Adapting COSMO for GPU accelerators: Learnings and consequences for model developers

S. Ruedisuehli[1], X. Lapillonne[1], O. Fuhrer[2]

[1]Center for Climate Systems Modeling (C2SM), ETH Zürich, [2]Federal Office of Meteorology and climatology MeteoSwiss, Zürich

stefan.ruedisuehli@env.ethz.ch

Overview

- In the COSMO POMPA[1] project a COSMO version leveraging GPU accelerators is developed.
- GPUs can provide significantly larger performance (typically factor 3x to 5x) compared to traditional CPUs while consuming a similar amount of energy.
- The GPU-port follows a two-fold approach:
  - The dynamical core has been re-written from scratch (C++/DSL “STELLA”).
  - The rest of the Fortran code has been retained, but refactored and expanded with OpenACC directives.
- The focus of this poster is on Fortran (mainly the physics):
  - How do we port the Fortran code to GPU?
  - What changes will be introduced into COSMO?
  - What tools/approaches do we use for development?

[1]Performance on Massively Parallel Architectures

Code changes I: Major restructurings

Working arrays

- Fortran automatic arrays always result in a costly memory allocation since there is no stack on GPUs.
- Most efficiently, all GPU arrays are allocated/deallocated only once before/after the timeloop.
- Thus all local working arrays of subroutines are moved to the parent module, and allocate/deallocate subroutines are added.
- To utility subroutines, working arrays are passed as arguments.

Block physics

- It is not strictly GPU-related, but implemented simultaneously.
- The i and j dimensions are merged into one, made possible by the lack of lateral dependencies in the physics.
- Data is copied to/from block before/after the physics, and all data fields are passed as argument to the parameterizations.

Working array-restructuring: An idealized example

```
PROGRAM cosmo
  timeloop: DO t=1,nt
    CALL physics

MODULE m_physics

  SUBROUTINE physics
    CALL radiation
    CALL turbulence

  SUBROUTINE radiation
    REAL :: rad1(nx,ny)
    INTEGER :: rad_i
    !$acc data create(rad1)

  SUBROUTINE turbulence
    REAL :: tur1(nx,ny)
    LOGICAL :: turb_l
    !$acc data create(tur1)
```

```
PROGRAM cosmo_lowmem
  timeloop: DO t=1,nt
    CALL physics_wk_alloc
    CALL physics
...
```

```
PROGRAM cosmo
  CALL physics_wk_alloc
  timeloop: DO t=1,nt
    CALL physics

MODULE m_physics
  REAL,ALLOCATABLE :: &
    rad1(:,:), tur1(:,:)

  SUBROUTINE physics
    CALL radiation
    CALL turbulence

  SUBROUTINE radiation
    INTEGER :: i_rad
    !$acc data present(rad1)

  SUBROUTINE turbulence
    LOGICAL :: l_tur
    !$acc data present(tur1)

  SUBROUTINE physics_wk_alloc
    ALLOCATE(rad1(ie,je))
    ALLOCATE(tur1(ie,je))
    !$acc data create(rad1)
    !$acc data create(tur1)
```

OpenACC

- Open standard to run Fortran or C code on GPU accelerators by adding directives to the code, which are simply ignored as comments if the code is compiled for CPU (like OpenMP).

```
PROGRAM sample_openacc
USE mod, ONLY: ie,je,fact,init
 IMPLICIT NONE
 REAL :: arr1(ie,je), arr2(ie,je)
  !$acc data create (arr1,arr2,fact)      ! start data region
  !$acc update device (fact)              ! copy fact to GPU
  CALL init (arr1, lacc=.TRUE.)           ! init. arr1 on GPU
  CALL comp_gpu (arr1, fact, arr2)        ! compute arr2 on GPU
  !$acc update host (arr2)                ! copy arr2 to CPU
  !$acc end data                          ! end data region
  PRINT*, MINVAL(arr2),MAXVAL(arr2)       ! print arr2 on CPU
CONTAINS
SUBROUTINE comp_gpu(arr)
 REAL, INTENT(IN) :: arr1, fact
 REAL, INTENT(OUT) :: arr2
 INTEGER :: i,j
  !$acc parallel present (arr1,fact,arr2) ! start parallel region
  !$acc loop gang                         ! parallelization detail
  DO j=1,je
    !$acc loop vector                     ! parallelization detail
    DO i=1,ie
      arr2(i,j) = arr1(i,j) * fact(i,j)   ! compute arr2 on GPU
    ENDDO
  ENDDO
  !$acc end parallel                      ! end parallel region
END SUBROUTINE comp_gpu
END PROGRAM sample_openacc
```

Advantages

- Existing code remains mostly unchanged and shared with CPU.
- Easy to learn and to port (simple) codes with.

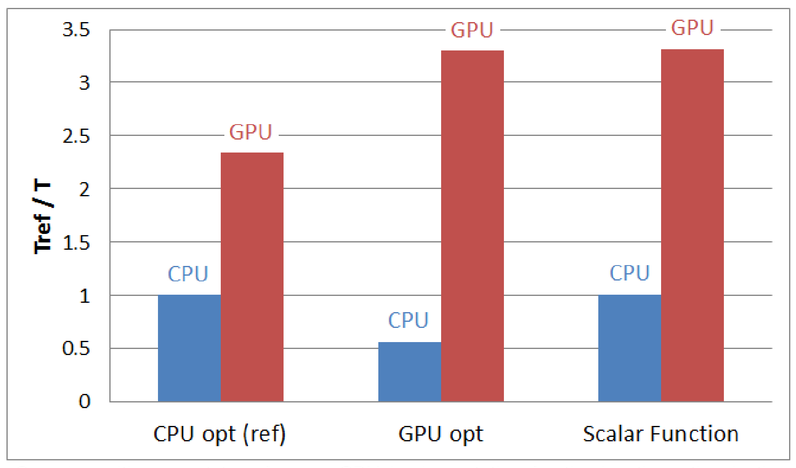
Disadvantages

- Array data in GPU memory has to be managed manually (copied back and forth) to avoid unnecessary data transfers.
- Hard to achieve performance portability.

Code changes II: Local optimizations

- Any GPU optimizations are constrained by the CPU performance of the code, which must not be degraded.
- The goal is to have as much shared source code as possible.
- Where absolutely necessary (most compute-intensive parts), special code is introduced for GPU. The most extreme case are the inversion routines inv_th/so in the radiation (*see graph*).
- On CPU, efficient kernels[2] are rather small and of low complexity so they can be vectorized by the compiler.
- On GPU, it is usually more efficient for kernels to do a lot of computational work.

[2]kernel: body of a loop (refers to loop constructs here)



Speedup with respect to reference CPU implementation for the inv_th routine in the radiation. Comparing GPU and CPU execution time for different optimization implementation.

Restructuring for GPU: A simiplyfied example

CPU

```
DO j=1,je
  DO i=1,ie
    tmp1(i,j) = f3_in1(i,j,1) * f2_in1(i,j)
  ENDDO
ENDDO

DO j=1,je
  DO i=1,ie
    tmp2(i,j) = tmp1(i,j) * f2_in2(i,j)**2
  ENDDO
ENDDO

!XL: too complex to vectorize
CALL compute_complex( f3_in2(:,:,:), tmp3(:,:,:) )

DO k=1,ke
  DO j=1,je
    DO i=1,ie
      f3_out(i,j,k) = 0.5*( tmp1(i,j,k) + tmp3(i,j,k) )
    ENDDO
  ENDDO
ENDDO
```

3D 2D 1D

- The increased work load makes the fused kernel more efficient on GPU, with scalarization providing an additional benefit.
- The fused kernel doesn’t vectorize on CPU because it is too complex, which outweighs the benefit from scalarization.

GPU

```
!$acc parallel
!$acc loop gang
DO j=1,je
  !$acc loop vector
  DO i=1,ie
    tmp1 = f3_in1(i,j,1) * f2_in1(i,j)
    tmp2 = tmp1 * f2_in2(i,j)**2
    DO k=1,ke
      !RUS: must be inlined
      CALL compute_complex_scalar( f3_in2(i,j,k), tmp3 )
      f3_out(i,j,k) = 0.5*( tmp1 + tmp3 )
    ENDDO
  ENDDO
ENDDO
!$acc end parallel
```

3D 2D 1D

Note that in this simplyfied case, the fused kernel might nevertheless be vectorized and run faster on CPU. In reality, the original code would be many times longer and the fused kernel correspondingly more complex.

Testing

COSMO Testsuite

- Short runs on reduced domain in many different configurations.
- Fast: <20 min for >20 tests
- Almost immediate technical validation of small changes.
- Thresholds can be set for certain time spans and variables to account for expected differences.
- Simple, modular framework makes it easy to add new tests or checkers.

```
[------------] TEST cosmo7/test_1: Only dynamics
  [  MATCH   ] run_success_check.py
  [  MATCH   ] existence_grib_out.sh
                 NOTE: setting thresholds to enforce bit-reproducibility
  [  MATCH   ] tolerance_check.py
  [  MATCH   ] output_tolerance_check.py
[  MATCH     ] RESULT cosmo7/test_1: Only dynamics
[------------] TEST cosmo7/test_2: Dynamics and physics
  [  MATCH   ] run_success_check.py
  [  MATCH   ] existence_grib_out.sh
                 NOTE: setting thresholds to enforce bit-reproducibility
  [  MATCH   ] tolerance_check.py
  [  MATCH   ] output_tolerance_check.py
[  MATCH     ] RESULT cosmo7/test_2: Dynamics and physics
[------------] TEST cosmo7/test_3: Dynamics, physics and assimilation
  [  MATCH   ] run_success_check.py
  [  MATCH   ] existence_grib_out.sh
                 NOTE: setting thresholds to enforce bit-reproducibility
                 ./tools/comp_yuprtest.py /workspace/scratch/ruestefa/CO
kspace/scratch/ruestefa/COSMO/cosmo/trunk/test/testsuite/data/cosmo7/tes
                 Absolute error:
                   nt    max_all         t     Test
                    0    0.00e+00    0.00e+00     OK
                    1    1.42e+00    6.14e-02     FAILED
                    2    4.77e+00    1.22e-01     FAILED
                    3    9.58e+00    1.82e-01     FAILED
                   10    7.32e+00    5.00e-01     FAILED
                   20    7.04e+00    7.79e-01     FAILED
                   30    7.23e+00    8.54e-01     FAILED
                   40    6.38e+00    7.46e-01     FAILED
                   50    5.86e+00    8.80e-01     FAILED
                   60    5.22e+00    8.10e-01     FAILED
                   70    5.01e+00    8.32e-01     FAILED
                   80    5.72e+00    1.36e+00     FAILED
                   90    8.17e+00    1.23e+00     FAILED
                  100    1.12e+01    8.38e-01     FAILED
                  110    1.56e+01    8.29e-01     FAILED
                  120    1.72e+01    8.61e-01     FAILED
  [   FAIL   ] tolerance_check.py
                 Error above threshold: 522 , max diff  4.831538e+13 at
                 Errors above threshold :
                 var       ee    lev       min     imin   jmin
                 FI_ANAI    24  17      -0.000000   30   32
                 FI_ANAI    24  17     -13.190679   73    7
  [   FAIL   ] output_tolerance_check.py
  [  MATCH   ] data_assimilation_check.rb
[   FAIL     ] RESULT cosmo7/test_3: Dynamics, physics and assimilation
```

Jenkins

- Fully-automized testing (on request and nightly) of compilation, testsuite, etc.
- Tests on all machines; with different compilers; on CPU and GPU; and in float and double.

Physics standalone

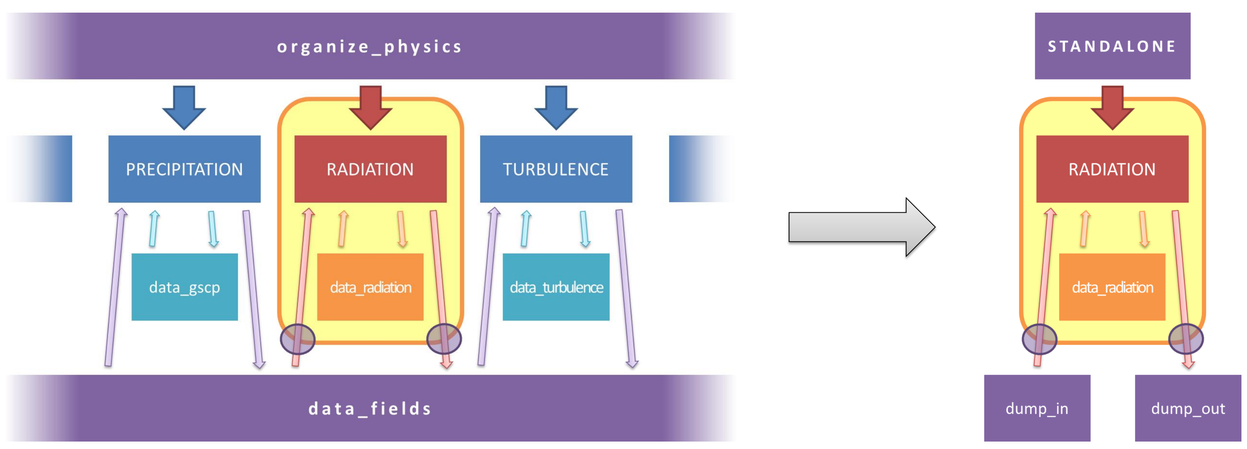
- Framework to work on physical parameterizations in isolation from the rest of the model.
- Necessary data fields are written to disk before and after the parameterization in a full model run.
- This data is used for initialization of the standalone and validation of it’s output.

Advantages

- Extremely fast testing/benchmarking.
- Helps increasing modularity (or illustrating the lack thereof).

Disadvantages

- Additional work to set it up (find all dependencies).
- So far only single timestep in one configuration.



Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

CSCS
Swiss National Supercomputing Centre

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich