



Automatic Differentiation of the Kokkos framework and the STL with Clad

Atell Krasnopolski, Petro Zarytskyi, Dr Vassil Vassilev
This work is partially supported by National Science Foundation under Grant OAC-2311471

Source

```
double fn(double x, double y) {  
    double t = 2*x + y*y;  
    return t;  
}
```



Clang



Compiled

Derivatives

```
void fn_grad(double x, double y,  
double *_d_x, double *_d_y) {  
    double _d_t = 0.;  
    double t = 2 * x + y * y;  
    _d_t += 1;  
    {  
        *_d_x += 2 * _d_t;  
        *_d_y += _d_t * y;  
        *_d_y += y * _d_t;  
    }  
}
```

Clad



Making an efficient language efficiently differentiable

This allows template usage!

Making STL differentiable

(C++ Standard Template Library)

```

double fnVec6(double x, double y) {
    std::vector<double> v(3, y);

    v.pop_back();
    double res = v.size()*x;

    v.erase(v.begin());
    res += v.size()*x;

    std::vector<double> w;
    w = v;
    w.clear();
    res += w.size()*x + v.size()*x;

    w.insert(w.end(), 5);
    res += w.size()*x;

    w.insert(w.end(), {y, x, y});
    w.insert(w.end(), v.begin(), v.end());

    w.assign(2, y);
    res += (w[0] == y && w[1] == y)*x;

    v[0] = x;
    w.assign(v.begin(), v.end());
    res += w[0];

    w.assign({3*x, 2*x, 4*x});
    res += w[1];

    return res;
}

```

Methods Operators Constructors

Library-specific extensions for Clad

(my job)

```
double fn(double x, double y) {
    double t = 2*x + y*y;
    return t;
}
```

Gradient (reverse mode)



```
void fn_grad(double x, double y,
double *_d_x, double *_d_y) {
    double _d_t = 0.;
    double t = 2 * x + y * y;
    _d_t += 1;
    {
        *_d_x += 2 * _d_t;
        *_d_y += _d_t * y;
        *_d_y += y * _d_t;
    }
}
```

df/dx (forward mode)



```
double fn_darg0(double x, double y) {
    double _d_x = 1;
    double _d_y = 0;
    double _d_t = 0 * x + 2 * _d_x + _d_y * y + y * _d_y;
    double t = 2 * x + y * y;
    return _d_t;
}
```



```
double fn(double x, double y) {  
    double t = 2*x + y*y;  
    return t;  
}
```

Gradient (reverse mode)

```
void fn_grad(double x, double y,  
double *_d_x, double *_d_y) {  
    double _d_t = 0.;  
    double t = 2 * x + y * y;  
    _d_t += 1;  
    {  
        *_d_x += 2 * _d_t;  
        *_d_y += _d_t * y;  
        *_d_y += y * _d_t;  
    }  
}
```

df/dx (forward mode)

```
double fn_darg0(double x, double y) {  
    double _d_x = 1;  
    double _d_y = 0;  
    double _d_t = 0 * x + 2 * _d_x + _d_y * y + y * _d_y;  
    double t = 2 * x + y * y;  
    return _d_t;  
}
```

`f(...)` calls `g(...)`
`f_darg0(...)` calls `g_pushforward(...)`

`g(...)`



`value`

`g_pushforward(...)`



`{value, d_value}`

```
double fn(double x, double y) {
    double t = 2*x + y*y;
    return t;
}
```

Gradient (reverse mode)

```
void fn_grad(double x, double y,
double *_d_x, double *_d_y) {
    double _d_t = 0.;
    double t = 2 * x + y * y;
    _d_t += 1;
    {
        *_d_x += 2 * _d_t;
        *_d_y += _d_t * y;
        *_d_y += y * _d_t;
    }
}
```

df/dx (forward mode)

```
double fn_darg0(double x, double y) {
    double _d_x = 1;
    double _d_y = 0;
    double _d_t = 0 * x + 2 * _d_x + _d_y * y + y * _d_y;
    double t = 2 * x + y * y;
    return _d_t;
}
```

```
double fn(double x, double y) {
    double t = 2*x + y*y;
    return t;
}
```

Gradient (reverse mode)



```
void fn_grad(double x, double y,
double *_d_x, double *_d_y) {
    double _d_t = 0.;
    double t = 2 * x + y * y;
    _d_t += 1;
    {
        *_d_x += 2 * _d_t;
        *_d_y += _d_t * y;
        *_d_y += y * _d_t;
    }
}
```

df/dx (forward mode)



```
double fn_darg0(double x, double y) {
    double _d_x = 1;
    double _d_y = 0;
    double _d_t = 0 * x + 2 * _d_x + _d_y * y + y * _d_y;
    double t = 2 * x + y * y;
    return _d_t;
}
```

`f(...)` calls `g(...)`

`f_grad(...)` calls `g_reverse_forw(...)`
and `g_pullback(...)`

`f(...)` calls `g(...)`

(for OOP)

`f_grad(...)` calls `g_reverse_forw(...)`
and `g_pullback(...)`

`f(...)` calls `g(...)`

`f_grad(...)` calls `g_reverse_forw(...)`
and `g_pullback(...)`

Can provide a better pullback than the generated one? Prove your own!

```
namespace clad::custom_derivatives {  
  
    void g_pullback(double v, double u,  
                    double d_y,  
                    double * d_v, double * d_u) {  
        ...  
    }  
  
}
```

```
namespace clad::custom_derivatives {  
  
    void g_pullback(double v, double u,  
                   double d_y,  
                   double * d_v, double * d_u) {  
        ...  
    }  
  
}
```

```
namespace clad::custom_derivatives {  
  
    void g_pullback(double v, double u,  
                   double d_y,  
                   double * d_v, double * d_u) {  
        ...  
    }  
  
}
```

```
namespace clad::custom_derivatives {  
  
    void g_pullback(double v, double u,  
                   double d_y,  
                   double * d_v, double * d_u) {  
        ...  
    }  
  
}
```


```
namespace clad::custom_derivatives {  
  
    void g_pullback(double v, double u,  
                    double d_y,  
                    double * d_v, double * d_u) {  
        ...  
    }  
  
}
```

Methods?
Operators?
Constructors?

```
namespace clad::custom_derivatives::class_functions {  
    ...  
}
```



```
template <typename T, ::std::size_t N, typename P>
void operator_subscript_pullback(
    ::std::array<T, N>* arr, typename ::std::array<T, N>::size_type idx, P d_y,
    ::std::array<T, N>* d_arr, typename ::std::array<T, N>::size_type* d_idx) {
    (*d_arr)[idx] += d_y;
}
```

```
template <typename T, ::std::size_t N, typename P>   
void operator_subscript_pullback(  
    ::std::array<T, N>* arr, typename ::std::array<T, N>::size_type idx, P d_y,  
    ::std::array<T, N>* d_arr, typename ::std::array<T, N>::size_type* d_idx) {  
    (*d_arr)[idx] += d_y;  
}
```

No class modification
No Clad codebase modification ↪ **Easy support**

No class modification
No Clad codebase modification \mapsto **Easy support**
(almost)



k o k k o s

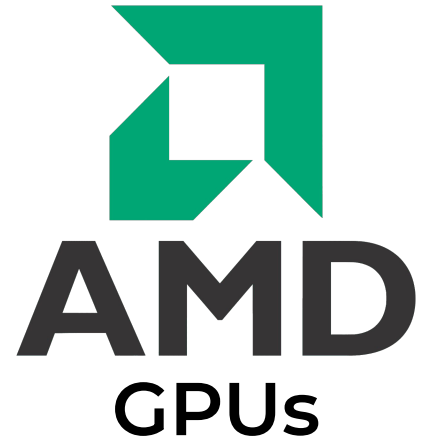
Source \mapsto Source Parallel \mapsto Parallel

```
Kokkos::parallel_for("name", Policy, f);
```




```
Kokkos::parallel_for("name", Policy, f);  
Kokkos::parallel_for("d_name", Policy, d_f);
```

Performance Portable Gradient Computations Using Source Transformation (a paper)



Kokkos-aware Clad #783

 Open kliegeois wants to merge 75 commits into vgvassilev:master from kliegeois:kokkos-PR 

 Conversation 20

 Commits 75

 Checks 80

 Files changed 13



kliegeois commented on Feb 22

First-time contributor 

This PR provides the implementation of a first set of features for the automatic generation of gradients of Kokkos-based code.

The content of this PR supports reverse mode of Kokkos `parallel_for` , `deep_copy` , and view accesses.

[@vgvassilev](#) [@brian-kelley](#)



 **Make the signatures of `KokkosBuiltins.h` more general** ✓


#1063 by gojakuch was merged last month

 **Provide pushforward methods for `Kokkos::View` indexing** ✓

#1061 by gojakuch was merged last month

 **Add support for `Kokkos::parallel_reduce` in the fwd mode** ✓


#1056 by gojakuch was merged last month

 **Add support for `Kokkos::fence` in the fwd mode** ✓


#1048 by gojakuch was merged on Aug 21 • Approved

 **Add support for `Kokkos::parallel_for` in the fwd mode** ✓


#1022 by gojakuch was merged on Aug 21

 **Add support for `Kokkos::resize` in the forward mode** ✓

#999 by gojakuch was merged on Jul 24

 **Prevent Clad from trying to create a void zero literal** ✓

#989 by gojakuch was merged on Jul 21 • Approved

 **Add basic Kokkos support for the original tests of #783** ✓

#977 by gojakuch was merged on Jul 22 • Approved

 **Fix the derivative of string literals in forward mode** ✓

#967 by gojakuch was merged on Jul 4 • Approved


 **Add Kokkos unittests** ✓

#826 by gojakuch was merged on Apr 19 • Approved

Rough edges (ongoing work)

No captures here yet.

```
double fn0(double x) {  
    auto _f = [] (double _x) {  
        return x*_x;  
    };  
    return _f(x) + 1;  
}
```



```
auto pack(double x) {  
    return std::make_tuple(x, 2*x, 3*x);  
}
```

```
double fnTuple1(double x, double y) {  
    double u, v = 288*x, w;
```

```
    std::tie(u, v, w) = pack(x+y);
```

```
    return v;
```

```
} // = 2x + 2y
```



No reverse mode support yet.

Try it out!
(we're use-case driven)