C2SM Center for Climate Systems Modeling

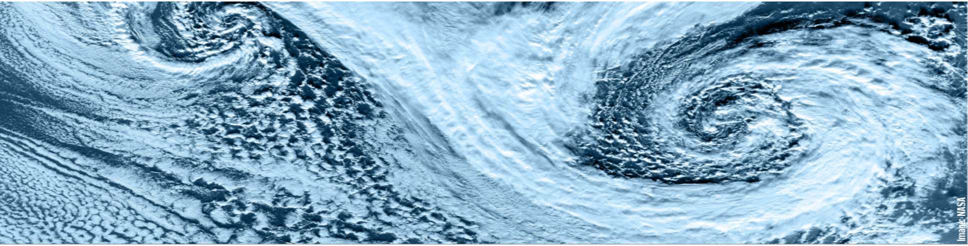
ETH zürich

Adapting Weather and Climate Models to Hybrid Architecures

Carlos Osuna, C2SM (ETH)
carlos.osuna@env.ethz.ch

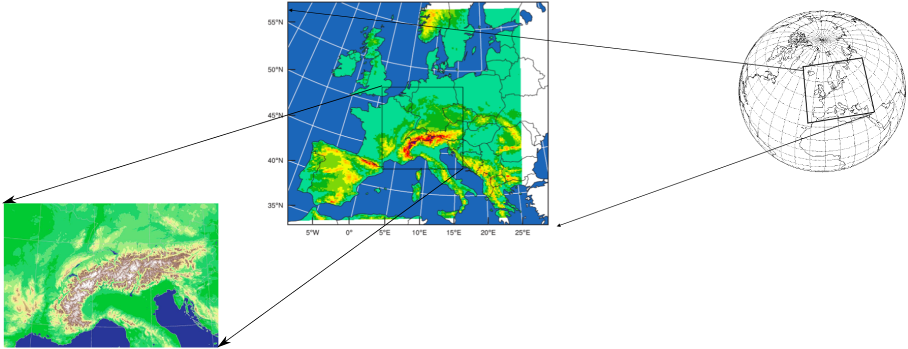
Andrea Arteaga, Oliver Fuhrer, Tobias Gysi, Xavier Lapillonne, Pascal Spoerri, Thomas Schulthess

PPAM 2015



COSMO Model

- COSMO is a regional atmospheric model used for:
  1. numerical weather prediction at 10 national weather services
  2. climate research studies at ∼50 universities



Motivation for porting COSMO to Accelerators

- Strict operational requirements for time to solution (time-compression factor ∼70 for MeteoSwiss) and costs of computing systems.
- However, strong interest in scientific community for increasing computational cost:

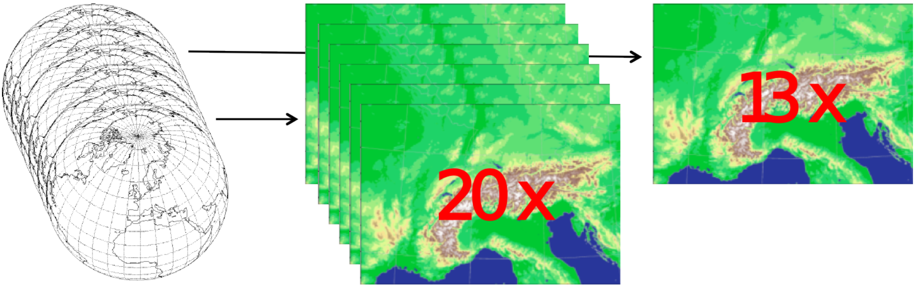
  1. High resolution (1 km horizontal resolution) weather forecast
  2. Ensemble weather forecast
  3. Cloud resolving climate simulations (2.2 km resolution) over the alps.

Scientific Challenges in COSMO

ECMFW-Model
18 / 9 km gridspacing
4x per day

COSMO-E
2.2 km gridspacing
582x390x60 gridpoints
2 x per day

COSMO-1
1.1 km gridspacing
1158 x 774 x 80
gridpoints
8 x per day



Next-generation system

Accounting for Moore's law (factor 4)

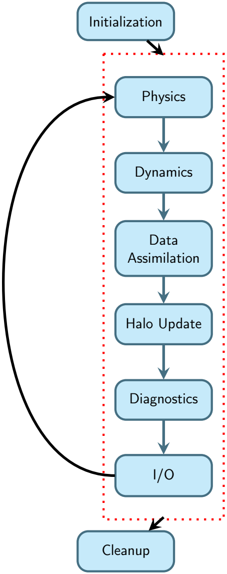
Motivation for porting COSMO to Accelerators

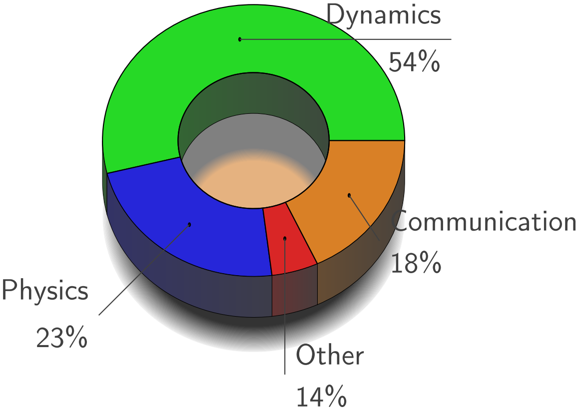
- Strict operational requirements for time to solution (time-compression factor $\sim$70 for MeteoSwiss) and costs of computing systems.
- However, strong interest in scientific community for increasing computational cost:
  1. High resolution (1 km horizontal resolution) weather forecast
  2. Ensemble weather forecast
  3. Cloud resolving climate simulations (2.2 km resolution) over the alps.
- Larger memory bandwidth of accelerators makes GPUs attractive computing architectures for memory bound codes:
  E5-2670 (Q1/2012) -> 51.2 GB/s vs K20X (Q4/2012) -> 250 GB/s

COSMO was fully ported to GPUs (work funded by HP2C initiative: DOI: 10.14529/jsfi140103)

COSMO Components

$\sim$ 400 k lines of code

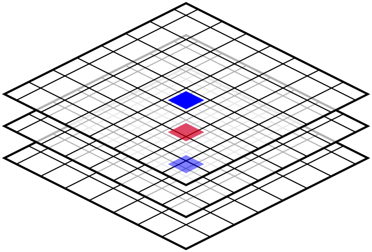




Physics

C2SM Center for Climate Systems Modeling

- Parametrized equations of physical processes not resolved at grid scale.
- Large codes
- Relatively simple stencil patterns in vertical columns (tridiagonal solves, pentadiagonal solves,...)



Physical Parametrizations were ported to GPUs using OpenACC

Physics

- Ported to GPU using OpenACC, retains portable Fortran code
- Fully optimized version requires some restructuring (loops, data layout)

CPU Optimized

```
do k=2,nk
  !$acc parallel
  !$acc loop gang vector
  do i=1,ni
    some code 1 ...
    c(i) = D*exp(a(i,k-1))
  end do
  !$acc end parallel
  !$acc parallel
  !$acc loop gang vector
  do i=1,ni
    a(i,k)=c(i)*a(i,k)
    some code 2 ...
  end do
!$acc end parallel
end do
```

GPU Optimized

```
!$acc parallel
!$acc loop gang vector
do k=2,nk
  do i=1,ni
    some code 1 ...
    zc=D*exp(a(I,k-1))
    a(I,k)=c(i)*a(I,k)
    some code 2 ...
  end do
end do
!$acc end paralle
```

Dynamical Core

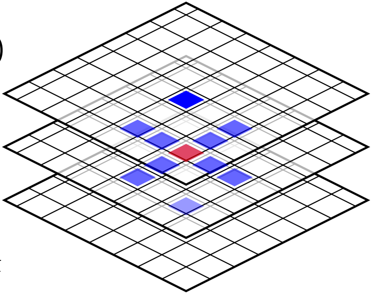
- Solves the Navier Stokes equations using finite difference methods on structured grids

$$\rho = \frac{d\mathbf{v}}{dt} - \nabla p + \rho \mathbf{g} - 2\boldsymbol{\Omega} \times (\rho \mathbf{v})$$

$$\frac{d\rho}{dt} = -\rho \nabla \cdot \boldsymbol{v}$$

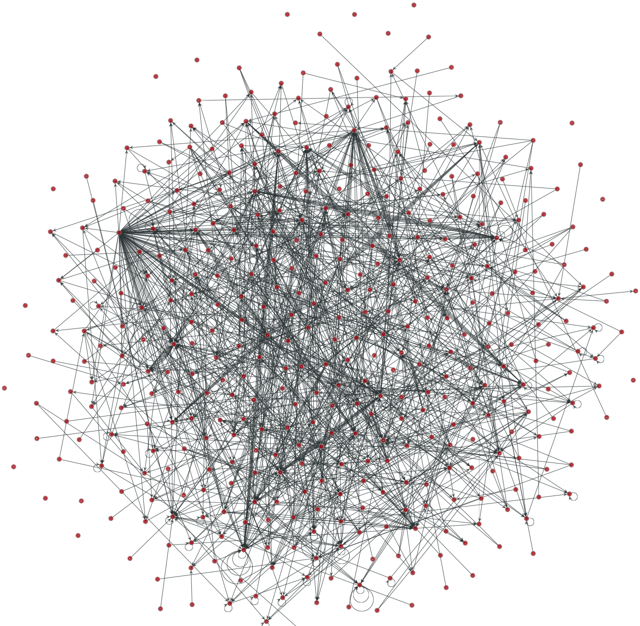
$$\rho \frac{dq^x}{dt} = -\nabla \cdot \boldsymbol{J}^x + I^x$$

$$\rho \frac{de}{dt} = -\rho \nabla \cdot \boldsymbol{v} - \nabla \cdot (\boldsymbol{J}_e + \boldsymbol{R}) + \epsilon$$



- Explicit discretization schemes produce large stencils in the horizontal (depending on the order)
- Vertical operators implicitly solved produce tridiagonal systems

Dynamical Core was ported to GPUs using STELLA DSL library



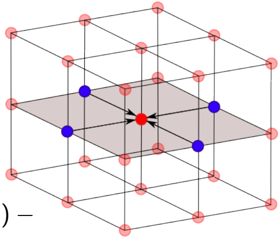
STELLA

- STELLA is a DSL for stencil codes on structured grids written in C++ (template metaprogramming).
- Single source code for multiple architectures, performance portable
- Separation of concerns: separates model and algorithm from hardware specific implementation and optimizations.

Separation Of Concerns

User description of mathematical model

$$\frac{\partial U}{\partial t} = -\alpha \nabla^2(\nabla^2 U)$$



$$\begin{aligned} lap(i,j) &= 4u(t,i,j) - u(t,i+1,j) - u(t,i-1,j) - \\ &\quad u(t,i,j+1) - u(t,i,j-1) \\ u(t+1,i,j) &= 4lap(t,i,j) - lap(t,i+1,j) - lap(t,i-1,j) - \\ &\quad lap(t,i,j+1) - lap(t,i,j-1) \end{aligned}$$

Separation Of Concerns

C2SM Center for Climate Systems Modeling

Generated Kernel for GPU

```
const int i = threadIdx.x;
const int j = threadIdx.y;
int i_h = 0;
int j_h = 0;

if(j < 2)
{
  i_h = i;
  j_h = (j==0 ? −1 : blockDim.y);
}
else if(j < 4 && i <= blockDim.y)
{
  i_h = (j==2 ? −1 : blockDim.x);
  j_h = i;
}

for(int k=0; k < kdim; ++k)
{
  lap(i,j) = − 4.0 ∗ phi(i,j,k)
    + phi(i+1,j,k) + phi(i−1,j,k)
    + phi(i,j+1,k) + phi(i,j−1,k);

  if(i_h != 0 || j_h != 0)
    lap(i_h, j_h) =
      − 4.0 ∗ phi(i_h,j_h,k)
      + phi(i_h+1,j_h,k) + phi(i_h−1,j_h,k)
      + phi(i_h,j_h+1,k) + phi(i_h,j_h−1,k);
  __syncthreads();
  flx(i,j,k) = lap(i+1,j,k) − lap(i,j,k);
  fly(i,j,k) = lap(i,j+1,k) − lap(i,j,k);
  if(i_h < 0)
    flx(i_h,j_h,k) = lap(i_h+1,j_h,k) −
                     lap(i_h,j_h,k);
  if(j_h < 0)
    fly(i,j_h,k) = lap(i,j_h+1,k) −
                   lap(i,j_h,k);
  __syncthreads();
  result(i,j) = phi(i,j,k) − alpha(i,j,k)∗(
    flx(i,j,k) − flx(i−1,j,k) +
    fly(i,j,k) − fly(i,j−1,k));
}
```

STELLA Syntax

```
template<typename TEnv>
struct Divergence {
  STENCIL_STAGE(TEnv)

  STAGE_PARAMETER(FullDomain, phi)
  STAGE_PARAMETER(FullDomain, lap)
  STAGE_PARAMETER(FullDomain, flx)

  static void Do(Context ctx, FullDomain) {
    ctx[div::Center()] = ctx[phi::Center()] −
      ctx[alpha::Center()] ∗ (ctx[flx::Center() −
      ctx[flx::At(iminus1)] + ctx[fly::Center() −
      ctx[fly::At(jminus1)] )
  }
};
```

```
IJKRealField dataIn, dataOut;

Stencil stencil;
StencilCompiler::Build(
  stencil,
  pack_parameters(
    Param<res, cInOut>(dataOut),
    Param<phi, cIn)(dataIn)
    Param<alpha, cIn)(dataAlpha)
  ),
  define_temporaries(
    StencilBuffer<lap, double>(),
    StencilBuffer<flx, double>(),
    StencilBuffer<fly, double>()
  ),
  define_loops(
    define_sweep<cKIncrement>(
      define_stages(
        StencilStage<Lap, IJRange<cIndented,−1,1,−1,1> >(),
        StencilStage<Flx, IJRange<cIndented,−1,0,0,0> >(),
        StencilStage<Fly, IJRange<cIndented,0,0,−1,0> >(),
        StencilStage<Divergence, IJRange<cComplete,0,0,0,0>
      )
    )
  )
);
```

GPU Optimizations

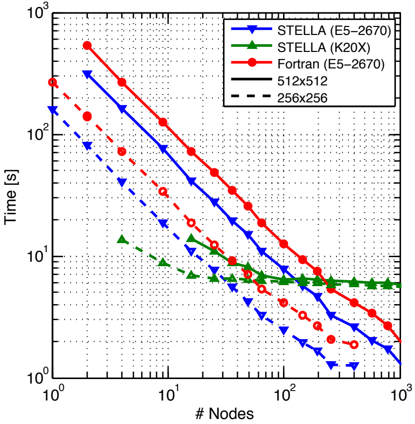
Dynamical Core speedup (vs fortran legacy) 1.8x (CPU) and 5.8x (GPU)

How to further exploit GPU optimization without changing application, incorporating new STELLA syntax elements?

- K parallelization
- Parallel Tridiagonal Solve
- Software Manage Caching

Strong Scaling for COSMO @GPUs

- GPUs show poor scalability beyond 64x64 grid points per domain, due to lack of parallelism

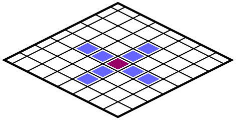


Strong scaling curves for the dynamical core of COSMO: "STELLA: A domain-specific tool for structured grid methods in weather and climate models", Proceedings of SuperComputing 2015

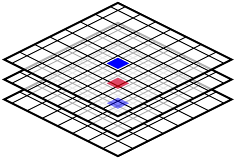
Improving Strong Scaling

- STELLA adds a syntax element that integrates new parallelization modes for the GPU backend:

1. a *k-parallel* mode which parallelizes over the vertical dimension, for stencils with only data dependencies in the horizontal



1. a parallel tridiagonal solver for tridiagonal systems that results from vertically-implicit discretizations.

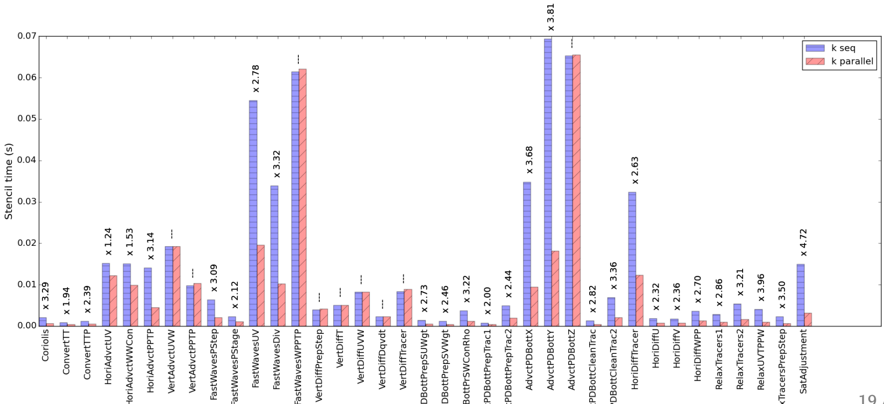


K Parallelization

C2SM Center for Climate Systems Modeling

- A STELLA keyword triggers a k parallelization mode, that increases the level of parallelism for GPUs

```
define_loops(
  define_sweep<cKParallel>(
    define_stages(
      StencilStage<Lap, IJRange<cIndented,−1,1,−1,1> >()
    )
  )
)
```



Parallel Tridiagonal Solve

- Vertical implicitly solved operators in the dynamical core generates tridiagonal systems which are solved using sequential Thomas algorithm

Forward Sweep	$c_k = \frac{c_k}{b_k - c_{k-1} a_k}$	$d_k = \frac{d_k - d_{k-1} a_k}{b_k - c_{k-1} a_k}$	$k = 1, ..., n$
Backward Sweep	$x_n = d_n$	$x_k = d_k - c_k x_{k+1}$	$k = n-1, ..., 1$

Parallel Tridiagonal Solve

C2SM Center for Climate Systems Modeling

- STELLA integrates a parallel tridiagonal solve that improves the performance at strong scaling compared to sequential algorithms
- HPCR solver provided by Jeremy Appleyard (NVIDIA), Mike Giles: "GPU implementation of finite difference solvers"

```
template<typename TEnv>
struct SetupStage
{
  STENCIL_STAGE(TEnv)

  static void Do(Context ctx, FullDomain) {
    ctx[ hpcr_acol ::Center()] = ...
    ctx[ hpcr_bcol ::Center()] = ...
    ctx[ hpcr_ccol ::Center()] = ...
    ctx[ hpcr_dcol ::Center()] = ...
  }
};
```

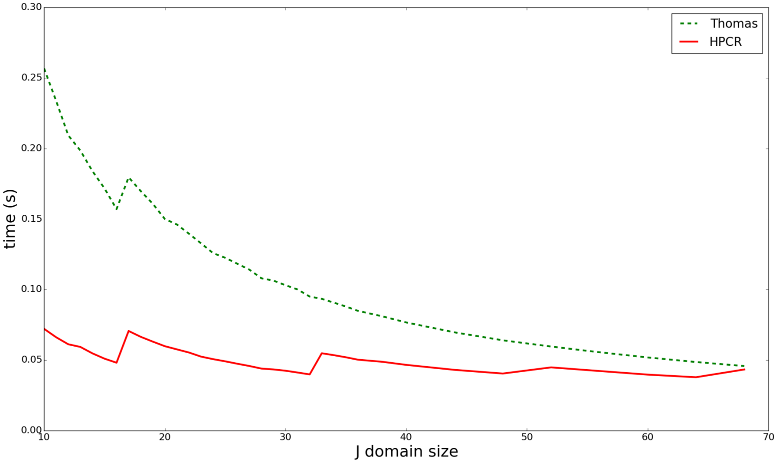
- compute of matrix and RHS coefficients using STELLA DSL

```
StencilCompiler::Build(
  StencilConfiguration<Real, TridiagSolve_BlockSize>
  pack_parameters( Param<result, cInOut>(res) ),
  define_loops(
    define_sweep<cTridiagonalSolve>(
      define_stages(
        StencilStage<SetupStage>(),
        StencilStage<TridiagonalSolveFBStage>(),
        StencilStage<WriteOutputStage>()
      )
    )
  )
);
```

- Solve tridiagonal system using library solver.

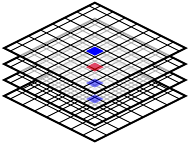
Parallel Tridiagonal Solve

time per system vs the size of J dimension (i size=32) for K20X



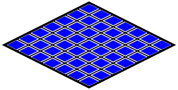
Software Managed Caching

- Leveraging data locality is of key importance for memory bound stencil codes.
- Except texture memory cache, GPU on-chip memory resources must be managed explicitly in the software.
- STELLA provides 3 type of cache syntax. The user describes access pattern and data reuse, still agnostic to hardware details



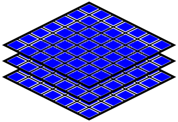
KCache: vertical data dependencies. Ring buffer in a vertical column stored in registers (private to each thread)

KCache<acol, cFlush,
  KWindow<−2,1> >()



IJCache: horizontal data dependencies. Full block stored in shared memory.

IJCache<lap, cLocal>()



IJKCache: data dependencies in a 3D box multiple levels stored in shared memory

IJKCache<div, cFill,
  IJKWindow<−1,1,−1,1,−2,0>

- Provides multiple synchronization (GMEM) policies: Local, Fill, Flush, FillAndFlush

Software Managed Caching

Cache effect on some dynamical core operators on a K20X

Stencil	cache policy	no Cache (s) (shared mem)	IJK cache (s)
AdvectionBottY	fill from mem	0.15	0.14
AdvectionBottX	fill from mem	0.077	0.044
FastWavesDivergence	local buffer	0.088	0.069

Programming Models for Hybrid Architectures

C2SM Center for Climate Systems Modeling

- We made a performance comparison between OpenACC and STELLA DSL for a horizontal diffusion and a vertical advection operators
- STELLA is faster ~2.0x for a naive (3 nested loops) implementation of OpenACC
- ~ 1.5x for an optimized OpenACC version (blocking, register caching, shared memory)

Programming Models for Hybrid Architectures

C2SM Center for Climate Systems Modeling

DSL

- retain single source code, abstracts implementation & optimization
- Optimal performance for multiple architectures (GPU, x86, XeonPhi,...)
- Change of paradigm has to be adopted by the community

OpenACC

- retain legacy (Fortran) code
- Not fully performance portable for non simple access patterns (like vertical stencils)
- Need to interoperate with other programming models for other architectures (x86, XeonPhi)...

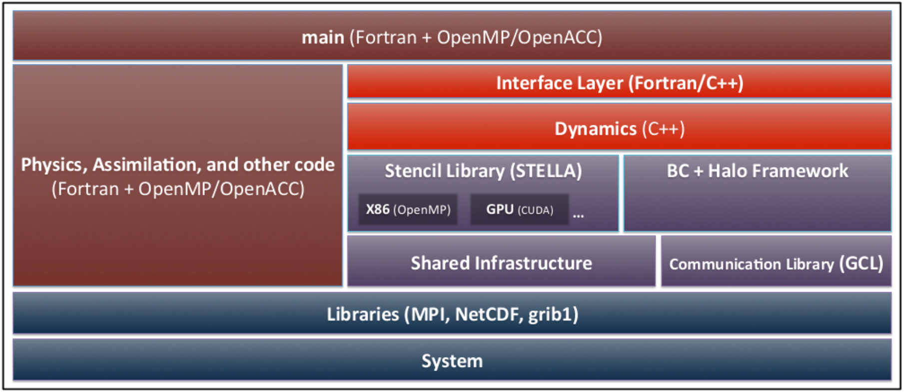
Combining multiple programming models (separated parts of the code) is probably a good compromise. But requires software infrastructure to connect data structures and programming language.

Conclusions

C2SM
Center for Climate
Systems Modeling

- COSMO fully ported to hybrid architectures using mixed programming models: OpenACC and STELLA DSL.
- Speedup factor obtained for the full model of 1.5x (CPU) and 4.5x (GPU) with respect to Fortran COSMO
- DSL power was exploited by further backend optimizations without modifying user code.
- Combing multiple levels of abstractions show clear benefits... but also an indication that the we still dont have a perfect programming model

BACKUPS


main (Fortran + OpenMP/OpenACC)
Physics, Assimilation, and other code
(Fortran + OpenMP/OpenACC)
Interface Layer (Fortran/C++)
Dynamics (C++)
Stencil Library (STELLA)
X86 (OpenMP)
GPU (CUDA)
...
BC + Halo Framework
Shared Infrastructure
Communication Library (GCL)
Libraries (MPI, NetCDF, grib1)
System