US-DOE Energy Exascale Earth System Model (E3SM) Exascale strategy and current progress

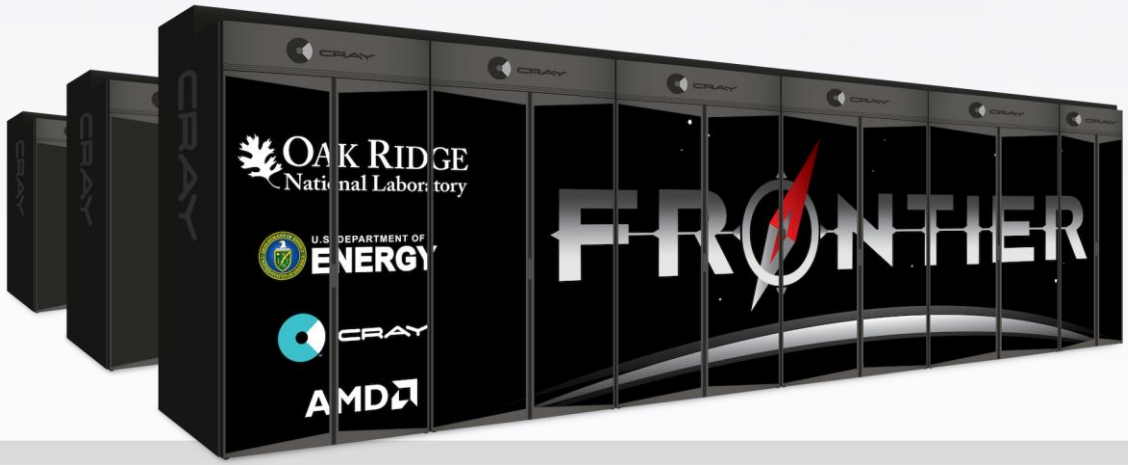
Mark Taylor (Sandia National Labs)
Peter Caldwell (Lawrence Livermore National Lab)
E3SM Nonhdyrostatic NGD group
E3SM Performance group

ESCAPE-2 Final Dissemination Workshop
September 3, 2021

E3SM Exascale strategy = Running on GPUs

All new DOE machines will be GPU based:

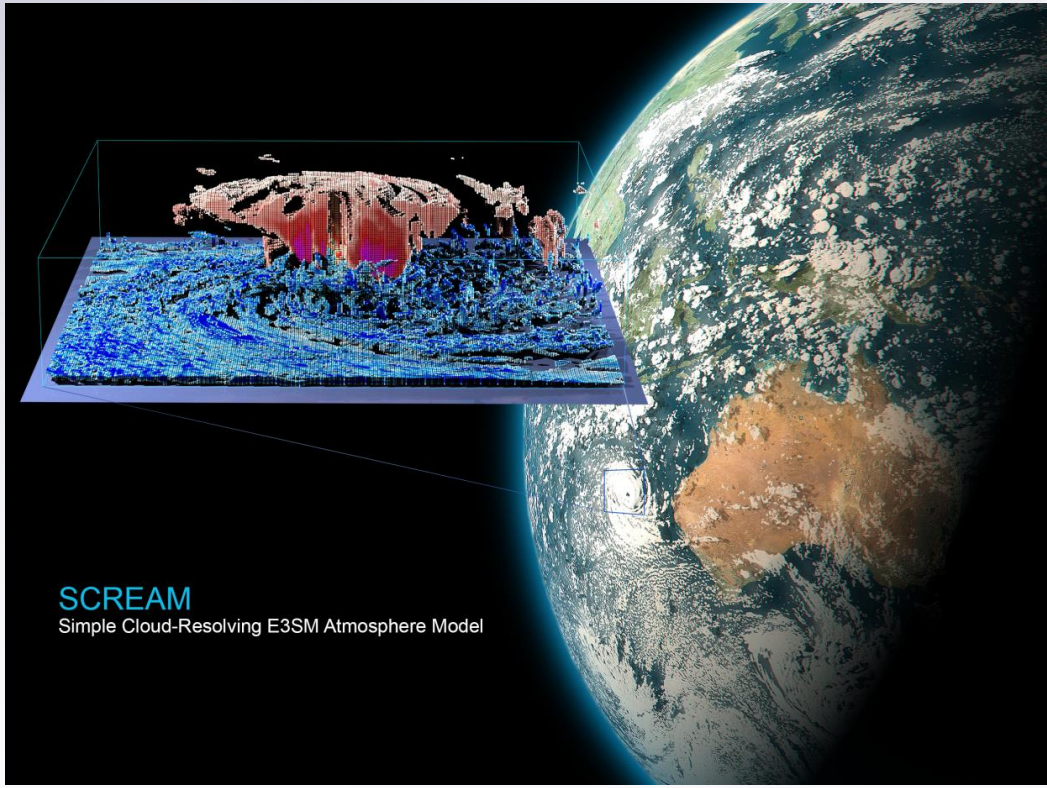
- 2021: NERSC Perlmutter
  - 6000 GPUs. 8 MW
  - 3000 CPU nodes (2022)
- 2022: OLCF Frontier
  - 1 CPU + 4 AMD GPUs per node. 30 MW
- 2023: ALCF Aurora
  - Intel GPU nodes. ~40MW



DOE will achieve exascale via increased parallelism rather than faster cores

E3SM performance portable strategy

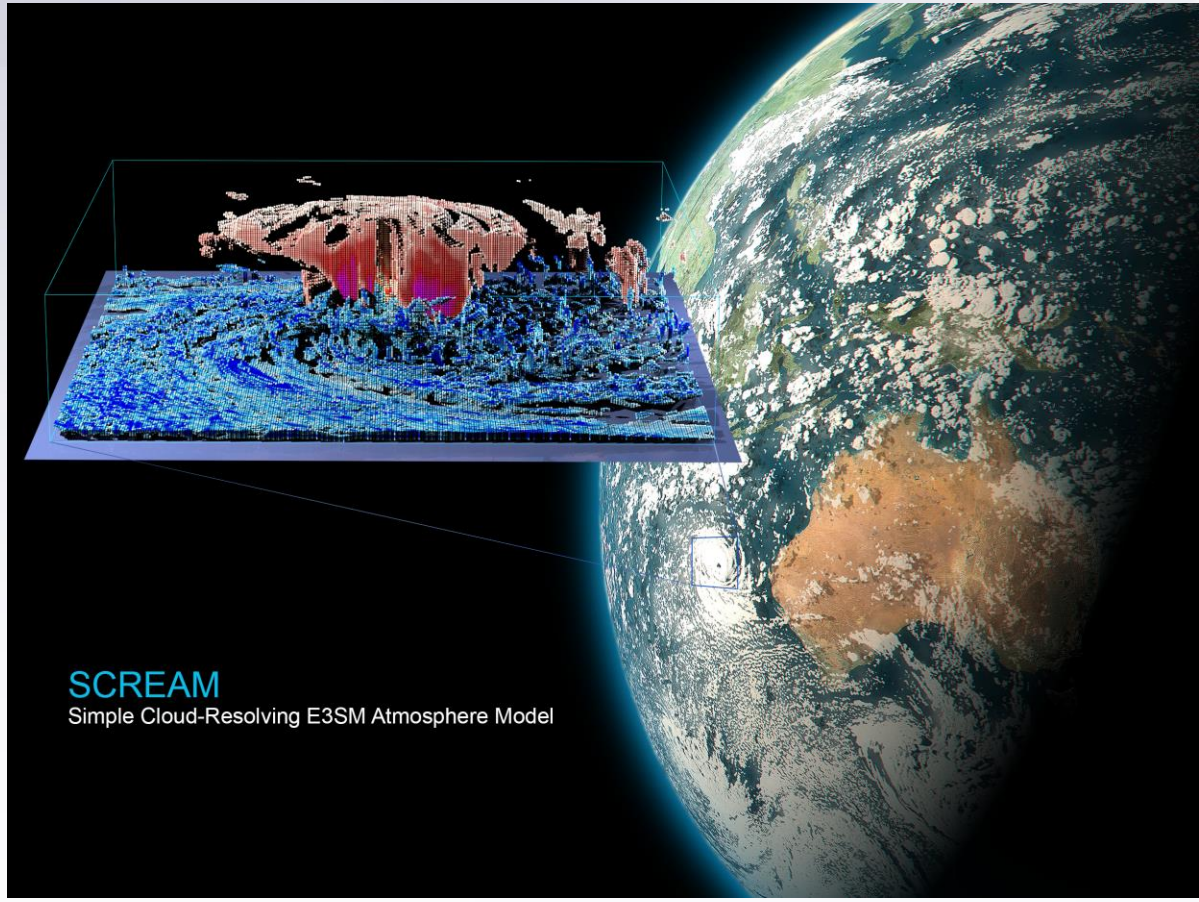
1. Land: F90 OpenACC/OpenMP
2. Ocean: F90 OpenACC/OpenMP
3. Superparameterized Ocean and Atmosphere (an ideal use for GPUs): C++/Kokkos
4. Sea Ice: transition to discrete element model, C++/Kokkos
5. Simple Cloud-Resolving E3SM Atmosphere Model (SCREAM): C++/Kokkos



Today we will focus on SCREAM because it is the most mature effort

SCREAM Status

1. V0: Fortran implementation
   a. NH dycore: Taylor et al., 2020
   b. Physics: SHOC, P3, RRTMG, prescribed aerosols)
   c. DYAMOND results: Caldwell et al., GMD, under review
2. V1: Rewrite in C++/Kokkos:
   a. NH Dycore port completed in 2020
   b. Physics port completed in 2021
   c. Driver to be completed in late 2021
   d. Bertagna et al., GMD 2019, SC 2020
3. Remaining slides: dycore benchmarks on NVIDIA V100 GPU systems



SCREAM-C++: Basics About Kokkos

1. Kokkos is a C++ library which provides an **abstraction layer** around code related to on-node parallelism (like loops and arrays)
   - This allows a single code to run efficiently on CPUs, GPUs, and whatever comes next
- Implementations are provided for Cuda, OpenMP, Serial, Pthreads, etc
  - Kokkos is developed by a large multi-disciplinary team with early access to new machines and compilers
  - Kokkos emphasizes getting its features into the C++ standard

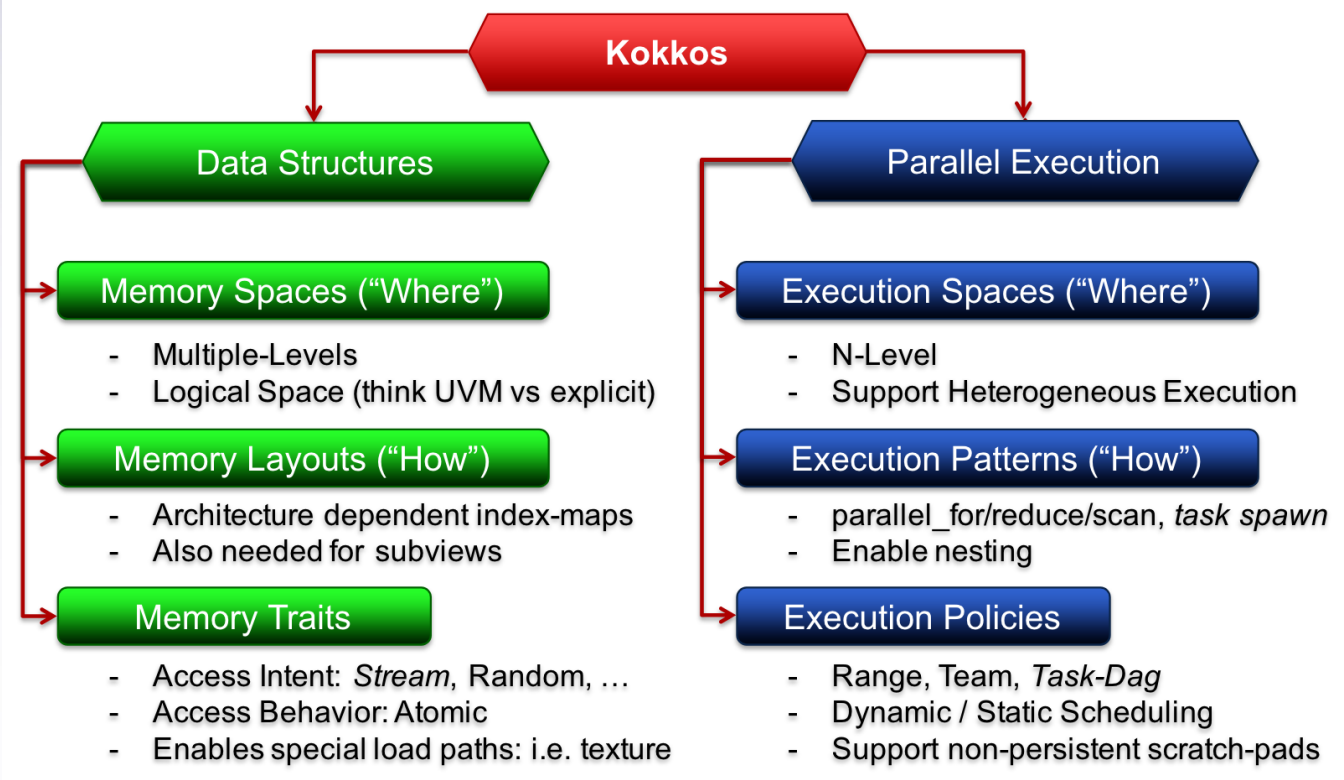


Fig: Diagram of Kokkos' abstraction hierarchies. From https://github.com/kokkos/kokkos-tutorials/blob/master/KokkosCapabilities.pdf

The Downside of C++:

- C++ and Kokkos make the code more complicated (see example)
- Code complexity may drive E3SM towards a model where only computer scientists can change code – but we hope not!
- Opportunity to remove decades of outdated code is a major benefit of switching languages
- Most of the complexity is due to “packs” needed to get C++ code to vectorize when running on CPUs

Original F90

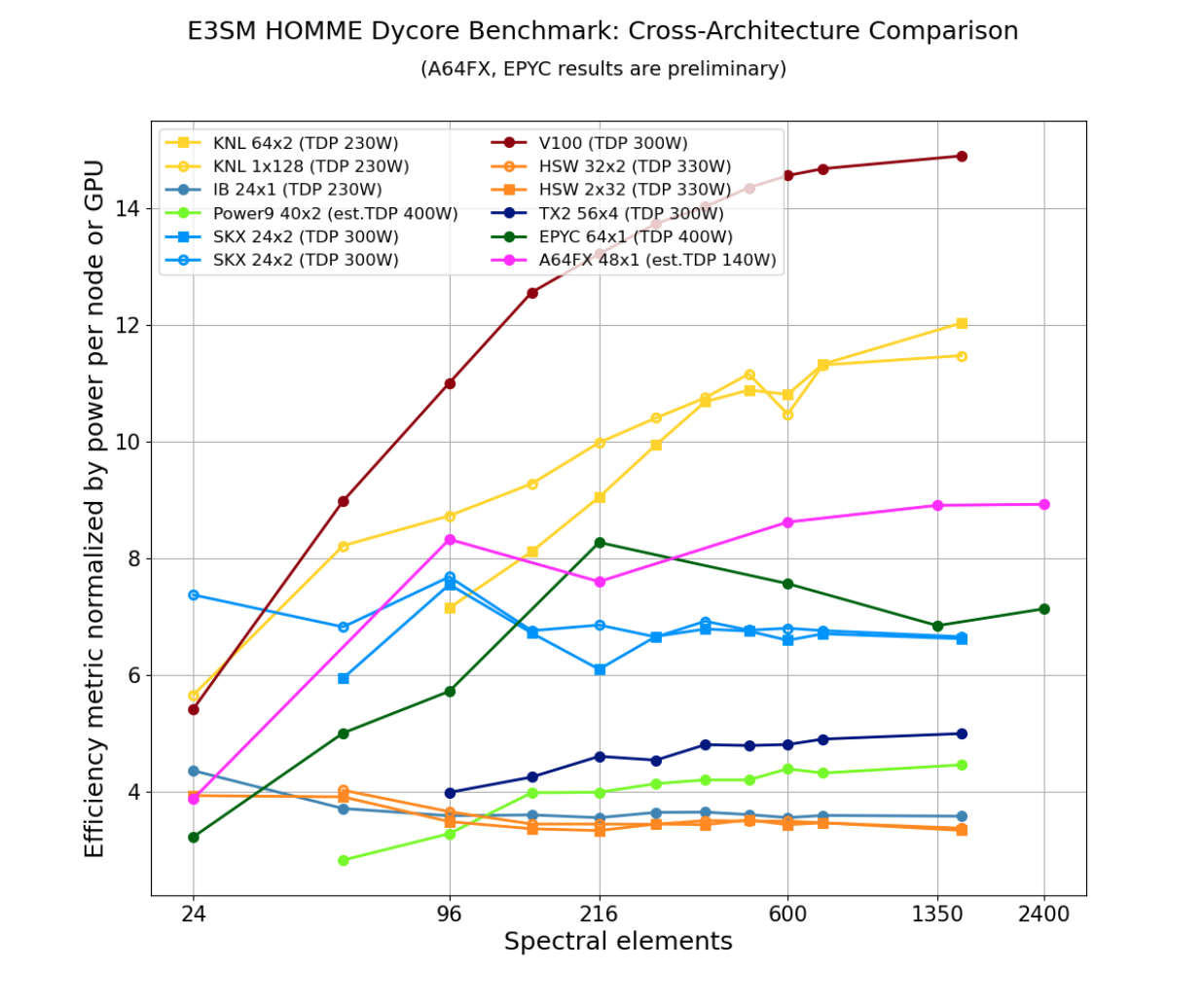
```
kloop_sedi_c2: do k = k_qxtop,k_qxbot,-kdir

   qc_notsmall_c2: if (qc_incld(k)>qsmall) then
      !-- compute Vq, Vn
      call get_cloud_dsd2(qc_incld(k),nc_incld(k),mu_c(k),rho(k),nu,dnu,   &
           lamc(k),tmp1,tmp2,lcldm(k))

      nc(k) = nc_incld(k)*lcldm(k)
      dum = 1._rtype / bfb_pow(lamc(k), bcn)
      V_qc(k) = acn(k)*bfb_gamma(4._rtype+bcn+mu_c(k))*dum/(bfb_gamma(mu_c(k)+4._rtype))
      V_nc(k) = acn(k)*bfb_gamma(1._rtype+bcn+mu_c(k))*dum/(bfb_gamma(mu_c(k)+1._rtype))

   endif qc_notsmall_c2
   Co_max = max(Co_max, V_qc(k)*dt_left*inv_dzq(k))

enddo kloop_sedi_c2
```

Ported to C++/Kokkos

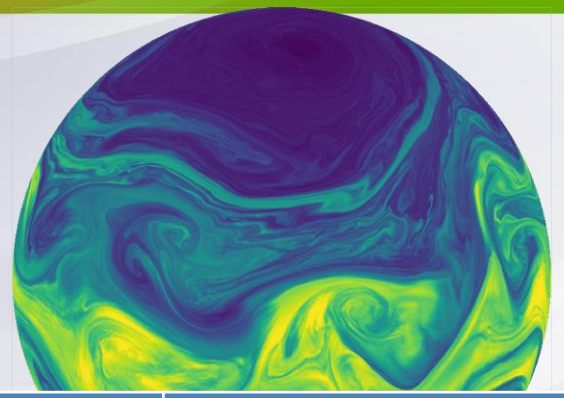
```
Kokkos::parallel_reduce(
  Kokkos::TeamThreadRange(team, kmax-kmin+1), [&] (int pk_, Scalar& lmax) {
    const int pk = kmin + pk_;
    const auto range_pack = scream::pack::range<IntSmallPack>(pk*Spack::n);
    const auto range_mask = range_pack >= kmin_scalar && range_pack <= kmax_scalar;
    const auto qc_gt_small = range_mask && qc_incld(pk) > qsmall;
    if (qc_gt_small.any()) {
      // compute Vq, Vn
      Spack nu, cdist, cdist1, dum;
      get_cloud_dsd2<false>(qc_gt_small, qc_incld(pk), nc_incld(pk), mu_c(pk), rho(pk), nu, dnu, lamc(pk), cdist,
      nc(pk).set(qc_gt_small, nc_incld(pk)*lcldm(pk));
      dum = 1 / (pack::pow(lamc(pk), bcn));
      V_qc(pk).set(qc_gt_small, acn(pk)*pack::tgamma(4 + bcn + mu_c(pk)) * dum / (pack::tgamma(mu_c(pk)+4)));
      if (log_predictNc) {
        V_nc(pk).set(qc_gt_small, acn(pk)*pack::tgamma(1 + bcn + mu_c(pk)) * dum / (pack::tgamma(mu_c(pk)+1)));
      }

      const auto Co_max_local = max(qc_gt_small, -1,
                                    V_qc(pk) * dt_left * inv_dzq(pk));
      if (Co_max_local > lmax)
        lmax = Co_max_local;
    }
}, Kokkos::Max<Scalar>(Co_max));
team.team_barrier();
```

DYCORE Performance: GPUs vs CPUs

1. Single-node or GPU benchmark from Bertagna et al., GMD 2019 (updated with additional CPUs)
2. Dycore running in a climate configuration (40 tracers) with simplified physics
3. **Performance metric per watt** (higher is better) as a function of workload per node or GPU
4. GPU (V100) provides significant performance increase, but only in the high-workload (per GPU) regime.
5. Future trends for GPUs and CPUs?



NGGPS Full System Benchmark



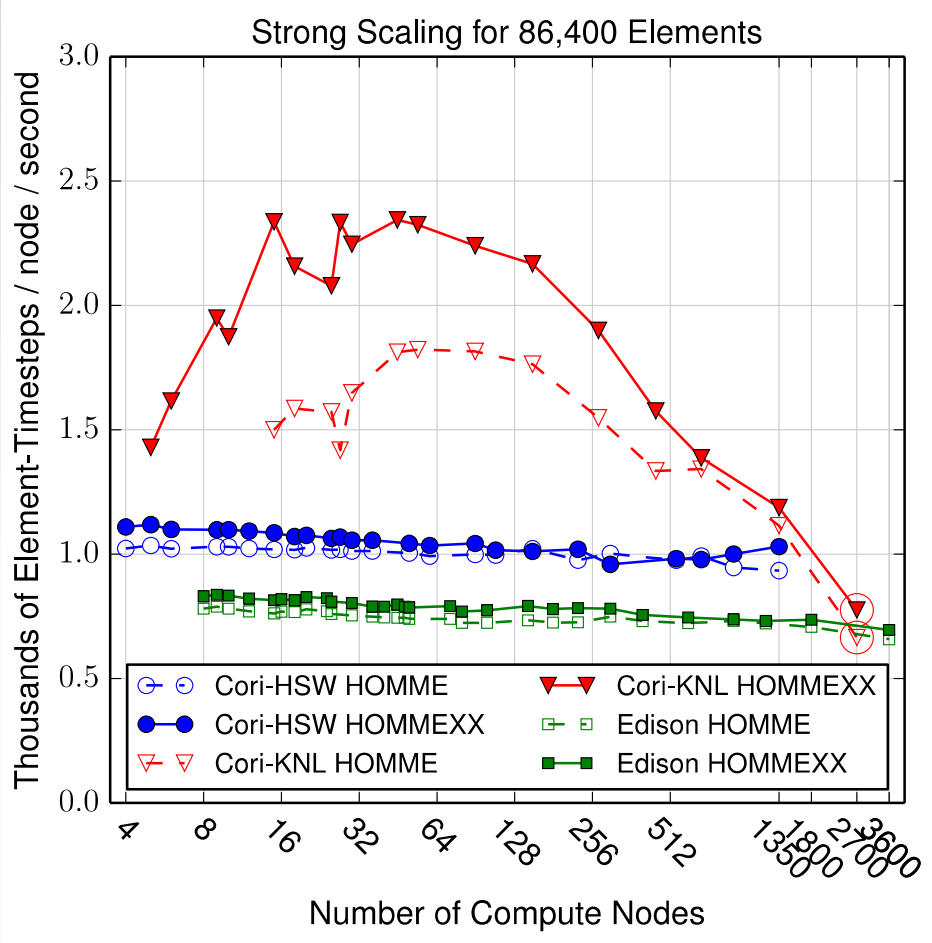
- NGGPS 3km (cloud resolving) benchmark problem. Benchmark from the National Weather Service
- Atmosphere dycore with realistic/operational configuration, 10 tracers and idealized physics
- Highlights from several generation of TOP500 computers and dycores.
- Dycore performance not keeping up with Linpack performance (but does track HPGC results)

Model	Computer (Linpack rating)	NGGPS 3km Benchmark
NOAA FV3 2015	Edison (2.6PF)	0.16 SYPD
HOMME (CESM) 2017	TaihuLight (125 PF)	0.34 SYPD
HOMME++ (SCREAM) 2021	Summit (200 PF)	1.38 SYPD
	Fugaku (440 PF)	?

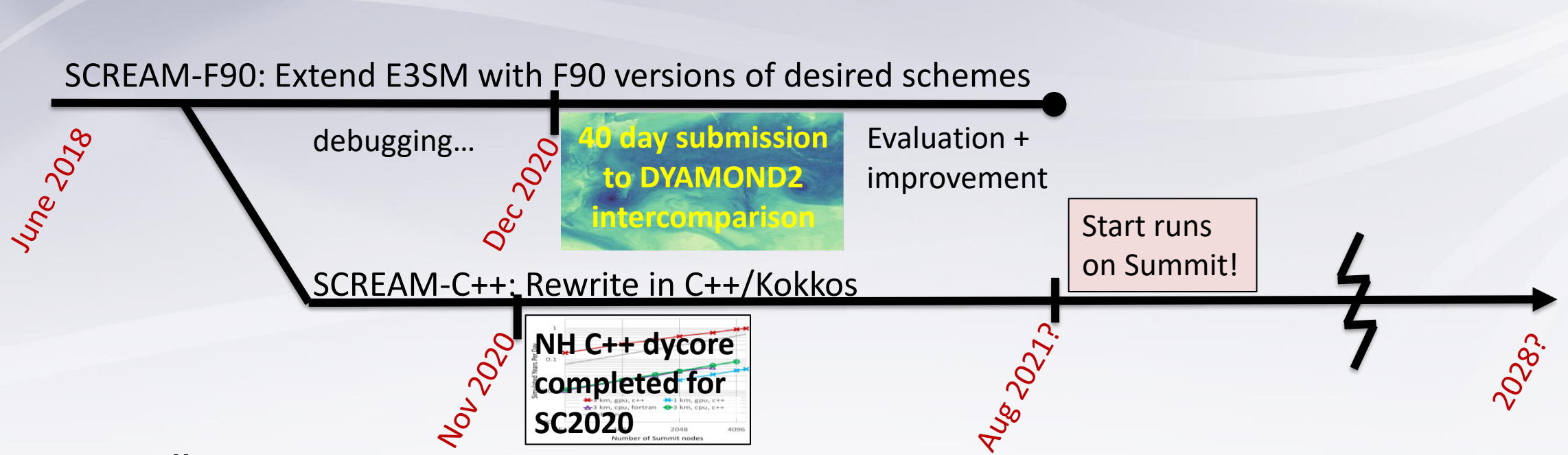
Extra Slides

DYCORE Performance on CPU systems FORTRAN vs C++

1. Key to good performance is vectorization
2. Fortran has excellent auto-vectorization. Competitive performance in C++ requires explicit vectorization via “packs” (Bertagna et al., GMD 2019)
3. Vectorized C++ code matches Fortran code on traditional CPUs, faster on Intel’s (since abandoned) KNL architecture
4. Vectorization not necessary on current GPUs, but this may change in the future



What is SCREAM?: The 2-Pronged Plan

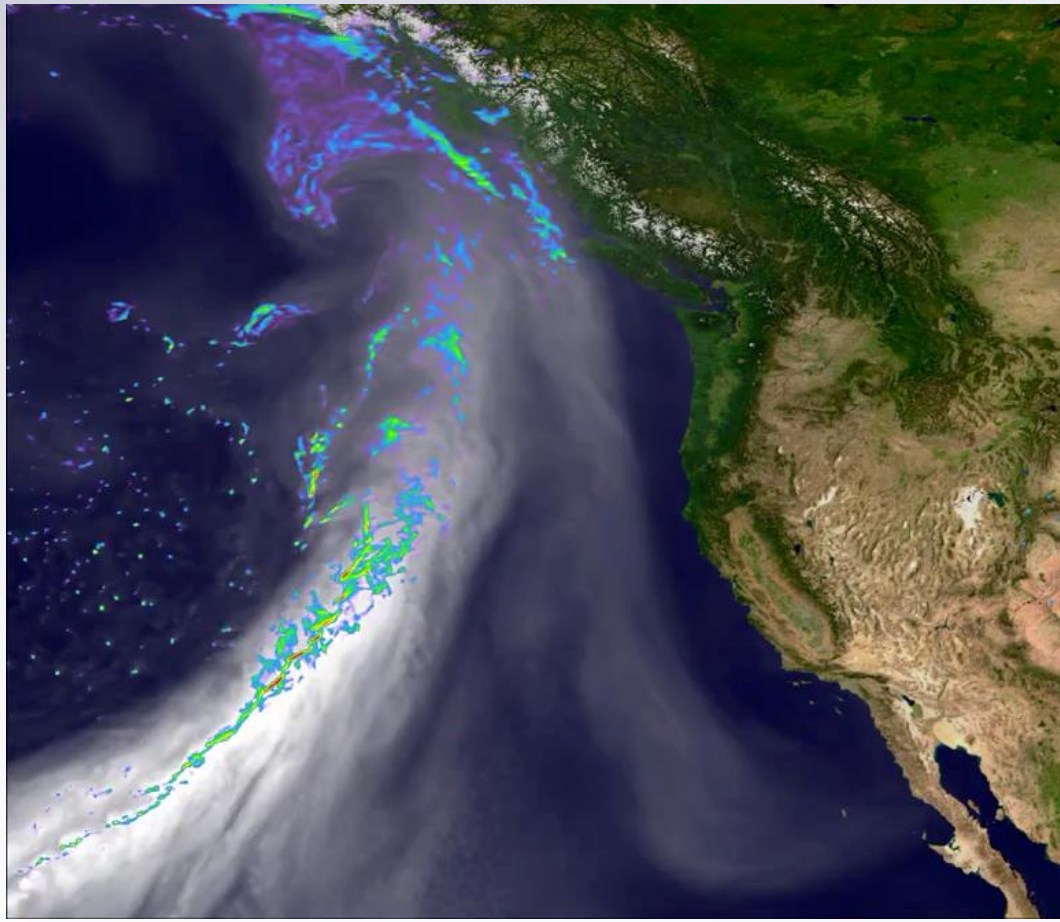


Important milestones:

1. F90 version used for DYAMOND2 simulation and overview paper (Dec 2020)
2. C++ dycore results submitted to SC2020 (Nov 2020)
3. Start production runs with full C++ model on Summit (Nov 2021?)

SCREAM-F90 DYAMOND2 Results

- A 40 day fixed-SST SCREAM-F90 run was submitted to the DYAMOND2 intercomparison
  - Results look great considering the preliminary nature of the project
  - C++ version should be scientifically identical



Movie: Precipitation (colors) and integrated water vapor (gray) for an atmospheric river from E3SM's DYAMOND2 simulation. By Paul Ullrich/UC Davis

Citation: Caldwell, Terai, and 28 coauthors, *Convection-Permitting Simulations with the E3SM Global Atmosphere Model*, Submitted to J. Adv. Model. Earth Syst. (2021). https://www.essoar.org/doi/abs/10.1002/essoar.10506530.1

Embracing Computer Science Best Practices

- A new model provides a new opportunity to do things right

- SCREAM was designed to make running and testing individual subroutines easy
  - Continuous integration with unit tests for all functions have helped catch many bugs (figure on right shows an example)

- Comprehensive design/tech docs are great for new users and for catching flaws/bugs

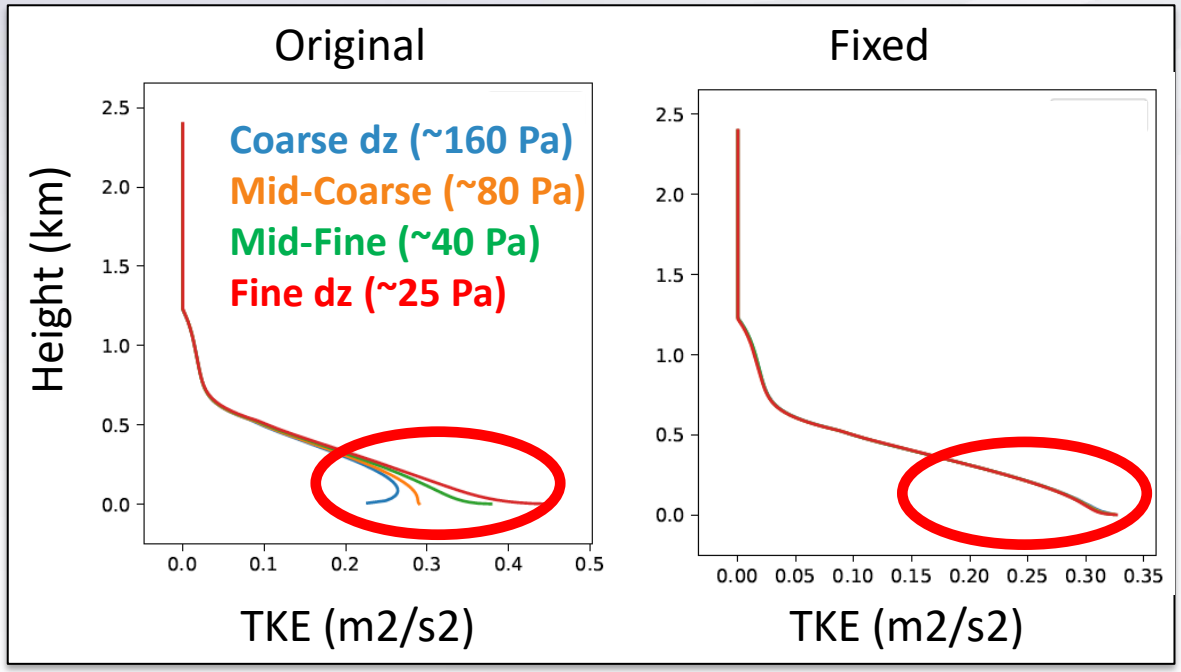
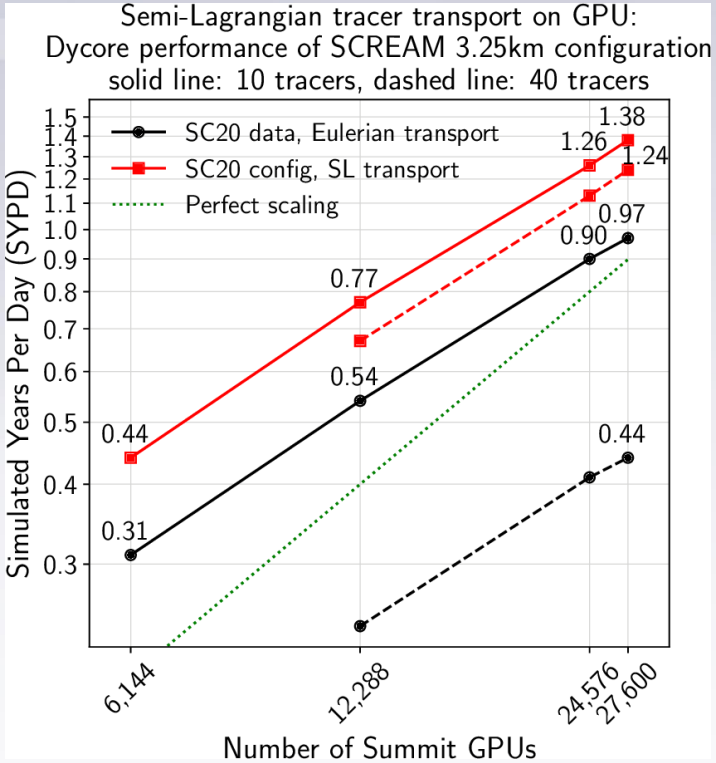


Fig: SHOC standalone simulations running the BOMEX test case (trade wind Cumulus) for 6 hrs with a variety of vertical resolutions

SCREAM Dycore performance on Summit

- Summit: DOE's 200PF GPU based pre-exascale system
- Using the US NWS cloud resolving global GCM benchmark (3km, 128L, 10 tracers, nonhydrostatic, production configuration)
- C++/Kokkos code obtains excellent scaling, **running at 1.38 SYPD on 27K GPUs**
- Such throughput will allow for climate length simulations



SCREAM dycore strong scaling out to 27K GPUs (NVIDIA V100). Recent adoption of SL tracer transport (solid red) improved performance to 1.38 SYPD. Dashed curves show performance with number of tracers increased from 10 to 40.

Animation of 500mb specific humidity in the idealized baroclinic instability test case used by the NWS benchmark.

Citation: L. Bertagna et. al., SC '20: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2020