Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

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

Towards a domain specific library for stencil methods on grids

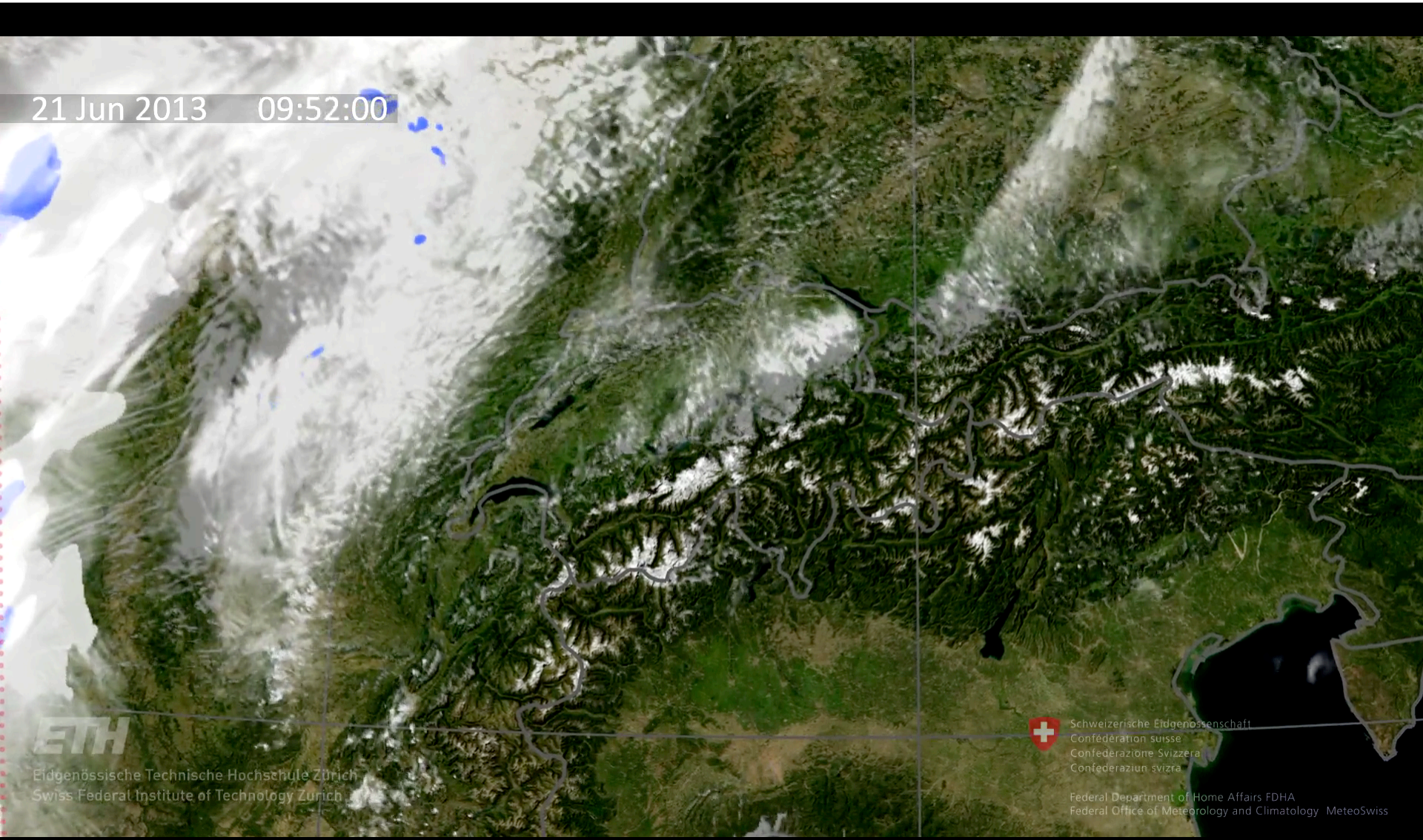
An example from numerical weather prediction and regional climate modeling

Oliver Fuhrer, MeteoSwiss

Co-authors: Tobias Gysi, Supercomputing Systems AG

Thomas Schulthess, CSCS

COSMO – Weather and Climate Model

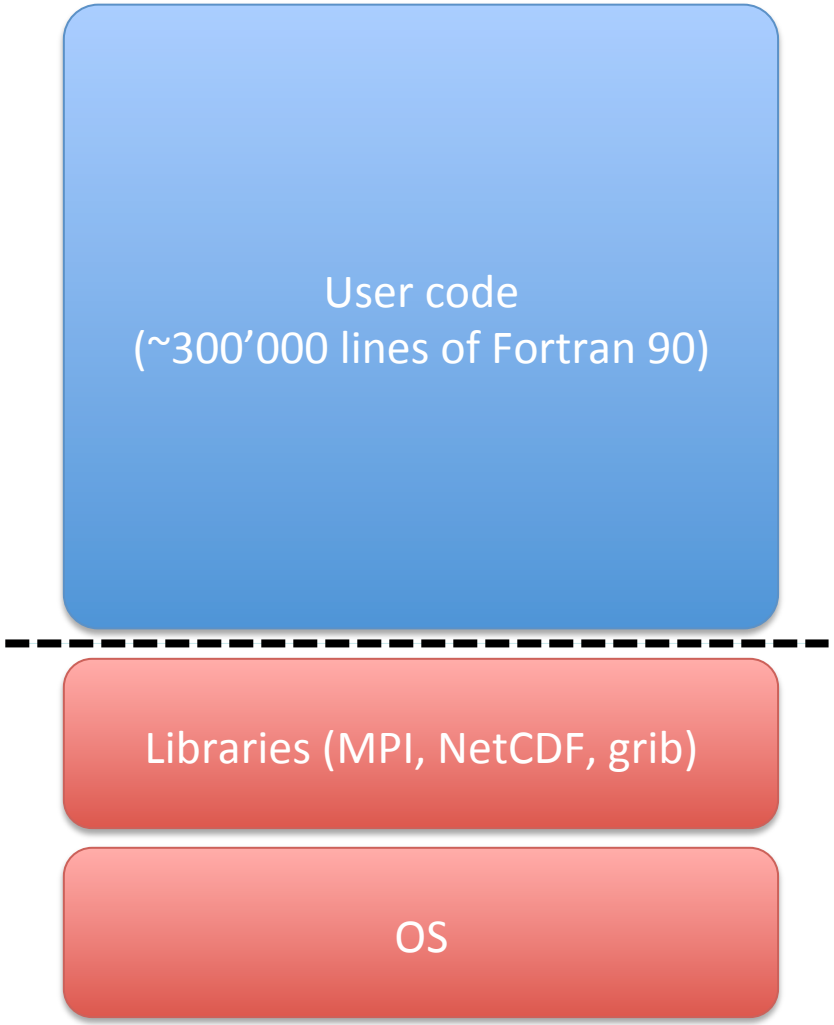


Starting Point

Goal

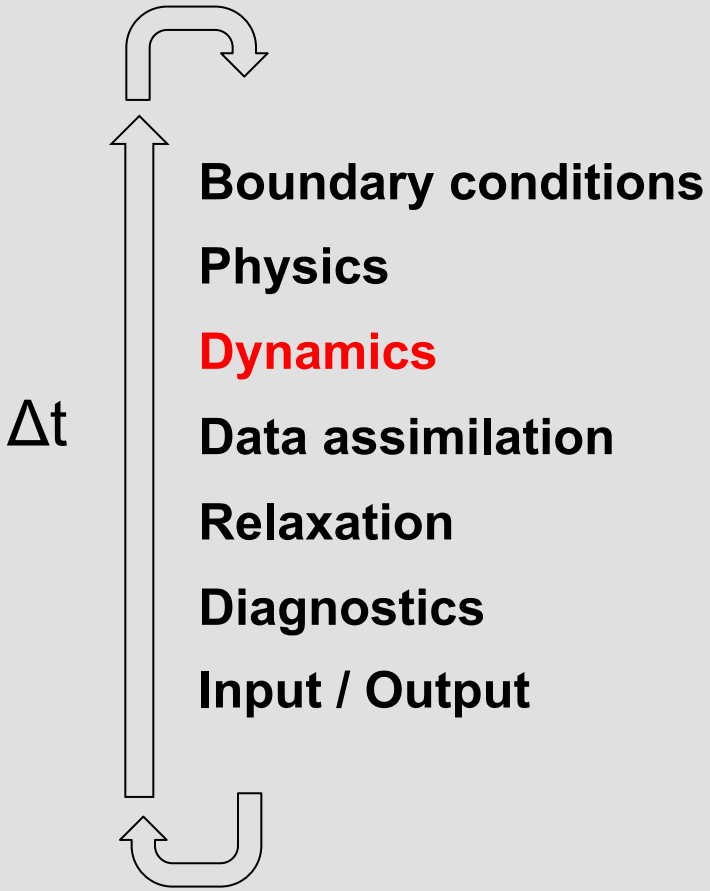
- Leap in capabilities
  - higher resolution
  - larger domains
  - ensembles

- Co-designed for new CSCS systems



COSMO Workflow

Initialization



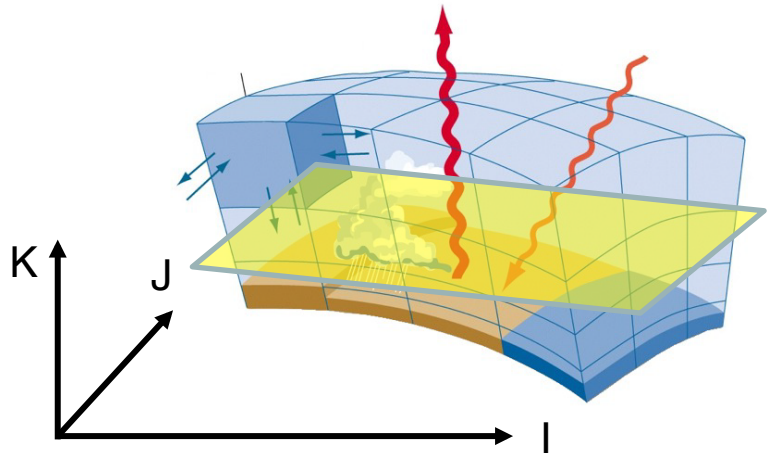
Dynamics Properties

- ~60% of runtime
- PDEs
- Finite differences
- Structured grid
- Memory bandwidth bound

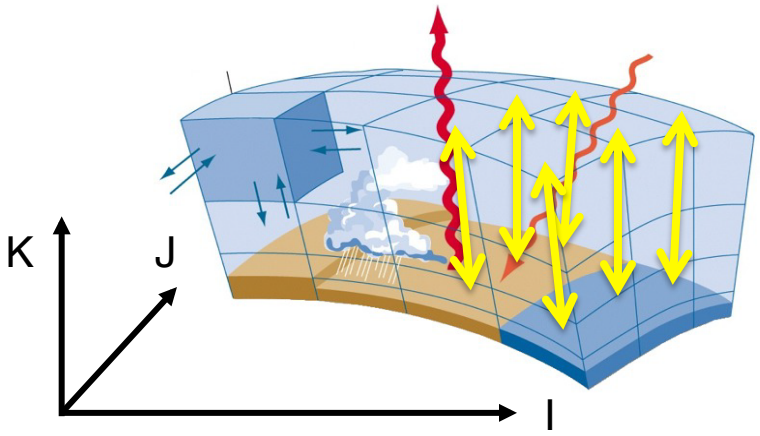
Cleanup

Algorithmic motifs

1. **Finite difference stencil computations**
   - Focus on horizontal IJ-plane accesses
   - No loop carried dependencies



2. **Tri-diagonal solves**
   - vertical K-direction stencils
   - Loop carried dependencies in K



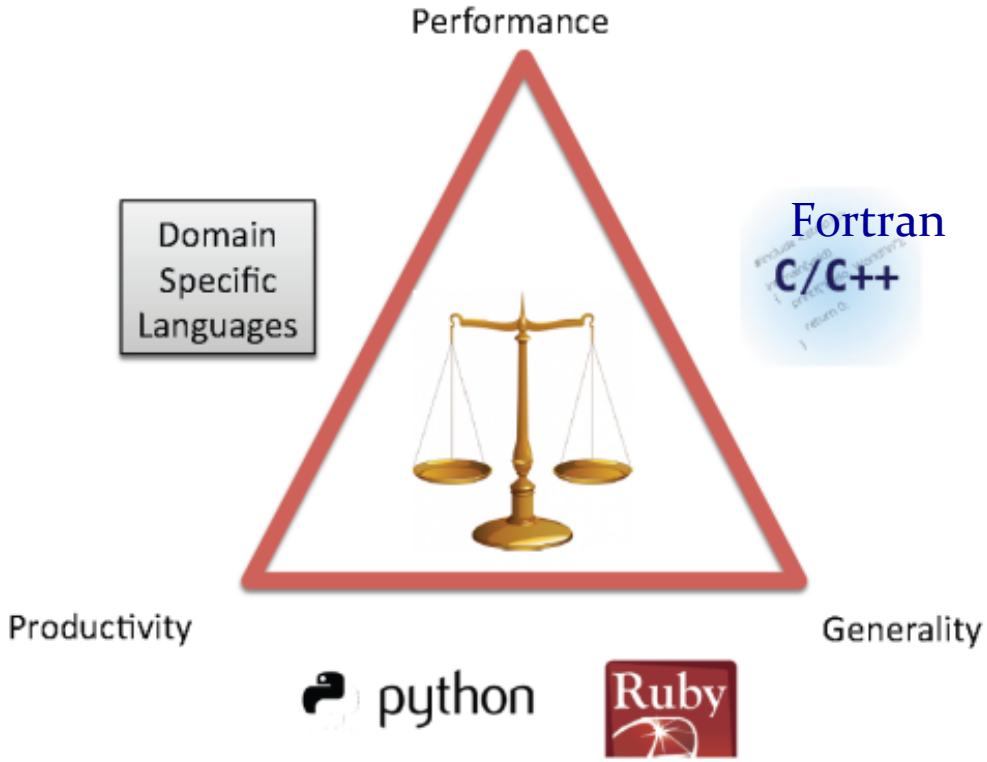
Dynamics Characteristics

- Performance critical
- Limited domain (stencils on structured grids)
- Concise high-level notation
- Complex, hardware dependent optimizations (data structures, parallelization, data locality)
- Not amenable to library function calls
- Little exchange of code
- Representative mini-apps

→ Very good fit for a domain-specific language

Domain-specific languages (DSLs)

- Flavors of implementation
  - Source-to-source translation
  - embedded in host language
  - compiler extension



STELLA Library

- Domain specific (embedded) language (DSEL)
- C++ host language
- Implemented using template meta-programming
- Geared towards COSMO
- (Partial) Abstractions
  - Data storage
  - Loops
  - Parallelization (on node)
  - Temporaries / Caching
  - Caching

Example: Laplacian

- Operator has two main components
  - Loop-logic defining the stencil application domain and order
  - Stencil defining the operator to be applied

```
do k = kstart, kend
  do j = jstart, jend
    do i = istart, iend
      lap(i, j, k) = -4.0_ir * data(i, j, k) +   &
        data(i+1, j  , k) + data(i-1, j  , k) +  &
        data(i  , j+1, k) + data(i  , j-1, k)
    end do
  end do
end do
```

STELLA (user code)

```
enum { data, lap };

template<typename TEnv>
struct Laplace
{
  STENCIL_STAGE(Tenv)
  STAGE_PARAMETER(FullDomain, data)
  STAGE_PARAMETER(FullDomain, lap)

  static void Do()
  {
    lap::Center() =
      -4.0 * data::Center() +
      data::At(iplus1) +
      data::At(iminus1) +
      data::At(jplus1) +
      data::At(jminus1);
  }
};
```

```
IJKRealField lapfield, datafield;
Stencil stencil;

StencilCompiler::Build(
pack_parameters(
    Param<lap, cInOut>(lapfield),
    Param<data, cIn>(datafield)
  ),
  concatenate_sweeps(
    define_sweep<KLoopFullDomain>(
      define_stages(
        StencilStage<Laplace, IJRangeComplete>()
      )
    )
  )
);

stencil.Apply();
```

STELLA (user code)

Stencil

```
enum { data, lap };

template<typename TEnv>
struct Laplace
{
  STENCIL_STAGE(Tenv)
  STAGE_PARAMETER(FullDomain, data)
  STAGE_PARAMETER(FullDomain, lap)

  static void Do()
  {
    lap::Center() =
      -4.0 * data::Center() +
      data::At(iplus1) +
      data::At(iminus1) +
      data::At(jplus1) +
      data::At(jminus1);
  }
};
```

Loop-logic

```
IJKRealField lapfield, datafield;
Stencil stencil;

StencilCompiler::Build(
pack_parameters(
    Param<lap, cInOut>(lapfield),
    Param<data, cIn>(datafield)
  ),
  concatenate_sweeps(
    define_sweep<KLoopFullDomain>(
      define_stages(
        StencilStage<Laplace, IJRangeComplete>()
      )
    )
  )
);

stencil.Apply();
```

Loop Definition

No one-to-one correspondence!
This is just for illustration.

```
concatenate_sweeps(
  define_sweep<cKIncrement>(
    define_stages(
      StencilStage<Operator1,
                   IJRange<-1,1,-1,1>,
                   KRangeFullDomain>(),
      StencilStage<Operator2,
                   IJRange<0,0,0,0>,
                   KRangeFullDomain>()
    )
  ),
  define_sweep<cKDecrement>(
    define_stages(
      StencilStage<Operator3,
                   IJRange<0,0,0,0>,
                   KRangeFullDomain>()
    )
  )
)
```

```
DO k = 1, ke
  DO j = jstart-1, jend+1
    DO i = istart-1, iend+1
      ! Operator1
    ENDDO
  ENDDO
  DO j = jstart, jend
    DO i = istart, iend
      ! Operator2
    ENDDO
  ENDDO
ENDDO
DO k = ke, 1, -1
  DO j = jstart, jend
    DO i = istart, iend
      ! Operator3
    ENDDO
  ENDDO
ENDDO
```

Functions

- Operators can be defined as functions (e.g. Laplacian, …)
- Functions can be nested

```
  static void Do() {
    res::Center() = Call<lap>::With( Call<lap>::With( data::Center() ) );
  }
```

- Code is closer discretized mathematical model

```
utens::Center() +=
    Call<Average>::With( mass2u,
        fc::Center() * Call<Average>::With( v2mass, v::Center() ) ) );
```

```
  z_fv_north = fc(i,j)   * ( v(i,j  ,k,nn) + v(i+1,j  ,k,nn) )
  z_fv_south = fc(i,j-1) * ( v(i,j-1,k,nn) + v(i+1,j-1,k,nn) )
         zfq = 0.25_ireals * ( z_fv_north + z_fv_south )
utens(i,j,k) = utens(i,j,k) + zfq
```

Buffers

- Buffers (of optimal size and placement) are used for passing information between operators

STELLA

```
define_buffers(
  StencilBuffer<a, Real, IJKColumn>(),
),
concatenate_sweeps(
  define_sweep<cKIncrement>(
    define_stages(
      StencilStage<Operator1,
                   IJRange<-1,1,-1,1>,
                   KRangeFullDomain>(),
      StencilStage<Operator2,
                   IJRange<0,0,0,0>,
                   KRangeFullDomain>()
    )
  )
)
```

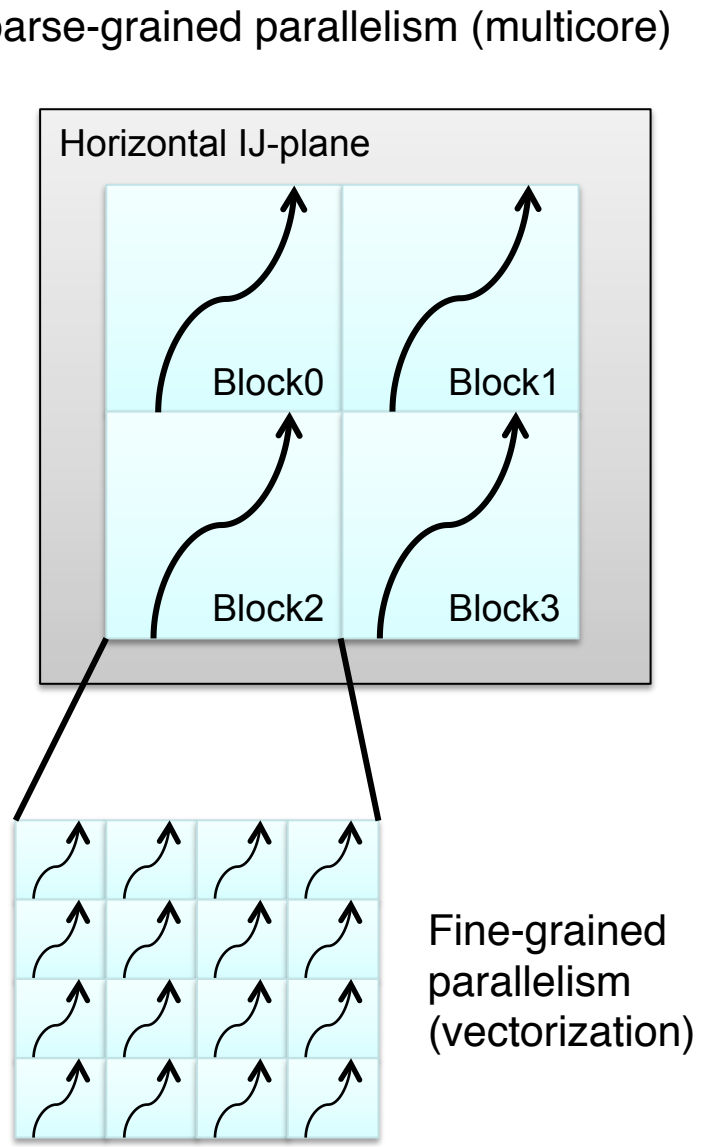
Fortran

```
DO k = 1, ke
  DO j = jstart-1, jend+1
    DO i = istart-1, iend+1
      a(i,j,k) = Operator1
    ENDDO
  ENDDO
  DO j = jstart, jend
    DO i = istart, iend
      ! Operator2 = f(a)
    ENDDO
  ENDDO
ENDDO
```

Parallelization Model

- **Multi-node parallelization (MPI)**
- Shared memory parallelization
  - Support for 2 levels of parallelism
- **Coarse-grained parallelism**
  - Split domain into blocks
  - Distribute blocks to cores
  - No synchronization & consistency required
- **Fine-grained parallelism**
  - Update block on a single core
  - Lightweight threads / vectors
  - Synchronization & consistency required

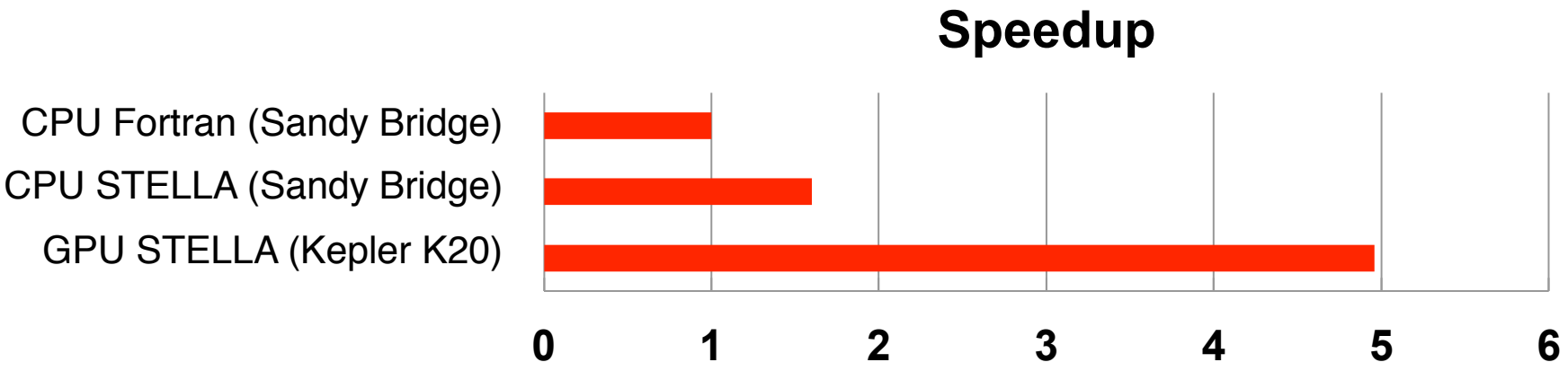
Similar to CUDA programming model
(is a good match for other platforms as well)



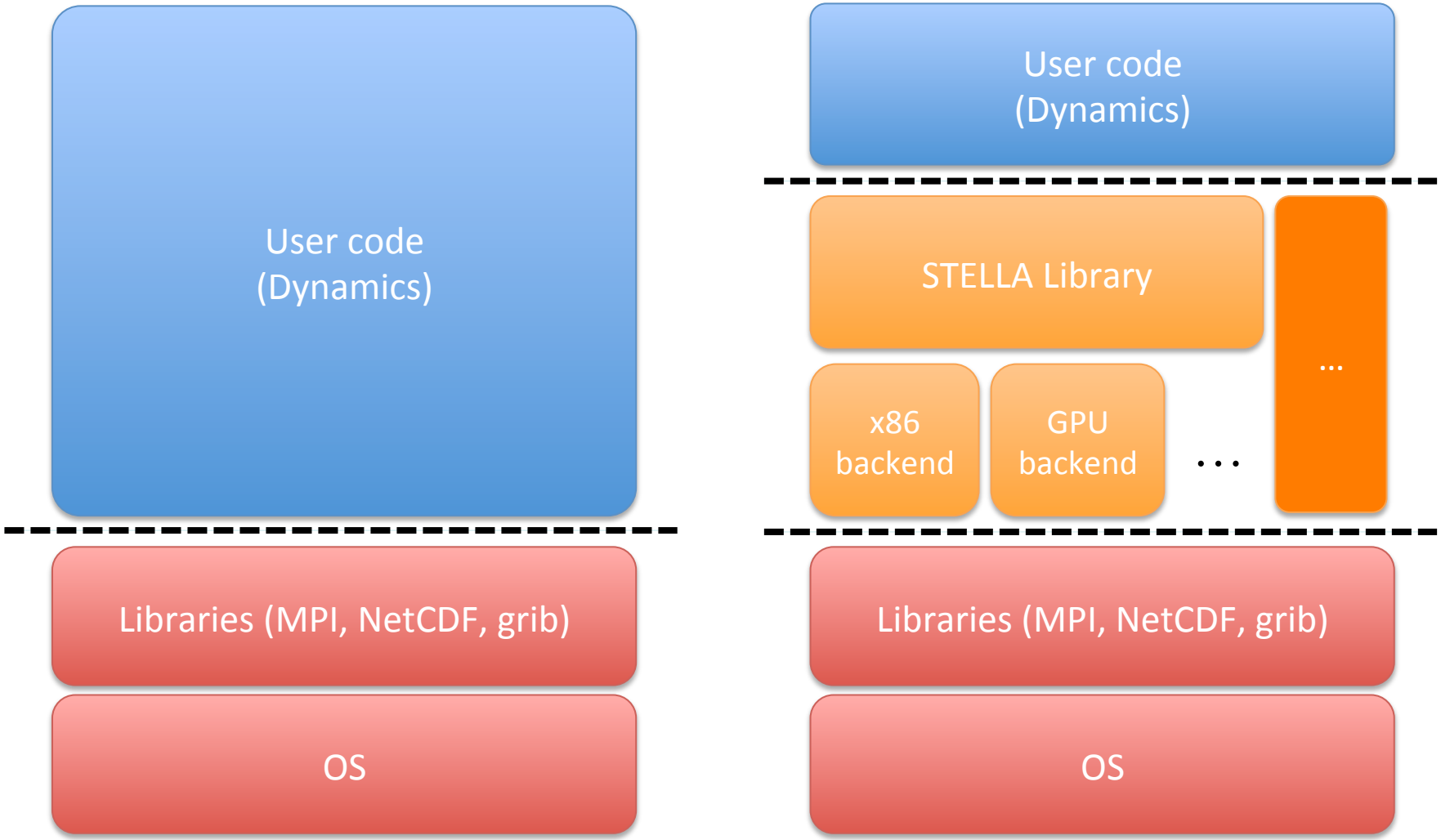
Dynamics Performance

- Test domain: 128 x 128 x 60 on a single CPU / GPU
- **CPU** (OpenMP, kji-storage)
  - Factor 1.6x – 1.7x faster than original COSMO
  - No explicit use of vector instructions (up to 30% improvement)
- **GPU** (CUDA, ijk-storage)
  - CPU vs. GPU is a factor 3.1x faster (SB vs. K20c)
  - Ongoing performance optimization

A single switch in order to compile for GPU



Separation of concerns



Current status

- DSL implementation of dycore to be merged back into official COSMO version until end of 2014
- Further improvements of STELLA ongoing
- Project to generalize STELLA to other stencil codes (e.g. global models, seismic wave propagation) funded
- COSMO version running on hybrid Piz Daint for regional climate modeling to start production April 1

DSL Approach

- (Partial) separation of concerns
- Way to consolidate $\tau_{application}$ vs. $\tau_{architecture}$
- Fixed set of features
- Suggests/enforces coding conventions and styles
- Balance between generality and complexity

DSEL vs. DSL

- **Pros**
  - Re-use of host-language toolchain (e.g. compiler, …)
  - Modest development effort
  - Powerful (host language features for free)

- **Cons**
  - Inferring information for optimization is difficult
  - Lengthy syntax, boilerplate
  - Bad error reporting

DSL Discussion Points

- **Adoption**
  - Disruptive change of programming model (→ training of users)
  - Maintenance of DSL library/compiler
- **Loss of domain information**
  - Loss of abstraction on source/compiler/debugger level can be painful
- **Interoperability**
  - DSLs must interoperate with existing code base (progressive migration)
  - With other DSLs (Earth system, multi-physics)

Conclusions

- No consensus has emerged to deliver both high performance with high programmer productivity
- DSLs are an interesting way forward
- Several efforts in weather/climate…
  - ATMOL (KNMI)
  - ICON DSL (DKRZ/MPI)
  - STELLA
  - …