Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Home Affairs FDHA
Federal Office of Meteorology and Climatology MeteoSwiss

DSL toolchains and performance optimizations for weather and climate codes

Carlos Osuna, Valentin Clement, Oliver Fuhrer, Stefan Moosbruffer, Tobias Wicky (MeteoSwiss)

And the GridTools team

5th HPC ENES Workshop - Lecce May 17-18, 2018

Outline

1. Why do we need abstractions and DSLs?
2. DSL evaluation: from past until today
3. Latest developments and future plans

Why do we need abstractions and DSLs?

- Compilers will not solve the problem!
- Parallel programming is not trivial:
  - Parallelizing one physical parameterization of a well structured code: 4 weeks (OpenACC or OpenMP)
- And explicit parallel programming models is **error prone**.
- We have complex and large parallel models. Adding support for accelerated codes **significantly increased maintenance**.
- Combining different programming models in our models will exponentially **increase complexity**. And hinders scientific development.
- Computing architectures are moving faster than we can adapt our models.
- **Optimizing** a model is hard, *on multiple architectures is not possible*

Why do we need abstractions and DSLs?

Adopting our models to emerging architectures crucial to achieve challenges on the next-generation supercomputers

Use of traditional programming models are **costly to maintain** on hybrid architectures.

Invest in software rather than hardware → Invest in technology that reduces the cost rather than maintenance.

Why do we need abstractions and DSLs?

DSLs and abstractions help supporting multiple architectures by removing implementation details from the model. They give us **flexibility, lower maintenance cost, better performance**.

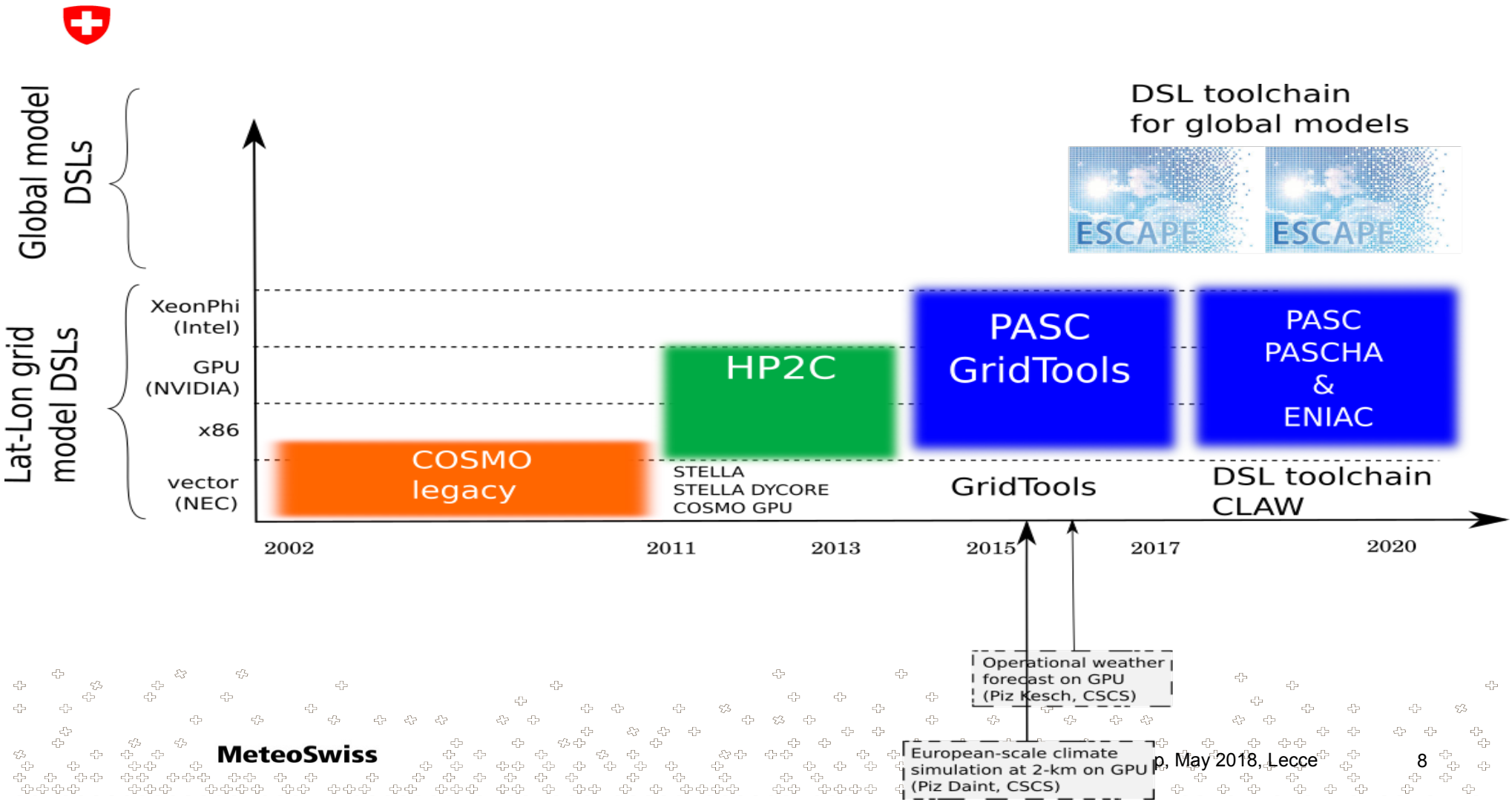
We need to find the right abstractions for our models.

Our traditional porting to hybrid architectures

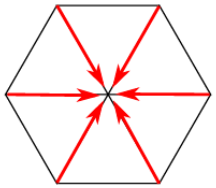
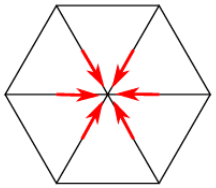
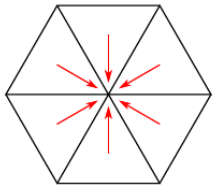
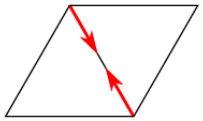
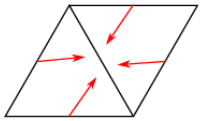
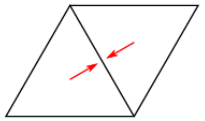
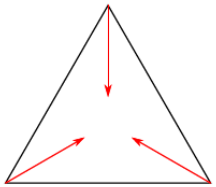
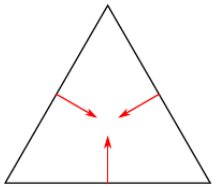
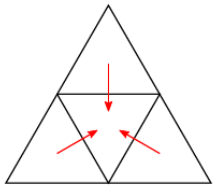
How to achieve portability for our models ?

```
#ifdef _OPENACC
!$ACC DATA PCOPYIN( psi_c ), PCOPYOUT( grad_norm_psi_e ), IF( i_am_accel_node .AND. acc_on )
!ACC_DEBUG UPDATE DEVICE( psi_c ), IF( i_am_accel_node .AND. acc_on )
!$ACC PARALLEL &
!$ACC PRESENT( ptr_patch, iidx, iblk, psi_c, grad_norm_psi_e ), &
!$ACC IF( i_am_accel_node .AND. acc_on )

!$ACC LOOP GANG
#else
!$OMP PARALLEL

!$OMP DO PRIVATE(jb,i_startidx,i_endidx,je,jk) ICON_OMP_DEFAULT_SCHEDULE
#endif
  DO jb = i_startblk, i_endblk

  CALL get_indices_e(ptr_patch, jb, i_startblk, i_endblk, &
                     i_startidx, i_endidx, rl_start, rl_end)

!$ACC LOOP VECTOR COLLAPSE(2)
#ifdef __LOOP_EXCHANGE
    DO je = i_startidx, i_endidx
      DO jk = slev, elev
#else
    DO jk = slev, elev
      DO je = i_startidx, i_endidx
#endif
       !
       ! compute the normal derivative
       ! by the finite difference approximation
       ! (see Bonaventura and Ringler MWR 2005)
       !
        grad_norm_psi_e(je,jk,jb) =  &
           &  ( psi_c(iidx(je,jb,2),jk,iblk(je,jb,2)) - &
           &    psi_c(iidx(je,jb,1),jk,iblk(je,jb,1)) )  &
           &  * ptr_patch%edges%inv_dual_edge_length(je,jb)

       ENDDO

    END DO

  END DO
#ifdef _OPENACC
!$ACC END PARALLEL
!ACC_DEBUG UPDATE HOST( grad_norm_psi_e ), IF( i_am_accel_node .AND. acc_on )
! Add $ser directives here
!$ACC END DATA
#else
!$OMP END DO NOWAIT

!$OMP END PARALLEL
#endif
```

```
#ifdef _OPENACC
!$ACC DATA PRESENT( psi_c ), PRESENT( grad_norm_psi_e ), IF( i_am_accel_node .AND. acc_on )
!ACC_DEBUG UPDATE DEVICE( psi_c ), IF( i_am_accel_node .AND. acc_on )
!$ACC PARALLEL &
!$ACC PRESENT( ptr_patch, iidx, iblk, psi_c, grad_norm_psi_e ), &
!$ACC IF( i_am_accel_node .AND. acc_on )

!$ACC LOOP GANG
#else
!$OMP PARALLEL

!$OMP DO PRIVATE(jb,i_startidx,i_endidx,je,jk) ICON_OMP_DEFAULT_SCHEDULE
#endif
  DO jb = i_startblk, i_endblk

  CALL get_indices_e(ptr_patch, jb, i_startblk, i_endblk, &
                     i_startidx, i_endidx, rl_start, rl_end)

!$ACC LOOP VECTOR COLLAPSE(2)
#ifdef __LOOP_EXCHANGE
    DO je = i_startidx, i_endidx
      DO jk = slev, elev
#else
    DO jk = slev, elev
      DO je = i_startidx, i_endidx
#endif
        !
        ! compute the normal derivative
        ! by the finite difference approximation
        ! (see Bonaventura and Ringler MWR 2005)
        !
        grad_norm_psi_e(je,jk,jb) =  &
          &  ( psi_c(iidx(je,jb,2),jk,iblk(je,jb,2)) - &
          &    psi_c(iidx(je,jb,1),jk,iblk(je,jb,1)) )  &
          &  * ptr_patch%edges%inv_dual_edge_length(je,jb)

      ENDDO

    END DO

  END DO
#ifdef _OPENACC
!ACC END PARALLEL
!ACC_DEBUG UPDATE HOST( grad_norm_psi_e ), IF( i_am_accel_node .AND. acc_on )
! Add user directives here
!$ACC END DATA
#else
!$OMP END DO NOWAIT

!$OMP END PARALLEL
#endif
```


Global model DSLs
Lat-Lon grid model DSLs
XeonPhi (Intel)
GPU (NVIDIA)
x86
vector (NEC)
DSL toolchain for global models
ESCAPE
ESCAPE
PASC GridTools
PASC PASCHA & ENIAC
HP2C
COSMO legacy
STELLA
STELLA DYCORE
COSMO GPU
GridTools
DSL toolchain CLAW
2002
2011
2013
2015
2017
2020
Operational weather forecast on GPU (Piz Kesch, CSCS)
European-scale climate simulation at 2-km on GPU (Piz Daint, CSCS)

DSL syntax for Connectivity on Unstructured Mesh

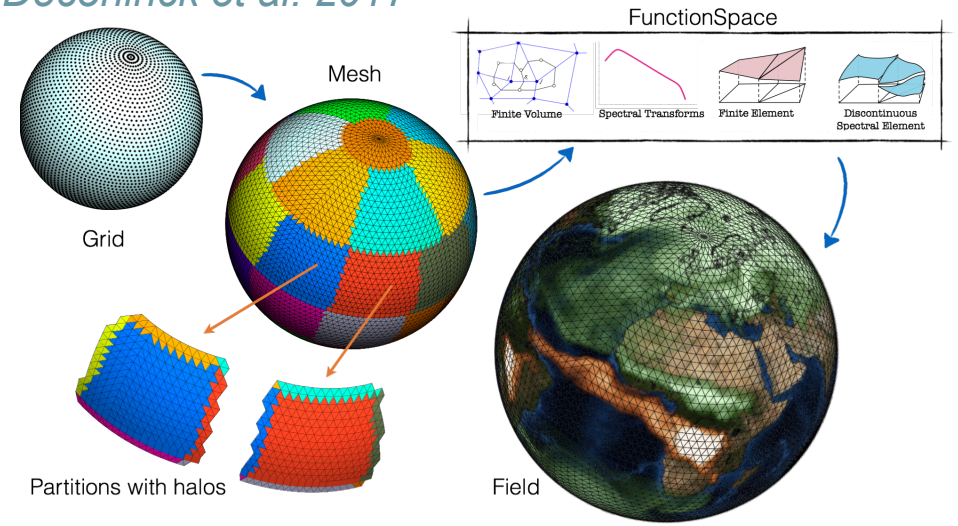
primal location	on_vertices()	on_edges()	on_cells()
vertex			
edge			
cell			

```
constepxr auto offsets =
    connectivity<edges, vertices, Color>::offsets();
for( auto off: offsets) {
  eval(flux) += eval( pD(off) );

}
```

```
eval(flux) = eval(sum_on_vertices(0.0, pD));
```

Atlas: Grid abstraction for GridTools backends

Deconinck et al. 2017




```
constepxr auto offsets =
    connectivity<edges, vertices, Color>::offsets();
for( auto off: offsets) {
  eval(flux) += eval( pD(off) );

}
```

```
eval(flux) = eval(sum_on_vertices(0.0, pD));
```

- The DSL syntax elements are grid independent.
- The same used code using GridTools DSL can be compiled for multiple Grids.
- **GridTools interoperating with Atlas library for meshes of weather and climate (ECMWF) in order to efficiently support multiple grids**

Example of GridTools DSL on ESCAPE dwarf: MPDATA

```
template <uint_t Color> struct upwind_flux {
  using flux = accessor<0, enumtype::inout, icosahedral_topology_t::edges>;
  using pD =
      in_accessor<1, icosahedral_topology_t::vertices, extent<0, 1, 0, 1>>;
  using vn = in_accessor<2, icosahedral_topology_t::edges, extent<0, 1, 0, 1>>;

  typedef boost::mpl::vector<flux, pD, vn> arg_list;

  template <typename Evaluation> static void Do(Evaluation &eval, k_full) {

    constexpr auto neighbors_offsets =
        connectivity<edges, vertices, Color>::offsets();
    constexpr auto ip0 = neighbors_offsets[0];
    constexpr auto ip1 = neighbors_offsets[1];

    float_type pos = math::max(eval(vn()), (float_type)0.);
    float_type neg = math::min(eval(vn()), (float_type)0.);

    eval(flux()) = eval(pos * pD(ip0) + neg * pD(ip1));
  }
};
```

```
subroutine compute_upwind_flux(this,pflux,pD,pVn)
type(MPDATA_type), intent(inout) :: this
real(wp), intent(out) :: pflux(:,:)
real(wp), intent(in)  :: pVn(:,:), pD(:,:)
real(wp) :: zpos, zneg
integer :: nb_edges
integer :: nb_levels
integer  :: jedge, jlev, ip1, ip2

call atlas_log%debug('compute_upwind_flux')

nb_edges  = this%dimensions%nb_edges
nb_levels = this%dimensions%nb_levels

!$OMP PARALLEL DO SCHEDULE(STATIC) PRIVATE(jedge,jlev,ip1,ip2,zpos,zneg)
do jedge = 1,nb_edges
  ip1 = iedge2node(1,jedge)
  ip2 = iedge2node(2,jedge)
  do jlev = 1,nb_levels
    zpos               = max(0._wp,pVn(jlev,jedge))
    zneg               = min(0._wp,pVn(jlev,jedge))
    pflux(jlev,jedge)  = pD(jlev,ip1)*zpos+pD(jlev,ip2)*zneg
  enddo
enddo
!$OMP END PARALLEL DO
end subroutine compute_upwind_flux
```

Composition of multiple MPDATA operators

```
m_upwind_fluxes = make_computation<gpu>(
 domain_uwf, grid_,
 make_multistage(execute<forward>(),
   make_stage<upwind_flux, edges>(
             p_flux(), p_pD(), p_vn()),
   make_stage<upwind_fluz, vertices>(
             p_fluz(), p_pD(), p_wn()),
   make_stage<fluxzdiv, vertices>(
             p_divVD(), p_flux(), p_fluz(),
             p_dual_volumes(), p_edges_sign()),
   make_stage<advance_solution, vertices>(
             p_pD(), p_divVD(), p_rho())));
```

Full MPDATA implemented using the GridTools DSL:

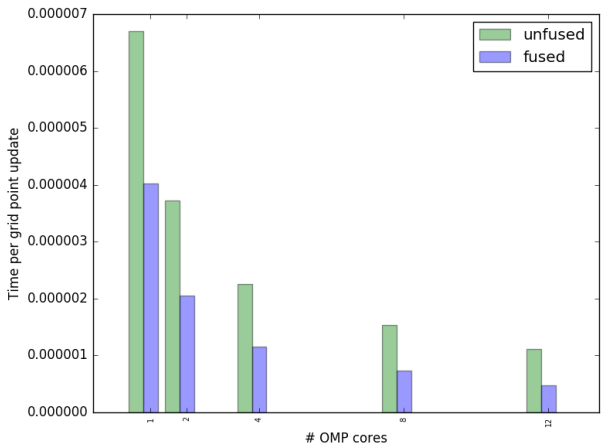
- ✓ upwind fluxes
- ✓ minmax limiter
- ✓ compute fluxes
- ✓ Limit fluxes
- ✓ Flux solution
- ✓ Rho correction

Composition of multiple MPDATA operators

```
m_upwind_fluxes = make_computation<gpu>(
 domain_uwf, grid_,
 make_multistage(execute<forward>(),
   make_stage<upwind_flux, edges>(
              p_flux(), p_pD(), p_vn()),
   make_stage<upwind_fluz, vertices>(
              p_fluz(), p_pD(), p_wn()),
   make_stage<fluxzdiv, vertices>(
              p_divVD(), p_flux(), p_fluz(),
              p_dual_volumes(), p_edges_sign()),
   make_stage<advance_solution, vertices>(
              p_pD(), p_divVD(), p_rho())));
```

Composition of stencils and internal loop fusion increases data locality.

W/o loop fusion: 8 loads, 4 stores
With loop fusion: 3 loads, 1 store



MPDATA time per grid point update (s) for O32 on Sandy Bridge

Advantages / Disadvantages

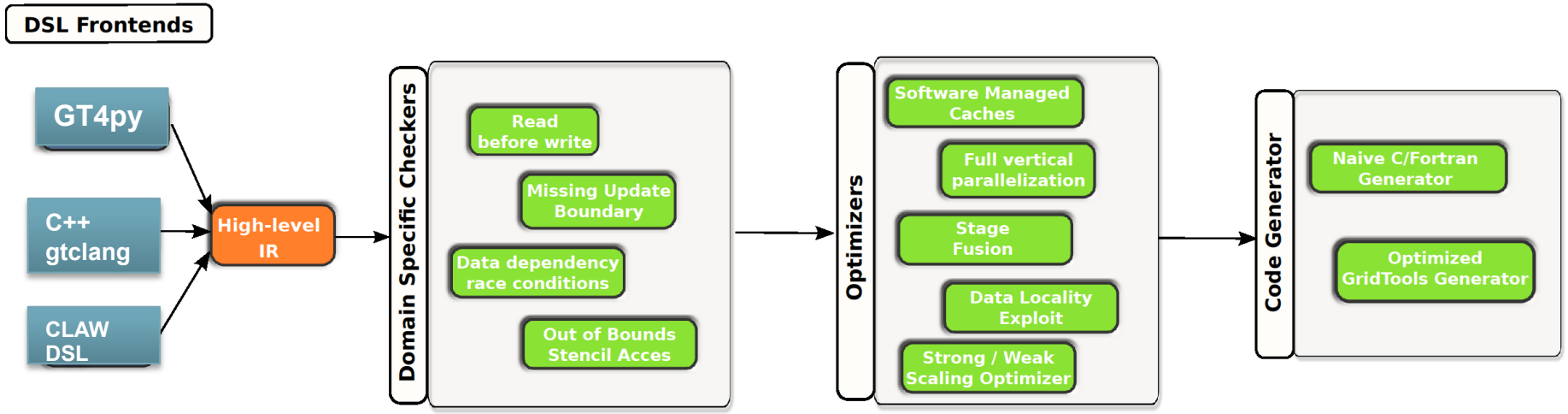
- We have demonstrated portability for (dwarfs) global weather models.
- Performance speedup wrt baseline (due to library optimizations fusion/data layouts/tiling/loop nesting, etc.).
- Solid parallel programming model
- Same numerical code compiles and runs on multiple architecture backends.

Advantages / Disadvantages

- Considerable boiler plate
- ~ same number of lines as compared to Fortran baseline
- Porting to DSL is still a considerable effort
- Requires expertise (C++ and understand the parallel model to obtain good performance).

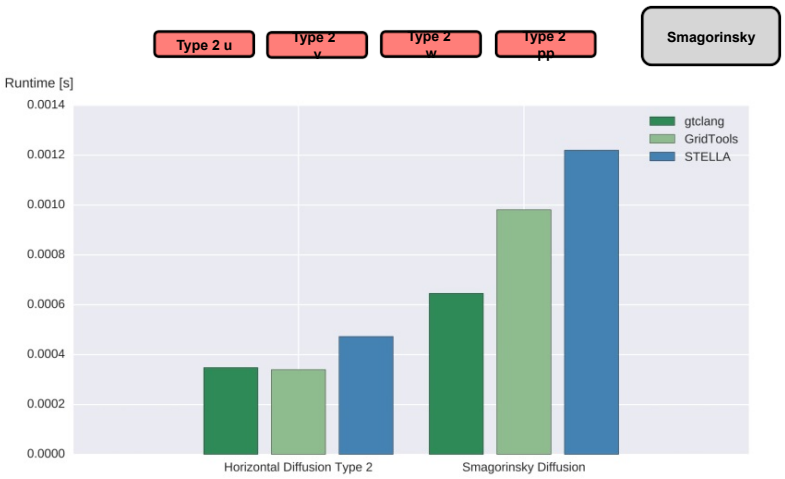
Next generation DSL will use current GridTools as a solid intermediate, and offer high level DSL frontends for fast scientific development

A DSL toolchain (PASCHA and ESCAPE2)



A COSMO Dycore under a toolchain

- Prototype of full dycore: (4 kLOC) vs STELLA dycore (27 kLOC)

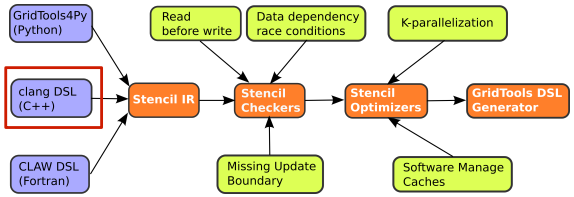


Horizontal Diffusion (P100)

Example – FastWaves UV

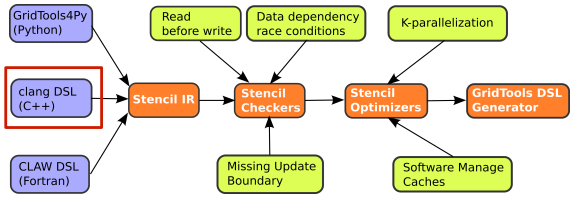
```
stencil_function sym_avg {
  storage data;
  offset off;
  double Do {
    return (data[off] + data[-off])*0.5;
} };
stencil uv {
  storage u_tens_stage, v_tens_stage, u_stage, v_stage;
  var uavg, vavg;
  Do {
    vertical_region(k_start, k_end) {
      var uatupos = sym_avg(i, u_stage);
      var vatupos = avg(i+1, avg(j-1, v_stage));
      uavg = uatupos * acrlat0;
      vavg = vatupos * EARTH_RADIUS_RECIP;
      u_tens_stage = horizontal_advection::driver(u_stage, uavg, vavg, eddlat, eddlon) +
                tgrlatda0 * u_stage * vatupos;
      var uatvpos = avg(i-1, avg(j+1, u_stage));
      var vatvpos = sym_avg(j, v_stage);
      uavg = uatvpos * acrlat1;
      vavg = vatvpos * EARTH_RADIUS_RECIP;
      v_tens_stage = horizontal_advection::driver(v_stage, uavg, vavg, eddlat, eddlon) +
        tgrlatda1 * uatvpos * vatvpos;
} } };
```

Safety First




```
stencil hd_type2 {
  storage u, utens;
  temporary_storage uavg;
  Do {
    vertical_region(k_start, k_end-1) {
      uavg = utens*0.5;
    }
    vertical_region(k_end-1, k_start) {
      u = (uavg[k+1]);
    }
  }
};
```

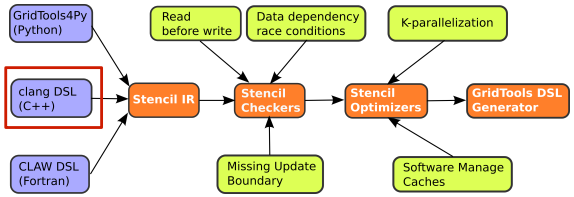
Safety First




```
stencil hd_type2 {
  storage u, utens;
  temporary_storage uavg;
  Do {
    vertical_region(k_start, k_end) {
      uavg = (u[i+1]+u[i-1])*0.5;
      utens = (uavg[j+1] + uavg[j-1]);
    }
  }
};
```

Missing halo update

Safety First

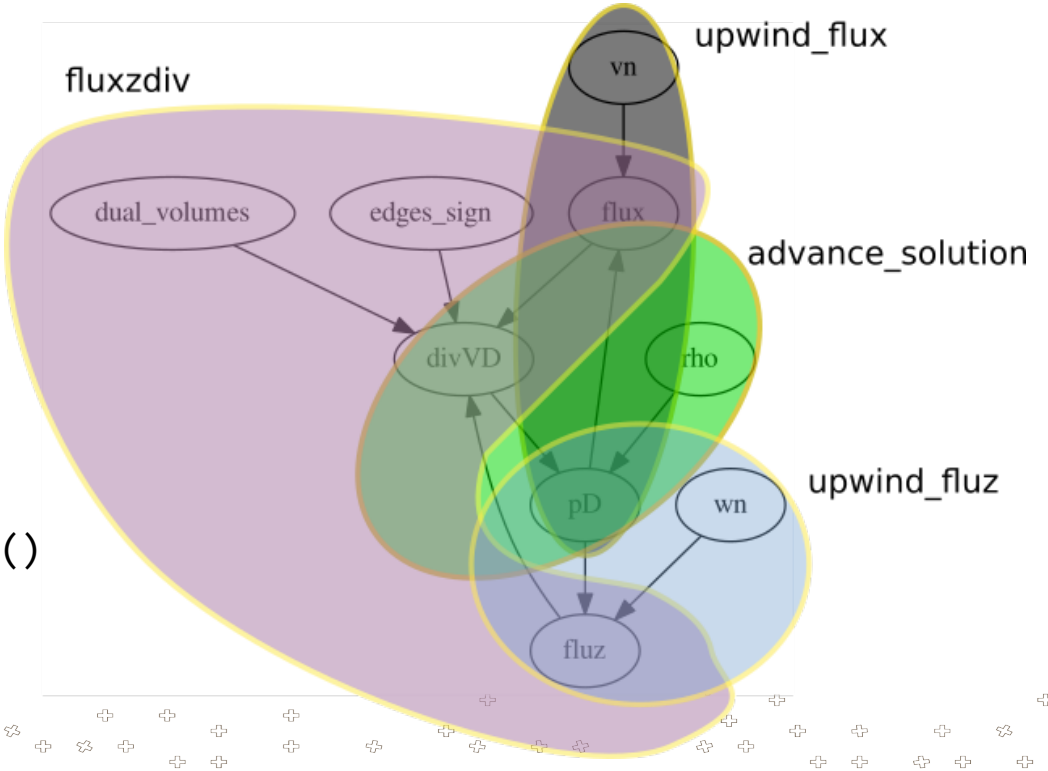



```
stencil hd_type2 {
  storage u, utens;
  temporary_storage uavg;
  Do {
    vertical_region(k_start, k_end) {
      uavg = (u[i+1]+u[i-1])*0.5;
      utens = (uavg[j+1] + uavg[j-1]);
    }
  }
};
```

Access uninitialized data

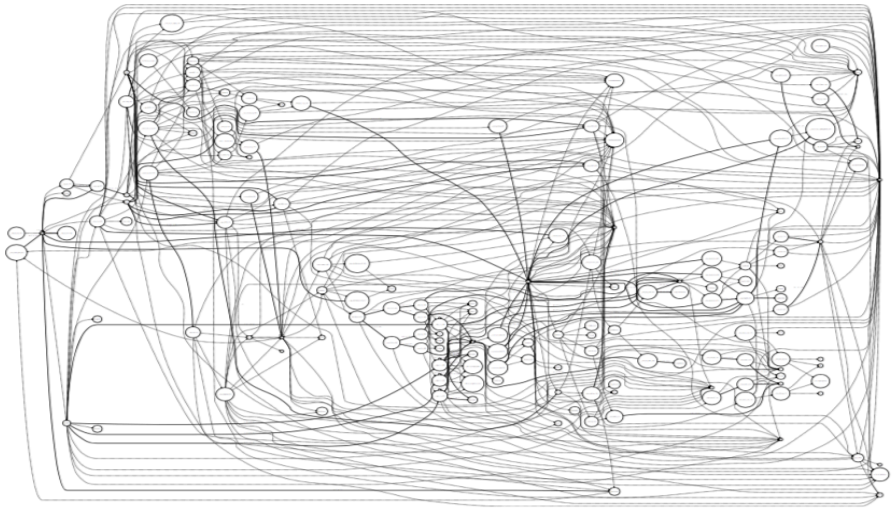
Automatic Optimizations and partitions

```
m_upwind_fluxes = make_computation<gpu>(
 domain_uwf, grid_,
 make_multistage(execute<forward>(),
   make_stage<upwind_flux, edges>(
              p_flux(), p_pD(), p_vn()),
   make_stage<upwind_fluz, vertices>(
              p_fluz(), p_pD(), p_wn()),
   make_stage<fluxzdiv, vertices>(
              p_divVD(), p_flux(), p_fluz(),
              p_dual_volumes(), p_edges_sign()
   make_stage<advance_solution, vertices>(
              p_pD(), p_divVD(), p_rho())));
```



Automatic Optimizations and partitions

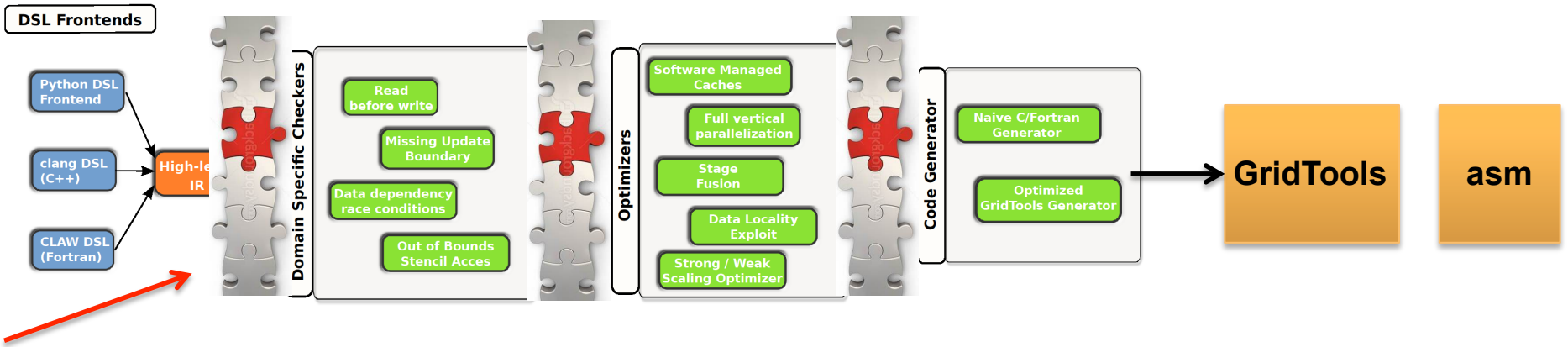
- Compilers will not solve the problem!
- DSL-based code can be automatically optimized for a specific hardware target
- E.g. “Design of a Compiler Framework for Domain Specific Languages for Geophysical s)



Example: Fast-waves solver

- Graph representation of code
- Rearrange for data-locality
- Run independent computations in parallel

HIR: Standard Interfaces for interoperability (ESCAPE)



HIR: standard interface between the domain specific **DSL, or parse of legacy code** and toolchain

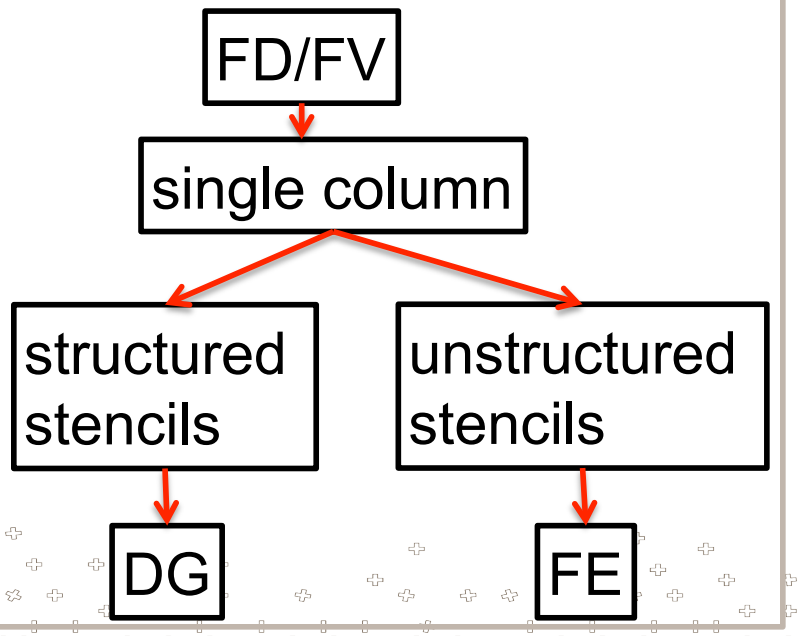
Internal IR: standard interface around optimizers, that allow to **inter-operate** different **optimizers**, **performance models** (Absinthe) and programming models [parallel IJ – sequential K][parallelize over dofs]

Standardization: HIR

Multiple DSL (thin) frontends can generate standard HIR scheme

C++
GTCLANG
Python
GridTools4Py
Fortran
CLAW
ESCAPE2 / Ocean
Fortran
PsyClone

Multiple schemes of HIR for different domains



Good Properties of an IR

- Language independent
- Complete
- HW neutral
- Extensible / enrichable
- Simple
- Orthogonal concepts

- **FieldDecl** [
  dimensions: i,j,k,h
  string: name
]
- **VarDecl** (scalar)
- **Computations**: multidimensional boxed loops over dimensions of the domain
- **StencilAST**: description of the computations within a Computation.
- **For loops**: iterations over loop bounds on scalars (not dimensions of the domain)
- **BoundaryCondition**: specification of computation applied on a boundary to a field

Standardization: HIR

$$b = a(i+1) + a(i-1)$$


```
{ «scheme» : «finite_difference_stencil»,
  «filename» : «/code/access_test.cpp»,
  «stencils» : [
    «name» : «hori_diff»,
    «loc» : {
      «line»: 24,
      «column»: 8
    },
```

```
«ast»: {
      «block_stmt»: {
        «line»: 46,
        «statements»: {
          «expr_stmt»: {
            «assignment_expr»: {
              «left»: {
                «field_access_expr»: {
                  «name»: «b»,
                  «offset»: [],
                }
              },
              «right»: {
                «binary_op»: {
                  «left»: {
                    «field_access_expr»: {
                      «name»: «a»,
                      «offset»: [1,0,0],
                    }
                  },
                  «right»: {
                    «field_access_expr»: {
                      «name»: «a»,
                      «offset»: [-1,0,0],
}}}}}}}}}}
```

Standardization: HIR

b = a(i+1) + a(i-1)

{ «scheme» : «finite_difference_stencil»,
  «filename» : «/code/access_test.cpp»,
  «stencils» : [
    «name» : «hori_diff»,
    «loc» : {
      «line»: 24,
      «column»: 8
    },

«ast»: {
    «block_stmt»: {
     «line»: 46,
     «statements»: {
      «expr_stmt»: {
       «assignment_expr»: {
        «left»: {
         «field_access_expr»: {
          «name»: «b»,
          «offset»: [],
         }
        },
        «right»: {
         «binary_op»: {
          «left»: {
           «field_access_expr»: {
            «name»: «a»,
            «offset»: [1,0,0],
           }
          },
          «right»: {
           «field_access_expr»: {
            «name»: «a»,
            «offset»: [-1,0,0],
}} }} } } } } } } }

Standardization: HIR

$$b = a(i+1) + a(i-1)$$


```
{ «scheme» : «finite_difference_stencil»,
  «filename» : «/code/access_test.cpp»,
  «stencils» : [
    «name» : «hori_diff»,
    «loc» : {
      «line»: 24,
      «column»: 8
    },
```

```
«ast»: {
      «block_stmt»: {
        «line»: 46,
        «statements»: {
          «expr_stmt»: {
            «assignment_expr»: {
              «left»: {
                «field_access_expr»: {
                  «name»: «b»,
                  «offset»: [],
                }
              },
              «right»: {
                «binary_op»: {
                  «left»: {
                    «field_access_expr»: {
                      «name»: «a»,
                      «offset»: [1,0,0],
                    }
                  },
                  «right»: {
                    «field_access_expr»: {
                      «name»: «a»,
                      «offset»: [-1,0,0],
}}}}}}}}}}
```

Standardization: HIR

$$b = a(i+1) + a(i-1)$$


```
{ «scheme» : «finite_difference_stencil»,
  «filename» : «/code/access_test.cpp»,
  «stencils» : [
    «name» : «hori_diff»,
    «loc» : {
      «line»: 24,
      «column»: 8
    },
```

```
«ast»: {
      «block_stmt»: {
        «line»: 46,
        «statements»: {
          «expr_stmt»: {
            «assignment_expr»: {
              «left»: {
                «field_access_expr»: {
                  «name»: «b»,
                  «offset»: [],
                }
              },
              «right»: {
                «binary_op»: {
                  «left»: {
                    «field_access_expr»: {
                      «name»: «a»,
                      «offset»: [1,0,0],
                    }
                  },
                  «right»: {
                    «field_access_expr»: {
                      «name»: «a»,
                      «offset»: [-1,0,0],
}}}}}}}}}}
```

Standardization: HIR

b = a(i+1) + a(i-1)

{ «scheme» : «finite_difference_stencil»,
  «filename» : «/code/access_test.cpp»,
  «stencils» : [
    «name» : «hori_diff»,
    «loc» : {
      «line»: 24,
      «column»: 8
    },
«ast»: {
  «block_stmt»: {
   «line»: 46,
   «statements»: {
    «expr_stmt»: {
     «assignment_expr»: {
      «left»: {
       «field_access_expr»: {
        «name»: «b»,
        «offset»: [],
       }
      },
      «right»: {
       «binary_op»: {
        «left»: {
         «field_access_expr»: {
          «name»: «a»,
          «offset»: [1,0,0],
         }
        },
        «right»: {
         «field_access_expr»: {
          «name»: «a»,
          «offset»: [-1,0,0],
}} }} } } } } } } }

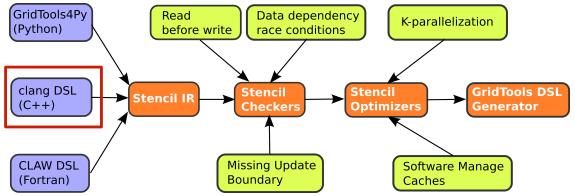
Conclusions

- **Maintenance cost**: as we incorporate performance and support for multiple architectures maintenance of our models will grow
- Abstractions give us **flexibility, lower maintenance cost and performance**.
- Next generation DSL and tools on the GridTools framework: PASC (COSMO), ESCAPE2 (global models)
- Standardization of programming models and tools for weather and climate domain: **HIR**

BACKUPS

Interoperatbility : Escaping the language

```
stencil_function laplacian {
  storage phi;
  Do {
    return 4.0 * phi - phi[i+1] - phi[i-1] - phi[j+1] - phi[j-1];
  }
};
stencil hd_type2 {
  storage hd, in;
  temporary_storage lap;
  Do {
    vertical_region(k_start, k_end) {
      lap = laplacian(in);
      hd = laplacian(lap);
    }
  }
};
#omp parallel for nowait
#acc …
for(int i=0; i < isize; ++i) {
  for(int j=0; j < jsize; ++j) {
    for(int k=0; k < ksize; ++k) {
      udiff(i,j,k) = hd(i+1,j,k) + hd(i-1,j,k);
}  }  }
```



DSL code

Standard C++ code
(+OpenMP, +OpenACC, +CUDA)