Engineering the Transition of Climate Models for Diverging Architectures in Next-generation Supercomputers

Mohamed Wahib

15th February 2018
ADAC Workshop

In Collaboration with

ETH/CSCS

Thomas Schulthess, Joost VandeVondele, Mauro Bianco, Carlos Osuna (MeteoSwiss), Lucas Benedicic, Hannes Vogt, and Others

RIKEN AICS

Naoya Maruyama (LLNL), Hisashi Yashiro

TokyoTech

Satoshi Matsuoka, Takayuki Aoki, Michel Muller

This Collaboration started from ADAC

ADAC Workshop – January 2016

- Maruyama saw a presentation of GridTools
- Maruyama PI of SPPEXA AIMES project (performance of Climate Models)

June 2016

- Wahib visits CSCS @Lugano

September 2016

- Mauro, Lucas visit RIKEN AICS @Kobe

ADAC Workshop – January 2017

- F2F Meeting

March 2017

- Maruyama, Yashiro, Wahib visit CSCS @Zurich

ADAC Workshop – June 2017

- F2F Meeting

ADAC Workshop – February 2018

- Full day F2F Meeting

“This code is what matters most to me, supercomputers, architectures come and ago; my code will be used in production for more than a decade”

Climate scientist in a heated discussion

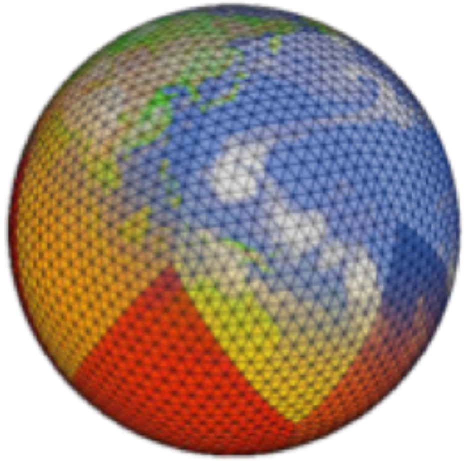
Summary

- Performance portability is a real challenge
  - Specially for legacy codes

- Climate models are complex codebases
  - Different components: different optimizing/adapting strategies

- NICAM, an atmospheric model, is an example of that
  - We discuss the ongoing effort for the transition of NICAM

NICAM



- **N**on-hydrostatic **Ic**osahedral **A**tmospheric **M**odel* (**NICAM**)
  - Development started in 2000
  - First global dx=3.5km run in 2004 using the Earth Simulator
  - First global dx=0.87km run in 2012 using the K computer
- A global model widely used in Japan

Main Institution Using NICAM

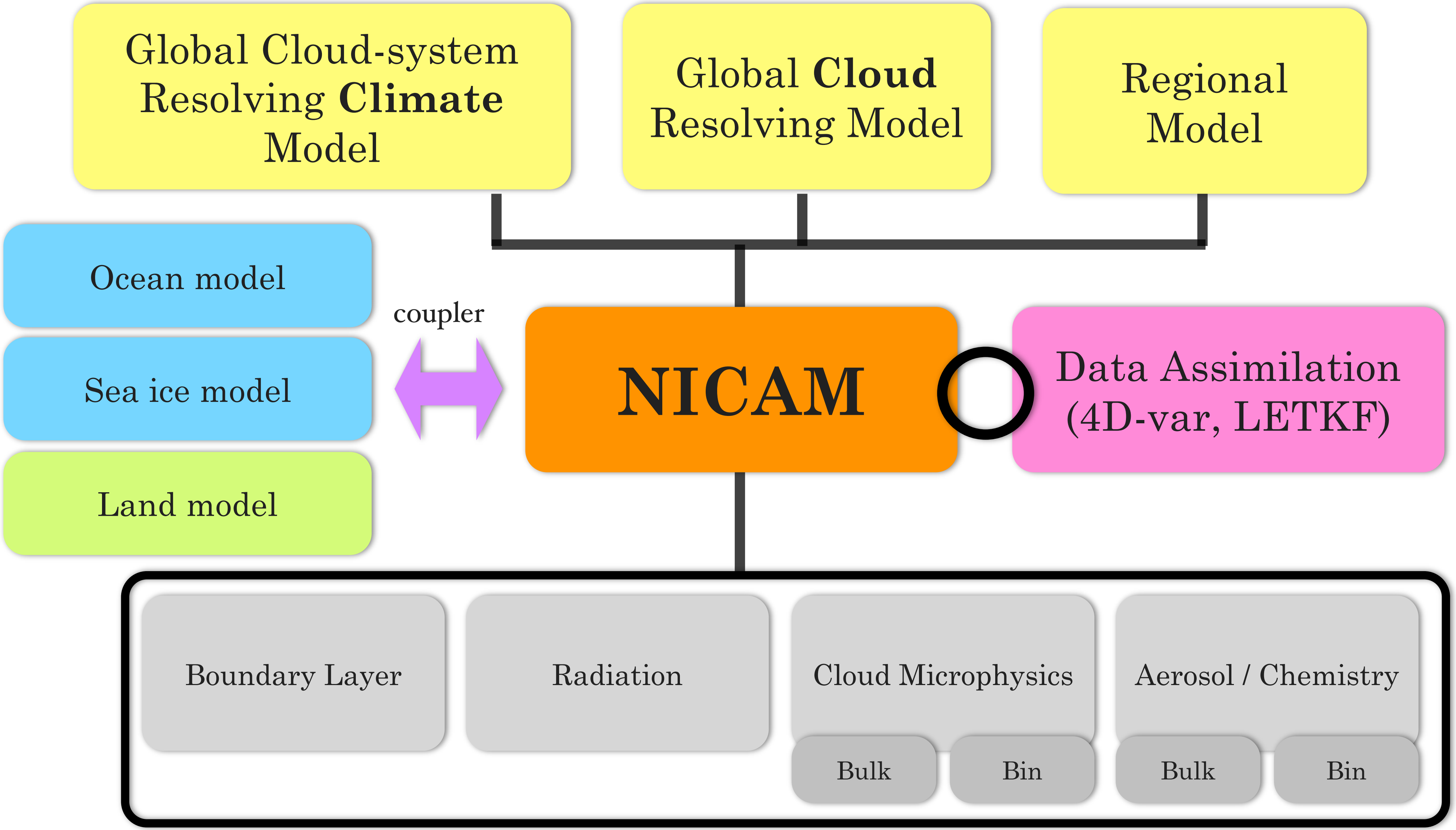
- U-Tokyo: Masaki Satoh (president)
  - Atmos.-Ocean coupling simulation, Cloud feedbacks research
- JAMSTEC
  - CMIP6, Equatorial region research (e.g. MJO), Typhoon research
  - Aerosol & chemistry research
- RIKEN: Hirofumi Tomita (vice president), Hisashi Yashiro (Chief developer)
  - Core development, optimization
  - Data assimilation development (NICAM-LETKF), by Dr. Miyoshi’s team
- NIES + MRI
  - GHGs simulation & DA (NICAM-TM, NICAM-4DVar)
  - Aerosol DA (NICAM-LETKF)
- JAXA
  - Aerosol and precipitation DA (NICAM-LETKF)

Main Projects Using NICAM

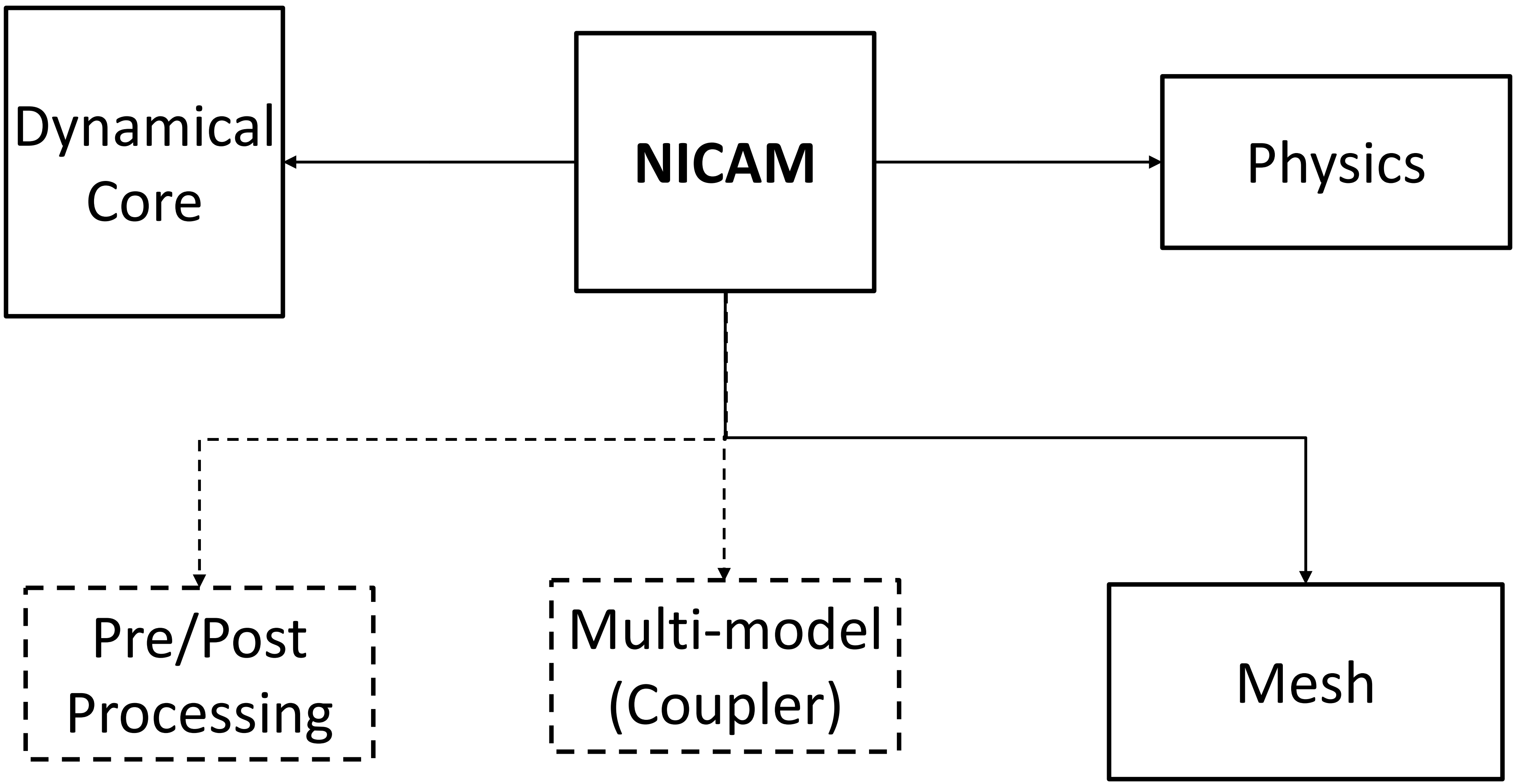
- The post-K computer
  - One of the target applications to evaluate the machine
  - Frontier studies with big simulation & big DA
- Tougou project (the K computer, Earth simulator)
  - Driving development and simulations for CMIP6 (Only HiresMIP, CFMIP)
- JAXA-PMM (the K computer, JAXA FX100)
  - DA with new precipitation satellites
- GOSAT-2 project (NIES SGI cluster + P100 GPU cluster)
  - A priori for the CO_2 satellite, estimation of global CO_2 emission
- Other projects in Japan (U-tokyo Oakforest-PACS, etc…)
  - ArCS: Arctic A-O coupling study, Typhoon in the future
- International collaborations
  - SPPEXA/AIMES, RCEMIP, DYAMOND project, CLIVAR, etc…

First Challenge: Complexity of the codebase

Components of a NICAM



Components of a NICAM



Dynamical Core

- Solve Navier-Stokes equations to simulate fluid in domain of interest
  - Atmosphere in NICAM's case

- Critical for performance
  - Most of application time spent in the dynamical core

- Computation
  - Hundreds of memory-bound loop nests (Stencils)
  - Neighbor communication
  - Horizontal dependency

- Mostly stable code
  - Written once; not changed frequently

Physics

- Physical parameterization
  - Radiation, microphysics, clouds … etc

- Computation
  - Mostly compute-bound (with branching)
  - No horizontal dependency (Column or point operations)
  - No Neighbor communication

- Actively changing codes
  - Scientists add/modify physics routines all the time

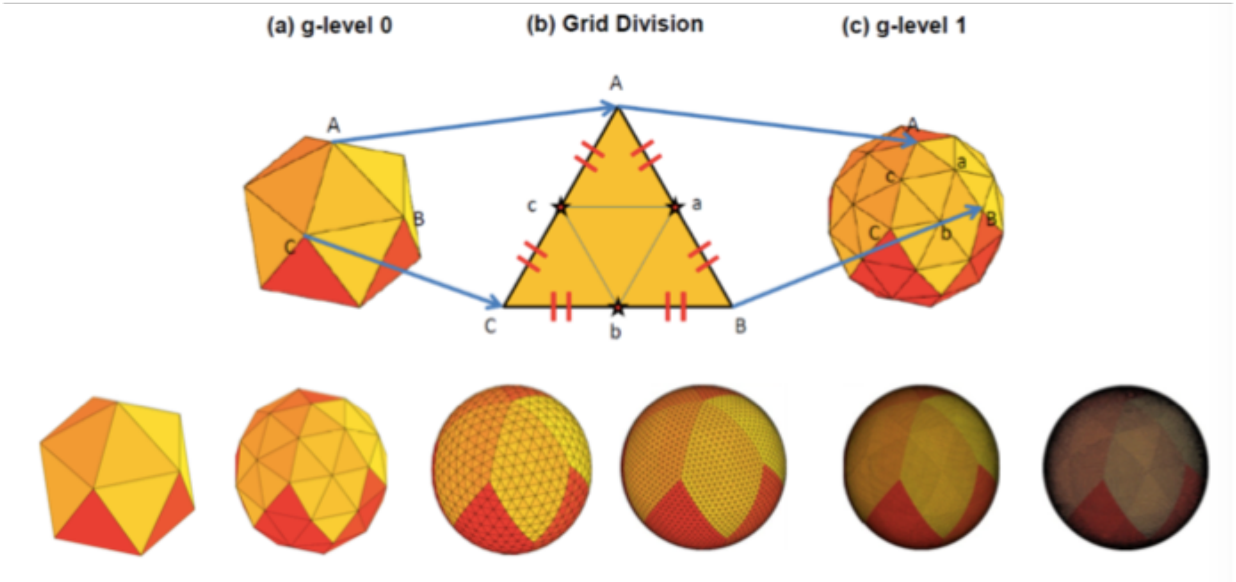
Mesh (1 of 2)

- Structured is more compute friendly
  - Regular access pattern of memory (and better locality)
  - Exchange of halo layers with neighbors is simple

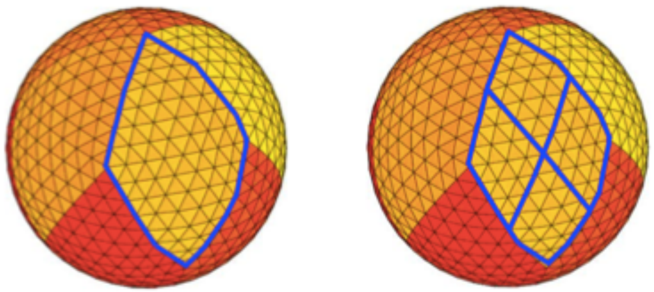
- NICAM, is icosahedral, yet manages the mesh as semi-structured
  - Regular computation
  - Pole points not included in regular region
  - Complex MPI communication scheme

Mesh (2 of 2)

- Grid points generated by recursive division

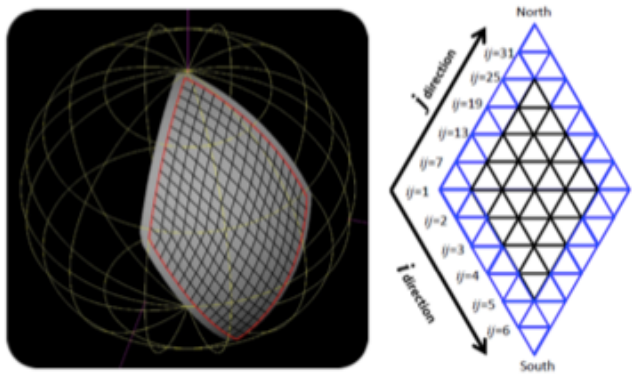


- Domain decomposition follows the same method: recursive



rlevel=0
10 regions

rlevel=1
40 regions



Other Components

- Coupling with an another model (ex: Ocean)

- Pre/Post Processing
  - Pre-processing
    - Data assimilation: Ensemble-based DA system (NICAM-LETKF)
    - Data assimilation: 4D-var DA system (NICAM-TM-4Dvar)

  - Post-processing
    - Remapping icosahedral grid to lat-lon grid (temporal bottleneck)

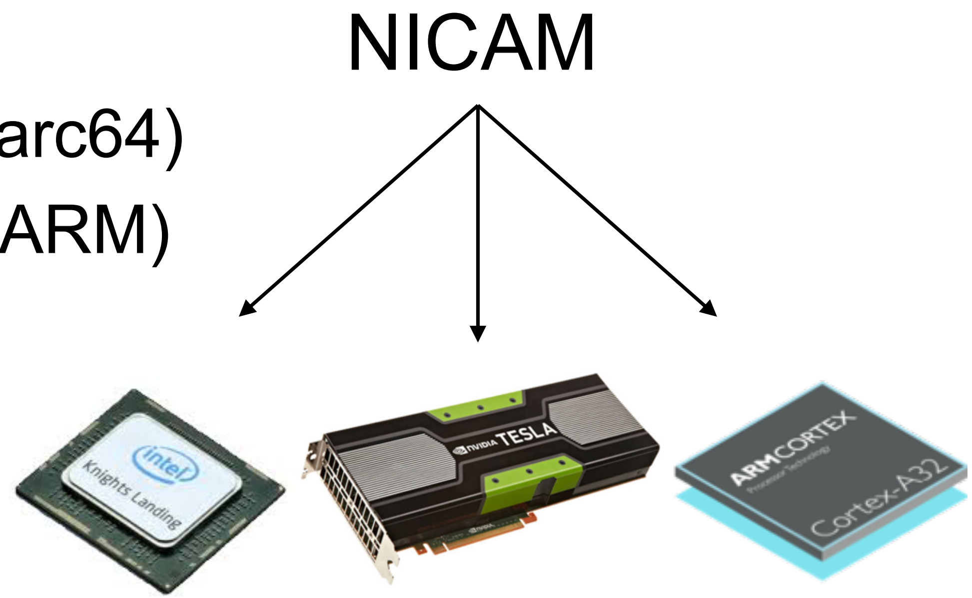
Summary

- DyCore: high-performance, low-productivity solution
  - GridTools
- Physics: average-performance, high productivity solution
  - GridTools Python, Hybrid Fortran, … etc
- Mesh: don't touch the communicator
  - Solutions for Icosahedral models are few and not mature

Second Challenge:
Performance portability

Performance Portability

- Performance portability is a real issue
  - Top three machines in Japan
  - T3 (GPU); OFP (KNL); K (Sparc64)
  - Future: ABCI (GPU); post-K (ARM)



- Qualifying NICAM atmospheric model for exscale
  - With a single codebase?
  - Excessive #if #def is not a single codebase
  - Directives not a good option either
    - No abstraction of the data layout in memory

About NICAM's Codebase

- Modular code
- Mostly written in Fortran90
  - MPI+OpenMP
  - Experimental OpenACC version of the dynamical core
- NICAM Full
  - Includes coupler, LETKF, excludes external ocean model
  - 330K lines in total, >40K in Dynamical core
- ~50 users, ~10 active developers

A path forward

- Collaborate with ETH/CSCS on DyCore
  - Using their GridTools Framework

- Explore solutions for the Physics
  - Python (ETH/CSCS)
  - HybridFortan (Prof. Aoki at TokyoTech)

- First we needed to investigate the prospect
  - NICAM benchmark

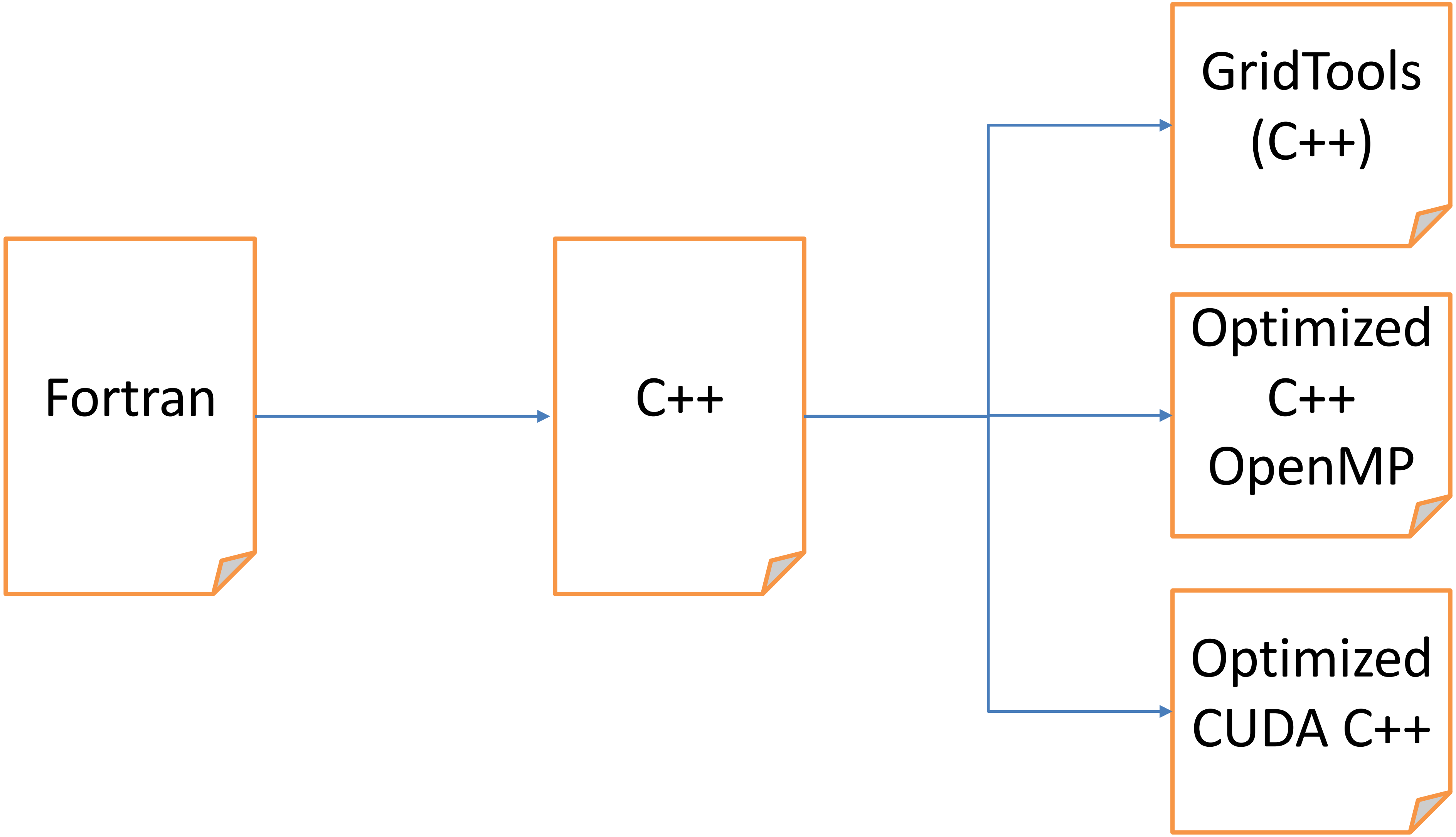
About NICAM Benchmarks

- Extracted from NICAM code
  - Note: benchmarks are proxy kernels

- They include benchmarks for:
    - Dynamical core: diffusion, divdamp, vi_rhow_solver
    - Physics: microphysics, radiation
    - Communicator: an elaborate communication scheme based on MPI
    - Note: all benchmarks include special code for the irregular polar regions

- Available publically on github
  - https://github.com/hisashiyashiro/nicam_dckernel_2016

Porting NICAM Benchmarks

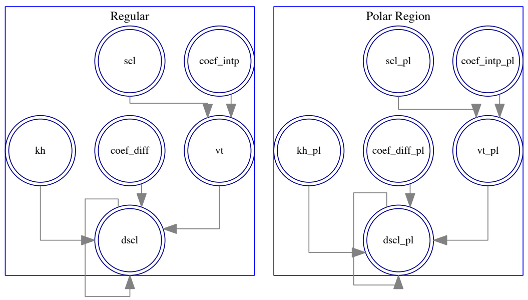


GridTools

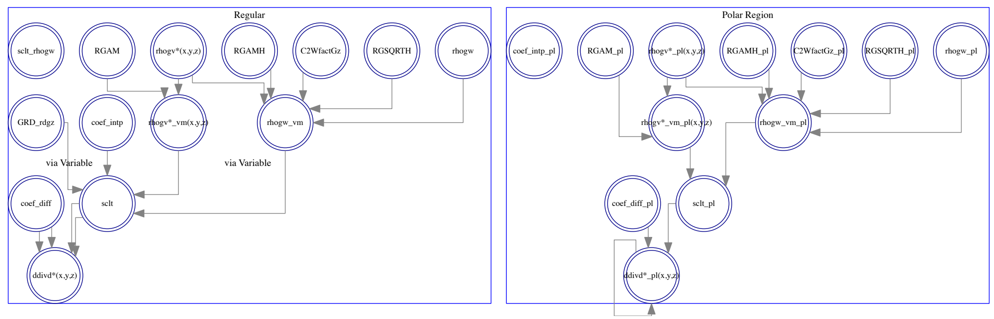
- A C++ framework for Solution of PDEs Using Stencils
  - Relies heavily on C++ templates
- Developed at ETH/CSCS
- Supports different backends
  - With emphasize on GPUs
- Used with COMSO
  - A production model

NICAM-DC Benchmarks

- Three benchmarks of the DyCore were evaluated for GridTools



Diffusion Benchmark



Divdamp Benchmark

- And currently evaluating the Physics benchmark
  - Using Hybrid Fortran (Prof. Aoki at TokyoTech)
  - A source-to-source framework

Results for NICAM Benchmark (1 of 2)

Table: Execution time (Seconds)

Benchmark	CUDA (Nvidia K40)			OpenMP (Broadwell-EP CPU E5-2630 v4 @2.20GHz)					
	Manual	Manual_opt (coal, sh_mem, occu, reg_pres)	GridTools	OMP_THREADS=1		OMP_THREADS=5		OMP_THREADS=10	
				Manual	GridTools	Manual	GridTools	Manual	GridTools
Diffusion	5.83	0.575 (5.23x OMP=10)	0.61 (4.93)	19.2 (1.0x)	19.4 (1.0x)	4.3 (4.46x)	5.8 (3.34x)	2.2 (8.72x)	3.01 (3.22x)
Divdamp (vgrid40_600m_24km)	35.23	3.15 (4.83x OMP=10)	3.17 (4.80x)	74.3 (1.0x)	74.3 (1.0x)	15.4 (4.81x)	30.08 (2.47x)	7.7 (4.78x)	15.22 (2.44x)
Vi_rhow_Solver (vgrid40_600m_24km)	0.91	0.288 (6.14x OMP=10)	0.311 (5.69x)	12.0 (1.0x)	12.1 (1.0x)	2.4 (4.9)	3.18 (3.8x)	1.23 (4.87)	1.77 (3.4x)

- Operation time only (I/O, validation, …, etc not included)
- Regular regions only (Polar regions excluded)
- 1000 iterations
- # regions = 1, # MPI ranks = 1, 130x130x42 grid

Results for NICAM Benchmark (2 of 2)

- Results were encouraging: promising results for GPUs
- Despite some challenges
  - C++ is never easy
    - Convoluted, verbose, and bloated
  - Dependency graph of operations must be analyzed
    - To improve the locality (performance issue, not correctness issue)
  - Special attention is given to the loop bounds and halo regions for every data arrays
  - Debugging is not trivial
    - A common feature in C++
- GridTools team made things easier (fully engaged)

Porting NICAM-DC

- The entire DyCore was ported

- An incremental approach
  - Port individual operator
  - Verify (use SerialBox2 tool to verify)
  - Move to the next operator

NICAM-DC Operators

- Ported incrementally to GridTools
  - Verified one-by-one

Snapshot of Input

Call the Operator

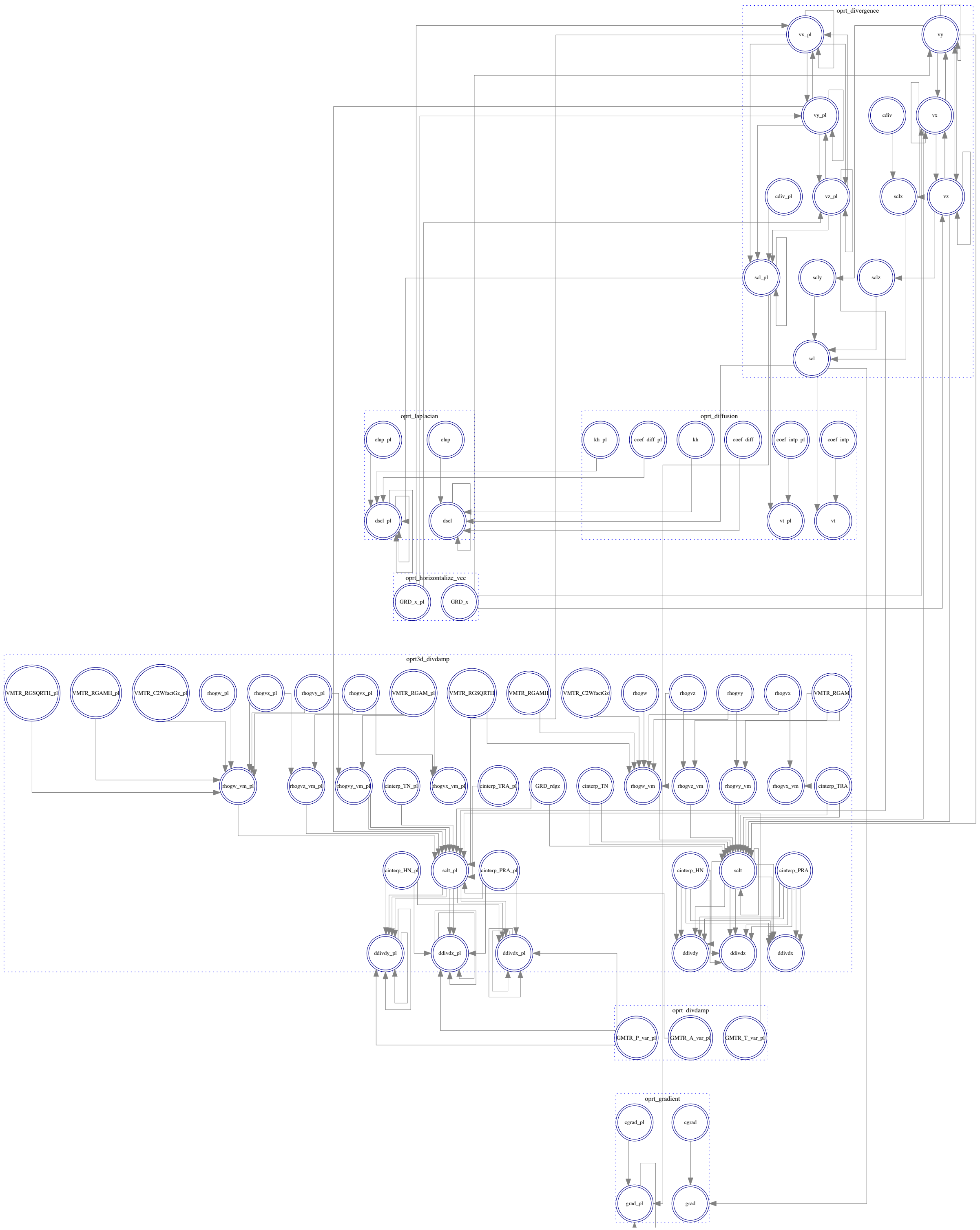
Snapshot of Output

```
!--- Horizontal flux convergence
 !$sersavepoint OPRT_divergence before
 !$ser mode write
 !$ser data ser scl=scl      (1:ADM_gall ,1:ADM_kall,1:ADM_lall )
 !$serdata ser vx=vx        (1:ADM_gall ,1:ADM_kall,1:ADM_lall )
 !$ser data ser vy=vy       (1:ADM_gall ,1:ADM_kall,1:ADM_lall )
 !$ser data ser vz=vz       (1:ADM_gall ,1:ADM_kall,1:ADM_lall )
 !$ser data ser coef_div=coef_div (1:ADM_nxyz ,1:ADM_gall,0:6   ,1:ADM_lall )
 call OPRT_divergence( div_rhogvh (:,:,:), div_rhogvh_pl (:,:,:),&! [OUT]
                  rhogvx_vm   (:,:,:), rhogvx_vm_pl   (:,:,:),&! [IN]
                  rhogvy_vm   (:,:,:), rhogvy_vm_pl   (:,:,:),&! [IN]
                  rhogvz_vm   (:,:,:), rhogvz_vm_pl   (:,:,:),&! [IN]
                  OPRT_coef_div(:,:,:,:), OPRT_coef_div_pl(:,:,:) )! [IN]
 !$sersavepointOPRT_divergence after
 !$sermodewrite
 !$serdataserscl=scl      (1:ADM_gall ,1:ADM_kall,1:ADM_lall )
```

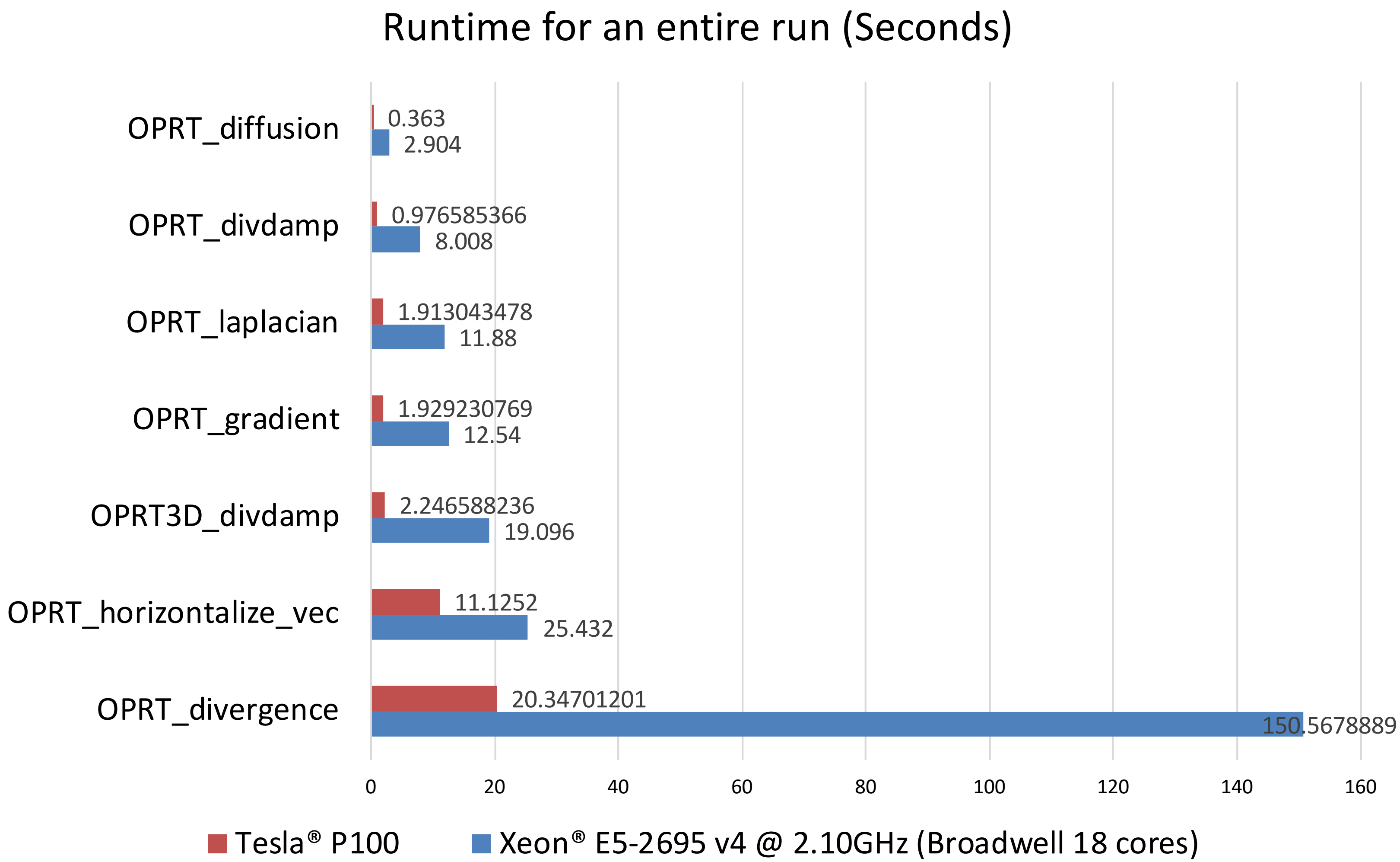
NICAM-DC Operators

```
do l = 1, ADM_lall
  do k = 1, kall
    do j = gmin, gmax
      do i = gmin, gmax
              ij     = (j-1)*iall + I
              ip1j   = ij + 1
              ip1jp1 = ij + iall + 1
              ijp1   = ij + iall
              im1j   = ij – 1
              im1jm1 = ij - iall -1
              ijm1   = ij – iall
              scl(ij,k,l) = scl(ij,k,l) + ( coef_div(XDIR,ij,0,l) * vx(ij    ,k,l) &
                    + coef_div(XDIR,ij,1,l) * vx(ip1j  ,k,l) &
                    + coef_div(XDIR,ij,2,l) * vx(ip1jp1,k,l) &
                    + coef_div(XDIR,ij,3,l) * vx(ijp1  ,k,l) &
                    + coef_div(XDIR,ij,4,l) * vx(im1j  ,k,l) &
                    + coef_div(XDIR,ij,5,l) * vx(im1jm1,k,l) &
                    + coef_div(XDIR,ij,6,l) * vx(ijm1  ,k,l) )
      enddo

      do i = gmin, gmax
              ij     = (j-1)*iall + I
        ip1j   = ij + 1
        ip1jp1 = ij + iall + 1
        ijp1   = ij + iall
              im1j   = ij – 1
              im1jm1 = ij - iall -1
              ijm1   = ij –
              scl(ij,k,l) = scl(ij,k,l) + ( coef_div(YDIR,ij,0,l) * vy(ij    ,k,l) &
                              + coef_div(YDIR,ij,1,l) * vy(ip1j  ,k,l) &
                              + coef_div(YDIR,ij,2,l) * vy(ip1jp1,k,l) &
                              + coef_div(YDIR,ij,3,l) * vy(ijp1  ,k,l) &
                              + coef_div(YDIR,ij,4,l) * vy(im1j  ,k,l) &
                              + coef_div(YDIR,ij,5,l) * vy(im1jm1,k,l)
                              + coef_div(YDIR,ij,6,l) * vy(ijm1  ,k,l) )
     enddo

     do i = gmin, gmax
              ij     = (j-1)*iall + I
              ip1j   = ij + 1
              ip1jp1 = ij + iall + 1
              ijp1   = ij + iall
              im1j   = ij – 1
              im1jm1 = ij - iall -1
              ijm1   = ij – iall
              scl(ij,k,l) = scl(ij,k,l) + ( coef_div(ZDIR,ij,0,l) * vz(ij    ,k,l) &
                              + coef_div(ZDIR,ij,1,l) * vz(ip1j  ,k,l) &
                              + coef_div(ZDIR,ij,2,l) * vz(ip1jp1,k,l) &
                              + coef_div(ZDIR,ij,3,l) * vz(ijp1  ,k,l) &
                              + coef_div(ZDIR,ij,4,l) * vz(im1j  ,k,l) &
                              + coef_div(ZDIR,ij,5,l) * vz(im1jm1,k,l) &
                              + coef_div(ZDIR,ij,6,l) * vz(ijm1  ,k,l) )
     enddo
   enddo
  enddo
enddo
```

```
struct OPRT_divergence_operator_01 {

typedef accessor<0, enumtype::inout, extent<0, 0, 0, 0>, 4> scl
typedef accessor<1, enumtype::in, extent<0, 0, 0, 0>, 4 > vx
typedef accessor<2, enumtype::in, extent<0, 0, 0, 0>, 4 > vy
typedef accessor<3, enumtype::in, extent<0, 0, 0, 0>, 4 > vz;
typedef accessor<4, enumtype::in, extent<0, 0, 0, 0>, 6 > coef_div;
typedef boost::mpl::vector<scl, vx, vy, vz, coef_div> arg_list;

template <typename evaluation>
GT_FUNCTION
static void Do(evaluation const & eval) {
dimension<1> i;             dimension<2> j;     dimension<3> k;
dimension<4> l;             dimension<1> d;     dimension<2> i_coef_div;
dimension<3> j_coef_div;    dimension<4> c;     dimension<5> l_coef_div;

eval(scl{i,j,k,l})  = eval(scl{i,j,k,l})
              + ( eval(coef_div{d,i_coef_div,j_coef_div,c+0,l_coef_div}) * eval(vx{i,j,k,l})
              +   eval(coef_div{d,i_coef_div,j_coef_div,c+1,l_coef_div}) * eval(vx{i+1,j,k,l
              +   eval(coef_div{d,i_coef_div,j_coef_div,c+2,l_coef_div}) * eval(vx{i+1,j+1,k,l})
              +   eval(coef_div{d,i_coef_div,j_coef_div,c+3,l_coef_div}) * eval(vx{i,j+1,k,l})
              +   eval(coef_div{d,i_coef_div,j_coef_div,c+4,l_coef_div}) * eval(vx{i-1,j,k,l})
              +   eval(coef_div{d,i_coef_div,j_coef_div,c+5,l_coef_div}) * eval(vx{i-1,j-1,k,l})
              +   eval(coef_div{d,i_coef_div,j_coef_div,c+6,l_coef_div}) * eval(vx{i,j-1,k,l}));

eval(scl{i,j,k})  = eval(scl{i,j,k})
              + ( eval(coef_div{d,i_coef_div,j_coef_div,c+0,l_coef_div}) * eval(vy{i,j,k})
              +   eval(coef_div{d,i_coef_div,j_coef_div,c+1,l_coef_div}) * eval(vy{i+1,j,k})
              +   eval(coef_div{d,i_coef_div,j_coef_div,c+2,l_coef_div}) * eval(vy{i+1,j+1,k})
              +   eval(coef_div{d,i_coef_div,j_coef_div,c+3,l_coef_div}) * eval(vy{i,j+1,k})
              +   eval(coef_div{d,i_coef_div,j_coef_div,c+4,l_coef_div}) * eval(vy{i-1,j,k})
              +   eval(coef_div{d,i_coef_div,j_coef_div,c+5,l_coef_div}) * eval(vy{i-1,j-1,k})
              +   eval(coef_div{d,i_coef_div,j_coef_div,c+6,l_coef_div}) * eval(vy{i,j-1,k}));

eval(scl{i,j,k,l})  = eval(scl{i,j,k,l})
              + ( eval(coef_div{d,i_coef_div,j_coef_div,c+0,l_coef_div}) * eval(vz{i,j,k,l})
              +   eval(coef_div{d,i_coef_div,j_coef_div,c+1,l_coef_div}) * eval(vz{i+1,j,k,l})
              +   eval(coef_div{d,i_coef_div,j_coef_div,c+2,l_coef_div}) * eval(vz{i+1,j+1,k,l})
              +   eval(coef_div{d,i_coef_div,j_coef_div,c+3,l_coef_div}) * eval(vz{i,j+1,k,l})
              +   eval(coef_div{d,i_coef_div,j_coef_div,c+4,l_coef_div}) * eval(vz{i-1,j,k,l})
              +   eval(coef_div{d,i_coef_div,j_coef_div,c+5,l_coef_div}) * eval(vz{i-1,j-1,k,l})
              +   eval(coef_div{d,i_coef_div,j_coef_div,c+6,l_coef_div}) * eval(vz{i,j-1,k,l}));
 }
};
```


oprt_divergence
vx_pl
vy
vy_pl
cdiv
vx
cdiv_pl
vz_pl
sclx
vz
scl_pl
scly
sclz
scl
oprt_laplacian
clap_pl
clap
dscl_pl
dscl
oprt_diffusion
kh_pl
coef_diff_pl
kh
coef_diff
coef_intp_pl
coef_intp
vt_pl
vt
oprt_horizontalize_vec
GRD_x_pl
GRD_x
oprt3d_divdamp
VMTR_RGSQRTH_pl
VMTR_RGAMH_pl
VMTR_C2WfactGz_pl
rhogw_pl
rhogvz_pl
rhogvy_pl
rhogvx_pl
VMTR_RGAM_pl
VMTR_RGSQRTH
VMTR_RGAMH
VMTR_C2WfactGz
rhogw
rhogvz
rhogvy
rhogvx
VMTR_RGAM
rhogw_vm_pl
rhogvz_vm_pl
rhogvy_vm_pl
cinterp_TN_pl
rhogvx_vm_pl
cinterp_TRA_pl
GRD_rdgz
cinterp_TN
rhogw_vm
rhogvz_vm
rhogvy_vm
rhogvx_vm
cinterp_TRA
cinterp_HN_pl
sclt_pl
cinterp_PRA_pl
cinterp_HN
sclt
cinterp_PRA
ddivdy_pl
ddivdz_pl
ddivdx_pl
ddivdy
ddivdz
ddivdx
oprt_divdamp
GMTR_P_var_pl
GMTR_A_var_pl
GMTR_T_var_pl
oprt_gradient
cgrad_pl
cgrad
grad_pl
grad

NICAM-DC Operators



- Running on 10 nodes: one MPI rank per node
- P100 uses GridTools generated kernels
- Xeon uses original Fortran code (OpenMP)
- Test case: ICOMEX_JW/gl05rl00z40pe10
- Total number of grid (horizontal) : 10240 (32 x 32 x 10)
- Number of vertical layers : 40
- Max of large step: 72, Max of small step: 6

GridTools Fortran Interface

- A prototype for using GridTools, from with in Fortran
  - Replace every call to a Fortran subroutine operator, with a call to a GridTools stencil sunctor(s)

```
dycore_repository=alloc_wrapped_dycore_repository(3, dim) !pass dimensions for the storage
dycore_repository_explicit=convert_dycore_repo(dycore_repository)
dycore=alloc_mini_dycore(3, dim, dycore_repository_explicit) !pass dimensions for the grid

call gt_push(dycore_repository,"in",in)
call gt_push(dycore_repository,"out",out)
call oprt_divergence(dycore)
call gt_pull(dycore_repository,"out",out)
```

Call to GridTools Stencil

Points to consider (1 of 3)

- Host code populating work arrays in-between operators
  - Move to a separate GridTools stencil (maintain program logic)
  - Move it to inside an operator (better performance)

```
call OPRT_laplacian( vtmp2          (:,:,:,4), vtmp2_pl          (:,:,:,4), & !  [OUT]
                     vtmp_lap1      (:,:,:,4), vtmp_lap1_pl      (:,:,:,4), & !  [IN]
                     OPRT_coef_lap (:,:,:),    OPRT_coef_lap_pl (:,:)     )   !  [IN]
```

Work arrays

```
!$omp parallel workshare
wk   (:,:,:) = rhog   (:,:,:) * CVdry * KH_coef_lap1   (:,:,:)
!$omp end parallel workshare
wk_pl(:,:,:) = rhog_pl(:,:,:) * CVdry * KH_coef_lap1_pl(:,:,:)
```

```
call OPRT_diffusion( vtmp2          (:,:,:,5),   vtmp2_pl          (:,:,:,5), & ! [OUT]
                     vtmp_lap1      (:,:,:,5),   vtmp_lap1_pl      (:,:,:,5), & ! [IN]
                     wk             (:,:,:),     wk_pl             (:,:,:),   & ! [IN]
                     OPRT_coef_intp (:,:,:,:,:), OPRT_coef_intp_pl (:,:,:,:), & ! [IN]
                     OPRT_coef_diff (:,:,:,:),   OPRT_coef_diff_pl (:,:,:)   )  ! [IN]
```

Points to consider (2 of 3)

- Setup code
  - Ex: initializing data arrays
  - Leave as is

- Moving data from Device to Host
  - Logging, check pointing, … etc
  - Blocking “sync” operation (or “clone”)

Points to consider (3 of 3)

- Sliced data arrays
  - Used extensively in original code
  - Not in GridTools (or CUDA Fortran)
  - Performance penalty of copying/moving entire array

- Numerical stability errors when running NICAM-DC
  - Fine on local machine, error on Piz Daint
  - Cray compilers on Piz Daint
    - Some compiler flag(s)?

Summary

- Performance portability is a real challenge
  - Specially for legacy codes

- Climate models are complex codebases
  - Different components: different optimizing/adapting strategies

- NICAM, an atmospheric model, is an example of that
  - We discuss the ongoing effort for the transition of NICAM

Moving forward

- We have an ambitious plan to push NICAM forward
- A full production-level solution is far ahead
- Not easy to engage all parties (a political challenge)
- We will keep you updated in the next ACAC ☺