Interoperability of Shared Memory Parallel Programming Models with Charm++

Jæmin Choi

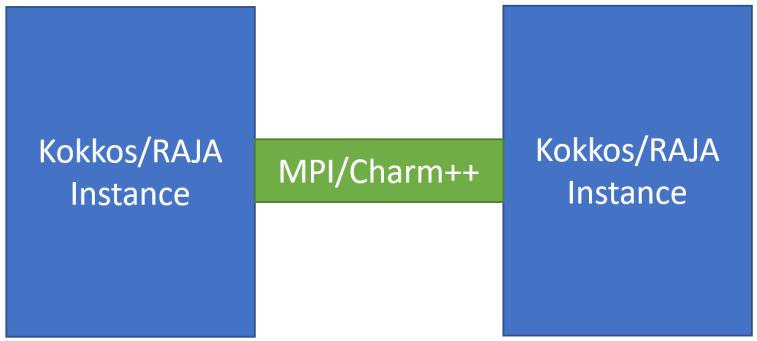
University of Illinois Urbana-Champaign

May 2, 2019

Overview

1. Why Interoperate with Charm++?
2. Compiling Libraries
3. Creating Hybrid Programs
4. Vector Addition Example
5. Kokkos vs. RAJA
6. Future Work

Why Interoperate with Charm++?



- **Kokkos** (SNL) and **RAJA** (LLNL)
  - 'Performance portability'
  - Abstractions for parallel execution and data management
- Limited to shared memory parallelism by itself
- Use MPI for distributed memory execution
- Charm++ is another option
  - Support for wide variety of architectures
  - Load balancing

Basic Interoperability

- Let Kokkos/RAJA handle shared memory parallelism
  - **OpenMP** backend for CPU
  - **CUDA** backend for GPU
- Use Charm++ for communication between processes (intra- & inter-node)

Compilation: Kokkos

```
mkdir build && cd build
../generate_makefile.bash --prefix=<absolute path to build> \
--with-cuda=<path to CUDA toolkit> --with-cuda-options=enable_lambda \
--with-openmp --arch=<CPU arch>,<GPU arch> --compiler=<path to included NVCC wrapper>
make -j kokkoslib
make install
```

- Assume GPUs are available
- OpenMP and CUDA backends
- **Headers** (`build/include`) and **library** file (`build/lib`) after install are all we need

Compilation: RAJA

```
mkdir build && mkdir install && cd build
cmake -DENABLE_CUDA=On -DCMAKE_INSTALL_PREFIX=<path to RAJA install folder> ../
make -j
make install
```

- Assume GPUs are available
- OpenMP and CUDA backends
- **Headers** (`install/include`) and **library** file (`install/lib`) after install are all we need

Creating a Kokkos/RAJA + Charm++ Hybrid Program

- Write Kokkos/RAJA code in a `.cpp` file
  - Can be put in the same file as Charm++ if GPU is not used (if CUDA backend not built)
- Write Charm++ code in a separate `.C` file
  - A nodegroup chare for each Kokkos/RAJA instance
- Compile Kokkos/RAJA code with NVCC
  - Additional options needed (e.g. `-fopenmp`)
  - Use NVCC wrapper with Kokkos
- Use `charmc` to compile Charm++ code and link
  - Need to link Kokkos/RAJA library
- Examples (Hello World, vector addition) in `examples/charm++/shared_runtimes/[kokkos,raja]`

Vector Addition Example: Kokkos

```
mainmodule vecadd {
  ...
  mainchare Main { ... }

  // Encapsulate a Kokkos instance/process
  nodegroup Process {
    entry Process();
    entry void run();
  }
}
```

Listing 1: `vecadd.ci`

Vector Addition Example: Kokkos

```
...
class Process : public CBase_Process {
public:
  Process() {
    kokkosInit(); // Calls Kokkos::initialize() internally
  }

  void run() {
    // Execute vector addition
    // Uses OpenMP by default, uses CUDA if use_gpu
    vecadd(n, CkMyNode(), use_gpu);

    kokkosFinalize(); // Calls Kokkos::finalize() internally

    // Contribute to Main to end the program
    ...
  }
}
```

Listing 2: `vecadd_charm.C`

Vector Addition Example: Kokkos

```
#include <Kokkos_Core.hpp>
...

// Views
typedef Kokkos::View<double*, Kokkos::LayoutLeft, Kokkos::CudaSpace> CudaView;
typedef Kokkos::View<double*, Kokkos::LayoutRight, Kokkos::CudaHostPinnedSpace> HostView;

// Functors
template <typename ViewType>
struct Compute {
  ViewType a, b;
  Compute(const ViewType& d_a, const ViewType& d_b) : a(d_a), b(d_b) {}
  KOKKOS_INLINE_FUNCTION
  void operator() (const int& i) const {
    a(i) += b(i);
  }
}
...

void vecadd(const uint64_t n, int process, bool use_gpu) {
  HostView h_a("Host A", n); CudaView d_a("Device A", n); CudaView d_b("Device B", n);
  Kokkos::parallel_for (Kokkos::RangePolicy<Kokkos::Cuda>(0, n),
                        Compute<CudaView>(d_a, d_b));
  Kokkos::deep_copy(h_a, d_a);
}
```

Listing 3: `vecadd_kokkos.cpp`

Vector Addition Example: RAJA

```
mainmodule vecadd {
  ...
  mainchare Main { ... }

  // Encapsulate a RAJA instance/process
  nodegroup Process {
    entry Process();
    entry void run();
  }
}
```

Listing 4: `vecadd.ci`

Vector Addition Example: RAJA

```
...
class Process : public CBase_Process {
public:
  Process() {
    // No initialization/cleanup needed
  }

  void run() {
    // Execute vector addition
    // Uses OpenMP by default, uses CUDA if use_gpu
    vecadd(n, CkMyNode(), use_gpu);

    // Contribute to Main to end the program
    ...
  }
}
```

Listing 5: `vecadd_charm.C`

Vector Addition Example: RAJA

```
void vecadd(const uint64_t n, int process, bool use_gpu) {
  double *h_a, *d_a, *d_b;

  cudaErrchk(cudaMallocHost((void**)&h_a, n * sizeof(double)));
  cudaErrchk(cudaMalloc((void**)&d_a, n * sizeof(double)));
  cudaErrchk(cudaMalloc((void**)&d_b, n * sizeof(double)));

  RAJA::forall<RAJA::cuda_exec<256>>(RAJA::RangeSegment(0, n),
    [=] RAJA_DEVICE (int i) {
      d_a[i] += d_b[i];
  });

  cudaErrchk(cudaMemcpy(h_a, d_a, n * sizeof(double), cudaMemcpyDeviceToHost));
}
```

Listing 6: `vecadd_raja.cpp`

Kokkos vs. RAJA

- Both allow **C++ functors** and **lambdas** for computation kernels
- Kokkos needs initialize and finalize calls
- Kokkos provides the View abstraction for memory management
- Explicit memory management in RAJA
- No performance difference in vector addition

Future Work

- What if we want more than one Kokkos/RAJA instance per node?
  - In NUMA environments, etc.
  - Should be able to pin Charm++ processes to a set of cores
- A more involved integration with Charm++ scheduler
- Other shared memory parallel frameworks: StarPU, OmpSS
- Performance comparison with standardized set of benchmarks

Thank You