2020 ALCF COMPUTATIONAL PERFORMANCE WORKSHOP

Argonne NATIONAL LABORATORY

KOKKOS / RAJA BREAKOUT SESSION

BRIAN HOMERDING
Argonne National Laboratory
Speaker

May 5th, 2020

KOKKOS

KOKKOS

```
> git clone https://github.com/kokkos/kokkos.git
> git clone https://github.com/kokkos/kokkos-tutorials.git
> cd kokkos-tutorials/Intro-Full/Exercises/01/Begin/
```

KOKKOS

exercise_1_begin.cpp

```
51 #include <sys/time.h>
52
53 // EXERCISE: Include Kokkos_Core.hpp.
54 //           cmath library unnecessary after.
55 // #include <Kokkos_Core.hpp>
56 #include <cmath>
57
58 void checkSizes( int &N, int &M, int &S, int &nrepeat );
```

KOKKOS

exercise_1_begin.cpp

```
 95   // Check sizes.
 96   checkSizes( N, M, S, nrepeat );
 97
 98   // EXERCISE: Initialize Kokkos runtime.
 99   //           Include braces to encapsulate code between initialize and finalize calls
100   // Kokkos::initialize( argc, argv );
101   // {
102
103   // Allocate y, x vectors and Matrix A:
104   double * const y = new double[ N ];
```

KOKKOS

exercise_1_begin.cpp

```
108   // Initialize y vector.
109   // EXERCISE: Convert outer loop to Kokkos::parallel_for.
110   for ( int i = 0; i < N; ++i ) {
111     y[ i ] = 1;
112   }
113
114   // Initialize x vector.
115   // EXERCISE: Convert outer loop to Kokkos::parallel_for.
116   for ( int i = 0; i < M; ++i ) {
117     x[ i ] = 1;
118   }
119
120   // Initialize A matrix, note 2D indexing computation.
121   // EXERCISE: Convert outer loop to Kokkos::parallel_for.
122   for ( int j = 0; j < N; ++j ) {
123     for ( int i = 0; i < M; ++i ) {
124       A[ j * M + i ] = 1;
```

KOKKOS

exercise_1_begin.cpp

```
108   // Initialize y vector.
109   Kokkos::parallel_for( "y_init", N, KOKKOS_LAMBDA ( int i ) {
110     y[ i ] = 1;
111   });
112
113   // Initialize x vector.
114   Kokkos::parallel_for( "x_init", M, KOKKOS_LAMBDA ( int i ) {
115     x[ i ] = 1;
116   });
117
118   // Initialize A matrix, note 2D indexing computation.
119   Kokkos::parallel_for( "matrix_init", N, KOKKOS_LAMBDA ( int j ) {
120     for ( int i = 0; i < M; ++i ) {
121       A[ j * M + i ] = 1;
122     }
123   });
```

KOKKOS

exercise_1_begin.cpp

```
138     // EXERCISE: Convert outer loop to Kokkos::parallel_reduce.
139     for ( int j = 0; j < N; ++j ) {
140       double temp2 = 0;
141
142       for ( int i = 0; i < M; ++i ) {
143         temp2 += A[ j * M + i ] * x[ i ];
144       }
145
146       result += y[ j ] * temp2;
147     }
148
```

KOKKOS

exercise_1_begin.cpp

```
138     Kokkos::parallel_reduce( "yAx", N, KOKKOS_LAMBDA ( int j, double &update ) {
139       double temp2 = 0;
140
141       for ( int i = 0; i < M; ++i ) {
142         temp2 += A[ j * M + i ] * x[ i ];
143       }
144
145       update += y[ j ] * temp2;
146     }, result );
```

KOKKOS

exercise_1_begin.cpp

```
182
183   // EXERCISE: finalize Kokkos runtime
184   // }
185   // Kokkos::finalize();
186
187   return 0;
188 }
```

KOKKOS

Build and Environment

THETA

```
> module swap          \
  PrgEnv-intel/6.0.5 \
  PrgEnv-gnu


> make KOKKOS_PATH=/path/to/kokkos \
       KOKKOS_DEVICES=”OpenMP"     \
       KOKKOS_ARCH=”KNL"
```

COOLEY

```
$ soft add +cuda-10.0
$ soft add +gcc-7.1.0



$ make KOKKOS_PATH=/path/to/kokkos \
       KOKKOS_DEVICES="Cuda"       \
       KOKKOS_ARCH="Kepler37"
```

RAJA

RAJA

```
> git clone --recursive https://github.com/llnl/raja.git
> cd raja/exercises/tutorial_halfday
```

RAJA

ex1_vector-addition.cpp

```
105   /// EXERCISE: Implement the vector addition kernel using a RAJA::forall
106   ///           method and RAJA::seq_exec execution policy type.
107   ///
108   /// NOTE: We've done this one for you to help you get started...
109   ///
110
111   using EXEC_POL1 = RAJA::seq_exec;
112
113   RAJA::forall< EXEC_POL1 >(RAJA::RangeSegment(0, N), [=] (int i) {
114     c[i] = a[i] + b[i];
115   });
116
117   checkResult(c, c_ref, N);
```

RAJA

ex1_vector-addition.cpp

```
128   std::cout << "\n Running RAJA SIMD vector addition...\n";
129
130   ///
131   /// TODO...
132   ///
133   /// EXERCISE: Implement the vector addition kernel using a RAJA::forall
134   ///           method and RAJA::simd_exec execution policy type.
135   ///
136
137   checkResult(c, c_ref, N);
```

RAJA

ex1_vector-addition.cpp

```
128   std::cout << "\n Running RAJA SIMD vector addition...\n";
129
130   using EXEC_POL2 = RAJA::simd_exec;
131
132   RAJA::forall< EXEC_POL2 >(RAJA::RangeSegment(0, N), [=] (int i) {
133     c[i] = a[i] + b[i];
134   });
135
136   checkResult(c, c_ref, N);
```

RAJA

ex1_vector-addition.cpp

```
191   std::cout << "\n Running RAJA OpenMP multithreaded vector addition...\n";
192
193   ///
194   /// TODO...
195   ///
196   /// EXERCISE: Implement the vector addition kernel using a RAJA::forall
197   ///           method and RAJA::omp_parallel_for_exec execution policy type.
198   ///
199
200   checkResult(c, c_ref, N);
```

RAJA

ex1_vector-addition.cpp

```
191   std::cout << "\n Running RAJA OpenMP multithreaded vector addition...\n";
192
193   using EXEC_POL4 = RAJA::omp_parallel_for_exec;
194
195   RAJA::forall< EXEC_POL4 >(RAJA::RangeSegment(0, N), [=] (int i) {
196     c[i] = a[i] + b[i];
197   });
198
199   checkResult(c, c_ref, N);
```

RAJA

ex1_vector-addition.cpp

```
213   std::cout << "\n Running RAJA CUDA vector addition...\n";
214
215   ///
216   /// TODO...
217   ///
218   /// EXERCISE: Implement the vector addition kernel using a RAJA::forall
219   ///           method and RAJA::cuda_exec execution policy type.
220   ///
221
222   checkResult(c, c_ref, N);
```

RAJA

ex1_vector-addition.cpp

```
 43 #if defined(RAJA_ENABLE_CUDA)
 44 const int CUDA_BLOCK_SIZE = 256;
 45 #endif
{…}
213   std::cout << "\n Running RAJA CUDA vector addition...\n";
214
215   using EXEC_POL5 = RAJA::cuda_exec<CUDA_BLOCK_SIZE>;
216
217   RAJA::forall< EXEC_POL5 >(RAJA::RangeSegment(0, N),
218                             [=] RAJA_DEVICE (int i) {
219     c[i] = a[i] + b[i];
220   });
221
222   checkResult(c, c_ref, N);
```

RAJA

Build and Environment

THETA

```
> module swap      \
  intel/19.0.5.281 \
  intel/18.0.0.128

> cd /path/to/raja
# add –DENABLE_TESTS=Off to
# scripts/alcf-builds/theta_intel18.sh
> ./scripts/alcf-builds/theta_intel18.sh
> cd build_alcf-theta-intel18.0
> make
```

COOLEY

```
$ soft add +cmake-3.9.1
$ soft add +clang-4.0
$ soft add +cuda-9.1

$ ./scripts/alcf-builds/
                cooley_nvcc9.1_clang4.0.sh
$ cd build_cooley-nvcc9.1_clang4.0
$ make
```

THANK YOU