COSMO Dynamical Core Redesign

Tobias Gysi
David Müller
Boulder, 8.9.2011
Project Environment
### Who is SCS?

<table>
<thead>
<tr>
<th>Company Profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Privately owned R&amp;D services company founded in 1993.</td>
</tr>
<tr>
<td>• More than 70 engineers and scientists in Technopark Zurich.</td>
</tr>
<tr>
<td>• 100% dedicated to our customer projects.</td>
</tr>
<tr>
<td>• All IP-Rights (including source-code, documentation, etc.) belong to our customers.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Innovation processes</td>
</tr>
<tr>
<td>• System design, HW, FW &amp; SW design</td>
</tr>
<tr>
<td>• Implementation &amp; test</td>
</tr>
<tr>
<td>• Lifetime support</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Core Competencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Fast communication &amp; data processing</td>
</tr>
<tr>
<td>• Enterprise applications</td>
</tr>
<tr>
<td>• Scaling &amp; highly reliable system architectures</td>
</tr>
<tr>
<td>• Algorithm optimizations</td>
</tr>
</tbody>
</table>
The COSMO model

- Limited-area model - http://www.cosmo-model.org/
- Operational at 7 national weather services
- O(50) universities and research institutes
HP2C

- 10 Projects from different domains - [http://www.hp2c.ch/](http://www.hp2c.ch/)
  - Cardiovascular simulation (EPFL)
  - Stellar explosions (University of Basel)
  - Quantum dynamics (University of Geneva)
  - ...
  - COSMO-CCLM

- COSMO-CCLM tasks
  1. Cloud resolving climate simulations (IPCC AR5)
  2. Adapt existing code (hybrid, I/O)
  3. Rewrite of dynamical core
Motivation
Feasibility Study

- The code is memory bandwidth limited
- Prototype implementation of the Fast Wave Solver showed
  - 2x performance increase through code optimization
  - Performance scales with memory bandwidth (strong interest in GPUs)
- But the optimizations are
  - Architecture dependent
  - Increase code complexity
- Conclusion
  - Try to rewrite the code while hiding the additional complexity
# Goals of the HP2C Dycore rewrite

<table>
<thead>
<tr>
<th>Goal</th>
<th>Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Correctness</td>
<td>Unit testing</td>
</tr>
<tr>
<td></td>
<td>Verification framework</td>
</tr>
<tr>
<td>2 Performance</td>
<td>Loop merging</td>
</tr>
<tr>
<td></td>
<td>Calculation on-the-fly</td>
</tr>
<tr>
<td>3 (Performance)</td>
<td>Split user application from platform specific library</td>
</tr>
<tr>
<td>Portability</td>
<td>Allow different storage orders and blocking schemes</td>
</tr>
<tr>
<td>x86-GPU-...</td>
<td>Support 3-level parallelism</td>
</tr>
<tr>
<td></td>
<td>Nodes</td>
</tr>
<tr>
<td></td>
<td>Cores</td>
</tr>
<tr>
<td></td>
<td>SIMD</td>
</tr>
<tr>
<td>4 Ease of Use</td>
<td>Single source code</td>
</tr>
<tr>
<td></td>
<td>Readability</td>
</tr>
<tr>
<td></td>
<td>Usability</td>
</tr>
<tr>
<td></td>
<td>Maintainability</td>
</tr>
</tbody>
</table>
Approach by Example
An Example – Fast Wave U-Update

- X-component (transformed into spherical, terrain-following coordinates)

\[
\frac{\partial u}{\partial t} = - \frac{1}{\rho \alpha \cos \phi} \left( \frac{\partial p'}{\partial \lambda} + \frac{\partial z}{\partial \lambda} \frac{\partial p'}{\partial \zeta} \right)
\]

- X-component (discretized form)

\[
\frac{\partial u}{\partial t} = - \frac{1}{\bar{\rho}^\lambda \alpha \cos \phi} \frac{180}{\pi \lambda} \left( \delta_{\lambda} p' + \frac{\delta_{\lambda} z^\zeta}{\delta_{\zeta} z^\lambda} \delta_{\zeta} p' \zeta^\lambda \right)
\]

- Fundamental operators

\[
\delta_{\lambda} x = x_{i+1,k} - x_{i,k} \quad \bar{x}^\lambda = \frac{1}{2} (x_{i+1,k} + x_{i,k})
\]

\[
\delta_{\zeta} x = x_{i,k+1} - x_{i,k} \quad \bar{x}^\zeta = \frac{1}{2} (x_{i,k+1} + x_{i,k})
\]
Fortran Version

```
DO k = MAX(2,kflat), ke
    zppgradcor(istart-1:iend+1,jstart-1:jend+1,k) = &
        pp(istart-1:iend+1,jstart-1:jend+1,k) * wgtfac(istart-1:iend+1,jstart-1:jend+1,k) + &
        pp(istart-1:iend+1,jstart-1:jend+1,k-1) * (1.0_ireals-wgtfac(istart-1:iend+1,jstart-1:jend+1,k))
ENDDO

DO  k = 1, ke
    IF ( k < kflat ) THEN
        DO  j = jstartu, jendu
            DO  i = ilowu, iendu
                zpgradx = (zpi(i+1,j,k) - zpi(i,j,k)) * zrhoqx_i(i,j,k) * dts
                u(i,j,k,nnew) = u(i,j,k,nnew) - zpgradx + suten(i,j,k)
            ENDDO
        ENDDO
    ELSE
        zdpdz(istart-1:iend+1,jstart-1:jend+1) = &
            zppgradcor(istart-1:iend+1,jstart-1:jend+1,k+1) - zppgradcor(istart-1:iend+1,jstart-1:jend+1,k)
        DO  j = jstartu, jendu
            DO  i = ilowu, iendu
                zdpdz(i+1,j) + zdpdz(i,j)zdzpz =  zdpdz(i+1,j) + zdpdz(i,j)
zdpdx =  pp(i+1,j,k) - pp(i,j,k)
zpgradx =  zdpdz(i+1,j) + zdpdz(i,j)
                u(i,j,k,nnew) = u(i,j,k,nnew) - zpgradx*zrhoqx_i(i,j,k) * dts + suten(i,j,k)
            ENDDO
        ENDDO
    ENDIF
ENDDO
```

2 full 3D loops
DO k = MAX(2,kflat), ke

zppgradcor(istart-1:iend+1,jstart-1:jend+1,k) = &
  pp(istart-1:iend+1,jstart-1:jend+1,k) * wgtfac(istart-1:iend+1,jstart-1:jend+1,k) + &
  pp(istart-1:iend+1,jstart-1:jend+1,k-1) * (1.0_ireals-wgtfac(istart-1:iend+1,jstart-1:jend+1,k))
ENDDO

DO  k = 1, ke
  IF ( k < kflat ) THEN
    DO  j = jstartu, jendu
      DO  i = ilowu, iendu
        zpgradx = (zpi(i+1,j,k) - zpi(i,j,k)) * zrhoqx_i(i,j,k) * dts
        u(i,j,k,nnew) = u(i,j,k,nnew) - zpgradx + suten(i,j,k)
      ENDDO
    ENDDO
  ELSE
    zdpdz(istart-1:iend+1,jstart-1:jend+1) = &
    zppgradcor(istart-1:iend+1,jstart-1:jend+1,k+1) - zppgradcor(istart-1:iend+1,jstart-1:jend+1,k)
    DO  j = jstartu, jendu
      DO  i = ilowu, iendu
        zdpdz(i+1,j) + zdpdz(i,j)
        zdpdx = pp(i+1,j,k) - pp(i,j,k)
        zpgradx = zdpdx + zdzpz
        u(i,j,k,nnew) = u(i,j,k,nnew) - zpgradx*zrhoqx_i(i,j,k) * dts + suten(i,j,k)
      ENDDO
    ENDDO
  ENDIF
ENDDO
Idea

- Write a library to abstract the underlying hardware platform (CPU / GPU)
  - Adapt loop order / storage layout to the platform
  - Optimize data field access / index calculation
  - Leverage software caching
- Problem
  - There is no straight forward stencil library API
- Approach split loop logic from update functions
  - Functors wrap the update functions
  - Domain Specific Embedded Language (DSEL) specifies the loops / control flow

C++ Metaprogramming allows to define a “loop language” / DSEL which generates the loops at compile time
Stencil Definition and Usage

Stencil fastWaveUUpdate;
typedef BlockSize<8,8> FastWaveUVBlockSize;

// stencil definition
fastWaveUUpdate.Init(
    "FastWaveU",
    dycoreRepository.calculationDomain(),
    StencilConfiguration<Real, FastWaveUVBlockSize>(),
    concatenate_stages(
        StencilStage<PPGradCorStage, KLoopTerrainCoordinates, IJBoundary<cIndented, 0,1, 0,1> >(),
        StencilStage<UStage, KLoopAtmosphere, IJBoundary<cComplete, 0,0, 0,0> >()
    ),
    create_context(
        Ref(dycoreRepository.u_out()),
        CRef(fastWaveRepository.u_pos()),
        CRef(dycoreRepository.uadvt()),
        CRef(dycoreRepository.rho_in()),
        CRef(fastWaveRepository.pp () ),
        CRef(fastWaveRepository.fx()),
        CRef(fastWaveRepository.wgtfac()),
        CRef(fastWaveRepository.dzdx()),
        Scalar("dts", fastWaveRepository.dts()),
        Buffer<KRangeTerrainCoordinates, IJKRealField>("ppgradcor")
    )
);

// stencil usage
for(int step=0; step<numberOfTimeSteps; ++step) fastWaveUUpdate.Apply();

3D loops are represented by stencil stages

Update Functor  K-Loop  IJ - Boundary

Define parameter tuple / context

Stencil stages can communicate efficiently using buffers
„Call Stack“

- Stencil Instance
- FastWaveUUpdate
- Stage 1
- Stage 2
- K-Loop
- TerrainCoordinates
- KLoop
- Atmosphere
- Update Functor
- PPGradCorStage
- UStage
- Stencil Function
- Gradient
Stencil Stage Definition - Pressure Gradient Correction

```cpp
template<typename TEnv>
struct PPGradCorStage
{
    STENCIL_STAGE(TEnv);

    STAGE_PARAMETER(TerrainCoordinates, 4, pp);
    STAGE_PARAMETER(TerrainCoordinates, 6, wgtfac);
    STAGE_PARAMETER(TerrainCoordinates, 9, ppgradcor);

    USING(Delta2K);

    __ACC__ static void Do(Context ctx, TerrainCoordinates)
    {
        ctx[ppgradcor::Center()] = ctx[Delta2K::With(wgtfac::Center(), pp::Center())];
    }

};
```

Parameter declaration

Function computing a weighted delta

Access fields via ctx object (setup using create tuple)
Stencil Stage Definition – U Update

\[
\text{template<typename TEnv>}
\]
\[
\text{struct UStage}
\]
\[
\begin{align*}
&\text{STENCIL\_STAGE(TEnv);} \\
&\text{STAGE\_PARAMETER(FullDomain, 0, u\_out);} \\
&\text{STAGE\_PARAMETER(FullDomain, 1, u\_pos);} \\
&\text{STAGE\_PARAMETER(FullDomain, 2, uadvt);} \\
&\text{STAGE\_PARAMETER(FullDomain, 3, rho\_in);} \\
&\text{STAGE\_PARAMETER(FullDomain, 4, pp);} \\
&\text{STAGE\_PARAMETER(FullDomain, 5, fx);} \\
&\text{STAGE\_PARAMETER(FullDomain, 8, dts);} \\
&\text{STAGE\_PARAMETER(TerrainCoordinates, 7, dzdx);} \\
&\text{STAGE\_PARAMETER(TerrainCoordinates, 9, ppgradcor);} \\
&\text{USING(Average); USING(Delta); USING(Gradient);}
\end{align*}
\]
\[
\text{\_\_ACC\_ static void Do(Context ctx, FullDomain)}
\]
\[
\begin{align*}
&\text{T rho = ctx[fx::Center()].} \\
&\text{\_\_ACC\_ static void Do(Context ctx, FullDomain)}
\end{align*}
\]

Boulder, 8.9.2011, © by Supercomputing Systems
Stencil Function Definition – Gradient

```cpp
template<typename TEnv>
struct Gradient {
    STENCIL_FUNCTION(TEnv);

    FUNCTION_PARAMETER(0, dir);
    FUNCTION_PARAMETER(1, data);
    FUNCTION_PARAMETER(2, deltak);
    FUNCTION_PARAMETER(3, deltaz);

    USING(Delta); USING(Sum);

    __ACC__ static T Do(Context ctx, TerrainCoordinates) {
        return
            ctx[Delta::With(dir(), data::Center())] +
            ctx[Sum::With(dir(), deltak::Center())] * ctx[deltaz::Center()];
    }

    __ACC__ static T Do(Context ctx, FlatCoordinates) {
        // delta z is zero for the flat coordinates
        return
            ctx[Delta::With(dir(), data::Center())];
    }
};
```

Stencil functions optionally return a value

Different update functions for different domains
Implementation
CPU Library Backend - Parallelization

• Storage
  – The data is stored continuously in K
  – Works well for the numerous tri-diagonal solves in the vertical
  – Allows efficient blocking in the IJ-plane

• Loops
  – Loop over all blocks (parallelized with OpenMP)
  – Loop over all stencil stages
  – Loop over IJ
  – Loop over K
CPU Library Backend - Buffers

- Buffers store intermediate values handed over from one stage to the next
  - “ppgradcor” in the Fast Wave U-update
- COSMO uses full 3D fields
  - Intermediate value stored in DRAM
- Due to blocking it is possible to allocate a field with the size of a block for every core
  - Intermediate values are stored in L2 cache
Status

• The HP2C dycore is functional and verified against COSMO
  – Fast Wave solver
  – Advection
    • 5th order advection
    • Bott 2 advection (cri implementation in z direction)
  – Implicit vertical diffusion and advection (slow tendencies)
  – Horizontal diffusion
  – Some smaller stencils like Coriolis, CalcRho etc.

• Performance portable CPU backend
  – Factor 1.5x – 1.6x faster than the standard COSMO implementation
  – SSE not used (expect another ~30%)

• Functional GPU / CUDA backend
  – No performance measurements so far
  – Room for performance tuning
Conclusions
Conclusions

• It works
  – Hardware platform specific code is hidden inside library
  – Performance portability feasible
  – Inside the library we could write “ugly” but fast code
    • Blocking
    • Iterators
  – The library has been successfully and efficiently used by experienced as well as ‘fresh’ domain scientists and SW engineers

• Work to be done
  – Achieve the expected GPU performance until end of 2011
  – Adoption of the code by the weather services / climate community
Discussion

Acknowledgements to all our collaborators at
• C2SM (Center for Climate Systems Modeling)
• CSCS
• DWD (Deutscher Wetterdienst)
• MeteoSwiss
• Nvidia