Portable User-level Resilience with Kokkos

ECP Annual Meeting 2020

PRESENTED BY

Keita Teranishi

Jeff Miles, Nicolas Morales, Carson Mould, Matthew Whitlock
Background

- Resilience is the norm for HPC
  - ECP Goal: 1 week MTBF with Checkpoint/Restart

- VeloC (SCR) serves the Checkpoint/Restart capability
  - Multilevel Checkpointing
  - Use of the new storage technology
  - APIs to support the major programming languages

- Modern C++ accommodates abstraction of data and computation for performance portability
  - Kokkos and Raja
  - We add “resilience” abstraction to allows seamless integration of VeloC+Kokkos
Resilient Kokkos

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

Portable Resilience is achieved with Kokkos by abstraction of resilience libraries.

Kokkos

Parallel Execution Runtime (Pthread, OpenMP, CUDA, HIP, SYCL etc.)

- Intel Multicore
- Intel Accelerator
- NVIDIA GPU
- AMD Multicore
- AMD GPU
- IBM Power
- ARM

Kokkos

Resilient Execution Runtime

- N Redundant Execution
- Paired Process Execution

Checkpointing Runtime

- Automatic with external library
- Manual/memory space abstraction
Kokkos View abstraction simplifies access to storage media

Kokkos::View< Data Type, Execution Space, Memory Space, .... >

Memory Space for Resilient Kokkos
Kokkos Checkpointing abstraction simplifies user code eliminating proprietary API calls

Direct Library access

```c
VELOC_Mem_protect(0, &i, 1, sizeof(int));
VELOC_Mem_protect(1, h, M * N, sizeof(double));
VELOC_Mem_protect(2, g, M * N, 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

```c
Kokkos::View<double *, ResilientSpace> h("H", M*N);
Kokkos::View<double *, ResilinetSpace> g("G", M*N);
for (i = 0; i < n; i++) {
    Kokkos::checkpoint(*ctx, "heatdis", i, [=]() {
        Kokkos::parallel_for(n,KOKKOS_LAMBDA(int i){
            // Compute h and g
            h(i) = g(i-1)+g(i+1) ... ;
        });
    });
}
```
Manual Checkpoint

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

<table>
<thead>
<tr>
<th>Kokkos View</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Data to be checkpointed is designated by a “mirror” view in a checkpoint memory space</td>
<td>typedef Kokkos::StdFileSpace cp_type;</td>
</tr>
<tr>
<td>• Checkpoint memory space manages list of checkpoint views and performs a deep copy from one to the other during checkpoint / restart operation.</td>
<td>cp_type sfs;</td>
</tr>
<tr>
<td>• Application code determines directory structure, but Kokkos will create and attach to file based memory spaces</td>
<td>typedef Kokkos::DirectoryManager&lt;cp_type&gt; dm_type;</td>
</tr>
</tbody>
</table>

```cpp
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 configuration:
  - Json file (“config.json”) or programmatic
  - Checkpoint interval (nth iteration or every n seconds)
  - Backend (e.g. VeloC)
- During Checkpoint and restart, data locations referenced by captured views are passed to context backend

Example

typedef Kokkos::View< double* > view_type;

auto ctx = KokkosResilience::make_context(MPI_COMM_WORLD, "config.json");

view_type m_data ( "data", D );
Kokkos::RangePolicy<> rp(0,D);

for (int n = 0; n < N; n++) {
  KokkosResilience::checkpoint( *ctx, "final",
    n, [=](){ mutable {
      Kokkos::parallel_for(rp,KOKKOS_LAMBDA(int i){
        m_data(i)=i;
      });
    });
  }
}
Current Status

Performance test
- MiniMD and 2D stencil
- Sandia’s small Cray XC40 (Being tested up to 80 nodes)
- ANL Theta
- NERSC Cori

Software Release
- Special branch in https://github.com/kokkos/kokkos (Coming Soon)

Future work
- Accelerator systems