Performance Portability Today: A NERSC Computational Scientist & AMReX Developer's Perspective

Performance Portability for ECP Applications and Software
ECP Community BoF Days
Feb 15, 2023

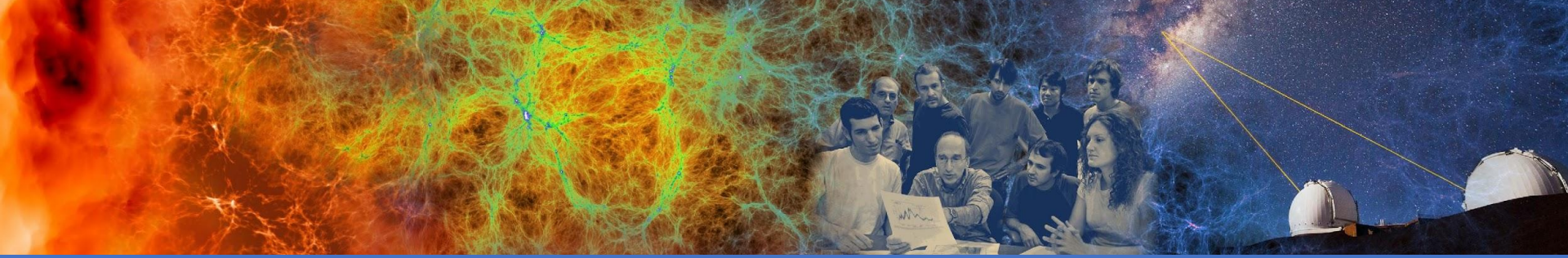
Kevin Gott
NERSC

Introduction

This talk has gone through a lot of different revisions.

Late last night, I realized *all* of these are highlights of a piece of my picture of performance portability today.

So, I'm going to try to do them all!



NERSC's Portability/Performance Strategy

Performance and Programming model strategy

- Support for mainstream languages and models on Perlmutter
  - C++, Fortran, python
  - CUDA, HIP, SYCL/DPC++
  - OpenMP, OpenACC
  - MPI
- NESAP, publications, docs.nersc.gov and user engagement
  - cast a wide net for hack-a-thons
  - partner with users to implement best practices
- Active engagement with vendors and community
  - e.g. LLVM, flang, standards groups, vendor NRE
- New “Programming Environment and Models” group, being led by Brandon Cook

Generic best practices

- Use optimized libraries
- parameterize data structures
  - don’t hard code SIMD, GPU warp size, etc
  - avoid hard coded magic numbers for things like tile or block sizes
- follow standards
  - compilers have varying levels of strictness, turn on the strict checks!
  - report any observed non-conformance in compilers to vendors
- minimize data movement
- adopt a kernel over loops mindset
  - i.e. replace sequential loops -> unsequenced parallel form

GPU programming at NERSC

Perlmutter Supports **All** Major GPU Programming Models

MPI: OpenMPI, MPICH, MVAPICH
CUDA: MILC, Chroma, HACC ...
CUDA FORTRAN: Quantum ESPRESSO, StarLord (AMREX)
OpenACC: VASP
OpenMP: QMCPACK, BerkeleyGW
Kokkos: LAMMPS, PELE, Chroma ...
Raja: SW4
HIP: Compatibility with Frontier applications
SYCL/DPC++: Compatibility with Aurora
Python, Spark, R, TensorFlow, Keras, Caffe PyTorch
Jupyter: Interactive Access to GPUs

NERSC, ALCF and Codeplay partnership on SYCL

- target SYCL 2020 (latest specification) support on Ampere A100 GPUs
- Open-source LLVM based compiler

OpenMP NRE partnership with NVIDIA

- Agreed upon subset of OpenMP features to be included in the PGI compiler
- OpenMP test suite created with micro-benchmarks, mini-apps, and the ECP SOLLVE V&V suite
- 5 NESAP/ECP application teams partnering with PGI to add OpenMP target offload directives
- Now part of production NVHPC SDK
  - Work continues on remaining issues / features required for some NESAP / ECP apps

And similar for OLCF / ALCF, too:

Scientific Libraries and Tools

Functionality	CPU	GPU	Notes
BLAS	**Cray LibSci**, **AMD BLIS**, PLASMA	**Cray LibSci_ACC**, **AMD roc/hipBLAS**, AMD rocAMD ROCm Tensile, **MAGMA**	MAGMA is open-source software led by the UTK Innovative Computing Laboratory
LAPACK	**Cray LibSci**, **AMD libFlame**, PLASMA	**Cray LibSci_ACC**, **AMD roc/hipSolver**, **MAGMA**	
ScaLAPACK	Cray LibSci	ECP SLATE, Cray LibSci_ACC	

Programming Environment

- Compilers Offered
  - Cray PE (C/C++ LLVM-based; Cray Fortran)
  - AMD ROCm (LLVM-based)
  - GCC
- Programming Languages & Models Supported (in which compilers)
  - C, C++, Fortran (all)
  - OpenMP (all) 5.0-5.2 in progress – most priority features complete, details vary
  - OpenACC (Cray PE – Fortran only & GCC)
  - HIP (AMD, Cray PE)
  - Kokkos/RAJA (all)
  - UPC (Cray PE, GCC)
- Transition Paths
  - CUDA: semi-automatic translation to HIP
  - CUDA Fortran: HIP kernels called from Fortran (a more portable approach)
    - CUDA Fortran kernels need to be translated to C++/HIP (manual process)
    - Fortran bindings to HIP and ROCm libraries and HIP runtime available through AMD's hipfort project

Items in green are also available on Summit

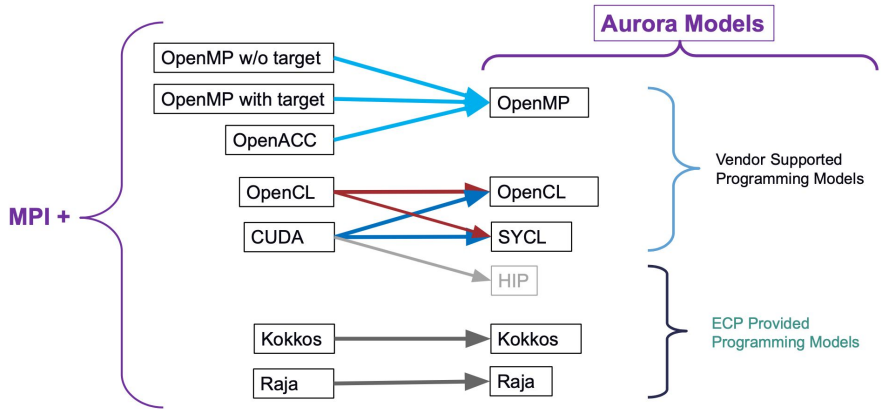
OAK RIDGE National Laboratory | LEADERSHIP COMPUTING FACILITY

FRONTIER

Aurora Software Stack (based on oneAPI)

- Languages
  - Fortran (2008 w/ OpenMP 5+)
  - C/C++ (w/ OpenMP 5+ and SYCL)
  - Python
- Libraries
  - oneAPI MKL (oneMKL)
  - oneAPI Deep Neural Network Library (oneDNN)
  - oneAPI Data Analytics Library (oneDAL)
  - MPI

Mapping of Programming Models to Aurora



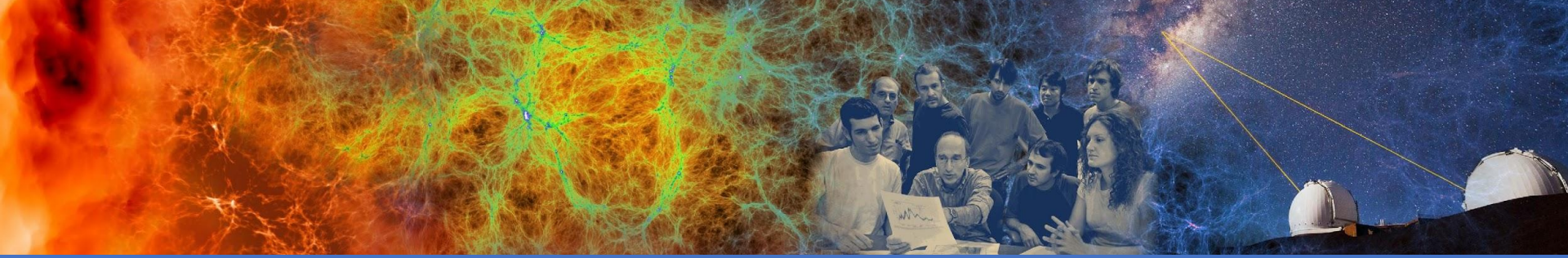
Results so far:

This strategy is working great: KPPs are being met, NESAP goals are being accomplished, best practices are being established, libraries are being optimized, users are being taught GPU strategies.

In 2022 NERSC saw strong adoption of Perlmutter and GPUs by NERSC users!

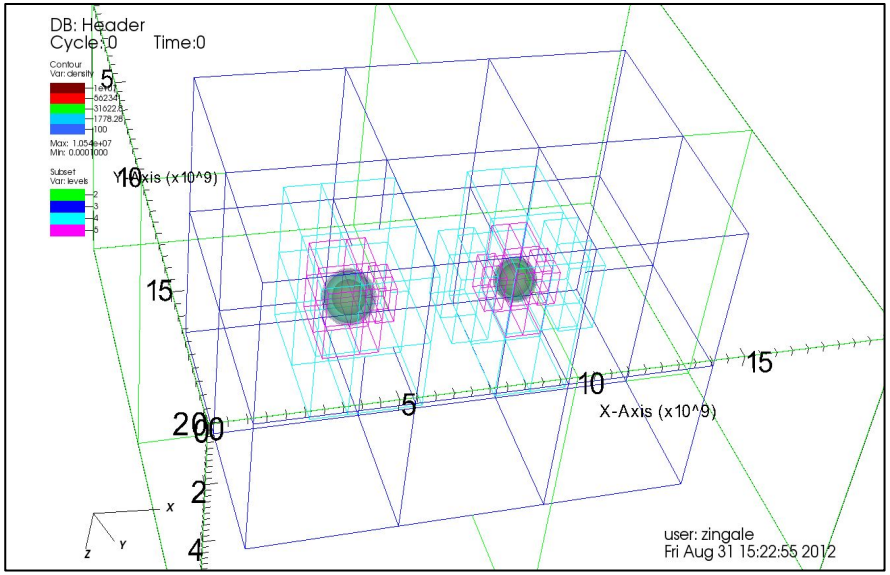
We have already seen a lot of great science come from Perlmutter and we're looking forward to doing even more!

If anything, there's too much to do!



WarpX / AMReX

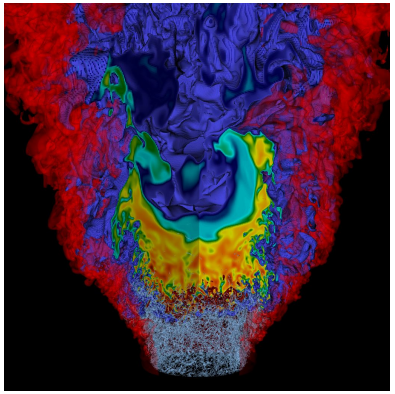
AMReX: Block-Structured AMR Software Framework



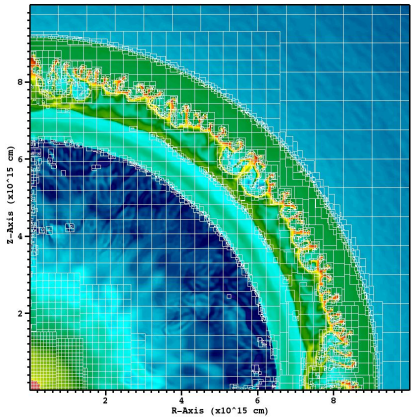
Initial setup for a pair of stars. Different color boxes show different levels of AMR refinement. Provided by the Mike Zingale and the CASTRO team.

- **Mesh, Particle, AMR**, Linear Solvers, Cut-Cell Embedded Boundary
- **Written in C++** (with optional Fortran interfaces)
  - **MPI + X**
  - OpenMP on CPU
  - CUDA, HIP, DPC++ internally on GPU
- Solution of parabolic and elliptic systems using **geometric multigrid solvers**
- Interoperability with Hypre, PETSc, SUNDIALS, HDF5 and other popular libraries.
- **Native I/O format** – supported by Visit, Paraview, yt, AmrVis.
- Fortran ordered data storage

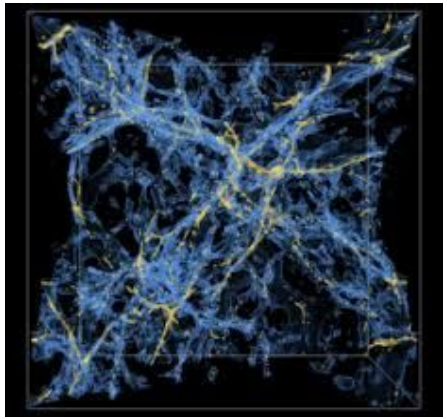
AMReX Across Science



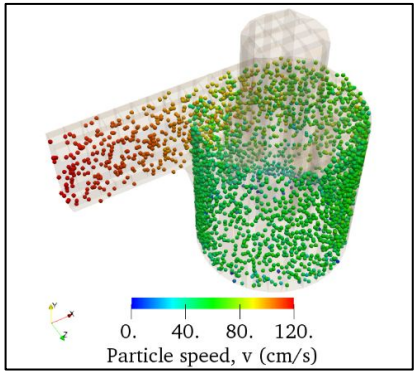
Combustion (Pele)



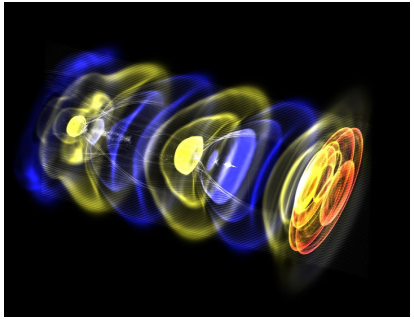
Astrophysics (Castro)



Cosmology (Nyx)



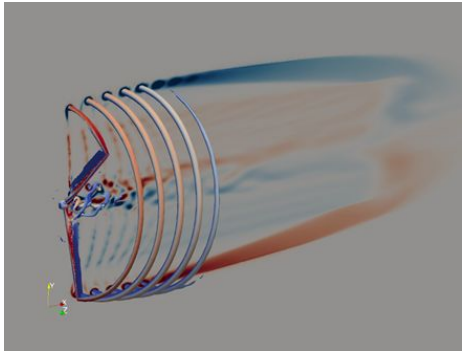
Multiphase flow (MFIX-Exa)



Accelerators (WarpX)

Other applications:

- Phase field models
- Microfluids
- Ionic liquids
- Non-Newtonian flow
- Fluid-structure interaction
- Shock physics
- Cellular automata
- Low Mach number astrophysics
- Defense science



AMRWind

Overview of AMReX / WarpX

AMReX is ready for hardware! (And has been for a while)

- Working with engineers to solve/research a variety of topics on NVIDIA, AMD and Intel architectures.
- Overall, AMReX and the applications are testing, getting good results, finding issues and reporting them to vendors: the community is contributing substantially!
- New AMReX applications being developed right now.

Example recent issues

Intel warp size changed from 16 to 32:

Works better for a large variety of kernels with most recent software and hardware.

Device Sync vs. Stream Synchs:

DeviceSync or a loop over StreamSync have two different performance outcomes on NVIDIA & AMD architechtures.

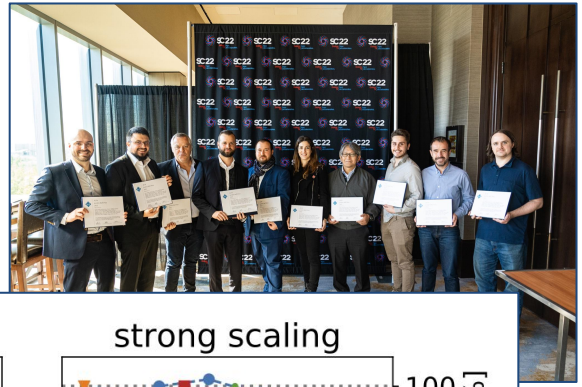
User team actively submitting PRs to get SYCL with CUDA backend fully functional:

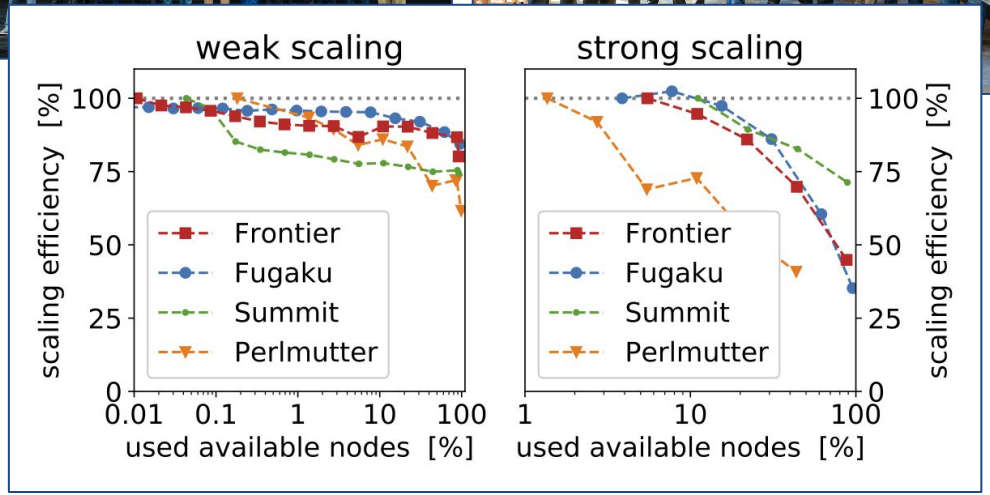
Very exciting and interesting. Uncovering lots to clean up.

Investigating long compile times in WarpX in SYCL:

Appears to be due to a forced inline function called multiple times.

WarpX won the 2022 Gordon Bell Award

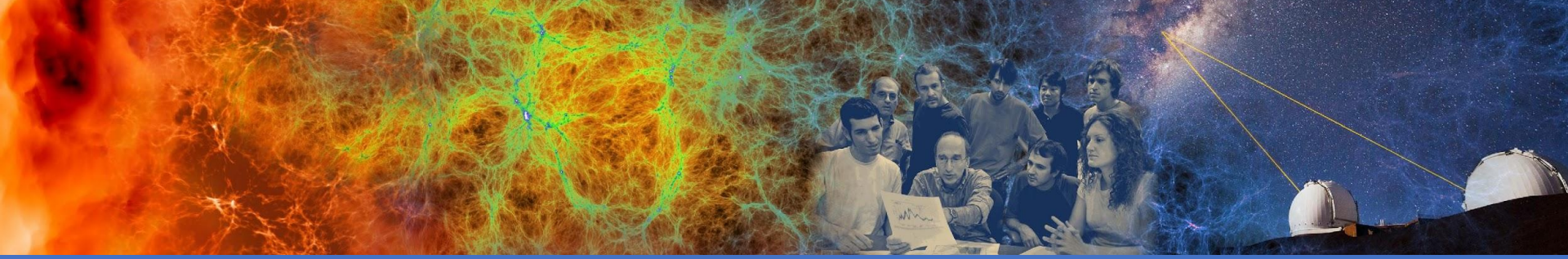




Accumulation of all the ECP efforts and work over the past ~3-4 years.

- Amazing application from Luca Fedeli's team at CEA.
- Incredible technical discussion from Axel Hubel, Jean-Luc Vay and WarpX team.
- Impressive results across 4 of the top 10 machines.

https://www.computer.org/csdl/proceedings-article/sc/2022/544400a025/1I0bSKaoECc



Stream-Triggered Communication

Stream Triggered Communication

Investigating using stream triggered MPI calls in AMReX's primary comm routines.

- Investigated to improve certain communication patterns.
- Also, may change the paradigm of AMReX codes – synchronization-free! – which could have application wide impact.

Working with a Georgia Tech graduate student, Yijian (Tim) Hu, to design, test, track many implementations, and generate a proof of concept in AMReX.

How it works

1) (Comm patterns are pre-calced and cached)
2) Post Receives – IRecv, alloc recvs.
3) Prepare Send Buffers – Calc sizes and allocate memory
   - Typically pinned + non-aware GPU has been fastest.
4) Pack Send Buffers
   - Single, fused launch to minimize launch overhead.
5) Post Sends – ISend
6) Do Local Work

No Wait

7) WaitAll for Sends
8) Unpack Sends
   - Single, fused launch again. Free Send Buffers
9) WaitAll for Recvs
10) Clean up – free Recv buffers, clear data object.

Finish

MPI calls are enqueued into a stream, allowing the CPU thread to do work while the critical code path (on the GPUs & NICs) is being completed.

- Requires also carefully enqueuing the freeing of buffers and cached information. (cudaHostFunc)
- Builds naturally into AMReX's comm pattern, which already includes pattern caching and non-blocking implementation.

Current Results

Got first successful results last week with NVIDIA's MPI-ACX over Cray-MPI:

```
*****************************
FB # 0 = 0.020994565    average: 0.020994645    clock: 0.020994645
FB # 1 = 0.020988905    average: 0.0209966045   clock: 0.041993209
FB # 2 = 0.020967132    average: 0.02098997633  clock: 0.062969929
FB # 3 = 0.020931765    average: 0.02097784575  clock: 0.083911383
FB # 4 = 0.020925833    average: 0.0209693428   clock: 0.104846714
FB # 5 = 0.020904793    average: 0.020960256    clock: 0.125761536
FB # 6 = 0.020927346    average: 0.02095712614  clock: 0.146699883
FB # 7 = 0.020918268    average: 0.02095356012  clock: 0.167628481
FB # 8 = 0.020944269    average: 0.02095357756  clock: 0.188582198
FB # 9 = 0.020930672    average: 0.0209522338   clock: 0.209522338
*****************************
```

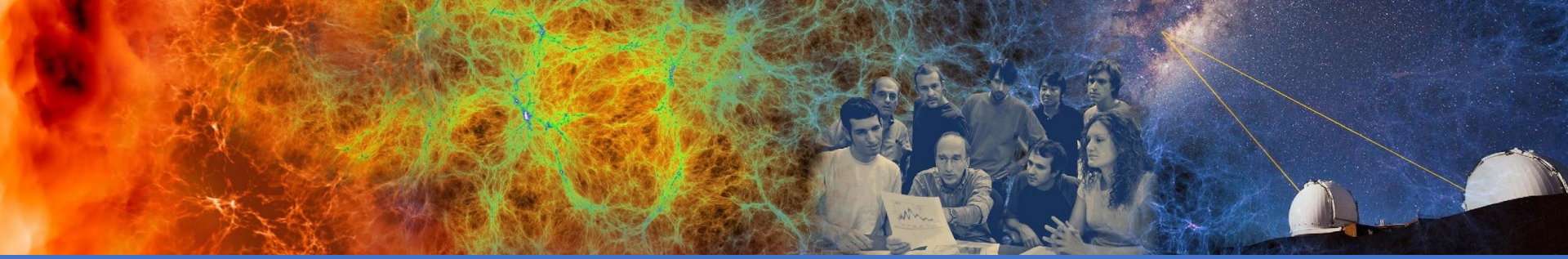
About 14% diff, ~ equal to CPU time in comms!

```
*****************************
FB # 0 =  started: 9.999999984e-08
FB # 0 =  launched: 0.004327438 average: 0.004327538    clock: 0.004327538
FB # 1 =  started: 0.004340422
FB # 1 =  launched: 0.009523309 average: 0.0069318655   clock: 0.013863731
FB # 2 =  started: 0.013876085
FB #0 completed = 0.02069591    average: 0.02069601     clock: 0.02069601
FB # 2 =  launched: 0.011178983 average: 0.008351689333 clock: 0.025055068
FB # 3 =  started: 0.025067973
FB # 3 =  launched: 0.0041478   average: 0.00730394325  clock: 0.029215773
FB # 4 =  started: 0.029226604
FB # 4 =  launched: 0.008718166 average: 0.007588954    clock: 0.03794477
FB # 5 =  started: 0.037957755
FB #1 completed = 0.033731163   average: 0.0190357925   clock: 0.038071585
FB # 5 =  launched: 0.004260298 average: 0.007036342167 clock: 0.042218053
FB # 6 =  started: 0.042228593
FB # 6 =  launched: 0.008609266 average: 0.007262551286 clock: 0.050837859
FB # 7 =  started: 0.050850985
FB #2 completed = 0.043506248   average: 0.01912744433  clock: 0.057382333
FB # 7 =  launched: 0.008684731 average: 0.0074419645   clock: 0.059535716
FB # 8 =  started: 0.059548471
FB # 8 =  launched: 0.004313941 average: 0.007095823556 clock: 0.063862412
FB # 9 =  started: 0.063873313
FB # 9 =  launched: 0.008669773 average: 0.0072543086   clock: 0.072543086
FB #3 completed = 0.049574692   average: 0.01866066625  clock: 0.074642665
FB #4 completed = 0.062576792   average: 0.0183606792   clock: 0.091803396
FB #5 completed = 0.073884634   average: 0.01864039817  clock: 0.111842389
FB #6 completed = 0.087641468   average: 0.01855286586  clock: 0.129870061
FB #7 completed = 0.09657007    average: 0.01842763188  clock: 0.147421055
FB #8 completed = 0.105032984   average: 0.01828682833  clock: 0.164581455
FB #9 completed = 0.117825679   average: 0.018169[illegible]992   clock: 0.181698992
Group # 1 = 0.118913617 average: 0.018278693
        clock: 0.18278693
```

Next Steps

Further proof of concept tests targeted to be completed next week for a talk at SIAM-CSE23 in two weeks:

- Test with different FABs, to capture potential gains including pattern calculation.
- Apply to Heat Equation (all mesh & Fill Boundary comms), as proof of user-friendly API.
- Test with MPICH – really want CUDA-Aware + ST.



A Moment of Recognition for our Code Developers (RSEs):

DOE's Code Developers (RSEs):

I posit: one of the biggest reasons for the ECP's success.

- Plan paths forward to accommodate everyone's needs and desires.
- Develop and improve build systems.
- Track a large number of libraries, features, tools and compiler statuses.
- Interact with users, vendors, scientists and PIs.
- Fix bugs, respond to issues inline with the overall code plan.
- Test new features, libraries, opportunities and workflows.
- Maintain, expand and respond to CI testing suites.
- Work with other developers to create missing tools and resources.
- Teach users software design, engineering and abstractions.

Often, our primary sources of knowledge, leadership, mentoring and stability.

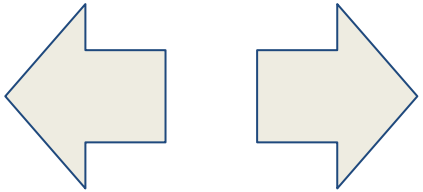
DOE's Code Developers (RSEs):

I also posit: this may be one of the most complex and difficult times to be a DOE HPC code developer.

Users:

Highly motivated to use the new features.

- Learned about the methods and techniques.
- Systems exist and cool science is at their fingertips.
- Want to try new things themselves, start new research, develop their careers.



Code Teams, ECP & Development Goals:

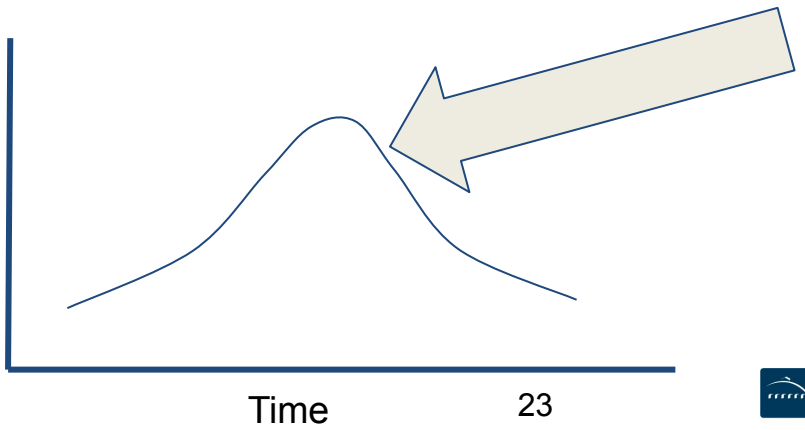
Testbeds are all available, but:

- Still in a period or rapid development and change.
- Continue to uncover new issues, bugs and feature requests as cross-comparisons continue.
- Continue to track, monitor and

Simple Example: GPU-Aware MPI

AMReX has been tracking GPU-Aware MPI performance for years.

- Initial addition was early in development, focusing on Summit.
- Added to AMReX as a run-time flag: use pinned comm buffer or a device comm buffer.
- Regularly tested to see if GPU-Aware is better – i.e. at hackathons.
- Just recently has started to see improvement, but it's still not ready to be turned on everywhere by default.
- But, interactions with other things are growing



GPU-Aware's research lifecycle is sitting right about here.

And so are a LOT of other features, tests, bug, issues, etc.

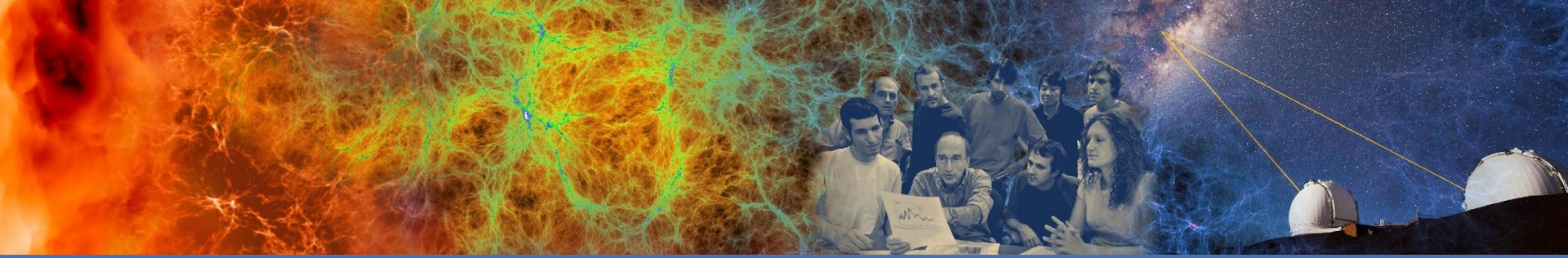
DOE's Code Developers (RSEs):

Sadly, I don't have any amazing ideas on how to help.

But, I would love to see the community discuss how to best support, develop and grow our RSE community into a more efficient, effective, permanent and stable (as they've done for our scientific code ecosystem).

I heavily support (if I had the power, I would require) a substantial discussion of the impact of RSEs in ECP's final report.

If anyone has ideas on how to make their lives better, reduce their workload, make things more efficient for them, please let me know.



Conclusions

Optimistic outlook with lots of room to grow:

This is an amazing time to be working in HPC, both as a scientist and as a developer.

ECP has been massively influential to users, researchers, and lots of cool new stuff is coming in the near future.

Hug (with permission) your local RSEs (or, help out in any other way that you can).