Learning to forget
Lessons from adopting and maintaining a weather and climate model for heterogeneous HPC systems

Oliver Fuhrer, MeteoSwiss
Contributions from X. Lapillonne\textsuperscript{1}, C. Osuna\textsuperscript{1}, M. Bianco\textsuperscript{2,3}, L. Benedicic\textsuperscript{2}, T. Schulthess\textsuperscript{2,3}

\textsuperscript{1}MeteoSwiss, \textsuperscript{2}CSCS, \textsuperscript{3}ITP ETH Zurich,

ADAC Workshop, 20.6.2018
Can you spot the weather model?
Reality
Where did we start in 2010?
### Operations in 2010

<table>
<thead>
<tr>
<th>Model</th>
<th>Grid Spacing</th>
<th>Frequency</th>
<th>Forecast Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>EZMWF-Modell</td>
<td>16 km</td>
<td>2 x per day</td>
<td>10 day</td>
</tr>
<tr>
<td>COSMO-7</td>
<td>6.6 km</td>
<td>3 x per day</td>
<td>3 day</td>
</tr>
<tr>
<td>COSMO-2</td>
<td>2.2 km</td>
<td>8 x per day</td>
<td>33 h</td>
</tr>
</tbody>
</table>
Strategy for next-generation
(computational effort relative to operational system)

ECMWF-Model
9 to 18 km gridspacing
2 to 4 x per day

COSMO-1
1.1 km gridspacing
8 x per day
1 to 2 d forecast

COSMO-E
2.2 km gridspacing
2 x per day
5 d forecast
21 members

Ensemble data assimilation: LETKF

= 40 x
Business as usual

**Cray XE6 (Albis/Lema)**
Current operational system at CSCS

**Next System**
Accounting for Moore’s law

*Not feasible! (power, floor space, cost)*
COSMO Model

350 kLOC of F90 + MPI + NEC directives
("optimized code")
Up or down?

- **Increase level of abstraction**
  - Hide implementation details
  - Can be disruptive

- **Decrease level of abstraction**
  - Add implementation details
  - Often incremental
Refactoring effort

- Domain-specific language (performance portable, re-usable)
- Separation of concerns
- High-level implementation

C++ / DSL rewrite

Fortran + directives

Fuhrer et al. 2014 (doi: 10.14529/jsfi140103)
Gysi et al. 2015 (doi: 10.1145/2807591.2807627)
Co-design (simultaneous software, hardware and workflow re-design) allowed MeteoSwiss to increase computational load by 40x within 4–5 years.

<table>
<thead>
<tr>
<th></th>
<th>Piz Dora</th>
<th>Piz Kesch</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time-to-solution</td>
<td>~26 CPUs</td>
<td>~7 GPUs</td>
<td>3.7 x</td>
</tr>
<tr>
<td>Energy-to-solution</td>
<td>10 kWh</td>
<td>2.1 kWh</td>
<td>4.8 x</td>
</tr>
<tr>
<td>Size of system (cabinets)</td>
<td>1.4</td>
<td>0.38</td>
<td>3.8 x</td>
</tr>
</tbody>
</table>

4x through investement into software.
Applications

- Used in production @MeteoSwiss
- Near-global simulations on full Piz Daint (4’888 GPU nodes, Fuhrer et al. 2018 GMD)

Visualization by Tarun Chadha (C2SM): clouds > $10^{-3}$ g/kg (white) and precipitation > $4 \times 10^{-2}$ g/kg (blue)
Fortran + MPI + OpenACC + ... is not the solution!
Software productivity gap!

durian:code fuhrer$ cloc E3SM/

<table>
<thead>
<tr>
<th>Language</th>
<th>files</th>
<th>blank</th>
<th>comment</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran 90</td>
<td>2812</td>
<td>244713</td>
<td>354664</td>
<td>1125623</td>
</tr>
<tr>
<td>C</td>
<td>1149</td>
<td>154791</td>
<td>250480</td>
<td>648455</td>
</tr>
<tr>
<td>Fortran 77</td>
<td>498</td>
<td>68946</td>
<td>98188</td>
<td>287216</td>
</tr>
<tr>
<td>HTML</td>
<td>441</td>
<td>4457</td>
<td>2927</td>
<td>162587</td>
</tr>
<tr>
<td>XML</td>
<td>914</td>
<td>15353</td>
<td>6546</td>
<td>142869</td>
</tr>
<tr>
<td>Bourne Shell</td>
<td>404</td>
<td>22088</td>
<td>25988</td>
<td>130941</td>
</tr>
<tr>
<td>C/C++ Header</td>
<td>399</td>
<td>11006</td>
<td>21174</td>
<td>60059</td>
</tr>
<tr>
<td>m4</td>
<td>50</td>
<td>5161</td>
<td>1170</td>
<td>49869</td>
</tr>
<tr>
<td>Python</td>
<td>329</td>
<td>12158</td>
<td>15054</td>
<td>45945</td>
</tr>
<tr>
<td>Perl</td>
<td>172</td>
<td>14406</td>
<td>21944</td>
<td>41502</td>
</tr>
<tr>
<td>TeX</td>
<td>172</td>
<td>4979</td>
<td>3830</td>
<td>29693</td>
</tr>
<tr>
<td>CMake</td>
<td>422</td>
<td>4679</td>
<td>6771</td>
<td>26563</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUM:</td>
<td>8408</td>
<td>586306</td>
<td>832057</td>
<td>2835375</td>
</tr>
</tbody>
</table>
Software productivity?

```c
#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, idx, blk, 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, l_startidx, l_endidx, je, jk) ICON_OMP_DEFAULT_SCHEDULE
#enddo
    do jb = l_startblk, l_endblk
        call get_indices_e(ptr_patch, jb, l_startblk, l_endblk, &
            l_startidx, l_endidx, rl_start, rl_end)

!$ACC LOOP VECTOR COLLAPSE(2)
#ifdef _LOOP_EXCHANGE
    do je = l_startidx, l_endidx
        do jk = slev, elev
#else
        do jk = slev, elev
            do je = l_startidx, l_endidx
        #endif
            ! compute the normal derivative
            ! by the finite difference approximation
            !(see Bonaventura and Ringler MWR 2005)
            grad_norm_psi_e(je, jk, jb) = &
                grad_norm_psi_e(je, jk, jb) &
                ( psi_c(idx(je, jb, 2), jk, blk(je, jb, 2)) - &
                grad_norm_psi_e(je, jk, jb, 1), jk, blk(je, jb, 1)) ) &
                * ptr_patch%edges%inv_dual_edge_length(je, jb)
#enddo
end do
#endif

!$ACC END PARALLEL
!$ACC DEBUG UPDATE HOST( grad_norm_psi_e ), IF( i_am_accel_node .AND. acc_on )
!$ ACC END DATA #else
!$OMP END DO NOWAIT
!$OMP END PARALLEL
#endif
```
Heavily optimized code is typically faster than Fortran + OpenACC, but also unreadable and unmaintainable!
Performance portability myth

- Radiation scheme on CPU (Intel E5-2690v3 “Haswell”) and GPU (NVIDIA Tesla K80) using Fortran + OpenMP + OpenACC

Lappillonne and Fuhrer, PPL, 2018
Clement et al. 2018, PASC’18
DSL in C++ may also not be the solution!
```cpp
enum { data, lap };

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

    static void Do(Context ctx, FullDomain) {
        ctx[lap::Center()] = -4.0 * ctx[data::Center()]
                          + ctx[data::At(iplus1)]
                          + ctx[data::At(iminus1)]
                          + ctx[data::At(jplus1)]
                          + ctx[data::At(jminus1)];
    }
};

IJKRealField lapfield, datafield;

Stencil stencil;
StencilCompiler::Build(
    stencil,
    "Example",
    calculationDomainSize,
    StencilConfiguration<Real, BlockSize<32,4>>(),
    pack_parameters(
        Param<lap, cInOut>(lapfield),
        Param<data, cIn>. (datafield)
    ),
    define_loops(
        define_sweep<cKIncrement>(
            define_stages(
                StencilStage<
                    Laplacian, IJRange<cComplete,0,0,0,0>,
                    KRangeFullDomain>
            )
        )
    )
);
STELLA DSL in C++ critique

+

- C++ well supported language
- Abstract (some) hardware dependent details
- High efficiency
- DSL allows for (partial) separation of concerns between domain-scientist and computer scientist

-

- C++ not well accepted by weather and climate community (boiler plate)
- No introspection / global optimization
- Backends complicated to implement (template meta-programming)
No turn key solution!

Is it time for a SDK for weather and climate?
Developmet Workflow

Mathematical Model

Discretization, Solver

Implementation

\[ T = T(x, z, t) \]
\[ T(x, z, t = 0) = T_0(x, z) \]
\[ \frac{\partial T}{\partial t} = \alpha \nabla^2 T \]

\[ \nabla^2 T_{i,k} = \frac{T_{i+1,k} + T_{i-1,k} + T_{i,k+1} + T_{i,k-1} - 4T_{i,k}}{\Delta^2} \]

```
do k = 2, nk-1
    do i = 2, ni-1
        lap(i,k) = ( T(i+1,k) + T(i-1,k) + T(i,k+1) + T(i,k-1) - 4*T(i,k) ) / dx / dy
    end do
end do
```
We need to learn to forget

High-level language for weather and climate
- No explicit data structure
- No loops / execution schedule
- No explicit threading / vectorization
- No directives
- No HW-dependent details
- ...

Separation of concerns

Mathematical Model
Discretization, Solver
High-level implementation
Domain-specific Compiler

climate scientist
domain scientist
computer scientist

Mathematical Model
Discretization, Solver
High-level implementation
Domain-specific Compiler

MeteoSwiss
• “SDK for weather and climate science”
• Joint development between CSCS / MeteoSwiss
• Domain-specific for Earth system model components
• Regional (production) and global grids (prototype)
• Multiple APIs (C++, Python, gtclang)
Example (GT4Py)

```python
# Definitions function
def horizontal_diffusion(data, weight):
    i = gt.Index()
    j = gt.Index()

    laplacian = gt.Equation()
    flux_i = gt.Equation()
    flux_j = gt.Equation()
    diffusion = gt.Equation()

    # Laplacian operator
    laplacian[i, j] = -4.0 * data[i, j] + (data[i-1, j] + data[i+1, j] + data[i, j-1] + data[i, j+1])

    # Horizontal flux
    flux_i[i, j] = laplacian[i+1, j] - laplacian[i, j]
# Vertical flux
    flux_j[i, j] = laplacian[i, j+1] - laplacian[i, j]
# Diffusion
    diffusion[i, j] = weight[i, j] * (flux_i[i-1, j] - flux_i[i, j] + flux_j[i, j-1] - flux_j[i, j])

    return diffusion

# Create computation domain
my_domain = gt.domain.Rectangle(((2, 2), (61, 61)))

# Create stencil object
stencil = gt.Stencil(definitions_func=horizontal_diffusion,
                      inputs={"data": array_a, "weight": array_b},
                      outputs={"diffusion": array_out},
                      domain=my_domain, mode=gt.mode.ALPHA)
```

- Mathematical operators (stencils)
- Data fields
- Region over which operator is applied
- Boundary conditions (not shown in example)
- High-level, declarative syntax
- Numpy as well as high-performance backends (x86 multi-core, NVIDIA GPU, Xeon Phi)
Example (gtclang)

```plaintext
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(...
    }
}
```

- Higher productivity and code safety
- Based on LLVM/Clang compiler framework
- Allows for global optimization “across kernels”
- Generates efficient code for x86 multi-core, NVIDIA GPUs, Intel Xeon Phi, and ARM (prototype)
- 4x – 6x reduction in LOC
Compiler Toolchain

DSL Frontends
- Python DSL Frontend
- clang DSL (C++)
- CLAW DSL (Fortran)

High-level IR

Domain Specific Checkers
- Read before write
- Missing Update Boundary
- Data dependency race conditions
- Out of Bounds Stencil Access

Optimizers
- Software Managed Caches
- Full vertical parallelization
- Stage Fusion
- Data Locality Exploit
- Strong / Weak Scaling Optimizer

Code Generator
- Naive C/Fortran Generator
- Optimized GridTools Generator
Summary

• Weather and climate community is struggling to keep up
  – very few models able to target multiple hardware architectures
  – software productivity gap
  – no turn-key solutions

• Large software effort for COSMO to enable efficient execution on multiple hardware architectures

• Learnings
  – It’s not about porting, it’s about maintaining
  – Fortran + directives exacerbate problem (ad-interim solution?)
  – DSL embedded C++ not readily adopted by domain-scientists
  – We need a high-level software development environment for weather and climate
Thank you! Questions?

“I want you to find a bold and innovative way to do everything exactly the same way it’s been done for 25 years.”
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