Porting the ICON Non-hydrostatic Dynamics to GPUs

William Sawyer (CSCS/ETH), Christian Conti (ETH),
Gilles Fourestey (CSCS/ETH)

IS-ENES Workshop on Dynamical Cores

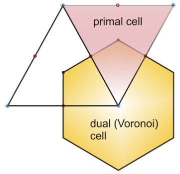
Dec. 14-16, 2011, Carlo V Castle, Lecce, Italy

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

The ICON Model

- ICOsahedral Non-hydrostatic model
- Multi-resolution grid (not supported here)
- Triangular cells
- Conservation laws
- ‘Bandwidth limited’
- Extensive use of indexing arrays
- Developers: MPI-M, DWD



ICON-GPU Project

- CSCS/C2SM offered its assistance with GPUs
- Funding through PRACE 2nd Impl. Phase work package 8
- Goal: compare GPU paradigms in terms of efficiency, usability and developer friendliness
- Non-hydrostatic solver (~5K l.o.c.), and physical parameterizations
- Paradigms chosen: OpenCL, CUDAFortran for dynamics, accelerator directives (PGI/Cray) for physical parameterizations
- OpenCL NH solver: 6 weeks, by PhD student (Conti)
- CUDAFortran NH solver: ~8 weeks (Sawyer)
- Xavier Lapillonne (C2SM/ETH): microphysics, radiation, turbulence with directives for ICON/COSMO physics

OPERATIONAL INTENSITIES

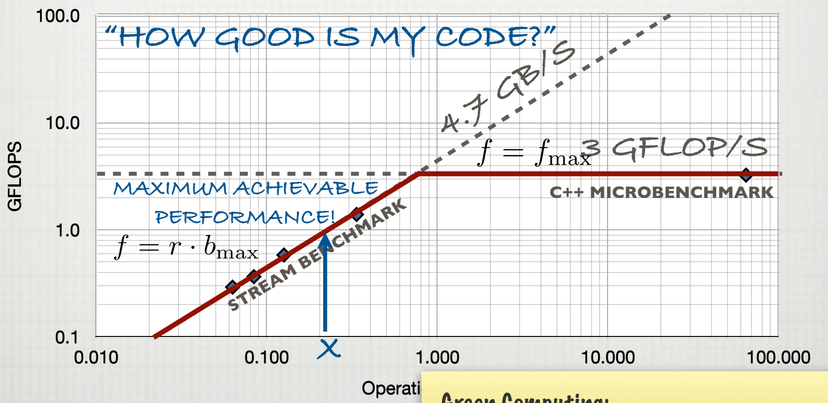
	add $z_i = x_i + y_i$	scale $z_i = \alpha x_i$	triad $z_i = \alpha x_i + y_i$
Intel Xeon W3520	$\frac{1}{12}$	$\frac{1}{8}$	$\frac{2}{12}$
4P AMD Opteron 8380	$\frac{1}{16}$	$\frac{1}{12}$	$\frac{2}{16}$
2P AMD Opteron 2435	$\frac{1}{12}$	$\frac{1}{8}$	$\frac{2}{12}$
NVIDIA Tesla S1070	$\frac{1}{12}$	$\frac{1}{8}$	$\frac{2}{12}$

OPERATIONAL INTENSITY $R = \dfrac{1}{4*(1 \text{ WRITE} + 2 \text{ READS})}$

THE ROOFLINE MODEL

S. Williams, A. Waterman, D. Patterson, "*Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures*", Communications of the ACM (CACM), April 2009.

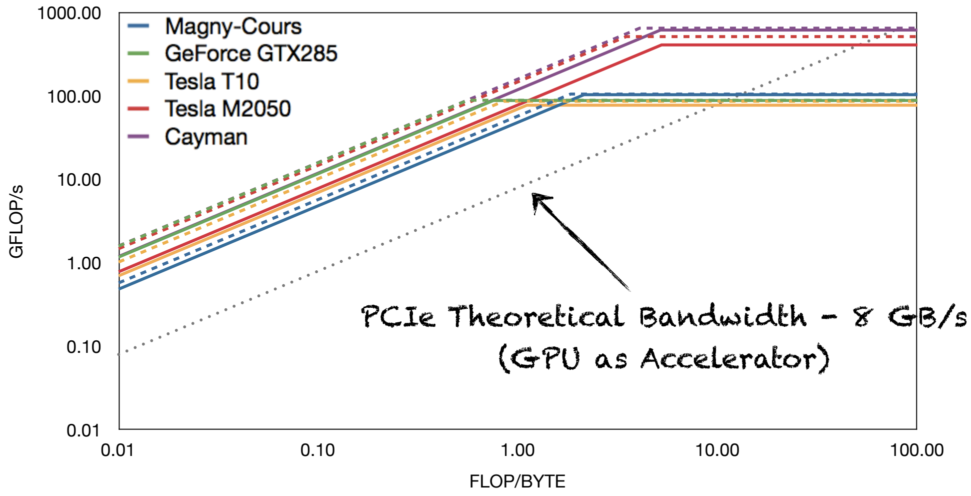
- ☐ OPERATIONAL INTENSITY R = FLOPS/MEMORY TRAFFIC (BYTES)
- ☐ PERFORMANCE MODEL FOR BOTH GPU AND CPU



◆ 4x Quad-Core AMD Opteron 8380 @ 2.5GHz - 1 T

Green Computing:
- computationally bound: reduce bus clock/s
- memory bound: reduce processor clock/s

Roofline of Various GPUs



Porting NH solver to GPUs

Fortran

```fortran
DO jb = i_startblk, i_endblk

  CALL get_indices_c(p_patch, jb, i_startblk, i_endblk, &
                     i_startidx, i_endidx, rl_start, rl_end)

  DO jk = 1, nlev
    DO jc = i_startidx, i_endidx
```

OpenCL

```c
const int jb = i_startblk + get_global_id(0);
const int jc = localStart[get_global_id(0)] + get_global_id(2);
const int jk = get_global_id(1);

if (jk < nlev && jb < i_endblk && jc < localEnd[get_global_id(0)])
{
    const int idx = jc + jk*nproma + jb*nproma*nlev;
```

CUDAFortran

```fortran
jb = blockidx%x + ( i_startblk -1 )
je = threadidx%x
jk = threadidx%y

IF ( ( i_startblk < jb .and. jb < i_endblk ) .or. &
     ( i_startblk == jb .and. i_startidx <= je ) .or. &
     ( i_endblk == jb .and. je <= i_endidx ) ) THEN
```

CUDAFortran Example

kernel invocation

```
  prepare_e(2,min_rledge_int-2)
  copy_blk_idx_to_gpu
  gpu_set(nproma,nlev)
  CALL idiv_1_z_vn_avg <<<g, b>>> ( i_startblk_d, i_endblk_d, i_startidx_d, i_endidx_d, &
&                                   nnew_d, quad_blk_d, quad_idx_d, vn_d, e_flx_avg_d,  &
&                                   z_vn_avg_d )
```

```
rl_start = 2
rl_end   = min_rledge_int-2
DO jb = i_startblk, i_endblk
  CALL get_indices_e(p_patch, jb, i_startblk, i_endblk, &
                     i_startidx, i_endidx, rl_start, rl_end)
  DO je = i_startidx, i_endidx
    DO jk = 1, nlev
      z_vn_avg(je,jk,jb) = p_nh%prog(n_new)%vn(je,jk,jb)*p_int%e_flx_avg(je,1,jb)         &
        + p_nh%prog(n_new)%vn(iqidx(je,jb,1),jk,iqblk(je,jb,1))*p_int%e_flx_avg(je,2,jb) &
        + p_nh%prog(n_new)%vn(iqidx(je,jb,2),jk,iqblk(je,jb,2))*p_int%e_flx_avg(je,3,jb) &
        + p_nh%prog(n_new)%vn(iqidx(je,jb,3),jk,iqblk(je,jb,3))*p_int%e_flx_avg(je,4,jb) &
        + p_nh%prog(n_new)%vn(iqidx(je,jb,4),jk,iqblk(je,jb,4))*p_int%e_flx_avg(je,5,jb)
    ENDDO
  ENDDO
ENDDO
```

kernel content

```
ATTRIBUTES (GLOBAL) &
& SUBROUTINE idiv_1_z_vn_avg( i_startblk, i_endblk, i_startidx, i_endidx, &
     &                        nnew, iqblk, iqidx, vn, e_flx_avg, z_vn_avg )
INTEGER, INTENT(IN)  :: i_startblk  !
   :
INTEGER, INTENT(IN)  :: iqblk(:,:,:)
INTEGER, INTENT(IN)  :: iqidx(:,:,:)
REAL(wp), INTENT(IN) :: vn(:,:,:,:)
REAL(wp), INTENT(IN) :: e_flx_avg(:,:,:)
REAL(wp), INTENT(OUT):: z_vn_avg(:,:,:)

  jb = blockidx%x + ( i_startblk -1 )
  je = threadidx%x
  jk = threadidx%y ! [1 .. nlev]

   IF ( ( i_startblk < jb .and. jb < i_endblk ) .or. &
        ( i_startblk == jb .and. i_startidx <= je ) .or. &
        ( i_endblk == jb .and. je <= i_endidx ) ) THEN

        z_vn_avg(je,jk,jb) = vn(je,jk,jb,nnew)*e_flx_avg(je,1,jb)        &
          + vn(iqidx(je,jb,1),jk,iqblk(je,jb,1),nnew)*e_flx_avg(je,2,jb) &
          + vn(iqidx(je,jb,2),jk,iqblk(je,jb,2),nnew)*e_flx_avg(je,3,jb) &
          + vn(iqidx(je,jb,3),jk,iqblk(je,jb,3),nnew)*e_flx_avg(je,4,jb) &
          + vn(iqidx(je,jb,4),jk,iqblk(je,jb,4),nnew)*e_flx_avg(je,5,jb)

   ENDIF

END SUBROUTINE idiv_1_z_vn_avg
```

OpenCL/CUDAFortran Approaches

OpenCL

- Minimal refactoring
- Extensive use of local (shared) memory
- Iteration space: 1D or 2D
- Blocking factor: nproma=1 optimal
- Simpler but more kernels, fewer IFs in kernels

PGI CUDAFortran

- Refactored to remove intermediate arrays
- More use of registers
- 1-D grid of thread blocks, each with 2D distribution
- nproma=8/16 optimal
- Fewer kernels, more IFs

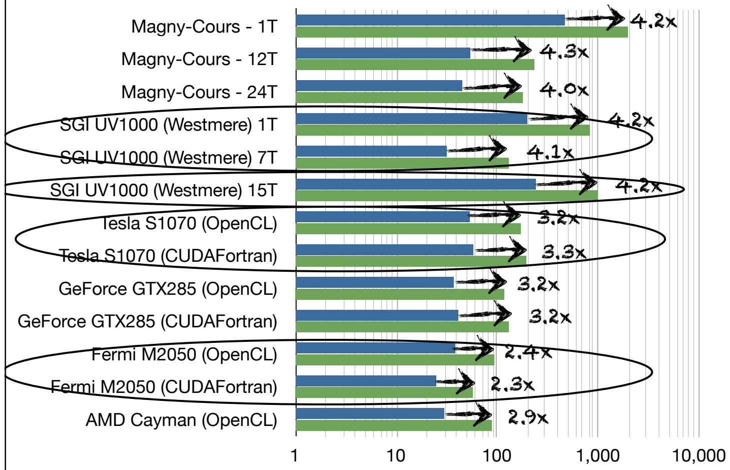
Implicit Vertical Solver

- Implicit solver requires a tridiagonal solution for each vertical column
- All 2-D arrays except one (z_q) can be replaced with registers; CUDAFortran version makes use of this

```
DO jk = 2, nlev
  DO jc = i_startidx, i_endidx
    z_gamma(jc,jk) =  dtime*cpd*p_nh%metrics%vwind_impl_wgt(jc,jb)* &
      p_nh%diag%theta_v_h(jc,jk,jb)/p_nh%metrics%ddqz_z_half(jc,jk,jb)
    z_a(jc,jk) = -z_gamma(jc,jk)*z_beta(jc,jk-1)*z_alpha(jc,jk-1)
    z_c(jc,jk) = -z_gamma(jc,jk)*z_beta(jc,jk  )*z_alpha(jc,jk+1)
    z_b(jc,jk) = 1.0_wp+z_gamma(jc,jk)*z_alpha(jc,jk) &
      *(z_beta(jc,jk-1)+z_beta(jc,jk))
  ENDDO
ENDDO
DO jk = 3, nlev
  DO jc = i_startidx, i_endidx
    z_g(jc,jk) = 1.0_wp/(z_b(jc,jk)+z_a(jc,jk)*z_q(jc,jk-1))
    z_q(jc,jk) = - z_c(jc,jk)*z_g(jc,jk)
  ENDDO
ENDDO
 :
DO jk = 3, nlev          ! Sweep down
  DO jc = i_startidx, i_endidx
    p_nh%prog(n_new)%w(jc,jk,jb) = (p_nh%prog(n_new)%w(jc,jk,jb)  &
      -z_a(jc,jk)*p_nh%prog(n_new)%w(jc,jk-1,jb))*z_g(jc,jk)
  ENDDO
ENDDO
DO jk = nlev-1, 2, -1   ! Sweep up
  DO jc = i_startidx, i_endidx
    p_nh%prog(n_new)%w(jc,jk,jb) = p_nh%prog(n_new)%w(jc,jk,jb)&
    &             +p_nh%prog(n_new)%w(jc,jk+1,jb)*z_q(jc,jk)
  ENDDO
ENDDO
```

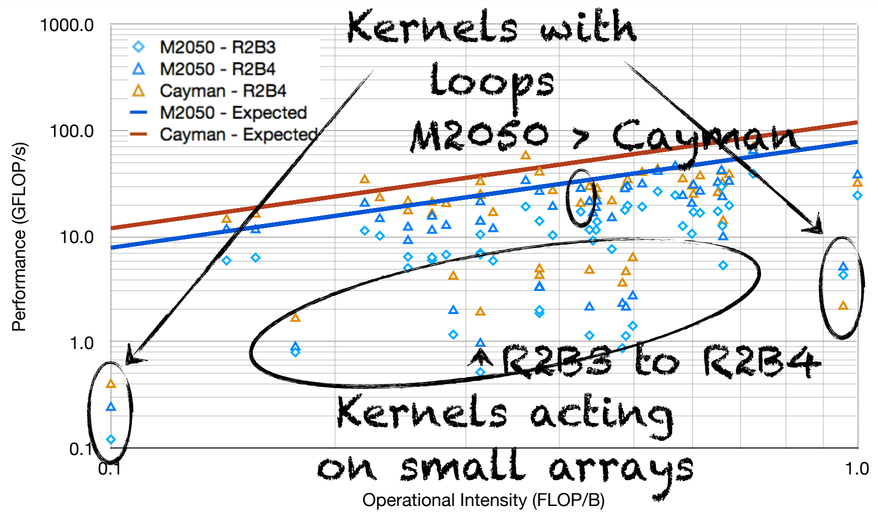
```
jb = blockidx%x + ( i_startblk -1 )  ! [i_startblk .. i_endblk]
jc = threadidx%x                     ! [1 .. nproma]
IF ( ( i_startblk < jb .and. jb < i_endblk ) .or. &
     ( i_startblk == jb .and. i_startidx <= jc ) .or. &
     ( i_endblk == jb .and. jc <= i_endidx ) ) THEN

    z_c = -z_gamma(jc,2,jb)*z_beta(jc,2,jb)*z_alpha(jc,3,jb)
    z_b = 1.0_wp+z_gamma(jc,2,jb)*z_alpha(jc,2,jb) &
          *(z_beta(jc,1,jb)+z_beta(jc,2,jb))
    z_q(jc,2) = -z_c/z_b
    w(jc,2,jb,n_new)= w(jc,2,jb,n_new)/z_b

    DO jk = 3, nlev    ! sweep down
        z_a = -z_gamma(jc,jk,jb)*z_beta(jc,jk-1,jb)*z_alpha(jc,jk-1,jb)
        z_c = -z_gamma(jc,jk,jb)*z_beta(jc,jk,jb)*z_alpha(jc,jk+1,jb)
        z_b = 1.0_wp+z_gamma(jc,jk,jb)*z_alpha(jc,jk,jb) &
          *(z_beta(jc,jk-1,jb)+z_beta(jc,jk,jb))

        z_g = 1.0_wp/(z_b+z_a*z_q(jc,jk-1))
        z_q(jc,jk) = - z_c*z_g
        w(jc,jk,jb,n_new) = (w(jc,jk,jb,n_new) -z_a*w(jc,jk-1,jb,n_new))*z_g
    ENDDO
    DO jk = nlev-1, 2, -1  ! Solve tridiagonal matrix for w, sweep up
        w(jc,jk,jb,n_new) = w(jc,jk,jb,n_new)+w(jc,jk+1,jb,n_new)*z_q(jc,jk)
    ENDDO
ENDIF
```

CPUs vs. GPUs



R2B03 (s.)	R2B04 (s.)
466	1954
54	231
45	179
197	820
31	129
239	992
53	171
58	193
37	117
41	130
38	93
25	57
30	88

OpenCL Kernels



CUDAFortran Time Distribution

```
calls   t_min       t_average   t_max       t_total
-------------------------------------------------------------------------------
total                  1     57.547s     57.547s     57.547s     57.547s    57.547
solve_nh            1000      .05614s     .05679s     .06111s    56.790s    56.790
nh_driver             10      5.722s      5.755s      5.886s     57.547s    57.547
intp                   1      .01501s     .01501s     .01501s     .01501s    0.015
vel tendencies      2000      .00797s     .00987s     .01238s    19.733s    19.733
cells to edges      2000      .00000s     .00044s     .00100s     .87150s    0.872
exner value         2000      .00007s     .00072s     .00193s     1.444s     1.444
rho and ddz_exner   2000      .00077s     .00101s     .00147s     2.011s     2.011
horizontal calcs    2000      .00104s     .00187s     .00296s     3.742s     3.742
rbf vt calc         2000      .00083s     .00090s     .00105s     1.798s     1.798
vn avg              2000      .00106s     .00107s     .00120s     2.149s     2.149
vn vt covariant ma  2000      .00374s     .00382s     .00455s     7.643s     7.643
div-related         2000      .00067s     .00069s     .00086s     1.379s     1.379
vertical calcs      2000      .00367s     .00375s     .00422s     7.500s     7.500
tridiagonal solver  2000      .00043s     .00044s     .00059s     .88492s    0.885
post calcs          2000      .00312s     .00345s     .00409s     6.901s     6.901
device copies          1      .17517s     .17517s     .17517s     .17517s    0.175
-------------------------------------------------------------------------------
```

- More optimizations possible!
- “vel tendencies” consists of 13 kernels, “vertical calcs” 5 kernels, “vn vt covariant” also 5, but still they seem to contain bottlenecks
- Device copies and tridiagonal solver appear not to be a problem

Aggregated NH Performance (DP)

- Fermi M2050 (CUDAFortran):
  - R2B3: 18.8 GFLOP/s
  - R2B4: 33.0 GFLOP/s
- Cayman (OpenCL):
  - R2B4: 21.2 GFLOP/s

Lessons learned

- Never underestimate the potential of a smart, motivated graduate student!
- CUDA/OpenCL programming not that difficult, but highly error-prone; debugging options limited
- CUDAFortran is much more 'appealing' to developers, but OpenCL is a portable paradigm
- Optimizations to both versions still possible

Future Work in PRACE 2IP WP8

- PRACE 2nd Implementation Phase work package 8: funding for 18 more months
- Overriding goal: *producing code that is useful to the community*
- Work directly with ICON development code base
- Planned: full MPI-GPU implementation, GPU programming paradigm to be defined...
- Computing challenge: GPU-to-GPU halo updates
- Use domain-specific language to describe solver; library to implement kernel operations?
- Other plans: looking for your input...

Acknowledgments

- Funding: PRACE 2IP WP8
- Leonidas Linardakis (MPI-M): technical support of ICON testbed code
- ICON team (DWD/MPI-M): collaborative effort

Physics Parameterizations

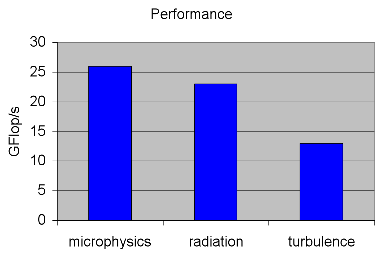
- To be shared between ICON and COSMO (European regional model)
- Currently ported to GPUs:
  - PGI : microphysics (hydci_pp), radiation (fesft), turbulence (only turbdiff yet)
  - OMP – acc (Cray) : microphysics, radiation
  - GPU optimization: loop reordering, replacement of arrays with scalars
  - Note: hydci_pp, fesft and turbdiff subroutines represents respectively 6.7%, 8% and 7.3% of the total execution time of a typical cosmo-2 run.

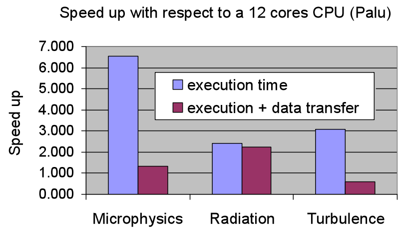
Physical Parameterizations

- 2D data fields inside the physics packages with one horizontal and one vertical dimensions: f (nproma,ke), with nproma = ie x je / nblock.
- 2D data fields inside the physics packages with one horizontal and one vertical dimensions: f (nproma,ke), with nproma = ie x je / nblock.
- Goals:
  - Parameterizations to be shared with COSMO (regional) model
  - Blocking strategy: all physics parametrization could be computed while data remains in the cache

```
call init_radiation
call init_turbulence
  …
do ib=1,nblock
  call copy_to block
  call organize_radiation
  …
  call
  organize_turbulence
  call copy_back
end do
```

Physics Performance



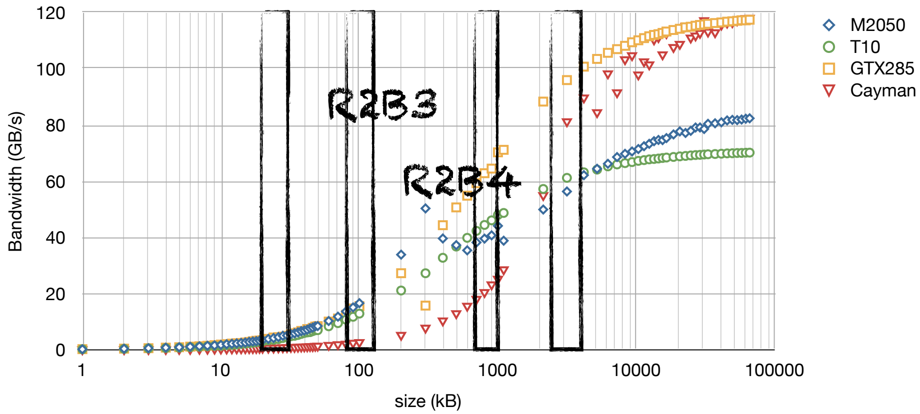


- Peak performance of Fermi card for double precision is 515 GFlop/s, i.e.,5%, 4.5% and 2.5% of peak performance for the microphysics, radiation and turbulence schemes, respectively

- Parallel CPU code run on 12 cores AMD magny-cours CPU – however there are no mpi-communications in these standalone test codes.

- Note the peak performance of Fermi card is 5 times that of the magny cours processor. Overhead of data transfer for microphysics and turbulence is very large.

GPU Bandwidths



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

Data Challenges

- Dycores (computational PDEs): native grid is increasingly incomprehensible.
- Data volume will change practices
- post- and pre-processing: now same order of magnitude as modeling to do the analysis
- Climate run restarts: becoming very large for high resolutions, must be done more often due to higher node failure rate, or use fault tolerant algorithms
- Metadata model for grids is missing... not only data model needed, but also tools to access the data. Models spit out too little metadata, e.g., the definition of the function space (FV, FE, ..)

Performance challenges

- Scalability...
- Global communication
- Per core performance still a concern
- Minimizing layers of parallelism to attain reasonable hybrid performance
- What can ‘exascale’ architectures give us scientifically?
- Do we really need exascale for climate research ?

Code challenges

Portability, programmability, software sustainability

- Unit testing: scientists need to do more, but education opportunities needed
- Adopting new programming paradigms, potential demise of explicit parallelism... what will the exascale programming model be?
- Are there alternatives to Fortran/C + MPI ??
- Insufficient training of code developers

Platform Challenges (hardware, OS, compiler, libraries)

- Reliability, fault-tolerance,
- Silent errors (error checking)
- Reproducibility if the same simulation is rerun (overlap with methodology). But how is it defined? Bit-4-bit for debugging?
- Ensure the same climate over different platforms (not only numerical issue). Error growth rate sufficient?

Ways to collaborate

- Need to make more effective use of hpc numerical libraries
- MetaFor dealt with meta data for climate forum (defunct) -- this should be continued within IS-ENES.
- Development of post- and pre-processing community (does CDO cover this?)
- Can we develop a CMIP-like protocol for machine verification?
- More involvement from standards' communities, e.g., FT-MPI within MPI-3 standardization
- Foster educational initiatives to incorporate best practices from software engineering principles. Long-term to develop new generation
- Develop climate kernels or mini-applications to provide computer scientists
- Creating, then advertising cooperation success-stories between computational and climate scientists
- Engagement with exascale community (EESI/IESP) -- documenting the needs of the ES community; need international cooperation programming paradigms