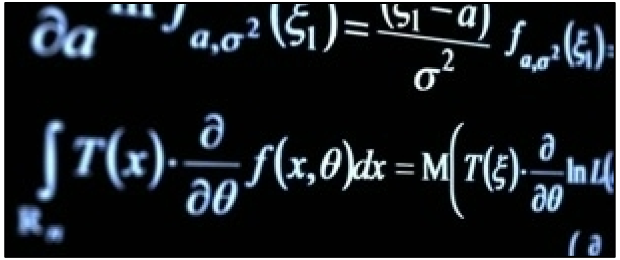
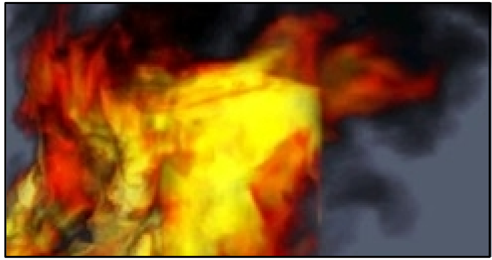
SAND2021-11197PE

Exceptional service in the national interest

Sandia National Laboratories





Kokkos Ecosystem: runtime, math library, tools

Unclassified Unlimited Release

Luc Berger-Vergiat*,* - 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.

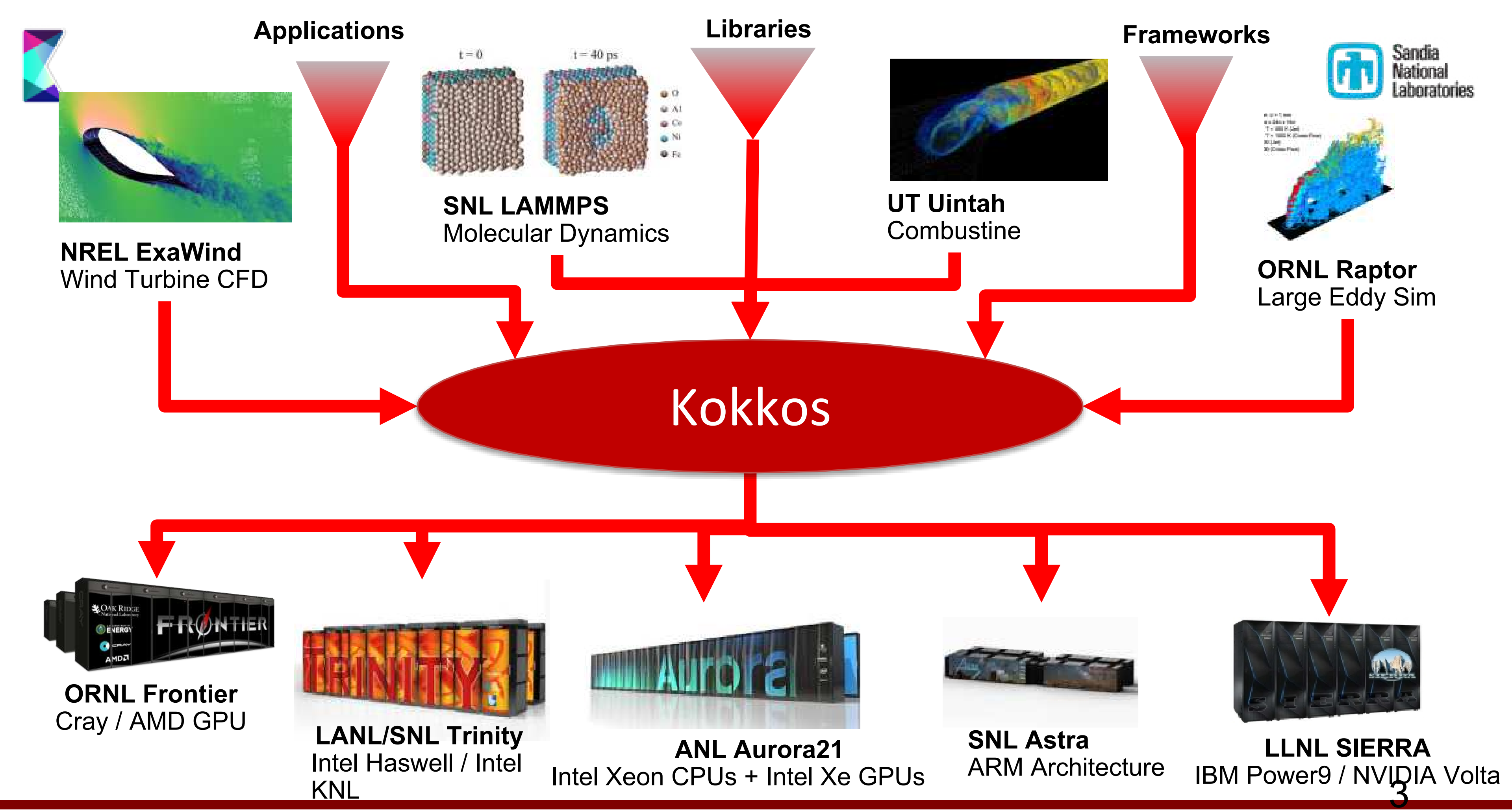
SAND2020-9180 C

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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-NA0003525.

Cost of Porting Code

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

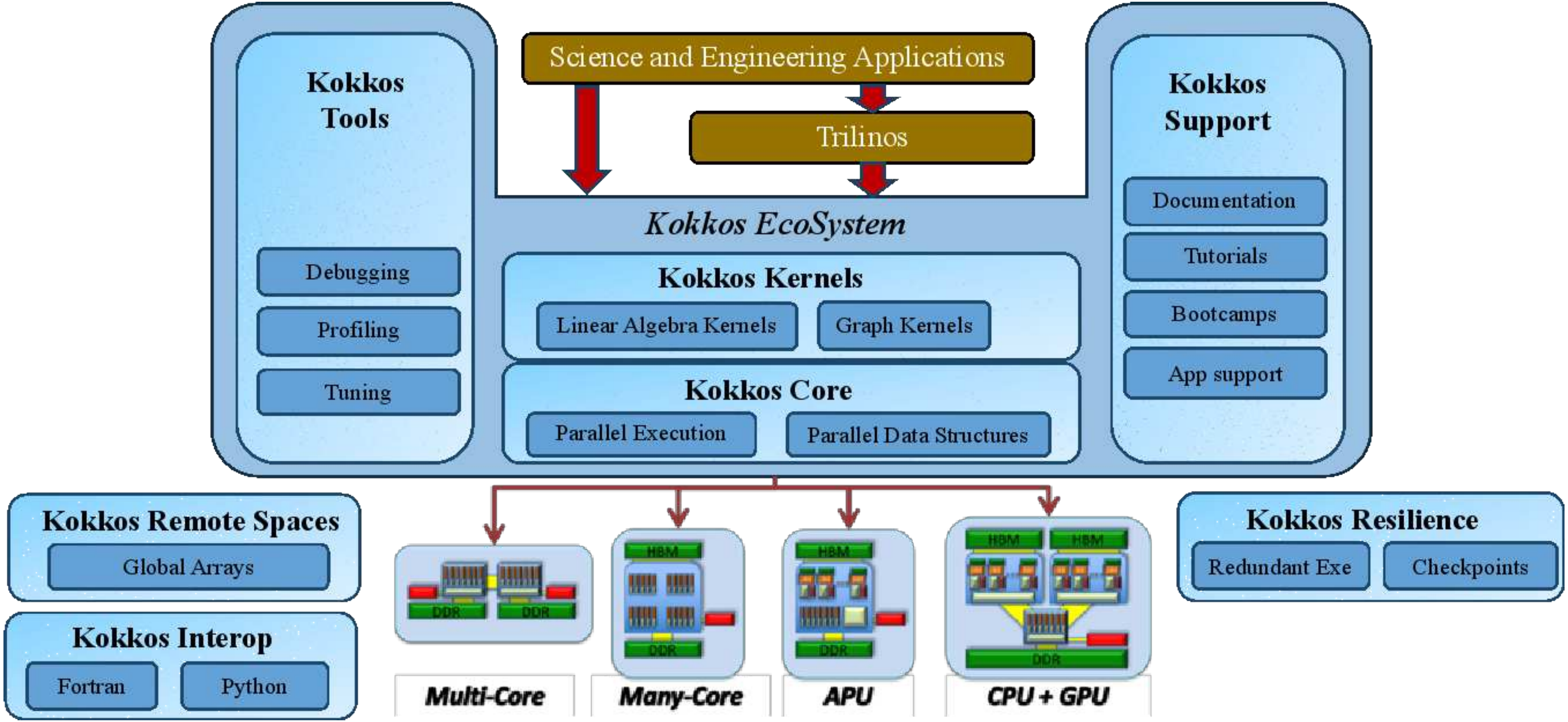


Applications
Libraries
Frameworks
Sandia National Laboratories
NREL ExaWind
Wind Turbine CFD
SNL LAMMPS
Molecular Dynamics
UT Uintah
Combustine
ORNL Raptor
Large Eddy Sim
Kokkos
ORNL Frontier
Cray / AMD GPU
LANL/SNL Trinity
Intel Haswell / Intel KNL
ANL Aurora21
Intel Xeon CPUs + Intel Xe GPUs
SNL Astra
ARM Architecture
LLNL SIERRA
IBM Power9 / NVIDIA Volta

What is Kokkos?

- 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

Kokkos Core: ***C.R. Trott,*** *N. Ellingwood, D. Ibanez, J. Miles, D. Hollman, V. Dang, Jan Ciesko, J. Wilke, L. Cannada, H. Finkel, N. Liber, D. Lebrun-Grandie, B. Turcksin, J. Madsen, D. Arndt, J. Madsen, R. Gayatri*
former: ***H.C. Edwards****, D. Labreche, G. Mackey, S. Bova, D. Sunderland,*

Kokkos Kernels: ***S. Rajamanickam,*** *L. Berger, V. Dang, N. Ellingwood, E. Harvey, B. Kelley, K. Kim, C.R. Trott, J. Wilke, S. Acer*

Kokkos Tools: ***D. Poliakoff,*** *S. Hammond, C.R. Trott, D. Ibanez, S. Moore, L. Cannada*

Kokkos Support: ***C.R. Trott,*** *G. Shipman, G. Lopez, G. Womeldorff,*
former: ***H.C. Edwards****, D. Labreche, Fernanda Foertter*

Kokkos Uptake

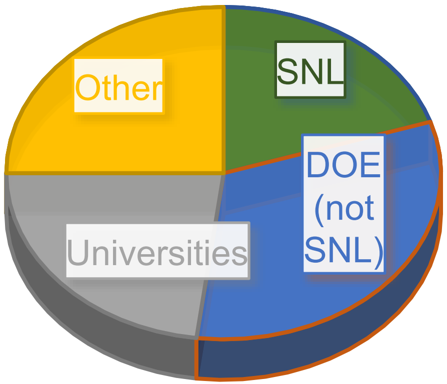
ECP Critical Dependencies

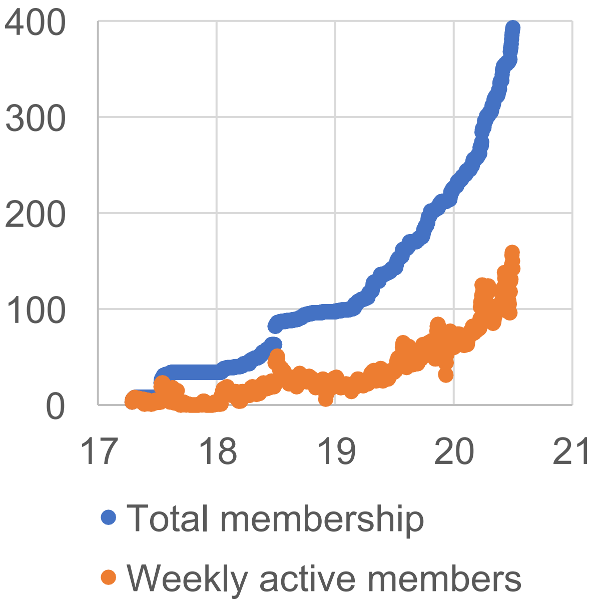
MPI	60
LLVM	57
C++	41
OpenMP	34
LAPACK CUDA	24 22
Fortran HDF5	21 21
BLAS	21
Kokkos	18
C	14
ALPINE	12

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

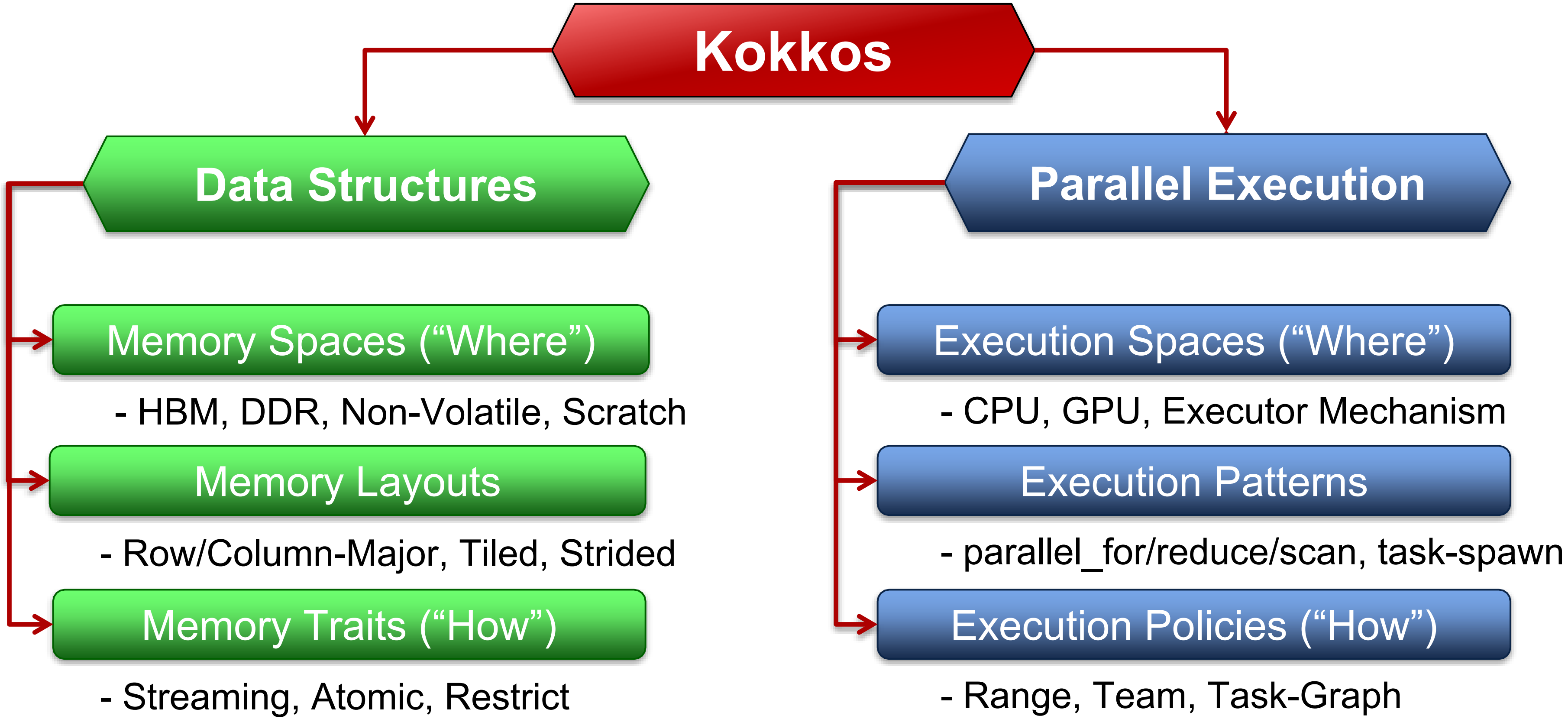
Kokkos Slack Users

- 525 registered users
  - 90 Institutions
  - Every continent
    - (-Antarctica)
- Doubles every year





Kokkos Core Abstractions



Kokkos Core Capabilities

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)

Sandia National Laboratories

More Kokkos Capabilities

- MemoryPool
- Reducers
- DualView
- parallel_scan
- ScatterView
- OffsetView
- LayoutRight
- StaticWorkGraph
- RandomPool
- UnorderedMap
- sort
- kokkos_malloc
- kokkos_free
- LayoutLeft
- Vector
- Bitset
- LayoutStrided
- UniqueToken
- ScratchSpace
- ProfilingHooks

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);
```

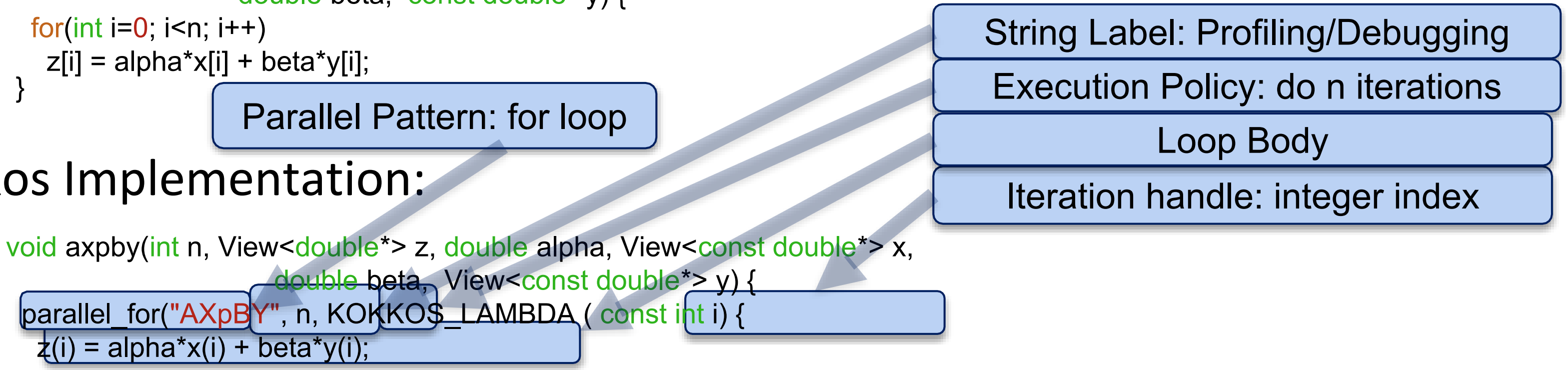
CG Solve: The AXPBY

- 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];
}
```

- 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;
}
```

- 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;
}
```

Parallel Pattern: loop with reduction

Iteration Index + Thread-Local Red. Variable

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

```
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

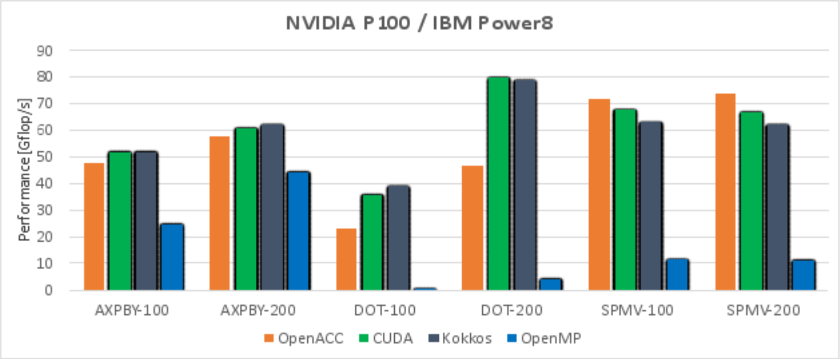
Row x Vector dot product

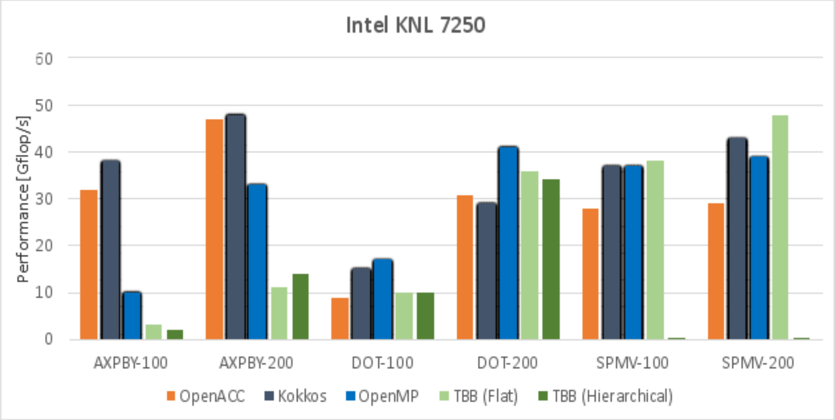
Distribute rows in workset over team-threads

Team Parallelism over Row Worksets

CG Solve: Performance

- Comparison with other Programming Models
- Straight forward implementation of kernels
- OpenMP 4.5 is immature at this point
- Two problem sizes: 100x100x100 and 200x200x200 elements

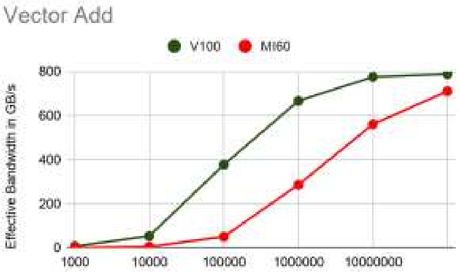


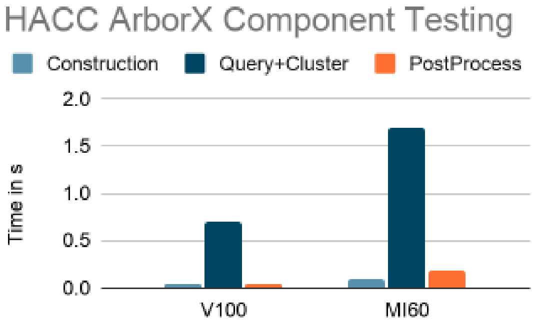


AMD Support Status

Frontier/El Capitan: HIP and OpenMP 5

- Primary development of HIP at ORNL
- Most Capabilities ready
- **PR testing for Kokkos on AMD GPUs in place**
- ArborX, Cabana, LAMMPS working with HIP
- Trilinos linear solvers are read
- Mesh and discretization next (support ExaWind/EMPIRE)





We are largely using our own machines, with the public software stack from Intel and AMD.

Kokkos 3.3 (Dec 2020):

- *HIP is largely feature complete*

Kokkos 3.4 (Feb/March 2021):

- *OpenMP Target largely feature 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):
- *OpenMP Target and DPC++ have most primary capabilities working*

Kokkos 3.4 (Feb/March 2021):
- *OpenMPTarget and DPC++/SYCL are largely feature complete*

Initial Kokkos Core functionality porting to Aurora done.

Kokkos Support

- 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/beneficial
- 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)
});
```

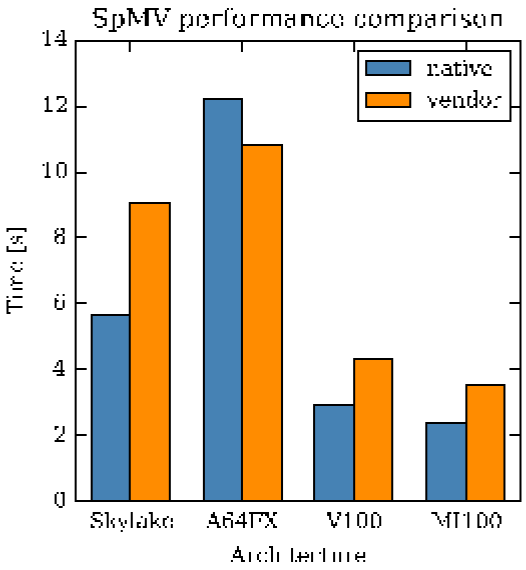
Example: CG Kokkos Kernels version

- Using Kokkos Kernels sparse and dense linear algebra simplifies CG implementation greatly

```
double toletance = 0.0; int iteration = 0;
while (tolerance < norm_res && iteration < 100) {
  std::cout << "Running CG iteration " << iteration
            << ", current resnorm = " << norm_res << '\n';
  /* Ap = A * p   */  KokkosSparse::spmv("N", 1, crsMat, pAll, 0, Ap);
  Space().fence();

  /* pAp_dot = dot(Ap , p ) */ const double pAp_dot = KokkosBlas::dot( p , Ap ) ;
  double alpha = old_rdot / pAp_dot ;
  /* x +=  alpha * p ;  */  KokkosBlas::axpby(alpha, p, 1.0, x_vector);
  /* r += -alpha * Ap ; */  KokkosBlas::axpby(-alpha, Ap, 1.0, r);

  const double r_dot = KokkosBlas::dot( r , r );
  const double beta = r_dot / old_rdot ;
  /* p = r + beta * p ; */  KokkosBlas::axpby(1.0, r, beta, p);

  norm_res = sqrt( old_rdot = r_dot );
  std::cout << "\tnorm_res:" << norm_res << " old_rdot:" << old_rdot<<  std::endl;
  ++iteration ;
}
```

Kokkos Kernels SpMV performance

- SpMV native implementation is specialized for:
  - Serial runs
  - OpenMP runs
  - GPU runs
  - Single vs. Multiple vectors
- Allows users to select following vendor TPLs
  - cuSPARSE
  - MKL
  - ArmPL in test only
  - rocSparse (PR in progress)



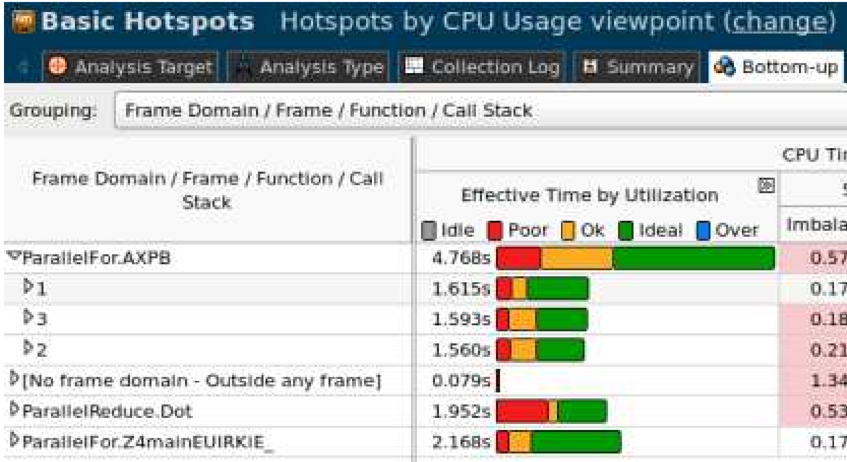
- Figure to the right shows native vs TPL implementation performance on various CPU and GPU architectures

Kokkos Tools

- 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

- 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:

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

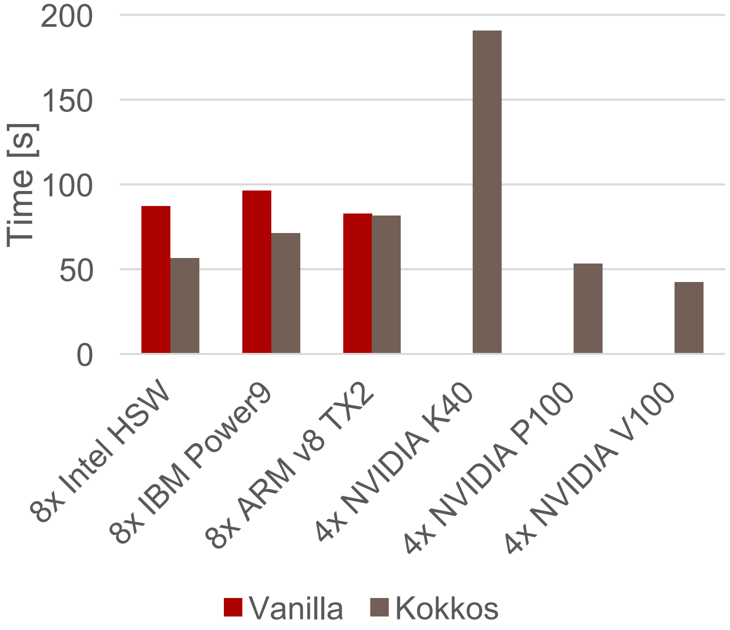
```
// Base case
Kokkos::parallel_for(
    TPolicy, KOKKOS_LAMBDA(TeamMember const& t) {
      int a = 0;

      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

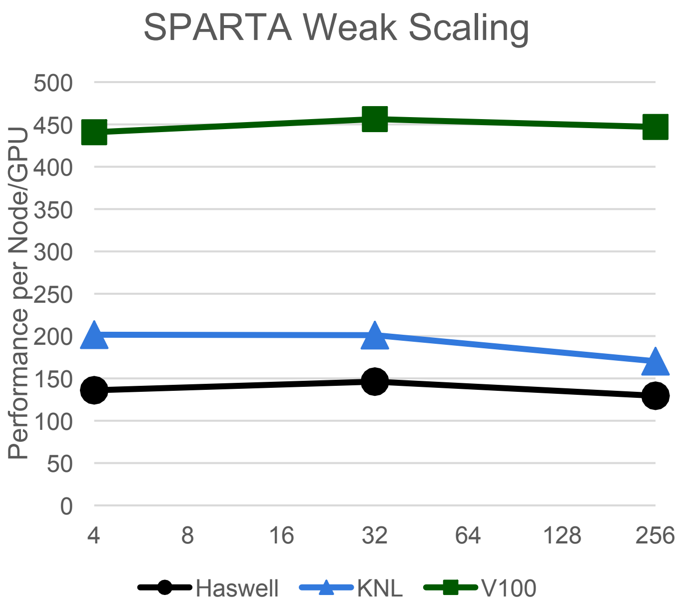
- Widely used Molecular Dynamics Simulations package
- Focused on Material Physics
- Over 500 physics modules
- Kokkos covers growing subset of those
- REAX is an important but very complex potential
  - USER-REAXC (Vanilla) more than 10,000 LOC
  - Kokkos version ~6,000 LOC
  - LJ in comparison: 200LOC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

Architecture Comparison Example in.reaxc.tatb / 196k atoms / 100 steps



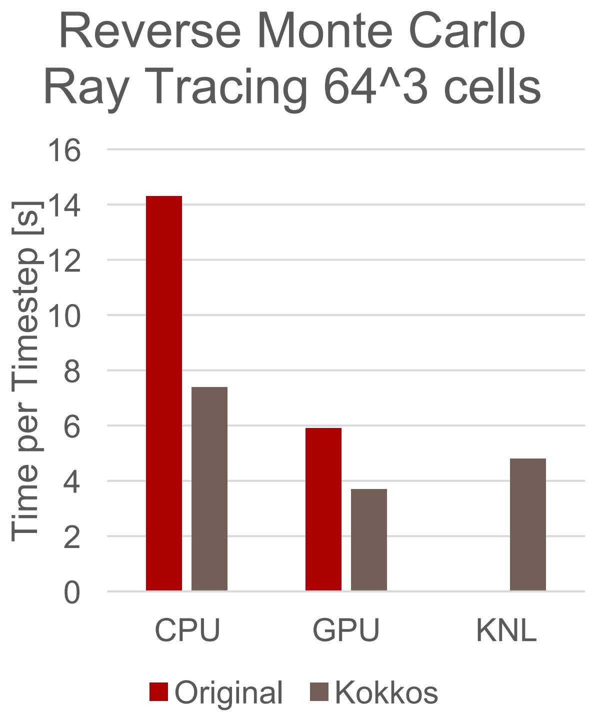
Sparta: Production Simulation at Scale

- **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



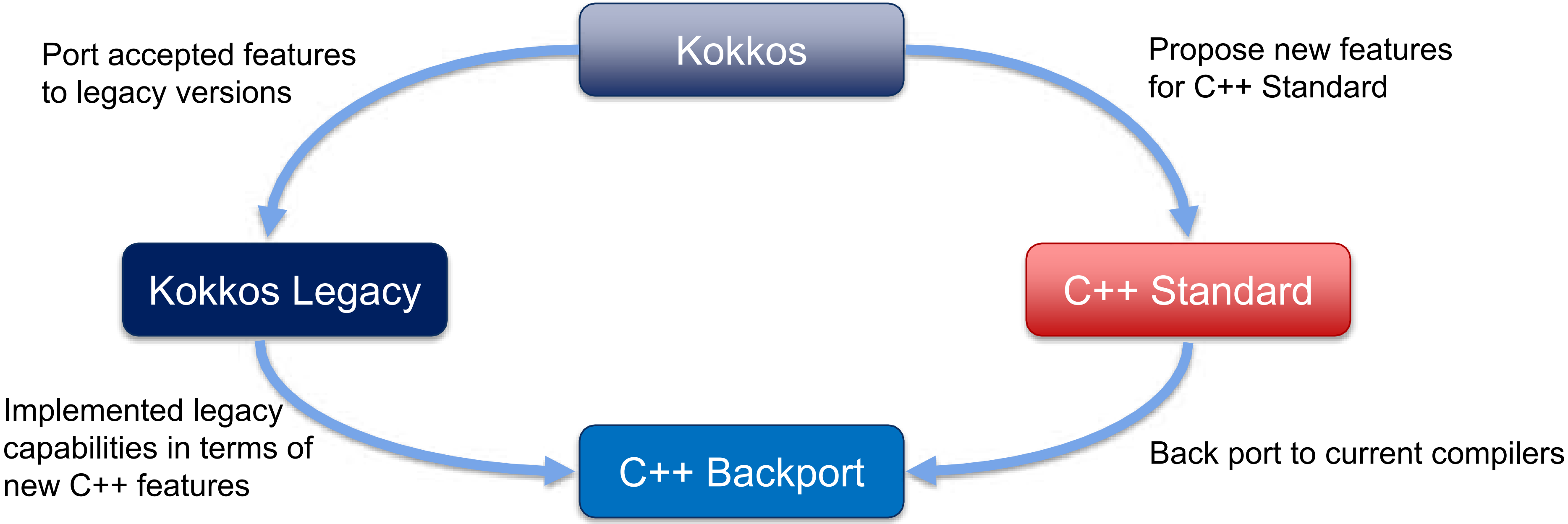
Uintah

- 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

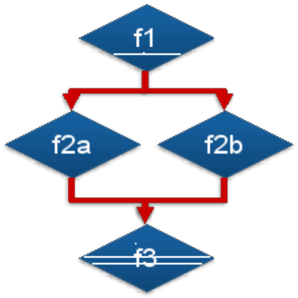
- First success: **atomic_ref**<T> in C++20
  - Provides atomics with all capabilities of atomics in Kokkos
  - **atomic_ref**(a[i])+=5.0; instead of **atomic_add**(&a[i],5.0);
- Next thing: **Kokkos::View** => **std::mdspan**
  - Provides customization points which allow all things we can do with **Kokkos::View**
  - 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

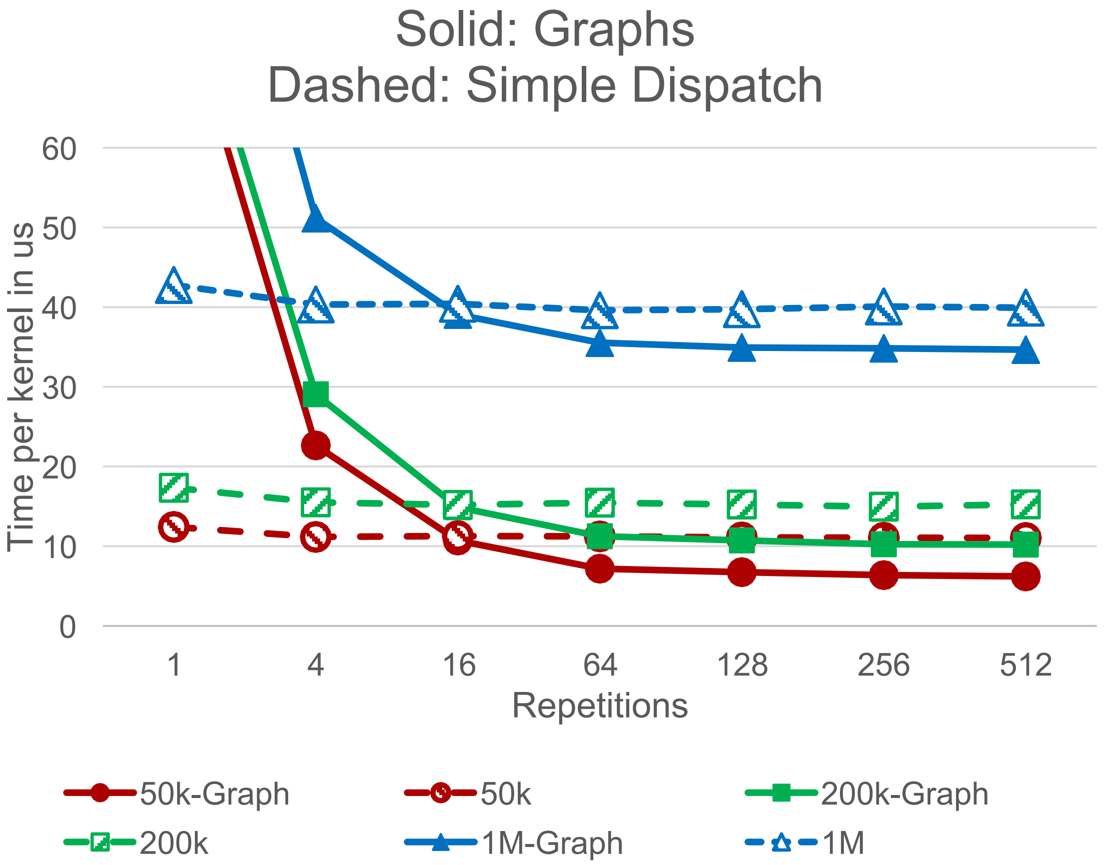
Tracking New Capabilities: Graphs

- 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%

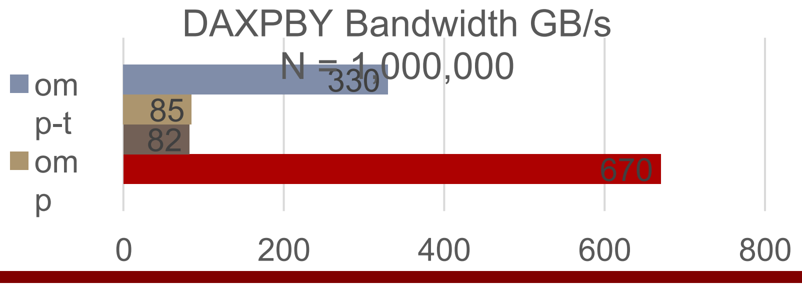
Next: look at reducing graph creation time

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)

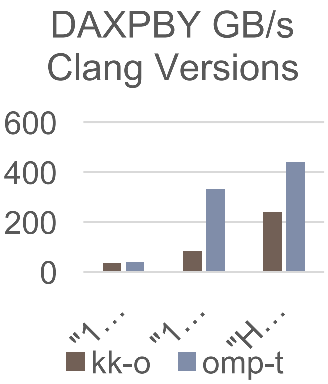


OpenMP Vector Add

```
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);
    });
  }
};
```

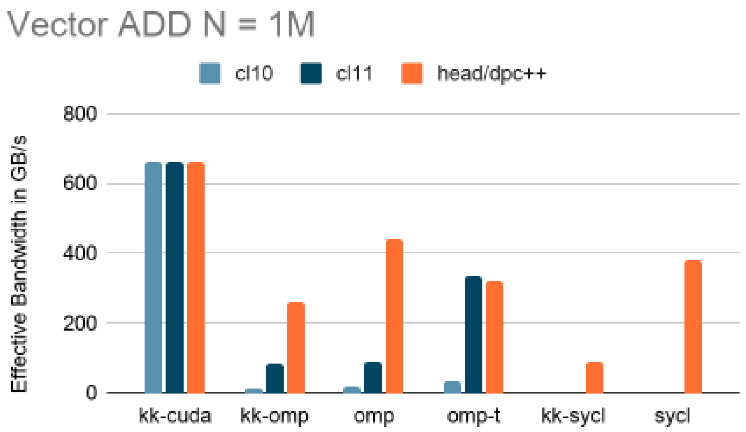


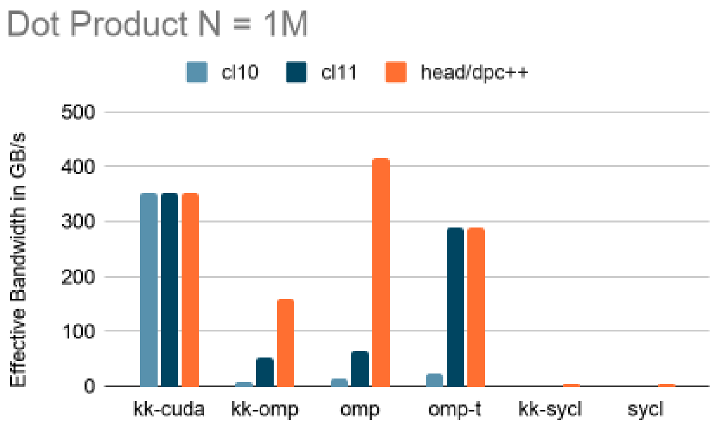
10.0: released March 2020
11.0: released October 2020

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





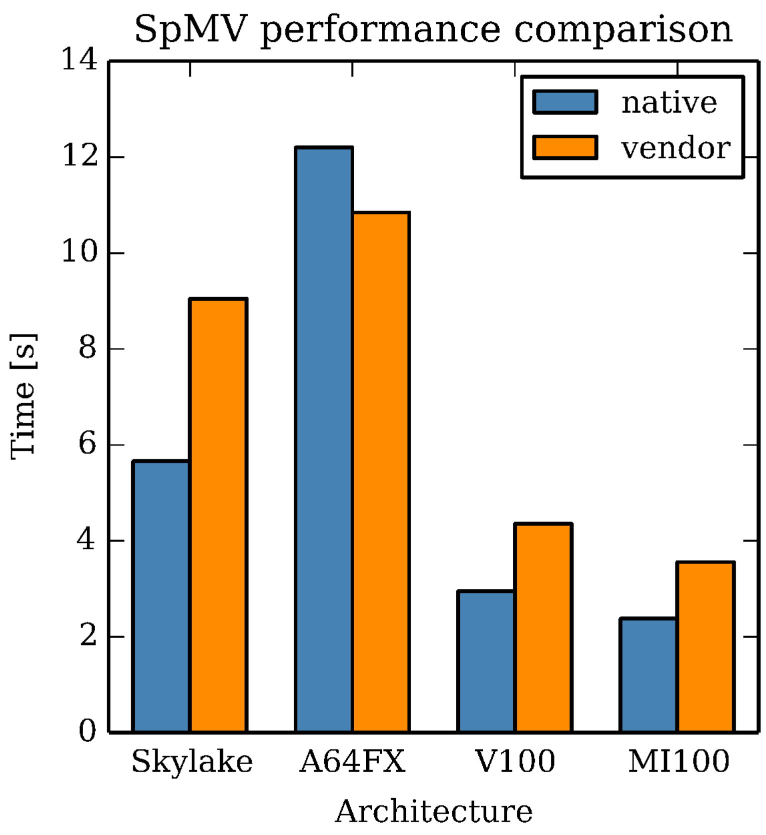
Sake: Kokkos Kernels

- What parts of your code are you porting to the accelerator, and what fraction of overall performance does this code account for in a realistic problem?
  - All of Kokkos Kernels is ported to accelerators, linear solvers can account for a large portion (up to 50%) of the overall performance of an application
- What accelerator programming environments were used? What is the long-term performance portability plan for exascale machines with different types of GPUs?
  - Kokkos Kernels relies on the Kokkos library to provide basic data structure and parallel execution policies. Currently all GPU architectures are supported through backends of Kokkos (Cuda, HIP, SYCL and OpenMP Target) and our algorithms are further tuned internally for performance.

    Currently the library is built and tested daily on Power9+V100 with the Kokkos Cuda backend and on Rome+MI100 with the Kokkos HIP backend. Work is ongoing to support daily testing of the OpenMP Target backend on both previously mentioned systems. Development is ongoing on JLSE systems with Kokkos SYCL backend.

Sake: Kokkos Kernels

- What single node speedup (if any) was achieved relative to the best performance on other classes of systems?
  - Shown on the right is the performance of our native SpMV implementation against vendor TPLs (MKL, ArmPL, cuSPARSE and rocSPARSE).
  - Our implementations strive to extract best performance on each architecture but also allow direct calls to vendor TPLs when possible or needed providing users with good baseline performance for most common linear algebra kernels.
  - Note that the results in figure to the right are subject to change depending on the matrix used for comparison, here two matrices representative of finite element/difference discretization were used



Sake: Kokkos Kernels

- What are the key bottlenecks, if any, to improving on-node performance, including plans for how to address them? For example, will there be a need to explore risky, fundamentally new algorithmic approaches, different mathematical formulations, or more fine tune for specific hardware features?
  - For Nvidia GPUs the performance is currently well established and no issues are foreseen
  - For AMD GPUs further tuning of the native algorithms is needed to accommodate specificities of the architecture such as wavefront size. Additionally rocBLAS and rocSPARSE needs to be expanded.
  - For Intel GPUs more issues exist, the OpenMP Target and SYCL backends are still under development with new bugs being reported to Kokkos. Some kernels are being refactored to favor reduction on single value instead of reducing on array of values.
  - New batched algorithms for dense and sparse linear algebra are being developed for specific applications need.

Sake: Kokkos Kernels – recent work

- New algorithms
  - MIS-2 kernels optimized and fully integrated with Trilinos/MueLu (multigrid package)
  - Batched sparse linear algebra and solvers (SpMV, CG and GMRES)
  - BsrMatrix and SpMV, will impactful for ATDM applications
- Library design
  - New stream interface: supports GEMV and GEMM on CUDA and HIP
  - Documentation publication automated at release time
  - clang-format checked during CI testing
  - Support for half precision
- Improvements
  - Optimized batched GEMM interface and performance (+3% DRAM utilization)
  - Add support for rocBLAS/rocSPARSE

Sake: Kokkos Kernels – upcoming work

- Algorithms
  - further support for BsrMatrix format: SpGEMM, Jacobi and Gauss-Seidel
  - batched Sparse Solvers: preconditioners, performance optimization and integration with applications
  - Device callable ODE solvers, potentially batched implementation too
  - Improve SpTRSV and SpILUK performance on device
  - Improve SpGEMM performance
- Library design
  - expand Stream interface to more kernels
  - Re-organize library with multiple build targets (allows subset of feature to be compiled)
  - Provide iterative solver interface (call from host to GPU)
  - Improve SYCL and OpenMP Target support and performance