Case Studies: Using MPI and OpenACC in Applications

William Sawyer, Claudio Gheller, Markus Wetzstein,
Leonidas Linardakis (MPI-M), Günther Zängl (DWD)

CSCS-USI Summer School
8.07.2014 Serpiano, Switzerland

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

The RAMSES code: overview

- RAMSES (R.Teyssier, A&A, 385, 2002): code to study of astrophysical problems
- It treats at the same time various components (dark energy, dark matter, baryonic matter, photons)
- Includes a variety of physical processes (gravity, magnetohydrodynamics, chemical reactions, star formation, supernova and AGN feedback, etc.)

- Open Source
- Fortran 90
- Code “size”: about 70000 lines
- MPI parallel (public version)
- OpenMP support (restricted access)

- http://irfu.cea.fr/Phocea/Vie_des_labos/Ast/ast_sstechnique.php?id_ast=904

RAMSES workflow

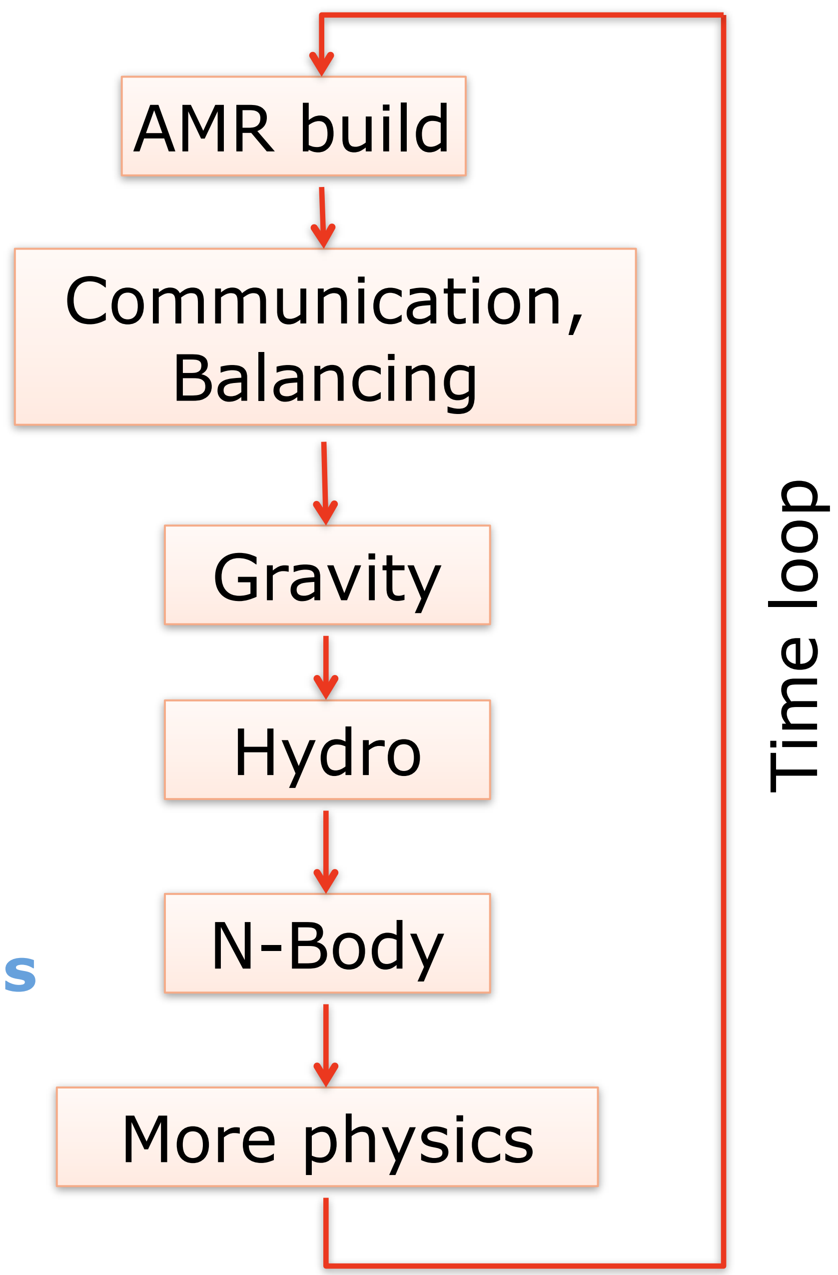
3D Eulerian Adaptive Mesh Refinement codes.

The code solves:

- **dark matter - N-body particle-mesh technique.**
- **gravity - multigrid technique.**
- **Hydrodynamics: various shock capturing methods.**
- **A number of additional physics processes**

Spatial discretization through and adaptive cartesian mesh

AMR provides high resolution ONLY where this is strictly necessary



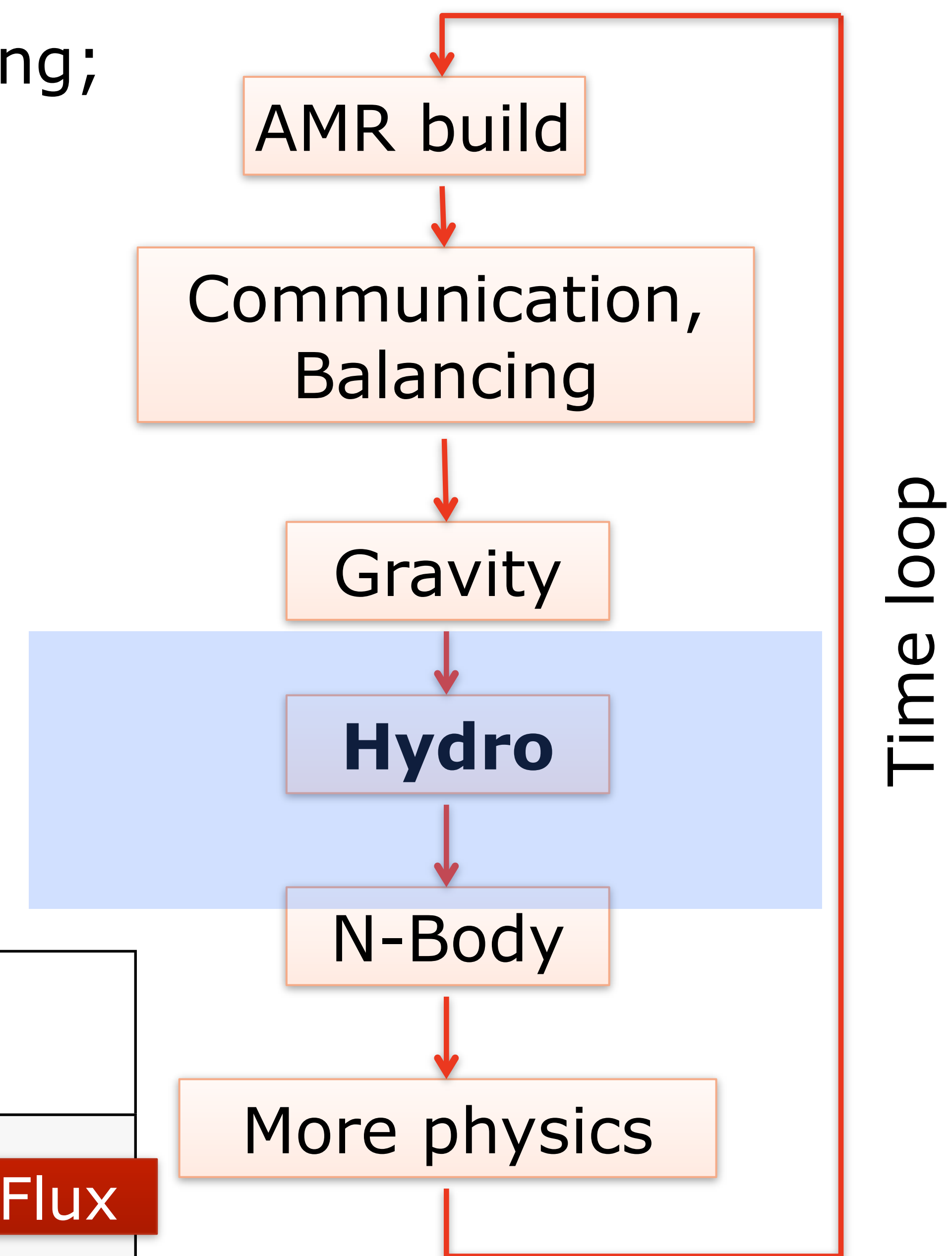
RAMSES: solving fluid dynamics

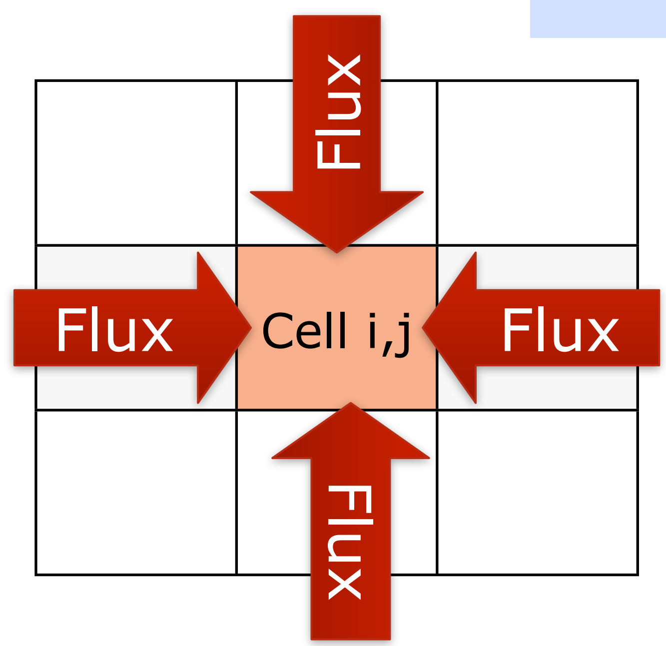
- Fluid dynamics is one of the key kernels;
- It is also among the most computational demanding;
- fluid dynamics is solved on a computational mesh solving three conservation equations: mass, momentum and energy:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho u) = 0$$

$$\frac{\partial}{\partial t}(\rho u) + \nabla \cdot (\rho u \otimes u) + \nabla p = -\rho \nabla \phi$$

$$\frac{\partial}{\partial t}(\rho e) + \nabla \cdot [\rho u\,(e + p/\rho)] = -\rho u \cdot \nabla \phi$$

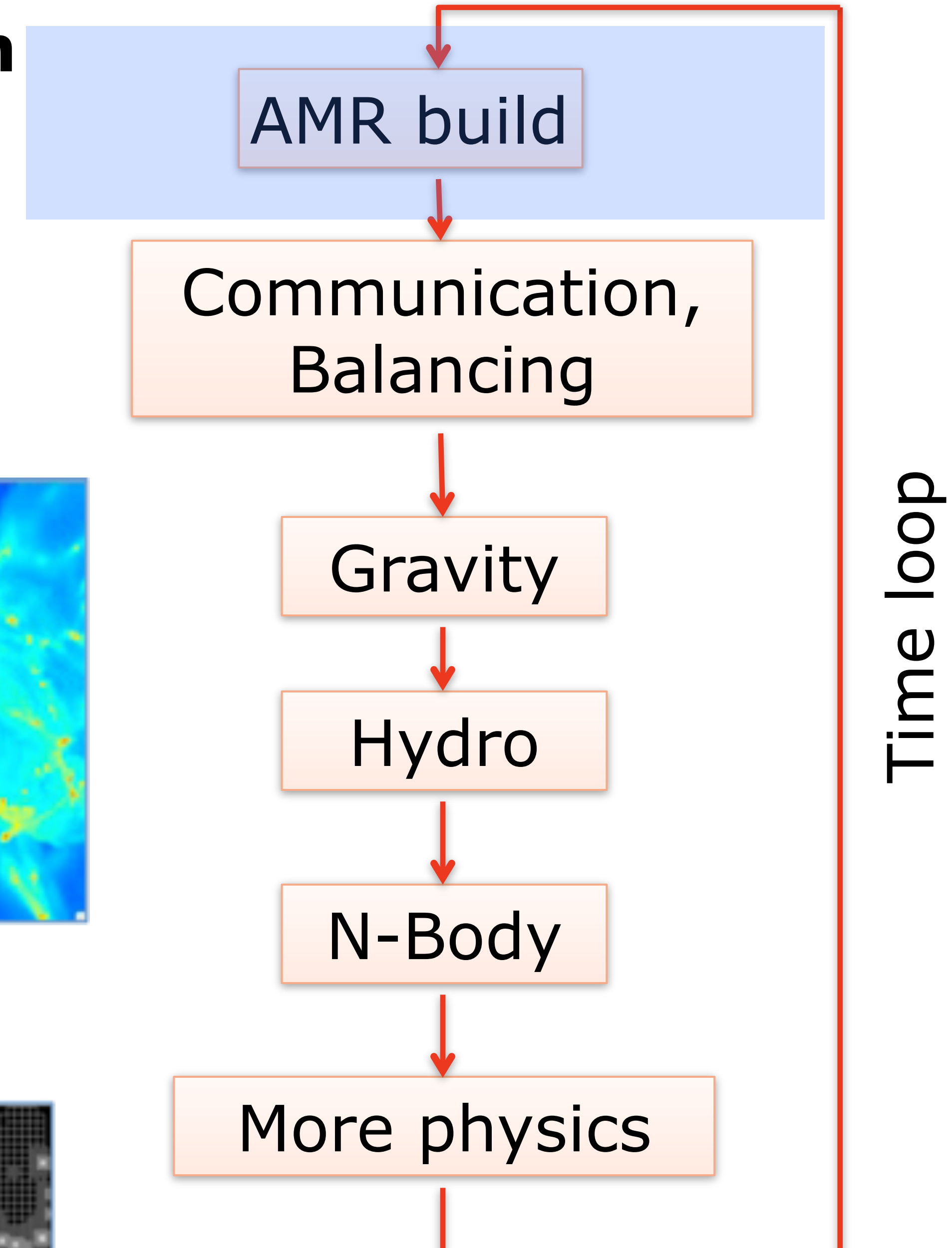


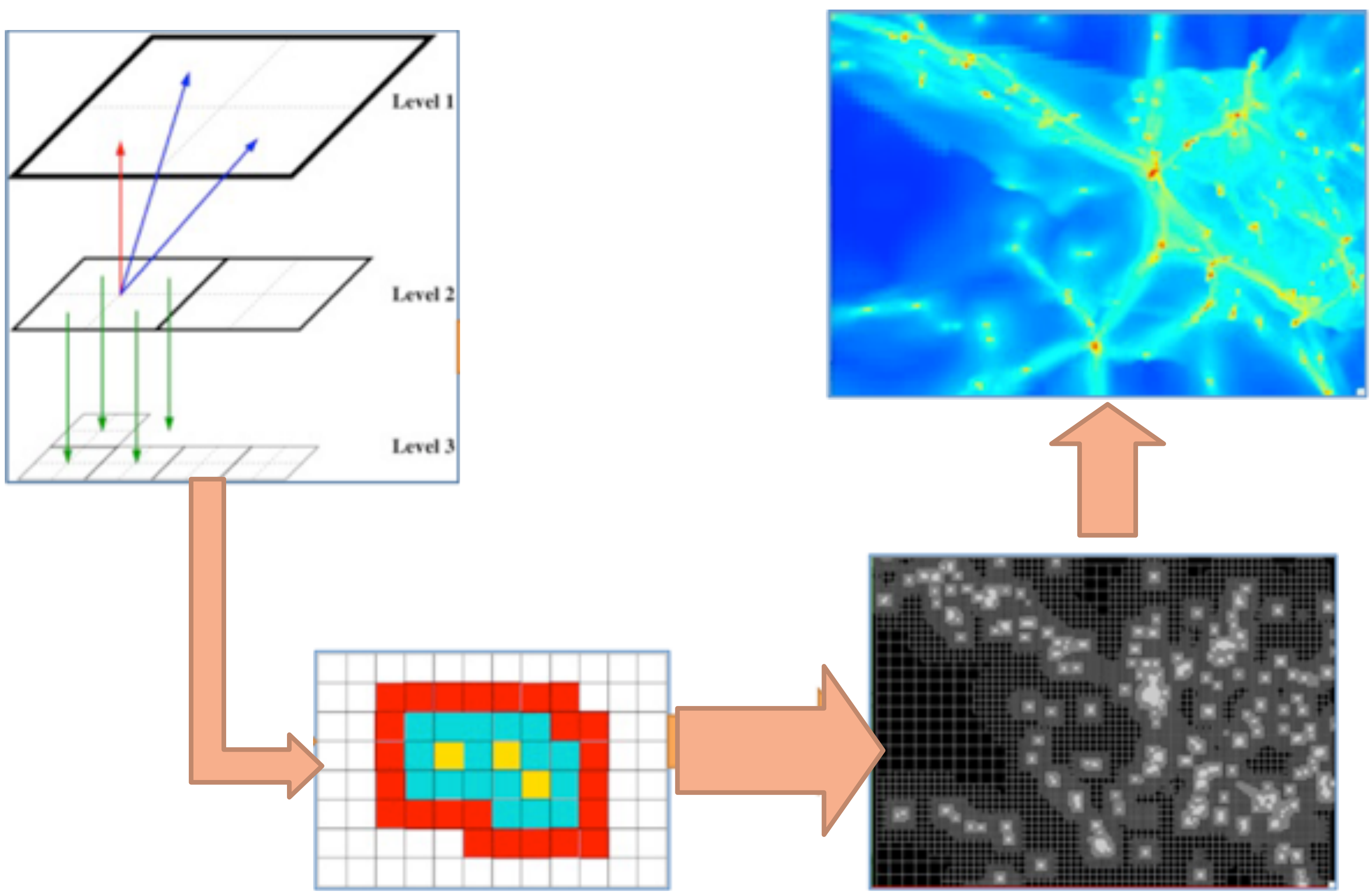


RAMSES AMR Mesh

Fully Threaded Tree with Cartesian mesh

- **CELL BY CELL refinement**
- **COMPLEX data structure**
- **IRREGULAR memory distribution**



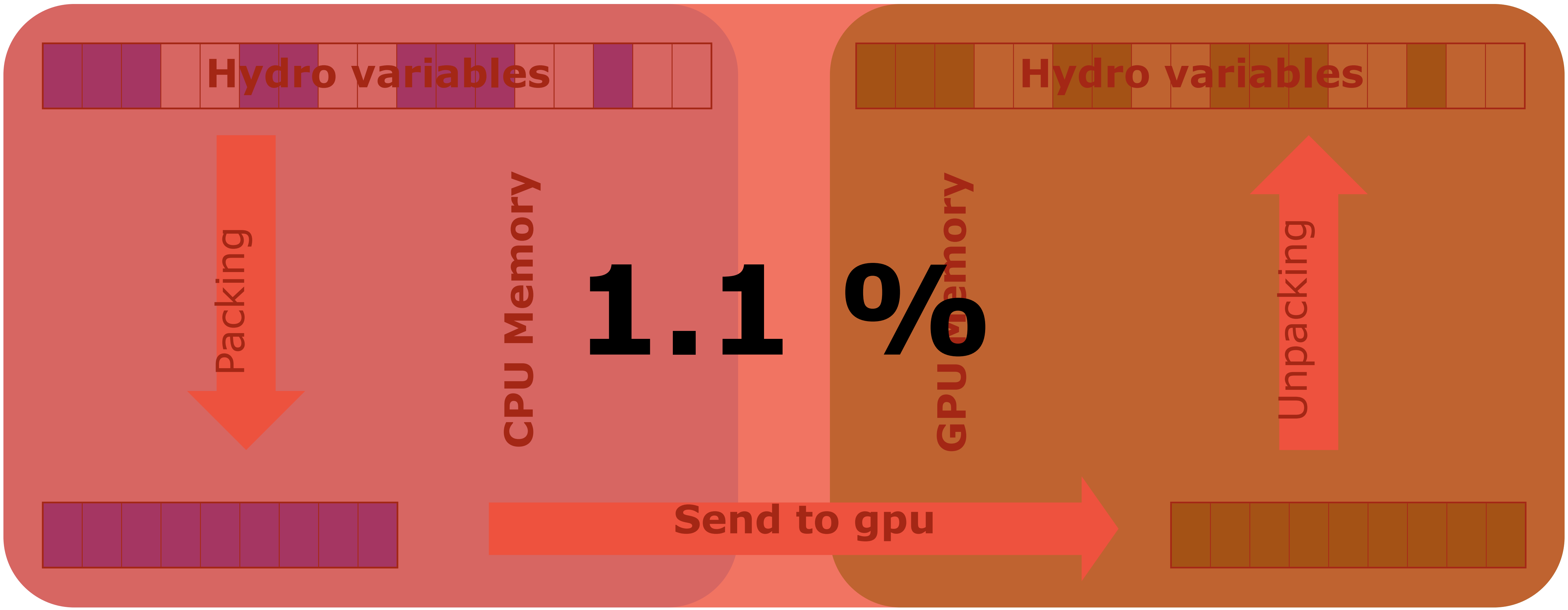


CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

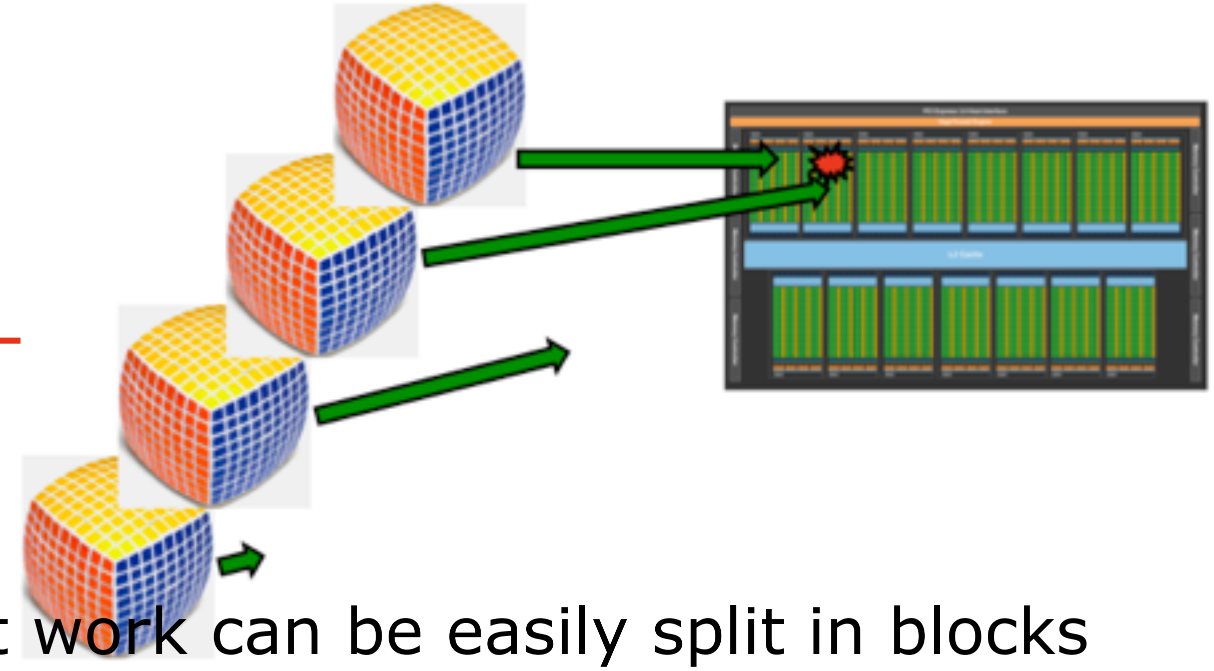
Moving data to/from the GPU

Send data to the gpu.

- **AMR grid's data is stored "random" in memory.**
- **Pack-unpack strategy level by level**
- **Has to be done every time step.**



CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre



On board the GPU…

1. Reorganization of memory in spatially contiguous patches, so that work can be easily split in blocks and coalescing memory can be exploited
2. Patches are grouped and pushed to the GPU cores. Groups size can be tuned in order to improve the occupancy
3. Patches build-up strongly benefits of the high memory bandwidth
4. Nested loops collapse used wherever possible
5. Gang and vector based work scheduling adopted (no particular benefit in using worker scheduling)
6. Offload data only when and where necessary (but this can be still improved – ongoing work)

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Performance analysis

Cosmological test with 3 levels of refinement
Levels 6 to 8

Cosmo 3 Levels (6-8)	T_tot	T_hydro		T_god_fine	T_copy	T_tot speedup	T_hydro speedup		T_god/ T_copy
		Sec	Percent				1 core vs 1gpu	1 cpu VS 1gpu	
orig_V10_N1	**155662**	**218**	**36.1 %**	56218					
orig_V10_N2	75905	27625	36.4	27625					
orig_V10_N4	36147	13207	36.5	13207					
orig_V10_N8	17	**6243**	**35.2 %**	6243					
orig_V10_N16	8775	2918	33.3	2918					
ACCyes_C1000_N1	**104811**	**009**	**2.9 %**	2270		**1.49**	**.68**	**2.07**	3.07
ACCyes_C1000_N2	49718	1425	2.9	1040	385	1.53	19.39	2.05	2.70
ACCyes_C1000_N4	23372	693	3.0	485	208	1.55	19.07		2.33
ACCyes_C1000_N8	11543	344	3.0	231	113	1.54	18.15		2.03
ACCyes_C1000_N16	5718	179	3.1	115	64	1.53	16.26		1.79

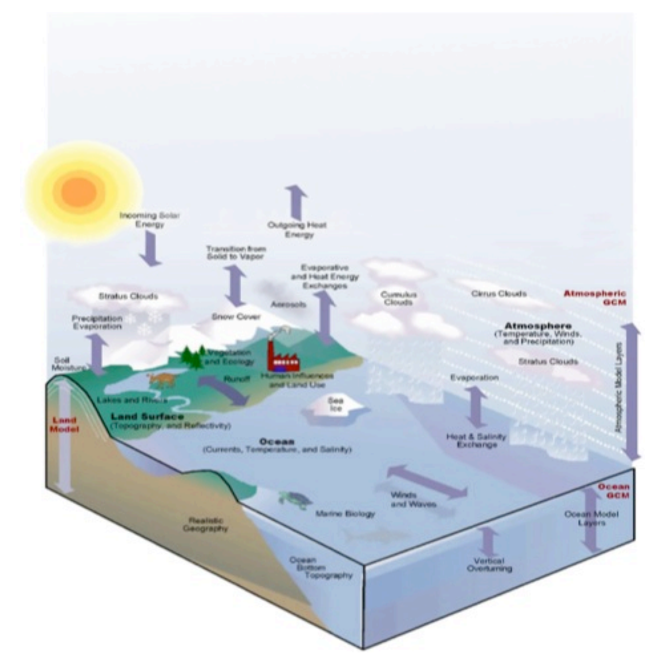
Performance results

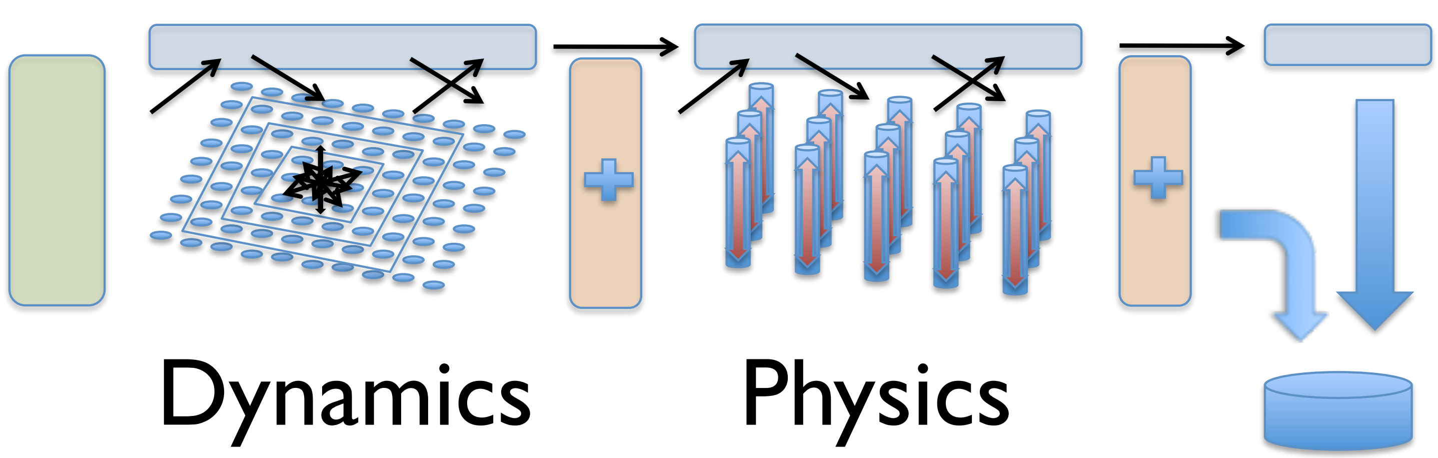
Hydro vars + AMR vars

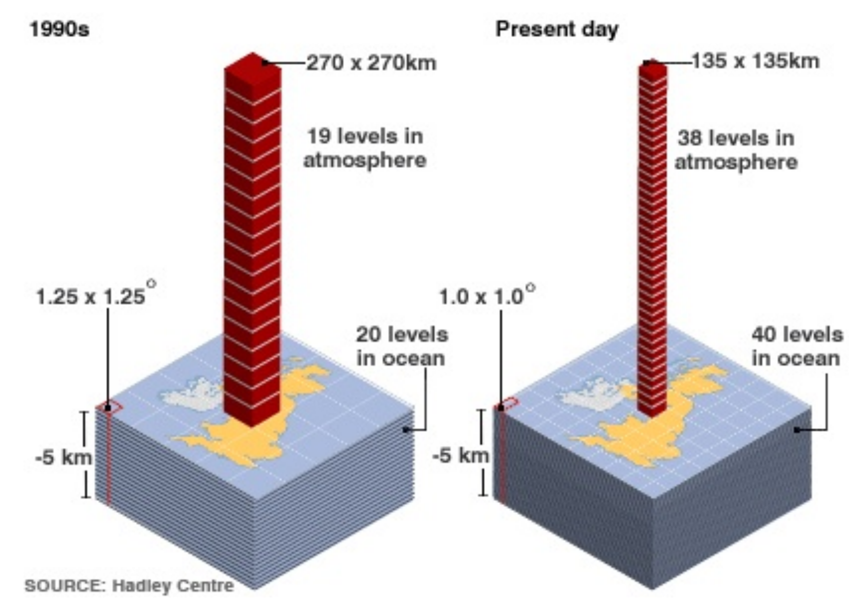
Cosmo 3 Levels (6-8)	T_tot	T_hydro		T_god_fine	T_copy	T_tot speedup	T_hydro speedup		T_god/ T_copy
		Sec	Percent				1core VS 1gpu	1 cpu VS 1gpu	
orig_V10_N1	155662	56218	36.1	56218					
orig_V10_N2	75905	27625	36.4	27625					
orig_V10_N4	36147	13207	36.5	13207					
orig_V10_N8	17755	6243	35.2	6243					
orig_V10_N16	8775	2918	3						
ACCyes_C1000_N1	104811	3009		2270	739	.49	18.68	2.0	3.07
ACCyes_C1000_N2	49718	1425		1040	385	.53	19.39	2.0	2.70
ACCyes_C1000_N4	23372	693		485	208	.55	19.07		2.33
ACCyes_C1000_N8	11543	344		231	113	.54	18.15		2.03
ACCyes_C1000_N16	5718	179		115	64	.53	16.26		1.79

Atmospheric General Circulation Model (AGCM)

- Earth is a giant heat engine
- Movement of the atmosphere ("Dynamics")
- Parameterization of sub-grid phenomena ("Physics")
- Dynamics and Physics calculate "tendencies" which alter "state"

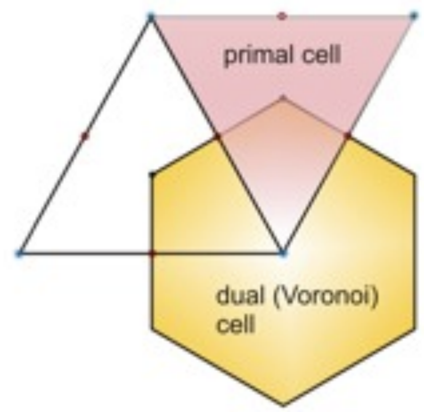






ICON NWP/Climate Model Overview

- ICOsahedral Non-hydrostatic model
- Dynamical core: conservation laws
- Triangular cells
- Nested grid
- Dwarf class: *unstructured grids*
- Extensive use of indexing arrays
- Memory bandwidth limited



PRACE 2IP Work Package 8: GPU-capable ICON

Goal: Implement a GPU-capable version of the ICON Non-hydrostatic dynamical core (NHDC) currently under development at the Max Planck Institute for Meteorology (MPI-M) and German Weather Service (DWD)

- *Completed:* OpenCL single-node NHCD implementation
- *Completed:* CUDAFortran single-node NHDC implementation
- *Presented:* results of single-node versions *(e.g. Boulder, Sep. 2011)*
- *Completed:* Refactored multi-node NHDC (based on r0211 DSL testbed Jun. 2012) in preparation for subsequent GPU implementation
- *Completed:* GPU-capable multi-node NHDC using MPI + OpenACC directives within the ICON domain-specific language (DSL) testbed
- *Planned:* implementation in main development trunk

First experiences (2011) Single-node prototype NHDC

- Graduate student implemented and validated OpenCL/C++ version of NHDC, about 60 kernels, in 6 weeks
- I implemented/validated NHDC in CUDAFortran (fewer, but more complex, kernels) in roughly 8 weeks
- Performance (CUDAFortran): K20x ~30% faster than 2xSandybridge
- Optimizations to both versions still possible
- CUDA/OpenCL programming not that difficult, but highly error-prone; debugging options limited; code validation crucial
- CUDAFortran is more ‘appealing’ to developers; OpenCL is the more portable paradigm *(but OpenCL 1.2/2.0 not supported by NVIDIA!)*
- Feedback from ICON developers: OpenCL and CUDAFortran *not viable for production version*
- Only valid option for multi-node version: *OpenACC ‘standard’*

ICON Data Structures

```
!  STATE VECTORS AND LISTS
  TYPE t_nh_state
    !array of prognostic states at different timelevels
    TYPE(t_nh_prog),  ALLOCATABLE :: prog(:)        !< shape: (timelevels)
    TYPE(t_var_list), ALLOCATABLE :: prog_list(:)   !< shape: (timelevels)
       :
    TYPE(t_nh_diag)    :: diag
    TYPE(t_var_list), ALLOCATABLE :: tracer_list(:) !< shape: (timelevels)
  END TYPE t_nh_state

! prognostic variables state vector
  TYPE t_nh_prog
    REAL(wp), POINTER :: &
      w(:,:,:),          & !> orthogonal vertical wind (nproma,nlevp1,nblks_c)     [m/s]
      vn(:,:,:),         & !! orthogonal normal wind (nproma,nlev,nblks_e)         [m/s]
      rho(:,:,:),        & !! density (nproma,nlev,nblks_c)                      [kg/m^3]
       :                   !! Several others
    TYPE(t_ptr_2d3d),ALLOCATABLE :: tracer_ptr(:)  !< pointer array: one pointer each tracer
  END TYPE t_nh_prog

  TYPE(t_nh_state), TARGET, ALLOCATABLE :: p_nh_state(:)

  ALLOCATE (p_nh_state(n_dom), stat=ist)
      :
  CALL construct_nh_state(p_patch(1:), p_nh_state, n_timelevels=2, l_pres_msl=l_pres_msl)
```

ICON NHDC Example: mean normal, tangent winds

```
!ICON_OMP_DO_STD PRIVATE(jb,i_startidx,i_endidx,jk,je, iqidx_1,iqblk_1,...)
    DO jb = i_startblk, i_endblk
!ICON_OMP_TASK_STD PRIVATE(i_startidx,i_endidx,jk,je, iqidx_1, iqblk_1,...) firstprivate(jb)
      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
        iqidx_1 = iqidx(je,jb,1)
        :
        DO jk = 1, nlev
          ! Average normal wind components
          ptr_vn(je,jk,jb) = p_int%e_flx_avg(je,1,jb)*p_nh%prog(nnew)%vn(je,jk,jb)&
            + p_int%e_flx_avg(je,2,jb)*p_nh%prog(nnew)%vn(iqidx_1,jk,iqblk_1) &
            :
          ! RBF reconstruction of tangential wind component
          p_nh%diag%vt(je,jk,jb) = p_int%rbf_vec_coeff_e(1,je,jb)   &
            * p_nh%prog(nnew)%vn(iqidx_1,jk,iqblk_1) &
            :
         ENDDO
        ENDDO
!ICON_OMP_END_TASK
    ENDDO
!ICON_OMP_END_DO
!ICON_OMP_WAIT_TASKS
```

ICON DSL primitives   Private indices   First/last block correction

Block number   Block size (usually 4 or 8)   Derived types

Testbed implementation: $ACC copies outside time

```
    iqidx_d                     = p_patch(1)%edges%quad_idx
    iqblk_d                     = p_patch(1)%edges%quad_blk
    e_flx_avg_d                 = p_int_state(1)%e_flx_avg
    prog_vn_now_d               = p_nh_state(1)%prog(nnow(1))%vn
    rbf_vec_coeff_e_d           = p_int_state(1)%rbf_vec_coeff_e
      :
!$ACC DATA COPY(iqidx_d,iqblk_d, ..., e_flx_avg_d, prog_vn_now_d, rbf_vec_coeff_e_d, ...

  TIME_LOOP: DO jstep = 1, nsteps
    ! dynamics stepping
    CALL integrate_nh(p_nh_state, p_patch, p_int_state, datetime, ... )
  ENDDO TIME_LOOP

!$ACC END DATA
```

Kernel invocation (inside non-hydrostatic solver):

```
    rl_start = 3
    rl_end = min_rledge_int - 2
    i_startblk = p_patch%edges%start_blk(rl_start,1)
    i_endblk   = p_patch%edges%end_blk(rl_end,i_nchdom)
    e_startidx = GET_STARTIDX_E(rl_start,1)
    e_endidx   = GET_ENDIDX_E(rl_end, MAX(1,p_patch%n_childdom))

#include "vn_and_vt_alt.inc"
```

ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

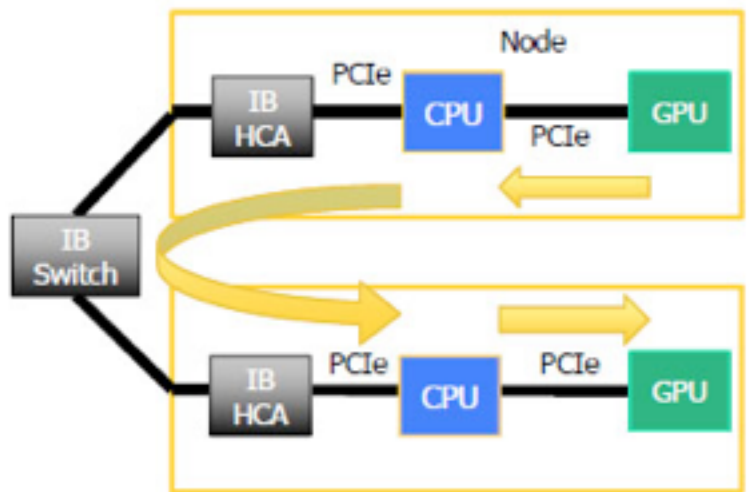
CSCS Swiss National Supercomputing Centre

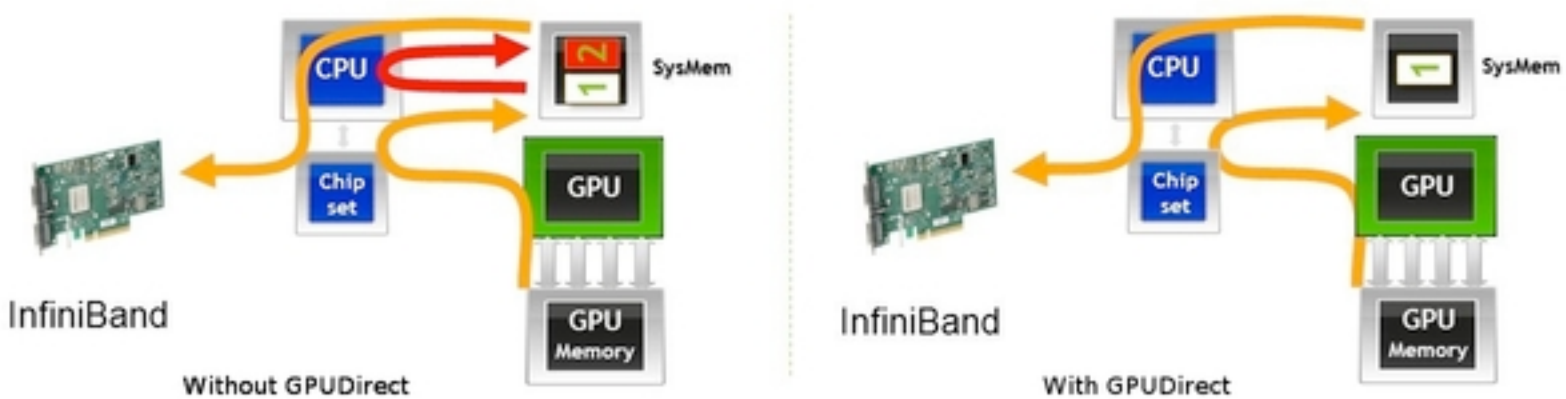
ICON DSL OpenACC Implementation

```
!$ACC PARALLEL &
!$ACC PRESENT( iqidx_d, ..., ptr_vn_d, e_flx_avg_d, vn_d, vt_d, rbf_vec_coeff_e_d )
!$ACC LOOP GANG PRIVATE( i_startidx, i_endidx, jb )
    DO jb = i_startblk, i_endblk
      IF ( i_startblk == jb ) THEN; i_startidx = e_startidx; ELSE; i_startidx = 1; ENDIF
      IF ( i_endblk == jb ) THEN; i_endidx = e_endidx; ELSE; i_endidx = nproma; ENDIF
!$ACC LOOP VECTOR
!DIR$ loop_info max_trips(MAX_NPROMA)
      DO je = i_startidx, i_endidx
        iqidx_1 = iqidx_d(je,jb,1)
        :
        DO jk = 1, nlev
          ! Average normal wind components
          ptr_vn_d(je,jk,jb) = e_flx_avg_d(je,1,jb)*vn_now_d(je,jk,jb)&
            + e_flx_avg_d(je,2,jb)*vn_now_d(iqidx_1,jk,iqblk_1) &
            :
          ! RBF reconstruction of tangential wind component
          vt_now_d(je,jk,jb) = rbf_vec_coeff_e_d(1,je,jb)  &
            * vn_now_d(iqidx_1,jk,iqblk_1) &
            :
         ENDDO
      ENDDO
    ENDDO
!$ACC END PARALLEL
```

Block size (usually 128-512)

GPU implementation of communication





ORIGINAL:

```
    DO i = 1, p_pat%n_send
      send_buf(1:ndim2,i) = send_ptr(p_pat%send_src_idx(i),1:ndim2,   &
        &                              p_pat%send_src_blk(i)-lbound3+1)
    ENDDO
```

ACCELERATED:

```
!$ACC DATA CREATE( send_buf, recv_buf )
!$ACC PARALLEL &
!$ACC PRESENT ( p_pat%send_src_idx, p_pat%send_src_blk, sendrecv )
!$ACC LOOP
      DO i = 1, n_send
        send_buf(1:ndim2,i) = sendrecv(p_pat%send_src_idx(i),1:ndim2,   &
          &                            p_pat%send_src_blk(i))
      ENDDO
!$ACC END PARALLEL
!$ACC UPDATE HOST( send_buf )
```

CCE supported this (unofficially?) but PGI did not; a PGI-amenable version would have taken extra effort; we forged ahead with CCE only

Key Kernel: vertical wind implicit (tridiagonal) solve

```
! This loop is special since z_q_alt has to be defined privately to this gang
!$ACC LOOP VECTOR PRIVATE( z_q_alt )
!DIR$ loop_info max_trips(MAX_NPROMA)
      DO jc = i_startidx, i_endidx
        z_q_alt(nlev) = 0.0_wp   ! Since z_alpha(nlev+1) == 0.0, never used?
        z_gamma_k   = dtime*cpd*metrics_vwind_impl_wgt_d(jc,jb)* &
        &  diag_theta_v_ic_d(jc,2,jb)/metrics_ddqz_z_half_d(jc,2,jb)
           :       ! Calculate other scalars
! Solve tridiagonal matrix for w for upper level
        prog_w_new_d(jc,2,jb)= prog_w_new_d(jc,2,jb)/z_b_scalar

        z_q_alt(2) = z_gamma_k * z_beta_k * z_alpha_kp1 / z_b_scalar
!$ACC LOOP SEQ
        DO jk = 3, nlev-1
          z_gamma_k   = dtime*cpd*metrics_vwind_impl_wgt_d(jc,jb)* &
          &  diag_theta_v_ic_d(jc,jk,jb)/metrics_ddqz_z_half_d(jc,jk,jb)
             :       ! Calculate other scalars
! Solve tridiagonal matrix for w
          prog_w_new_d(jc,jk,jb) = (prog_w_new_d(jc,jk,jb)  &
            -z_a_scalar*prog_w_new_d(jc,jk-1,jb))*z_g_scalar

! Define z_q_alt for next level
          z_q_alt(jk) = z_gamma_k * z_beta_k * z_alpha_kp1 * z_g_scalar
        ENDDO
```

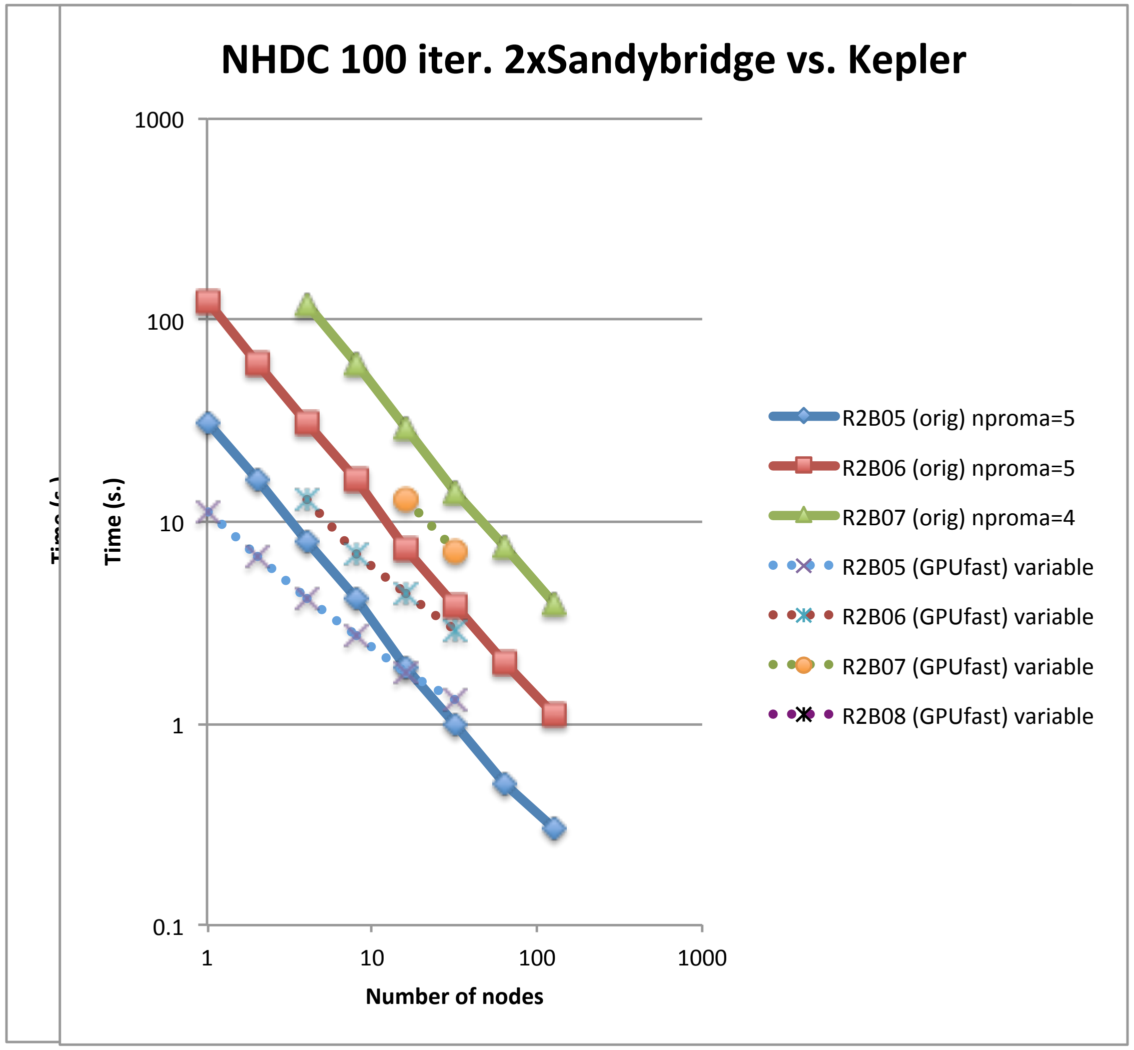
MPI+OpenACC: How long did it take?

Creation of shadow arrays for all fields	15 days
Moving data region outside main time loop	10 days
Validation infrastructure (needed for debugging)	25 days
Merging in latest software releases	10 days
Insertion of directives (NHDC solver)	2 days
Insertion of directives (Communication)	3 days
Tweaking of directives, compiler workarounds	5 days
Optimization of directives for best performance (many thanks to Cray's Vince Graziano)	10 days

➔ *Perhaps a full code rewrite is not prohibitive*

MPI+OpenACC first results: Sandybridge node vs. K20x

- Compare original (GNU) on Cray XC30 (2x Sandybridge sockets) vs. XK7 node with Kepler K20x (Cray CCE)



- *Fair comparison*
- OpenACC faster for cases where memory is fully exploited
- Weak scaling comparable, CPU strong scaling better
- OpenACC version can be further optimized (compare to single-node prototypes)

- *After optimizations: MPI+OpenACC factor 2x for cases of interest*

OpenACC Coding Objective: mean normal, tangent winds

```
!$ACC PARALLEL &
!$ACC PRESENT( iqidx, ..., ptr_vn, p_int%e_flx_avg, p_nh%prog(nnew)%vn, &
!$ACC          p_nh%diag%vt, p_int%rbf_vec_coeff_e )
!$ACC IF( i_am_compute_node )
!$ACC LOOP GANG PRIVATE( i_startidx, i_endidx, jb )
    DO jb = i_startblk, i_endblk
      CALL get_indices_e(p_patch, jb, i_startblk, i_endblk, &
                         i_startidx, i_endidx, rl_start, rl_end)
!$ACC LOOP VECTOR
      DO je = i_startidx, i_endidx
        iqidx_1 = iqidx(je,jb,1); iqblk_1 = ... ; iqidx_2 = ...;   ! etc.
        DO jk = 1, nlev
          ! Average normal wind components
          ptr_vn(je,jk,jb) = p_int%e_flx_avg(je,1,jb)*p_nh%prog(nnew)%vn(je,jk,jb)&
            + p_int%e_flx_avg(je,2,jb)*p_nh%prog(nnew)%vn(iqidx_1,jk,iqblk_1) &
            :
          ! RBF reconstruction of tangential wind component
          p_nh%diag%vt(je,jk,jb) = p_int%rbf_vec_coeff_e(1,je,jb)  &
            * p_nh%prog(nnew)%vn(iqidx_1,jk,iqblk_1) &
            :
         ENDDO
        ENDDO
    ENDDO
!$ACC END PARALLEL
```

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

OpenACC coding objective: full or selective deep copies ?

```
!$ACC DATA COPY( p_patch(1)%edges%vertex_blk,        p_patch(1)%edges%vertex_idx,        &
!$ACC            p_patch(1)%comm_pat_v%n_send,       p_patch(1)%comm_pat_v%n_pnts,       &
!$ACC            p_patch(1)%comm_pat_v%send_src_idx, p_patch(1)%comm_pat_v%send_src_blk, &
!$ACC            p_nh_state(1)%prog(nnow(1))%vn,     p_nh_state(1)%prog(nnew(1))%vn,     &
!$ACC            p_nh_state(1)%diag%vn_ie,           p_nh_state(1)%diag%vt,              &
                  :

  TIME_LOOP: DO jstep = 1, nsteps

    ! Lots of stuff we won't put on the GPU at this time
       :
    CALL integrate_nh(datetime, 1, jstep, dtime, dtime_adv, 1)
       :
  ENDDO TIME_LOOP
!$ACC END DATA
```

Selective deep copy (Cray CCE only; undocumented feature)

```
!$ACC DATA COPY( p_patch, p_nh_state, .... )
  TIME_LOOP: DO jstep = 1, nsteps

    ! Lots of stuff we won't put on the GPU at this time
       :
    CALL integrate_nh(datetime, 1, jstep, dtime, dtime_adv, 1)
       :
  ENDDO TIME_LOOP
!$ACC END DATA
```

Full deep copy (CCE documented feature; limitations with ICON pointers)

OpenACC validation strategy

```
#if defined( _OPENACC )
!$ACC DATA CREATE ( z_w_concorr_me,  z_w_concorr_mc,   z_w_con_c,       z_w_con_c_full, &
!$ACC               z_kin_hor_e,     z_ddxn_ekin_e, z_vt_ie ), &
!$ACC      PRESENT( p_diag%vt, p_diag%dvn_ie_ubc, p_prog%vn, p_diag%vn_ie, &
!$ACC               p_diag%e_kinh, p_diag%w_concorr_c )
#else
!$OMP PARALLEL PRIVATE(rl_start, rl_end, i_startblk, i_endblk)
#endif

    rl_start = 3
    rl_end = min_rledge_int - 2
    i_startblk = p_patch%edges%start_blk(rl_start,1)
    i_endblk   = p_patch%edges%end_blk(rl_end,i_nchdom)
    e_startidx  = p_patch%edges%start_idx(rl_start,1)
    e_endidx    = p_patch%edges%end_idx(rl_end,MAX(1,p_patch%n_childdom))

#include "vn_ie_and_vt_ie_and_kin_hor_e_and_w_concorr_me_ACC.inc"
#if defined( TEST_MODE && _OPENACC )
!$ACC UPDATE HOST ( p_diag%vn_ie, z_w_concorr_me ) &
!$ACC        IF( i_am_compute_node )

! Test e_kinh, w_concorr_me
    CALL sync_patch_array(SYNC_E,p_patch,p_diag%vn_ie, "vn_ie")
    CALL sync_patch_array(SYNC_E,p_patch,z_w_concorr_me, "z_w_concorr_me")
#endif
```

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

OpenACC: Experiences

- Tried both `$ACC KERNELS` and `$ACC PARALLEL`, settled on latter. Fine distinctions between the two are lost on application developers. *Why do we need both?*
- Played with `$ACC LOOP WORKER` but could not find any benefit
- Struggled with `$ACC CACHE` in critical vertical implicit solve; could not use it in this context
- Cray-specific directive was key optimization:

```
!DIR$ loop_info max_trips(MAX_NPROMA)
```

- *Did not utilize CCE's support for full deep copy*

OpenACC: Reflections

- OpenACC is the right idea: try to consolidate accelerator functionality into standardized directives
- OpenACC is not yet mature; significant functionality missing, vendors may interpret and implement standard differently, e.g. derived types
- Inserting directives may be quick, but refactoring and optimizing code for GPU are not; perhaps full rewrite is not so much more work...

Internships @ CSCS

- Dedicated essentially to Swiss/EU master students
- 2 to 6 months period that can be spent at CSCS working on a science-computational science topic
- Next round will open by the beginning of the summer
- Examples of past topics:
  - "Investigating the D Programming language in HPC"
  - "Refactoring and Optimization of the RAMSES codes on the GPUs"
  - "Generic Communication Library Development, testing and optimization"
  - "Analysis of data compression techniques to reduce climate model output"
- http://www.cscs.ch/about/working_at_cscs/internships/index.html (or on ETHZ and EPFL web sites)

Acknowledgments

- Funding: PRACE 2IP WP8, Grant RI-283493
- ICON team (MPI-M/DWD): collaborative effort
- CSCS Future Systems and User Support staff
- Technical support from:
  - Cray: Vince Graziano, others...
  - PGI: Michael Wolfe, Mat Colgrove, others...
  - NVIDIA: Peter Messmer, others...

- Thanks to you for listening!