Dynamical Core Rewrite

Tobias Gysi
Oliver Fuhrer
Carlos Osuna

COSMO GM13, Sibiu

Fundamental question

How to write a model code which…

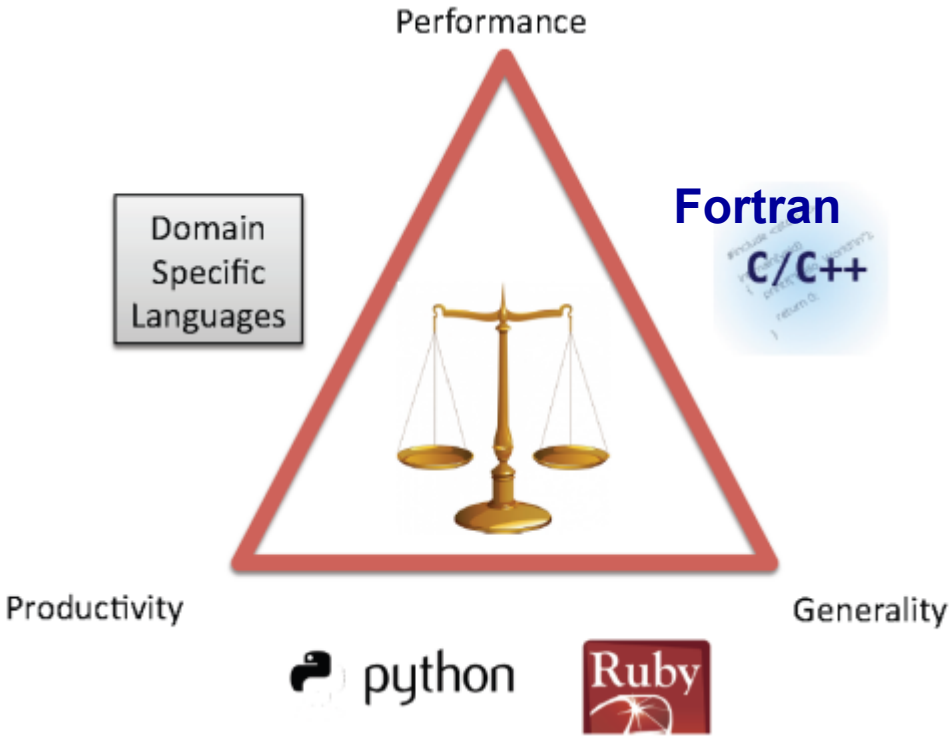
- **allows productive development by domain scientists**
- **runs efficiently on different HPC architectures**
- **continues to do so in the future**

- Clear trend in HPC architectures to become heterogeneous (GPUs, MIC, …)
- Programming models are not getting simpler (OpenMP, OpenACC, NEC directives, software managed memory, …)
- Accelerators are an attractive alternative for COSMO, but we will always want to run on a plain CPU machine

It is not clear how to solve this with the current COSMO code!

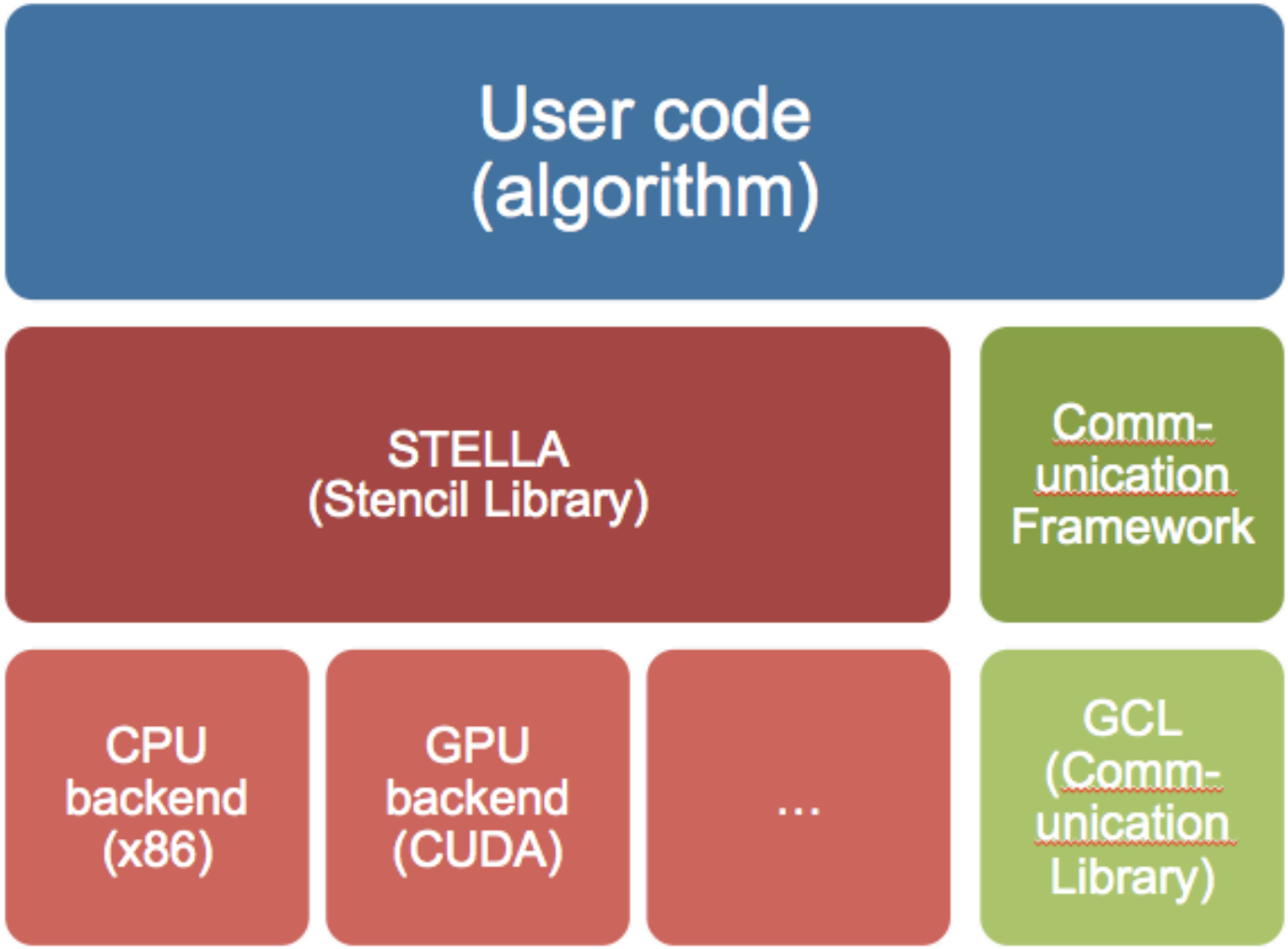
Stencil Library (DSEL)

- Separate user code (algorithm) from hardware specific implementation (optimization)



This is a fundamental change for COSMO code with distinct pros / cons

Library approach



STELLA usage

```
  DO k = 1, ke
!CDIR OUTERUNROLL=4
    DO   j  = 2 , je-1
!CDIR ON_ADB(lap)
!CDIR ON_ADB(s)
      DO i  = 2 , ie-1
        lap (i,j,k)   = s (i+1,j,k) + s (i-1,j,k) - 2.0_ireals*s(i,j,k) &
                      +  crlato(j)*(s(i,j+1,k) - s(i,j  ,k))           &
                      -  crlatu(j)*(s(i,j  ,k) - s(i,j-1,k))
      ENDDO
    ENDDO
  ENDDO
```

STELLA usage

```
__ACC__
static T Do(Context ctx)
{
    ctx[data_out::Center()] = - (T)2.0 * ctx[data_in::Center()]
      + ctx[data_in::At(iplus1)] + ctx[data_in::At(iminus1)]
      + ctx[crlatvo::Center()] * ctx[Call<Delta>::With(jplus1 , data_in::Center())]
      + ctx[crlatvu::Center()] * ctx[Call<Delta>::With(jminus1, data_in::Center())];
}
```

```
// setup the tracer stencil
StencilCompiler::Build(
    stencil_,
    "HorizontalDiffusionTracers",
    dycoreRepository.calculationDomain(),
    StencilConfiguration<Real, HorizontalDiffusionTracersBlockSize>(),
    pack_parameters(
        Param<data_out, cInOut>(data_out_),
        Param<data_in, cIn>(data_in_),
    ),
    concatenate_sweeps(
        define_sweep<cKIncrement>(
            define_stages(
                StencilStage<LapStage, IJRange<cComplete,-2,2,-2,2>,
                                       KRange<FullDomain,0,0> >(),
            )
        )
    )
);
```

Documentation & Publications

- Documentation
  - Stencil library (implementation)
  - Communication framework
  - Serialization framework
  - Style-guide

- Stencil library workshop material (“users guide”)

- See http://hpcforge.org/

- Publications
  - Gysi et al. 2013 (in preparation)

Effort to learn STELLA

- C++ and STL knowhow required
- No need to learn OpenMP, CUDA, template-metaprogramming

- Users
  - Tobias Gysi
  - Carlos Osuna
  - Oliver Fuhrer
  - Ben Cumming (PostDoc, support, indirect addressing)
  - Men Muheim (PhD, horizontal diffusion)
  - Florian Dörfler (MSc, vertical diffusion, Coriolis)
  - Katharina Riedinger (BSc, Bott advection)
  - Andrea Arteaga (BSc, workshop)
  - Kevin Wallimann (BSc, semi-Lagrangian advection)
  - Michael Baldauf (PhD, new FW-solver)
  - Nicolò Lardelli (MSc, horizontal turbulent diffusion)
  - Joseph Charles (PhD, microphysics)
  - Xavier Lapillonnne (moisture divergence)

STELLA Support

- Library is in good shape
- Handover from Tobias to Ben
- Maintenance tasks
  - Incremental improvements (ijk-caching, indirect addressing, usability, …)
  - Bugs
  - Gatekeeper for modifications
- Support tasks
  - Last level troubleshooting

Recent Developments – STELLA

- **Functionality improvements**
  - Refactoring temporary fields (Ben Cumming)
  - Dynamic indexing (Ben Cumming)
  - Switch case DSEL extension which supports runtime switches (Tobias Gysi)

- **Performance improvements**
  - Re-implementation of software managed caching (Tobias Gysi)
  - Do not allocate temporary fields which are not accessed due to caches (Tobias Gysi)

Recent Developments - Dycore

- **Additional dycore features**
  - Relaxation (Carlos Osuna)
  - Saturation Adjustments (Carlos Osuna)
  - Strang splitting support for AdvectionPD (Tobias Gysi)
  - Moisture divergence (Xavier Lapillonne)
  - Other developments are work in progress, e.g. other advection schemes

- **Missing features**
  - Only RK-core considered
  - new FW-solver
  - Several options (PD-advection, advection order, …)
  - …

Software Managed Caching

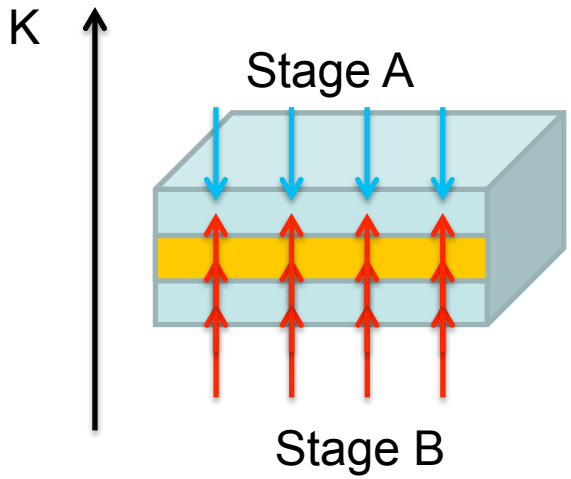
- Software controlled cache management
  - Data is placed explicitly in the caches
  - Data is removed from the caches if it is not needed anymore
- Application knowledge helps to used small caches efficiently
  - Do not cache fields accessed once
  - Do not write back modified cache lines not accessed anymore
  - Immediately remove data from cache if it is not used anymore

- We have implemented software managed caching for the GPU backend

Software Managed Cach

Important for efficient communication between K loop levels

- Sliding window in K direction is stored in GPU registers
- Used to buffer multiple K levels no accesses in IJ
- Optionally filled and / or flushed from the underlying field
- E.g. buffer all intermediate fields of z advection




```
/* stage A */
temp(i,j,k+1) = 2.0;

/* stage B */
result(i,j,k) =
    temp(i,j,k+1) +
    temp(i,j,k) +
    temp(i,j,k-1);
```

Usage Example

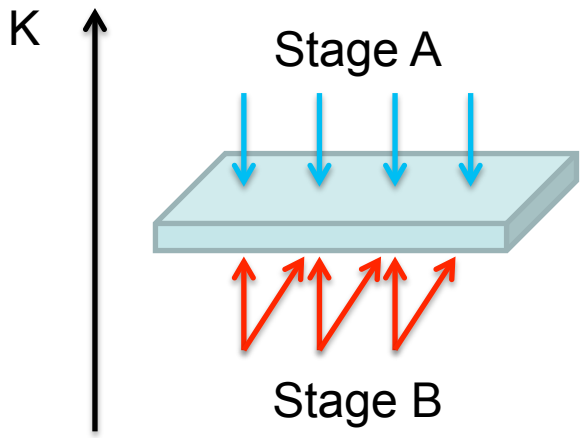
Keep all AdvectionPDZ intermediates in registers:

```
define_loops(
  define_sweep<cKIncrement>(
    define_caches(
      KCache<data_in, cFill, KWindow<-2,1>, KRange<FullDomain,0,0> >(),
      KCache<sqrtgrs, cFill, KWindow<-2,2>, KRange<FullDomain,0,0> >(),
      KCache<icr, cLocal, KWindow<-1,2>, KRange<FullDomain,-2,2> >(),
      KCache<wcfrac, cLocal, KWindow<-1,2>, KRange<FullDomain,0,1> >(),
      KCache<fip, cLocal, KWindow<-1,2>, KRange<FullDomain,0,1> >(),
      KCache<fim, cLocal, KWindow<-1,2>, KRange<FullDomain,0,1> >(),
      KCache<fcr, cLocal, KWindow<-1,2>, KRange<FullDomain,0,1> >(),
      KCache<flux, cLocal, KWindow<-2,1>, KRange<FullDomain,0,1> >()
    ),
    define_stages(
      StencilStage<ICRStage, IJRange<cComplete,0,0,0,0>, KRange<FullDomain,-2,0> >(),
      StencilStage<FIPAndFIMStage, IJRange<cComplete,0,0,0,0>, KRange<FullDomain,-1,0> >(),
      StencilStage<FCRStage, IJRange<cComplete,0,0,0,0>, KRange<FullDomain,1,1> >(),
      StencilStage<FluxStage, IJRange<cComplete,0,0,0,0>, KRange<FullDomain,2,2> >(),
      StencilStage<DataStage, IJRange<cComplete,0,0,0,0>, KRange<FullDomain,2,2> >()
    )
  )
)
```

Software Managed Caching – IJCache

Important for efficient communication between threads

- IJ plane is stored in shared memo
- Used to buffer multiple IJ positions no accesses in K
- Simple local cache not filled and / or flushed from the underlying field
- E.g. buffer intermediate values of horizontal diffusion




```
if(/* in range of stage A */)
{
    temp(i,j,k) = 2.0;
}
__syncthreads();

if(/* in range of stage B */)
{
    result(i,j,k) =
        temp(i,j,k) +
        temp(i+1,j,k);
}
```

Software Managed Caching – IJCache Example

Keep all HorizontalDiffusion intermediates in shared memory:

```
define_loops(
  define_sweep<cKIncrement>(
    define_caches(
      IJCache<lap, KRange<FullDomain,0,0> >(),
      IJCache<flx, KRange<FullDomain,0,0> >(),
      IJCache<fly, KRange<FullDomain,0,0> >(),
      IJCache<rxp, KRange<FullDomain,0,0> >(),
      IJCache<rxm, KRange<FullDomain,0,0> >()
    ),
    define_stages(
      StencilStage<LapStage, IJRange<cComplete,-2,2,-2,2>, KRange<FullDomain,0,0> >(),
      StencilStage<FluxStage, IJRange<cComplete,-2,1,-2,1>, KRange<FullDomain,0,0> >(),
      StencilStage<RXStage, IJRange<cIndented,-1,1,-1,1>, KRange<FullDomain,0,0> >(),
      StencilStage<LimitFluxStage, IJRange<cIndented,-1,0,-1,0>, KRange<FullDomain,0,0> >(),
      StencilStage<DataStage, IJRange<cComplete,0,0,0,0>, KRange<FullDomain,0,0> >()
    )
  )
)
```

Benchmarks

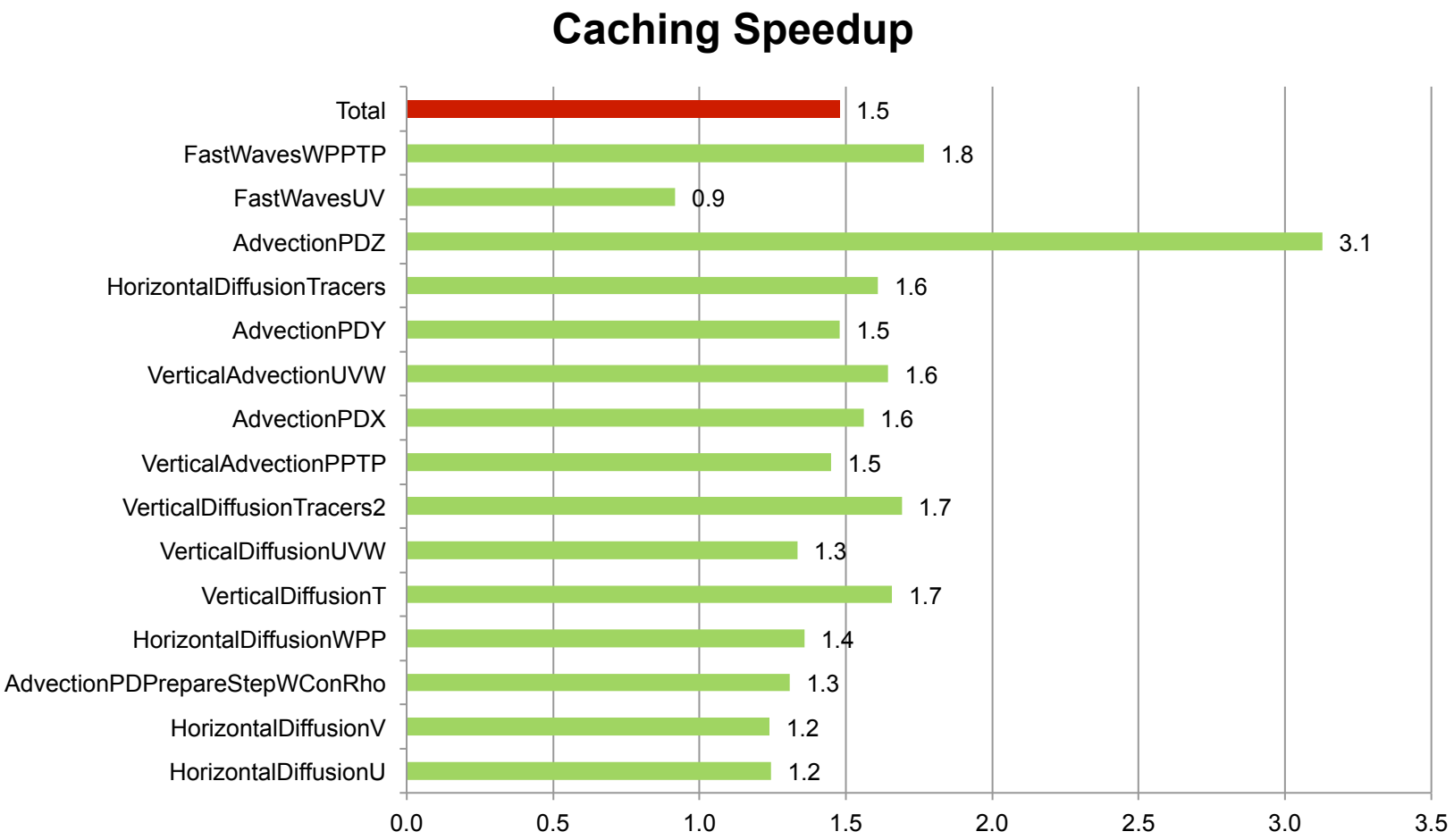
- **CPU performance** was measured on a single socket of Piz Daint
  - Sandy Bridge E5-2670 @ 2.60GHz
  - With hyperthreading

- **GPU performance** was measured on my Windows PC
  - Tesla K20c (roughly 10-20% slower than a K20x)
  - ECC on
  - CUDA 5.5

- Single node measurements on a 128 x 128 data set

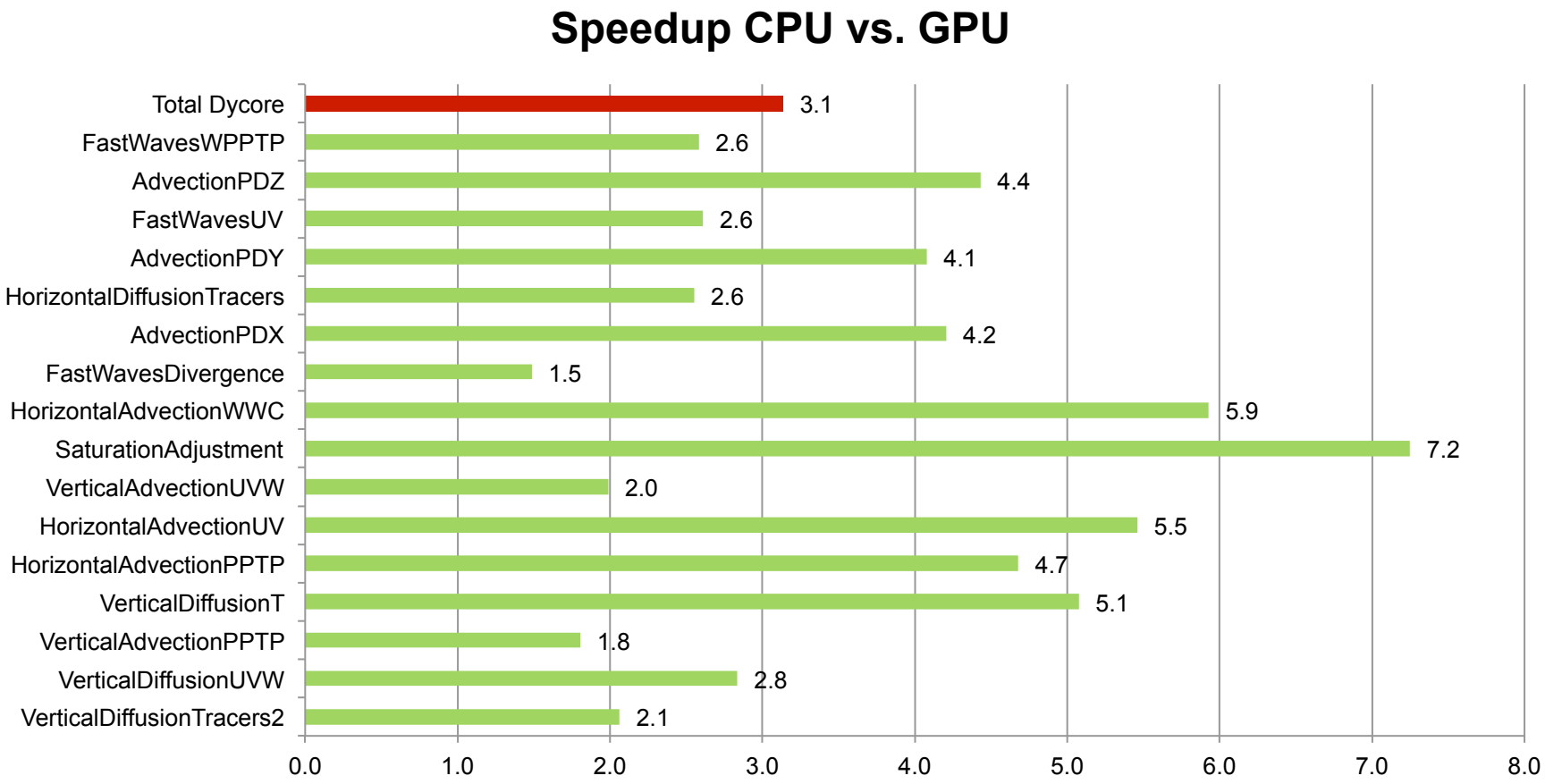
Benchmark – Caching vs. no Caching

Caching improves the GPU performance by a factor 1.5x



Benchmark – CPU vs. GPU

A tesla K20c shows a 3.1x speedup over a Sandy Bridge socket



Next Steps…

- Dynamical core
  - Implement important missing parts
- STELLA library
  - Continuous development
    - Usability
    - ijk-caches
    - Tuning of GPU backend
  - Next generation
    - Generalization
    - Block-structured / unstructured grids

Thank you!

- Questions?