Sandia National Laboratories

kokkos

SAND2019-0508PE

Kokkos Roadmap 3.0

C.R. Trott, *D. Sunderland, N. Ellingwood, D. Ibanez, S. Bova, J. Miles, D. Hollman, D. Lanbreche, V. Dang*

Sandia National Laboratories
www.github.com/kokkos

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.

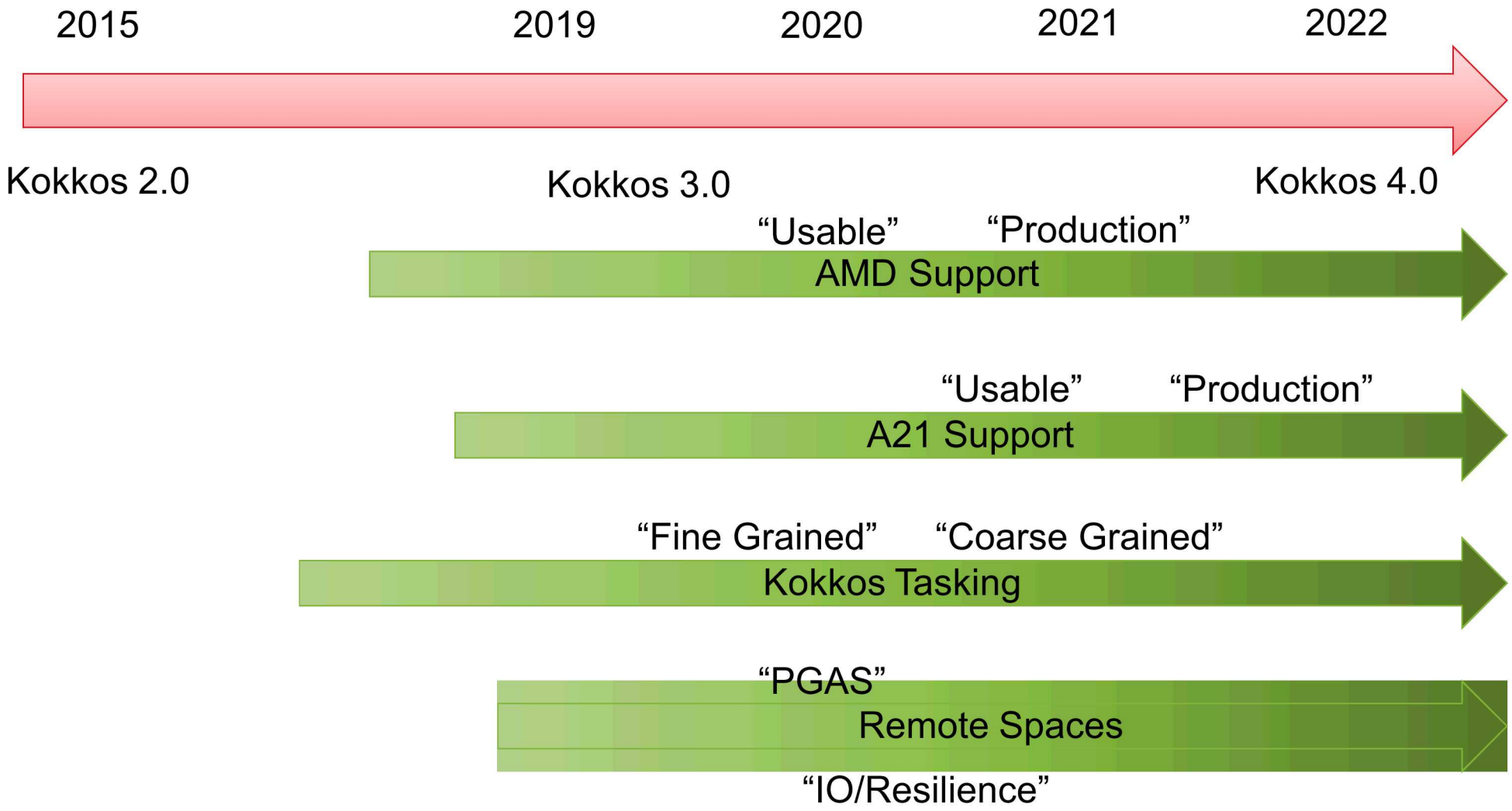
ECP
EXASCALE COMPUTING PROJECT

NNSA National Nuclear Security Administration

U.S. DEPARTMENT OF ENERGY | Office of Science

SAND2018-13126 C

Kokkos Timeline



2018 – What's New?

- Reorganization of the Kokkos Team
  - Carter Edwards left for NVIDIA => Christian Trott took over as PI
  - Team expanded: Jeff Miles, Vinh Dang, David Hollman
- Production support for Summit/Sierra
  - Tons of subtle issues around Volta's new threading semantics
- Deprecation of Features in preparation for Kokkos 3.0
  - https://github.com/kokkos/kokkos/wiki/DeprecationPage
  - Expect Kokkos 3.0 in H1 2019

LayoutTiled

- LayoutTiled : a requested feature from the SPARC development team
- SPARC
  - ATDM code approximates Navier-Stokes equations for atmospheric re-entry
  - is built on Kokkos and includes two independent algorithms
    - structured finite volume method (SFVM) uses a 4-dimensional array
    - unstructured finite volume method (UFVM) uses a 2-dimensional array
- The observed performance of the structured algorithm is less than unstructured*
- It is believed the culprit is the stride of the access pattern with the 4-d array
- Could tiling increase the locality?

Using LayoutTiled

- Designed so that minimal changes from other layouts are required to use it. Eg, if you have

```
using View4DType = View<double****, layout4D >;
```

  - Change the layout4D from e.g. ***LayoutRight*** to

```
Experimental::LayoutTiled<Iterate::Right, Iterate::Right, Tz_size, Ty_size, Tx_size>;
```

  - Change the MDRangePolicy from non-tiled

```
Experimental::MDRangePolicy <Kokkos::Rank<3>>({0,0,0}, {Nx,Ny,Nz}) ;
```

    to Tiled

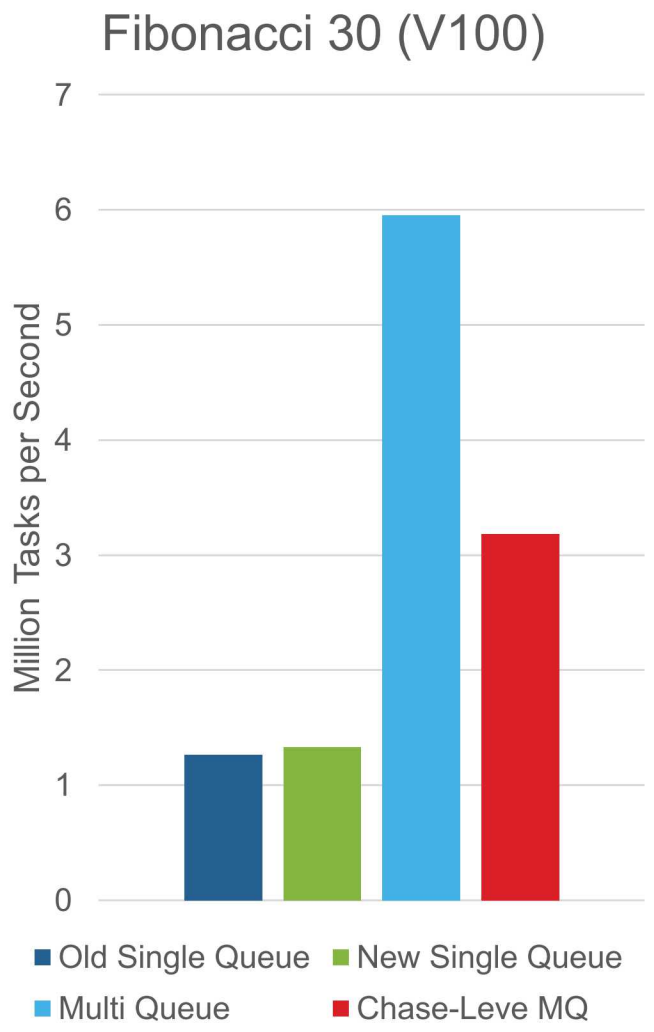
```
MDRangePolicy <Rank<3>>({0,0,0}, {Nx,Ny,Nz}, {Tx_size, Ty_size, Tz_size})
```

- The same functor code works with both layouts

```
KOKKOS_INLINE_FUNCTION
void operator() (int i, int j, int k) const {
     dfdx(k,j,i) = (fields(k+1,j,i, fieldIndex) - fields(k,j,i, fieldIndex) )*invhx;
}
```

Improving Kokkos Task Support

- 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



Kokkos Tasking Proposed Interface Changes

- Very minor interface changes so far:
  - Now requires the user to get the scheduler instance from the team member parameter (rather than storing it in a data member of the user's task functor)
  - Kokkos::task_spawn() and Kokkos::respawn() become sheduler.task_spawn() and scheduler.respawn()
  - Future type is now scheduler-specific; users should now use Kokkos::BasicFuture<ValueType, SchedulerType> instead of Kokkos::Future
    - (An alias is provided for backwards compatibility)
- Future directions:
  - More backend scheduling strategies, focusing on less allocation.
  - Correct use of memory orders in atomics on Volta and later
  - Aggregation of Kokkos::TaskSingle tasks into team tasks, when possible, and aggregation of team tasks for better use of cooperative groups
  - Long term: integration with macro-tasking (streams, CudaGraphs) for a uniform expression of coarse-grained tasking and fine-grained tasking in one programming model.

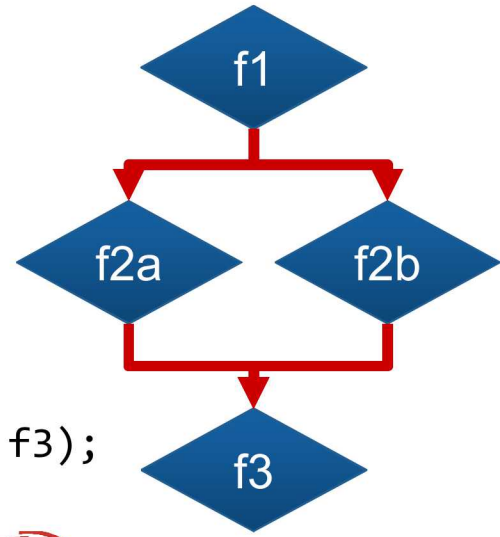
Tasking and C++ Standard

- 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

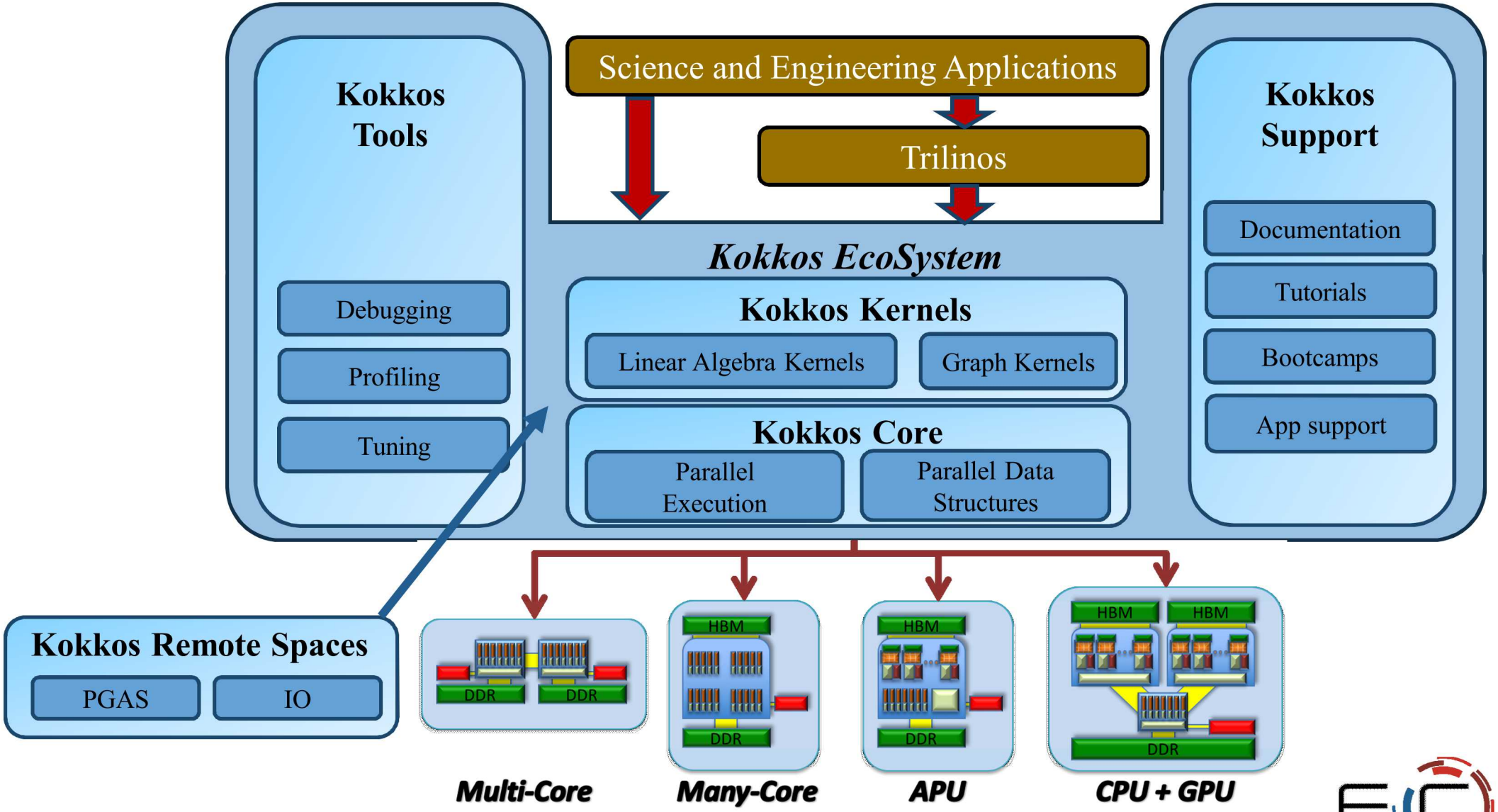
```
DefaultExecutionSpace spaces[2];
partition( DefaultExecutionSpace(), 2, spaces);
// f1 and f2 are executed simultaneously
parallel_for( RangePolicy<>(spaces[0], 0, N), f1);
parallel_for( RangePolicy<>(spaces[1], 0, N), f2);
// wait for all work to finish
fence();
```

  - 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);
```

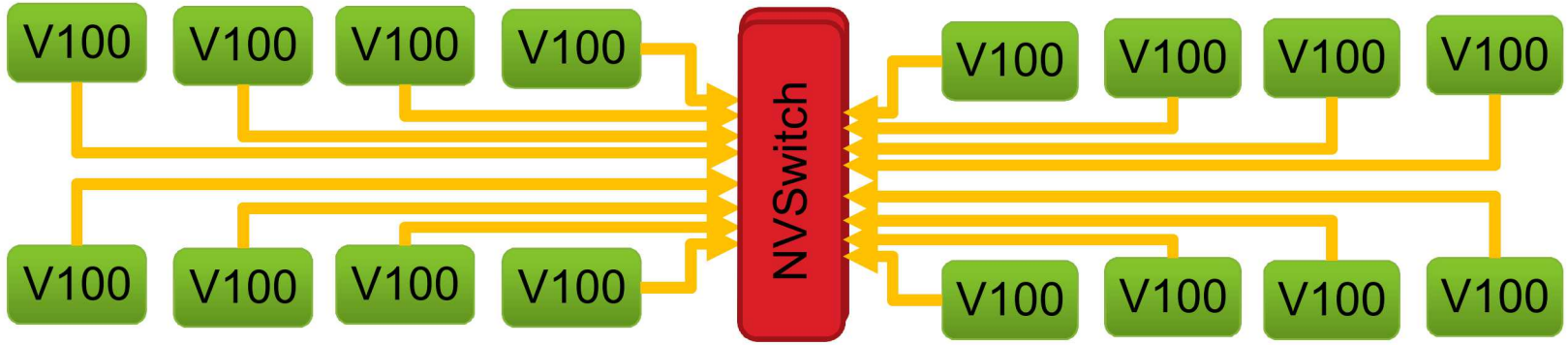


The Kokkos EcoSystem: Remote Spaces



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

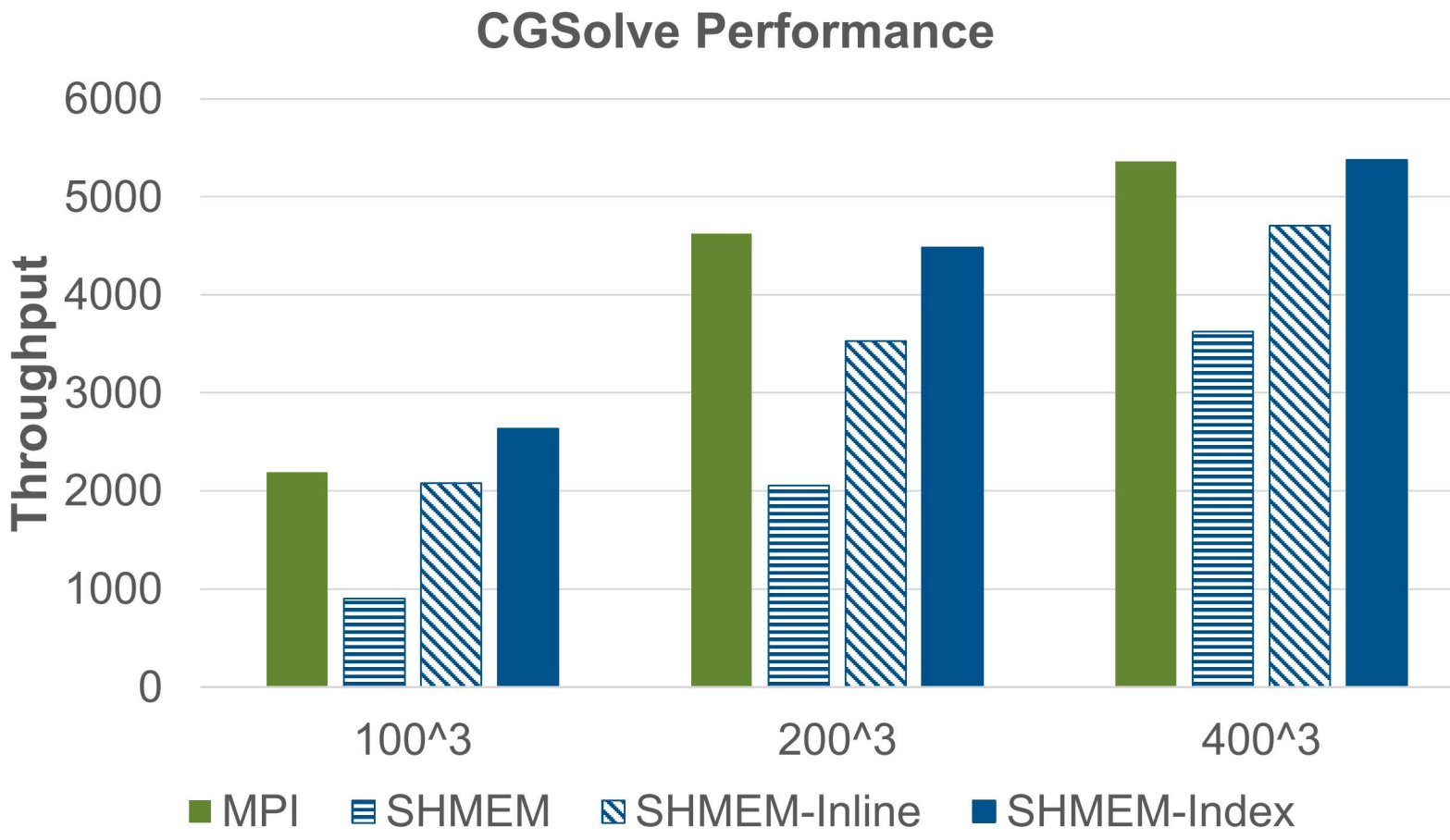


- Idea: Add new memory spaces which return data handles with shmem semantics to Kokkos View
  - `View<double**[3], LayoutLeft, NVShmemSpace> a(“A”,N,M);`
  - Operator `a(i,j,k)` returns:

```
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
- 3 Variants
  - Full use of SHMEM
  - Inline functions by ptr mapping
  - Explicit by-rank indexing



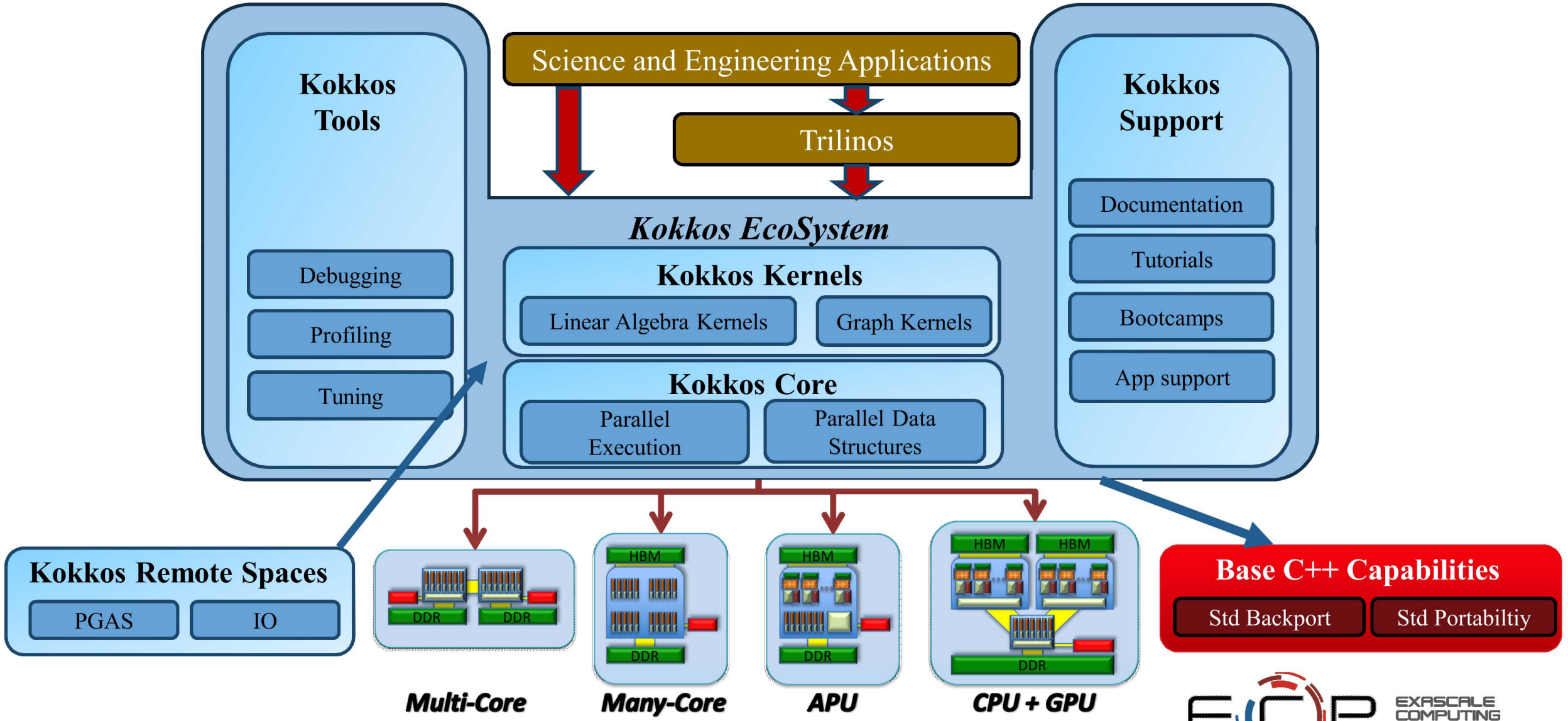
Kokkos Remote Spaces: Resilience/IO

- View abstraction flexible enough to help with more complex tasks
  - As with PGAS we can define custom storage and return types
  - Special hooks to do actions during launch of kernels
- Explore utilization of Views for IO and Resilience
  - HDF5 memory space
  - In memory cross MPI rank snapshotting
  - Automatic idempotentization of kernels
  - Interface to existing data warehouse libraries?

```
// Concept
View<int**, CheckPointSpace> a(“A”,N,M);
CheckPointSpace::
  create_check_point(“CP1”);
CheckPointSpace::
restore_from_check_point(a,“A”,“CP1”)
```

- Early stage exploration development
  - Many collaboration opportunities

The Kokkos EcoSystem



Base C++ Capabilities

- Collaboration with LLNL RAJA team
- Avoid replication of providing basic portable C++ capabilities
  - complex<…>, pair<…>, etc. for GPUs
- Backporting of future C++ capabilities
  - mdspan for multi dimensional arrays as underlying technology for Kokkos::View
  - atomic_ref for portable atomic support
- Expect first usable release end of 2019

Platform Support

- NVIDIA production support in place
- IBM/Intel/AMD/ARM CPU production support is in place
- AMD: Initial Support but complete restart happening based on AMD feedback
  - AMD's preferred (most mature) frontend changed: ROCm native to HIP
  - Expect usable HIP backend early 2020
  - Production support early 2021
- A21 Support: Tracking known information
  - Expect usable backend end of 2020
  - Production support end of 2021