2021 ALCF COMPUTATIONAL PERFORMANCE WORKSHOP

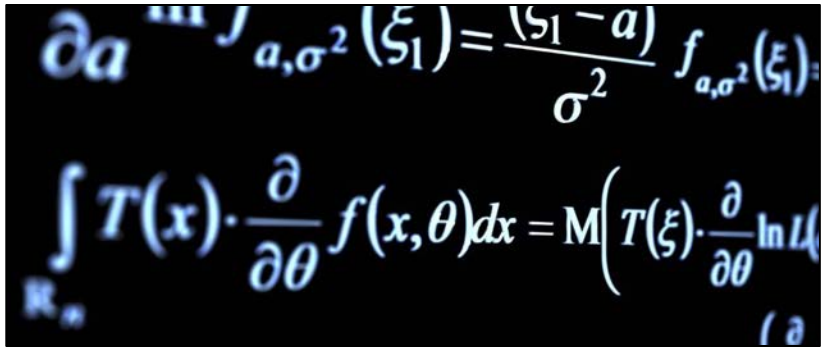
Argonne NATIONAL LABORATORY

KOKKOS PROGRAMMING MODEL

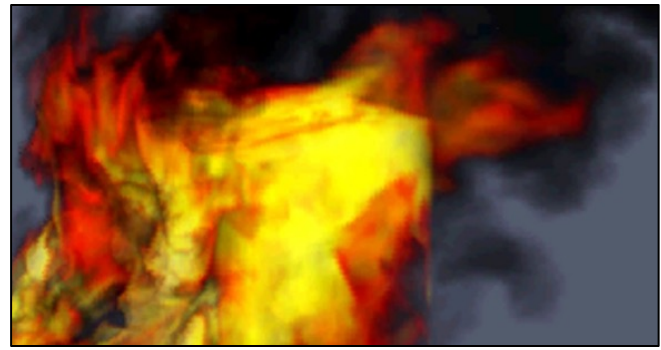
NEVIN LIBER
Argonne National Laboratory
Speaker

Exceptional service in the national interest

Sandia National Laboratories



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



The Kokkos Ecosystem

Unclassified Unlimited Release

Christian 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.

What is Kokkos?

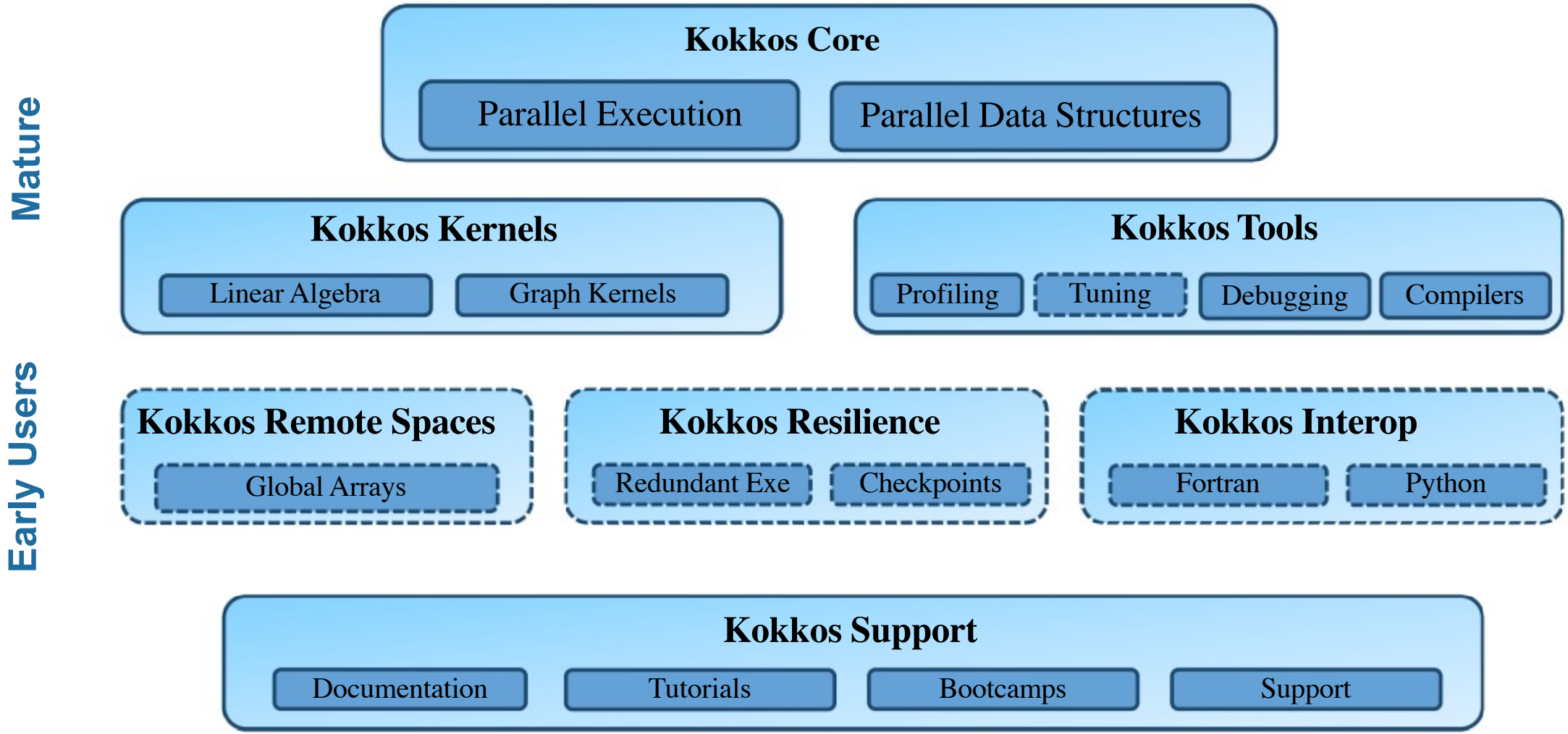
Sandia National Laboratories

- A C++ Programming Model for Performance Portability
  - Implemented as a template library on top of CUDA, OpenMP, HPX, SYCL, …
  - 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.

Kokkos 3.4.00

- Released 2021-Apr-26
  - Requires C++14
  - Some backends (e.g., SYCL) require C++17
  - Significant work on OpenMPTarget and SYCL backend functionality

Kokkos EcoSystem



Developer Team

- ***Kokkos Core***: 12 Developers
- 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

Sandia National Laboratories

Sandia National Laboratories

OAK RIDGE National Laboratory

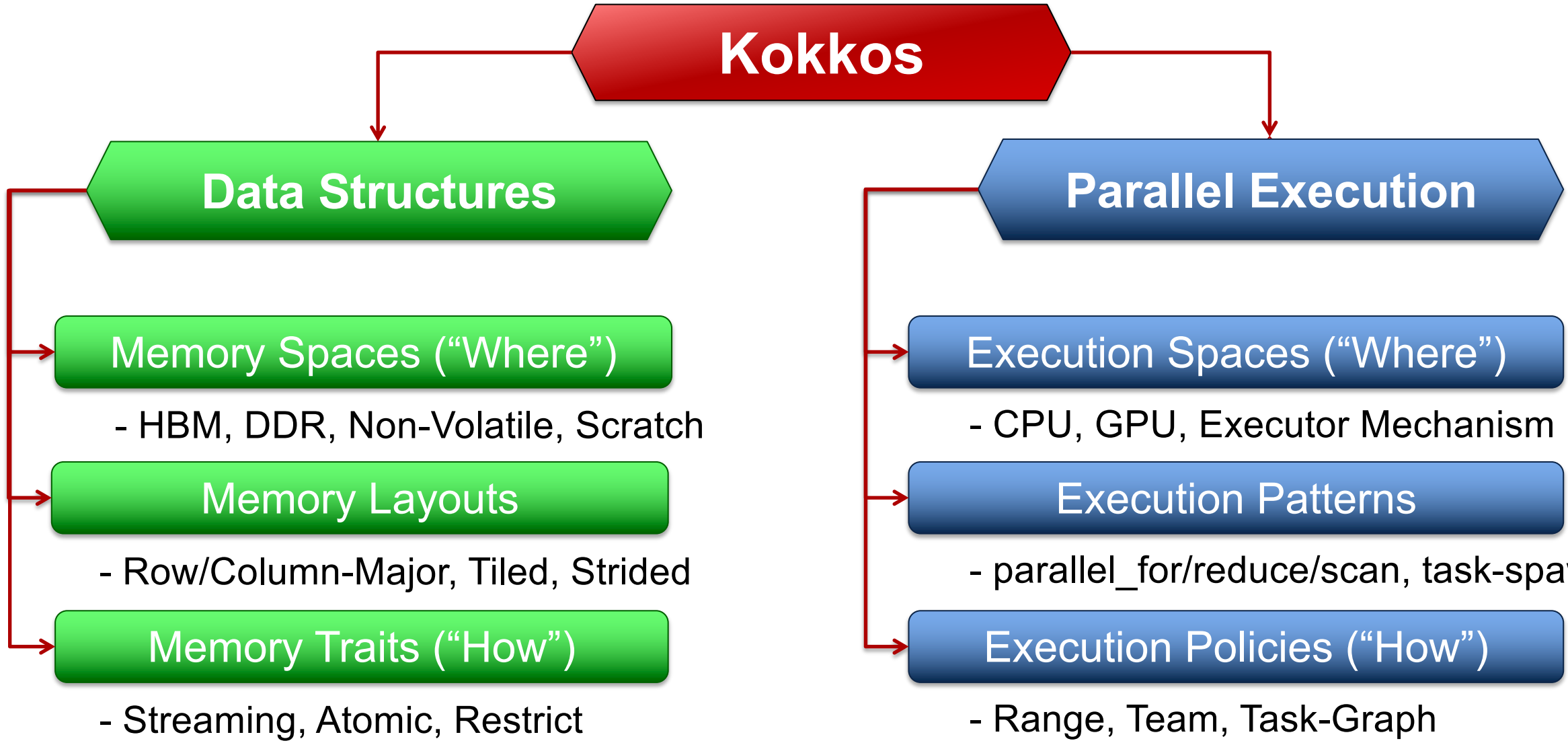
Argonne NATIONAL LABORATORY

Los Alamos NATIONAL LABORATORY EST. 1943

BERKELEY LAB

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

Kokkos Core Abstractions

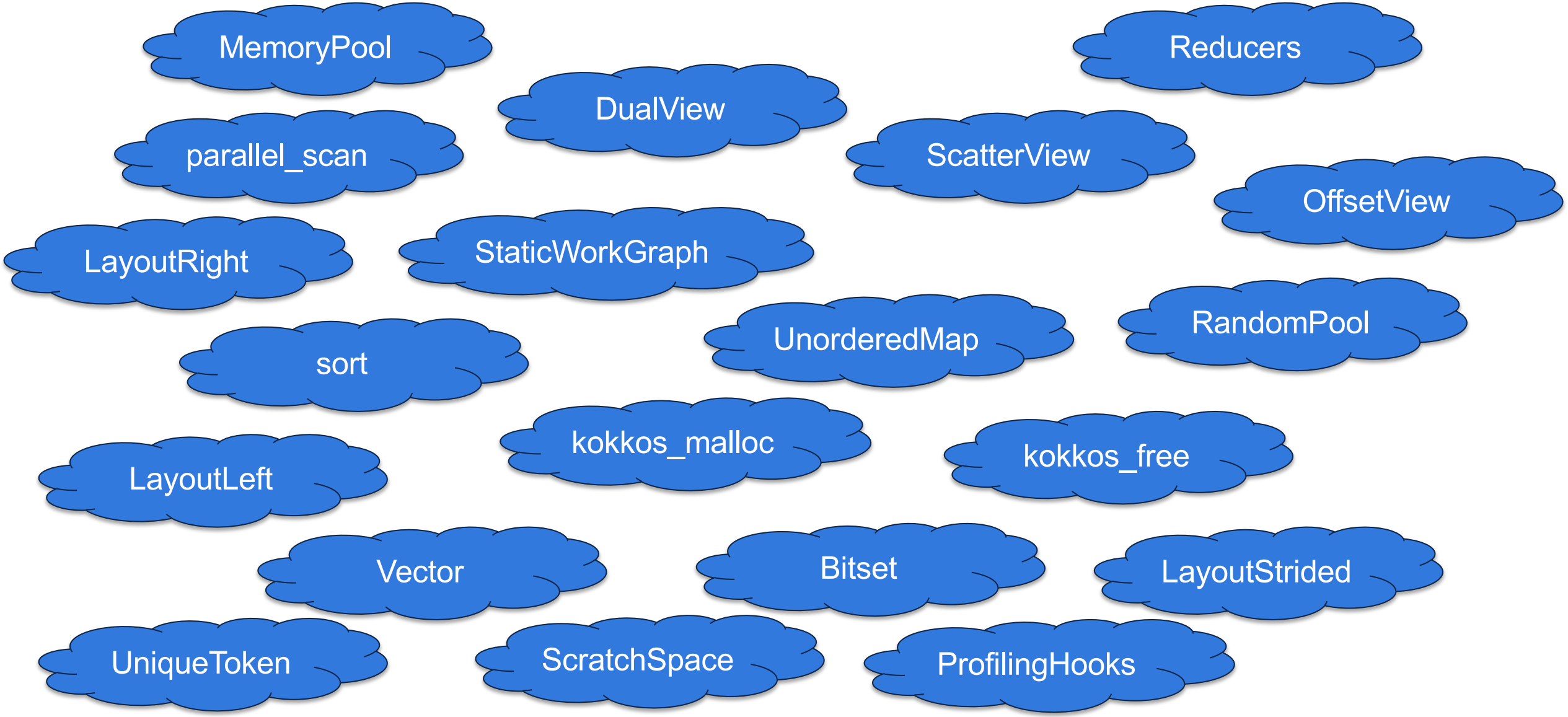


Kokkos Core Capabilities

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), DPC++/SYCL (experimental)

More Kokkos Capabilities



Example: Conjugent Gradient Solver

Sandia National Laboratories

- 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;
}
```

- 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

```
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

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

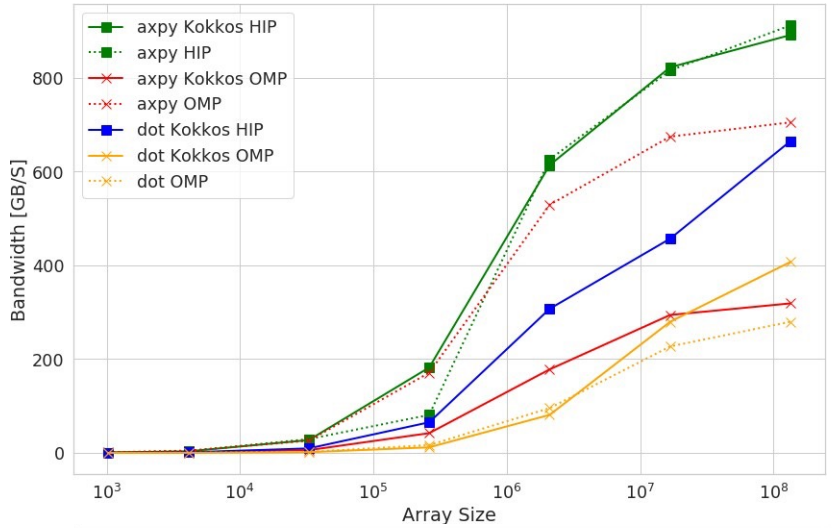
Distribute rows in workset over team-threads

Team Parallelism over Row Worksets

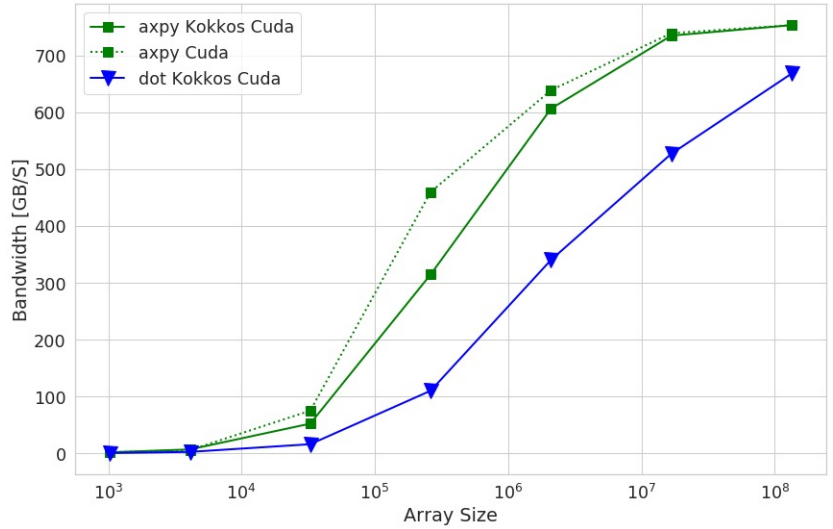
Performance Comparison: V100 vs MI100

AXPBY and DOT

MI100

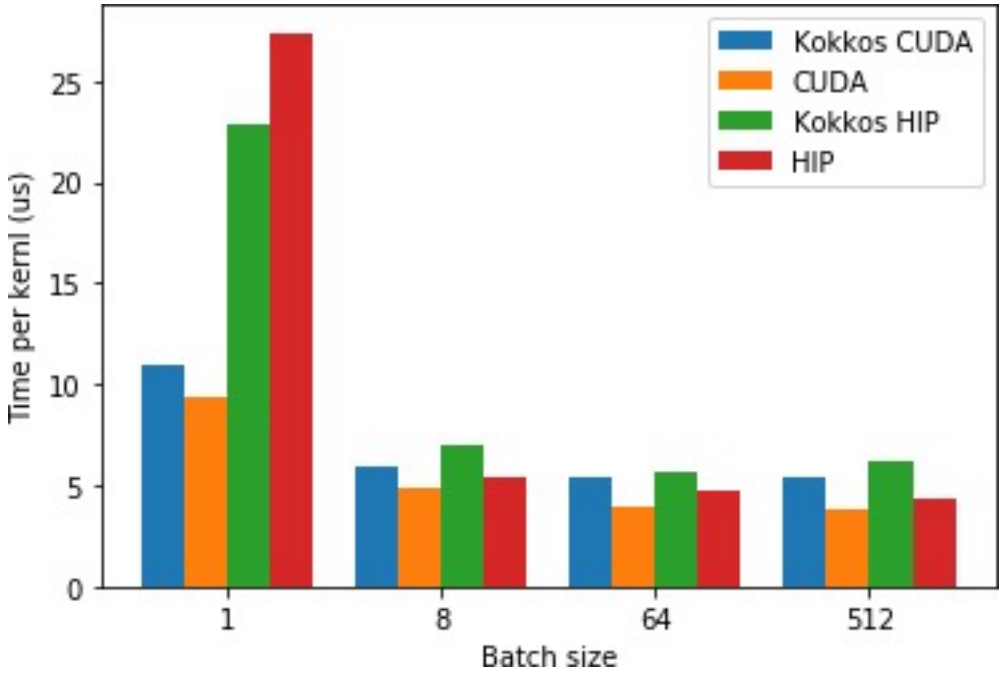


V100



Launch Latency Benchmark

(Batch Size = # of kernels before fence)



MI100 results are from our own system (lascaux02)

Kokkos Kernels

Sandia National Laboratories

- 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)
});
```

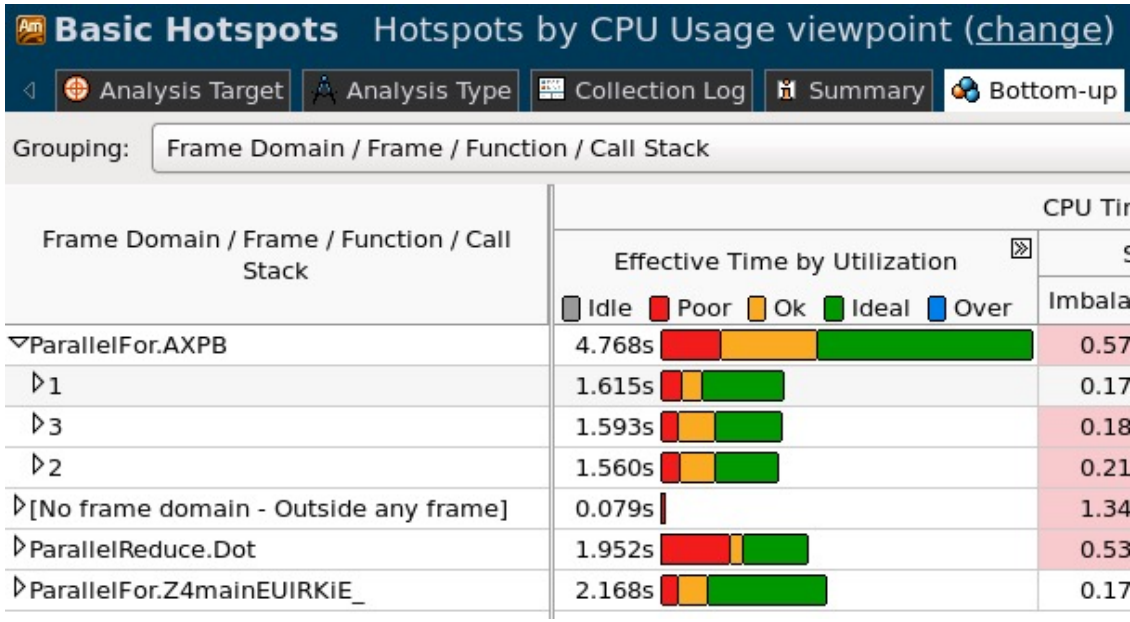
Kokkos Tools

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

- 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

Sandia National Laboratories

- 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);
    });
```

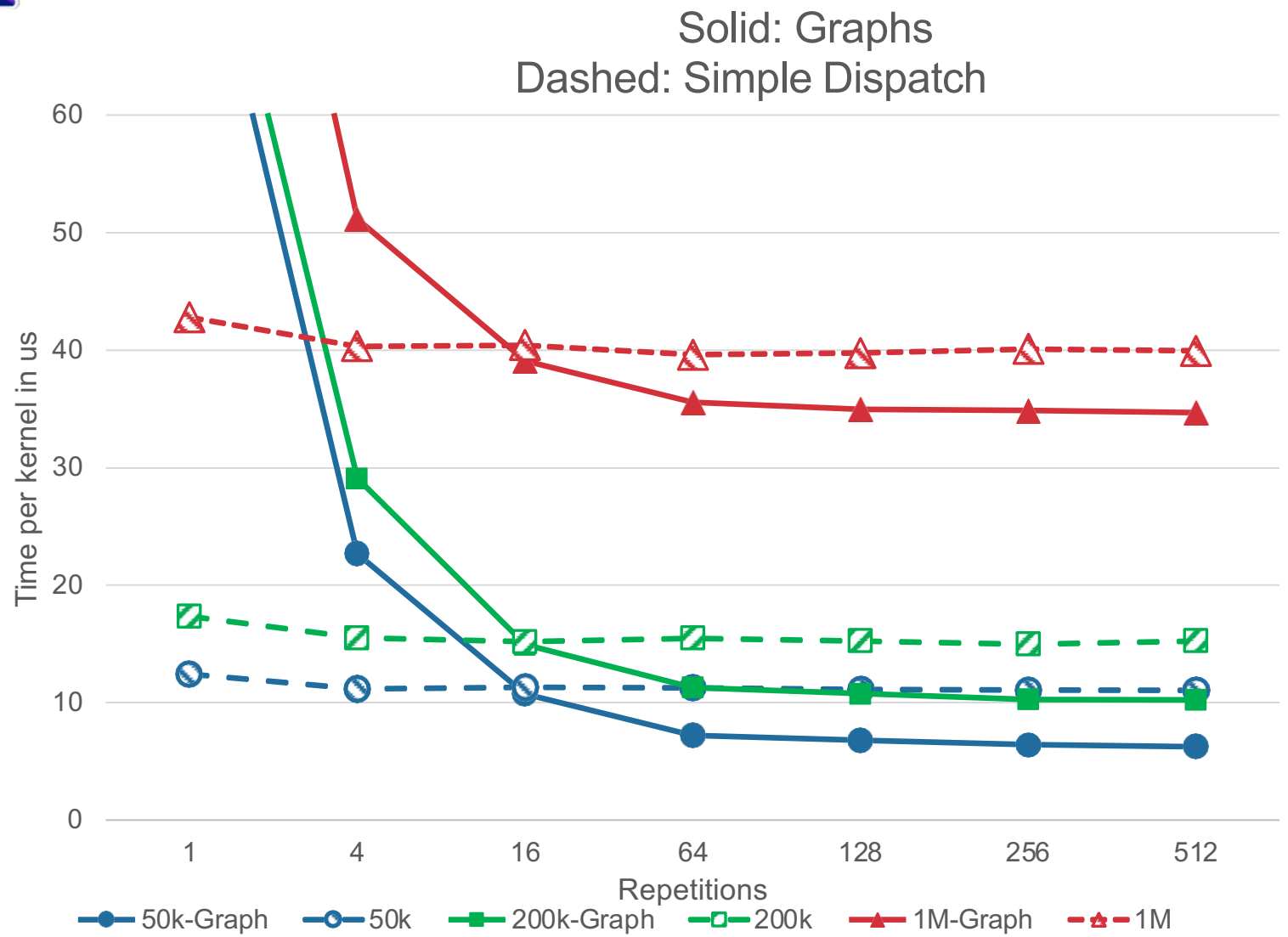
CUDA Graphs & KokkosGraphs

Sandia National Laboratories

- Kokkos Graphs in Release 3.3 (January 2021) expose CUDA Graphs
  - One Design Principle: less foot guns then CUDA Graphs
- API Design: Scoped creation/Explicit dependencies/no implicit capture

```
View<double*> x, y, z;
View<double, CudaHostPinnedSpace> x_dot_z;
double alpha, beta, gamma, threshold;
DefaultExecutionSpace ex();

auto graph = create_graph(ex, [&](auto root) {
      auto f_xpy = root.then_parallel_for(N, axpby_func{alpha,x,beta,y});
      auto f_zpy     = root.then_parallel_for(N, axpby_func{gamma,z,beta,y});

      //_____________________________________________
      auto ready = when_all(f_xpy, f_zpy);

      // _____________________________________________
      ready.then_parallel_reduce(N, dot_func{x,z}, x_dot_z);
};

while(x_dot_z()<threshold) { graph.submit(); Kokkos::fence(); }
```

Kokkos Graphs Benchmark



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

Kokkos Remote Spaces

Sandia National Laboratories

Add support for distributed memory spaces to facilitate software development for multi-node and multi-GPU executions.

Allow code reuse to scale across multiple memory address spaces at minimal development effort

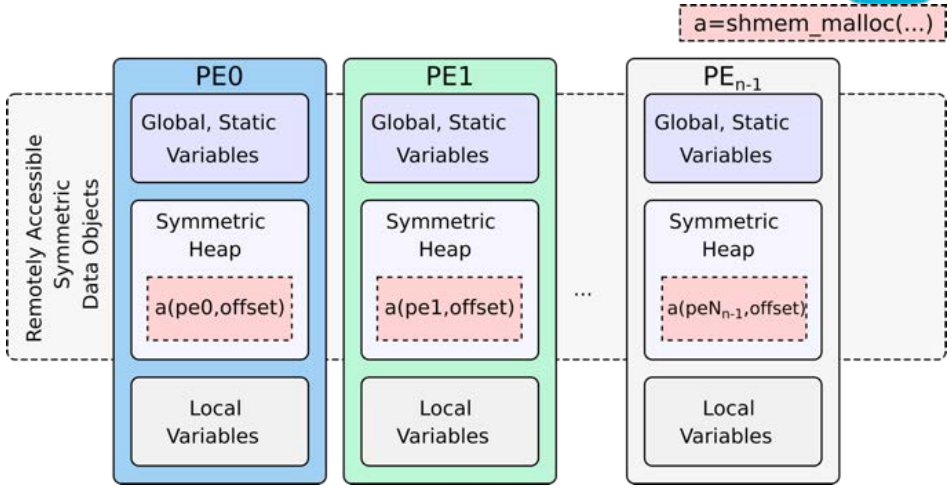
Adds distributed View support through templating on a remote memory space

Several remote memory space back-ends are support: MPI One-sided, SHMEM and NVSHMEM

Underlying implementation allocates data on the symmetric heap (all participants allocate the same size)

In development

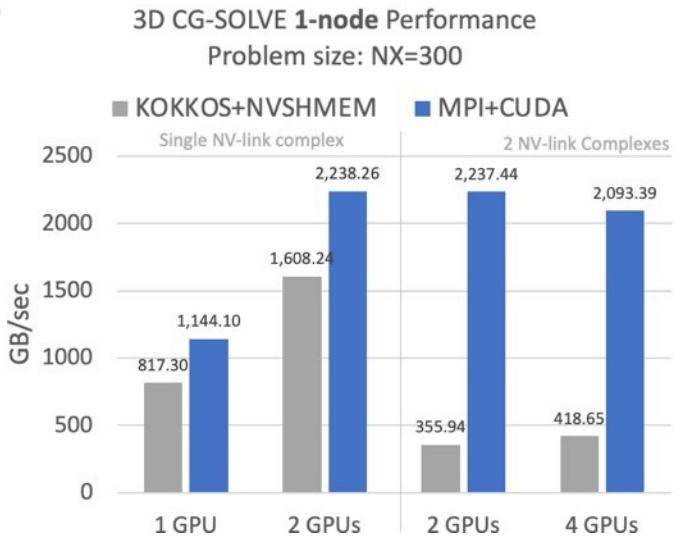
- Data shaping (subviews), block transfers (local_deep_copy) and memory access traits
- Performance optimizations (aggregation and caching)
- Support for other RMA implementations

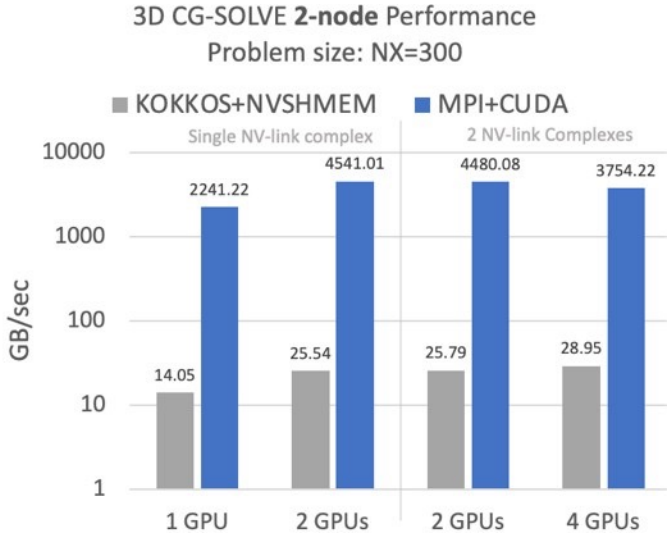


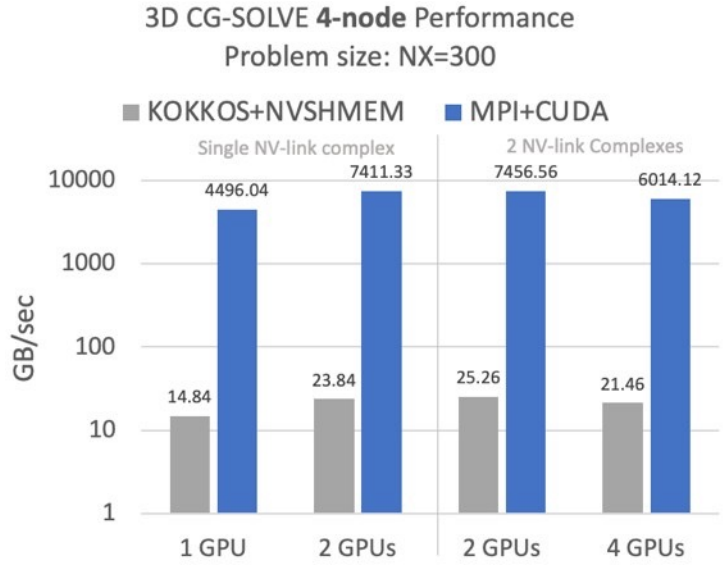

```
using RemoteSpace_t = Kokkos::Experimental::SHMEMSpace;
Kokkos::View<T**,RemoteSpace_t> a("A",numPEs,myN);

a(0,0) = 6; //Writes 6 to view a on PE 0 at offset 0
a(1,8) = 3; //Writes 3 to view a on PE 1 at offset 8
```

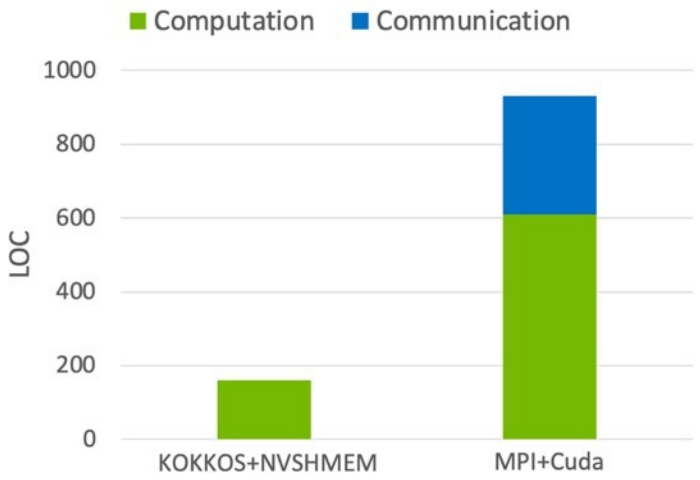
Examples: First dimension describes PE*. Note: This is a current design choice but subject to change as the PE can be deduced at runtime (at a small cost).

Example: CGSolve









Initial results with CGSolve: (relative to reference implementation with MPI+Cuda)

- 76% perf (1 NUMA)
- 20% perf (2 NUMA)
- Reduction in LOC: 5x

GPU can hide latency of fine-grained remote operations within a NUMA (GPU complex), but not across NIC/interconnect.

Important:

- **Packing, reuse and overlapping required**
- **Implicit Caching and Buffering**

Some Kokkos Users

Sandia National Laboratories

Pacific Northwest NATIONAL LABORATORY

Atlas 3D

Los Alamos NATIONAL LABORATORY EST. 1943

Sandia National Laboratories

ARL

NREL NATIONAL RENEWABLE ENERGY LABORATORY

MICHIGAN STATE

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

TECHNISCHE UNIVERSITÄT MÜNCHEN

IPP Max-Planck-Institut für Plasmaphysik

BERKELEY LAB

Rensselaer

Platform Strategy

Why OpenMP too?
- Fallback in case of compiler issues
- Interoperability with OpenMP libraries

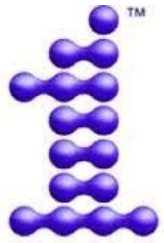
Sandia National Laboratories

Name	Perlmutter	Frontier	Aurora	ElCapitan	Crossroads	Fugaku
Facility	NERSC	ORNL	ANL	LLNL	LANL	RIKEN
Year	2021	2021	2022	2023	2022	2020
Hardware	NVIDIA GPU + AMD CPU	AMD GPU + AMD CPU	Intel GPU + Intel CPU	AMD GPU + AMD CPU	Intel CPU	ARM CPU
Toolchain	NVCC, NVC++	ROCM, Cray	Intel OneAPI	ROCM, Cray	Intel OneAPI, Cray, GCC	Fujitsu, GCC, ARM Clang
CUDA	✓					
OpenMP	(✓)	(✓)	(✓)	(✓)	✓	✓
DPC++/SYCL			✓			
HIP		✓		✓		

Kokkos SYCL backend

Sandia National Laboratories

- SYCL
  - Open standard (Khronos group)
  - Parallel heterogeneous computing
  - Intel implementation called DPC++
- Hardware support
  - Targeting Intel GPUs
  - CI running on NVIDIA V100



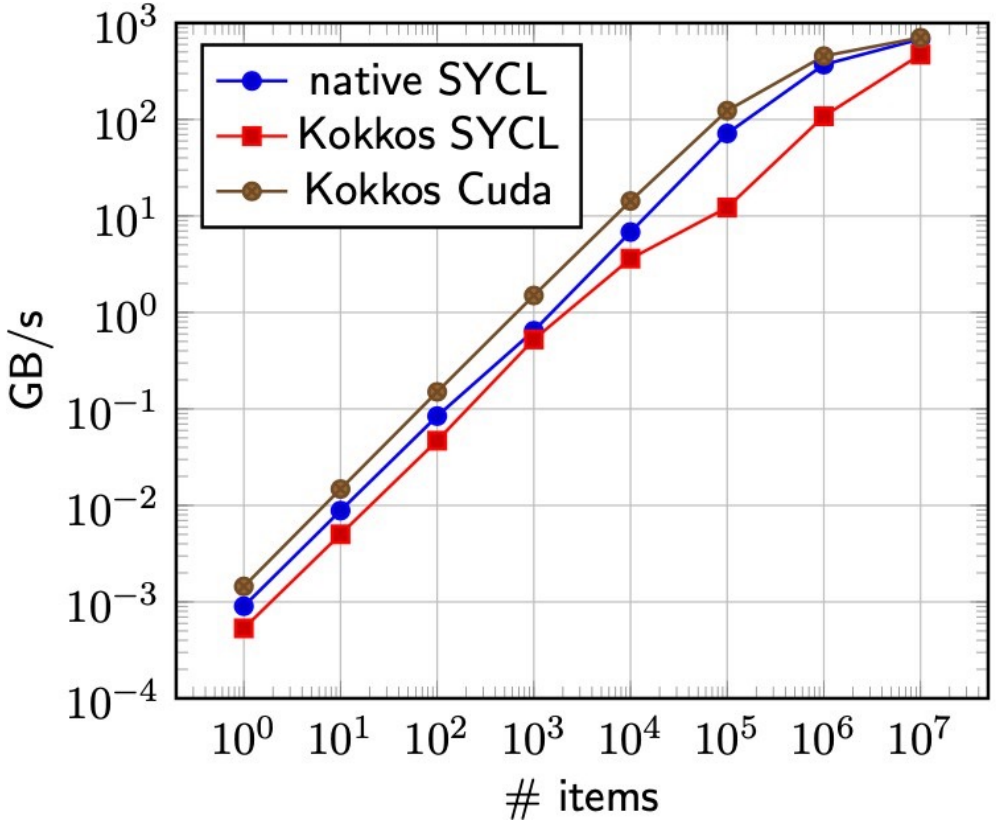
oneAPI

- Introduced in Kokkos-3.3 (released in Dec'20)
- Near feature complete in Kokkos-3.4 (Apr'21)
- Using DPC++ extensions of SYCL - may not work with other SYCL implementations
- Not yet implemented
  - atomics for types larger than 64bits
  - WorkGraphPolicy
  - Dynamic Task Graphs
- Future work will focus on performance
- Known to work with SYCL backend: ArborX, Cabana, LAMMPS

Kokkos SYCL backend – V100 Performance

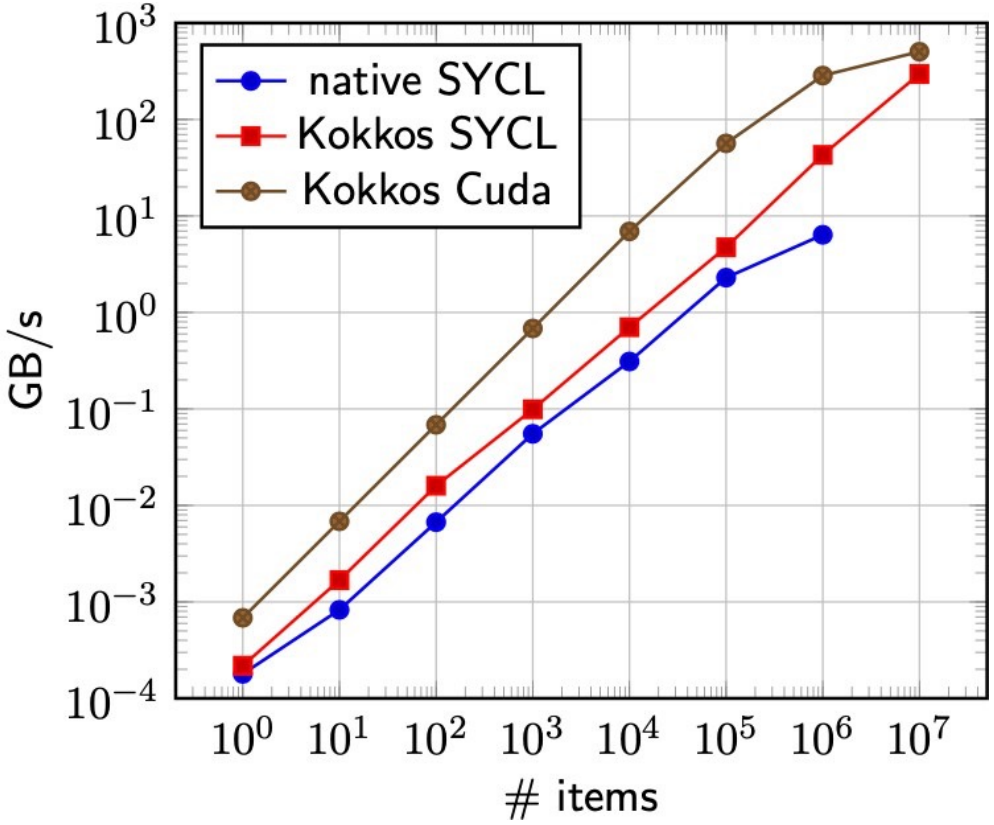
Sandia National Laboratories

AXPBY
(double scalar type)



parallel_for, AXPY

DOT
(double scalar type)



parallel_reduce, DOT

OpenMPTarget Backend

- Secondary backend for all GPU platforms
- Majority of features are available
- Started working with applications to test the backend

Compiler/Architecture Support Matrix

Architecture	clang	icpx	rocm	nvhpc	cce
NVIDIA (V100)					
AMD (MI60/MI100)					
Intel (Gen9)					

support partial support

Kokkos HIP backend

Sandia National Laboratories

- HIP:
  - native language of AMD's GPU
  - very similar to CUDA
  - supports both Nvidia and AMD's GPU
  - Still compiler issues -> tracking latest release very closely

AMD ROCm

- Primary backend for AMD GPUs
  - Do not support NVIDIA GPUs (use CUDA backend instead)
- Introduced in Kokkos-3.1 (released in Apr'20)
- Feature complete except for dynamic tasking
  - Focus so far has been on adding feature, now moving to performance
- We require ROCm 3.8, we will require 4.1 soon

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
- https://kokkosteam.slack.com Slack channel for user support

Kokkos: The Lectures

Sandia National Laboratories

- 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

THANK YOU