Development and GPU porting efforts for a high-performance low-Mach CFD solver based on unstructured adaptive grids

V. Moureau, F. Gava, P. Bénard, G. Lartigue, K. Bioche, …
CORIA, CNRS UMR6614, Normandie Univ, UNIROUEN, INSA Rouen, France

P. Begou, G. Balarac, G. Ghigliotti,– LEGI, Grenoble, France

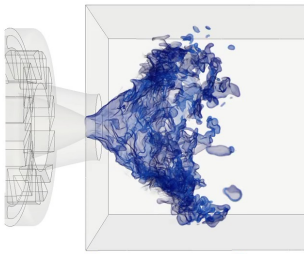
A. Froehly, C. Dobrzynski – LMB/INRIA Bordeaux, France

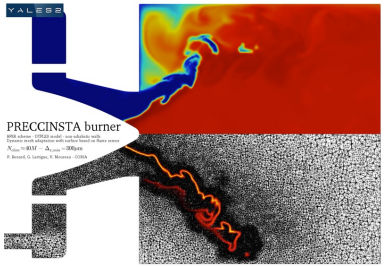
R. Mercier, M. Cailler, J. Leparoux – SAFRAN Tech, Magny-les-H., France

NUMPEX, 12 juin 2024

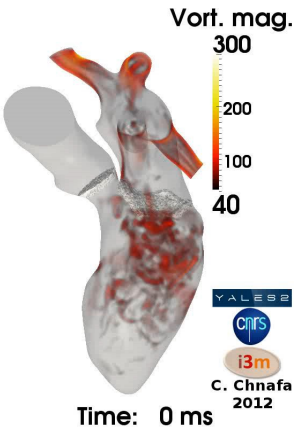
A multi-physics & High-Fidelity platform for complex flows

Gaseous combustion

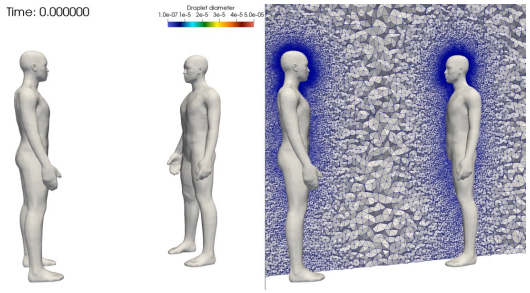




Bio-mechanics

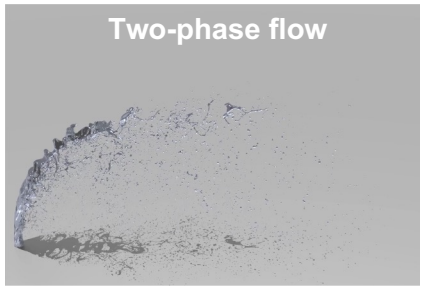


Aerosol dispersion

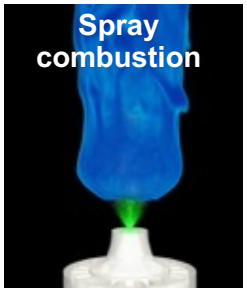


Sloshing in tanks

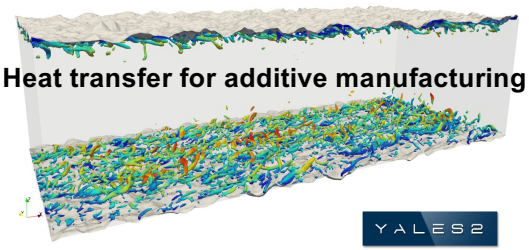
Two-phase flow



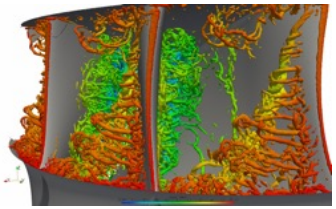
Spray combustion

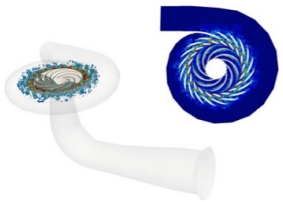


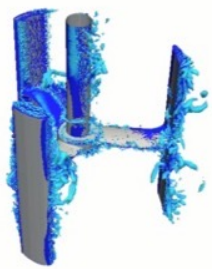
Heat transfer for additive manufacturing

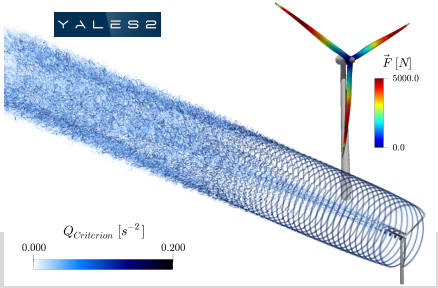


Renewable energies









YALES2

SAFRAN

arianeGROUP

GE Renewable Energy

INERIS

CORIA
UMR 6614

Air Liquide

The University of Vermont

SIEMENS Gamesa RENEWABLE ENERGY

CORNELL UNIVERSITY

TotalEnergies

SOLVAY
asking more from chemistry®

IFP Energies nouvelles

GDTECH engineering

IMFT

EM2C

IMAG

LEGI
LABORATOIRE DES ECOULEMENTS GEOPHYSIQUES ET INDUSTRIELS

ONERA
THE FRENCH AEROSPACE LAB

UMONS
Université de Mons

VUB
VRIJE UNIVERSITEIT BRUSSEL

TU Delft
Delft University of Technology

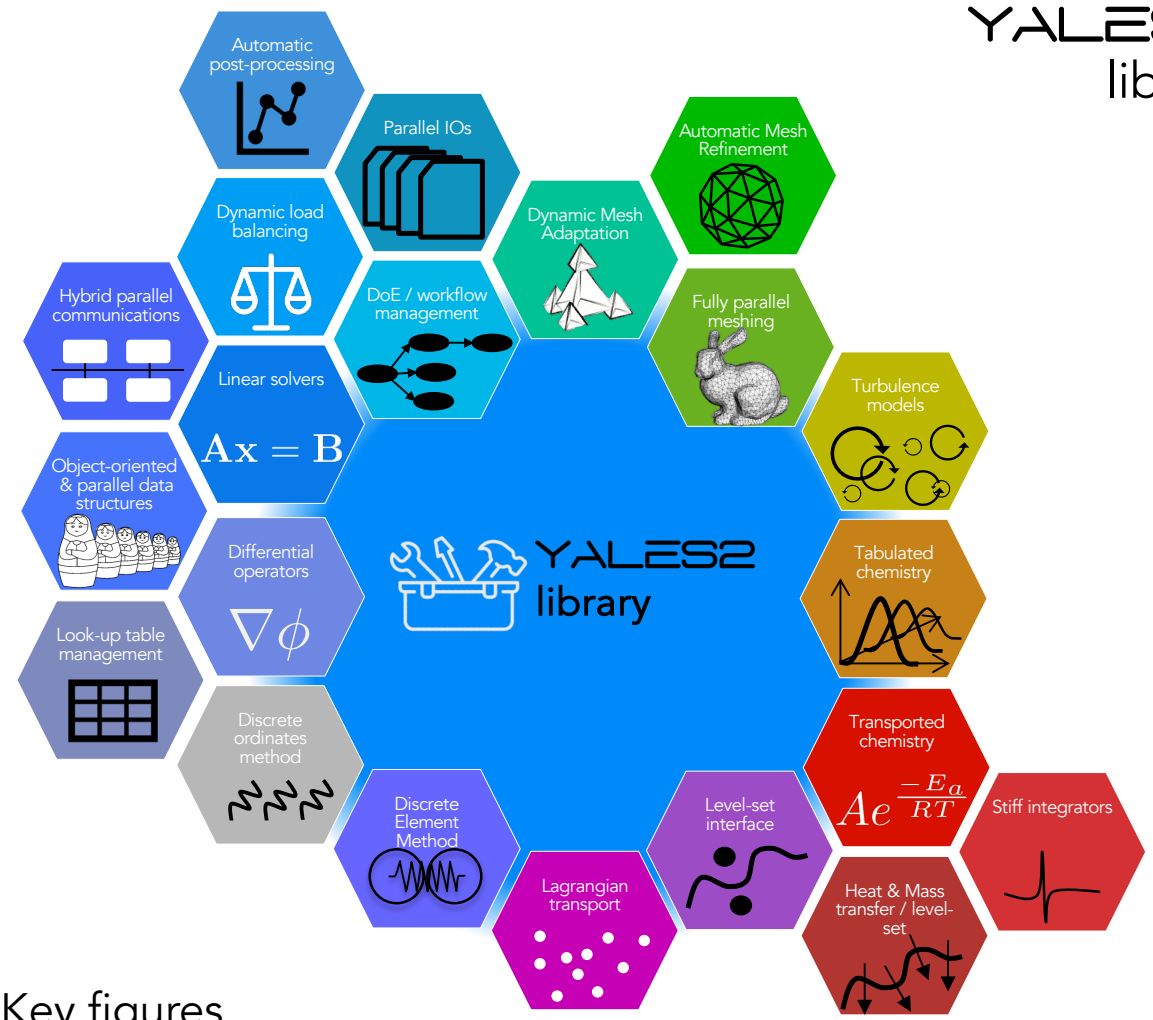
YPSO
Your Process Secured & Optimized

FACTO

- Developed by CORIA, the French Combustion Community and others
  - 600+ researchers/engineers trained at CORIA since 2009
  - 300+ articles (Google Scholar)

The YALES2 network

The CFD platform



YALES2 solvers

ICS — Incompressible at constant density

VDS — Incompressible at variable density

SPS — Spray with level-set and ghost-fluid method

ALE — Arbitrary Lagrangian Eulerian

GFS — Granular flow with Discrete Element Method

HTS — Heat transfer

MHD — Magneto-hydro-dyn

SMS+FS — Structural mech.

ACS — Acoustics

BOI — Boiling

CPS — Compressible flow

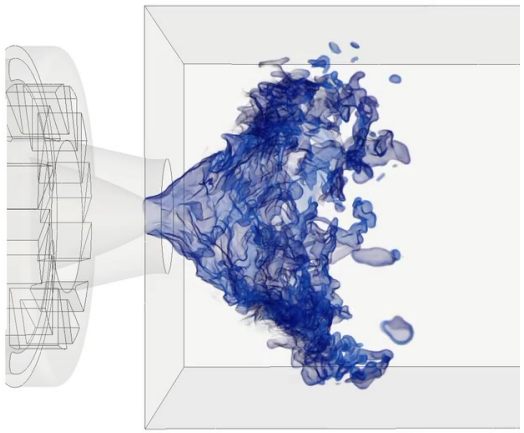
YALES2 features

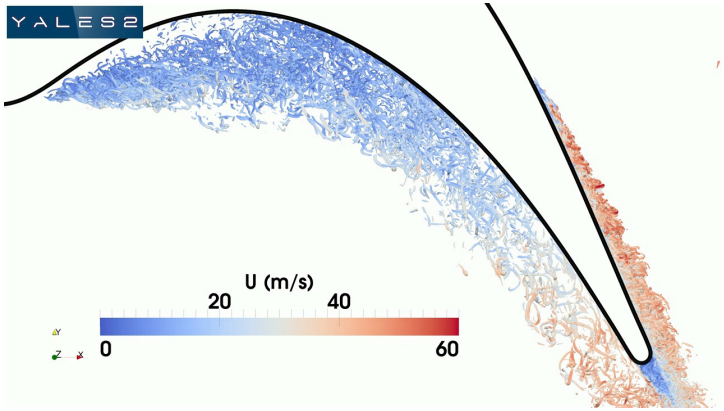
- High fidelity
- Multiphysics
- High performance

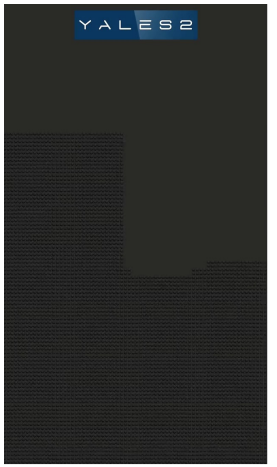
- Key figures
  - 17 major releases
  - 930 000 object-oriented F2008 lines
  - 19 300+ commits
  - 300+ active branches
  - 1 050+ merge requests
  - 120+ contributors

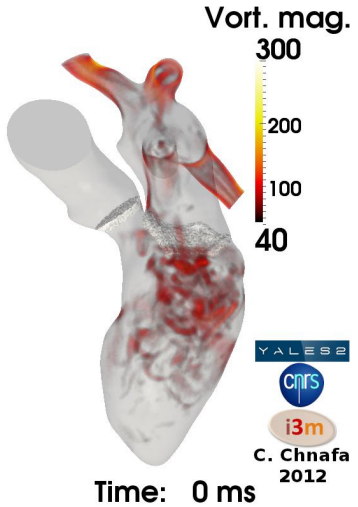
CFD platform: YALES2

- Key features
  - **Unstructured meshes** and **adaptive grid refinement**
  - Low-Mach number Navier-Stokes (incompressible and variable density)
  - Full-Mach number Navier-Stokes (explicit compressible with shock capturing method)
  - **4-level domain decomposition** [3] and hybrid OpenMP/MPI communications
  - Highly efficient solvers for linear system inversion (PCG, DPCG) [4]
  - **4th-order** central pair-based **finite-volume method**
  - **4th-order** time integration (hybrid Lax-Wendroff / Taylor scheme)
  - Numerical scheme hybridation
  - Two-phase flows (Lagrangian particles), **spray and atomization** (Levelset), flashing and cavitation (Multifluid)
  - Combustion modeling (Tabulated or **finite-rate chemistry**, NOx model, …)
  - Suited for massively parallel computing (>32 000 procs)



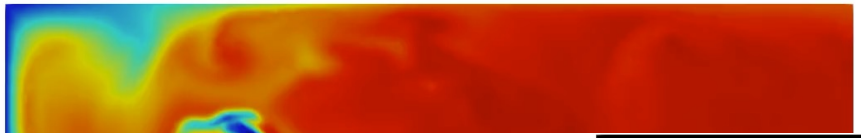


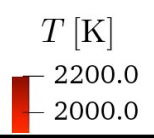


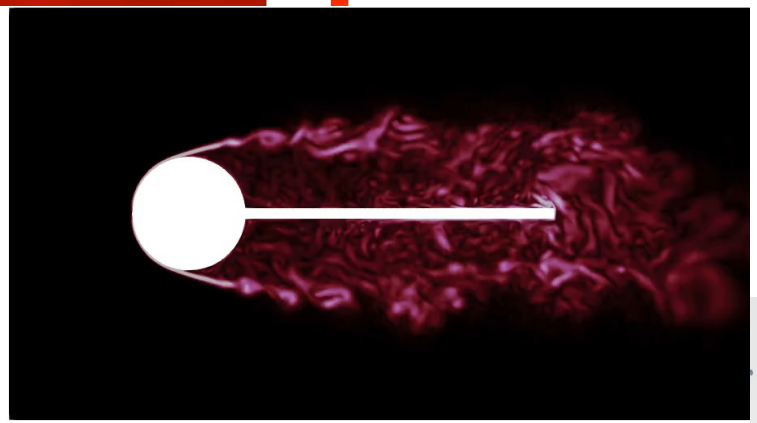


[1] http://www.coria-cfd.fr
[2] http://success.coria-cfd.fr
[3] Moureau et al., CR Mecanique, 2011
[4] Malandain et al., JCP, 2013

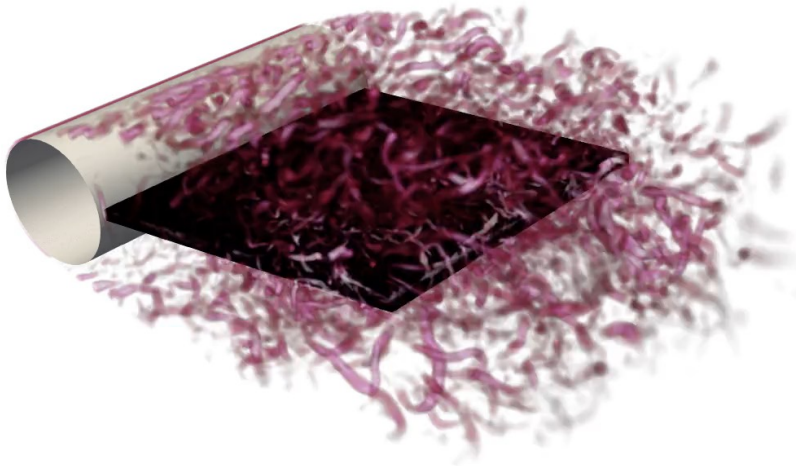
A major feature of the platform

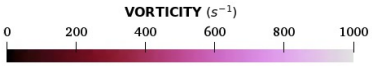


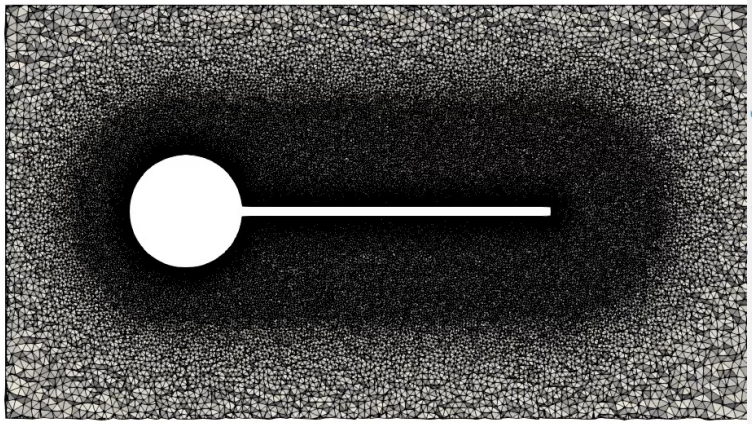


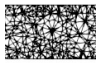


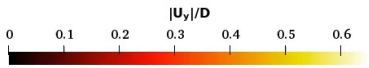
PRE
BFER scheme
Dynamic me
$N_{elem} \approx 4($
P. Benard, G









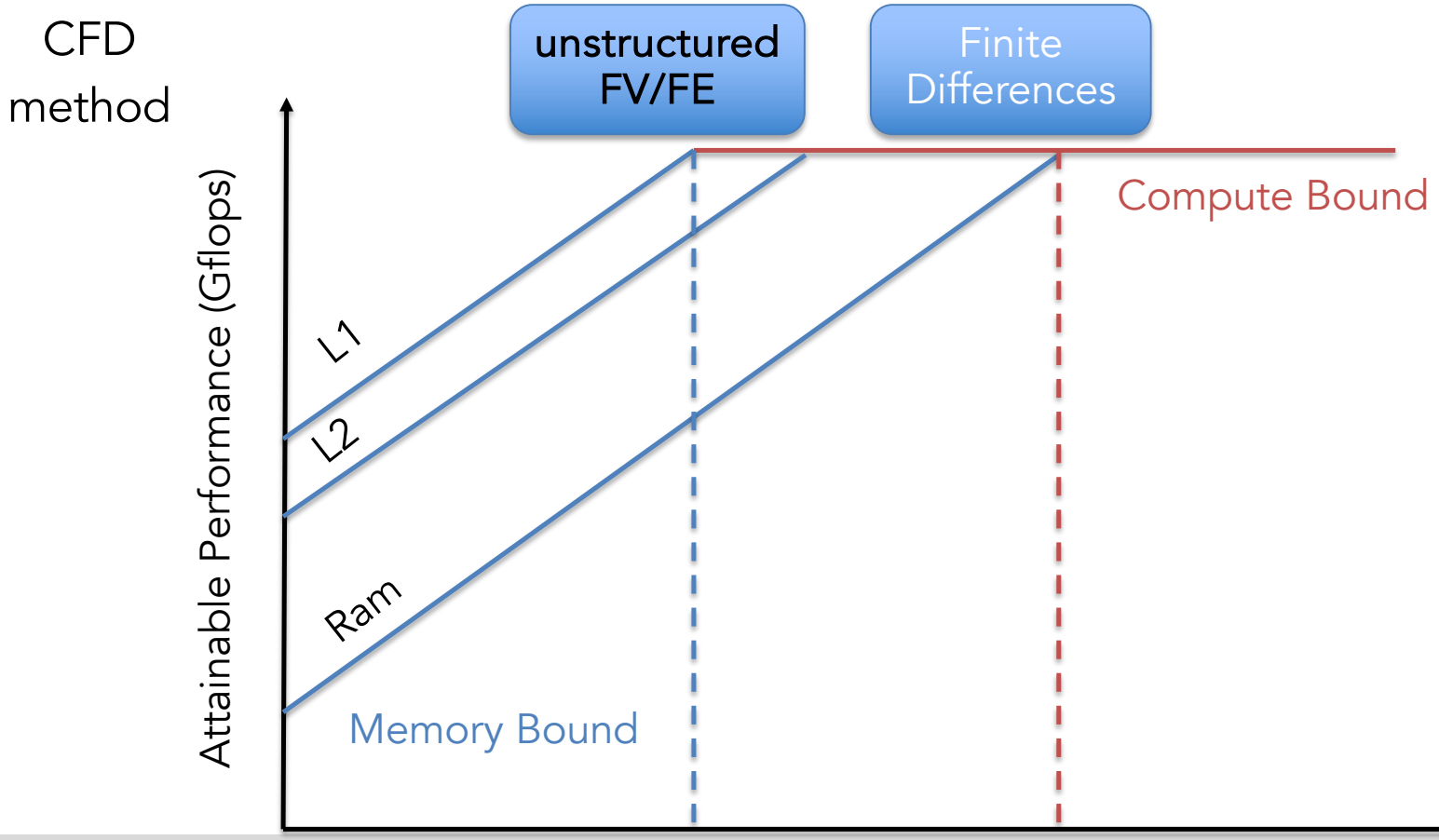


Courtesy J. Vernier, J. Leparoux, R. Mercier, SAFRAN TECH

Partitioning and cache-blocking

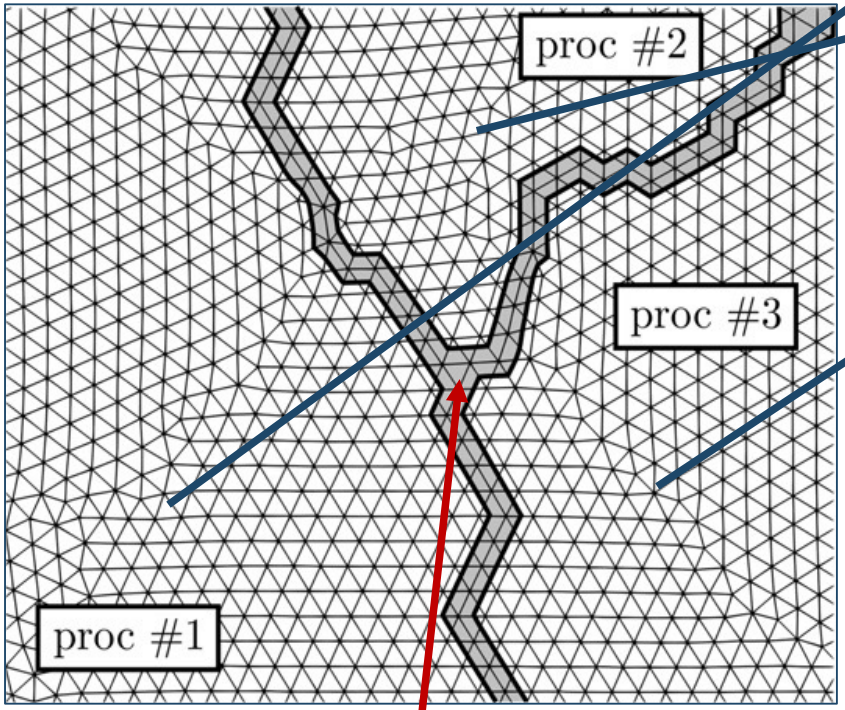
Importance of memory accesses in unstructured FV codes

- Code performance on a CPU can be limited by:
  - Processor speed (compute bound)
  - **Memory access speed** (memory bound)
  - The roofline model

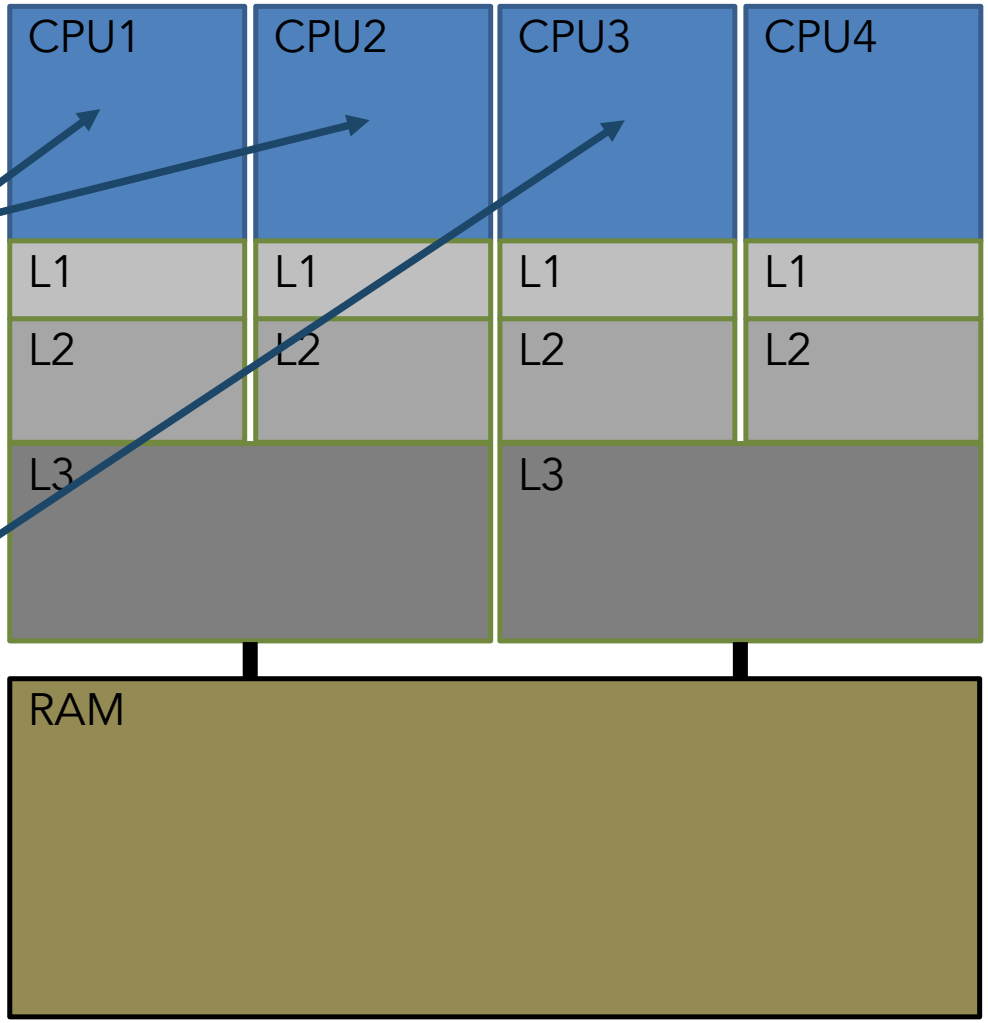


CPU: Parallel computation and domain decomposition

- Large problems can not be computed by a single process
- Domain decomposition to divide the problem amongst many processes
  - More memory available
  - More computational power
  - Communication needed



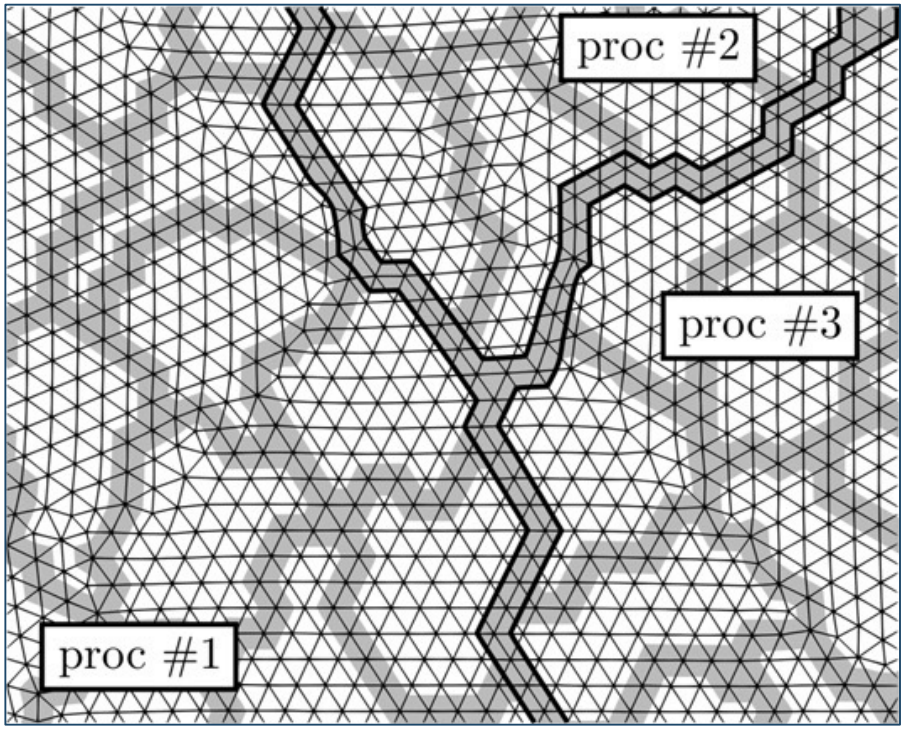
Data on these nodes have to be exchanged between processes



MPI/OpenMP + cache-blocking: 3-level domain decomposition

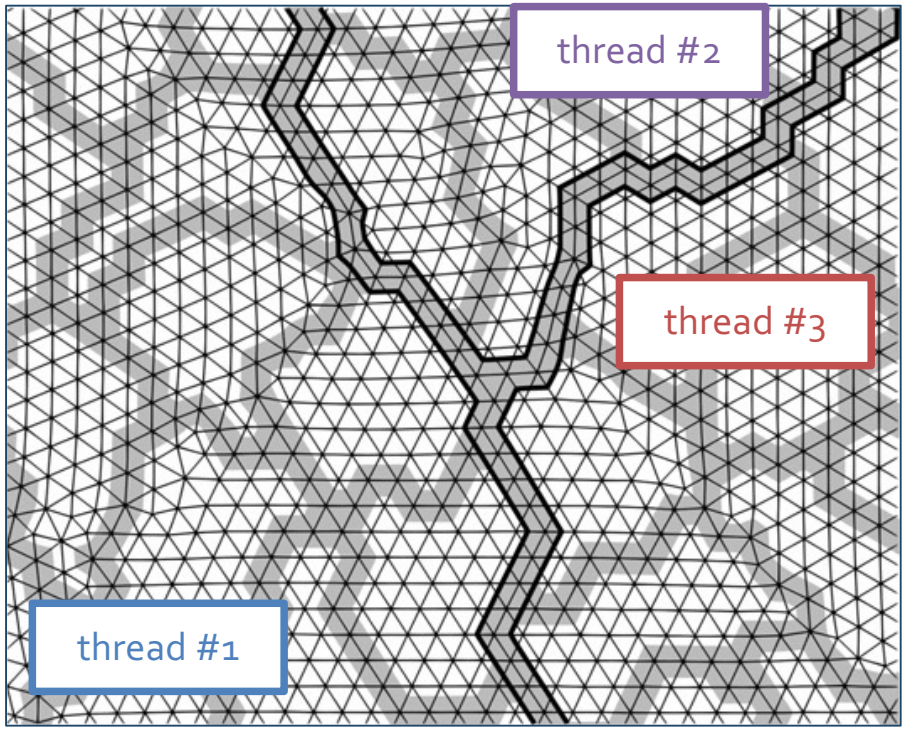
MPI + OpenMP + in-thread domain decomposition

Full MPI [1]



- Cell groups for cache-blocking
- Cell group size ~ 2000
- **Overhead: node duplication**

Coarse-grain Hybrid OpenMP + MPI

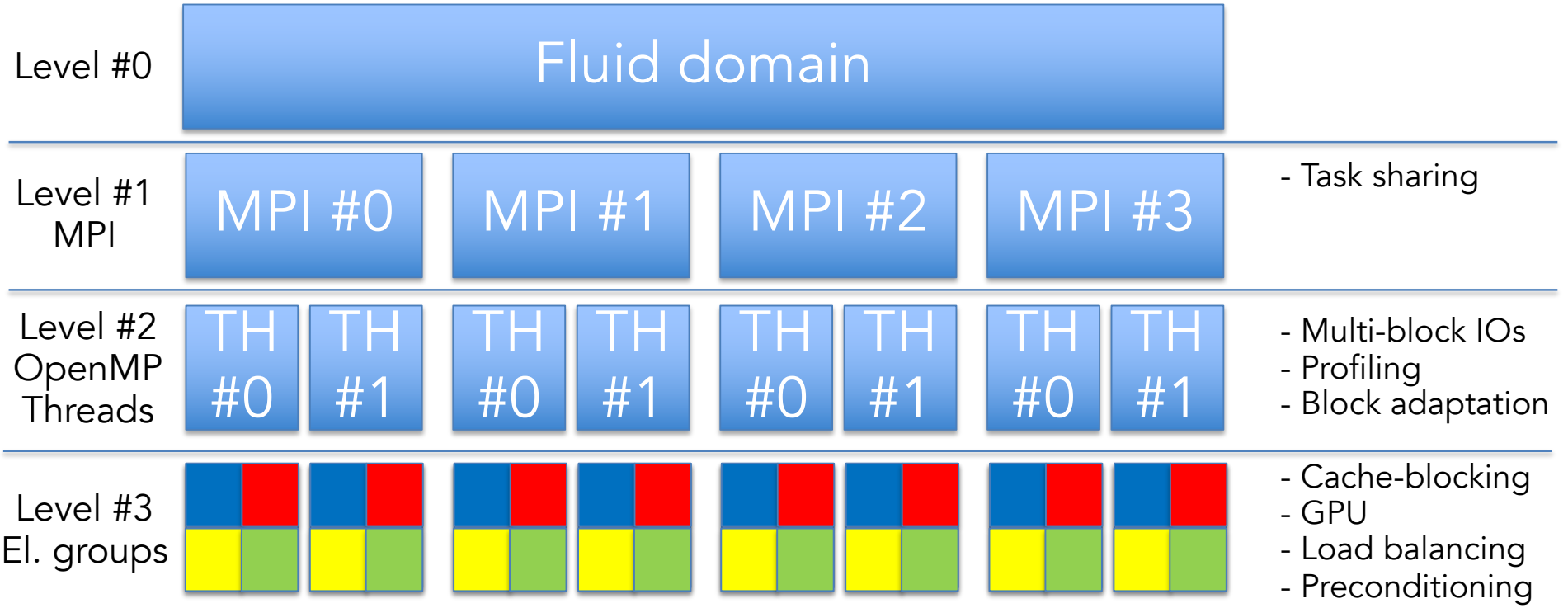


- Threads substitute MPI ranks
- Fewer MPI ranks but thread-safety
- Overhead:
  - Threads must communicate
  - Node duplication

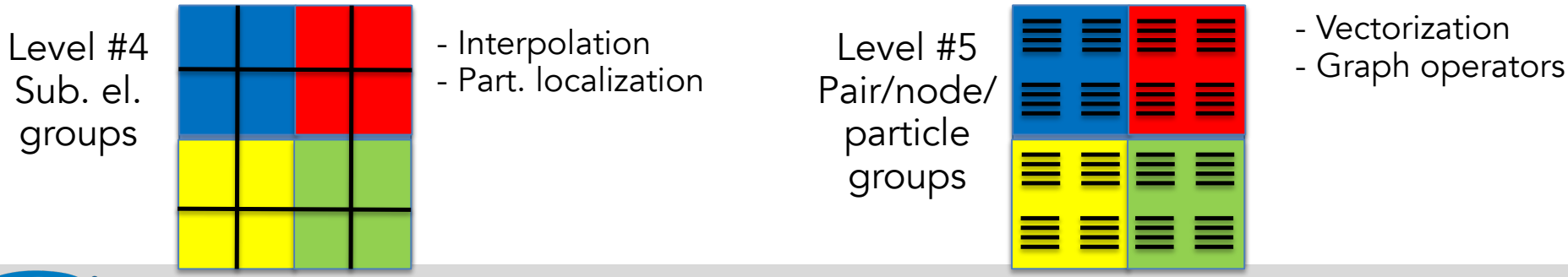
[1] Moureau et al., CR Mecanique, 2011

Hierarchical domain decomposition in YALES2

- Explicit partitioning



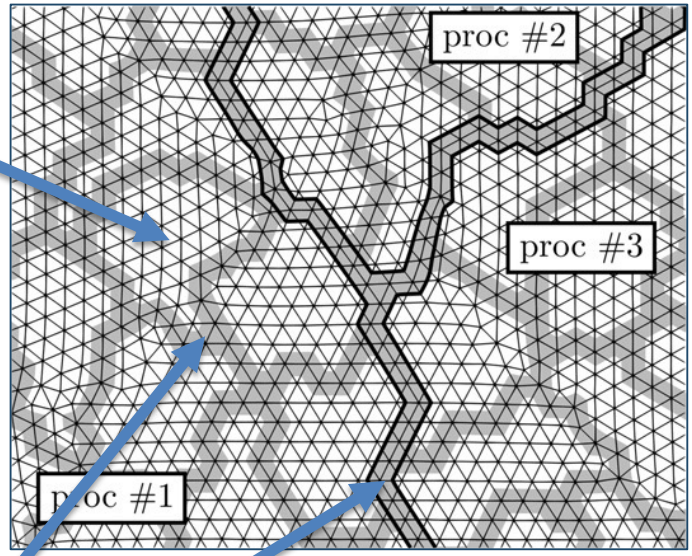
- Implicit partitioning



A typical loop on CPU

- Trying to hide all the synchronization code

```
!loop on the cell groups
do n=1,grid%nel_grps
   el_grp => grid%el_grps(n)%ptr

   residual_val => residual_ptr%r1_ptrs(n)%ptr%val
   phi_np1_val => phi_np1_ptr%r1_ptrs(n)%ptr%val
   udotAp_val => udotAp_ptr%r1_ptrs(n)%ptr%val
   el_grp_pair2node1_val => el_grp%pair2node1%val
   el_grp_pair2node2_val => el_grp%pair2node2%val

   !loop on the node pairs (edges)
   npair = el_grp%npair

   do ip=1,npair
      ino1 = el_grp_pair2node1_val(ip)
      ino2 = el_grp_pair2node2_val(ip)

      ...

      ! residual update
      residual_val(ino1) = residual_val(ino1) + coeff
      residual_val(ino2) = residual_val(ino2) - coeff
   end do

   call update_int_comm(n,residual_ptr,grid,COMM_ADD)
end do
call update_grid_data(residual_ptr,grid,COMM_ADD)
```



GPUs & APUs

- Working on OpenACC port since 2017
  - Benefits from array objects (r1_t, r2_t, …, i1_t, i2_t, …)
  - GPU memory management in the data structures

```
!=============================================================
!# <STRUCTURE NAME=r2_t>
!# <DESCRIPTION>Pointer for two-dimension real variables.</DESCRIPTION>
type r2_t
  real(WP), allocatable :: val(:,:) !# <VAR NAME=val TYPE=real(:,:)>Values</VAR>
  character(len = LEN_MAX) :: name  !# <VAR NAME=name TYPE=character>Name of the pointer</VAR>
  integer :: name_len               !# <VAR NAME=name_len TYPE=integer>Name length</VAR>
  integer :: hash                   !# <VAR NAME=hash TYPE=integer>Hash</VAR>
  integer :: flag                   !# <VAR NAME=flag TYPE=integer>Flag</VAR>
  logical :: used                   !# <VAR NAME=used TYPE=logical>In use flag</VAR>
  integer :: dim1                   !# <VAR NAME=dim1 TYPE=integer>First dimension</VAR>
  integer :: dim2                   !# <VAR NAME=dim2 TYPE=integer>Second dimension</VAR>
  integer :: allocdim1              !# <VAR NAME=allocdim1 TYPE=integer>First dimension allocation</VAR>
  integer :: allocdim2              !# <VAR NAME=allocdim2 TYPE=integer>Second dimension allocation</VAR>
  integer :: exchange               !# <VAR NAME=exchange TYPE=integer>To exchange or not</VAR>
  integer :: ptr_index              !# <VAR NAME=ptr_index TYPE=integer>Pointer index at creation</VAR>
  type(r2_t), pointer :: next       !# <VAR NAME=next TYPE=r2_t>Next pointer</VAR>
end type r2_t
!# </STRUCTURE>
!=============================================================
```

```
  if (acc_is_present(r2%val)) then
     !$acc update device(r2)
     !$acc enter data create(r2%val)
     !$acc parallel loop collapse(2) present(r2%val,r2_tmp) copyin(new_allocdim1,new_allocdim2)
     do j=1,new_allocdim2
        do i=1,new_allocdim1
           r2%val(i,j) = r2_tmp(i,j)
        end do
     end do
     !$acc exit data delete(r2_tmp) finalize
   end if
```

GPUs & APUs

- Many helpers to minimize the impact of OpenACC on development

```
!loop on the cell groups
do n=1,grid%nel_grps
   el_grp => grid%el_grps(n)%ptr

   call get_array_val(residual_ptr%r1_ptrs(n)%ptr,residual_val,update_on_device=.true.)
   call get_array_val(phi_np1_ptr%r1_ptrs(n)%ptr,phi_np1_val,update_on_device=.true.)
   call get_array_val(udotAp_ptr%r1_ptrs(n)%ptr,udotAp_val,update_on_device=.true.)
   call get_array_val(el_grp%pair2node1,el_grp_pair2node1_val,check_on_device=.true.)
   call get_array_val(el_grp%pair2node2,el_grp_pair2node2_val,check_on_device=.true.)

   !loop on the node pairs (edges)
   npair = el_grp%npair

   !$acc parallel loop gang vector default(present) &
   !$acc & private(norm,normal,scal_val,uAp,corr,coeff,ip_diff,grad,vprod)
   do ip=1,npair
      ino1 = el_grp_pair2node1_val(ip)
      ino2 = el_grp_pair2node2_val(ip)

      ...

      ! residual update
      !$acc atomic update
      residual_val(ino1) = residual_val(ino1) + coeff
      !$acc end atomic
      !$acc atomic update
      residual_val(ino2) = residual_val(ino2) - coeff
      !$acc end atomic
   end do

   call update_int_comm(n,residual_ptr,grid,COMM_ADD,update_on_host=.true.)
end do
```

GPUs & APUs

- Status of OpenACC port in YALES2_2024.04
  - One solver is ported (SCS), two main solvers on-going (ICS, ECS)

- Supported on
  - NVIDIA, Jean-Zay with gfortran and nvfortran
  - Adastra AMD MI250: done with Cray compiler cce17 and works on multi-gpus with correct results

- Support to come
  - AMD MI300A (APU): CPU port done during ECFD7 Jan. 2024

- Many issues encountered along the road
  - Conditional pragmas have different behaviors in nvfortran and cce17
  - All kernels were asynchronous with Cray compiler
  - cce17 crashes when compiling big files, fix to come in cce19
  - Compilation time…