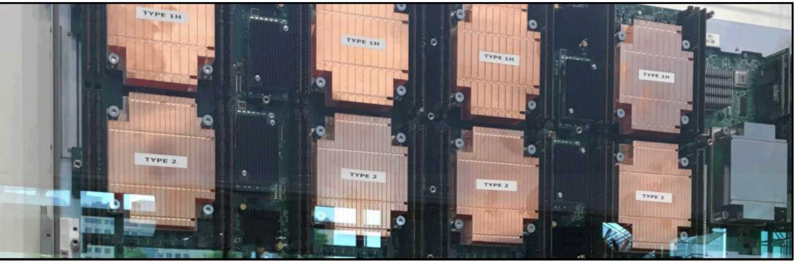
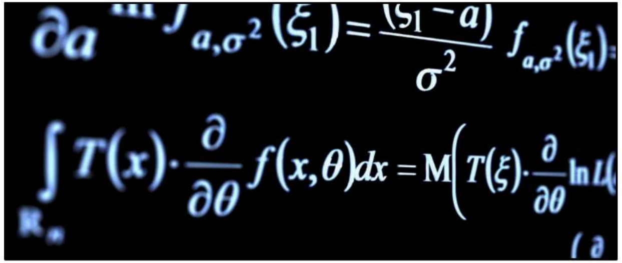
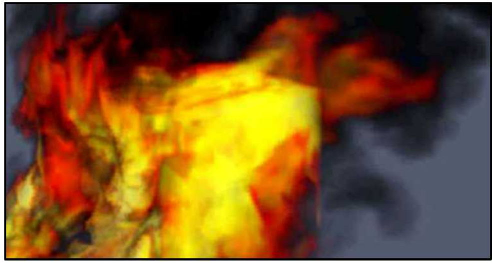
SAND2019-11367PE







The Kokkos C++ Performance Portability EcoSystem

Unclassified Unlimited Release

Christian R. Trott, - Center for Computing Research

Sandia National Laboratories/NM

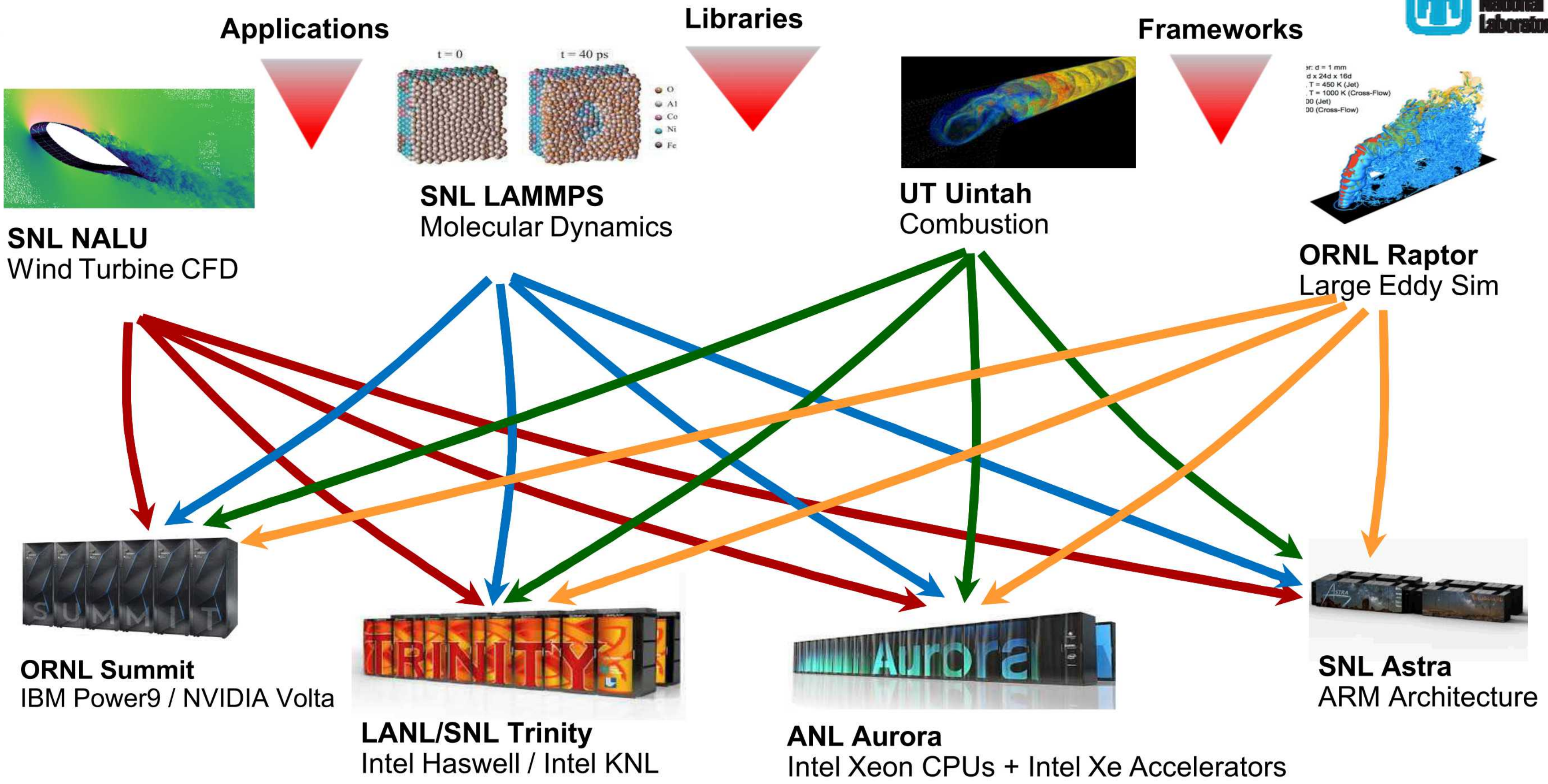
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

SAND2019-3111 C

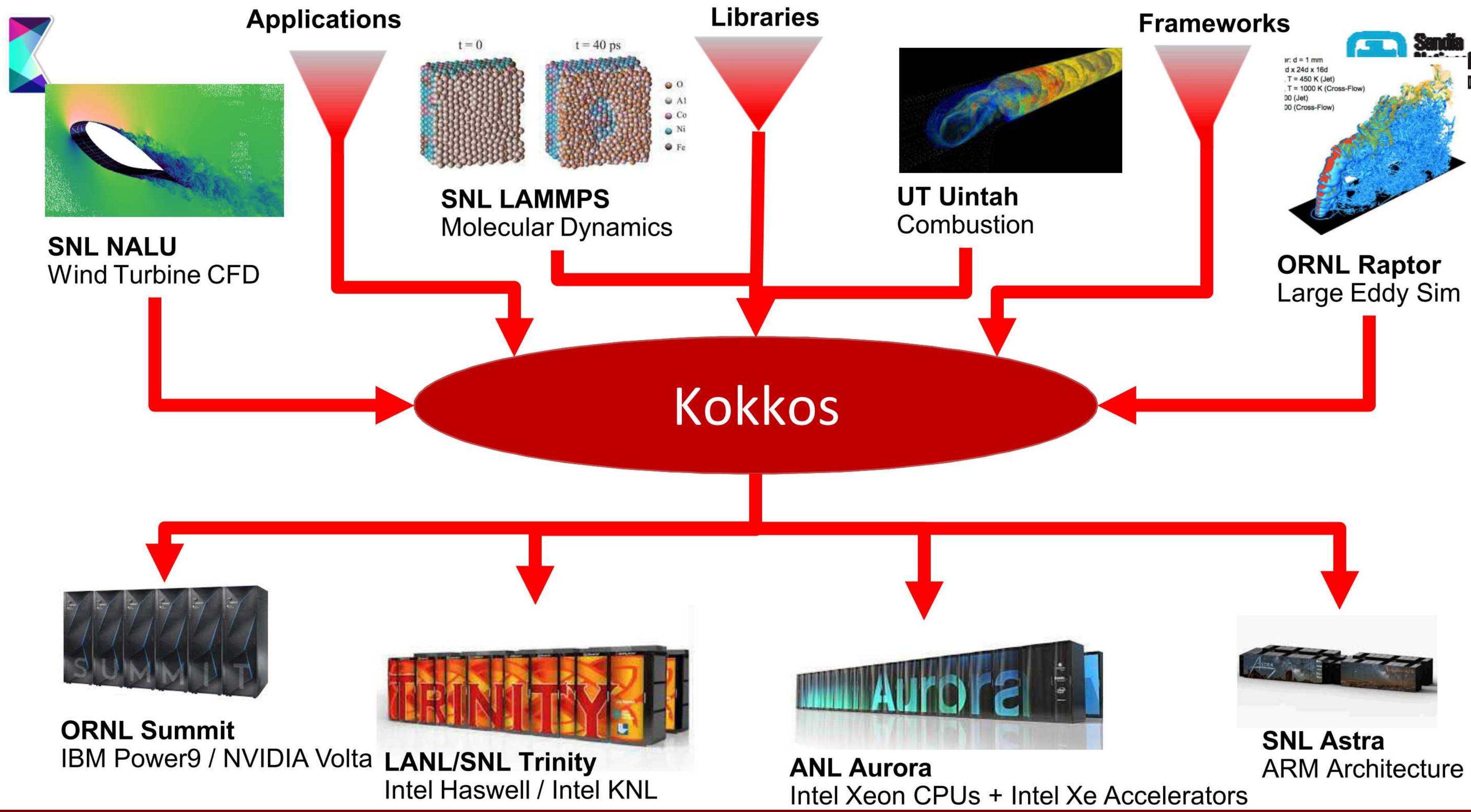
Cost Of Software

10 LOC / hour ~ 20k LOC / year

- Optimistic estimate: 10% of an application needs to get rewritten for adoption of Shared Memory Parallel Programming Model
- Typical Apps: 300k – 600k Lines
  - Uintah: 500k, QMCPack: 400k, LAMMPS: 600k; QuantumEspresso: 400k
  - Typical App Port thus 2-3 Man-Years
  - Sandia maintains a couple dozen of those
- Large Scientific Libraries
  - E3SM: 1,000k Lines x 10% => 5 Man-Years
  - Trilinos: 4,000k Lines x 10% => 20 Man-Years



Applications
Libraries
Frameworks
t = 0
t = 40 ps
O
Al
Co
Ni
Fe
SNL NALU
Wind Turbine CFD
SNL LAMMPS
Molecular Dynamics
UT Uintah
Combustion
ORNL Raptor
Large Eddy Sim
ORNL Summit
IBM Power9 / NVIDIA Volta
LANL/SNL Trinity
Intel Haswell / Intel KNL
ANL Aurora
Intel Xeon CPUs + Intel Xe Accelerators
SNL Astra
ARM Architecture


Applications
Libraries
Frameworks
t = 0
t = 40 ps
O
Al
Co
Ni
Fe
SNL NALU
Wind Turbine CFD
SNL LAMMPS
Molecular Dynamics
UT Uintah
Combustion
ORNL Raptor
Large Eddy Sim
Kokkos
ORNL Summit
IBM Power9 / NVIDIA Volta
LANL/SNL Trinity
Intel Haswell / Intel KNL
ANL Aurora
Intel Xeon CPUs + Intel Xe Accelerators
SNL Astra
ARM Architecture

Outline

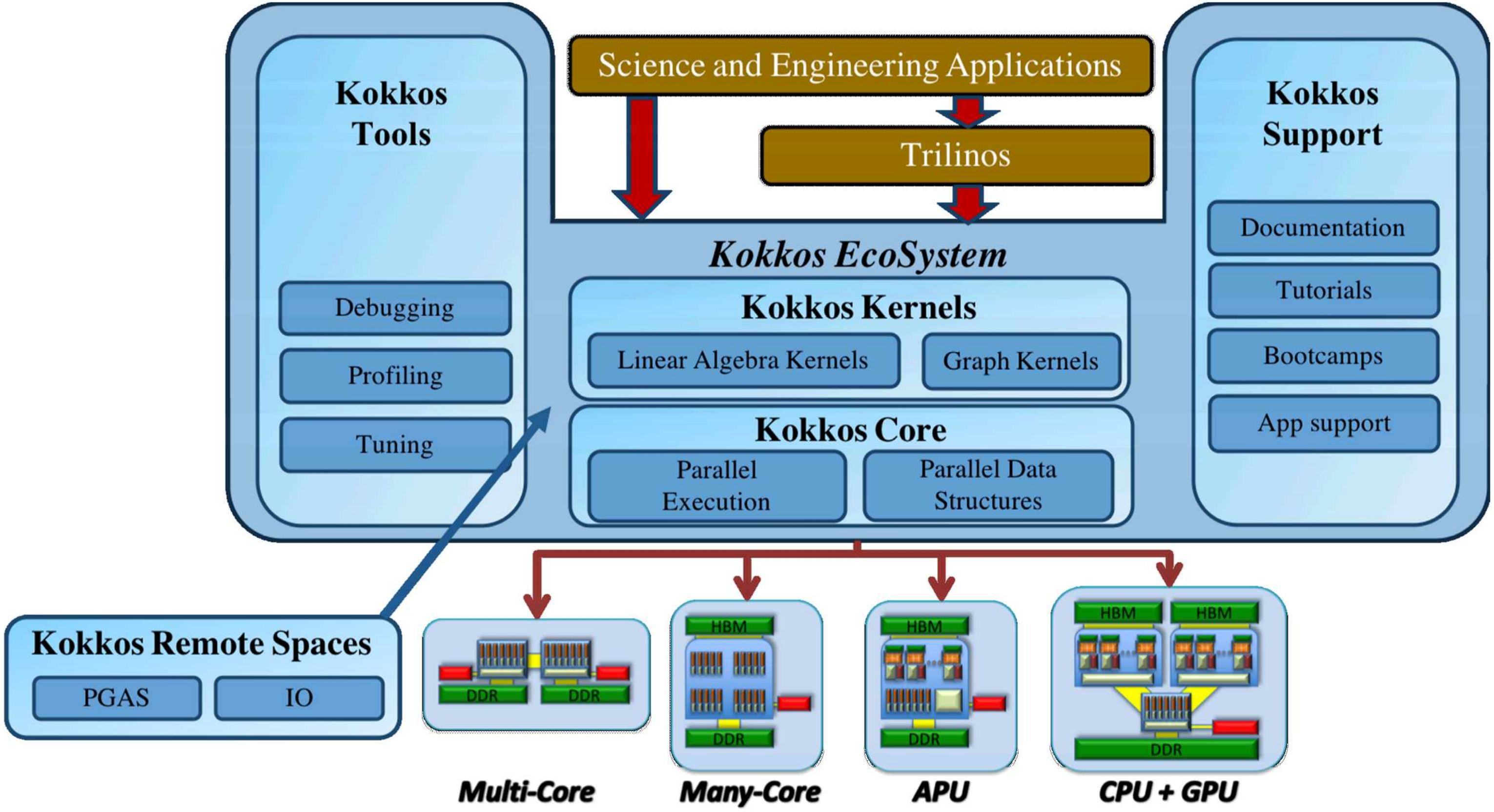
- The Kokkos EcoSystem
  - Abstractions and Capabilities
  - CG-Solve as an Example
  - Kokkos Kernels & Tools
- Kokkos Applications
- Quo vadis?
  - C++ Standard Interactions
  - Properties for a more descriptive programming model
  - Enhanced asynchronous execution

What is Kokkos?

- A C++ Programming Model for Performance Portability
  - Implemented as a template library on top of CUDA, OpenMP, ROCm, …
  - 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 allow inside into Kokkos
- It is Open Source
  - Maintained and developed at https://github.com/kokkos
- It has many users at wide range of institutions.

NOT an Intel Product!

Kokkos EcoSystem



Kokkos Development Team

Kokkos Core:	***C.R. Trott,*** *D. Sunderland, N. Ellingwood, D. Ibanez, J. Miles, D. Hollman, V. Dang,*
	H. Finkel, N. Liber, D. Lebrun-Grandie, B. Turcksin, J. Wilke, D. Arndt
	former: ***H.C. Edwards****, D. Labreche, G. Mackey, S. Bova*
Kokkos Kernels:	***S. Rajamanickam,*** *N. Ellingwood, K. Kim, C.R. Trott, V. Dang, L. Berger, J. Wilke, W. McLendon*
Kokkos Tools:	***D. Poliakoff,*** *S. Hammond, C.R. Trott, D. Ibanez, S. Moore*
Kokkos Support:	***C.R. Trott,*** *G. Shipman, G. Lopez, G. Womeldorff, and all of the above as needed*
	former: ***H.C. Edwards****, D. Labreche, Fernanda Foertter*

Kokkos Core Abstractions

Kokkos

Data Structures

- **Memory Spaces (“Where”)**
  - HBM, DDR, Non-Volatile, Scratch
- **Memory Layouts**
  - Row/Column-Major, Tiled, Strided
- **Memory Traits (“How”)**
  - Streaming, Atomic, Restrict

Parallel Execution

- **Execution Spaces (“Where”)**
  - CPU, GPU, Executor Mechanism
- **Execution Patterns**
  - parallel_for/reduce/scan, task-spawn
- **Execution Policies (“How”)**
  - Range, Team, Task-Graph

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), ROCm (experimental)

More Kokkos Capabilities

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

Loop Body

```
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:

Iteration Index + Thread-Local Red. Varible

```
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

```
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;
  }
}
```

Inner dot product row x vector

CG Solve: Sparse Matrix Vector Multiply

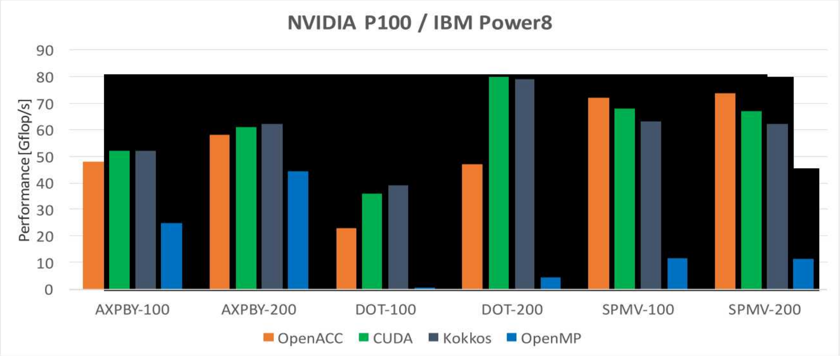
```
void SPMV(int nrows, View<const int*> A_row_offsets,
          View<const int*> A_cols, View<const double*> A_vals,
          View<double*> y,
          View<const double*, MemoryTraits< RandomAccess>> x) {

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

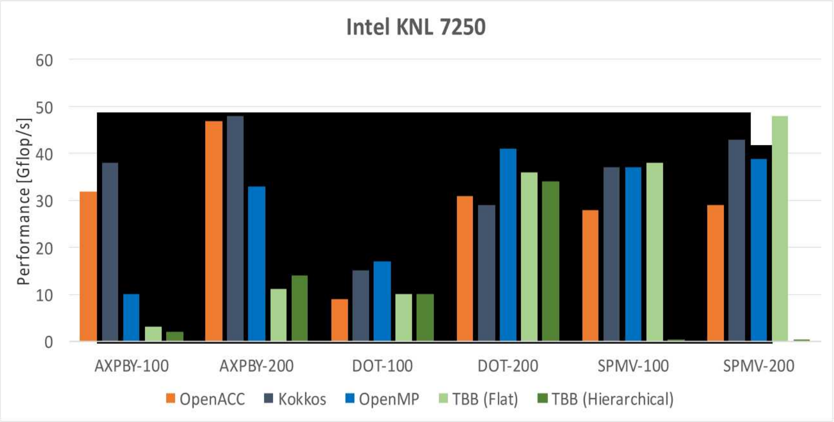
Enable Texture Fetch on x

Row x Vector dot product

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





Tasking Example Code

```
template< typename Scheduler >
struct FibonacciTask {
  using sched_type = Scheduler;
  using future_type = BasicFuture< long, Scheduler >;
  future_type fib_m1, fib_m2;
  const long n;

  KOKKOS_INLINE_FUNCTION
  TestFib( const value_type arg_n )
    : fib_m1(), fib_m2(), n( arg_n ) {}

  KOKKOS_INLINE_FUNCTION
  void operator()( typename sched_type::member_type & member, value_type & result ) {
    auto& sched = member.scheduler();
    if ( n < 2 ) { result = n; }
    else if ( !fib_m2.is_null() && !fib_m1.is_null() ) { result = fib_m1.get() + fib_m2.get(); }
    else {
      fib_m2 = task_spawn( TaskSingle( sched, TaskPriority::High ), FibonacciTask( n - 2 ) );
      fib_m1 = task_spawn( TaskSingle( sched ), FibonacciTask( n - 1 ) );

      BasicFuture<void, Scheduler> dep[] = { fib_m1, fib_m2 };
      BasicFuture<void, Scheduler> fib_all = sched.when_all( dep, 2 );


      if ( !fib_m2.is_null() && !fib_m1.is_null() && !fib_all.is_null() ) {
        respawn( this, fib_all, TaskPriority::High );
      } else { Kokkos::abort( "TestFib insufficient memory" ); }
    }
  }
```

Scheduler obtained from arguments: task could be a lambda

Spawn child tasks

Make compound dependency

Respawn task with new deps

If dependencies are not NULL this is respawn

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
- View have all their 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);
```

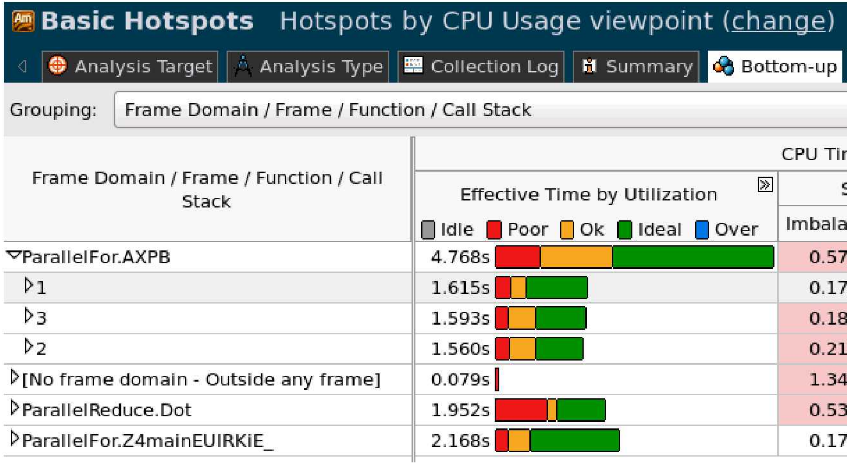
```
// Kokkos Kernels
double alpha, beta; View<double**> A,B,C;
gemm('N','N',alpha,A,B,beta,C);
```

- 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 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
- One 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



DOE Machine Announcements

- Now publicly announced that DOE is buying both AMD and Intel GPUs
  - Argonne: Cray with Intel Xeon + Intel Xe Compute
  - ORNL: Cray with AMD CPUs + AMD GPUs
  - NERSC: Cray with AMD CPUs + NVIDIA GPUs
- Have been planning for this eventuality:
  - Kokkos ECP project extended and refocused to include developers at Argonne, Oak Ridge, and Lawrence Berkeley - staffing is in place
  - HIP backend for AMD: main development at ORNL
    - The current ROCm backend is based on a compiler which is now deprecated …
  - SYCL for Intel: main development at ANL
  - OpenMPTarget for AMD, Intel and NVIDIA, lead at Sandia

Supporting Aurora

- Two backend plans
  - SYCL: will need Intel proposed extensions
    - ANL will lead development
  - OpenMPTarget: OpenMP 5.x based
    - NERSC/SNL will lead development
- Timeline:
  - Q2 FY20: Initial capabilities, enough for many miniApps
  - Q4 FY20: Functional backends
  - FY21: Production support

OpenMPTarget Backend

- Started work on this more than 2 years ago
  - Hindered by compiler bugs: 15 min work on backend, 6 hours work on compiler bug reproducer, 6 months wait for fix, repeat
  - With Clang 9 first time this isn't the case
- Got some capabilities:
  - RangePolicy: parallel_for, parallel_reduce
  - MDRangePolicy: parallel_for
  - Views

OpenMPTarget Issues

- Virtual Functions
  - Technically not supported: need to discuss with compiler vendors
- TeamPolicy:
  - Scratch memory: Not clear whether this is relevant for Aurora
  - Handling of SIMD? – In OpenMP right now inconsistent across vendors:
    - Does it do anything? Are vector lanes redundantly executing?

Kokkos

```
parallel_for("K",TeamPolicy<>(N,AUTO,8),
  [=] (team_t& team) {
    int i = team.league_rank();
    int t = team.team_rank();
    parallel_for(TeamThreadRange(team,M),
      [&] (int j) {
      parallel_for(ThreadVectorRange(team,K),
        [&] (int k) { … });
    });
});
```

OpenMP

```
#pragma omp teams distribute target
for(int i=0; i<N; i++) {
  #pragma omp parallel
  {
    int t = omp_thread_num();
    #pragma omp for
    for(int j=0; j<M; j++) {
      #pragma omp simd
      for(int k=0; k<K; k++) { ... }
}}}
```

OpenMPTarget Issues II

- Arbitrary reductions
  - Stateful reducers are cumbersome (need to replace the value type with some wrapper, which contains the stateful reducer)
- Can we get the equivalent of CudaSpace, CudaUVMSpace, and CudaHostPinnedSpace?
  - Not clear that we can currently do that in OpenMP, problem for Trilinos which currently relies on page migratable memory …
- Equivalent of Stream support, or at least asynchronous dispatch?
- Arbitrary Atomic Operations
  - Need to implement our own most likely

SYCL Backend

- Started recently both with Codeplays and Intels compiler
- Not much working yet
  - RangePolicy: parallel_for works with Codeplay
- Looking into some of the problems around restrictions of SYCL such as kernel naming
- We likely need to rely on Intel proposed extensions
  - A good chunk of which are already implemented!

SYCL Issues I

- Kernel naming and lambdas
  - Can not name a kernel implicitly templated on a lambda
  - Relying on Intel not requiring names
- Data management
  - SYCL auto data management is not in line with general Kokkos data management philosophy
  - Could be workable, would require somewhat nasty internal plumbing
  - Current plan: rely on Intel proposed extensions for raw allocations and deep_copy

SYCL Issues II

- Virtual functions not supported
- Reductions:
  - Need to implement our own: not sure how to do efficient ones need shuffle operations, and scratch memory
- Arbitrary atomics
  - Need to implement our own

RAJA will be there too!

- Expect RAJA to be working on A21
- OpenMP target based backend is largely functional on other platforms
  - Will work as part of ECP RAJA/Kokkos project to make sure that this works with Intel compiler
- SYCL backend is getting explored
  - Will need Intel extensions for good usability
  - Production support a question of needs by users, and whether OpenMP target backend will work well enough
- If you are using/want to use RAJA on A21 let us know
  - Work by Argonne members of RAJA/Kokkos will to a large degree be guided by user requests.

Supporting Aurora

- OpenMP Target Offload underway
  - Need to have some discussions on details of how to implement standard
- SYCL just begun
  - Problems in the current standard
  - BUT: Intel is addressing them
- We expect both backends will be viable
- Major concerns are already known to Intel
  - A good chunk are already addressed:
    - Standalone allocations
    - Kernel Naming

Kokkos Based Projects

- Production Code Running Real Analysis Today
  - We got about **12** or so.
- Production Code or Library committed to using Kokkos and actively porting
  - Somewhere around **35**
- Packages In Large Collections (e.g. Tpetra, MueLu in Trilinos) committed to using Kokkos and actively porting
  - Somewhere around **65**
- Counting also proxy-apps and projects which are evaluating Kokkos (e.g. projects who attended boot camps and trainings).
  - Estimate **100-150** packages.

Some Kokkos Users

Sandia National Laboratories

Pacific Northwest NATIONAL LABORATORY

Los Alamos NATIONAL LABORATORY EST. 1943

Sandia National Laboratories

ARL

MICHIGAN STATE

NREL NATIONAL RENEWABLE ENERGY LABORATORY

OAK RIDGE National Laboratory

Argonne NATIONAL LABORATORY

U.S. NAVAL RESEARCH LABORATORY

CSCS

THE UNIVERSITY OF UTAH

DE LA RECHERCHE À L'INDUSTRIE cea

JÜLICH Forschungszentrum

GT

TUM TECHNISCHE UNIVERSITÄT MÜNCHEN

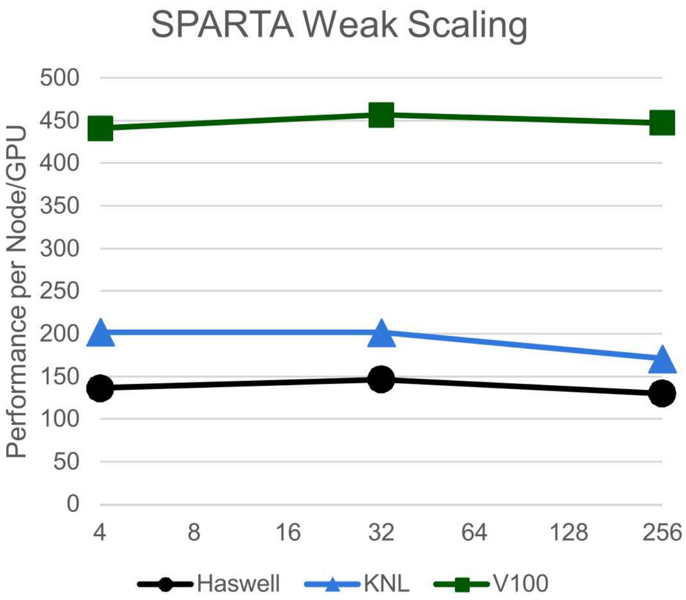
IPP Max-Planck-Institut für Plasmaphysik

BERKELEY LAB

Rensselaer

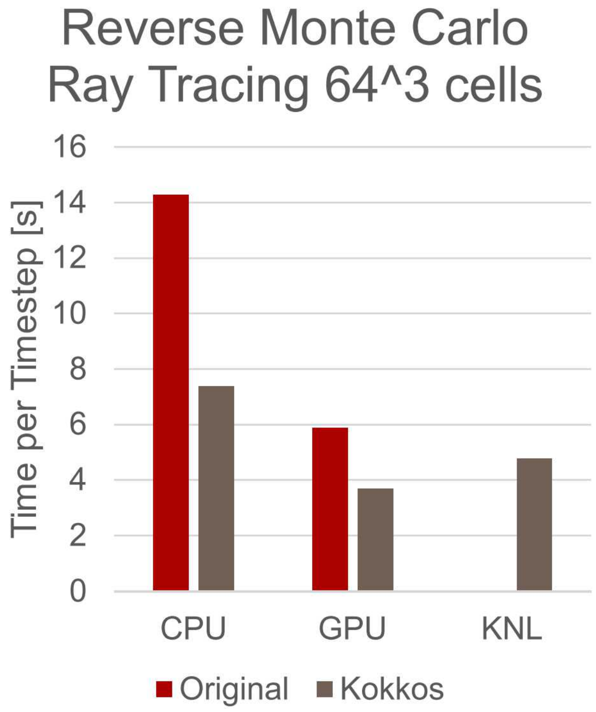
Sparta: Production Simulation at Scale

- **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

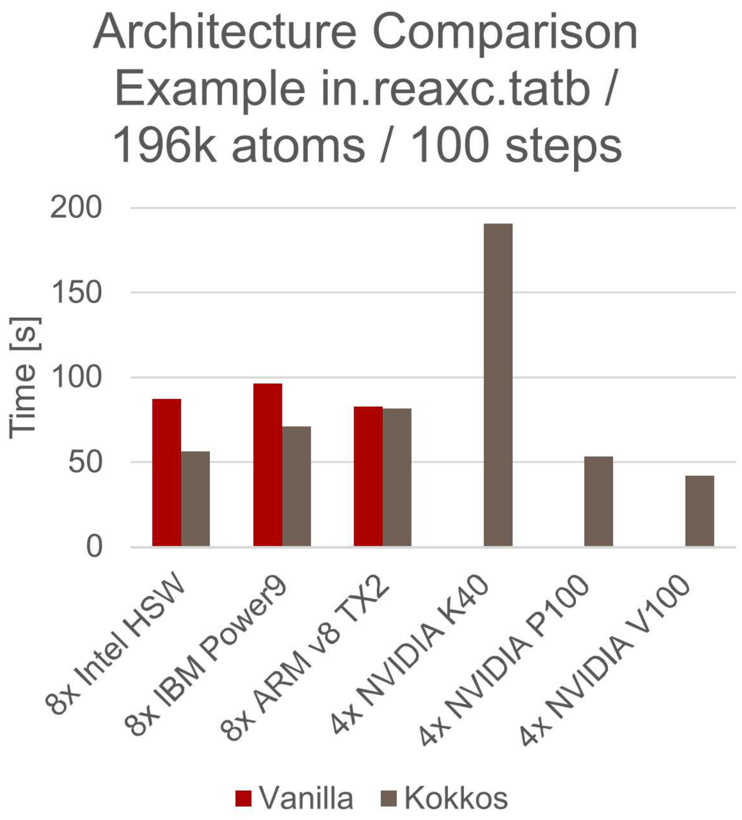


Questions: Dan Sunderlan

LAMMPS

Questions: Stan Moore

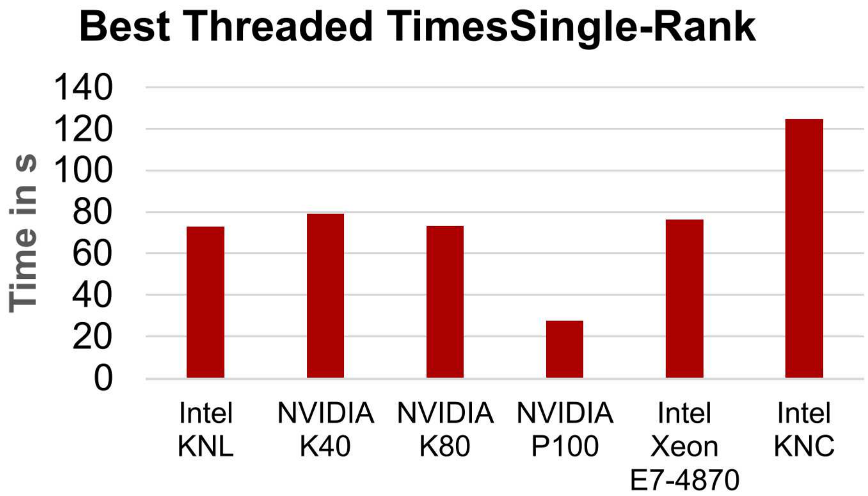
- Widely used Molecular Dynamics Simulations package
- Focused on Material Physics
- Over 500 physics modules
- Kokkos covers growing subset of those
- REAX is an important but very complex potential
  - USER-REAXC (Vanilla) more than 10,000 LOC
  - Kokkos version ~6,000 LOC
  - LJ in comparison: 200LOC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

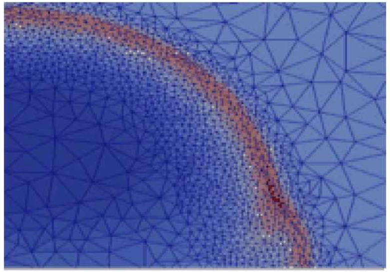


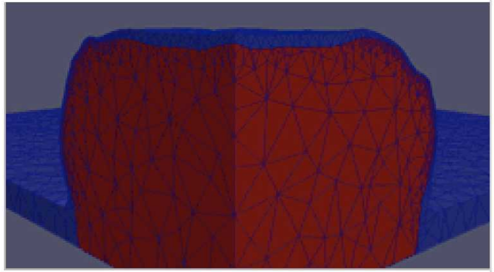
Alexa

- Portably performant shock hydrodynamics application
- Solving multi-material problems for internal Sandia users
- Uses tetrahedral mesh adaptation

Questions: Dan Ibanez



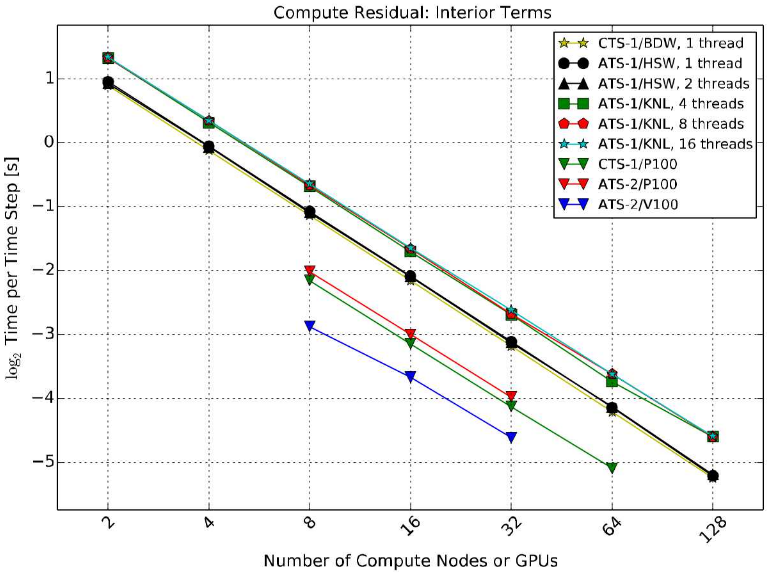




- All operations are Kokkos-parallel
- Test case: metal foil expanding due to resistive heating from electrical current.

SPARC Courtesy of: Micah Howard

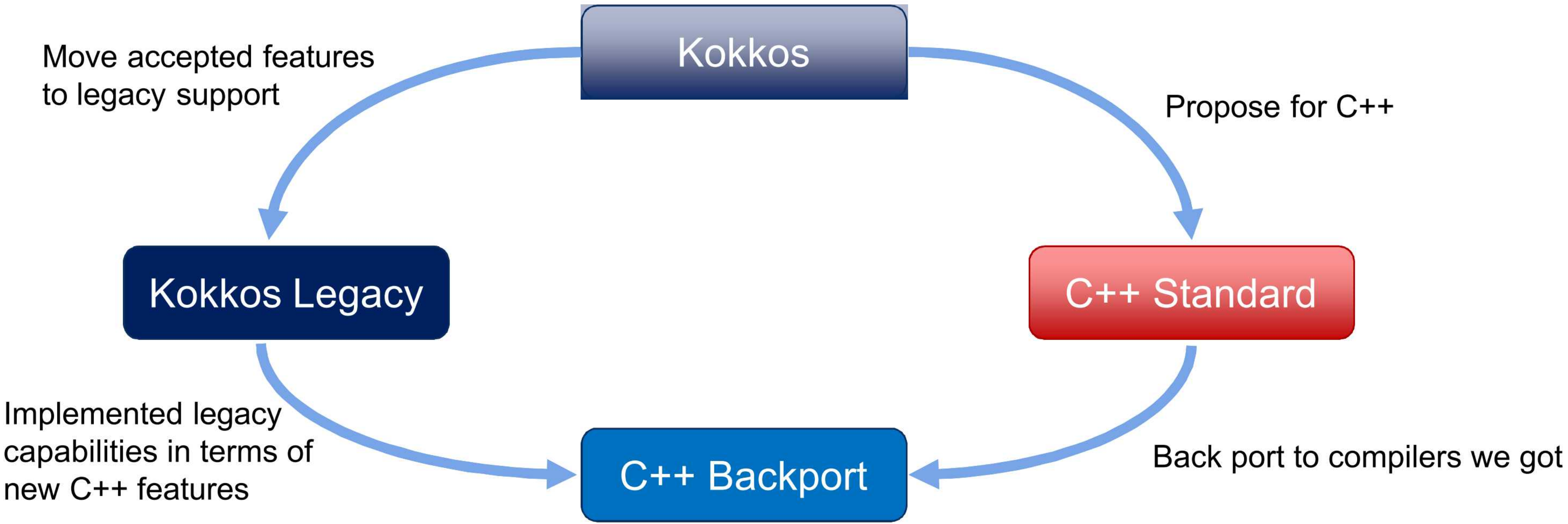
- Goal: solve aerodynamics problems for Sandia (transonic and hypersonic) on 'leadership' class supercomputers
- Solves compressible Navier-Stokes equations
- Perfect and reacting gas models
- Laminar and RANS turbulence models -> hybrid RANS-LES
- Primary discretization is cell-centered finite volume
- Research on high-order finite difference and discontinuous Galerkin discretizations
- Structured and unstructured grids



8 Sierra nodes (32x V100) equivalent to ~80 Trinity nodes (160x Haswell 16c CPU) for Residual Computation

Aligning Kokkos with the C++ Standard

- Long term goal: move capabilities from Kokkos into the ISO standard
  - Concentrate on facilities we really need to optimize with compiler



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** (just began design work)

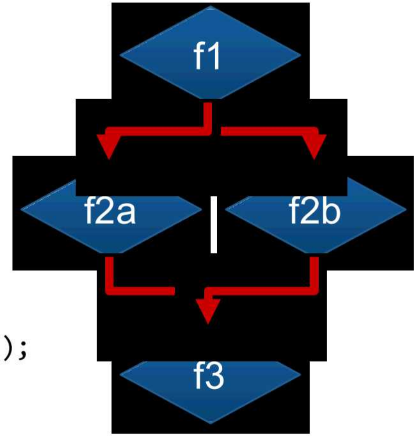
Towards C++23 Executors

- C++ standard is moving towards more asynchronicity with Executors
  - Dispatch of parallel work consumes and returns new kind of future
- Aligning Kokkos with this development means:
  - Introduction of Execution space instances (CUDA streams work already)

```
DefaultExecutionSpace spaces[2];
partition( DefaultExecutionSpace(), 2, spaces);
// f1 and f2 are executed simultaneously
parallel_for( RangePolicy<>(spaces[0], 0, N), f1);
parallel_for( RangePolicy<>(spaces[1], 0, N), f2);
// wait for all work to finish
fence();
```

  - Patterns return futures and Execution Policies consume them

```
auto fut_1 = parallel_for( RangePolicy<>("Funct1", 0, N), f1 );
auto fut_2a = parallel_for( RangePolicy<>("Funct2a", fut_1,0, N), f2a);
auto fut_2b = parallel_for( RangePolicy<>("Funct2b", fut_1,0, N), f2b);
auto fut_3 = parallel_for( RangePolicy<>("Funct3", all(fut_2a,fut2_b),0, N), f3);
fence(fut_3);
```



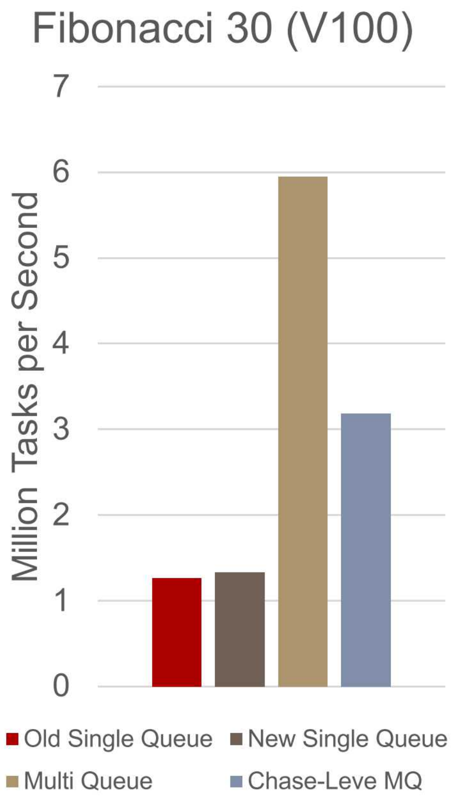
Links

- https://github.com/kokkos Kokkos Github Organization
  - **Kokkos:** *Core library, Containers, Algorithms*
  - **Kokkos-Kernels:** *Sparse and Dense BLAS, Graph, Tensor (under development)*
  - **Kokkos-Tools:** *Profiling and Debugging*
  - **Kokkos-MiniApps:** *MiniApp repository and links*
  - **Kokkos-Tutorials:** *Extensive Tutorials with Hands-On Exercises*
- https://cs.sandia.gov Publications (search for ’Kokkos’)
  - Many Presentations on Kokkos and its use in libraries and apps
- http://on-demand-gtc.gputechconf.com Recorded Talks
  - Presentations with Audio and some with Video

Improved Fine Grained Tasking

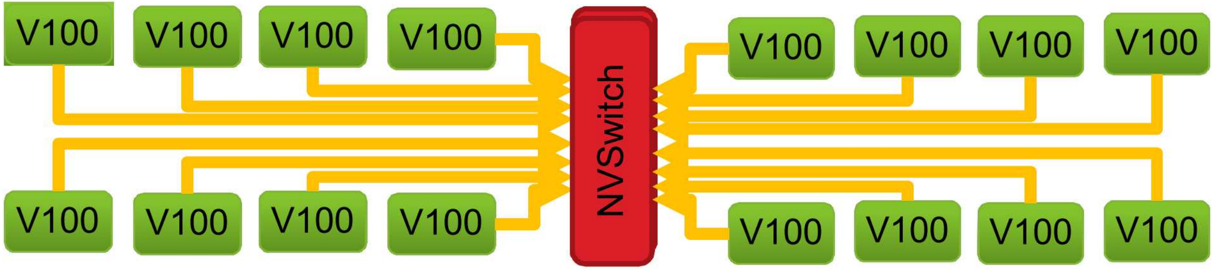
- Generalization of `TaskScheduler` abstraction to allow user to be generic with respect to scheduling strategy and queue
- Implementation of new queues and scheduling strategies:
  - Single shared LIFO Queue (this was the old implementation)
  - Multiple shared LIFO Queues with LIFO work stealing
  - Chase-Lev minimal contention LIFO with tail (FIFO) stealing
  - Potentially more
- Reorganization of Task, Future, TaskQueue data structures to accommodate flexible requirements from the TaskScheduler
  - For instance, some scheduling strategies require additional storage in the Task

Questions: David Hollman



Kokkos Remote Spaces: PGAS Support

- PGAS Models may become more viable for HPC with both changes in network architectures and the emergence of "super-node" architectures
  - Example DGX2
  - First "super-node"
  - 300GB/s per GPU link



- Idea: Add new memory spaces which return data handles with shmem semantics to Kokkos View
  - `View<double**[3], LayoutLeft, NVShmemSpace> a("A",N,M);`
  - Operator `a(i,j,k)` returns:

```
template<>
struct NVShmemElement<double> {
   NVShmemElement(int pe_, double* ptr_):pe(pe_),ptr(ptr_) {}
   int pe; double* ptr;
   void operator = (double val) { shmem_double_p(ptr,val,pe); }
};
```

PGAS Performance Evaluation: miniFE

- Test Problem: CG-Solve
  - Using the miniFE problem N^3
  - Compare to optimized CUDA
  - MPI version is using overlapping
  - DGX2 4 GPU workstation
  - Dominated by SpMV (Sparse Matrix Vector Multiply)
  - Make Vector distributed, and store global indicies in Matrix
- 3 Variants
  - Full use of SHMEM
  - Inline functions by ptr mapping
    - Store 16 pointers in the View
  - Explicit by-rank indexing
    - Make vector 2D
    - Encode rank in column index