Operational numerical weather prediction on a GPU-accelerated cluster supercomputer

OLD Meteoswiss operational system

Cray XE6 (Albis/Lema)
MeteoSwiss operational system
~4 years
OLD Meteoswiss operational system

**ECMWF-Model**
16 km grid spacing
2 x per day 10 day forecast

**COSMO-7**
\( \Delta x = 6.6 \text{ km}, \Delta t = 60 \text{ s} \)
393 x 338 x 60 cells
3 x per day 72 h forecast

**COSMO-2**
\( \Delta x = 2.2 \text{ km}, \Delta t = 20 \text{ s} \)
520 x 350 x 60 cells
7 x per day 33 h forecast
1 x per day 45 h forecast
Requirements for New operational setup

- COSMO-1 (1 km high resolution) and COSMO-E (ensemble)
- Total computation cost for requirements = 40x

COSMO-1
1.1 km gridspacing
8 x per day
1 to 2 d forecast
(in production since 1/04/2016)

COSMO-E
2.2 km gridspacing
2 x per day
5 d forecast
21 members
(production planned Mai/June 2016)

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

Ensemble data assimilation: LETKF
13x
7x
20x
How to reach 40x in 5 years?

Key Ingredients

• Processor performance (Moore’s Law) ~2.8x
• Increase in number of sockets ~1.3x
• Port to accelerators (GPUs) ~2.3x
• Code improvement ~1.7x
• Increase utilization of system ~2.8x

\[~4x\]
New MeteoSwiss hybrid HPC system

Piz Kesch (Cray CS Storm)

• 2 Cabinets (production & failover) installed at CSCS in July 2015

• 12 “Fat” compute nodes per cabinet
  ▪ 2 Intel CPU Xeon E5 2690 (Haswell)
  ▪ 8 Tesla K80 GPUs (each with 2 GK210 chip)
New MeteoSwiss hybrid HPC system

![Diagram of the hybrid HPC system with K80 nodes and QPI connections.]

MeteoSwiss

17th Workshop on High Performance Computing in Meteorology, Oct 2016
Carlos Osuna
GPU communication with MPI

- Every Pack-ME every pair GCL::Pack-MPI_Isend is generating large gaps in the streams due to delays in the host timeline (cudaDeviceSynchronize and cuda API calls)
- Collaboration with OSU, we will need mvapich2 GDS

```
cudaEventSynchronize
cudacudaEventElapsedTime
```

7.5 % total time
How to port a full weather model to GPUs?

Up or down?

- **Increase level of abstraction**
  - Remove details of implementation
  - “Disruptive change”

- **Lower level of abstraction**
  - Add implementation details
  - „Incremental change“

Source: Oliver Fuhrer
The COSMO model on GPU

- Take advantage of the high computational capacity of GPUs
- Low compute intensity: avoid GPU-CPU data transfer
- Full GPU port strategy: all computations on the GPU

Initialization → CPU

- Prepare step → OpenACC port
- Physics → OpenACC port
- Dynamics → STELLA re-write (C++/Cuda) Interface
- Data assimilation → Mixed OpenACC/CPU Interface
- Halo-update → Communication library (GCL)
- Diagnostics → OpenACC port
- I/O → Mixed OpenACC/CPU

Δt

Cleanup

Transfer to CPU on I/O step
DOWN – Lower Level of abstraction

- Incremental adaptation to various programming models
- Traditionally OpenMP, OpenACC,
- Limited flexibility to custom platform dependent adaptations (#ifdef)

```
DO jb = i_startblk, i_endblk
  DO jc = i_startidx, i_endidx
    DO jk = slev, elev
      div_vec_c(jc,jk,jb) = vec_e(iid(ic,jb,1),jk,iblk(ic,jb,1)) * ptr_int%geofac_div(jc,1,jb) + &
      vec_e(iid(ic,jb,2),jk,iblk(ic,jb,2)) * ptr_int%geofac_div(jc,2,jb) + &
      vec_e(iid(ic,jb,3),jk,iblk(ic,jb,3)) * ptr_int%geofac_div(jc,3,jb)
    ENDDO
  ENDDO
ENDDO
ENDDO
```
DOWN – Lower Level of abstraction

- Incremental adaptation to various programming models
- Traditionally OpenMP, OpenACC,
- Limited flexibility to custom platform dependent adaptations (#ifdef)

```
!$ACC DATA COPYIN(vec_e) COPYOUT(div_vec_c)
!$ACC KERNELS LOOP, GANG(32), WORKER(8)
OMP PARALLEL DO STATIC
```
UP - STELLA / GridTools Library

- Domain-specific embedded language in C++
- Developed for stencil computations
- Separation of concern between model and hardware implementation
- Back-end optimized for different architectures (CPU, GPU)

```cpp
struct Laplace
{
    typedef in_accessor<0, range<-1,1,-1,1>> u;
    typedef out_accessor<1> lap;

    template<typename Evaluation>
    static void Do(Evaluation const& eval, full_domain)
    {
        eval(lap()) = eval(-4*u() + u(i+1) + u(i-1) + u(j+1) + u(j-1));
    }
};
```
Performance Portability - Why UP (I)?

<table>
<thead>
<tr>
<th></th>
<th>runtime</th>
<th>grid size</th>
<th>block size</th>
<th>occupancy</th>
<th>DRAM throughput</th>
<th>shared memory</th>
<th>register usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>non-blocked (naive)(^{1})</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K20X</td>
<td>0.53 ms</td>
<td>60 x 1 x 1</td>
<td>128 x 1 x 1</td>
<td>0.266</td>
<td>75.1 GB/s</td>
<td>68.0 GB/s</td>
<td>0 B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>60 x 1 x 1</td>
<td>128 x 1 x 1</td>
<td>0.265</td>
<td>84.8 GB/s</td>
<td>76.5 GB/s</td>
<td>0 B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>60 x 1 x 1</td>
<td>128 x 1 x 1</td>
<td>0.266</td>
<td>116.2 GB/s</td>
<td>35.5 GB/s</td>
<td>0 B</td>
</tr>
<tr>
<td>K20</td>
<td>0.68 ms</td>
<td>60 x 1 x 1</td>
<td>32 x 4 x 1</td>
<td>0.285</td>
<td>70.3 GB/s</td>
<td>26.3 GB/s</td>
<td>0 B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>60 x 1 x 1</td>
<td>32 x 4 x 1</td>
<td>0.284</td>
<td>39.1 GB/s</td>
<td>40.5 GB/s</td>
<td>0 B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>60 x 1 x 1</td>
<td>32 x 4 x 1</td>
<td>0.285</td>
<td>45.2 GB/s</td>
<td>45.5 GB/s</td>
<td>0 B</td>
</tr>
<tr>
<td>block</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K20</td>
<td>0.54 ms</td>
<td>128 x 1 x 1</td>
<td>32 x 4 x 1</td>
<td>0.600</td>
<td>15.9 GB/s</td>
<td>16.1 GB/s</td>
<td>4.272 KB</td>
</tr>
<tr>
<td>K20</td>
<td>0.56 ms</td>
<td>7680 x 1 x 1</td>
<td>32 x 4 x 1</td>
<td>0.670</td>
<td>15.4 GB/s</td>
<td>16.1 GB/s</td>
<td>4.272 KB</td>
</tr>
<tr>
<td>shared</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K20</td>
<td>0.54 ms</td>
<td>128 x 1 x 1</td>
<td>32 x 4 x 1</td>
<td>0.600</td>
<td>15.9 GB/s</td>
<td>16.1 GB/s</td>
<td>4.272 KB</td>
</tr>
<tr>
<td>shared-3D</td>
<td></td>
<td>7680 x 1 x 1</td>
<td>32 x 4 x 1</td>
<td>0.670</td>
<td>15.4 GB/s</td>
<td>16.1 GB/s</td>
<td>4.272 KB</td>
</tr>
<tr>
<td>STELLA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K20X</td>
<td>0.29 ms</td>
<td>128 x 6 x 1</td>
<td>32 x 10 x 1</td>
<td>0.90</td>
<td>42.0 GB/s</td>
<td>34.8 GB/s</td>
<td>6.68 KB</td>
</tr>
<tr>
<td>K20</td>
<td>0.35 ms</td>
<td>128 x 6 x 1</td>
<td>32 x 10 x 1</td>
<td>0.90</td>
<td>25.7 GB/s</td>
<td>23.3 GB/s</td>
<td>6.68 KB</td>
</tr>
</tbody>
</table>

For simple dynamical core operators DSL is 1.5-2x faster than best optimized OpenACC code.
COSMO Dynamical Core operators dependency graph

~35 different operators

Diverse computational patterns:
- 3D stencils, 2D stencils, tridiagonal solvers, …

Performance Portability - Why UP (II)?
Preparing for performance portability of future models
Energy efficient Scalable Algorithms for weather Prediction at Exascale

Peter Bauer
Preparing for future models

GridTools On Irregular Grids

- Portability of dynamical cores of global models
- Abstraction of the grid and the computing architecture
- Exploit maximum performance

«Increase data locality should be a top priority»

Thomas Schulthess, HPC in Meteorology, 2016
Data Locality on unstructured meshes

Loop fusion and redundant computations on Lat-Lon grids

!OMP PARALLEL DO
DO iblock=1,nblocks
  DO k=1,nz
    DO j=jstart-1,jend+1
      DO i=istart-1,iend+1
        lap(i,j,k,iblock) = -4*u(i,j,k,iblock) + u(i+1,j,k,iblock) + u(i-1,j,k,iblock) + u(i,j+1,k,iblock) + u(i,j-1,k,iblock)
      ENDDO
    ENDDO
  ENDDO
DO j=jstart,jend
  DO i=istart,iend
    udiff(i,j,k,iblock) = -4*lap(i,j,k,iblock) + lap(i+1,j,k,iblock) + lap(i-1,j,k,iblock) + lap(i,j+1,k,iblock) + lap(i,j-1,k,iblock)
  ENDDO
ENDDO
ENDDO
Data Locality on unstructured meshes

On irregular grids?

DO jb = i_startblk, i_endblk
   DO jc = i_startidx, i_endidx
      DO jk = slev, elev
         div(jc,jk,jb) = u(iid(ic,jb,1),jk,iblk(ic,jb,1)) * ptr_int%geofac_div(jc,1,jb) + &
         u(iid(ic,jb,2),jk,iblk(ic,jb,2)) * ptr_int%geofac_div(jc,2,jb) + &
         u(iid(ic,jb,3),jk,iblk(ic,jb,3)) * ptr_int%geofac_div(jc,3,jb)
      ENDDO
   ENDDO
ENDDO

Halo_exchange
DO jb = i_startblk, i_endblk
   DO jc = i_startidx, i_endidx
      DO jk = slev, elev
         div2(jc,jk,jb) = div(iid(ic,jb,1),jk,iblk(ic,jb,1)) * ptr_int%geofac_div(jc,1,jb) + &
         div(iid(ic,jb,2),jk,iblk(ic,jb,2)) * ptr_int%geofac_div(jc,2,jb) + &
         div(iid(ic,jb,3),jk,iblk(ic,jb,3)) * ptr_int%geofac_div(jc,3,jb)
      ENDDO
   ENDDO
ENDDO

Are we abandoning Loop Fusion on unstructured meshes?
Equal partitioner of octahedral grid

Structured partitioner of octahedral grid
\[ \text{div}(\mathbf{v})_i := \frac{1}{A_i} \sum_{l \in E(i)} v_{nl} (\mathbf{N}_l \cdot \mathbf{n}_{i,l}) l \]

Unstructured DSL syntax:

\[ \text{on_edges} \text{(sum_reduction, v(), l()) / A()); } \]
\[(\nabla^2_d \mathbf{v})_l \cdot \mathbf{N}_l := \text{grad}_n \left[ \text{div}(\mathbf{v}) \right]_l - \text{grad}_\tau \left[ \text{curl}(\mathbf{v}) \right]_l\]
DOWN - Preparing for future models

Performance portability: COSMO Radiation

-30%

-35%
CLAW: Low-level transformations

CLAW Compiler low-level transformations:
- Loop fusion
- Loop reordering
- Loop extraction
- Loop hoisting
- Caching
- On the fly computation
- Array notation to do statement
- Code removal
- Conditional directive enabling
SUBROUTINE lw_solver(...)  
!$claw define dimension icol(1:ncol) &  
!$claw parallelize  
DO igpt = 1, ngpt
  DO ilev = 1, nlay
    tau_loc(ilev) = max(tau(ilev,igpt) ...  
    END DO  
  DO ilev = 1, nlay
    trans(ilev) = exp(-tau_loc(ilev))  
    END DO  
  DO ilev = nlay, 1, -1
    radn_dn(ilev,igpt) = trans(ilev) * radn_dn(ilev+1,igpt) + ...  
    END DO  
  DO ilev = 2, nlay + 1
    radn_up(ilev,igpt) = trans(ilev-1) * radn_up(ilev-1,igpt) + ...  
    END DO  
END DO  
END SUBROUTINE lw_solver

Dependency on the vertical dimension only
CLAW One column abstraction

Original code
(Architecture agnostic)

- A single source code
- Specify a target architecture for the transformation
- Specify a compiler directives language to be added

```
clawfc --directive=openacc --target=gpu -o mo_lw_solver.acc.f90 mo_lw_solver.f90
clawfc --directive=openmp --target=cpu -o mo_lw_solver.omp.f90 mo_lw_solver.f90
clawfc --directive=openmp --target=mic -o mo_lw_solver.mic.f90 mo_lw_solver.f90
```
Conclusions

- New setup COSMO-1 and COSMO-E running on hybrid GPU system
- About $4x$ was gained by moving to GPUs as compared to traditional CPUs
Performance comparison with CPU only system

Piz Dora (Cray XC40)
- “Traditional” CPU based system
- Compute nodes with 2 Intel Xeon E5-2690 v3 socket (Haswell)
- Pure compute rack
- Rack has 192 compute nodes
- Very high density (supercomputing line)

- Performance comparison for the COSMO-E ensemble members
# Results

<table>
<thead>
<tr>
<th></th>
<th>Piz Dora</th>
<th>Piz Kesch</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy per member</td>
<td>10 kWh</td>
<td>2.1 kWh</td>
<td>4.8 x</td>
</tr>
<tr>
<td>Time with 8 sockets</td>
<td>3.9 h</td>
<td>1.0 h</td>
<td>3.9 x</td>
</tr>
<tr>
<td>per member</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cabinets required to run</td>
<td>1.4</td>
<td>0.38</td>
<td>3.8 x</td>
</tr>
<tr>
<td>ensemble at required</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time-to-solution</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Porting with OpenACC directives

```c
!$acc data create(a,b,c)
!$acc update device(b,c)
!$acc parallel
  !$acc loop gang vector
  do i=1,N
    a(i)=b(i)+c(i)
  end do
!$acc end parallel
!$acc update host(a)
!$acc end data
```

- Directive-based programming model for C++ and Fortran
- Provides abstraction to define parallel region, data locality (GPU, CPU) and mapping to specific hardware parallelism (gang, worker, vector)
- Enable to port large codes to GPU with acceptable efforts
- Used for porting the physics, assimilation, I/O.

Device memory management
Generates GPU kernel. Map parallelism to architecture
Performance portability with OpenACC

- In some cases CPU and GPU have different optimization requirement

**CPU**: compute bound
- Auto-vectorization: small loop
- Pre-computation

**GPU**: memory bound limited
- Benefit from large kernels: reduce kernel launch overhead, better computation/memory access overlap
- Loop re-ordering and scalar replacement
- On the fly computation

Optimize code for GPU runs 1.9x slower on CPU

Current solution: different code paths for time critical components with pre-processor macros
CLAW approach

- Goal: Provide language abstraction for performance portability in climate and weather model
- Directives with code transformation

```fortran
SUBROUTINE inv_th(pclc,pcal, ...)  
  INTEGER:: kilsd  
  !$acc parallel  
  !$acc loop collapse(3)  
  !$claw loop-interchange (k,i,j)  
  DO i=istart,iend  
    DO j=jstart,jend  
      DO k=kstart,kend  
        ! Computation is done here  
        END DO  
      END DO  
    END DO  
  END DO  
  !$acc end parallel  
END SUBROUTINE inv_th
```

CLAW
- Code manipulation with AST
- Based on the OMNI compiler
- Transformed code can be compile with standard compiler
CLAW language definition are available on github:

https://github.com/C2SM-RCM/claw-language-definition
References


Thank you
Increase of x40 in computational cost of operational setup

Key ingredients

- Processor performance (Moore’s law) ~2.8 x
- Code refactoring and port to GPUs ~3.9 x
- Increase utilization of system ~2.8 x
- Increase in number of sockets ~1.3 x
- Target system architecture to application