Portability and Scalability of the COSMO Weather and Climate Model on Heterogeneous Architectures

Carlos Osuna\textsuperscript{1}, Prof. T. Hoefler\textsuperscript{2}
O. Fuhrer\textsuperscript{1}, X. Lapillonne\textsuperscript{1}, S. Moosbrugger\textsuperscript{2}, F. Thaler\textsuperscript{3},
J. VandeVondele\textsuperscript{3}, T. Wicky\textsuperscript{2}

\textsuperscript{1}MeteoSwiss, \textsuperscript{2}SPCL ETH, \textsuperscript{3}CSCS ETH
Outline

• Regional numerical weather prediction: state of the art

• Portability of NWP models: COSMO

• Task parallelism for strong scalability of weather models

• Conclusions
Regional Weather Forecast: State of the Art
Weather model – COSMO – 1 km

19 Jun 2013 01:46:00
Reality (July 20, 2013)
Can weather forecast improve with more computational resources?
Increase resolution

- Improve model representation and physical processes

2 x horizontal resolution ≈ 8-10 x computational cost
Ensemble prediction

- Run N time the same forecast with perturbation
- Assess variability of weather situation
- Probabilistic forecast and warning

computational cost = N x
Moore’s law is over, what do we do?

K. Flamm 2017, IEEE Computing in Science & Engineering

Physical limits
- 2004 end of frequency scaling → multi-core
- 2012 end of rapid cost decline → constant $/transistor
- 2018 heat dissipation constraints → dark silicon, end of multi-core
- 2021 end of reduction in feature size → ?

Specialized hardware
increased on chip parallelism

GPUs
Many-core processors
AI Accelerators (e.g. Google Tensor Flow, Intel Nervana)
FPGAs
Current weather models are achieving low percentage of peak performance of modern computing architectures.

And incapable of running on modern accelerators.

COSMO only NWP model running operationally on heterogeneous architecture [PASC and HP2C initiatives]
Moore’s law is over, what do we do?

K. Flamm 2017, IEEE Computing in Science & Engineering

**Physical limits**
- 2004 end of frequency scaling $\rightarrow$ multi-core
- 2012 end of rapid cost decline $\rightarrow$ constant $\$/transistor
- 2018 heat dissipation constraints $\rightarrow$ dark silicon, end of multi-core
- 2021 end of reduction in feature size $\rightarrow$ ?

PASCHA PASC: Portability and Scalability of the COSMO Weather and Climate Model on Heterogeneous Architectures
The COSMO model on GPU

- Local area numerical weather prediction and climate model (cosmo-model.org), 350 KLOC F90 + MPI
- Full GPU port strategy: avoid GPU-CPU data transfer
- Approach: OpenACC directives + Domain Specific Language (DSL) re-write
Why is it so hard to efficiently use modern architectures for weather and climate models?
Computational patterns of weather models

- Compact horizontal stencil computations on 3D gridded fields
- Implicit solvers in the vertical dimension (tridiagonal solve)

- Large collection of libraries and solutions optimized for multiple architectures: Halide, YASK, Blitz, OpenFOAM, TensorFlow, cuSPARSE, etc.

Why is NWP and climate stuck with 100k-1M LOC of inefficient and non portable Fortran codes?
The dynamical core of COSMO dissected
Why is it so hard to efficiently use modern architectures for weather and climate models?

- Diverse set of many different computations on a large set of prognostic variables
- Special treatment of vertical dimension (not 3D)
Horizontal Advection on a GridTools DSL

Each hyper-node of the DAG is a GPU kernel or OMP parallel computation

```cpp
struct UVStage {
    using u_tens = inout_accessor<0>;
    using v_tens = inout_accessor<1>;
    using u = in_accessor<2, extent<-3,3,-3,3>>;
    using v = in_accessor<3, extent<-3,3,-3,3>>;
    //....
    template< typename Evaluation >
    GT_FUNCTION static void Do(Evaluation& eval) {
        Real uatupos = SymmetricAverage<X, 1>(eval, u_stage());
        Real vatupos = Average<X, 1, NestedAverage<Y, -1>>(eval, v_stage());
        Real uavg = uatupos * eval(acrlat0());
        Real vavg = vatupos * EARTH_RADIUS_RECIP;
        eval(u_tens()) = call<HorizontalAdvectionDriver>::with(eval, u_stage(),
                          uavg, vavg, eddlat0(), eddlon0()) + eval(tglatda0() * u_stage()) * vatupos;
        // same for V
    }
};
```
Horizontal Advection on a GridTools DSL

- No loops
- No storage layout
- No parallelization
- GPU & CPU backends.
- Portable dynamical core codes
- Loop nesting, data locality, prefetching, etc optimizations internal to library

```cpp
struct UVStage {
    using u_tens = inout_accessor<0>
    using v_tens = inout_accessor<1>
    using u = in_accessor<2, extent<-3,3,-3,3>, extent_from<3,3-3,3>>
    using v = in_accessor<3, extent<-3,3-3,3>>;
    //...
    template<typename Evaluation> GT_FUNCTION static void Do(Evaluation& eval) {
        Real uatupos = SymmetricAverage<X, 1>(eval, u_stage());
        Real vatupos = Average<X, 1, NestedAverage<Y, -1>>(eval, v_stage());
        Real uavg = uatupos * eval(acrlat0());
        Real vavg = vatupos * EARTH_RADIUS_RECIP;
        eval(u_tens()) = call<HorizontalAdvectionDriver>::with(eval, u_stage(), uavg, vavg, eddlat(), eddlon()) + eval(tgrlatda0() * u_stage()) * vatupos;
        // same for V
    }
};
```
COSMO on KNL & AVX-512

- Goal: prepare COSMO for KNL and AVX-512
- Constraints: retain a portable version of the code for NVIDIA GPUs and KNL / AVX-512

- COSMO dynamical core is operational using STELLA DSL at MeteoSwiss.
- STELLA is being replaced by GridTools.

- GridTools backend fully developed (at CSCS) for KNL/AVX-512 (no change in dycore required)
- Subset of physical parametrizations ported to KNL
- ~10 PM effort [CSCS PASC, PASCHA PASC & IPCC projects]
Developing a backend for GridTools DSL

Simple stencils are used to evaluate the bandwidth obtained for KNL on 3D stencils for the optimized KNL strategy (memory layouts, looping, prefetching, etc)
Evaluation of KNL on COSMO dynamical core operators

Horizontal Diffusion

Vertical Advection

Higher is better
Evaluation of KNL on COSMO physical parameterizations

Fortran + OpenMP

Lower is better
Strong Scalability

- Fixed domain size
- Fixed time to solution

How to increase the problem? Increase resolution?

CFL requires $\Delta x / \Delta T$ constant: 2x resolution increase -> $\frac{1}{2}$ time step.

Increasing resolution requires even more strong scalability (keep time-to-solution)
Task parallelism

• Task parallelism is the only way to increase parallelism when data parallelism is not enough.

• Weather and climate models have multiple components that «could» run in parallel: physical parametrizations (turbulence, radiation, microphysics), dynamical core, etc.

• Task parallelism did not show much impact in performance on traditional CPUs.

• BUT, It will have a large impact on massively parallel accelerators
Dynamical core of COSMO DAG
Advection operators: fused vs parallel

horizontal advection

vertical advection

fused

Horizontal advection

vertical advection
- **Fusion** is an important optimization that increases **data locality**.
- All accesses (edges) within an hyper-edge will access memory of a kind of cache within the chip (shared mem, registers, texture cache, L1, etc)
• **Task parallelism** is an **orthogonal** optimization that increases parallelism, and achieved bandwidth on massively parallel architectures.

• Crucial for strong scalability
Task parallelism applied to hyper-DAG of COSMO

- Applying task parallelism to current hyper-DAG of COSMO based on STELLA/GridTools DSL

<table>
<thead>
<tr>
<th>Size</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>39x28</td>
<td>40 %</td>
</tr>
<tr>
<td>54x54</td>
<td>31 %</td>
</tr>
<tr>
<td>128x128</td>
<td>6 %</td>
</tr>
</tbody>
</table>
Fusion or Task parallelism?

We need data-centric programming models, that can reason about data flow of our programs.

Avoid expressing data parallelism and task parallelism, hardcoded in Fortran models.

Efficient choices on today’s architectures are obsolete for different or future computing architectures.
High level language(s) for weather and climate

- Large legacy model need to be adapted
- Separations of concerns
- Increase abstraction: no explicit data structure, loops, HW-dependent details
- More optimizations, task parallelism
- Higher productivity and code safety

gtclang language prototype:
- Safety in numerics and parallel codes
- 10x reduction in LOC

Example gtclang

```cpp
function avg {
    offset off
    storage in
    avg = 0.5 * ( in(off) + in() )
}

function coriolis_force {
    storage fc, in
    coriolis_force = fc() * in()
}

operator coriolis {
    storage u_tend, u, v_tend, v, fc
    vertical_region ( k_start , k_end ) {
        u_tend += avg(j-1, coriolis_force(fc, avg(...
        v_tend -= avg(i-1, coriolis_force(fc, avg(...
    }
}
```

See T. Wicki poster and talk at «Bridging the software productivity gap for climate and weather»
Conclusions

• Achieving **portability** of COSMO model by means of a GridTools backend for **KNL / AVX-512** and OpenMP optimizations in Fortran physical parametrizations.

• Exploring how to enable **task parallelism** within existing technology

• Developing **data-centric programming models**, for concise, portable and future proof weather models
MeteoSwiss
Operation Center 1
CH-8058 Zurich-Airport
T +41 58 460 91 11
www.meteoswiss.ch

MeteoSvizzera
Via ai Monti 146
CH-6605 Locarno-Monti
T +41 58 460 92 22
www.meteosvizzera.ch

MétéoSuisse
7bis, av. de la Paix
CH-1211 Genève 2
T +41 58 460 98 88
www.meteosuisse.ch

MétéoSuisse
Chemin de l’Aérologie
CH-1530 Payerne
T +41 58 460 94 44
www.meteosuisse.ch