Implementing Differentiation of the Kokkos Framework in Clad

GSoC 2024 project by Atell Krasnopolski
Mentors: Vaibhav Thakkar, Vassil Vassilev, Petro Zarytskyi

What is Kokkos

The Kokkos C++ Performance Portability Ecosystem is a production level solution for writing modern C++ applications in a hardware-agnostic way. It is part of the US Department of Energies Exascale Project – the leading effort in the US to prepare the HPC community for the next generation of supercomputing platforms

kokkos

Kokkos applications

The Kokkos framework is used in several domains including climate modelling where gradients are an important part of the simulation process. This project aims at teaching Clad to differentiate Kokkos entities in a performance-portable way.

AppName	Area	Institution	Website	Status	Uses KokkosKernels	Contact Name	Contact Email
Albany	Climate	Sandia	https://github.com/sandialabs/Albany	Porting/Production	Y	Mauro Perego, Irina Tezaur	mperego@sandia.gov, ikalash@sandia.gov
LGR	Shock Hydrodynamics	Sandia	https://github.com/SNLComputation/lgrtk	Production	Y	Dan Ibanez	daibane@sandia.gov
Aria	Thermal Fluid Multi Physics	Sandia		Porting	Y	Jonathan Clausen	jclause@sandia.gov
LAMMPS	Molecular Dynamics	Sandia	https://github.com/lammps/lammps	Production	N	Stan Moore	–
Trilinos-Tpetra	Distributed Sparse Linear Algebra Package	Sandia	https://github.com/trilinos/trilinos	Production	Y	Karen Devine	–
Trilinos-Phalanx	DAG-based Assembly	Sandia		Production	Y	Roger Pawlowski	rppawlo@sandia.gov
Trilinos-Panzer	Finite Element Tools	Sandia		Production/Porting	Y	Roger Pawlowski	rppawlo@sandia.gov

Kokkos basics

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

int main(int argc, char* argv[]) {
    Kokkos::initialize(argc,argv);

    int N = atoi(argv[1]);

    Kokkos::parallel_for("Loop1", N, KOKKOS_LAMBDA (const int i) {
        printf("Greeting from iteration %i\n",i);
    });

    Kokkos::finalize();
}
```

Goals (desired behaviour) Progress

The goal is to implement the differentiation of the Kokkos high-performance computing framework including the support of:

- Kokkos functors, forward mode
- Kokkos lambdas, very limited lambda support in both modes
- Kokkos methods such as parallel_for, parallel_reduce and deep_copy, as well as the general support for Kokkos::View data structures,

needs to be merged

almost done (forward mode)

basically done in the forward mode

- Enhance existing benchmarks demonstrating effectiveness of Clad for Kokkos

hasn't been touched yet

Progress

gojakuch
18 commits 2,146 ++ 273 --

Almost everything from the list of the first coding period deliverables in the project proposal has been done successfully.

Other things I've made in the meantime

- std::arrays (fwd)
- "Introduction to Clang for Clad contributors" with Christina
- null statement support (both modes)
- operators with side-effects in ifs and fors
- primitive lambda support (both modes)
- string support (fwd)
- as well as opened & fixed other issues related to the main project

Example 1

```
double f_basics_deep_copy_2(double x, double y) {
  const int N = 2;

  Kokkos::View<double*, Kokkos::LayoutLeft, Kokkos::HostSpace> a("a", N);
  Kokkos::View<double*, Kokkos::LayoutLeft, Kokkos::HostSpace> b("b", N);

  Kokkos::deep_copy(a, 3 * y + x + 50);
  b(1) = x * y;
  Kokkos::deep_copy(b, a);

  b(1) = b(1) + a(0) * b(1);

  a(1) = x * x * x;
  a(0) += a(1);

  return a(0); // derivative of this wrt y is constantly 3
}
```

Example 2

```
double f_basics_resize_2(double x, double y) {
  Kokkos::View<double** [3], Kokkos::LayoutLeft, Kokkos::HostSpace> a("a", 3,
                                                                      2);

  Kokkos::deep_copy(a, 3 * x + y);
  a(2, 1, 0) = x * y;

  Kokkos::resize(a, 5, 5);

  return a(2, 1, 0);
}
```

```
template <typename View> struct Foo {
  View& res;
  double& x;

  Foo(View& _res, double& _x) : res(_res), x(_x) {}

  KOKKOS_INLINE_FUNCTION
  void operator()(const int i) const { res(i) = x * i; }
};
double parallel_for_functor_simplest_case_rangepol(double x) {
  Kokkos::View<double[5], Kokkos::HostSpace> res("res");

  Foo<Kokkos::View<double[5], Kokkos::HostSpace>> f(res, x);

  f(0);

  Kokkos::parallel_for(
      "polynomial",
      Kokkos::RangePolicy<Kokkos::DefaultHostExecutionSpace>(1, 5), f);

  return res(3);
}
```

Generic approach

```
namespace class_functions {
/// Kokkos arrays
template <class DataType, class... ViewParams>
clad::ValueAndPushforward<Kokkos::View<DataType, ViewParams...>,
                          Kokkos::View<DataType, ViewParams...>>
constructor_pushforward(
    clad::ConstructorPushforwardTag<Kokkos::View<DataType, ViewParams...>>,
    const ::std::string& name, const size_t& idx0, const size_t& idx1,
    const size_t& idx2, const size_t& idx3, const size_t& idx4,
    const size_t& idx5, const size_t& idx6, const size_t& idx7,
    const ::std::string& d_name, const size_t& d_idx0, const size_t& d_idx1,
    const size_t& d_idx2, const size_t& d_idx3, const size_t& d_idx4,
    const size_t& d_idx5, const size_t& d_idx6, const size_t& d_idx7) {
  return {Kokkos::View<DataType, ViewParams...>(name, idx0, idx1, idx2, idx3,
                                                idx4, idx5, idx6, idx7),
          Kokkos::View<DataType, ViewParams...>(
              "_diff_" + name, idx0, idx1, idx2, idx3, idx4, idx5, idx6, idx7)};
}
} // namespace class_functions
```

Generic approach

```
template <class PolicyP, class ... PolicyParams,
          class FunctorType> // multi-dimensional policy
void parallel_for_pushforward(
    const ::std::string& str,
    const ::Kokkos::MDRangePolicy<PolicyP, PolicyParams ...>& policy,
    const FunctorType& functor, const ::std::string& /*d_str*/,
    const ::Kokkos::MDRangePolicy<PolicyP, PolicyParams ...>& /*d_policy*/,
    const FunctorType& d_functor) {
  ::Kokkos::parallel_for(str, policy, functor);
  diff_parallel_for_MDP_call_dispatch<
      ::Kokkos::MDRangePolicy<PolicyP, PolicyParams ...>, FunctorType,
      typename ::Kokkos::MDRangePolicy<PolicyP, PolicyParams ...>::work_tag,
      ::Kokkos::MDRangePolicy<PolicyP, PolicyParams ...>::rank>::run(str, policy,
                                                                     functor,
                                                                     d_functor);
}
```

Generic approach

```
95  template <class Policy, class FunctorType, class T>
96  struct diff_parallel_for_MDP_call_dispatch<Policy, FunctorType, T, 2> {
97    static void run(const ::std::string& str, const Policy& policy,
98                    const FunctorType& functor, const FunctorType& d_functor) {
99      ::Kokkos::parallel_for("_diff_" + str, policy,
100                             [&functor, &d_functor](const T x, auto&&... args) {
101                               functor.operator_call_pushforward(
102                                   x, args..., &d_functor, &x, 0, 0);
103                             });
104   }
105 };
106 template <class Policy, class FunctorType>
107 struct diff_parallel_for_MDP_call_dispatch<Policy, FunctorType, void, 2> {
108   static void run(const ::std::string& str, const Policy& policy,
109                   const FunctorType& functor, const FunctorType& d_functor) {
110     ::Kokkos::parallel_for(
111         "_diff_" + str, policy, [&functor, &d_functor](auto&&... args) {
112           functor.operator_call_pushforward(args..., &d_functor, 0, 0);
113         });
114   }
115 };
```

Next steps

- Finish the forward mode goals
- Start working on the reverse mode implementations and fix potential blocker issues, as this is where Clad can shine for Kokkos
- Develop extensive lambda support and continue working on other issues assigned to me in the meantime
- Possibly try to make the produced code more readable or easily copyable, if there's some time left

kokkos