Parallelizable Physics Using Subcolumns

Gunther Huebler
Collaborators: Vincent Larson, John Dennis, Niklas Selke

Motivation

Trends in computational power are shifting toward many-core architectures:

- Systems with nodes containing GPUs are popping up everywhere
- Intel and AMD both offer CPUs with more than 100 threads

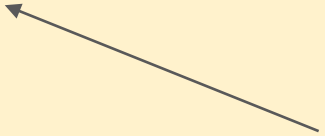
The efficiency of climate models that can utilize this type of parallelism will be passively improved for years to come, but leading climate models aren't currently capable of doing so

Three ways to use highly parallel supercomputers for climate modelling:

1. Run at high resolutions (e.g. 3 km). However, such simulations are limited in duration. These are process simulations rather than climate projections.

1. Run ensembles. This is useful for uncertainty quantification.

1. Use parallelizable subgrid physics. E.g., subcolumns.



Our goal

Subcolumns provide a flexible way of improving results

Subcolumn calculations allow us to estimate the effects of subgrid variability on microphysical process rates.

More subcolumns => better estimations

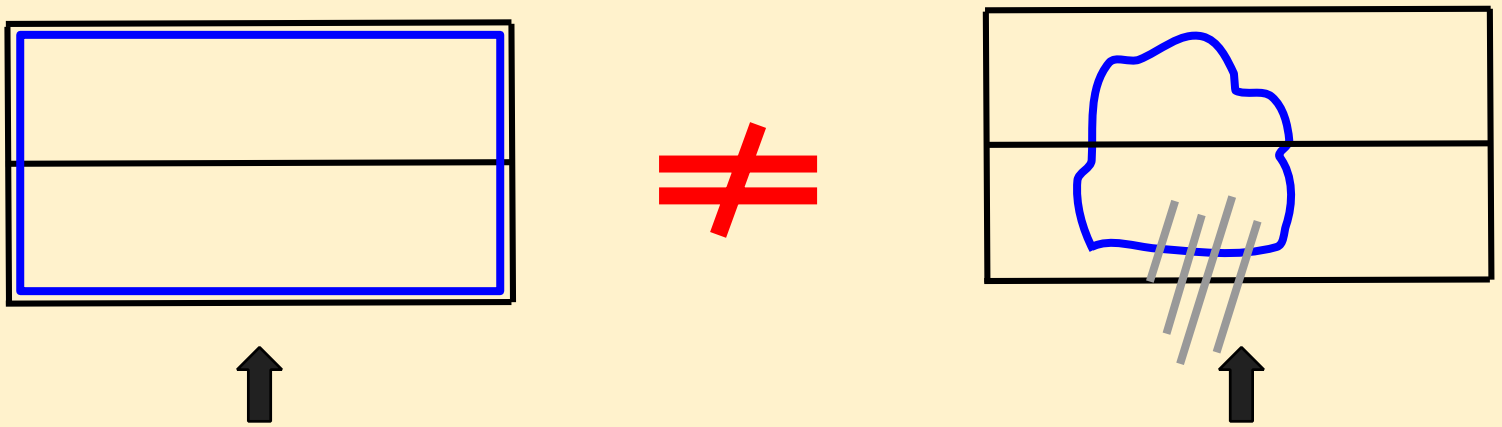
Subcolumns are highly parallel. As higher core count supercomputers are made available, subcolumn-enabled models will see improvements in accuracy without a reduction in throughput.

Subcolumnized models can be run at low resolutions and long time steps, maintaining the ability to run 100-year climate simulations.

The physical problem subcolumns are designed to solve:

We'd like to drive microphysical processes using subgrid-scale variability.

For instance, we'd like to account for the effects of partial cloudiness on drizzle rate. We'd also like to account for within-cloud variability.



How a model with no subgrid information handles microphysics

What we are attempting to do

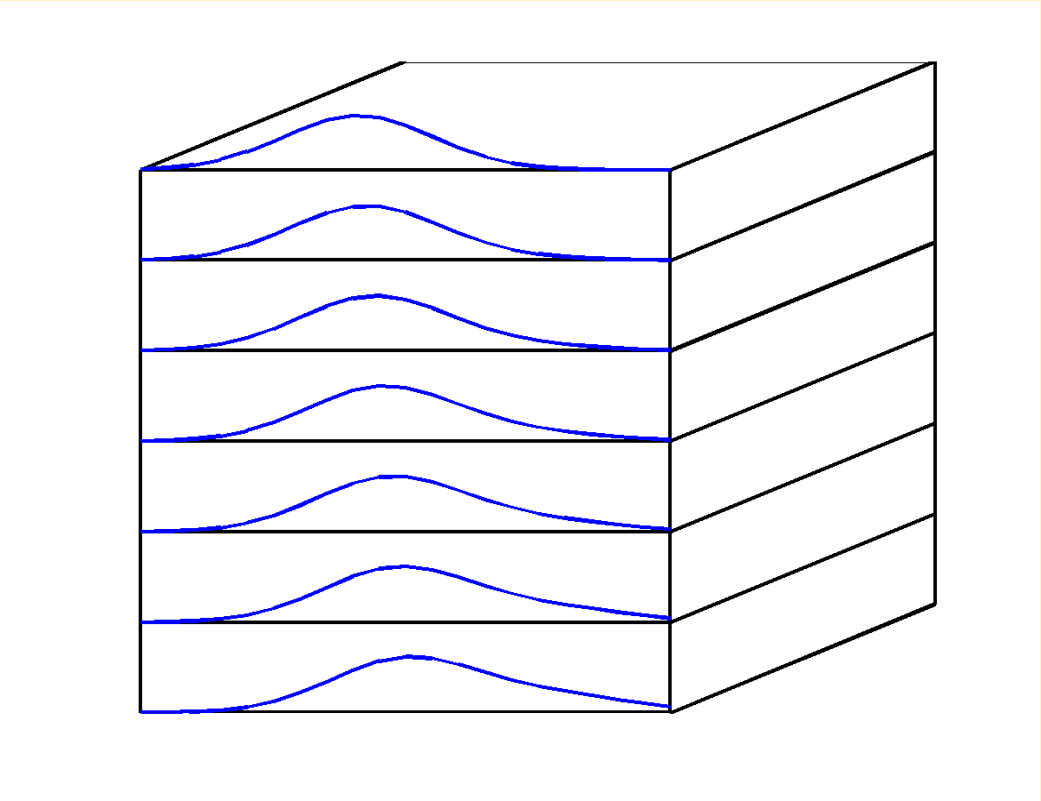
Our subcolumn generator is named SILHS - Subgrid Importance Latin Hypercube Sampler

SILHS is an existing subcolumn model used to calculate grid-box averages of physical process rates (Larson et al. 2005; Larson and Schanen 2013; Raut and Larson 2016)

Currently, SILHS is used only to average microphysical process rates, but has the potential to be applied to radiative transfer or aerosols.

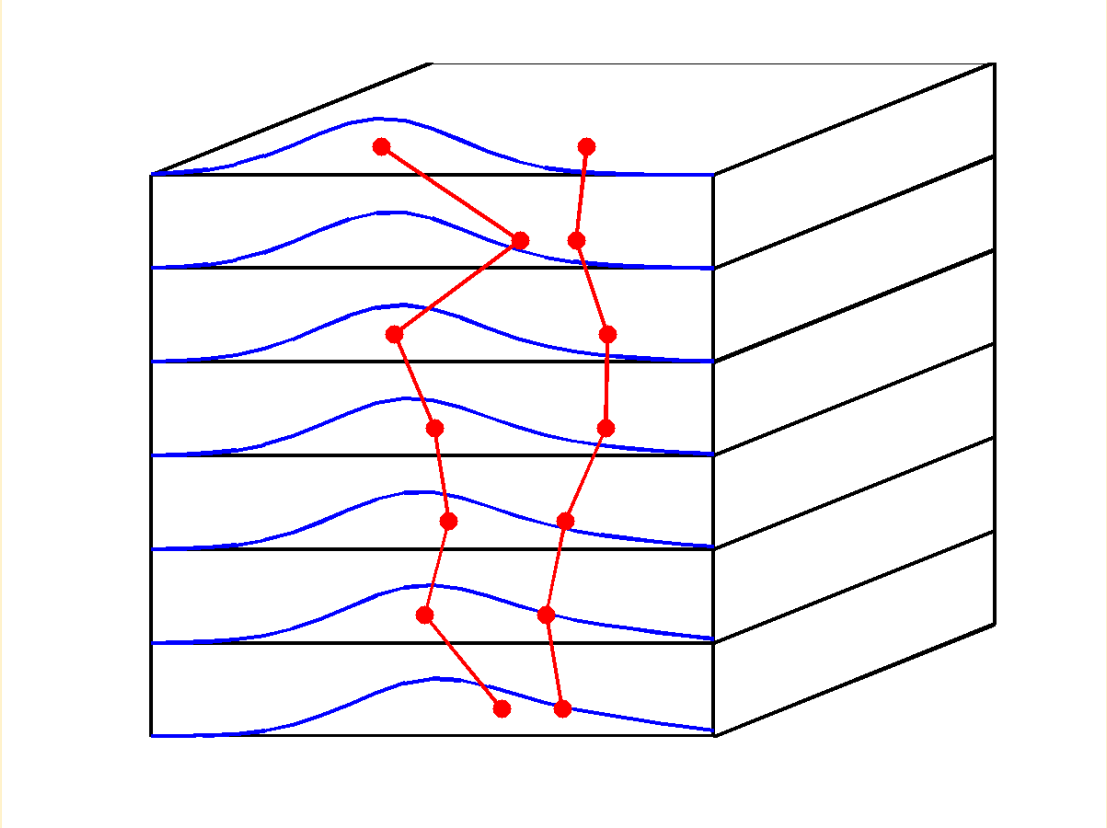
SILHS' 4-step method to parameterize subgrid variability and drive microphysics:

1. Predict the probability density function (PDF) of subgrid variability at each grid level (done by CLUBB).



SILHS' 4-step method to parameterize subgrid variability and drive microphysics:

2. Generate subcolumns that are consistent with the subgrid PDF at each level and satisfy a suitable overlap assumption (done by SILHS).



SILHS' 4-step method to parameterize subgrid variability and drive microphysics:

3. Feed each subcolumn into the microphysics parameterization (MG2).

4. Average the microphysical tendencies from the subcolumns and feed them back into the large-scale (host) model.

Results with SILHS in host models have been produced

SILHS has been tested in a number of host models (CAM, CESM, E3SM, SAM, WRF), and has been shown to improve results.

Even small numbers of subcolumns can improve results.

Large numbers of subcolumns reduce noise and converge to analytic results.

Increasing numbers of subcolumns changes results

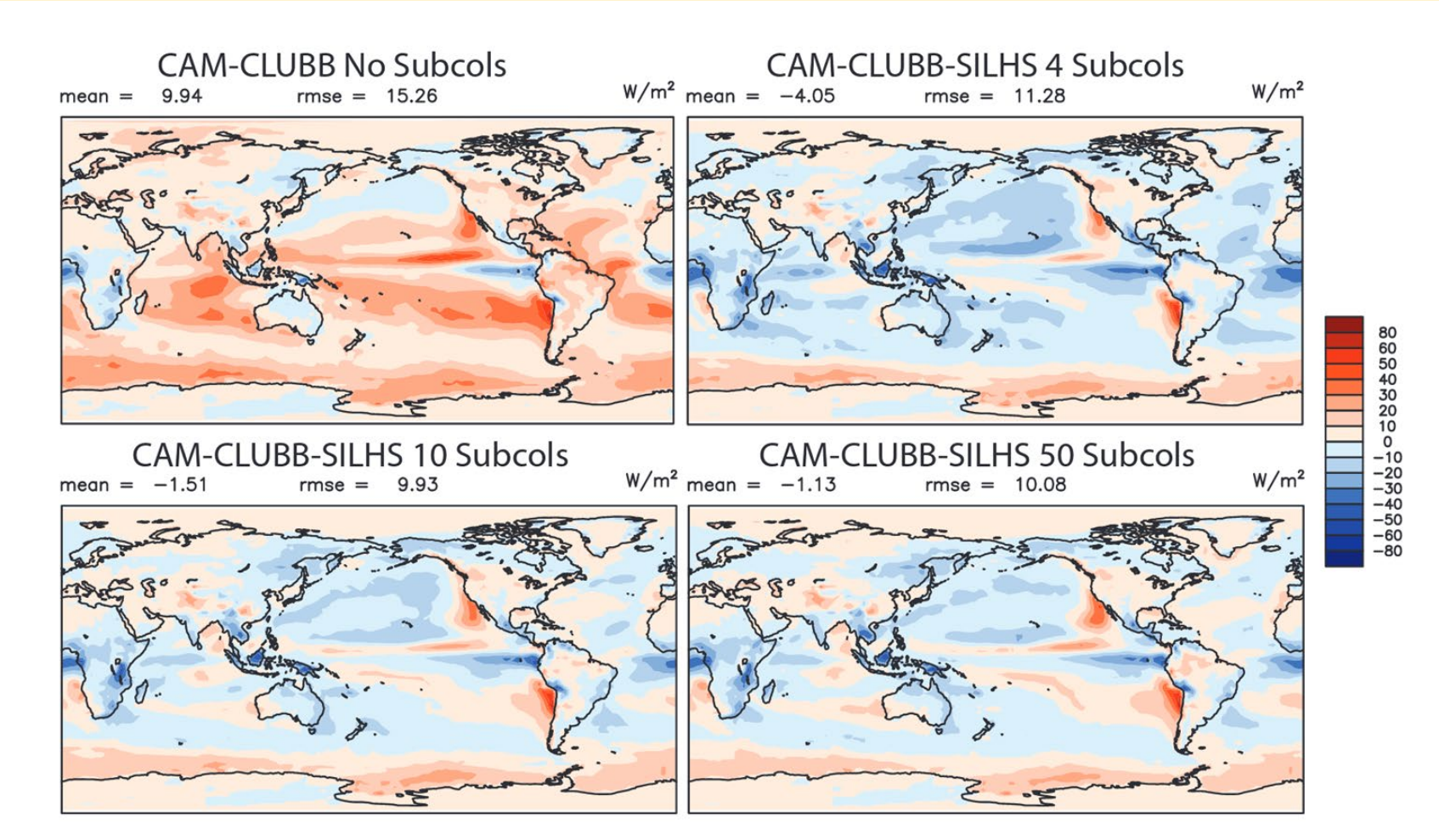


Diagram shows a 5 year average of cloud brightness. This run was tuned using 10 subcolumns. Thayer-Calder et al. (2015)

Large numbers of subcolumns converge to analytic results.

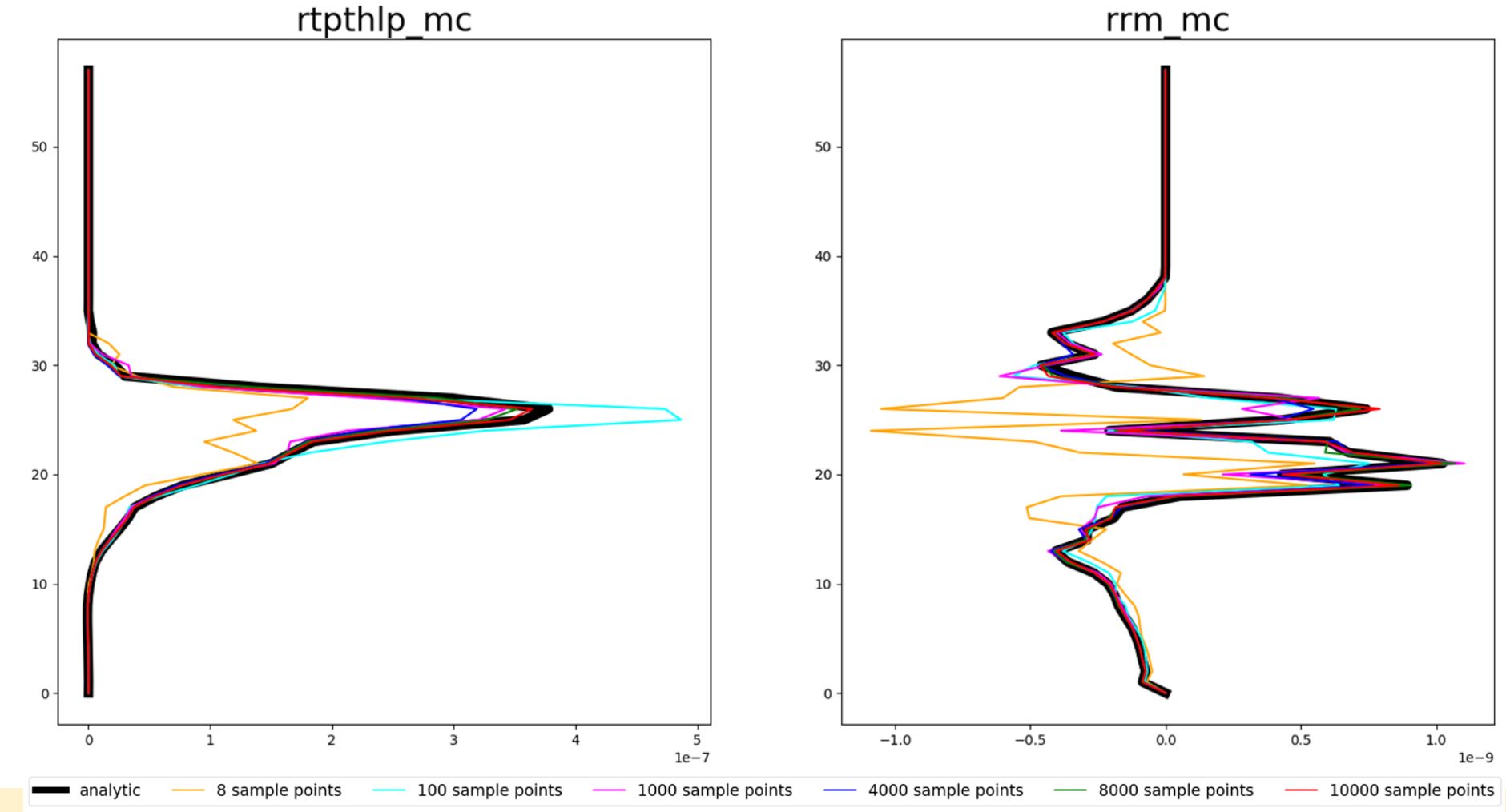


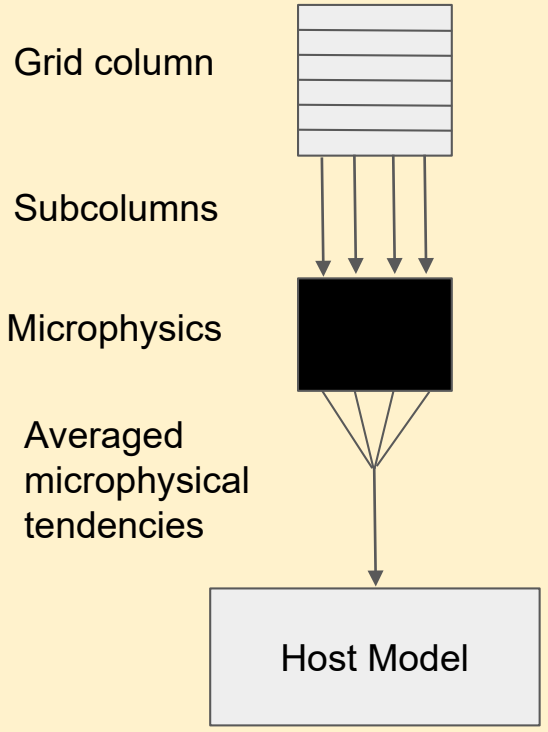
Figure courtesy of Niklas Selke

More subcolumns can improve solution accuracy and are more affordable on highly parallel systems

Subcolumns are embarrassingly parallel with respect to each other.

The cost of a single subcolumn run on a single core is roughly equivalent to the cost of n subcolumns run on n cores.

Without parallelizing the code, the additional cost of the microphysics calls makes using large numbers of subcolumns infeasible.



We are refactoring SILHS to run efficiently on GPUs

GPUs are able to complete 1000s of calculations in parallel

Consider a simple calculation over all vertical levels of every subcolumn drawn from a single grid column:

- This requires (#vertical levels * #subcolumns) calculations.
- With 50 grid levels and 100 subcolumns, each calculation needs to be done 5000 times

To run efficiently on GPUs, parallelism needs to be exposed on a lower level

GPU cores are less powerful and flexible than CPU cores

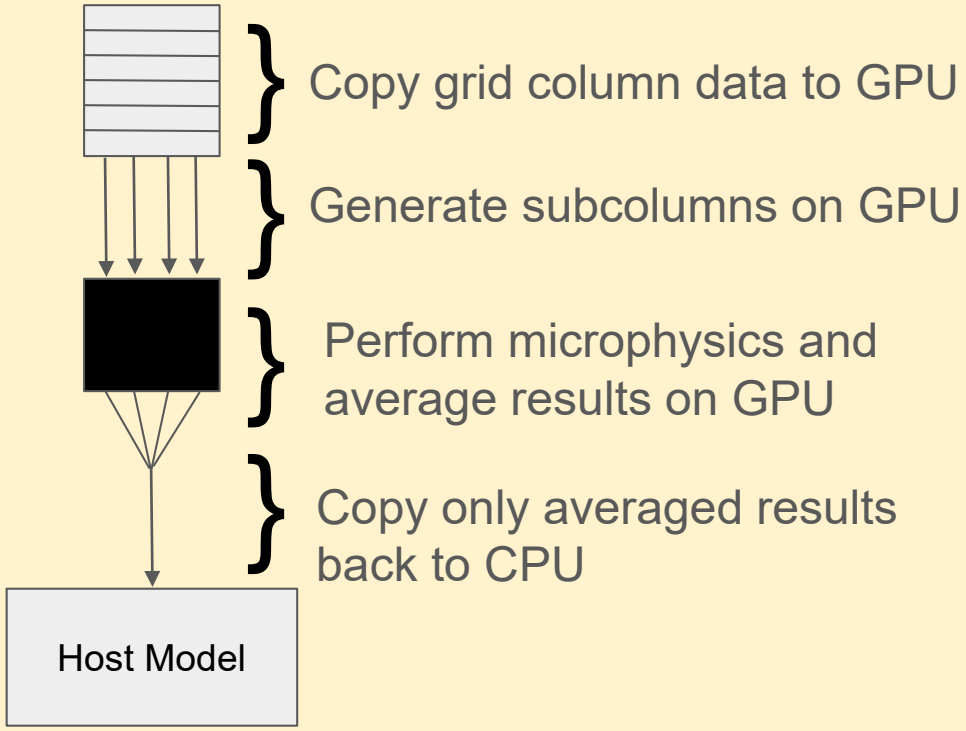
- CPU cores can execute complicated and dissimilar tasks (e.g. you could assign each core to a different subroutine)
- GPU cores can only complete simple and similar tasks (e.g. each core multiplies two numbers, but with each core assigned unique numbers)

In theory, compilers could detect high level parallelism and propagate it down to the lowest level, but this would require some wizardry that compilers are nowhere close to.

Challenges: We need to minimize data transfers to the GPU to make subcolumn calculations efficient

Copying data to and from the GPU device is expensive.

We need to minimize this data transfer to ensure subcolumn calculations run efficiently.



Challenges: Branchy code is inefficient

Consider this example:

```
for i=1, nz
  if ( x(i) > 0 ) then
    *expensive calculation 1*
  else
    *expensive calculation 2*
  end if
end for
```

Running this code on a GPU will likely result in each thread executing both expensive calculations.

GPUs group threads into “warps”, where each thread within a warp executes the same instructions. 32 threads per warp with current architectures.

If one thread in a warp needs to use the first calculation, and another in the same warp needs to use the second calculation, all threads in the warp need to complete both calculations. This is called warp divergence.

Challenges: Kernel sizes need to be small

On a GPU, a kernel is a routine that is executed in parallel, e.g. a loop.

Kernels can be too large to run efficiently. For example, a loop that uses many variables can cause too much memory or register usage in one spot.

```
for i=1, nz
  x(i) = a(i) + b(i)
  y(i) = c(i) + d(i)
  z(i) = e(i) + f(i)
end for
```

Breaking up large kernels into smaller ones can reduce these bottlenecks.

```
for i=1, nz
  x(i) = a(i) + b(i)
end for
for i=1, nz
  y(i) = c(i) + d(i)
end for
for i=1, nz
  z(i) = e(i) + f(i)
end for
```

Pushing loops down to the lowest levels is necessary for GPU code, but also improves CPU code

Ultimately, it is up to the compilers to determine how the work for parallel sections of code is distributed.

A loop with a subroutine in it may be completely parallel, but this is generally too difficult for compilers to detect this parallelism. Push those loops down.

Compilers are best at optimizing and parallelizing simple loops. This is true for code being compiled for either CPUs or GPUs, and even if the CPU code is not being parallelized over threads.

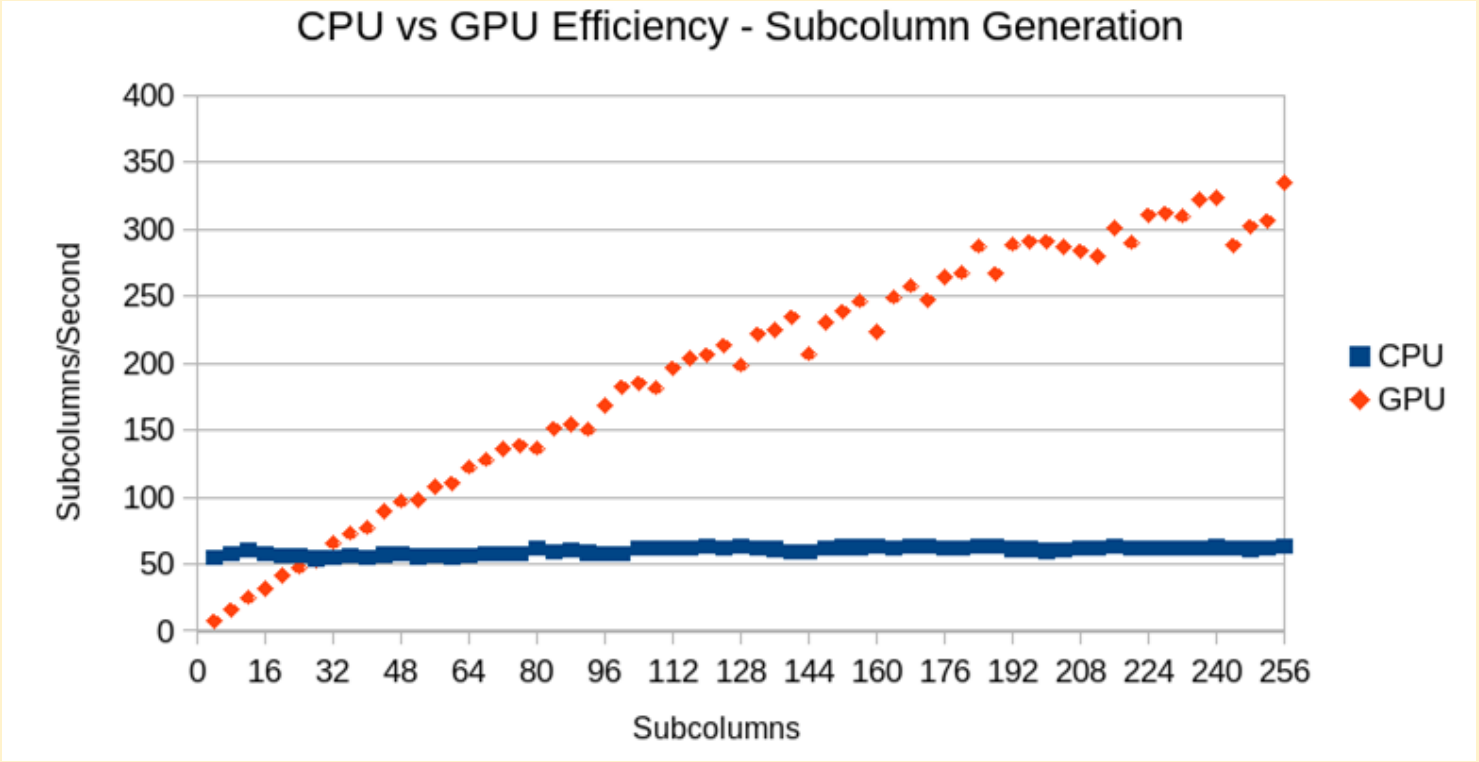
Major coding hurdles are complete

The portion of code that generates subcolumns on the GPU is complete.

We have a version of MG2 running on GPUs, courtesy of John Dennis.

The code bases have been tested for efficiency, but still need to be stitched together in a single version of CAM and optimized.

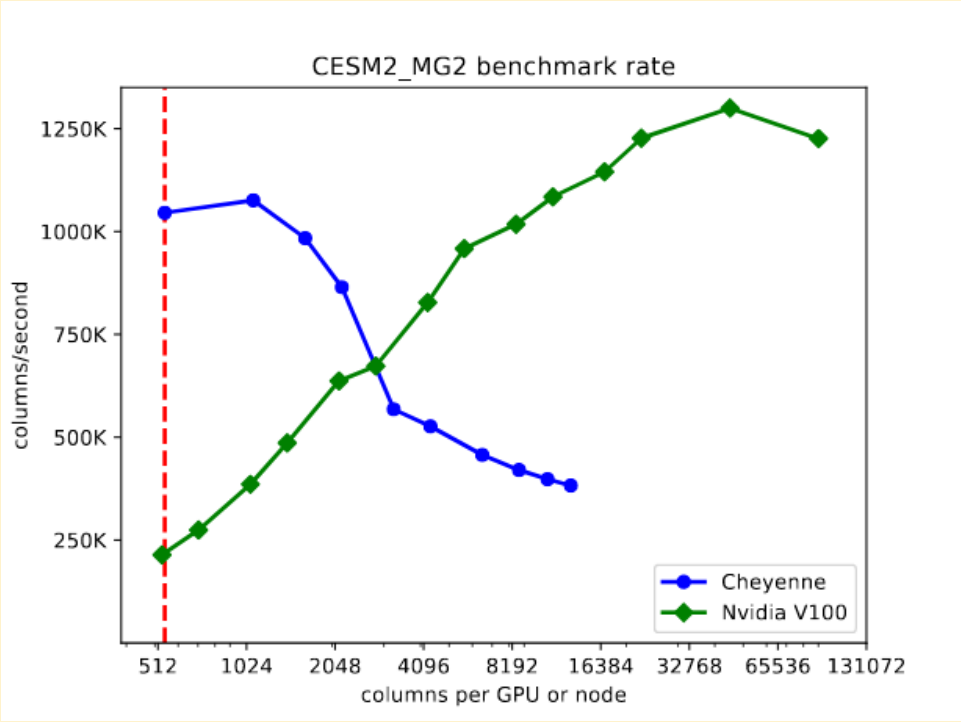
Efficiency analysis: Subcolumn generation



This figure shows the number of subcolumns that can be generated per second using a single Intel Xeon CPU core, compared to using a P4000 GPU. The test was run with a single column model.

This demonstrates that the efficiency of the subcolumn generation method on a many-core architecture improves as more subcolumns are used.

Efficiency analysis: Microphysics (MG2)



This figure shows the number of subcolumns that can be handled per second by MG2 vs a GPUized version of MG2. Courtesy of John Dennis.

This demonstrates that the efficiency of the microphysics call on a many-core architecture improves as more subcolumns are used.

Efficiency analysis: Interpretation

The true additional cost of using a fully GPUized version of SILHS cannot be evaluated using these results.

These tests demonstrate that when run on many-core architectures, the efficiency of SILHS improves as more subcolumns are used.

As many-core architectures improve and become more available, we will be able to either:

- Maintain subcolumn counts and see increases in throughput
- Increase subcolumn counts while maintaining throughput

Future versions of SILHS could run efficiently on GPUs and many-core CPUs

Since CPUs are capable of parallelism on low levels also, code that exploits parallelism well on GPUs can do so with CPUs as well, with minor changes.

CPUs still don’t have the core counts to compete with GPUs for massively parallel tasks, but SILHS has a configurable number of subcolumns, allowing us to fit the number of subcolumns to the amount of cores available.

Recap

Recently, and in the foreseeable future, increases in computational power are being driven by the shift to many-core architectures.

This type of parallelism doesn't improve our ability to run long duration climate simulations using existing models.

As more cores (CPU or GPU) are made available, our subcolumn method can improve the accuracy of existing models without affecting throughput.

There's still work to be done, but initial tests are promising and the project has a lot of future potential.