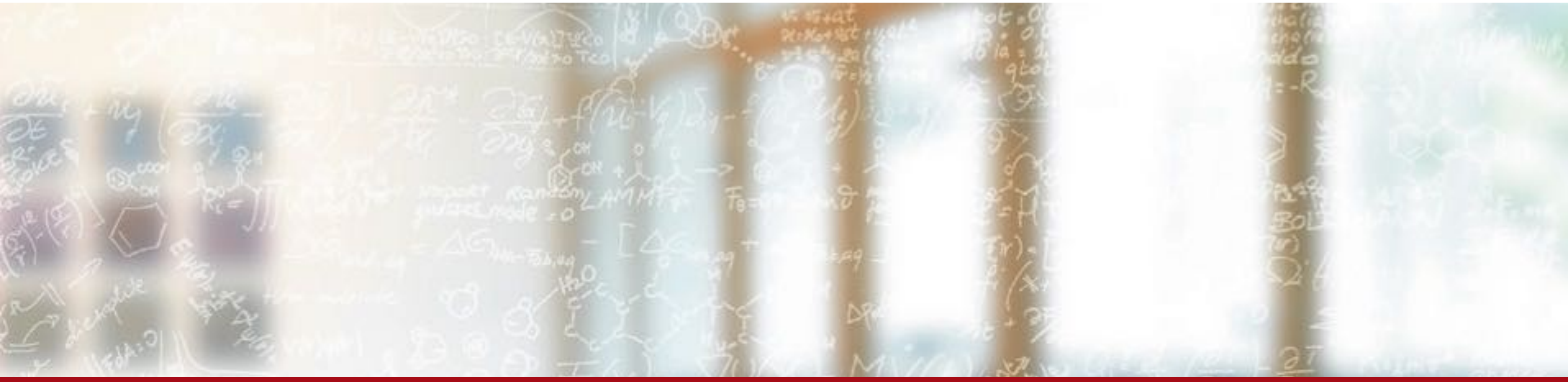
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Stencil-Miniapp hands-on exercise: Can it use a library?

Will Sawyer, Swiss National Supercomputing Centre (CSCS)

CSCS/USI Summer School
July 25, Riva San Vitale, Switzerland

Which library fits best?

- Sparse matrices: CUSPARSE handles these
  - ➡ but in this case matrix is not explicitly constructed
  - ➡ simple pentadiagonal matrix programmed as operator Ax
- CG method: CUSP, ViennaCL, MAGMA could cover that
  - ➡ but most of the complexity is in BLAS-1 operators + diffusion!
  - ➡ these libraries mostly support explicit creation of matrix
- CUBLAS for BLAS-1 operations?
  - ➡ Minimal benefit, not appropriate for diffusion
- BLAS-1 and diffusion can be formulated with iterators
  - ➡ Thrust a possibility
- PETSc and Trilinos would presumably work… next year?
- For diffusion operator: "stencil" library would be very useful

Thrust recap

- Standard template library (STL-like) support for devices
- Offers host/device vectors:
  - ➡ Data::Field become thrust::host/device_vector
- support iterators: X.begin(), X.end()
- Modest number of features
  - ➡ various iterators
  - ➡ transform, for_each to operate on functors
  - ➡ zip operators to generalize to tuples
- *Conceptually: Thrust is a neat wrapper around CUDA, removing "Chevron" (<<<>>>) syntax and allowing iteration over vectors*

Thrust linear algebra

```
double dot_thrust(thrust::device_vector<double>& X, thrust::device_vector<double>& Y)
{
    // returns X^T Y
    return thrust::inner_product(X.begin(), X.end(), Y.begin(), 0.0);
}

void scaled_diff_thrust(double A, thrust::device_vector<double>& X,
                        thrust::device_vector<double>& Y,
                        thrust::device_vector<double>& Z)
{
    // Z <- A * (X - Y)
    thrust::transform(X.begin(), X.end(), Y.begin(), Z.begin(),
                      scaled_diff_functor(A));
}

            struct scaled_diff_functor
            {
                const double a;
                scaled_diff_functor(double _a) : a(_a) {}
                __host__ __device__
                    double operator()(const double& x, const double& y) const {
                        return a * (x - y);
                    }
            };
```

Thrust: counting iterator

```
#include <thrust/iterator/counting_iterator.h>
...
// create iterators
thrust::counting_iterator<int> first(10);
thrust::counting_iterator<int> last = first + 3;

first[0]   // returns 10
first[1]   // returns 11
first[100] // returns 110

// sum of [first, last)
thrust::reduce(first, last);   // returns 33 (i.e. 10 + 11 + 12)
```

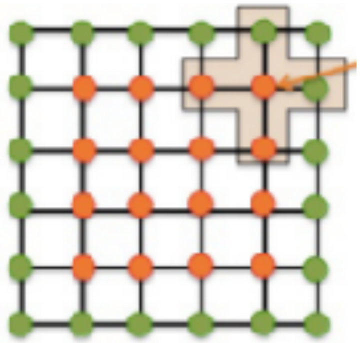
Thrust: zipping objects

```
#include <thrust/iterator/zip_iterator.h>
...
// initialize vectors
thrust::device_vector<int>  A(3);
thrust::device_vector<char> B(3);
A[0] = 10;  A[1] = 20;  A[2] = 30;
B[0] = 'x'; B[1] = 'y'; B[2] = 'z';

// create iterator (type omitted)
first = thrust::make_zip_iterator(thrust::make_tuple(A.begin(), B.begin()));
last  = thrust::make_zip_iterator(thrust::make_tuple(A.end(),   B.end()));

first[0]   // returns tuple(10, 'x')
first[1]   // returns tuple(20, 'y')
first[2]   // returns tuple(30, 'z')

// maximum of [first, last)
thrust::maximum< tuple<int,char> > binary_op;
thrust::tuple<int,char> init = first[0];
thrust::reduce(first, last, init, binary_op); // returns tuple(30, 'z')
```

Diffusion: interior region (not next to boundaries)

$$\frac{ds_{i,j}}{dt} = \frac{D}{\Delta x^2}\left(-4s_{i,j} + s_{i+1,j} + s_{i-1,j} + s_{i,j+1} + s_{i,j-1}\right) + Rs_{i,j}(1 - s_{i,j})$$

```
struct interior_functor
{
    const int nx, ny;
    const double alpha, dxs;
    interior_functor(int _nx, int _ny, double _alpha, double _dxs) : nx(_nx), ny(_ny),
                     alpha(_alpha), dxs(_dxs) {}

    template <typename Tuple>  // arguments: 0:count 1:X_OLD  2:U  3:S
    __host__ __device__
    void operator()(Tuple t)
    {
        int n = thrust::get<0>(t); // this is the counting iterator
        int i = n%nx;
        int j = n/nx;
        bool is_interior = i<(nx-1) && j<(ny-1) && i>0 && j>0;
        if(is_interior) {
           thrust::get<3>(t) = -(4. + alpha) * thrust::get<2>(t)          // central point
                             + *(&thrust::get<2>(t)-1) + *(&thrust::get<2>(t)+1) // EW
                             + *(&thrust::get<2>(t)-nx) + *(&thrust::get<2>(t)+nx) // NS
                             + alpha * thrust::get<1>(t)
                             + dxs * thrust::get<2>(t) * (1.0 - thrust::get<2>(t));
        }
    }
};
```



ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

CSCS Swiss National Supercomputing Centre

Diffusion: for_each, counting_iterator, zip

```
void diffusion_thrust(int nx, int ny, double alpha, double dxs,
       thrust::device_vector<double>& BND_W, thrust::device_vector<double>& BND_E,
       thrust::device_vector<double>& BND_S, thrust::device_vector<double>& BND_N,
       thrust::device_vector<double>& X_OLD, thrust::device_vector<double>& U,
       thrust::device_vector<double>& S
                     )
{
    const int N = nx*ny;
    thrust::counting_iterator<uint> n_first(0);
    thrust::counting_iterator<uint> n_last = n_first + N;

    // apply the transformation
    thrust::for_each(
      thrust::make_zip_iterator(thrust::make_tuple(n_first, X_OLD.begin(),
                                                   U.begin(), S.begin())),
      thrust::make_zip_iterator(thrust::make_tuple(n_last, X_OLD.end(), U.end(), S.end())),
      interior_functor(nx,ny,alpha,dxs));
}
```

Thrust Miniapp exercise

- Get latest: `cd SummerSchool2018; git pull`
- Implementation: `cd miniapp/thrust`
- Build: `cat readme.md`
- Check unit tests (interactive): `srun unit_tests`

- linalg.cu: fill in four TODOs
- operators.cu: fill in two TODOS
- advanced: MPI+Thrust implementation

```
scaled_diff_thrust         : passed
fill_thrust                : passed
axpy_thrust                : passed
  expected 10.5 got 0
add_scaled_diff_thrust     : failed
scale_thrust               : passed
  expected 15.5 got 0
lcomb_thrust               : failed
copy_thrust                : passed
dot_thrust                 : passed
  expected 4.47214 got 0
norm2_thrust               : failed
```

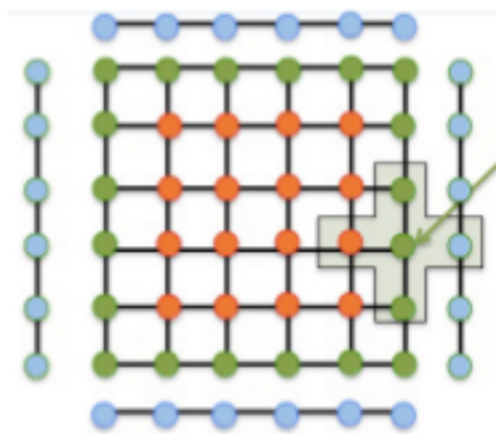
TODO: add_scaled_diff_thrust (linalg.cu)

```
// computes y = x + alpha*(l-r)
// y, x, l and r are vectors
// alpha is a scalar
void add_scaled_diff_thrust(
     double A, thrust::device_vector<double>& X, thrust::device_vector<double>& L,
     thrust::device_vector<double>& R, thrust::device_vector<double>& Y)
{
// TODO:  make tuple using make_zip_iterator where T = (X,L,R,Y)
    thrust::for_each(thrust::make_zip_iterator(thrust::make_tuple( XXXX ),
                     thrust::make_zip_iterator(thrust::make_tuple( YYYY ),
                     add_scaled_diff_functor(A));
}

struct add_scaled_diff_functor
{
    const double a;
    add_scaled_diff_functor(double _a) : a(_a) {}
     template <typename Tuple>
    __host__ __device__
        void operator()(Tuple t) const {
// TODO: program Y = X + a * (L - R); arguments of tuple are 0:X  1:L  2:R  3:Y
             thrust::get<3>(t) = …. ;
        }
};
```

Exercise: your turn

`linalg.cu:`

// TODO: program the return value
// TODO: program Y = X + a * (L - R); arguments of tuple are 0:X 1:L 2:R 3:Y
// TODO: implement the norm using sqrt and thrust::inner_product
// TODO: make tuple using make_zip_iterator where T = (X,L,R,Y)



`operators.cu:`

// TODO: implement the four corners: SW, NW, SE, NE
// TODO: zip up tuple for boundary_functor and invoke with for_each

Thrust implementation epilogue

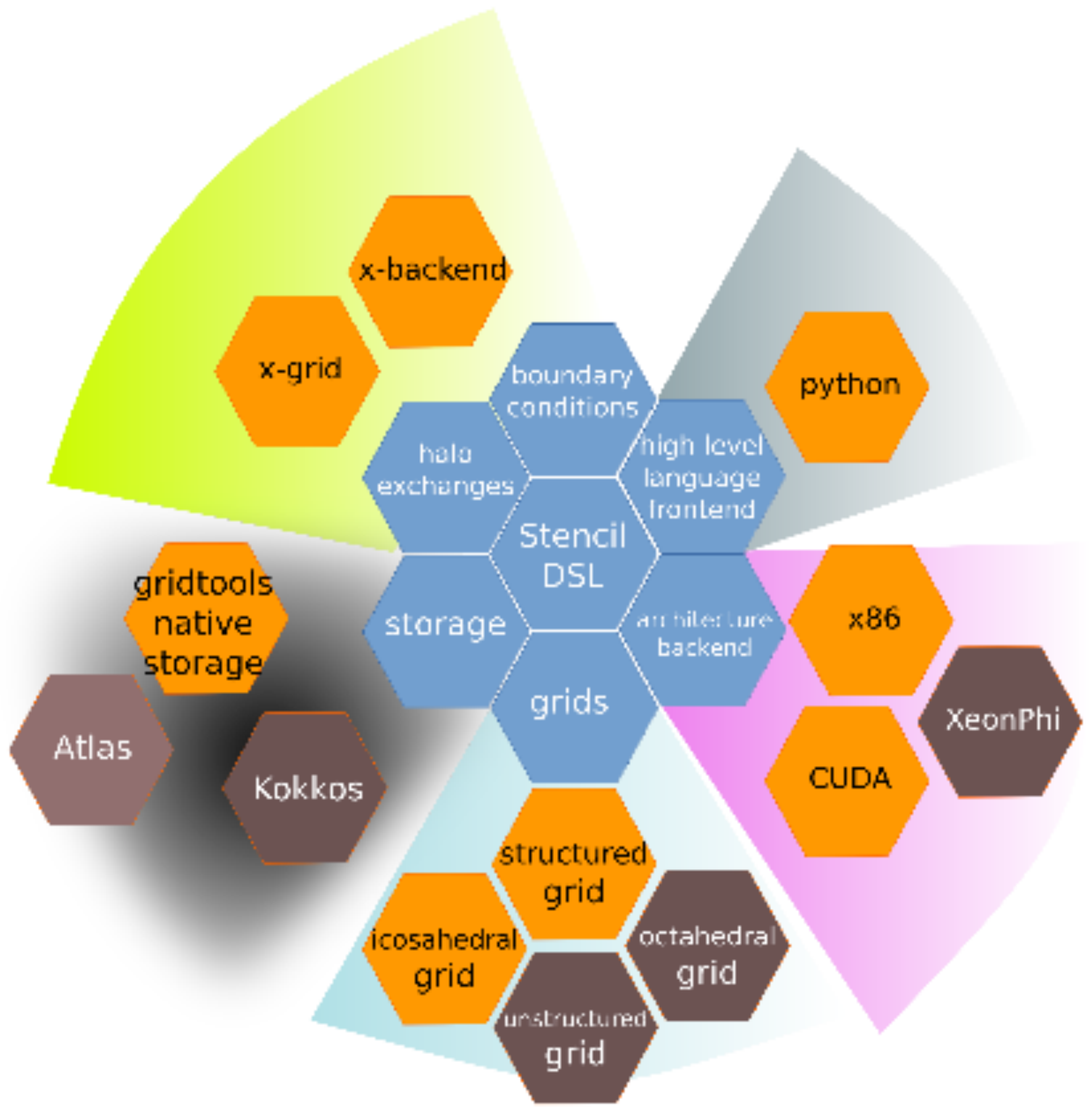
- Implementation is more succinct
- Update host/device is clearer with host/ device_vector
- No Chevron syntax
- Performance better than out-of-box CUDA implementation
- But it's still CUDA…
- True 2D data layout is difficult

Can these ideas be extended into a stencil framework?

```
CUDA
      39 cuda_helpers.h
     181 data.h
      86 linalg.h
      22 operators.h
      14 stats.h
      22 data.cu
     353 linalg.cu
     263 main.cu
     347 operators.cu
      10 stats.cu
     224 unit_tests.cu
    1561 total

THRUST
      39 cuda_helpers.h
      24 data.h
      77 linalg.h
      23 operators.h
      14 stats.h
       8 data.cu
     308 linalg.cu
     267 main.cu
     175 operators.cu
      10 stats.cu
     169 unit_tests.cu
    1114 total
```

What is GridTools?

- Set of C++ APIs / libraries
- Larger class of *stencil-based* problems
  - From structured to “less” structured
- Performance Portability
- Intuitive interface
  - Get application field specific concepts
- Basic building blocks
  - Reminiscent to the application fields
  - Each application could be more precise
- Composability
  - Shared data structures,
    naming conventions, API structure
- Interoperability
  - Even usable from Python



GridTools 7-point stencil

```
struct d3point7
    using out = accessor<0, inout >;
    using in = accessor<1, in, extent<-1,1,-1,1,-1,1> >;

    template <typename Evaluator> GT_FUNCTION
    static void Do(Evaluator eval) {
        eval(out{}) = t.0 * eval(in{})
                    - (eval(in{-1,0,0})+eval(in{+1,0,0}))
                    - (eval(in{0,-1,0})+eval(in{0,+1,0}))
                    - (eval(in{0,0,-1})+eval(in{0,0,+1}));
    }
};
```

long-wave solver (Fortran)

```
! Two-stream radiative transfer in an absorbing/emitting atmosphere (no scattering)
 subroutine lw_solver_noscat( … )
   do igpt = 1, ngpt     ! Spectral “sub-columns”
     ! Floor on optical depth to avoid ill-conditioning in layer emission function
     do ilev = 1, nlay
       tau_loc(:,ilev) = max(tau(:,ilev,igpt)/mu(:,igpt), 2._wp * TINY(1._wp))
     end do
     ! Transmittance: exp(-tau/mu)
     trans(:,:)  = exp(-tau_loc(:,:))
     ! Downward propagation
     do ilev = 2, nlay+1
       radn_dn(:,ilev,igpt) = trans(:,ilev-1) * radn_dn(:,ilev-1,igpt) +  &
                              (1._wp - trans(:,ilev-1)) *                   &
                              lay_emission(lay_source(:,ilev-1,igpt),  &
                                           lev_source_dn(:,ilev,igpt), &
                                           tau_loc(:,ilev-1), trans(:,ilev-1))
     end do
     ! Surface reflection and emission
     radn_up(:,sfcLev,igpt) = radn_dn(:, sfcLev,igpt)*(1._wp - sfc_emis(:,igpt)) + &
                              sfc_src(:        ,igpt)        *  sfc_emis(:,igpt)

     ! Upward propagation
     do ilev = nlay, 1, -1
       :
     end do
   end do
```

Example: "upward" functor

```
template < typename T >
struct lw_upward_top_is_nlay {
  static const int n_args = 8;
  typedef accessor< 0, enumtype::in >              tau;
     :
  typedef accessor< 6, enumtype::in >              radn_dn;
  typedef accessor< 7, enumtype::inout >           radn_up;
  typedef boost::mpl::vector< tau, mu, lay_source, lev_source_inc, sfc_src, sfc_emis, radn_dn, radn_up> arg_list;

  template < typename Evaluation >
  GT_FUNCTION static void Do(Evaluation const &eval, above_kminimum ) {
    T tau_loc = (eval(tau(0,0,-1))/eval(mu()))>__TWO_EPSILON__ ? (eval(tau(0,0,-1))/eval(mu())) : __TWO_EPSILON__ ;
    T trans   = exp( -tau_loc );
    T one_minus_trans =    (T)1.0 - trans;

    if ( trans < __ONE_MINUS_SPACING__ ) {
      eval( radn_up(0,0,0) ) = trans * eval( radn_up(0,0,-1) ) +
        one_minus_trans*LAY_EMISSION_LEFT(eval(lay_source(0,0,-1)), eval(lev_source_inc(0,0,0)), tau_loc, trans) ;
    } else {
      eval( radn_up(0,0,0) ) = trans * eval( radn_up(0,0,-1) ) ;
    }
  }
  template < typename Evaluation >
  GT_FUNCTION static void Do(Evaluation const &eval, kminimum ) {   // Treatment of surface
    eval( radn_up(0,0,0) ) = eval( radn_dn(0,0,0) ) * ( (T) 1.0 - eval( sfc_emis() ) ) +
      eval( sfc_src() ) * eval( sfc_emis() );
  }

};
```

Boilerplate: storage / composition

```
typedef sk_backend::storage_info< 0, layout_ikj, halo<0,0,0>, aligned<0> > meta_t_ikj_nlayp1;
typedef sk_backend::storage_info< 1, layout_ikj, halo<0,0,0>, aligned<0> > meta_t_ikj;
typedef sk_backend::storage_info< 2, layout_ij, halo<0,0,0>, aligned<0> >  meta_t_ij;
typedef sk_backend::storage_type< float_type, meta_t_ikj_nlayp1 >::type storage_type_ikj_nlayp1;
typedef sk_backend::storage_type< float_type, meta_t_ikj >::type storage_type_ikj;
typedef sk_backend::storage_type< float_type, meta_t_ij > ::type storage_type_ij;

// Definition of the placeholders.  The order is significant for the user
typedef arg< 0, storage_type_ikj >        p_tau;
typedef arg< 1, storage_type_ij >         p_mu;
  :
typedef arg< 7, storage_type_ikj_nlayp1 > p_radn_dn;
typedef arg< 8, storage_type_ikj_nlayp1 > p_radn_up;

// Array of placeholders to be passed
typedef boost::mpl::vector< p_tau,  p_mu, p_lay_source, p_lev_source_inc, p_lev_source_dec,
  p_sfc_src, p_sfc_emis, p_radn_dn, p_radn_up > accessor_list;

comp_lw = gridtools::make_computation< sk_backend >( domain, grid,
        gridtools::make_multistage(   // backward: do kmaximum first
          execute< backward >(), gridtools::make_stage< lw_downward_top_is_nlay<double> >(
            p_tau(), p_mu(), p_lay_source(), p_lev_source_dec(), p_sfc_src(), p_sfc_emis(), p_radn_dn() ) ),

        gridtools::make_multistage(   // forward: do kminimum first
          execute< forward >(), gridtools::make_stage< lw_upward_top_is_nlay<double> >(
            p_tau(), p_mu(), p_lay_source(), p_lev_source_inc(),
            p_sfc_src(), p_sfc_emis(),p_radn_dn(), p_radn_up() ) )
       );
```

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

LW_solver constructor

```
class lw_solver_noscat{
 public:
 lw_solver_noscat(int ncol, int nlay, int ngpt,
                  double tau[], double mu[], double lay_source[],
                  double lev_source_inc[], double lev_source_dec[],
                  double sfc_src[], double sfc_emis[],
                  double radn_dn[], double radn_up[] )
   :
   meta_ikj_nlayp1((uint_t)ncol, (uint_t)ngpt, (uint_t)nlay+1 ),
   meta_ikj((uint_t)ncol, (uint_t)ngpt, (uint_t)nlay ),
   meta_ij((uint_t)ncol, (uint_t)ngpt, (uint_t)1 ),
   st_tau(meta_ikj, tau, "tau"),
   st_mu(meta_ij, mu, "mu"),
      :
   st_radn_dn(meta_ikj_nlayp1, radn_dn, "downward radiation"),
   st_radn_up(meta_ikj_nlayp1, radn_up, "upward radiation"),
   domain( (p_tau() = st_tau),
           (p_mu() = st_mu),
                :
           (p_radn_dn() = st_radn_dn),
           (p_radn_up() = st_radn_up)),
   grid({0, 0, 0, ncol - 1, ncol}, {0, 0, 0, ngpt - 1, ngpt})
 {
   assert( ncol > 0 );
   assert( ngpt > 0 );
   assert( nlay > 0 );
   assert( tau );
   assert( mu );
   grid.value_list[0] = 0;                  // Splitter 0 location
   grid.value_list[1] = nlay - 1;           // Splitter 1 location
 }
```

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

GPU-enabled libraries summary

- The take home message is:

  Don't recreate the wheel
- A limited number of GPU-libraries available, e.g. CUxxxx (vendor), Thrust, ViennaCL, ...
- Parallel distributed memory libraries in development, e.g., D-MAGMA, PETSc, Trilinos, within overarching message-passing framework
- *BUT: current GPU library support is sobering*
- *Please let us know your needs!*