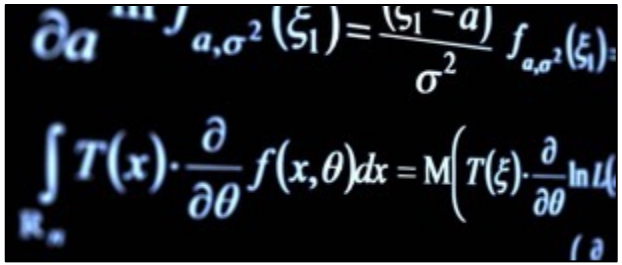
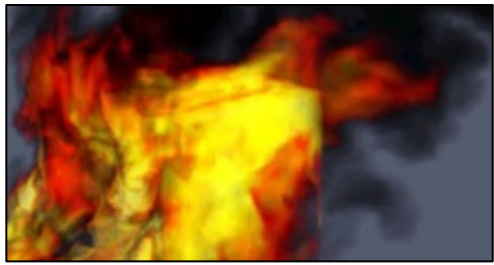
Exceptional service in the national interest

Sandia National Laboratories



$$\frac{\partial}{\partial a} f_{a,\sigma^2}(\xi_1) = \frac{(\xi_1 - a)}{\sigma^2} f_{a,\sigma^2}(\xi_1)$$

$$\int T(x) \cdot \frac{\partial}{\partial \theta} f(x,\theta) dx = \mathrm{M}\left(T(\xi) \cdot \frac{\partial}{\partial \theta} \ln L \right.$$



Kokkos An Overview

Unclassified Unlimited Release

Christian R. Trott, - Center for Computing Research

Sandia National Laboratories/NM

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

SAND2020-9180 C

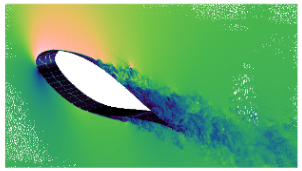
Cost of Porting Code

Sandia National Laboratories

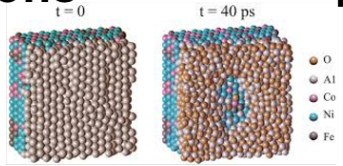
10 LOC / hour ~ 20k LOC / year

- Optimistic estimate: 10% of an application is modified to adopt an on-node Parallel Programming Model
- Typical Apps: 300k – 600k Lines
  - 500k x 10% => Typical App Port 2.5 Man-Years
- Large Scientific Libraries
  - E3SM: 1,000k Lines x 10% => 5 Man-Years
  - Trilinos: 4,000k Lines x 10% => 20 Man-Years

Applications



Computational Fluid Dynamics

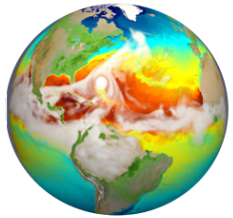


Molecular Dynamics

Libraries

Electro Magnetics

Frameworks



Climate Simulation

Kokkos

ORNL Frontier
AMD GPUs

LANL/SNL Trinity
Intel CPUs

ANL Aurora
Intel GPUs

Riken Fugaku
ARM CPUs

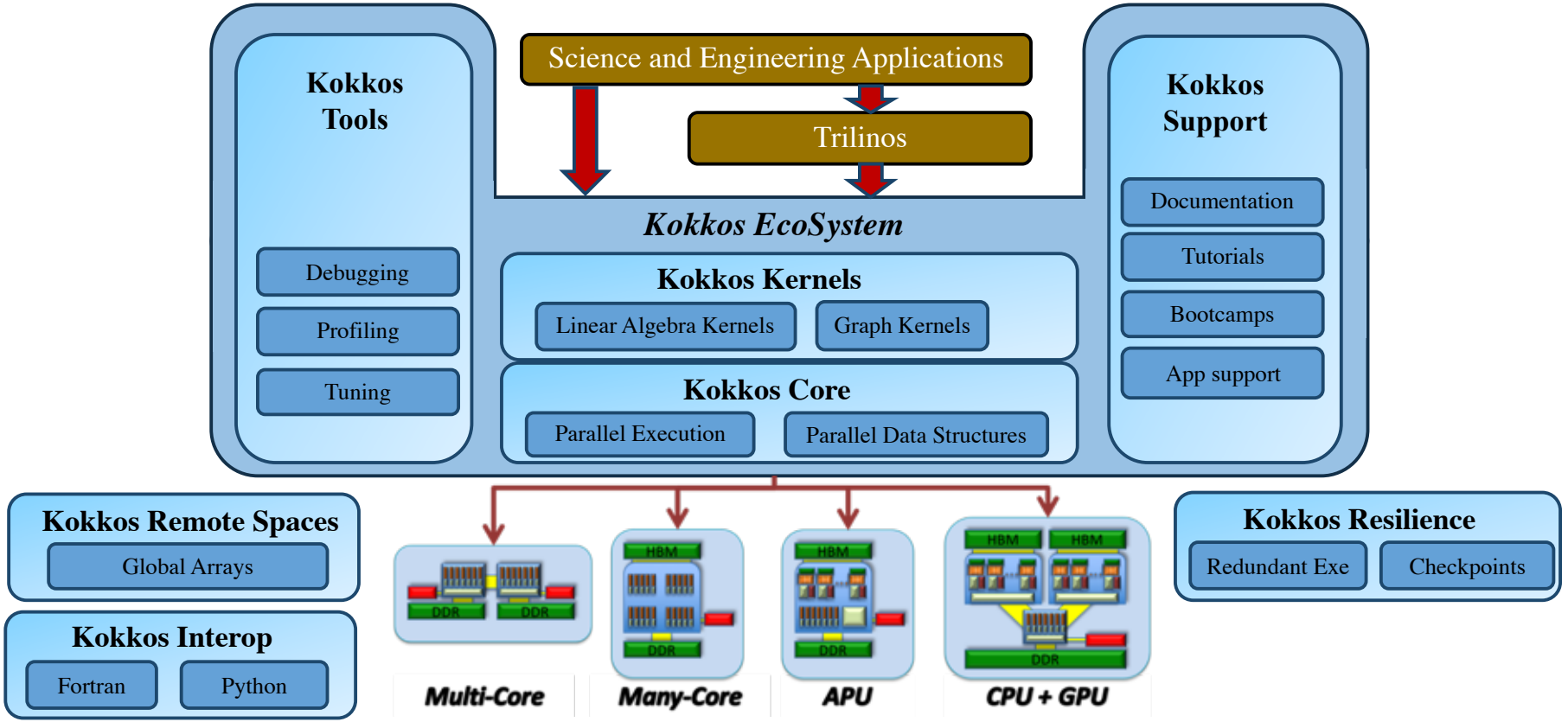
NERSC Permutter
NVIDIA GPUS

What is Kokkos?

Sandia National Laboratories

- A C++ Programming Model for Performance Portability
  - Implemented as a template library on top of CUDA, OpenMP, HPX, …
  - Aims to be descriptive not prescriptive
  - Aligns with developments in the C++ standard
- Expanding solution for common needs of modern science/engineering codes
  - Math libraries based on Kokkos
  - Tools which enable insight into Kokkos
- It is Open Source
  - Maintained and developed at https://github.com/kokkos
- It has many users at wide range of institutions.

Kokkos EcoSystem



Transitioning To Community Project

- ***Kokkos Core***: 15 Developers (8 SNL)
- More code contributions from non-SNL
  - >50% of commits from non-Sandians
- Sandia leads API design
- Other labs lead backend implementations
- Other subprojects largely by Sandia so far

Papers:

[The Kokkos EcoSystem: Comprehensive Performance Portability For High Performance Computing](#)
C.R. Trott et al., Computing in Science & Engineering, 2021
[Kokkos 3: Programming Model Extensions for the Exascale Era](#)
C.R. Trott et al., IEEE Transactions on Parallel and Distributed Systems, 2021
[Kokkos: Enabling manycore performance portability through polymorphic memory access patterns](#)
H.C. Edwards et al., Journal of Parallel and Distributed Computing, 2014

Kokkos Uptake

Sandia National Laboratories

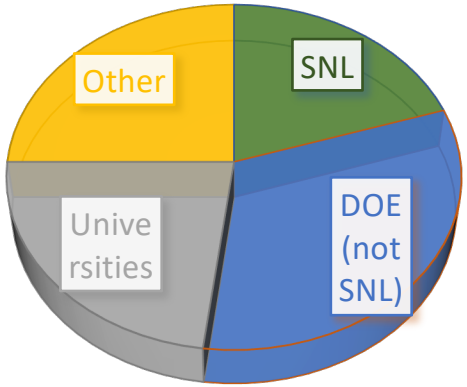
ECP Critical Dependencies

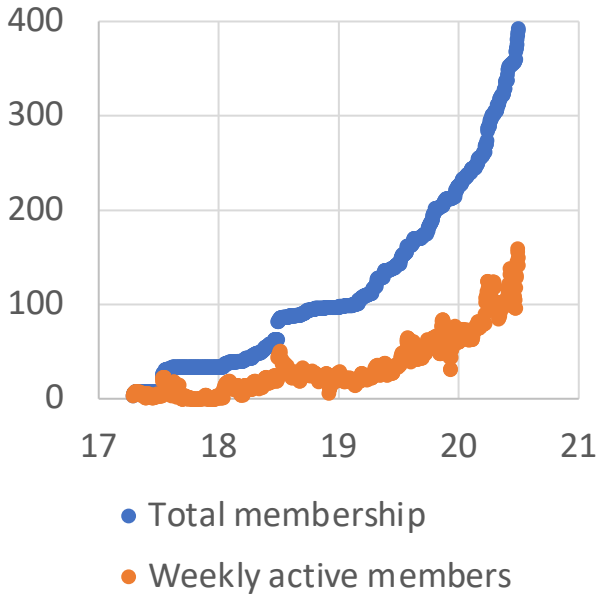
MPI	60
LLVM	57
C++	41
OpenMP	34
LAPACK CUDA	24 22
Fortran HDF5	21 21
BLAS	21
Kokkos	18
C	14
ALPINE	12

hypre	11
DAV-SDK	11
VTK-m	11
Trilinos	10
ADIOS	8
SPACK	8
SCALAPACK FFT	8 7
OpenACC	7
MPI-IO	6
PnetCDF	6
Tau	6

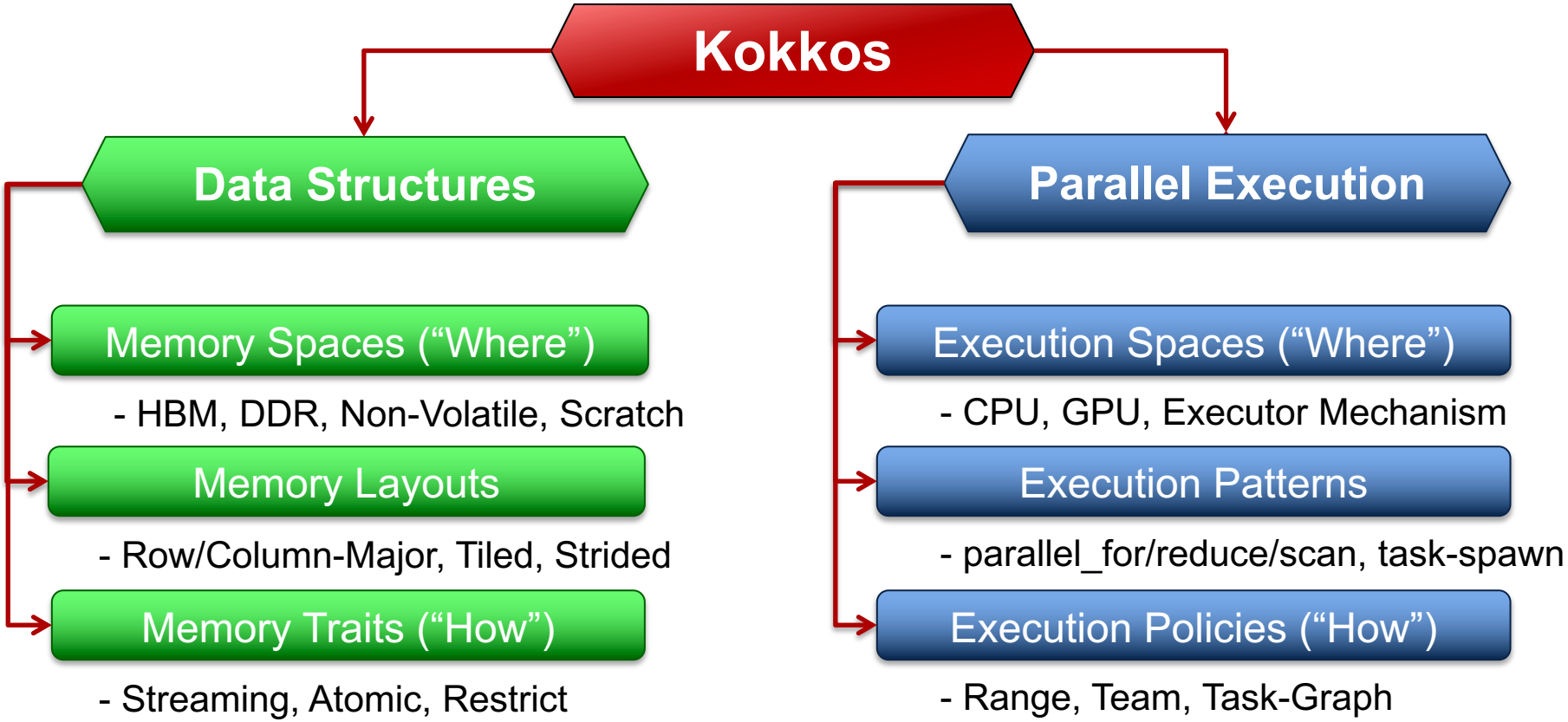
Kokkos Slack Users

- 660 registered users
  - 90 Institutions
  - Every continent
    - (-Antarctica)





Kokkos Core Abstractions

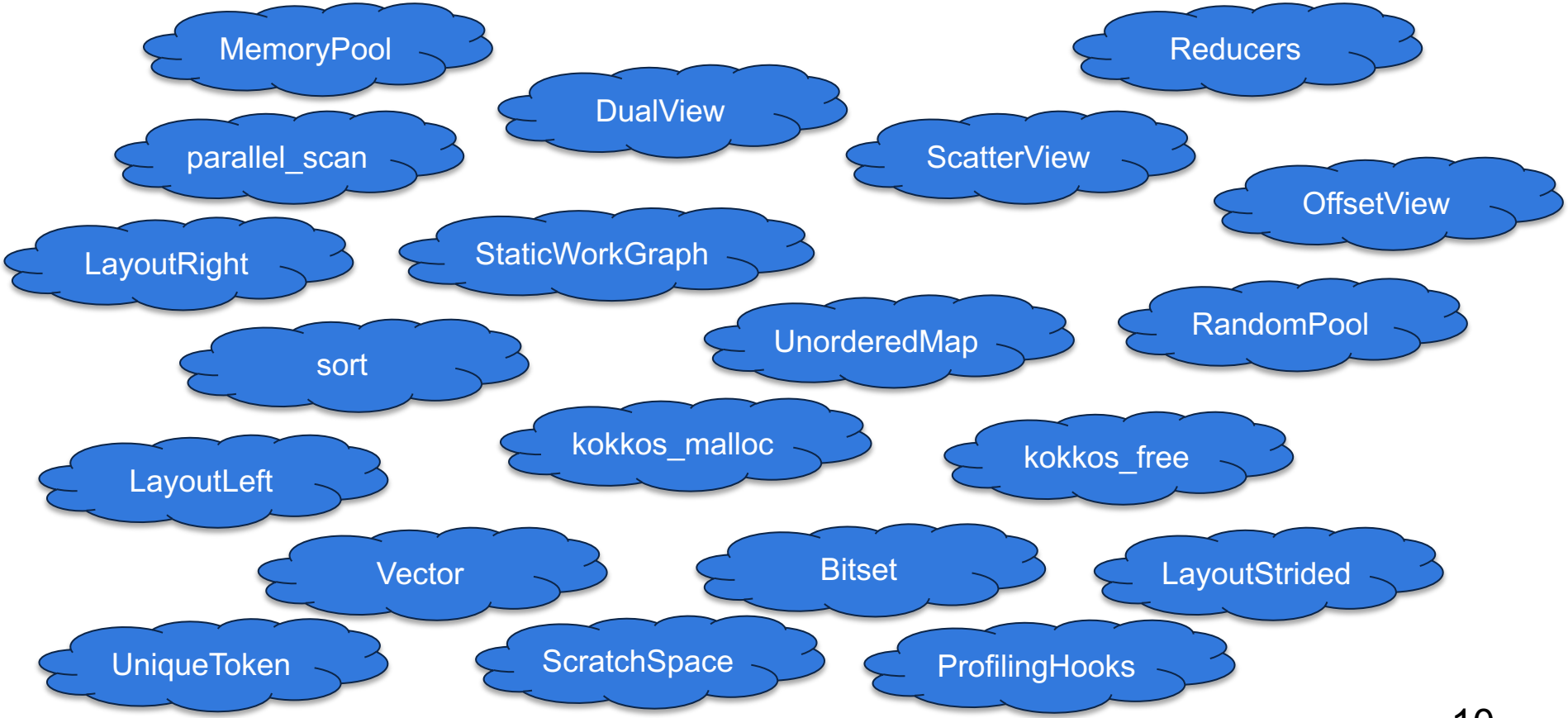


Kokkos Core Capabilities

Sandia National Laboratories

Concept	Example
Parallel Loops	**parallel_for**(N, **KOKKOS_LAMBDA** (int i) { ...BODY… });
Parallel Reduction	**parallel_reduce**(**RangePolicy**<ExecSpace>(0,N), **KOKKOS_LAMBDA** (int i, double& upd) { …BODY... upd += ... }, Sum<>(result));
Tightly Nested Loops	**parallel_for**(**MDRangePolicy**<**Rank**<3> > ({0,0,0},{N1,N2,N3},{T1,T2,T3}, **KOKKOS_LAMBDA** (int i, int j, int k) {…BODY...});
Non-Tightly Nested Loops	**parallel_for**(**TeamPolicy**<**Schedule**<**Dynamic**>>(N, TS), **KOKKOS_LAMBDA** (Team team) { … COMMON CODE 1 ... **parallel_for**(**TeamThreadRange**(team, M(N)), [&] (int j) { ... INNER BODY... }); … COMMON CODE 2 ... });
Task Dag	**task_spawn**(**TaskTeam**(scheduler , priority), **KOKKOS_LAMBDA** (Team team) { … BODY });
Data Allocation	**View**<double**, Layout, MemSpace> a(“A”,N,M);
Data Transfer	**deep_copy**(a,b);
Atomics	atomic_add(&a[i],5.0); View<double*,MemoryTraits<AtomicAccess>> a(); a(i)+=5.0;
Exec Spaces	Serial, Threads, OpenMP, Cuda, HPX (experimental), HIP (experimental), OpenMPTarget (experimental)

More Kokkos Capabilities



Example: Conjugent Gradient Solver

- Simple Iterative Linear Solver
- For example used in MiniFE
- Uses only three math operations:
  - Vector addition (AXPBY)
  - Dot product (DOT)
  - Sparse Matrix Vector multiply (SPMV)
- Data management with Kokkos Views:

```
View<double*,HostSpace,MemoryTraits<Unmanaged> > h_x(x_in, nrows);
View<double*> x("x",nrows);
deep_copy(x,h_x);
```

CG Solve: The AXPBY

- Simple data parallel loop: Kokkos::parallel_for
- Easy to express in most programming models
- Bandwidth bound
- Serial Implementation:

```
void axpby(int n, double* z, double alpha, const double* x,
                             double beta,  const double* y) {
  for(int i=0; i<n; i++)
    z[i] = alpha*x[i] + beta*y[i];
}
```

Parallel Pattern: for loop

String Label: Profiling/Debugging

Execution Policy: do n iterations

Loop Body

Iteration handle: integer index

- Kokkos Implementation:

```
void axpby(int n, View<double*> z, double alpha, View<const double*> x,
                                   double beta,  View<const double*> y) {
  parallel_for("AXpBY", n, KOKKOS_LAMBDA ( const int i) {
    z(i) = alpha*x(i) + beta*y(i);
  });
}
```

CG Solve: The Dot Product

- Simple data parallel loop with reduction: Kokkos::parallel_reduce
- Non trivial in CUDA due to lack of built-in reduction support
- Bandwidth bound
- Serial Implementation:

```
double dot(int n, const double* x, const double* y) {
  double sum = 0.0;
  for(int i=0; i<n; i++)
    sum += x[i]*y[i];
  return sum;
}
```

Parallel Pattern: loop with reduction

Iteration Index + Thread-Local Red. Varible

- Kokkos Implementation:

```
double dot(int n, View<const double*> x, View<const double*> y) {
  double x_dot_y = 0.0;
  parallel_reduce("Dot",n, KOKKOS_LAMBDA (const int i,double& sum) {
    sum += x[i]*y[i];
  }, x_dot_y);
  return x_dot_y;
}
```

CG Solve: Sparse Matrix Vector Multiply

- Loop over rows
- Dot product of matrix row with a vector
- Example of Non-Tightly nested loops
- Random access on the vector (Texture fetch on GPUs)

Outer loop over matrix rows

Inner dot product row x vector

```
void SPMV(int nrows, const int* A_row_offsets, const int* A_cols,
          const double* A_vals, double* y, const double* x) {
  for(int row=0; row<nrows; ++row) {
    double sum = 0.0;
    int row_start=A_row_offsets[row];
    int row_end=A_row_offsets[row+1];
    for(int i=row_start; i<row_end; ++i) {
      sum += A_vals[i]*x[A_cols[i]];
    }
    y[row] = sum;
  }
}
```

CG Solve: Sparse Matrix Vector Multiply

Sandia National Laboratories

```
void SPMV(int nrows, View<const int*> A_row_offsets,
          View<const int*> A_cols, View<const double*> A_vals,
          View<double*> y,
          View<const double*, MemoryTraits< RandomAccess>> x) {

  // Performance heuristic to figure out how many rows to give to a team
  int rows_per_team = get_row_chunking(A_row_offsets);

  parallel_for("SPMV:Hierarchy", TeamPolicy< Schedule< Static > >
       ((nrows+rows_per_team-1)/rows_per_team,AUTO,8),
    KOKKOS_LAMBDA (const TeamPolicy<>::member_type& team) {

    const int first_row = team.league_rank()*rows_per_team;
    const int last_row = first_row+rows_per_team<nrows? first_row+rows_per_team : nrows;

    parallel_for(TeamThreadRange(team,first_row,last_row),[&] (const int row) {
      const int row_start=A_row_offsets[row];
      const int row_length=A_row_offsets[row+1]-row_start;

      double y_row;
      parallel_reduce(ThreadVectorRange(team,row_length),[&] (const int i, double& sum) {
        sum += A_vals(i+row_start)*x(A_cols(i+row_start));
      } , y_row);
      y(row) = y_row;
    });
  });
}
```

Enable Texture Fetch on x

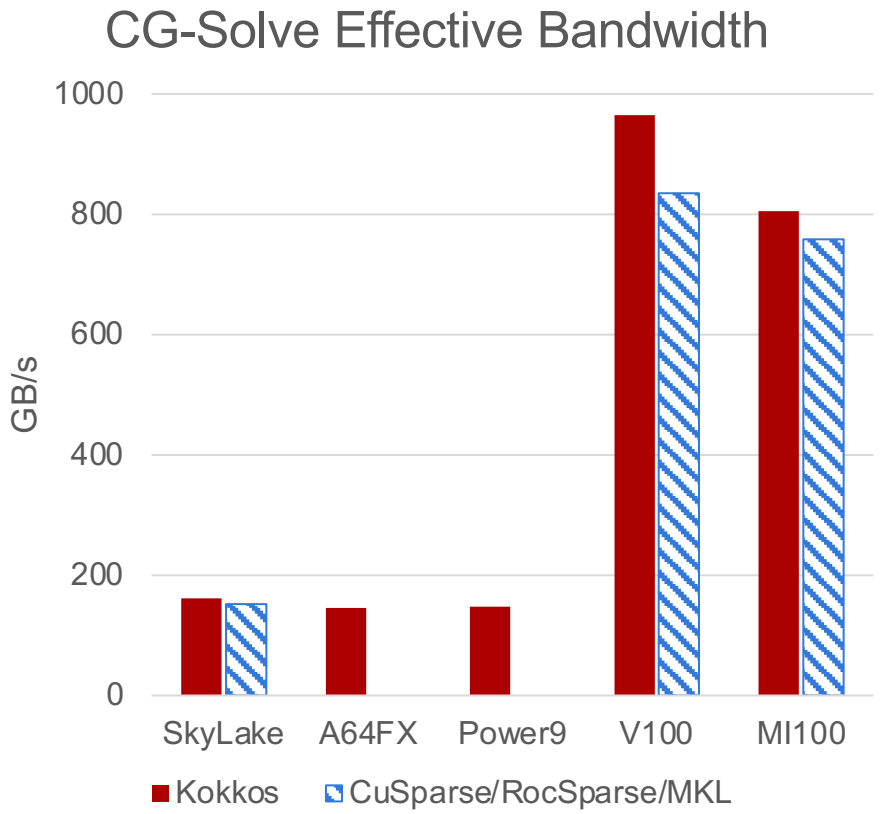
Row x Vector dot product

Distribute rows in workset over team-threads

Team Parallelism over Row Worksets

CG Solve Performance

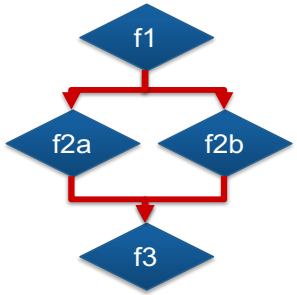
- CG-Solve as discussed above
- Also try replacing SPMV with TPL
- Running 100x100x100 heat conduction problem
  - ”MiniFE” Proxyapp setup
- Measure effective Bandwidth
  - Algorithmical memory ops per time
- Why is this beating vendor libs?
  - Its complicated, but a real effect



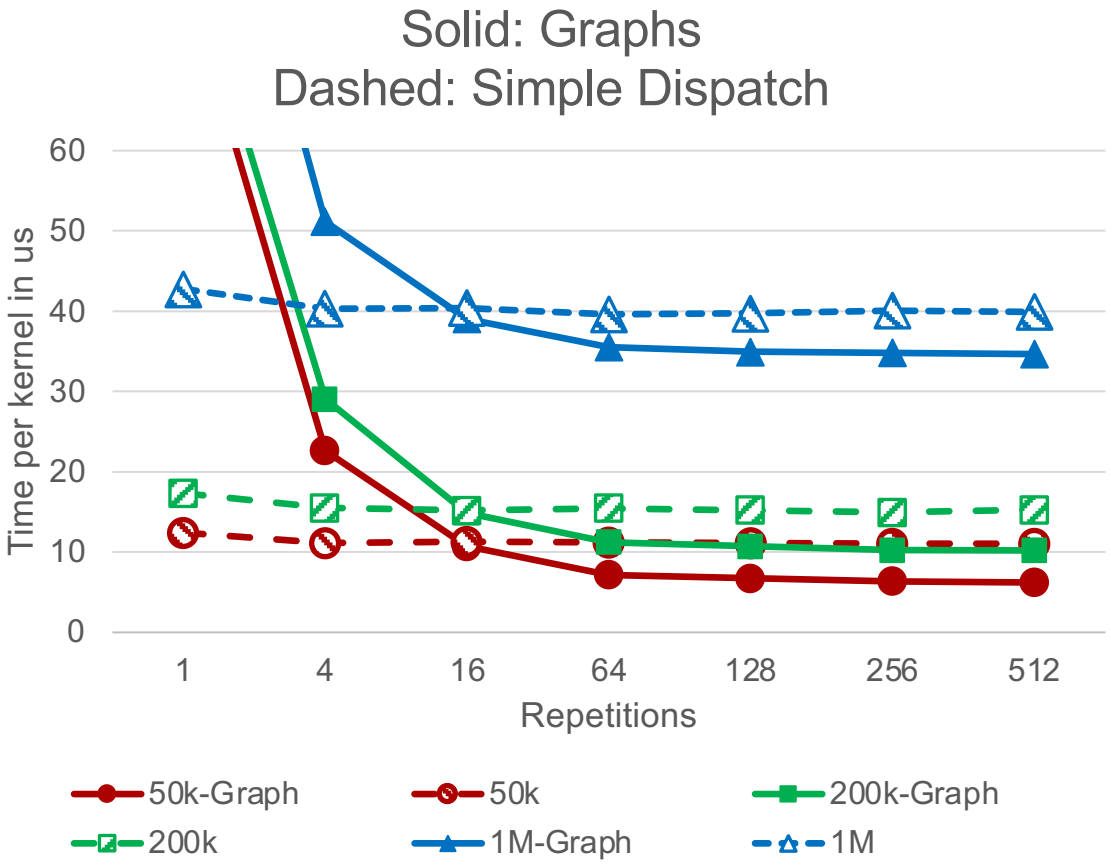
Tracking New Capabilities: Graphs

- Build static graphs of kernels
  - Can use CUDAGraphs as backend
  - Allows repeated dispatch
- Helps with Latency Limited codes
  - Cuts down on launch latency
  - Can leverage streams to overlap work
  - Infers overlapping from dependencies
- Prototype release part of Kokkos 3.3

```
const auto graph = Kokkos::Experimental::create_graph(
  [=](auto root) {
  auto f1 = root.then_parallel_for(
    Kokkos::RangePolicy<>(0, 1), KOKKOS_LAMBDA(long) {…});
  auto f2a = f1.then_parallel_for(
    Kokkos::RangePolicy<>(0, 1), KOKKOS_LAMBDA(long) {…});
  auto f2b = f1.then_parallel_for(
    Kokkos::RangePolicy<>(0, 1), KOKKOS_LAMBDA(long) {…});
  when_all(f2a, f2b).then_parallel_reduce(
    Kokkos::RangePolicy<>(0, 1), KOKKOS_LAMBDA(long) {…}
    result);
});

while(result()>threshold {
  graph.submit();
  graph.get_execution_space().fence();
}
```



Benchmark the Example



Can reuse graph:

- In solver iterations
- Between solves if matrix structure unchanged

>100 reuses could be realistic

Throughput Improvement:

- 50K 78%
- 200k 49%
- 1M 15%

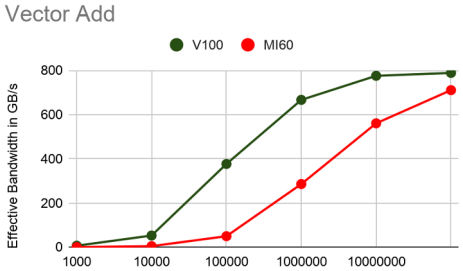
Next: look at reducing graph creation time

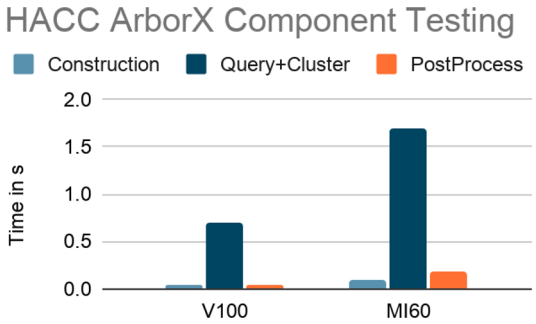
AMD Support Status

Sandia National Laboratories

Frontier/El Capitan: HIP and OpenMP 5

- Primary development of HIP at ORNL
- Most Capabilities ready
  - Fine grained tasking is missing
- **PR testing for Kokkos on AMD GPUs in place**
- ArborX, Cabana, LAMMPS working with HIP
- Trilinos (4,000k lines HPC library) works.





We are largely using our own machines (not ECP EAS), with the public software stack from Intel and AMD.

Kokkos 3.3 (Dec 2020):

- *HIP is largely feature complete*

Kokkos 3.4 (April2021):

- *SYCL Support Largely Complete*

Kokkos Core functionality porting to Frontier nearly complete

Aurora Support Status

Programming Models: DPC++/SYCL + OpenMP 5

- Primary work for DPC++ at ANL and ORNL
  - Shifted ORNL team members from HIP to DPC++ since HIP is in much better shape
- DPC++/SYCL was long blocked by compiler issues
  - Worked with Intel to get those fixed
  - Now primary capabilities are merged to develop branch
- **PR testing DPC++/SYCL in place**
  - Intel DPC++/SYCL testing is done on NVIDIA GPUs …
  - Leverages clang capability to target different backend

We are largely using our own machines (not ECP EAS), with the public software stack from Intel and AMD.

Kokkos 3.3 (Dec 2020):

- *OpenMPTarget and DPC++ have most primary capabilities working*

Kokkos 3.4 (April 2021):

- *DPC++/SYCL is largely feature complete*

Initial Kokkos Core functionality porting to Aurora done.

Kokkos Support

Sandia National Laboratories

- The Kokkos Lectures
  - 8 lectures covering most aspects of Kokkos
  - 15 hours of recordings
  - > 500 slides
  - >20 exercises
- Extensive Wiki
  - API Reference
  - Programming Guide
- Slack as primary direct support

https://kokkos.link/the-lectures

- Module 1: Introduction
  - Introduction, Basic Parallelism, Build System
- Module 2: Views and Spaces
  - Execution and Memory Spaces, Data Layout
- Module 3: Data Structures and MDRangePolicy
  - Tightly Nested Loops, Subviews, ScatterView,…
- Module 4: Hierarchical Parallelism
  - Nested Parallelism, Scratch Pads, Unique Token
- Module 5: Advanced Optimizations
  - Streams, Tasking and SIMD
- Module 6: Language Interoperability
  - Fortran, Python, MPI and PGAS
- Module 7: Tools
  - Profiling, Tuning , Debugging, Static Analysis
- Module 8: Kokkos Kernels
  - Dense LA, Sparse LA, Solvers, Graph Kernels

Kokkos Kernels

- BLAS, Sparse and Graph Kernels on top of Kokkos and its View abstraction
  - Scalar type agnostic, e.g. works for any types with math operators
  - Layout and Memory Space aware
- Can call vendor libraries when available
- Views contain size and stride information => Interface is simpler

```
// BLAS
int M,N,K,LDA,LDB; double alpha, beta; double *A, *B, *C;
dgemm('N','N',M,N,K,alpha,A,LDA,B,LDB,beta,C,LDC);
```

```
// Kokkos Kernels
double alpha, beta; View<double**> A,B,C;
gemm('N','N',alpha,A,B,beta,C);
```

- Interface to call Kokkos Kernels at the teams level (e.g. in each CUDA-Block)

```
parallel_for("NestedBLAS", TeamPolicy<>(N,AUTO), KOKKOS_LAMBDA (const team_handle_t& team_handle) {
  // Allocate A, x and y in scratch memory (e.g. CUDA shared memory)
  // Call BLAS using parallelism in this team (e.g. CUDA block)
  gemv(team_handle,'N',alpha,A,x,beta,y)
});
```

Kokkos Tools

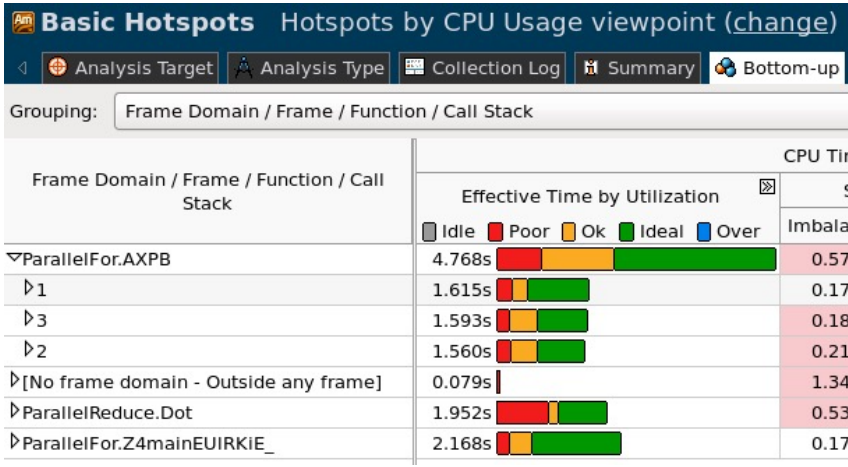
Sandia National Laboratories

- Profiling
  - New tools are coming out
  - Worked with NVIDIA to get naming info into their system
- Auto Tuning (Under Development)
  - Internal variables such as CUDA block sizes etc.
  - User provided variables
  - Same as profiling: will use dlopen to load external tools
- Debugging (Under Development)
  - Extensions to enable clang debugger to use Kokkos naming information
- Static Analysis (Under Development)
  - Discover Kokkos anti patterns via clang-tidy

Kokkos-Tools Profiling & Debugging

Sandia National Laboratories

- Performance tuning requires insight, but tools are different on each platform
- KokkosTools: Provide common set of basic tools + hooks for 3rd party tools
- Common issue: abstraction layers obfuscate profiler output
  - Kokkos hooks for passing names on
  - Provide Kernel, Allocation and Region
- No need to recompile
  - Uses runtime hooks
  - Set via env variable



Kokkos Tools Integration with 3rd Party

- Profiling Hooks can be subscribed to by tools, and currently have support for TAU, Caliper, Timemory, NVVP, Vtune, PAPI, and SystemTAP, with planned CrayPat support
- HPCToolkit also has special functionality for models like Kokkos, operating outside of this callback system

TAU Example:

TAU: ParaProf: Statistics for: node 0, thread 0 - examinimd_ompt_phase.ppk

Name	Exclusive TIME	Inclusive TIME	Calls	Child Calls
.TAU application	0.143	96.743	1	832
Comm::exchange	0.001	0.967	6	142
Comm::exchange_halo	0.001	4.702	6	184
Comm::update_halo	0.004	31.347	95	1,330
Kokkos::parallel_for CommMPI::halo_update_pack [device=0]	0.002	0.506	190	190
Kokkos::parallel_for CommMPI::halo_update_self [device=0]	0.003	0.597	380	380
Kokkos::parallel_for CommMPI::halo_update_unpack [device=0]	0.002	0.97	190	190
MPI_Irecv()	0.001	0.001	190	0
MPI_Send()	29.268	29.268	190	0
MPI_Wait()	0.001	0.001	190	0
OpenMP_Implicit_Task	0.041	1.985	760	760
OpenMP_Parallel_Region parallel_for<Kokkos::RangePolicy<CommMPI::Ta	0	0.504	190	190
OpenMP_Parallel_Region parallel_for<Kokkos::RangePolicy<CommMPI::Ta	0.08	0.968	190	190
OpenMP_Parallel_Region void Kokkos::parallel_for<Kokkos::RangePolicy<	0.001	0.594	380	380
OpenMP_Sync_Region_Barrier parallel_for<Kokkos::RangePolicy<CommMI	0.489	0.489	190	0
OpenMP_Sync_Region_Barrier parallel_for<Kokkos::RangePolicy<CommMI	0.875	0.875	190	0
OpenMP_Sync_Region_Barrier void Kokkos::parallel_for<Kokkos::RangePol	0.58	0.58	380	0

Kokkos Tools Static Analysis

- clang-tidy passes for Kokkos semantics
- Under active development, requests welcome
- IDE integration

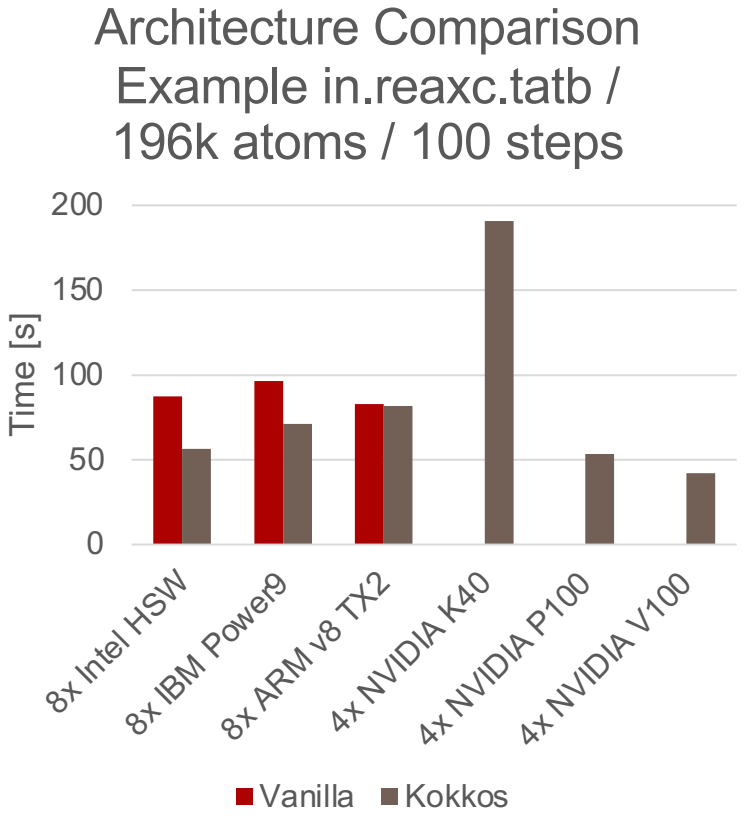
```
// Base case
Kokkos::parallel_for(
    TPolicy, KOKKOS_LAMBDA(TeamMember const& t) {
      int a = 0;

      Kokkos::parallel_for(TTR(t, 1), [&](int i) { Lambda capture modifies reference capture variable 'a' that is a local
        a += 1;
        cv() += 1;
      });
    });

// One with variable Lambda
Kokkos::parallel_for(
    TPolicy, KOKKOS_LAMBDA(TeamMember const& t) {
      int b       = 0;
      auto lambda = [&](int i) { Lambda capture modifies reference capture variable 'b' that is a local
        b += 1;
        cv() += 1;
      };
      Kokkos::parallel_for(TTR(t, 1), lambda);
    });
```

LAMMPS

Questions: Stan Moore

Sandia National Laboratories

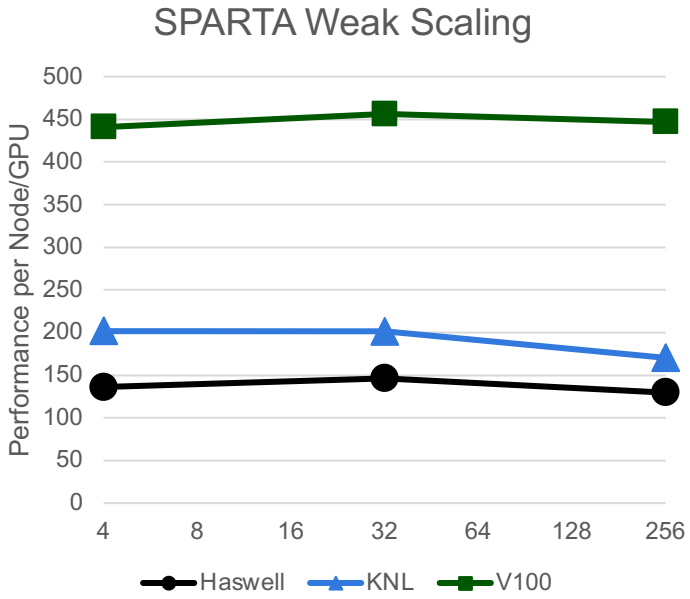
- Widely used Molecular Dynamics Simulations package
- Focused on Material Physics
- Over 500 physics modules
- Kokkos covers growing subset of those
- REAX is an important but very complex potential
  - USER-REAXC (Vanilla) more than 10,000 LOC
  - Kokkos version ~6,000 LOC
  - LJ in comparison: 200LOC
  - Used for shock simulations



Sparta: Production Simulation at Scale

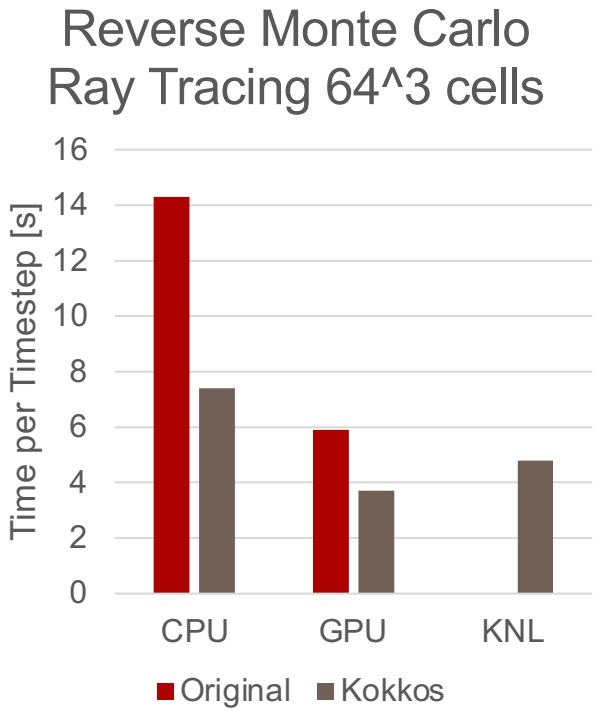
Sandia National Laboratories

- **S**tochastic **PA**rallel **R**arefied-gas **T**ime-accurate **A**nalyzer
- A direct simulation Monte Carlo code
- Developers: *Steve Plimpton, Stan Moore, Michael Gallis*
- Only code to have run on all of Trinity
  - 3 Trillion particle simulation using both HSW and KNL partition in a single MPI run (~20k nodes, ~1M cores)
- Benchmarked on 16k GPUs on Sierra
  - Production runs now at 5k GPUs
- Co-Designed Kokkos::ScatterView



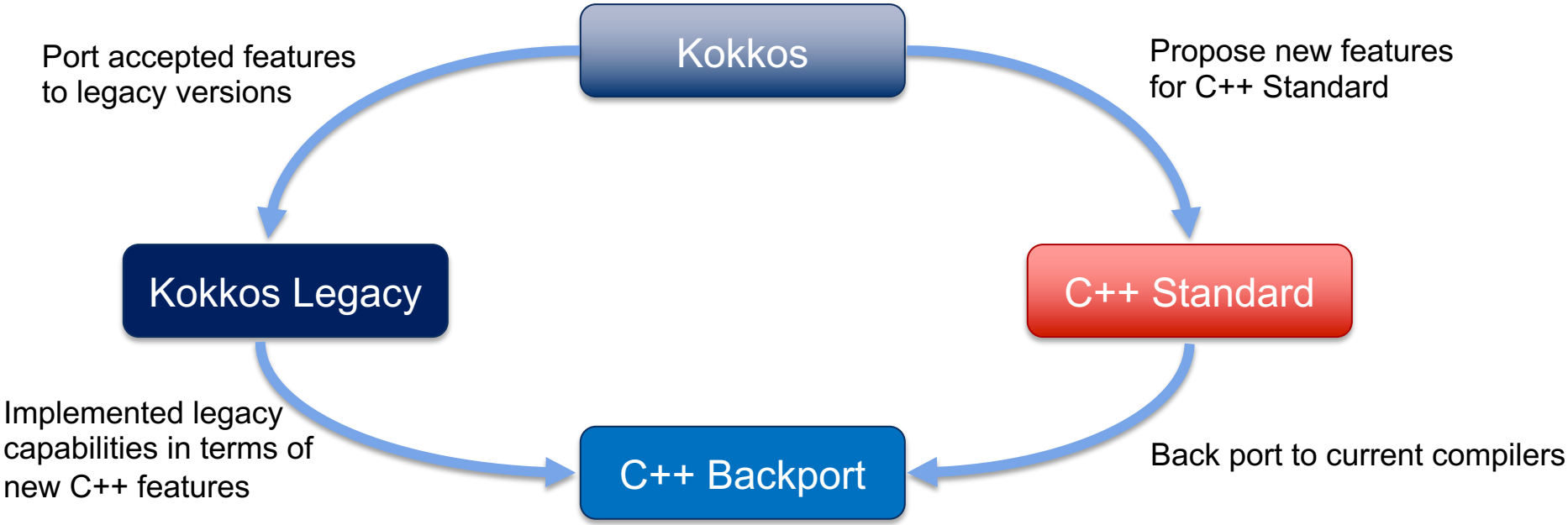
Uintah

- System wide many task framework from University of Utah led by Martin Berzins
- Multiple applications for combustion/radiation simulation
- Structured AMR Mesh calculations
- Prior code existed for CPUs and GPUs
- Kokkos unifies implementation
- Improved performance due to constraints in Kokkos which encourage better coding practices



Questions: Dan Sunderland

Kokkos - C++ Standard integration cycle



C++ Features in the Works

Sandia National Laboratories

- First success: **atomic_ref**<T> in C++20
  - Provides atomics with all capabilities of atomics in Kokkos
  - **atomic_ref**(a[i])+=5.0; instead of **atomic_add**(&a[i],5.0);
- Next thing: **Kokkos::View** => **std::mdspan**
  - Provides customization points which allow all things we can do with **Kokkos::View**
  - Better design of internals though! => Easier to write custom layouts.
  - Also: arbitrary rank (until compiler crashes) and mixed compile/runtime ranks
  - We hope will land early in the cycle for C++23 (i.e. early in 2020)
  - Production reference implementation: https://github.com/kokkos/mdspan
- Also C++23: Executors and **Basic Linear Algebra:** https://github.com/kokkos/stdblas

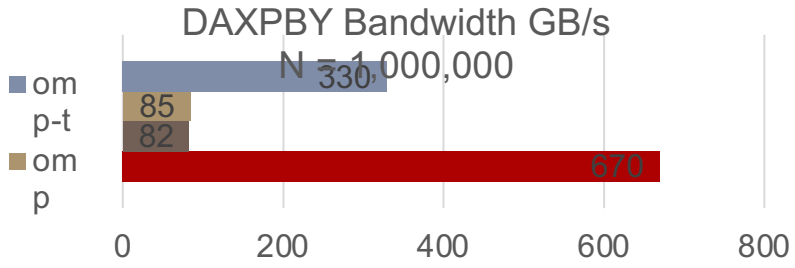

Sandia
National
Laboratories

OpenMPTarget Status

- Most capabilities are now working
  - Until earlier in 2020 limited by compiler bugs
- Using primarily main line clang/llvm
  - Are also working with Intel and NVIDIA
  - Started working with AMD and HPE
- Next phase: concentrating on performance
  - C++ performance very fragile
  - We are ramping up collaboration with compiler engineers

Vector Add Performance Illustration

- Simple problem, should clearly be bandwidth limited
- Using clang/llvm 11, CUDA 10.1, NVIDIA V100
- Kokkos/CUDA (kk-c), Kokkos/OMPT (kk-o), Native OMPT (omp), Native OMPT with temporaries (omp-t)

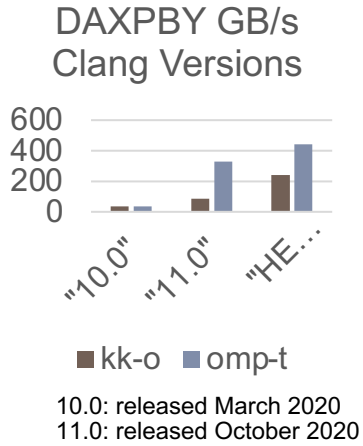


OpenMP Vector Add

```
struct Foo {
  int N;
  double *x, *y, *z;
  void axpby() {
    // Need temporaries here for 4x performance gain
    int N_ = N;
    double *xp = x, *yp = y, *zp = z;
    #pragma omp target teams distribute parallel for \
      simd is_device_ptr(xp,yp,zp) data map(to: N_)
    for(int i=0; i<N_; i++) {
      zp[i] = xp[i] + yp[i];
    }
  }
};
```

Kokkos Vector Add

```
struct Foo {
  View<double*> x,y,z;
  int N;
  void axpby() {
    parallel_for("axpby", N,
      KOKKOS_LAMBDA(int i) {
      z(i) = x(i) + y(i);
    });
  }
};
```



10.0: released March 2020
11.0: released October 2020

Takeaway: Performance is still very fragile!

A more comprehensive Frontend/Compiler comparison

- Comparing simple vector add and dot product
  - Also implemented straight forward native implementation
  - No hoops jumped through to optimize
  - 1M length, not huge, but also not trivial, i.e. latency impact expected but not dominant?
    - If purely bandwidth bound this would be 24us for axpby@1TB/s and 16us for dot
  - clxx denotes clang/llvm version