Progress update: High-resolution offline ELM simulation over North America

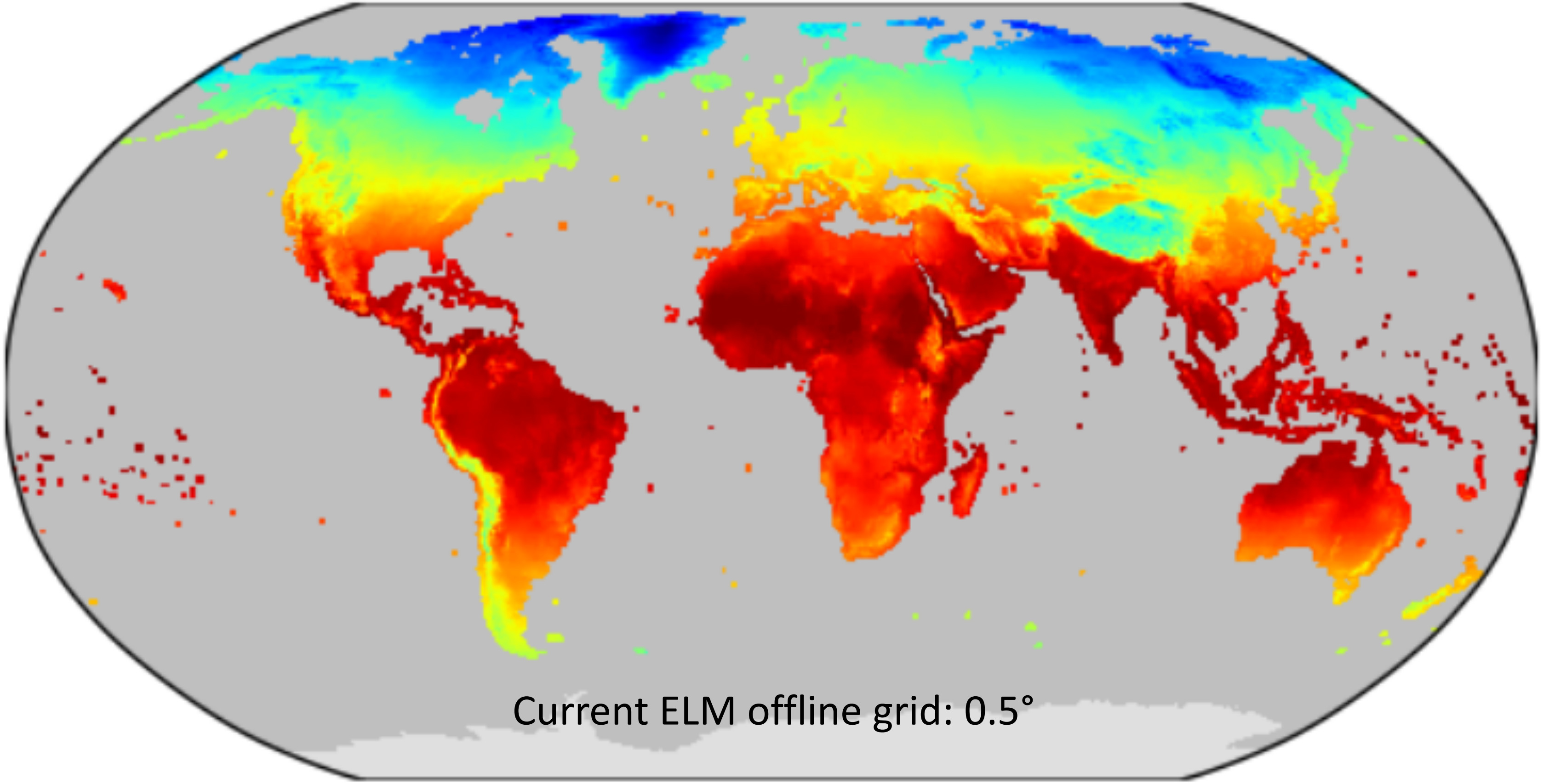
Peter Thornton, Peter Schwartz, Dali Wang, Rupesh Shrestha, Michele Thornton, Shih-Chieh Kao, Marcia Branstetter, Fengming Yuan,

ORNL

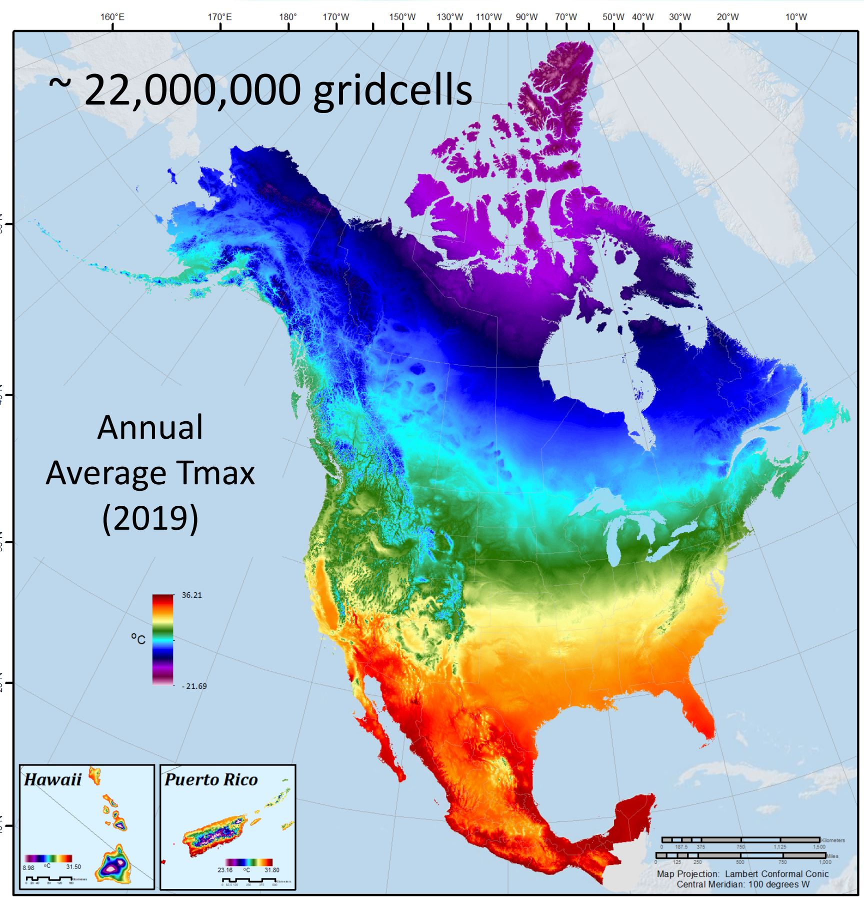
Project tasks and schedule

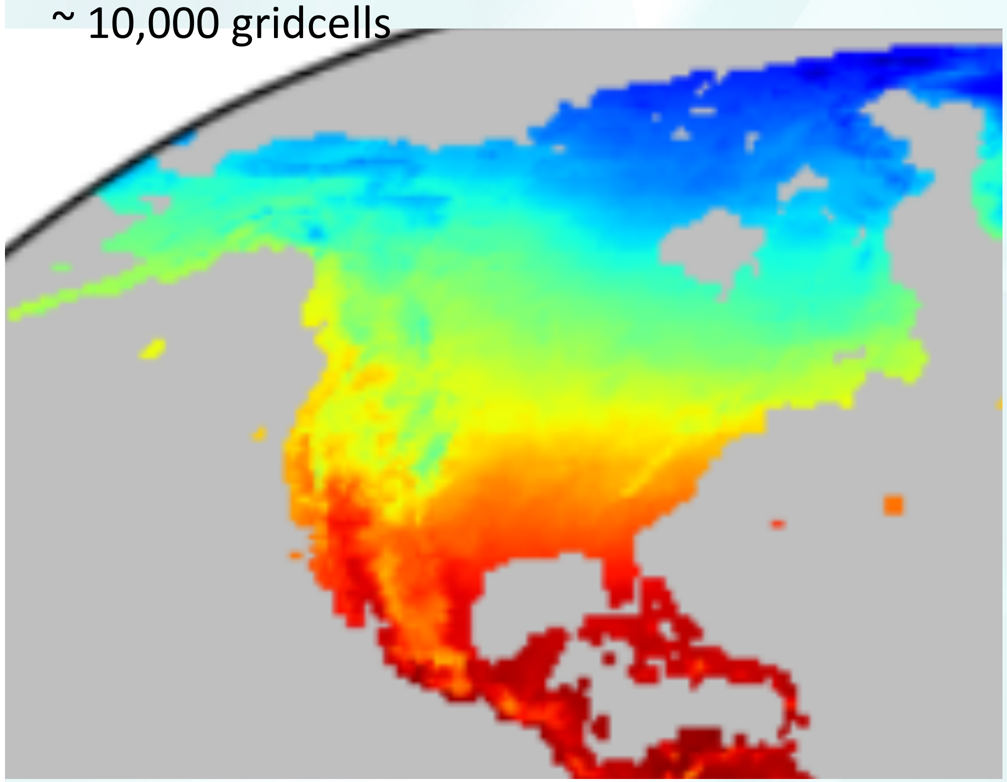
Year	Month	Task 1	Task 2	Task 3	Task 4	Task 5
2019	Jun	Construct hi-res surface weather forcings, including sub-daily down-scaling	Assemble other hi-res surface data			
	Jul	🟩	🟨			
	Aug	🟩	🟨	Adapt ELM with coupler bypass method for GPU system (Summit)		
	Sep	🟩	🟨	🟩		
	Oct	🟩	🟨	🟩		
	Nov	🟩	🟨	🟩	Apadpt ELM I/O methods for GPU system (init, history and restart)	
	Dec	🟩	🟨	🟩	🟩	
2020	Jan	🟩	🟨	🟩	🟩	
	Feb	🟩	🟨	🟩	🟩	Execute NA spinup runs
	Mar	🟩	🟨	🟩	🟩	🟨
	Apr	🟩	🟨	🟩	🟩	🟨
	May	🟩	🟨	🟩	🟩	🟨

Target: 1km^2 grid resolution over N. America

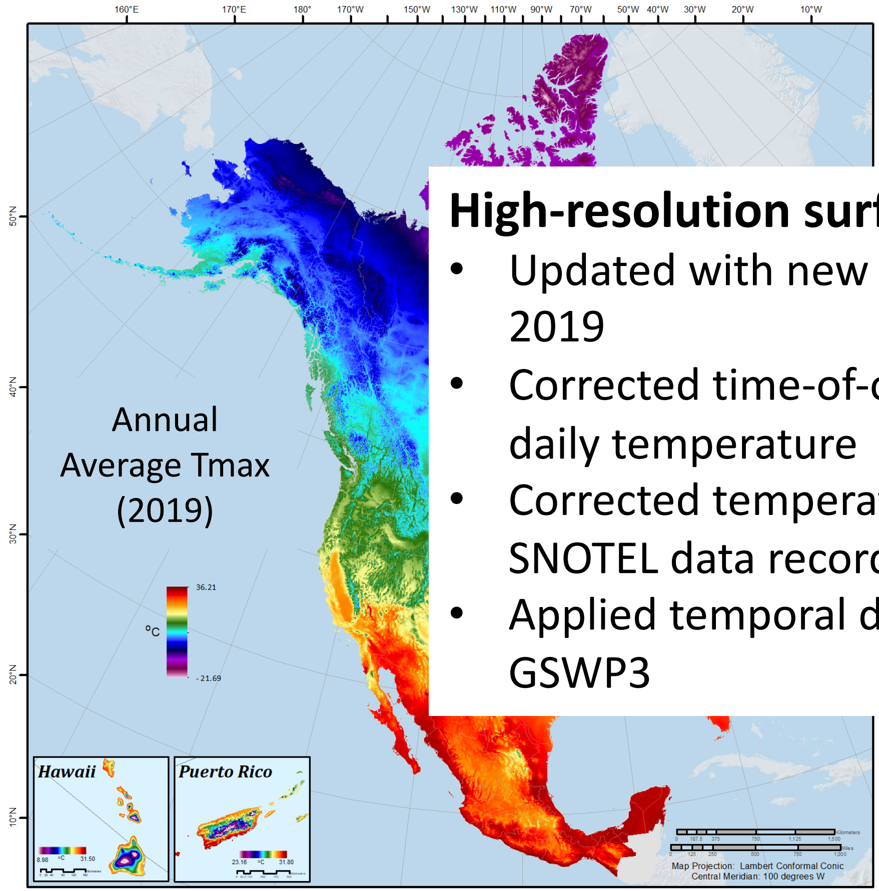

Current ELM offline grid: 0.5°

Target: 1km^2 grid resolution over N. America

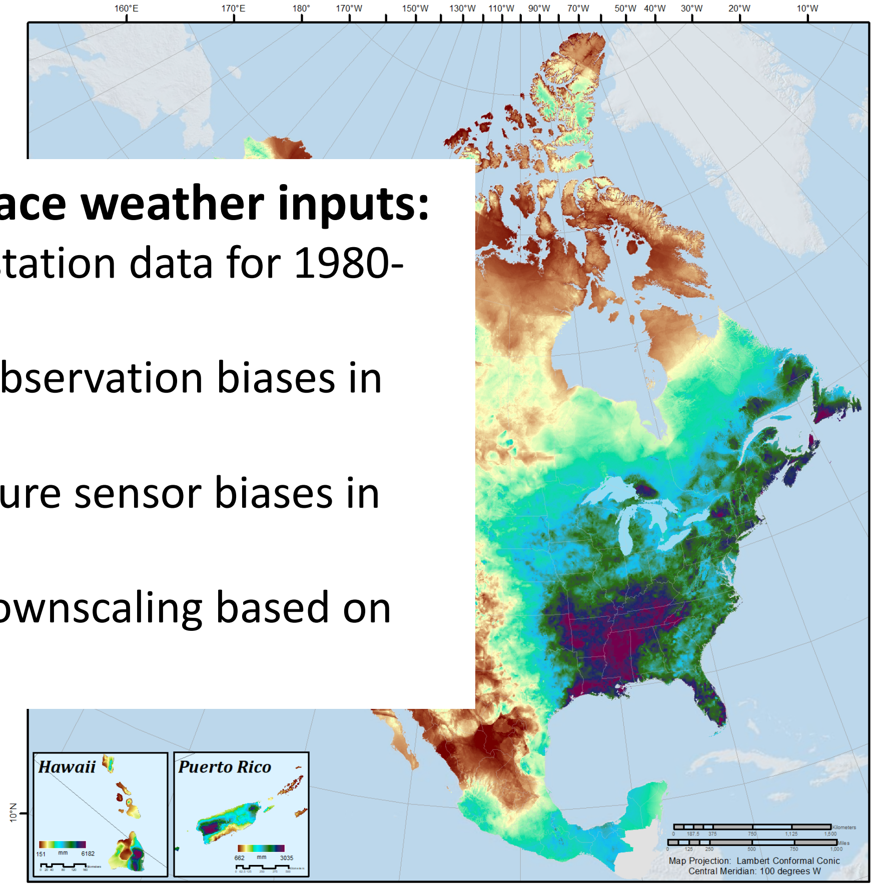




Target: 1km^2 grid resolution over N. America



Annual Average Tmax (2019)
36.21
°C
- 21.69
Hawaii
Puerto Rico
Map Projection: Lambert Conformal Conic
Central Meridian: 100 degrees W

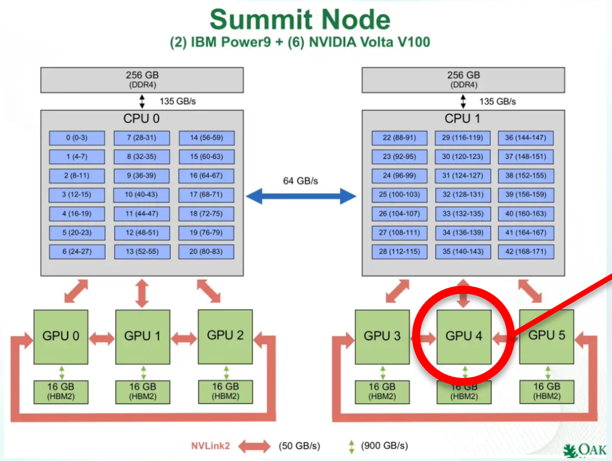


Hawaii
Puerto Rico
Map Projection: Lambert Conformal Conic
Central Meridian: 100 degrees W

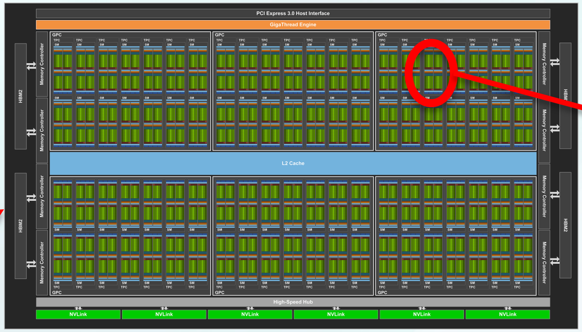
High-resolution surface weather inputs:

- Updated with new station data for 1980-2019
- Corrected time-of-observation biases in daily temperature
- Corrected temperature sensor biases in SNOTEL data record
- Applied temporal downscaling based on GSWP3

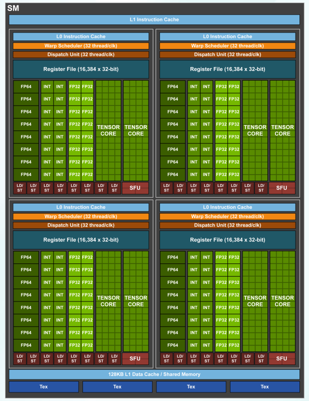
Computational strategy: OpenACC on Summit



Each Summit node has 6 NVIDIA Volta V100 GPUs. We plan to have 1 ELM MPI task per GPU, so 6 MPI tasks per node



Each GPU has 80+ Streaming Multiprocessors (SMs) and 16 GB of shared memory (HBM2)

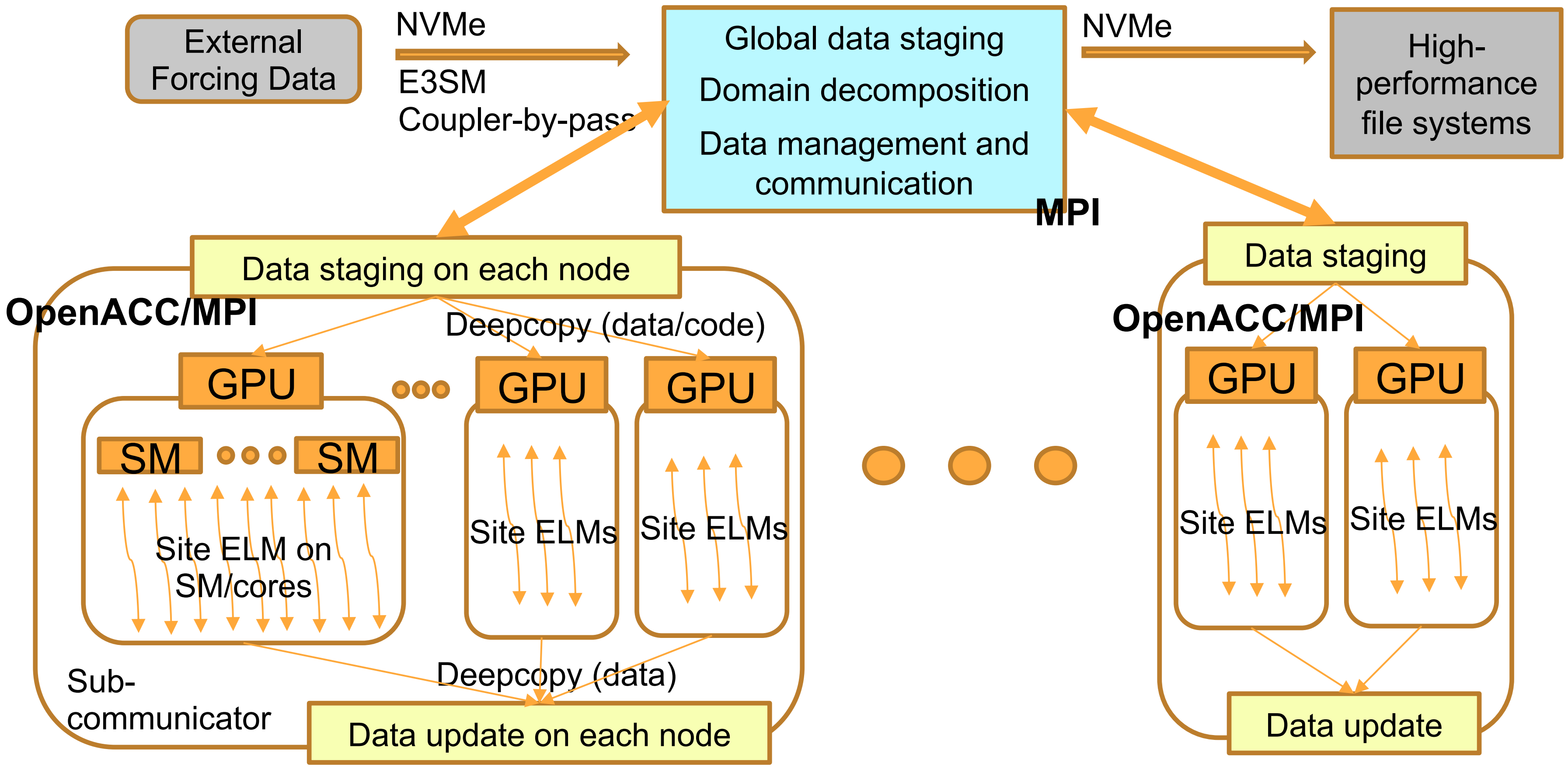


SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
FP64
INT
FP32
TENSOR CORE
LD/ST
SFU
128KB L1 Data Cache / Shared Memory
Tex

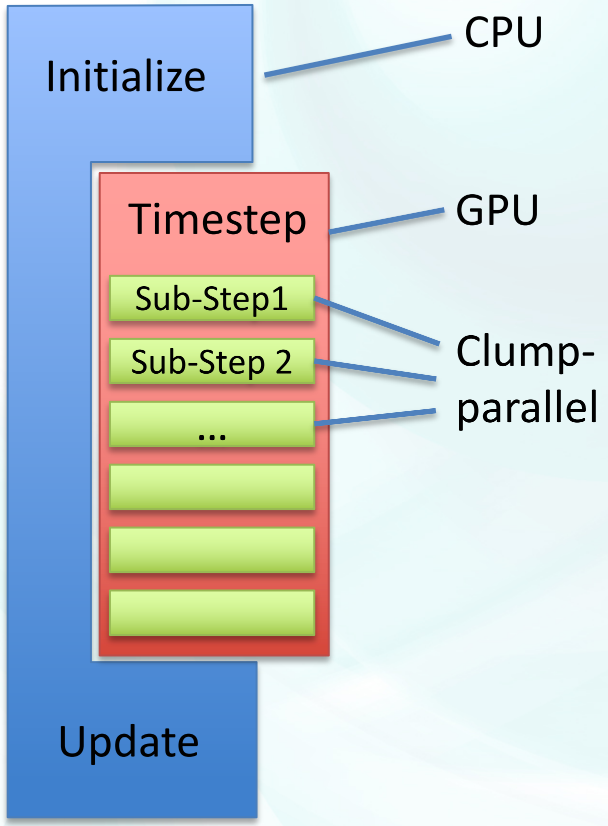
Each SM has 32 double precision cores, which can be "over-subscribed" with threads to an extent that depends in part on availability of register space and heap space.

Our approach is to use the existing "clumps" parallelism in ELM (traditionally connected to OpenMP), and tie it to the double precision cores on the V100 GPUs via OpenACC, using **1 gridcell per clump**.

Parallel strategy and data management

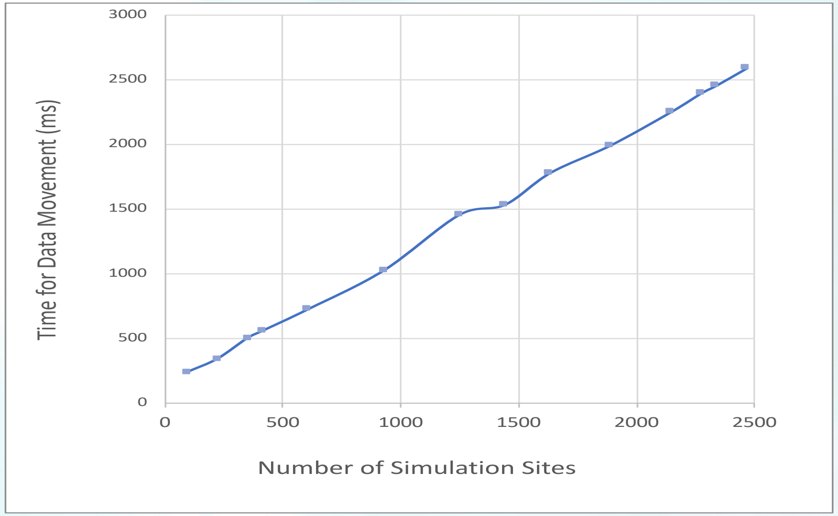


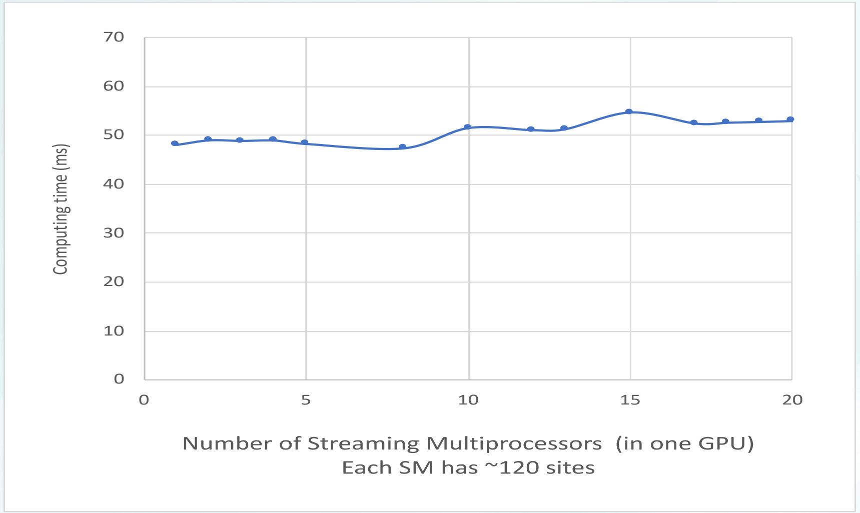
OpenACC implementation within ELM timestep



- Using OpenACC deep copy method to move entire ELM data structure (all clumps) onto GPU.
- Sequential execution of major science routines in ELM run step on GPU (6 clump-parallel regions).
- Update CPU on exit from GPU block, to allow I/O
- Using the **Functional Unit Testing** framework developed by Dali Wang during E3SM Phase 1 to rapidly prototype the GPU kernel
- Initial performance tests showed **near-ideal scaling** of compute time out to ~10,00 clumps per GPU.
- As expected, data transfer time scales with number of clumps per GPU.

Timing for Data Movement and Computation

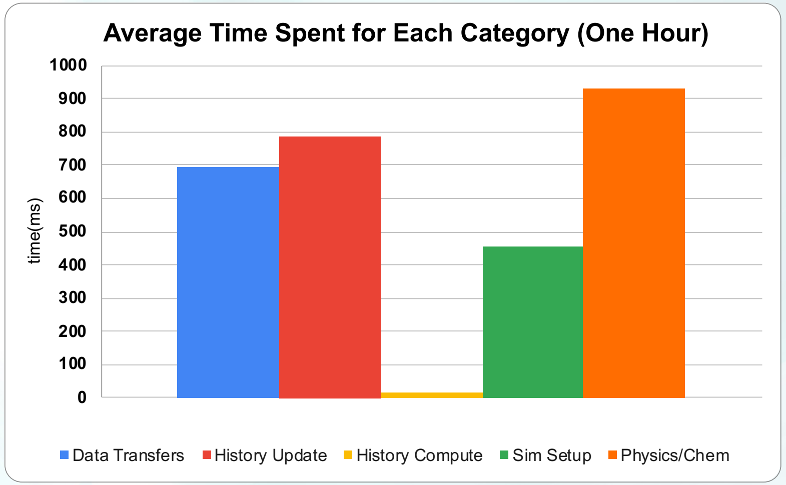


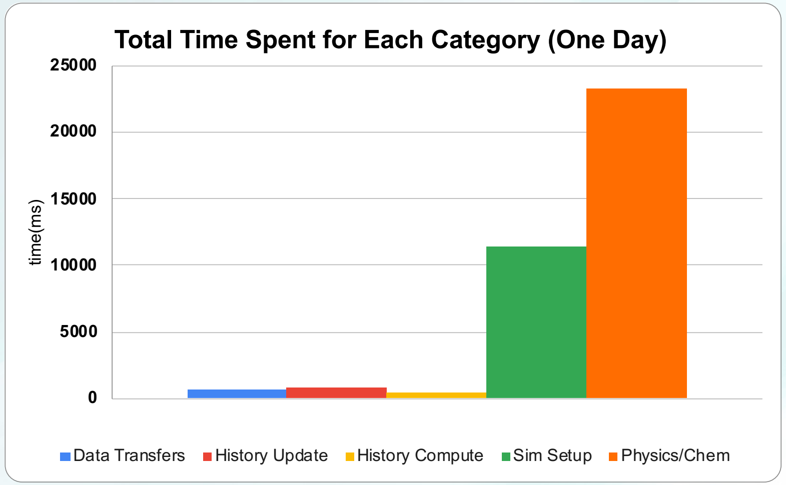


Time required for initial data movement onto GPU scales with the number of threads (gridcells)

Time required for computation is level as more of the SMs on a GPU device are used to do more work

Baseline time profiling





The dominance of data transfers and history update decreases with longer compute periods

Summary of problem size and timing

- Memory constraints mean that we can get about 2000 gridcells to run in parallel on a single Summit GPU device
- With 6 devices per node, 12K gridcells per node
- With 22M gridcells, we could efficiently use about 1800 Summit nodes (out of 4600)

- Current GPU timing is 160 seconds per model day, or about 1.5 SYPD
- Many opportunities for additional optimization (discussed later), but with current timing:
- 200 year spinup simulation on Summit would take about 5 months

Accelerating ELM with OpenACC(Data)

Approach: 1 MPI task controls 1 GPU and use existing clump structure to have each GPU thread compute 1 gridcell.

- Module variables must be in !$acc declare directive.
- Deepcopy enabled via compilation flag
  - Currently, PGI couldn't get correct structure for all derived types — even for types that held parameter
  - Changing to pointer elements works
- Unstructured Data Regions used to transfer data only at beginning and end of run and history tapes when needed.

Directive must accompany variable declaration

```
type(DecompCNParamsType) :: DecompCNParamsInst
!$acc declare create(DecompCNParamsInst)
```

Changing to pointers Example

```
type, public :: DecompCNParamsType

      real(r8), pointer :: cn_s1_cn => null()

      real(r8), pointer :: cn_s2_cn => null()
```

Accelerating ELM with OpenACC(Routines)

- Every routine used in a GPU region must have GPU code generated — including intrinsic functions
  - Removed custom timing, error checking, and I/O functions.
  - Device code for most routines is generated by the !$acc routine seq directive.
  - Slicing arrays of derived types is not supported — must make local pointer first (e.g., associate clause)
  - Class methods are not supported — must create separate routine and pass variable as argument.
- Developed python script to recursively parse routines to add acc directives, make consistent changes and identify class methods to streamline adding new modules/developments
- Introduced clump parallelism to code that didn't support it, such as history and accumulation buffer update.

Class Method example

```
!call col_nf%Summary(bounds, num_soilc, filter_soilc)
call colnf_summary_acc(col_nf,bounds, num_soilc,    filter_soilc,
dt)
```

Array slicing (error due to implicit intrinsic)

```
!call c2g(bounds,  col_cf%nee(begc:endc), &
!    lnd2atm_vars%nee_grc(begg:endg), &
!    c2l_scale_type= unity, l2g_scale_type=unity)
call c2g(bounds, nee(begc:endc) , nee_grc(begg:endg) , &
    c2l_scale_type= unity, l2g_scale_type=unity)
```

Performance Optimization

- Allocating dynamic memory is slower on GPU than CPU.
- Highest priority optimization is slight refactoring to replace local arrays with scalars
    - If can't, then allocate based on relevant filter and not entire clump.
    - Has greatly increased performance in routines done so far –many more to go.
- Next steps:
    - Enable compiler optimizations such as in-lining, unrolling, etc.. (-OX flags crash)
    - Increase parallelism for routines that are most compute intensive (e.g., history buffer is parallelized across fields as well as gridcells)
    - Task parallelism and better data locality for routines that aren't compute heavy but with hundreds of global memory accesses.
    - Fine tuning of OpenACC parameters: gangs, registers, etc..

```
real(r8) :: diffus (bounds%begc:bounds%endc,1:nlevdecomp+1)
real(r8) :: adv_flux(bounds%begc:bounds%endc,1:nlevdecomp+1)
real(r8) :: a_tri(bounds%begc:bounds%endc,0:nlevdecomp+1)
real(r8) :: b_tri(bounds%begc:bounds%endc,0:nlevdecomp+1)
real(r8) :: c_tri(bounds%begc:bounds%endc,0:nlevdecomp+1)
real(r8) :: r_tri(bounds%begc:bounds%endc,0:nlevdecomp+1)
real(r8) :: d_p1_zp1(bounds%begc:bounds%endc,1:nlevdecomp+1)
real(r8) :: d_m1_zm1(bounds%begc:bounds%endc,1:nlevdecomp+1)
real(r8) :: f_p1(bounds%begc:bounds%endc,1:nlevdecomp+1)
real(r8) :: f_m1(bounds%begc:bounds%endc,1:nlevdecomp+1)
real(r8) :: pe_p1(bounds%begc:bounds%endc,1:nlevdecomp+1)
real(r8) :: pe_m1(bounds%begc:bounds%endc,1:nlevdecomp+1)
real(r8) :: dz_node(1:nlevdecomp+1)
real(r8) :: conc_trcr(bounds%begc:bounds%endc,0:nlevdecomp+1)
```

After:

```
real(r8) :: diffus_j, diffus_jm1, diffus_jp1
real(r8) :: adv_flux_j,adv_flux_jm1, adv_flux_jp
real(r8) :: a_tri(num_soilc,0:nlevdecomp+1)
real(r8) :: b_tri(num_soilc,0:nlevdecomp+1)
real(r8) :: c_tri(num_soilc,0:nlevdecomp+1)
real(r8) :: r_tri(num_soilc,0:nlevdecomp+1)
real(r8) :: d_p1_zp1
real(r8) :: d_m1_zm1
real(r8) :: pe_p1
real(r8) :: pe_m1
real(r8) :: dz_node
real(r8) :: conc_trcr(num_soilc,0:nlevdecomp+1) !
```

Underway now:

- 40,000 gridcell (subset) simulations on Summit using production datasets to evaluate input data staging
- Continued OpenACC optimization at subroutine level