
module_gcamusa_L169.nonghg_NEI_scaling_USA	module_emissions_L169.nonghg_NEI_scaling
module_gcamusa_L170.nonghg_ceds_scaling_USA	module_emissions_L170.nonghg_ceds_scaling
module_gcamusa_L270.nonghg_nei_to_gcam	module_emissions_L270.nonghg_nei_to_gcam

These name changes restore users' ability to run the **driver()** function in gcamdata, both with the gcamusa modules enabled or disabled (set in constants.R). Image 1 in the validation section shows that this reassignment resulted in no changes to the XMLs (though other components of this proposal do result in XML changes).

Calculate Direct Air CO2 Capture (DAC) capacity

Prior to this proposal, the potential DAC "resources" assigned to each region were read in from an exogenous data input file, inst/extdata/energy/A62.calibration.csv. However, the values in this file were simply computed from the CO2 storage resources in each region multiplied by a scaler calculated from an assumed DAC limit in the USA. Because the A62.calibration.csv file had hard-wired region numbers, it didn't work for model branches with modified country-to-region mappings, where any additional regions would effectively not have DAC included as a mitigation option. This method is shifted so that it is entirely performed within gcamdata, and the A62.calibration.csv file is removed. Breaking out new model regions does not require any DAC-specific adjustments.

Fix technology naming error in buildings calibration file A44.share_serv_fuel.csv

The hydrogen update included in GCAM v6.0 added hydrogen technologies to the global buildings module. These technologies were also added (with zero values) to the historical calibration data table energy/A44.share_serv_fuel.csv. However, their names in the calibration table were mixed up (e.g., residential services within the commercial sector). This mismatch was handled in gcamdata such that the technologies ended up getting named correctly and assigned zero calibration values in the building_det.xml file, so revising their names does not change the resulting XML file constructed, but the revision corrects the incongruous items in the A44.share_serv_fuel.csv file and avoids relying on gcamdata's fortuitous methods of naming technologies and assigning default calibration values that circumvented the error.

Misspellings

This bugfix addresses two typos. The first is in input/policy/forcing_target_6p0.xml, where the policy name was changed from "forcing_4p5" to "forcing_6p0". The second is in input\gcamdata\inst\extdata\aglu for "A_agRegionalSubsector.csv" and "A_agRegionalTechnology.csv", where "dairy" was misspelled.

FixedFinalDemand

The "fixed" final demand is simply an `AFinalDemand` subclass which always just returns the exogenously specified "final demand" values regardless or price or income. This was a feature requested for developing the iron and steel module. We add it here as such a feature is more broadly useful for diagnostics and testing. While the code is added in this proposal, it is not utilized for anything. An example XML is attached which swaps the existing Aluminum final demand with the `FixedFinalDemand`, and results are provided in the Validation section.

Eliminate duplicative or un-needed member variables

Total Land Use Change Emissions

We store the total land use change emissions in addition to the total Above ground land use change emission and Below ground land use change emissions. The total is always calculated as above + below. We noticed one edge case: stopping the model in 1975 and examining the `mTotalEmissions` variable reveals the forward shadow (the **future** uptake / emissions which *would* result from the sum total of all past land use decision) wasn't calculated and won't be until the next model period is run (the above and below were being calculated and reported correctly). Given the `mTotalEmissions` member variable is duplicative and a relatively significant amount of memory, we decided to just remove it and replace all instance where the total was required with above + below.

mReduction in AEmissionsControl

We had been storing mReduction for each emissions control object. The only benefit of storing this data is that it could be reported in the debug.xml file. However, given how many instances of these objects we have and that they need to be marked STATE (performance critical), we decided to avoid saving it.

Add postCalc to NonCO2Emissions

We add a `postCalc` method to `NonCO2Emissions` which allows us to move calibrating emissions coefficients (via input-emissions) and saving emissions control adjusted emissions coefficients to `postCalc` rather than each time the model iterates. This allows us to avoid marking these variables as STATE. In addition, we move the ARRAY of control adjusted emissions coefficients from the NonCO2Emissions object to `LinearControl`, which is where they are utilized. Given there are far fewer `LinearControl` objects instantiated than `NonCO2Emissions` objects, this reduces the total amount of memory used as well.

Node Unmanaged Land Value

We add an initial call to `setUnmangedLandValue` during `initCalc`, which effectively sets `mUnmanagedLandValue` in all the `LandNode` objects and ensures they do not need to get set again during `World.calc`, thus removing the need to mark them as STATE.

Remove carbonIncreaseRate

We remove the `carbonIncreaseRate` input parameter and its associated member variable in `LandAllocator` and `LeadLeaf`, given this feature is rarely used and could be replicated in a more generic way using `price-adjust` on the `CO2_LUC` linked GHG policy object.

Remove Land Expansion Constraint feature

We remove the `landConstraintCurve` feature from the `ALandAllocatorItem` as it is largely duplicative of the more generic `land-constraint-policy` feature.

Un-tag Sector::mPrice as STATE

The sector price was mistakenly marked as STATE. It is only ever set during `postCalc,` so we remove the tag.

Remove mTechChangeCalc from Technology

We remove the `mTechChangeCalc` and associated `mAlphaZero` member variables from Technology as they are never actually utilized in practice.

Minor Solver bugs

We've noticed a few edge cases which can periodically cause hundreds of wasted iterations. These changes are not likely to improve solver performance more generally. Changes include:

- Not checking if all markets are solved in the Broyden solver before taking our first "step" (and therefore derivative which take tens to hundreds of iterations to compute). We now do a quick check during setup and exit early if they are all solved before doing any real work.
- When we "fail" to take a step in Broyden we would return immediately. Usually the "failed" steps is very close to the last one, however in some rare cases it could be significantly worse. Therefore, we simply revert and re-run the last "good" step before returning just in case.
- In some cases, we could end up having the Preconditioner setting the price to zero, where it will be stuck until we go a full cycle through the solver and the Preconditioner will then "fix" the zero price the next time. We simply avoid setting to absolute zero in the first place.

Address solution variability when re-running the same scenario

We have noticed variability in solution iteration counts even when re-running the exact same scenario, sometimes to the point where a scenario will occasionally fail to solve. This situation is quite frustrating to users, particularly when debugging or developing new features. It turns out there are two main reasons for this variability:

A low probability bug in TechVintageVector

In core model proposal 281 we introduced the concept of a `TechVintageVector`, which an array sized and indexed exactly to be able to hold only as much data for as many model periods for which the Technology containing the array would operate for. One of the challenges that we needed to overcome to accomplish that was a strategy to temporarily hold data, either default values or set during `XMLParse`, **before** we knew what the lifetime of the technology was. To do this we need some unique ID to serve as a key to look up from a `TechVintageVector` to its temporarily stored data so the data can be transferred over when ready. We had chosen the memory address of the `TechVintageVector` to serve as that key. However, in rare instances we could feasibly run into the case that:

1. We create a `TechVintageVector` **A** in memory address X and intend to `XMLParse` some value into it.
2. Later come across a delete="1" XML directive thus freeing memory address X for *any* other use.

3. Later created a different `TechVintageVector` **B** in **exactly** the same memory address X and did **not** intend to `XMLParse` any values into it.
4. When we go to initialize **B** we will end up giving it the data that was intended for **A** which is incorrect.

The probability that this happens is very low and depends on operating systems, memory allocator schemes, and memory demands from other processes on the system. Nevertheless, we have verifiably observed this phenomenon.

To address this issue, we simply use a new static variable explicitly for generate "keys" for the `TechVintageVector` / temporary storage look up. The new static index will continue to increase with each new instance of `TechVintageVector` regardless of if they are to be deleted or not, thus avoiding this error.

Variability from floating point values from parallel calculations

In GCAM 5.4 we enabled the use of GCAM Parallel by default and have since observed variability in solution iteration counts even while re-running the same exact scenario. Some related bugs were fixed in core model proposal 349, however the fundamental issue remained: Computers can only represent Real numbers by approximation. As such, there is a possibility of Catastrophic Cancellation such that a + b + c != a + c + b. From here it follows that if the values of a / b / c could be computed in parallel and the exact order of completion and therefore summation could change from attempt to attempt then we should expect **slightly** different results each time. In practice we see roundoff errors in GCAM on the order of magnitude of 1e-10 in the worst case and 1e-16 most typically. Given all the various tolerances and thresholds in the solution algorithms, this ends up being enough to trigger different solution behaviors.

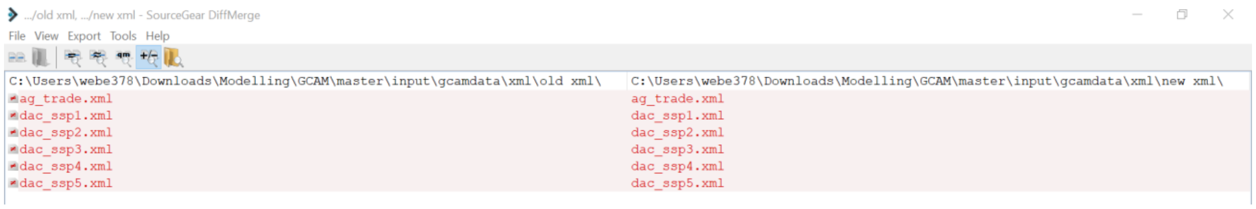
However, there are approaches for dealing with such round off errors even in the face of parallel computations. A couple possibilities are laid out nicely in Collange 2015. In this proposal, we opt for a relatively simple approach: apply the Kahan-Babuška-Neumaier summation method when accumulating supplies and demand for each `Market`. The basic concept is to include an extra (STATE) member variable for each supply and demand to explicitly capture and accumulate round off / truncation errors during `addToDemand` / `addToSupply` and can then be used to apply a correction in `getDemand` / `getSupply`, thus eliminating any variability in supplies and demands and therefore variability in the solver and ultimate iteration counts.

Temporary fix for SSP2 + 2.6 solution issues

During each GCAM release we update the 2.6 forcing target XMLs to include an updated initial guess for the carbon price. The idea being it should speed up the process of converging on the carbon price that achieves the forcing target for subsequent CMPs. For SSP2 + 2.6, the updated guess in GCAM v6.0 seems to have backfired; where the scenario fails to solve the initial guess so bad that the model never recovers. Here we propose a temporary fix: to lower the initial guess; the model then seems to be able to work its way back up and solve just fine subsequently.

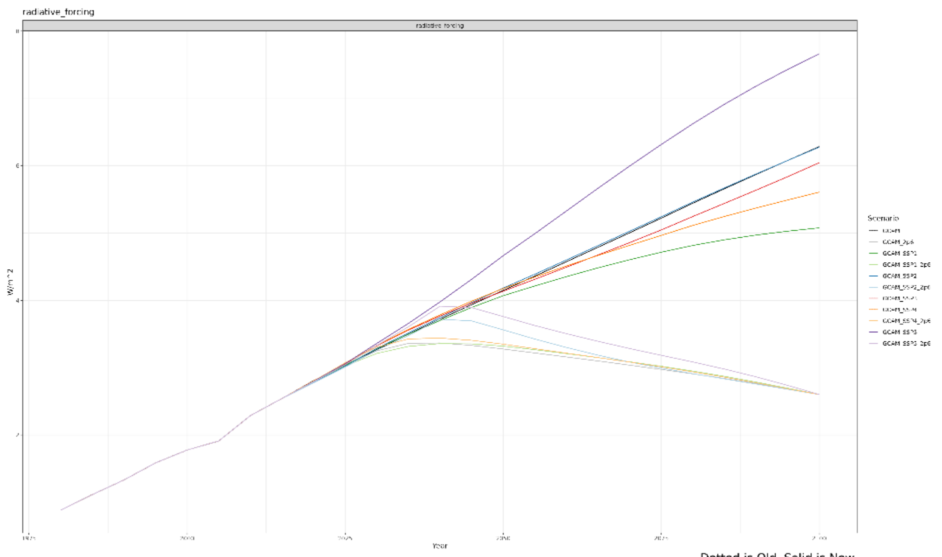
Validation

Image 1: A file diff of the folders containing all old and new XMLs. The ag_trade.xml difference can be explained by a misspelling (diary → dairy) and the five DAC XMLs showing differences can be explained by revised DAC calibration.

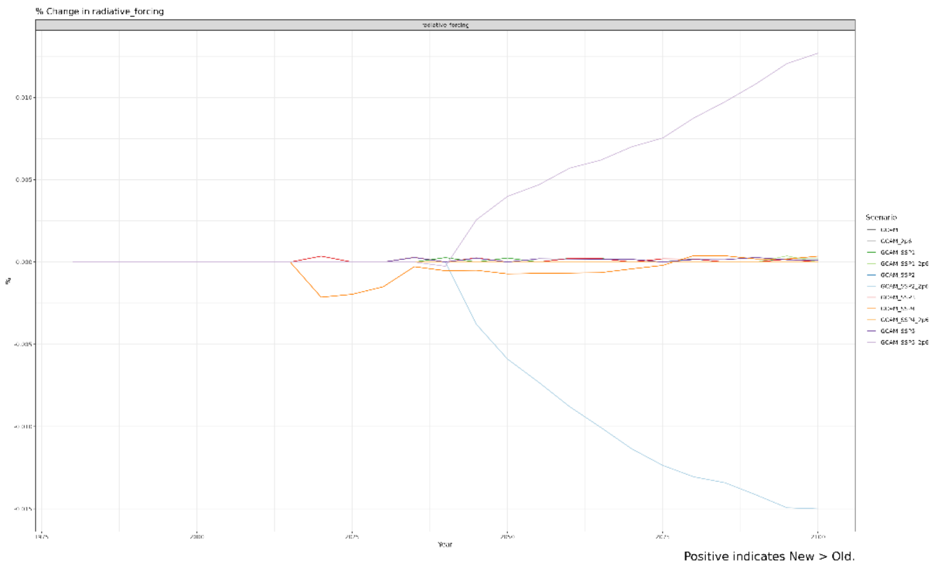

.../old xml, .../new xml - SourceGear DiffMerge
File View Export Tools Help

C:\Users\webe378\Downloads\Modelling\GCAM\master\input\gcamdata\xml\old xml\	C:\Users\webe378\Downloads\Modelling\GCAM\master\input\gcamdata\xml\new xml\
ag_trade.xml	ag_trade.xml
dac_ssp1.xml	dac_ssp1.xml
dac_ssp2.xml	dac_ssp2.xml
dac_ssp3.xml	dac_ssp3.xml
dac_ssp4.xml	dac_ssp4.xml
dac_ssp5.xml	dac_ssp5.xml

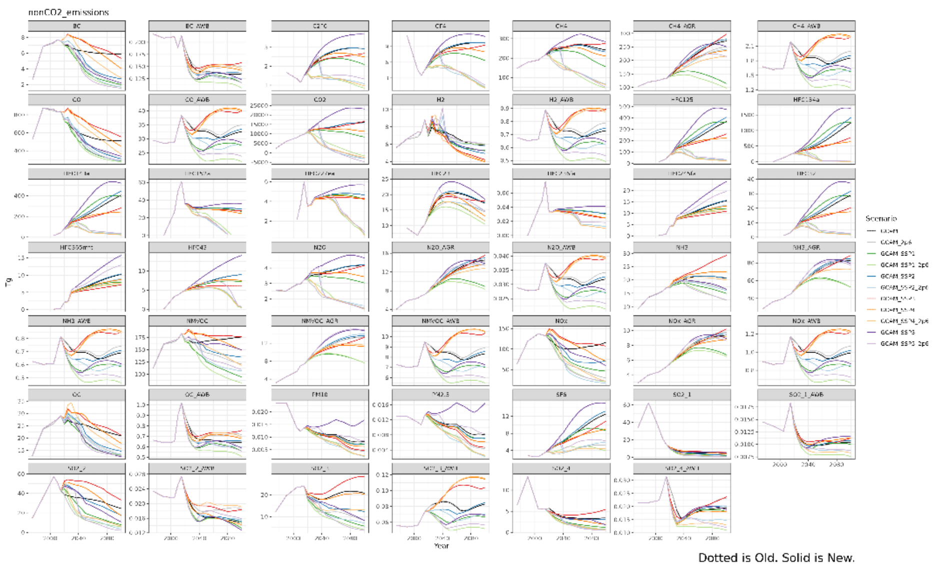
The following are anomaly detector results. There are small variations, mostly in SSP 2 and SSP 5 2.6 W/m^2 forcing target scenarios, which seem to have solved to a slightly different climate. Note: the elimination of variability applies in this proposal but not the current core; thus, small variations (below solution tolerance) are still expected to show up in this comparison.

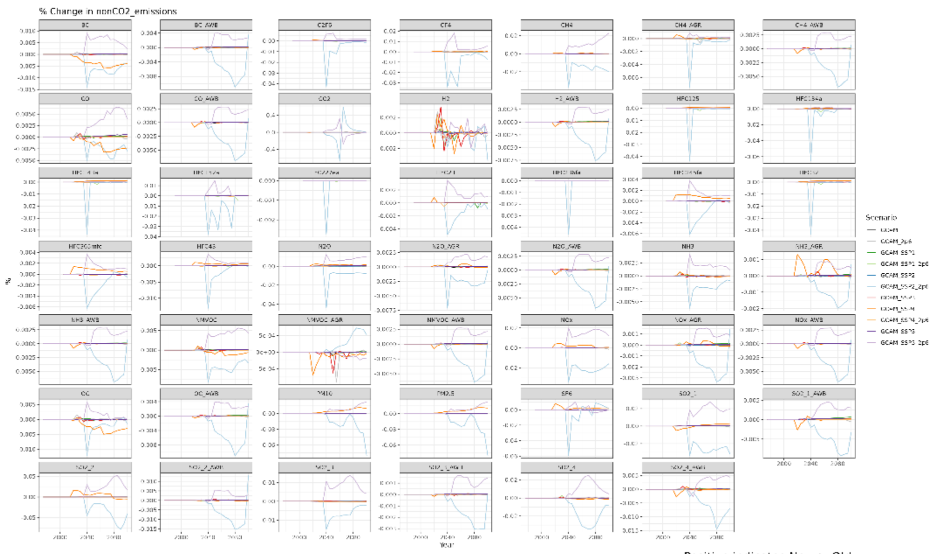


Dotted is Old. Solid is New.



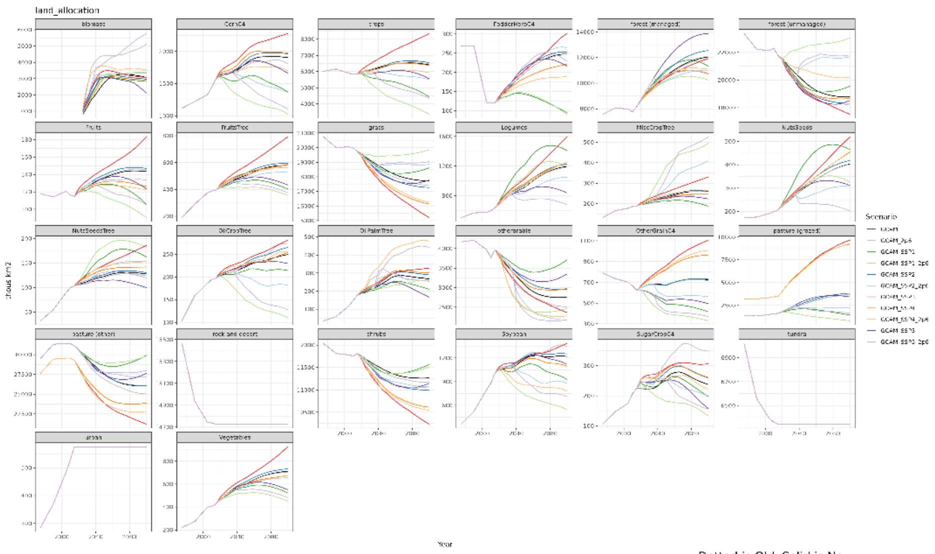
Several changes in this proposal impact non-CO2 objects. The non-CO2 emission results show no meaningful change (besides small variations below solution tolerance):



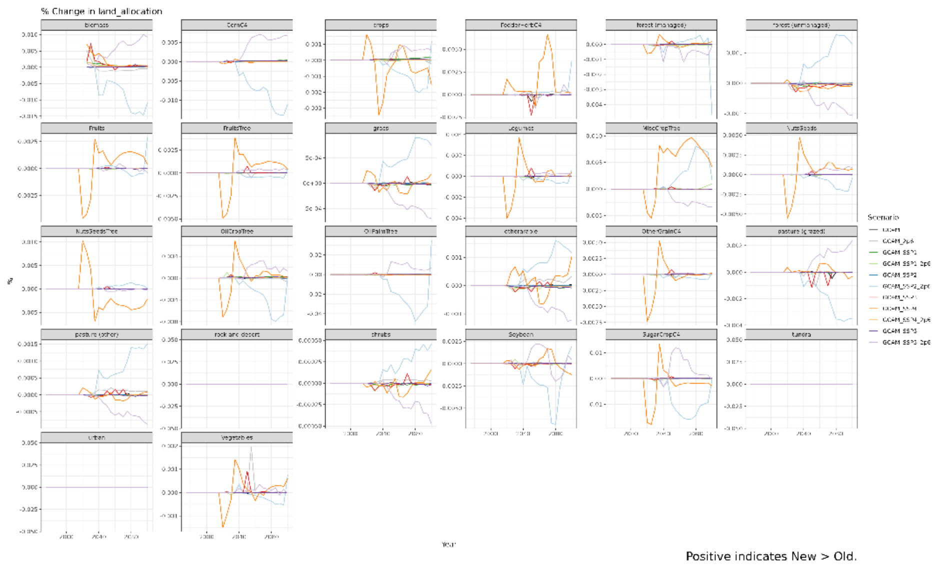


Positive indicates New > Old.

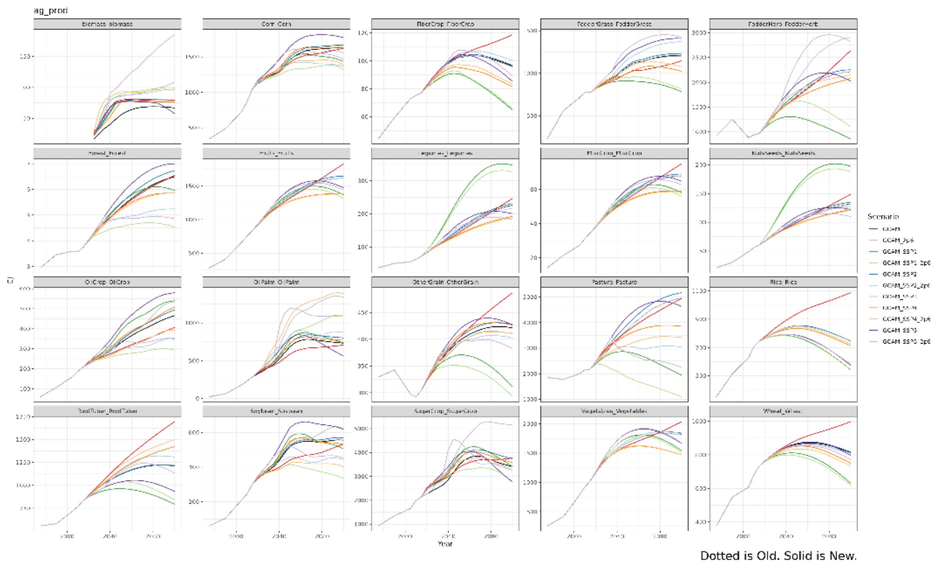
This is also the case for land use:

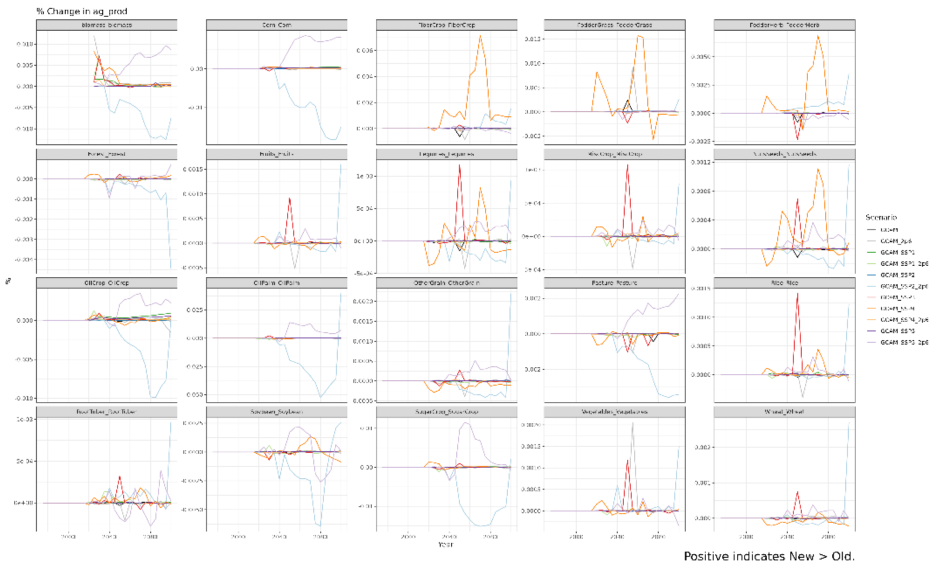


Dotted is Old. Solid is New.



And agricultural production:





Performance

Many of these changes were focused on reducing memory and often more specifically targeting STATE member variables. The reason for the focus on STATE was that, in principle, these are performance critical and if we could get them small *enough* they will fit into cache lines on the CPU and have an outsized impact on runtime. Although the notion of *enough* depends on hardware. The following are some statistics from the model validation runs:

	Master					Proposal				
	Iterations	Scenario Time (sec)	Iterations / sec	Memory (GB)	State (# of vars)	Iterations	Scenario Time (sec)	Iterations / sec	Memory (GB)	State (# of vars)
GCAM	6,038	1,139	5.30	9.91	4,173,772	5,830	841	6.94	10.17	2,411,815
GCAM_2p6	25,046	4,277	5.86	10.53	4,173,967	25,263	3,572	7.07	10.05	2,412,140
GCAM-USA_Ref	8,760	2,855	3.07	13.63	9,106,910	7,876	2,418	3.26	13.12	5,093,624
GCAM_SSP4	6,279	1,172	5.36	10.45	4,115,053	6,990	955	7.32	10.11	2,378,763
GCAM_SSP4_2p6	22,603	3,856	5.86	10.45	4,115,314	21,883	3,237	6.76	10.03	2,379,198

Here we see only slight reduction in total memory usage ~ 500 MB (note the memory usage comes from the Maximum Resident Set Size which unfortunately can be inconsistent as noted in the link; the 500 MB reduction is also what we see on MacOS which seems to be more consistent in reporting from run to run). However, there does appear to be a noticeable increase in performance normalized by iterations / second. Note the value for the "State" column in number of state variables which are all of type double, in terms of MB for the GCAM scenario the comparison is 32 MB vs 18.5 MB, for instance.

Variability:

To check variability, we simply re-run the same scenario multiple times and observe the exact same iteration counts and a diff of the solver_log.csv is identical (except for the time stamp).

In addition, we use *gcamwrapper* to run a single iteration 100 times and observe any variations in supply and demand. Doing in the in the current core we see the range from 1e-10 to 1e-16. With the proposed branch it is exactly zero:

```
> sd = create_solution_debugger(g, 1)
> x0 <- get_prices(sd, T)
> fx0 <- get_fx(sd)
> d0 = get_demand(sd, T)
> s0 = get_supply(sd, T)
> d0.unsc = get_demand(sd, F)
> s0.unsc = get_supply(sd, F)
> p0.unsc = get_prices(sd, F)
>
> eval_demand <- function(dummy) {
+     evaluate(sd, x0, T, F)
+
+     get_demand(sd, F)
+ }
> eval_supply <- function(dummy) {
+     evaluate(sd, x0, T, F)
+
+     get_supply(sd, F)
+ }
> # run 100 iterations for each supply and demand all at the same price
vector and record the values
> demands <- sapply(seq(1,100), eval_demand)
> supplys <- sapply(seq(1,100), eval_supply)
> # calculate the largest variation in values by market over all 100 attempts
> rownames(demands) <- names(x0)
> vari_d <- sapply(names(x0), function(n, D) { max(D[n,]) - min(D[n,])},
demands)
> rownames(supplys) <- names(x0)
> vari_s <- sapply(names(x0), function(n, D) { max(D[n,]) - min(D[n,])},
supplys)
> sort(vari_d, decreasing=F)[1:10]
                  USAcoal               USAcrude oil            USAnatural gas
                        0                          0                         0
            globaluranium                 USAbiomass      USAdistributed_solar
                        0                          0                         0
            USAgeothermal  USAonshore wind resource USAoffshore wind resource
                        0                          0                         0
    USADDGS and feedcakes
                        0
> sort(vari_s, decreasing=T)[1:10]
                  USAcoal               USAcrude oil            USAnatural gas
                        0                          0                         0
            globaluranium                 USAbiomass      USAdistributed_solar
                        0                          0                         0
            USAgeothermal  USAonshore wind resource USAoffshore wind resource
```

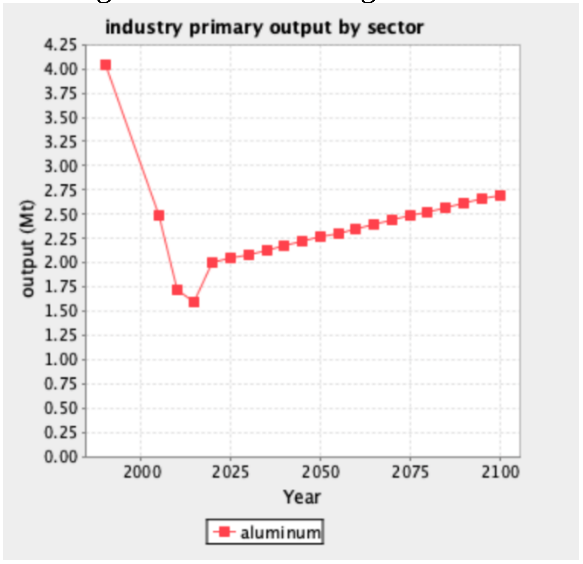
```
                            0                          0                          0
    USADDGS and feedcakes
                            0
> sum(abs(vari_d))
[1] 0
> sum(abs(vari_s))
[1] 0
```

Fixed Final Demand:

The following is an example XML to swap the "aluminum" `energy-final-demand` for the `fixed-final-demand`:

```
<scenario>
    <world>
        <region name="USA">
            <energy-final-demand name="aluminum" delete="1" />
            <fixed-final-demand name="aluminum">
                <service year="1975">3.17207479964381</service>
                <service year="1990">4.04944185218166</service>
                <service year="2005">2.48423106976744</service>
                <service year="2010">1.726</service>
                <service year="2015">1.58964429530201</service>
                <service year="2020">2.0</service>
                <!-- note the C++ will interpolate values -->
                <service year="2100">2.7</service>
            </fixed-final-demand>
        </region>
    </world>
</scenario>
```

Which gives in the following results:


industry primary output by sector
output (Mt)
Year
aluminum