Programming Cuda and OpenCL
A Case Study Using Modern C++ Libraries

Frameworks

- Cuda
  - NVIDIA
  - Large set of libraries
  - Compute kernels compiled to PTX (low level)
- OpenCL
  - Cross platform
  - API - Boilerplate code
  - Compute kernels compiled to C-like sources (higher level)

Libraries

- (C)MTL4 (The Matrix Template Library)
  - Linear algebra library
  - DSL embedded in c++
  - High level, compile time transformations
  - Cuda
- VexCL (Vector Expression Template Library)
  - Convenient vector and matrix
  - OpenCL
  - Reduce boilerplate code
- ViennaCL (The Vienna Computing Library)
  - Linear Algebra
  - Cuda and OpenCL (only OpenCL in article)
- Thrust
  - Resembles c++ STL
  - Reference point

Ordinary differential equation

- Derivatives with respect to only one variable $\frac{\mathrm{d}x}{\mathrm{d}t} = \dot{x} = f(x,t), \qquad x(0) = x_0.$
- With PDE, surface change over time, ODE particle moving through time
- Eulers method:

$x(t_0)$ $\quad x(t + \Delta t)$

$$x\,(t + \Delta t) = x(t) + \Delta t\ f(x(t), t).$$

Odeint

- C++ library for solving ODE's numerically $\frac{\mathrm{d}x}{\mathrm{d}t} = \dot{x} = f(x,t), \qquad x(0) = x_0.$
- Use odeint solving cababilities with gpgpu libraries
- State type, algebra, operation.

$$\dot{x} = -\sigma\,(x-y)\,, \quad \dot{y} = Rx - y - xz, \quad \dot{z} = -bz + xy.$$

Odeint – Stepper (runge-kutta)

```
template<typename State, typename Value = double, typename Deriv = State,
         typename Time = Value,
         typename Algebra = typename algebra_dispatcher< State >::algebra_type,
         typename Operations = typename operations_dispatcher< State >::operations_type,
         typename Resizer = initially_resizer>
```

1. `typename State`

   The state type.

2. `typename Value = double`

   The value type.

3. `typename Deriv = State`

   The type representing the time derivative of the state.

4. `typename Time = Value`

   The time representing the independent variable - the time.

5. `typename Algebra = typename algebra_dispatcher< State >::algebra_type`

   The algebra type.

6. `typename Operations = typename operations_dispatcher< State >::operations_type`

   The operations type.

7. `typename Resizer = initially_resizer`

   The resizer policy type.

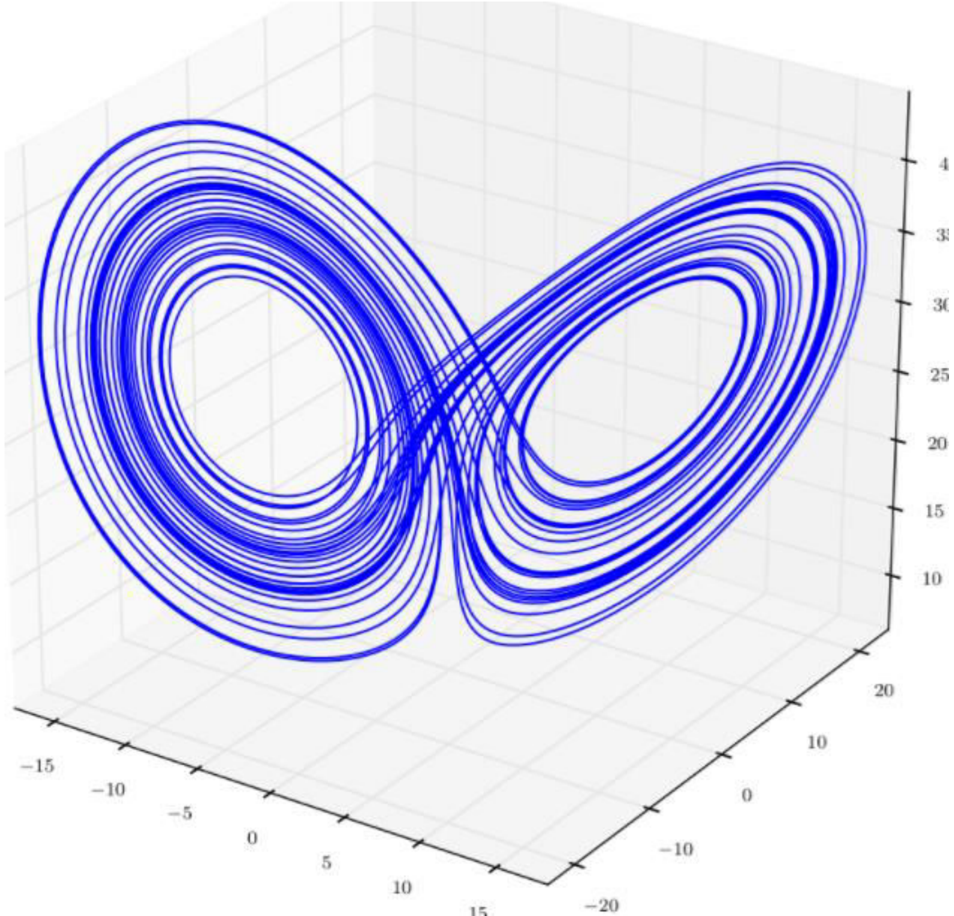
Odeint - integrate

```
size_t integrate_const(Stepper stepper, System system, State & start_state,
                       Time start_time, Time end_time, Time dt,
                       Observer observer);
```

```
integrate_const( stepper_type() , lorenz , x , value_type(0.0) , t_max , dt );
```

Lorenz system

$$\dot{x} = -\sigma\left(x - y\right), \quad \dot{y} = Rx - y - xz, \quad \dot{z} = -bz + xy.$$



Lorenz - Thrust

```
typedef thrust::device_vector<double> state_type;

struct lorenz_system {
    size_t N;
    const state_type &R;
    lorenz_system( size_t n, const state_type &r) : N(n), R(r) { }
    void operator()(const state_type &x, state_type &dxdt, double t) const;
};
```

```
void lorenz_system::operator()(const state_type &x, state_type &dxdt,
                               double t) const
{
    thrust :: for_each(
        thrust :: make_zip_iterator(  thrust :: make_tuple(
            R.begin(),
            x.begin(),  x.begin() + N, x.begin() + 2 * N,
            dxdt.begin(),  dxdt.begin() + N, dxdt.begin() + 2 * N ) ),
        thrust :: make_zip_iterator(  thrust :: make_tuple(
            R.end(),
            x.begin() + N, x.begin() + 2 * N, x.end(),
            dxdt.begin() + N, dxdt.begin() + 2 * N, dxdt.end() ) ),
        lorenz_functor()  );
}
```

```
struct lorenz_functor {
    template< class T >
    __host__  __device__  void operator()( T t ) const {
        double R = thrust::get<0>(t);
        double x = thrust::get<1>(t);
        double y = thrust::get<2>(t);
        double z = thrust::get<3>(t);
        thrust :: get<4>(t) = sigma * ( y - x );
        thrust :: get<5>(t) = R * x - y - x * z;
        thrust :: get<6>(t) = -b * z + x * y ;
    }
};
```

Lorenz - CMTL4

```
typedef mtl::dense_vector<double>    vector_type;
typedef mtl::multi_vector<vector_type> state_type;

struct lorenz_system {
    const vector_type &R;
    explicit lorenz_system(const vector_type &R) : R(R) { }

    void operator()(const state_type& x, state_type& dxdt, double t) {
        dxdt.at(0) = sigma * (x.at(1) - x.at(0));
        dxdt.at(1) = R * x.at(0) - x.at(1) - x.at(0) * x.at(2);
        dxdt.at(2) = x.at(0) * x.at(1) - b * x.at(2);
    }
};
```

```
( lazy(dxdt.at(0)) = sigma * (x.at(1) - x.at(0)) ) ||
( lazy(dxdt.at(1)) = R * x.at(0) - x.at(1) - x.at(0) * x.at(2) ) ||
( lazy(dxdt.at(2)) = x.at(0) * x.at(1) - b * x.at(2) );
```

150 % overhead with a 3-component vector with 4K entries compared to one vector of size 12K

Lorenz - VexCL

```
typedef vex::multivector<double, 3> state_type;

struct lorenz_system {
    const vex::vector<double> &R;
    lorenz_system(const vex::vector<double> &r) : R(r) {}

    void operator()(const state_type &x, state_type &dxdt, double t) const {
        dxdt(0) = sigma * (x(1) − x(0));
        dxdt(1) = R * x(0) − x(1) − x(0) * x(2);
        dxdt(2) = x(0) * x(1) − b * x(2);
    }
};
```

```
dxdt = std::tie(    sigma * (x(1) − x(0)),
                    R * x(0) − x(1) − x(0) * x(2),
                    x(0) * x(1) − b * x(2)          );
```

1 Kernel call instead of 3 -> 25% performance gain (Large systems)

ViennaCL

```
typedef fusion::vector<
    viennacl :: vector<double>, viennacl::vector<double>, viennacl::vector<double>
    > state_type;

struct lorenz_system {
    const viennacl::vector<double> &R;
    lorenz_system(const viennacl::vector<double> &r) : R(r) {}

    void operator()(const state_type &x, state_type &dxdt, double t) const {
        typedef viennacl::generator :: vector<value_type> vec;

        const auto &X = fusion::at_c<0>(x);
        const auto &Y = fusion::at_c<1>(x);
        const auto &Z = fusion::at_c<2>(x);

        auto &dX = fusion::at_c<0>(dxdt);
        auto &dY = fusion::at_c<1>(dxdt);
        auto &dZ = fusion::at_c<2>(dxdt);

        viennacl :: generator :: custom_operation op;
        op.add( vec(dX) = sigma * (vec(Y) - vec(X)) );
        op.add( vec(dY) = element_prod(vec(R), vec(X)) - vec(Y)
                          - element_prod(vec(X), vec(Z)) );
        op.add( vec(dZ) = element_prod(vec(X), vec(Y)) - b * vec(Z) );
        op.excecute()
    }
};
```

Kernel is created once and buffered

Results

	Lorenz attractor	
	Time	T-put
		NVIDIA
Thrust	242.78	105 (71%)
CMTL4	237.91	108 (73%)
VexCL	246.58	104 (70%)
ViennaCL	259.85	99 (66%)
		AMD Radeon
VexCL	149.49	171 (65%)
ViennaCL	148.69	172 (65%)
		Intel (
Thrust	2 336.14	11 (43%)
VexCL (AMD)	2 329.00	11 (43%)
VexCL (Intel)	2 372.70	11 (42%)
ViennaCL (AMD)	2 322.78	11 (43%)
ViennaCL (Intel)	2 322.39	11 (43%)