SAND2019-8761C

Sandia National Laboratories

Performance Portable Resilience with Kokkos

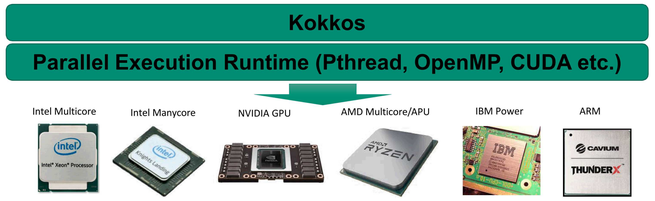
Background

Problem Description

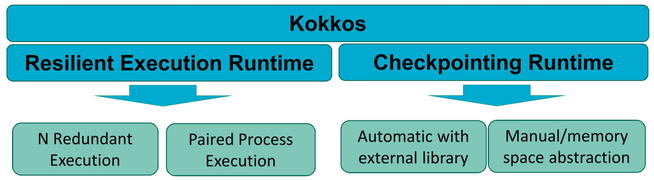
- HPC architectures are vastly different and continuously evolving
- Large scale computer systems remain prone to hardware failures
- No standard interface for HPC resilience libraries

Approach

Portability is achieved with Kokkos by abstraction of memory access and execution runtime

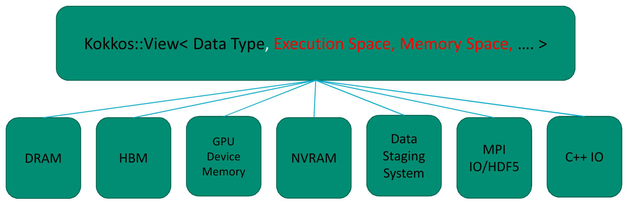


Portable Resilience is achieved with Kokkos by abstraction of resilience libraries



Memory Space Abstraction

Kokkos View abstraction simplifies access to storage media



Code Simplification

Kokkos Checkpointing abstraction simplifies user code eliminating proprietary API calls

Direct Library access

```
VELOC_Mem_protect(0, &i, 1, sizeof(int));
VELOC_Mem_protect(1, h, M * nbLines, sizeof(double));
VELOC_Mem_protect(2, g, M * nbLines, sizeof(double));
int v = VELOC_Restart_test("heatdis", 0);
if (v > 0) {
    VELOC_Restart_test is returning
    assert(VELOC_Restart("heatdis", v) == VELOC_SUCCESS);
} else
    i = 0;
    while (i < n) {
    // iteratively compute the heat distribution
    // (5): checkpoint every K iterations
    if (i % K == 0)
        assert(VELOC_Checkpoint("heatdis", i) ==
                                VELOC_SUCCESS);
     // increment the number of iterations
     i++;
    }
}
```

Kokkos

```
for (i = 0; i < n; i++) {
   KokkosResilience::checkpoint( "resilience_context, "final",
                                 n, [=]() mutable {
      Kokkos::parallel_for(rp,KOKKOS_LAMBDA(const int i){
         m_data(i)=i;
      }), KokkosResilience::filter::
                     nth_iteration_filter< 10 >{} );
}
```

References

- H. C. Edwards, C. R. Trott and D. Sunderland, "Kokkos: Enabling manycore performance portability through polymorphic memory access patterns," Journal of Parallel and Distributed Computing , vol. 74, no. 12, pp. 3202-3216, 2014.
- I.P. Egwutuoha, D. Levy, B. Selic, S. Chen, "A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems", The Journal of Supercomputing, September 2013, Vol. 65, Issue 3, pp 1302-1326
- Argonne National Laboratory, "VeloC" Argonne National Laboratory, 2018. [Online]. Available: https://veloc.readthedocs.io/en/latest/.

Implementation

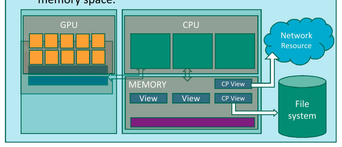
Manual Checkpoint

Provide simple interface to checkpoint data using Kokkos views and memory space concept

- Data to be checkpointed is designated by a "mirror" view in a checkpoint memory space
- Checkpoint memory space manages list of checkpoint views and performs a deep copy from one to the other during checkpoint / restart operation.
- Application code determines directory structure, but Kokkos will create and attach to file based memory spaces

Kokkos View

- The Kokkos view provides an abstract data parallel accessor to storage media simplifying data movement and adaptation to different hardware
- Checkpoint views are mirrors of host views with matching extents and layouts, but in an off-node memory space.



Example

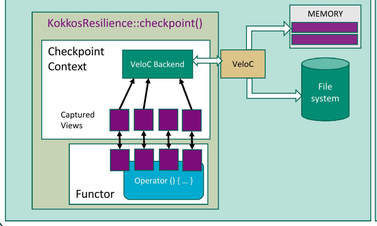
```
typedef Kokkos::StdFileSpace cp_type;
cp_type sfs;
typedef Kokkos::DirectoryManager<cp_type> dm_type;

auto x_cp = Kokkos::create_chkpt_mirror( sfs, atom.x );
auto v_cp = Kokkos::create_chkpt_mirror( sfs, atom.v );

for ( int n = 0; n < ITERATION_MAX; n++ ) {

    // iteration loop …


    if ( ( n % CHECKPOINT_FREQ ) == 0 ) {
       // create directory <PATH>/n and attach to cp
       dm_type::set_checkpoint_directory( true,
                                  cp_path.c_str(), n);
       cp_type::checkpoint_views();
    }
}
```

Automatic Checkpointing

Simplify checkpoint/restart selection with abstraction to Checkpoint implementation

- Data to be checkpointed is captured from Kokkos views contained in functor
- Captured Views are passed to checkpoint context
- Checkpoint context aggregates a "Backend" implementation which is specific to existing checkpointing interface
  - e.g. VeloC
- During Checkpoint and restart, data locations referenced by captured views are passed to context backend

Automatic Checkpointing with VeloC



Example

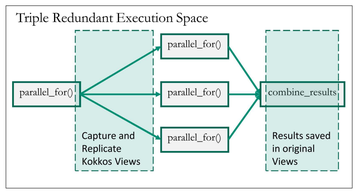
```
typedef Kokkos::View< double* > view_type;

typedef KokkosResilience::Context<
          KokkosResilience::VeloCCheckpointBackend > rc_type;
rc_type resilience_context = rc_type(MPI_COMM_WORLD,
                                     "config.dat");
view_type m_data ( "data", D );
Kokkos::RangePolicy<> rp(0,D);

for (int n = 0; n < N; n++ ) {
  KokkosResilience::checkpoint( *resilience_context, "final",
                                n, [=]() mutable {
      Kokkos::parallel_for(rp,KOKKOS_LAMBDA(const int i){
         m_data(i)=i;
      }), KokkosResilience::filter::
                     nth_iteration_filter< 10 >{} );
}
```

Redundant execution

Resilience is achieved by executing the same operation with multiple streams

- Redundant execution occurs in three steps
  - Replicate data referenced in functor
  - Concurrent execution using three parallel execution spaces
  - Recombine modified data using voting mechanism
- Replicated Data captured through Kokkos Views
- Concurrent execution is via Streams (CUDA) or Tasks (OpenMP)
- Voting step uses comparison operator sensitive to datatype
  - '==' for enumerated types (int, long, char, bool, etc.)
  - |a-b|<Ɛ for float and double

Triple Redundant Execution Space



Example

```
typedef Kokkos::View< double*, Kokkos::CudaSpace > view_type;

view_type m_data ( "data", N );
typename view_type::HostMirror v =
                          Kokkos::create_mirror_view(m_data);
Kokkos::RangePolicy<Kokkos::ResCuda> rp(0,N);

Kokkos::parallel_for(rp,KOKKOS_LAMBDA(const int i){
    m_data(i)=i;
});
Kokkos::fence();
Kokkos::deep_copy(v, m_data);
```

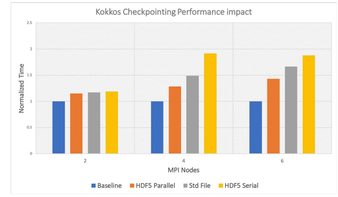
Results

Manual Checkpoint

- Benchmark manual checkpoint using miniMD (Kokkos mini app for molecular dynamics)
- Checkpoint taken every 10 iterations
- Note that because of the nature of the miniMD application, the checkpoint size also scales with the number of nodes.

Setup Name	Description
Baseline	Kokkos MiniMD application with MPI + OpenMP
HDF5 Parallel	Baseline + manual checkpoint to single DataSize x (MPI Size) file
Std File	Baseline + manual checkpoint to (MPI Size) files using std::stream
HDF5 Serial	Baseline + manual checkpoint to (MPI Size) files using HDF5

Description of test setup illustrated in comparison plots



Automatic Checkpoint with VeloC backend

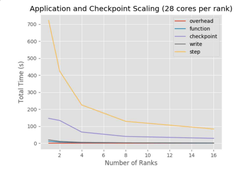
- Benchmark automatic checkpoint using miniMD (Kokkos mini app for molecular dynamics)
- Checkpoint taken every 10 iterations
- Checkpoint size scales with the number of nodes – see table

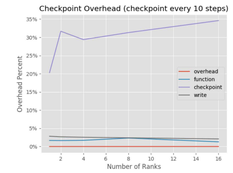
MPI Ranks	Data Size	VeloC Metadata Size	Checkpoint Scratch Size
1	3.1GB	0	61GB
2	1.6GB	1.6GB	123GB
4	799MB	267MB	84GB
8	407MB	59MB	73GB
16	211MB	31MB	76GB

Data usage by MPI rank for VeloC backend

Stage	Description
overhead	Non-VeloC overhead from Kokkos Resilience objects
function	Computation within the checkpoint functor
checkpoint	Checkpoint operation include VeloC overhead and file writes
write	File writes only
step	Entire time step including MPI and non-captured computations

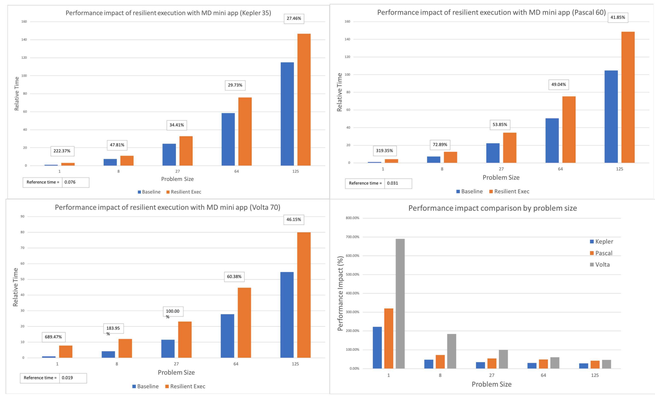
Description of stages illustrated in comparison plots





Triple Redundancy Execution with Resilient Cuda Space

- Benchmark execution redundancy using miniMD (Kokkos mini app for molecular dynamics)
- Redundant execution only added to the integration loop (accounted for in the figures)



Conclusions

- Checkpointing through Kokkos View adds a seem-less point of data integration
  - Kokkos provides automatic or manual interface giving users selective control over implementation
  - Easily integrate with existing checkpoint libraries without changing user code
- Kokkos execution resilience provides software redundancy without adding code complexity
  - Able to take advantage of available hardware concurrency through streams
  - Automatic data replication and recombination using parallel execution when possible minimizes execution cost

U.S. DEPARTMENT OF ENERGY
NNSA National Nuclear Security Administration

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.
SAND No. _____

Sandia National Laboratories