Solving ODEs with CUDA and OpenCL

Using Boost.Odeint

Karsten Ahnert[1,2]

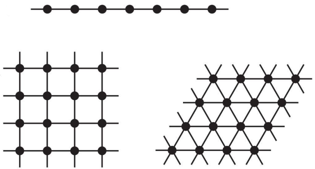
Mario Mulansky[2], Denis Demidov[3], Karl Rupp[4], and Peter Gottschling[5]

[1] Ambrosys GmbH, Potsdam [2] Institut für Physik und Astronomie, Universität Potsdam
[3] Kazan Branch of Joint Supercomputer Center, Russian Academy of Sciences, Kazan
[4] Mathematics and Computer Science Division, Argonne National Laboratory
[5] SimuNova, Dresden & Inst. Scientific Computing, TU Dresden

Februar 2, 2013

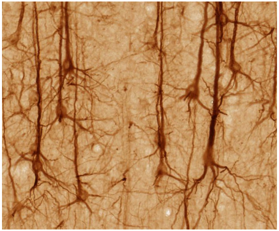
Motivation – Solve large systems of ODEs

Lattice systems

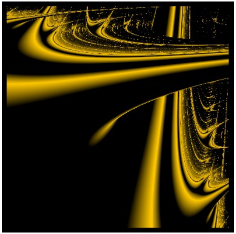


Discretiztations of PDEs

ODEs on graphs



Parameter studies



Numerical integration of ODEs

Find a numerical solution of an ODE and its IVP

$$\dot{x} = f(x,t) \; , \qquad x(t=0) = x_0$$

Example: Explicit Euler

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

```
typedef array< double , 2 > state_type;

struct ode {
  void operator()( const state_type &x,
    state_type &dxdt, double t) const {
  ..
  }
};

euler< state_type > stepper;
stepper.do_step( ode , x , t , dt );
```

Algebras and operations

Euler method

$$\text{for all i :} \qquad x_i(t + \Delta t) = x_i(t) + \Delta t \cdot f_i(x)$$

```
typedef euler< state_type ,
  value_type , deriv_type , time_type,
  algebra , operations , resizer > stepper;
```

- Algebras perform the iteration over *i*.
- Operations perform the elementary addition.

Algebras and operations

Algebra has to have defined the following member functions:

- `algebra.for_each1( x1 , unary_operation );`
- `algebra.for_each2( x1, x2, binary_operation );`
- ...

`Operations` is a class with the following (static) functors:

- `scale_sum1` // *calculates* $y = a1 * x1$
- `scale_sum2` // *calculates* $y = a1 * x1 + a2 * x2$
- ...

```
algebra.for_each3( x1 , x0 , F1 ,
   Operations::scale_sum2( 1.0, b1*dt );
```

This computes: $\vec{x}_1 = 1.0 \cdot \vec{x}_0 + b_1 \Delta t \cdot \vec{F}_1$.

Algebra and operations

- `range_algebra` – Default algebra, supporting Boost.Range
- `default_operations` – Default operations
- `vector_space_algebra` – Types with vector space semantic, i.e. `y = a1*x1 + a2*x2`. Can be used by all types supporting expression templates.
- `thrust_algebra`, `thrust_operations` – Thrust's device vectors

GPU Frameworks

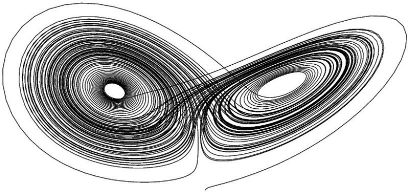
- **VexCL** - Vector Expression Framework
  - Sparse matrix support, expression templates
  - `github.com/ddemidov/vexcl`
- **ViennaCL** - Linear algebra framework
  - Not restricted to OpenCL
  - `sourceforge.net/projects/viennacl`
- **Thrust** - general purpose algorithm library
  - Mimicks the STL interface for CUDA devices
  - No expression templates, heavy use of iterators
  - Is shipped with CUDA
  - `thrust.github.com`
- **MTL4** - CUDA version of the Matrix template libary
  - Expression templates
  - `www.simunova.com/gpu_mtl4`

Example - Parameter study of Lorenz system

$$\begin{aligned}\dot{x} &= -\sigma(y - x)\\ \dot{y} &= Rx - y - xz\\ \dot{z} &= -bz + xy\end{aligned}$$

Dependence of chaoticity on parameter R

Solve many ODEs in parallel x_i, y_i, z_i, R_i



VexCL

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

struct sys_func
{
  const vector_type &R;
  sys_func( const vector_type &_R ) : R( _R ) { }
  void operator()(
    const state_type &x, state_type &dxdt, double t)
  {
    dxdt = std::tie( sigma * (x(1) - x(0)) ,
                     R * x(0) - x(1) - x(0) * x(2),
                     x(0) * x(1) - b * x(2) );
  }
};

odeint::runge_kutta4<
  state_type , double , state_type , double ,
  odeint::vector_space_algebra , odeint::default_operations
  > stepper;

odeint::integrate_const( stepper , sys_func( R ) ,
  X , t_start , t_max , dt );
```

ViennaCL

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

struct sys_func { ... };   // Details come soon

odeint::runge_kutta4<
  state_type , double , state_type , double ,
  odeint::fusion_algebra , odeint::viennacl_operations
> stepper;

odeint::integrate_const( stepper , sys_func( R ) ,
  X , t_start , t_max , dt );
```

ViennaCL

```
struct sys_func {
  const vector_type &R;
  sys_func( const vector_tyoe &_R ) : R( _R ) { }

  void operator()( const state_type &x , state_type &dxdt , double t ) const {
    using namespace viennacl::generator;

    static symbolic_vector<0,double> sym_dX;   // same for sym_dY, sym_dZ
    ...
    static symbolic_vector<3,double> sym_X;    // same for sym_Y, sym_Z
    ...
    static symbolic_vector<6,double> sym_R;
    static cpu_symbolic_scalar<7,double> sym_sigma;
    static cpu_symbolic_scalar<8,double> sym_b;

    static custom_operation lorenz_op(
      sym_dX = sym_sigma * (sym_Y - sym_X),
      sym_dY = element_prod(sym_R, sym_X) - sym_Y - element_prod(sym_X, sym_Z),
      sym_dZ = element_prod(sym_X, sym_Y) - sym_b * sym_Z,
      "lorenz");

    // unpack fusion vectors x, dxdt
    const auto &X = fusion::at_c< 0 >( x );    // same for Y,Z;
    ...
    auto &dX = fusion::at_c<0>( dxdt );        // same for dY,dZ
    ...

    viennacl::ocl::enqueue(lorenz_op(dX, dY, dZ, X, Y, Z, R, sigma, b));
  }
};
```

Thrust

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

struct sys_func { ... };    // Details come soon

typedef runge_kutta4<
  state_type , double , state_type , double ,
  thrust_algebra , thrust_operations > stepper_type;


integrate_const( stepper_type() , sys_func( R ) ,
  X , double(0.0) , t_max , dt );
```

Thrust

```
struct sys_func
{
  struct lorenz_functor { ... }   // Details come soon

  sys_func( const state_type &R ) : m_N( R.size() ) , m_R( R ) { }

  template< class State , class Deriv >
  void operator()(  const State &x , Deriv &dxdt , double t ) const
  {
    thrust::for_each(
      thrust::make_zip_iterator( thrust::make_tuple(
        boost::begin( x ) ,
        boost::begin( x ) + m_N ,
        boost::begin( x ) + 2 * m_N ,
        m_R.begin() ,
        boost::begin( dxdt ) ,
        boost::begin( dxdt ) + m_N ,
        boost::begin( dxdt ) + 2 * m_N  ) ) ,
      thrust::make_zip_iterator( thrust::make_tuple(
        boost::begin( x ) + m_N ,
        boost::begin( x ) + 2 * m_N ,
        boost::begin( x ) + 3 * m_N ,
        m_R.begin() ,
        boost::begin( dxdt ) + m_N ,
        boost::begin( dxdt ) + 2 * m_N ,
        boost::begin( dxdt ) + 3 * m_N  ) ) ,
      lorenz_functor() );
  }

  size_t m_N;
  const state_type &m_R;
};
```

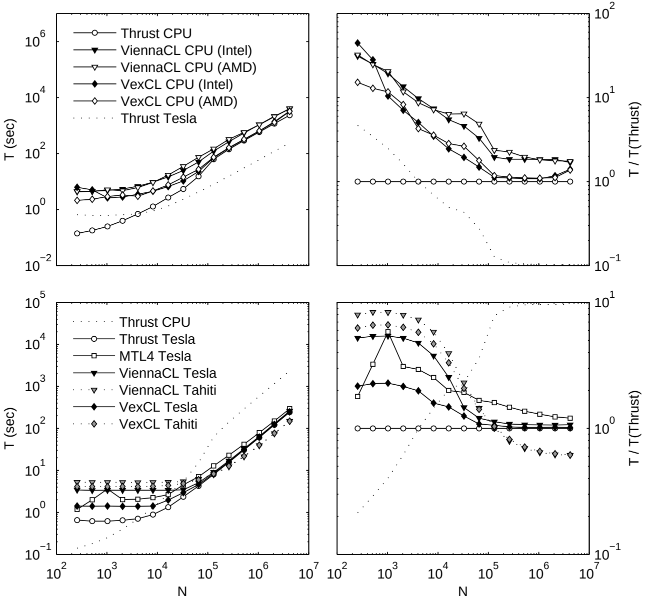
Thrust

```
struct sys_func
{
  struct lorenz_functor
  {
    template< class T >
    __host__ __device__
    void operator()( T t ) const
    {
      double R = thrust::get< 3 >( t );
      double x = thrust::get< 0 >( t );
      double y = thrust::get< 1 >( t );
      double z = thrust::get< 2 >( 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 ;
    }
  };

  ...
};
```

CUDA MTL4

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

struct sys_func
{
  explicit sys_func(const vector_type &R) : R(R) { }

  void operator()(const state_type& x, state_type& dxdt, double)
  {
    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);
  }

  const vector_type &R;
};

odeint::runge_kutta4<state_type, double, state_type, double,
  odeint::vector_space_algebra , odeint::default_operations> stepper;
odeint::integrate_const(stepper, sys_func( R ), X, 0.0, t_max, dt);
```

Performance



Conclusion

- The GPU libraries differ by usability
  - Expression templates simplify code and make them more expressive
- Performance is more or less equal
  - OpenCL has an overhead for generating the kernels during runtime
- Optimize by hand

Programming CUDA and OpenCL: A Case Study Using Modern C++ Libraries. Denis Demidov, Karsten Ahnert, Karl Rupp, Peter Gottschling. arXiv:1212.6326.

All code is available from

`github.com/ddemidov/gpgpu_with_modern_cpp`