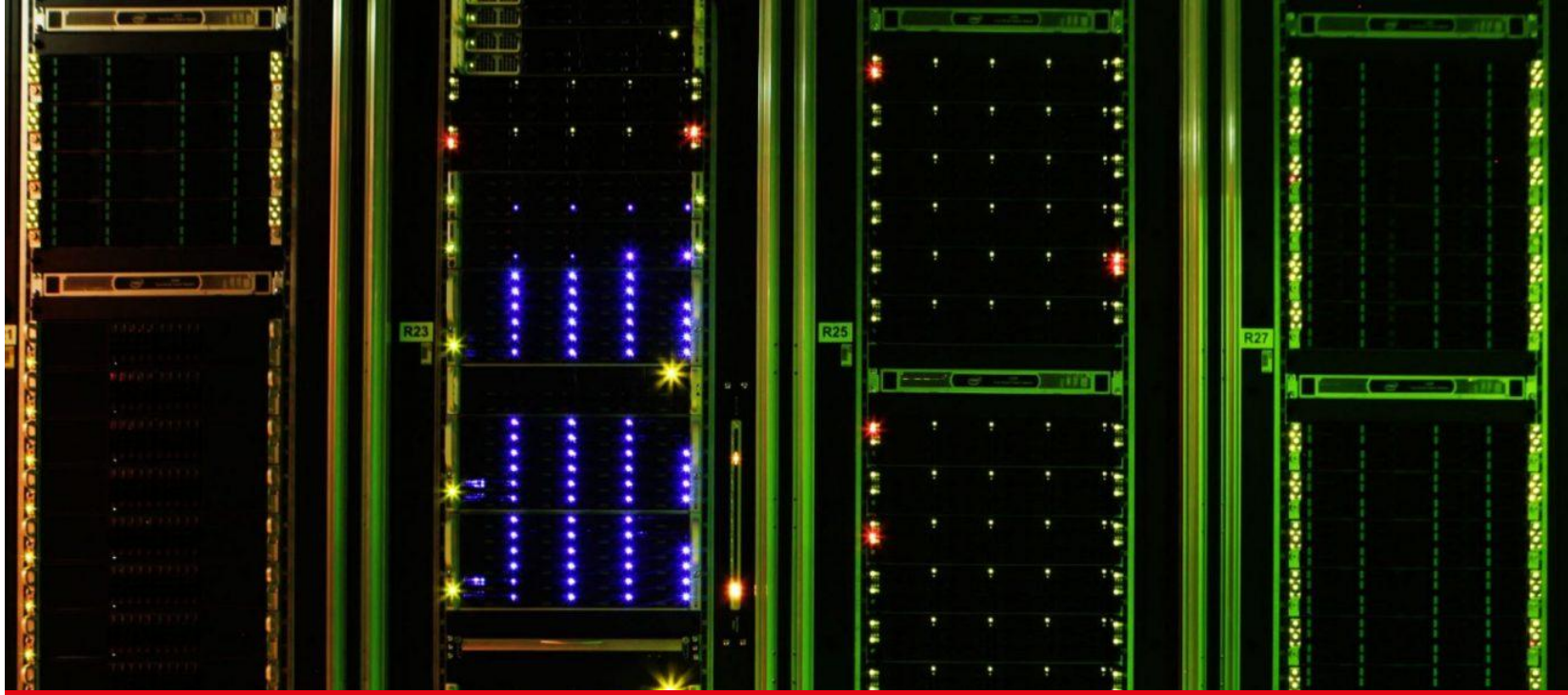


Use of OpenMP & OpenMP offload in GBS

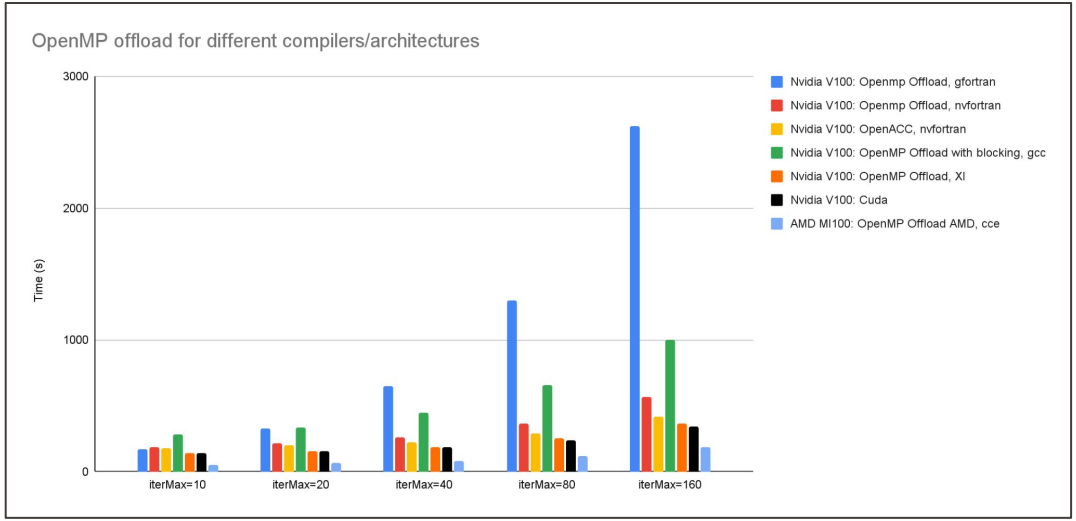
OpenMP Offload for GPU

- OpenMP offload works for Intel, Nvidia and AMD GPUs
- Only a single source code
- Performance with the classical stencil-based Jacobi example:
  - High dependency on compilers and architectures
  - On Nvidia, better performance reached with Xlf compiler on Marconi100

Jacobi algorithm

```
! Array allocation
  allocate(A(N, N))
  allocate(Anew(N, N))
! Array initialization
  call init_array(A, N, 7.0_dp)
  call init_array(Anew, N, 7.0_dp)
! set pointers
  A_pointer1 => A
  A_pointer2 => Anew
! Jacobi iterations
  do iterk=1,iter_max
     do j = 2, N-1
        do i = 2, N-1
           A_pointer2(i, j) = 0.25 * (A_pointer1(i, j+1) + &
                A_pointer1(i, j-1) + A_pointer1(i+1, j) + A_pointer1(i-1, j))
        end do
     end do
! swap of pointers
     call swap(A_pointer1,A_pointer2)
  enddo
```

Performance for different compilers/architectures (Fixed size array: N = 8192 x 8192)



OpenMP Offload in GBS

- Goal: use of Openmp-Offload in plasma evolution
- Compare OpenMP offload and cuda

OpenMP Offload in GBS

- Example of OpenMP offloading in Gradient computation:

```
!Compute perpendicular gradients
subroutine perpendicular_gradients
  use fields
  use array
  use gradients
  use time_integration, only: updatetlevel
  use model,only:nlpol

  use prec_const
  implicit none

  !Perpendicular gradients
  !$omp target enter data map(alloc:strmfy,strmfx,pi_y,pi_x,thetay,thetax,tempey,strmfz,pi_z)
  !$omp target enter data map(alloc:theta_curv_op,tempe_curv_op,tempi_curv_op,strmf_curv_op,theta_curv_op_v)

  !$omp task depend(in:strmf)
  call grady_n2n(strmf(:,:,:),strmfy(:,:,:))
  call gradx_n2n(strmf(:,:,:),strmfx(:,:,:))
  call gradz_n2n(strmf(:,:,:),strmfz(:,:,:))
  call curv_n2n(strmf(:,:,:), strmf_curv_op)
  !$omp end task

  !$omp task depend(in:theta)
  call grady_n2n(theta(:,:,:,updatetlevel),thetay(:,:,:))
  call gradx_n2n(theta(:,:,:,updatetlevel),thetax(:,:,:))
  call curv_n2n(theta(:,:,:,updatetlevel), theta_curv_op )
  call curv_n2v(theta(:,:,:,updatetlevel),theta_curv_op_v)
  !$omp end task

  !$omp task depend(in:tempe)
  call grady_n2n(tempe(:,:,:,updatetlevel),tempey(:,:,:))
  call curv_n2n(tempe(:,:,:,updatetlevel), tempe_curv_op)
  !$omp end task

  !$omp task depend(in:pri)
  call grady_n2n(pri(:,:,:),pi_y(:,:,:))
  call gradx_n2n(pri(:,:,:),pi_x(:,:,:))
  call gradz_n2n(pri(:,:,:),pi_z(:,:,:))
  !$omp end task
  !$omp target exit data map(from:strmfy,strmfx,pi_y,pi_x,thetay,thetax,tempey,strmfz,pi_z)

  !$omp task depend(in:tempi)
  call curv_n2n(tempi(:,:,:,updatetlevel), tempi_curv_op )
  !$omp end task

! !$omp target exit data map(from:theta_curv_op,tempe_curv_op,tempi_curv_op,strmf_curv_op,theta_curv_op_v)

end subroutine perpendicular_gradients
```

OpenMP Offload in GBS

- Example of OpenMP offloading in Gradient computation:

```
! parallel gradient for finite differences 4rth order from n grid to n grid
subroutine gradpar_v2v_fd4(f, flu, frd, f_grad)

  use prec_const

  implicit none
  real(dp), dimension(iysg:iyeg,ixsg:ixeg,izsg:izeg), intent(in) :: f
  real(dp), dimension(iysg:iyeg,ixsg:ixeg,izsg:izeg), intent(out):: f_grad
  real(dp), dimension(iysg:iyeg,ixsg:ixeg,izsg:izeg)      :: f_z,f_y,f_x
  real(dp), dimension(iylg:ny_zg,ixsg:ixeg,2), intent(in) :: flu, frd
  integer :: ix, iy,iz

  !  f_grad(:,:,:) = nan_

  !$omp target enter data map(alloc:f_z,f_y,f_x)

  call gradz_n2n_fd4(f,f_z )
  call grady_n2n_fd4(f,f_y)
  call gradx_n2n_fd4(f,f_x)

  !$omp target teams distribute parallel do simd collapse(3)
  do iz = izs,ize
     do ix = ixs, ixe
        do iy = iys, iye
           f_grad(iy,ix,iz) = gradpar_z*f_z(iy,ix,iz) + gradpar_y_v(iy,ix)*f_y(iy,ix,iz)&
                + gradpar_x_v(iy,ix)*f_x(iy,ix,iz)
        end do
     end do
  end do
  !$omp end target teams distribute parallel do simd
  !$omp target exit data map(delete:f_z,f_y,f_x)

end subroutine gradpar_v2v_fd4
```

OpenMP for GPU On Marconi100

- We compared CUDA implementation (see Nicola's talk) vs OpenMP-offload one
- Setup: Reduced TCV at 0.9T, 2 timesteps
  - Turbulent mode
  - Nx = 400, ny = 800, nz = 4
  - 1 node M100@CINECA
    - 1 NVIDIA V100 16GB
  - IBM Xlf Compiling Environment

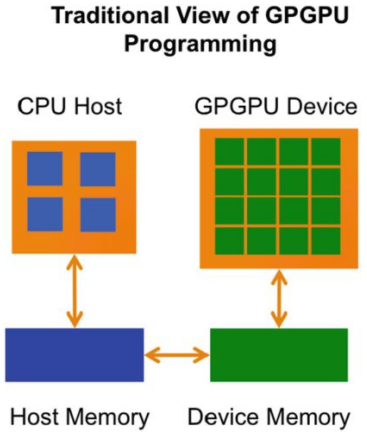
Subroutine	CUDA	OpenMPOffload
Global Plasma module	0.85	0.89
parallel gradients	0.13	0.13
no Boussinesq vorticity	0.2	0.15
interpolation	0.019	0.01
perpendicular gradient	0.018	0.01
giroviscous term	0.01	0.04
diffusion operator	0.02	0.01

OpenMP in GBS for CPU

- Each GPU is usually associated to a single MPI process
- A multi-cores socket is usually associated to a GPU
- Use OpenMP to exploit remaining CPU cores
- Use OpenMP from OpenMP-offload version is straightforward - compilation option
  - e.g: *ifort -qopenmp -qno-openmp-offload*

—> compilation output:

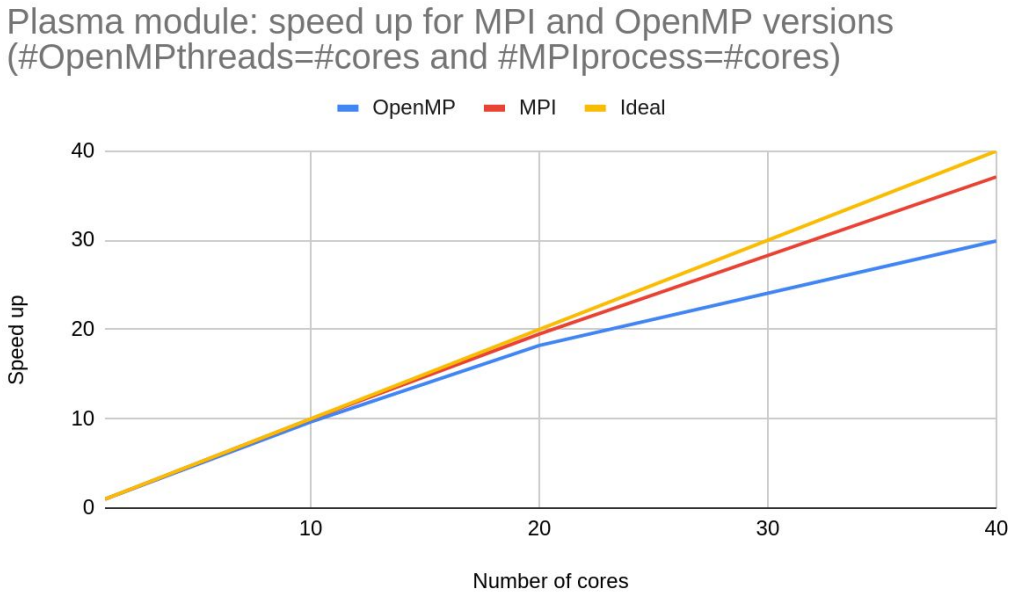
remark #8711: OpenMP directive disabled via command line.*
!$omp target teams distribute parallel do simd collapse(2)
*---------^*



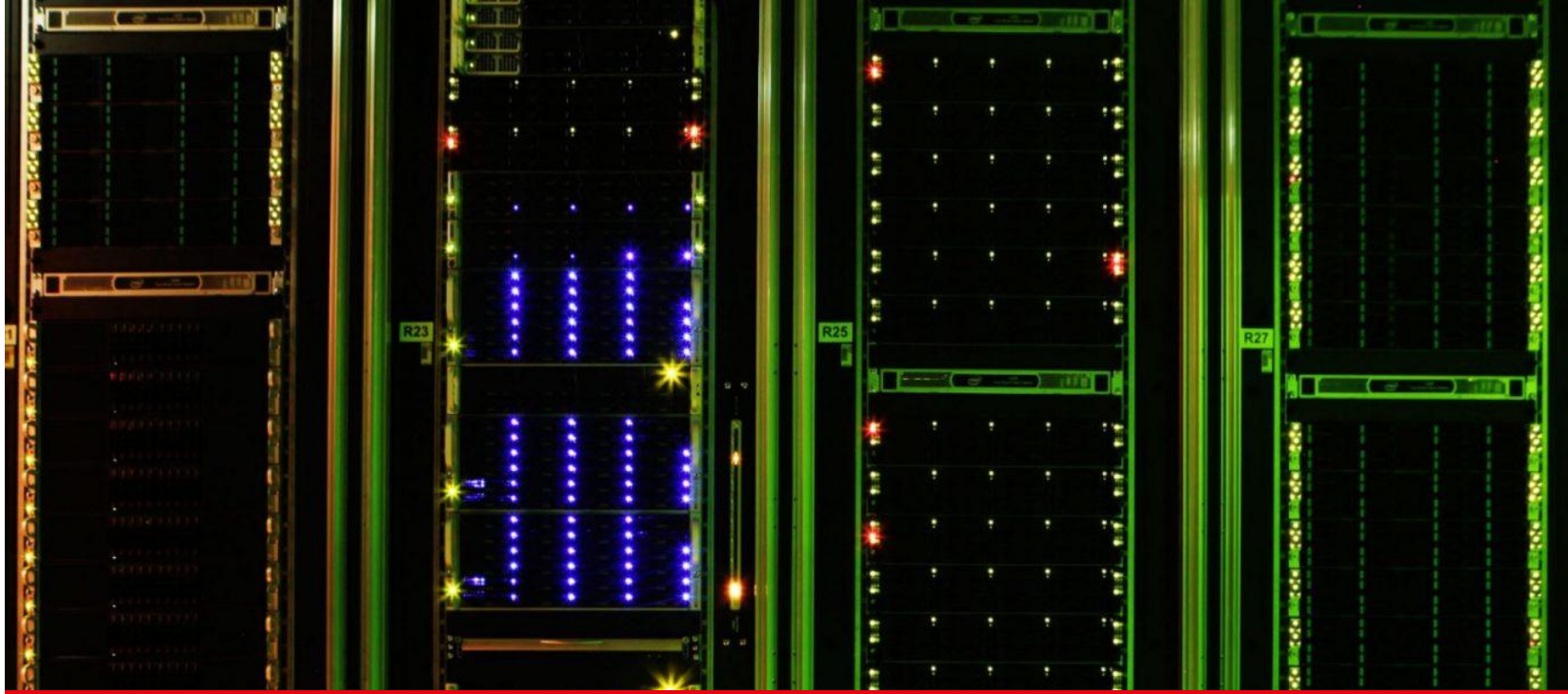
OpenMP in GBS for CPU

- We compared initial CPU serial implementation, pure OpenMP one and pure MPI one

- Setup: Reduced TCV at 0.9T, 2 timesteps
  - Turbulent mode
  - Nx = 600, ny = 1200, nz = 4
  - 1 node izar
    - 2 Intel Xeon-Gold processors running at 2.1 GHz, with 20 cores each
    - Intel compiler



EPFL



Use of OpenACC in GBS-Plasma

OpenACC in GBS-RHS for GPU

- Use of OpenACC to port to GPU
  - Iterative process
  - “Fast” learning curve
  - Single source code
  - Easily exchanged to OpenMP for more portability
  - More efficient than OpenMP on Nvidia GPU for some compilers

OpenACC in GBS-Plasma for GPU

OpenACC in GBS-RHS for GPU

- Use of Piz-daint to test OpenACC with PGI compiler
- Goals: “replace” OpenMP directives use in previous work by OpenACC directives for loop and data transfer:
  - first, using managed memory with OpenACC compiling with -acc -ta=tesla:managed
  - then optimize data transfer following current openmp offload data transfer

typical kernel in GBS-RHS

```
! parallel gradient for finite differences 4rth order from n grid to n grid
subroutine gradpar_v2v_fd4(f, f_grad)

  use prec_const

  implicit none
  real(dp), dimension(iysg:iyeg,ixsg:ixeg,izsg:izeg), intent(in) :: f
  real(dp), dimension(iysg:iyeg,ixsg:ixeg,izsg:izeg), intent(out):: f_grad
  real(dp), dimension(iysg:iyeg,ixsg:ixeg,izsg:izeg)      :: f_z,f_y,f_x
  integer :: ix, iy,iz

  !$acc enter data create(f_z,f_y,f_x)
  !$omp target enter data map(alloc:f_z,f_y,f_x)

  call gradz_n2n_fd4(f,f_z )
  call grady_n2n_fd4(f,f_y)
  call gradx_n2n_fd4(f,f_x)

  !$omp target teams distribute parallel do simd collapse(3)
  !$acc parallel loop collapse(3) !present(f_grad,f_y,f_x,f_z,gradpar_y_v,gradpar_x_v)
  do iz = izs,ize
     do ix = ixs, ixe
        do iy = iys, iye
           f_grad(iy,ix,iz) = gradpar_z*f_z(iy,ix,iz) + gradpar_y_v(iy,ix)*f_y(iy,ix,iz)&
                + gradpar_x_v(iy,ix)*f_x(iy,ix,iz)
        end do
     end do
  end do
  !$omp end target teams distribute parallel do simd
  !$omp target exit data map(delete:f_z,f_y,f_x)
  !$acc exit data delete(f_z,f_y,f_x)

end subroutine gradpar_v2v_fd4
```

OpenACC in GBS-RHS for GPU

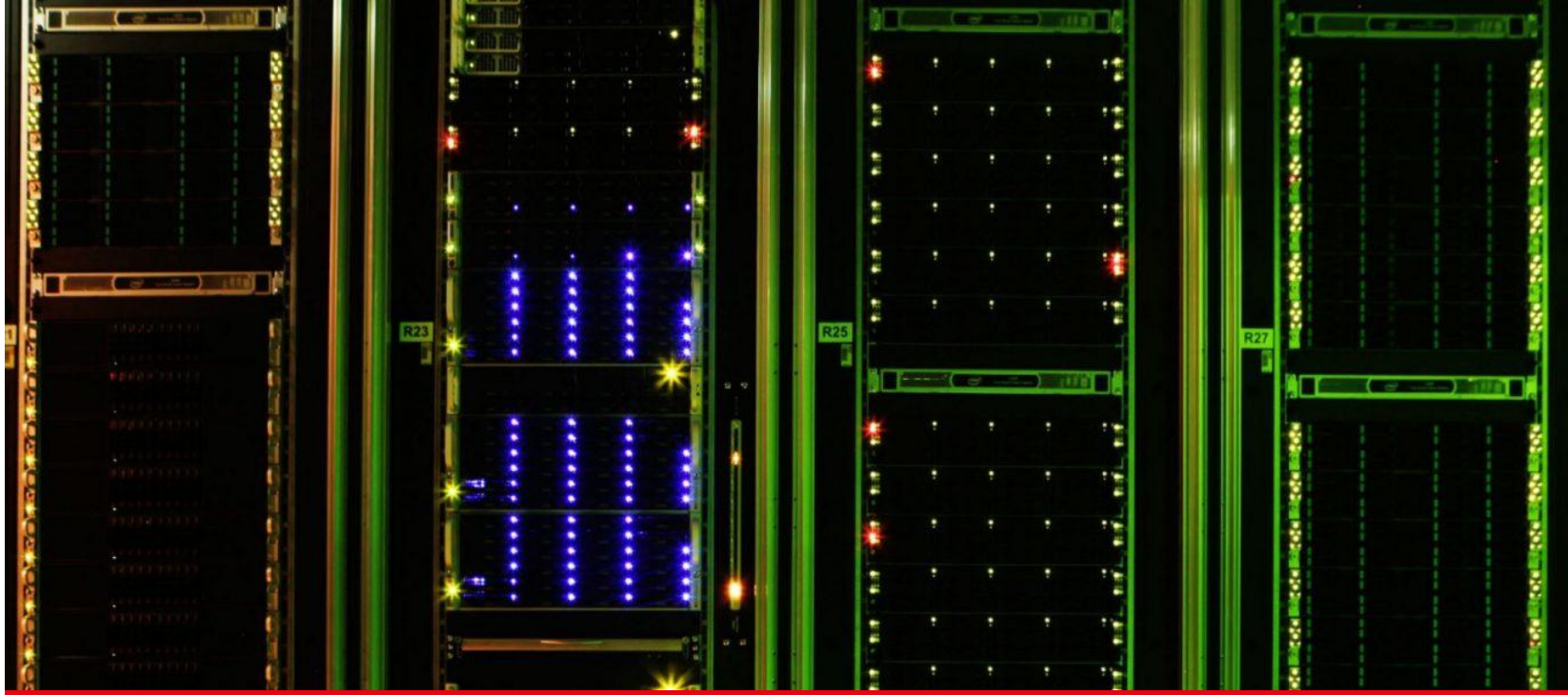
- We compared initial CPU serial implementation vs OpenACC one
- Setup: Reduced TCV at 0.9T, 2 timesteps
  - Turbulent mode
  - Nx = 1000, ny = 2000, nz = 4
  - 1 node piz-daint@CSCS
    - 1 NVIDIA Tesla P100 16GB
  - PGI Compiling Environment

Subroutine	CPU serial (one core)	OpenACC (managed option)		OpenACC (data transfer optimized)	
	time(s)	time (s)	speed up	time (s)	speed up
Global Plasma module	**159.**	**57**	**x2.8**	**10.5**	**x15.1**
parallel gradients	112.7	21.5	x2.9	1.44	x78
no Boussinesq vorticity	11.3	7	x1.58	1.6	x6.4
interpolation	3.3	2.9	x1.3	0.37	x8.8
perpendicular gradient	6.23	4.1	x1.5	0.74	x8.4
transfer HtoD/DtoH + Gpu page fault		35		7.	

OpenACC in GBS-RHS for GPU

- We compared initial CPU MPI implementation vs OpenACC one
- Setup: Reduced TCV at 0.9T, 2 timesteps
  - Turbulent mode
  - Nx = 1000, ny = 2000, nz = 4
  - 1 node piz-daint@CSCS
    - 12-cores Intel Xeon - 2.6GHz
    - 1 NVIDIA Tesla P100 16GB
  - PGI Compiling Environment

Subroutine	12-cores MPI (1 node)	OpenACC (data transfer optimized)	
	time(s)	time (s)	speed up
Global Plasma module	**23.**	**10.5**	**2.2**
parallel gradients	16.	1.44	11.1
no Boussinesq vorticity	3.	1.6	1.8
interpolation	1.	0.37	2.7
perpendicular gradient	1.6	0.74	2.1



Porting GBS neutral module to GPUs

First result with OpenACC

Neutral dynamics with the method of characteristics

- Consider simple kinetic neutral model (single species, ionization, charge exchange, and recombination) + B.C.

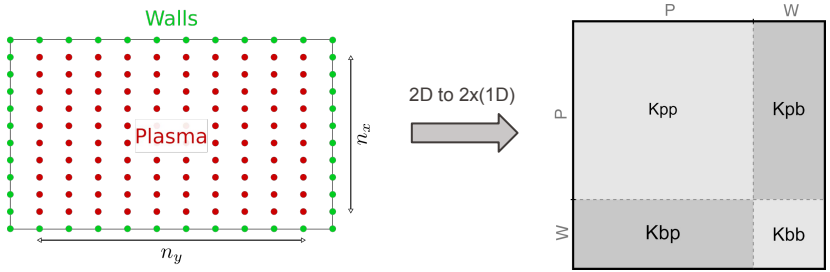
$$\frac{\partial f_\mathrm{n}}{\partial t} + \mathbf{v}\cdot\frac{\partial f_\mathrm{n}}{\partial \mathbf{x}} = -\nu_\mathrm{iz} f_\mathrm{n} - \nu_\mathrm{cx}\left(f_\mathrm{n} - \frac{n_\mathrm{n}}{n_\mathrm{i}} f_\mathrm{i}\right) + \nu_\mathrm{rec} f_\mathrm{i} \quad \boxplus \quad f_\mathrm{n}(\mathbf{x}_\mathrm{b},\mathbf{v}) = (1-\alpha_\mathrm{refl})\Gamma_\mathrm{out}(\mathbf{x}_\mathrm{b})\chi_\mathrm{in}(\mathbf{x}_\mathrm{b},\mathbf{v}) + \alpha_\mathrm{refl}[f_\mathrm{n}(\mathbf{x}_\mathrm{b},\mathbf{v}-2\mathbf{v}_\mathrm{p}) + f_\mathrm{i}(\mathbf{x}_\mathrm{b},\mathbf{v}-2\mathbf{v}_\mathrm{p})]$$

- Solution found using the noise-free characteristics method (and various approximations),see [Wersal and Ricci, Nucl. Fusion, 55 (2015)]:

$$\begin{bmatrix} n_\mathrm{n} \\ \Gamma_\mathrm{out} \end{bmatrix} = \begin{bmatrix} K_{\mathrm{p}\to\mathrm{p}} & K_{\mathrm{b}\to\mathrm{p}} \\ K_{\mathrm{p}\to\mathrm{b}} & K_{\mathrm{b}\to\mathrm{b}} \end{bmatrix} \cdot \begin{bmatrix} n_\mathrm{n} \\ \Gamma_\mathrm{out} \end{bmatrix} + \begin{bmatrix} n_\mathrm{n,rec} \\ \Gamma_\mathrm{out,rec} + \Gamma_\mathrm{out,i} \end{bmatrix} \quad \Rightarrow \quad \boldsymbol{Ax = b}$$

- with **matrix elements resulting from complex integrals in space and velocity**, involving Bessel functions etc., e.g.

$$K_{\mathrm{p}\to\mathrm{p}}(\boldsymbol{x}_\perp, \boldsymbol{x}'_\perp) = \int_0^\infty \frac{1}{r'_\perp}\Phi_{\perp\mathrm{i}}(\boldsymbol{x}'_\perp, \boldsymbol{v}_\perp)\exp\left[-\frac{1}{v_\perp}\int_0^{r'_\perp}\nu_\mathrm{eff}(\boldsymbol{x}''_\perp)\mathrm{d}r''_\perp\right]\mathrm{d}v_\perp$$



~$N_x N_y$ grid ~$(N_x N_y)^2$ elements

Code organization & strategy

Profiling the neutral module gives three main bottlenecks:

- compute_K: compute the K matrices
- Solve: solve neutral system
- Get_moments: compute various neutral moments for the plasma and/or diag.
  - Combination of compute_K and matrix/vector multiplication

For ease of development:

- Implemented a miniapp with only neutrals (no solver -> third party)
- Used in OpenACC hackathon & basis for student project (Louis Jaugey)

Preliminary results

- Only compute_K has been fully ported to GPU
- Initial timings show that GPU vs CPU (1 MPI task) ~5.4x speedup
  - smaller TCV case: 50x50 grid, 50 points in velocity space, 30 points for interpolations
  - Izar cluster @ EPFL, Xeon-Gold @ 2.1 GHz, NVIDIA V100 PCIe 32 GB
  - Adding tasks reduces the speedup because trivially parallel w/o solver

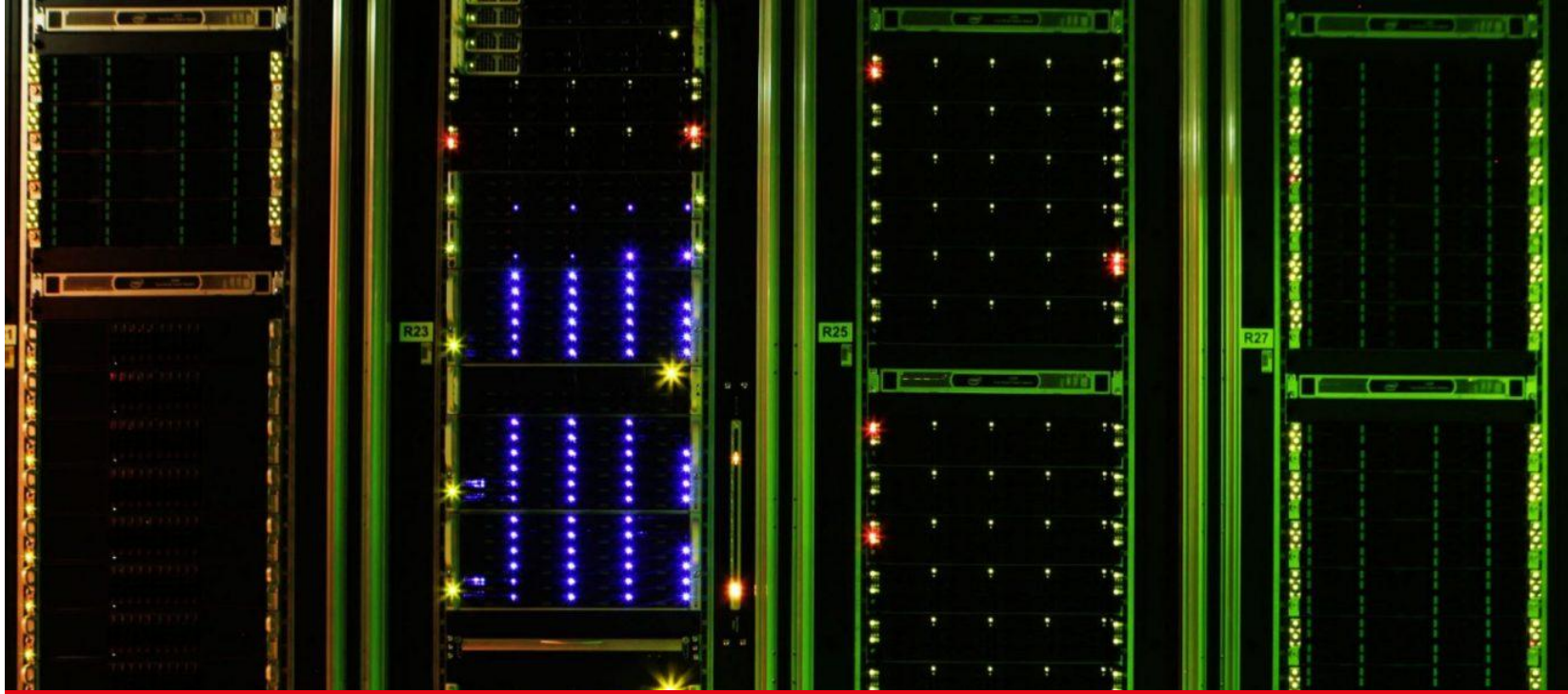
Discussion

- Data transfers GPU <-> CPU are critical; reduce them as much as possible
  - K matrices are huge ~$(NxNy)^2$
- Current work is to port get_moments (compute_K + mat/vec algebra)
  - Further speedup from mat/vec algebra done on GPU
  - Only need to transfer vectors back to GPU

Conclusions & next steps

- First OpenACC implementation in the neutrals and RHS
- Directive based porting allows:
  - Same source code
  - Relatively quick porting
  - “Portability”

- Finish porting all the computations of get_moments to GPU
- Port solver part to GPU
- Optimization of memory requirements
  - Avoid unnecessary arrays
  - Multi-GPU computations
  - Use communication via GPU-to-GPU interconnect to update ghost cells, avoiding transfer GPU<->CPU



TSVV-3, 6 - FLUIDOPT: Profiling and opimisation Soledge3X

- **Goals:**
  - profiling Soledge3X on SCITAS and Marconi clusters
  - implement performance metrics to understand main bottlenecks
  - **optimize and porting to GPU some parts of the code**

- **Current status:**
  - Soledge3X uses MPI+OpenMP
  - it relies on a mix explicit-implicit scheme
  - **it uses Petsc, Pastix, Hypre for implicit solvers**
  - many profilings have been performed
  - Soledge3X using PETSc has been installed on SCITAS and Marconi clusters with Intel toolchain
  - regular contacts with developers

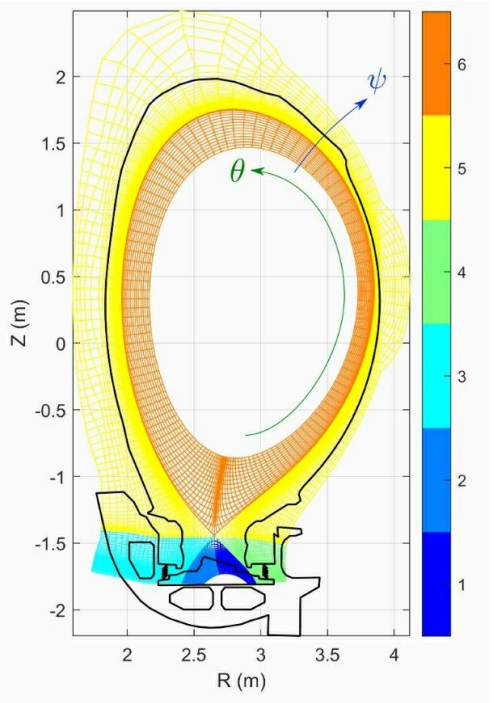
- **Previous conclusions:**
  - Profiling shows most of the computation time is spent within the libraries to solve the 3D electric potential implicit equation
  - MPI parallel efficiency depends on the ratio of the number of MPI processes and the number of magnetic flux surfaces
  - OpenMP is quite efficient except for linear solvers (PETSC doesn't use threads)

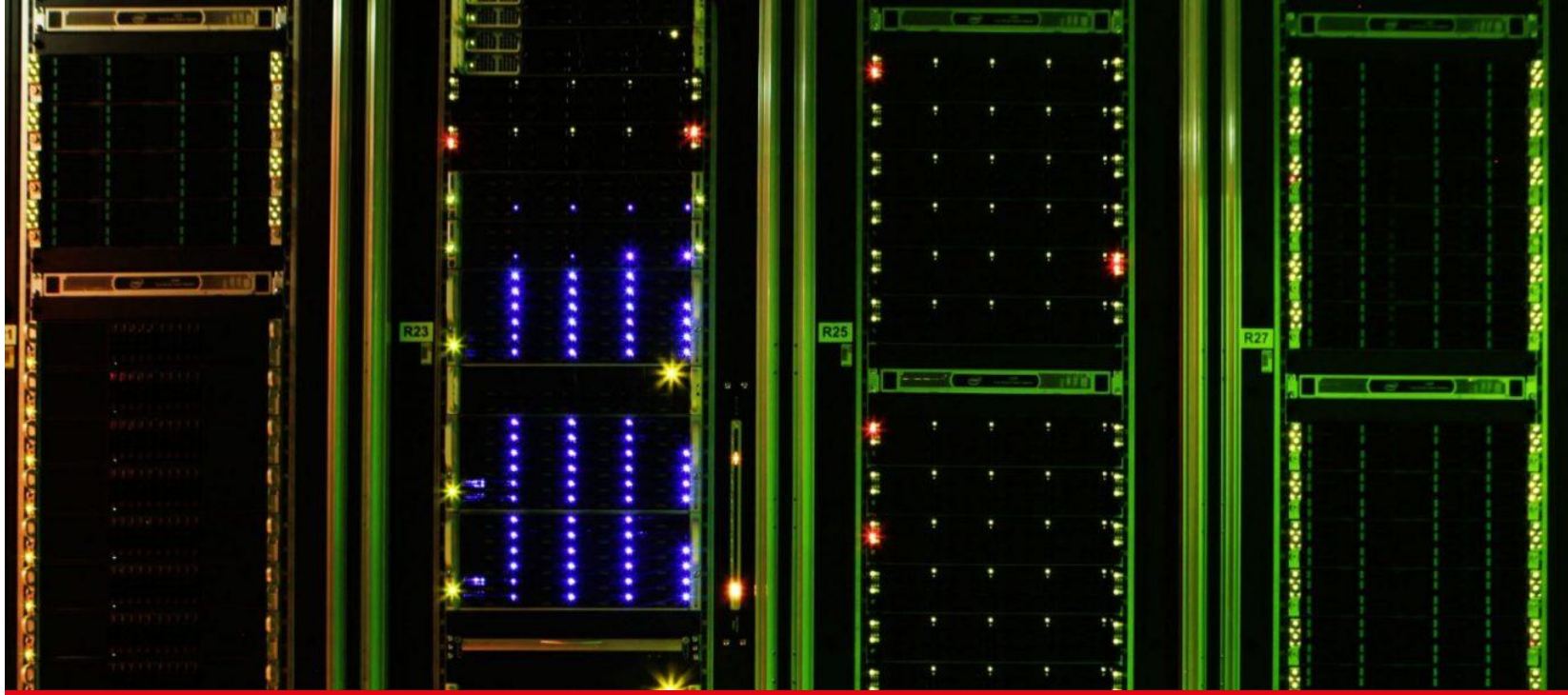
Time-stepping scheme

- Main loop algorithm regarding main CPU time-consuming routines

Parallelization

- Spatial discretization:
  - structured grid in the (ψ,θ,φ) coordinate system aligned with magnetic flux surfaces (ψ associated with the magnetic flux)
  - the solvers *evolveImplicitMomentum* and *evolveImplicitEnergy* are built using 2D stencils located in magnetic flux surface:
    → independant linear 2D mesh-based solvers are called for each value of ψ (magnetic flux surface)
  - However, the solver *evolveImplicitElectricPotential* is 3D mesh-based
- PETSC, PASTIX and HYPRE can be used for implicit solvers
- MPI domain decomposition according to the (ψ,θ,φ) structured grid: the domain is in priority decomposed along the ψ direction (according the magnetic flux surface workload), then along the θ direction
- MPI communicator for each magnetic flux surface (each value of ψ), useful for 2D mesh-based solvers
- OpenMP is used for each MPI process, except in PETSC and HYPRE solvers





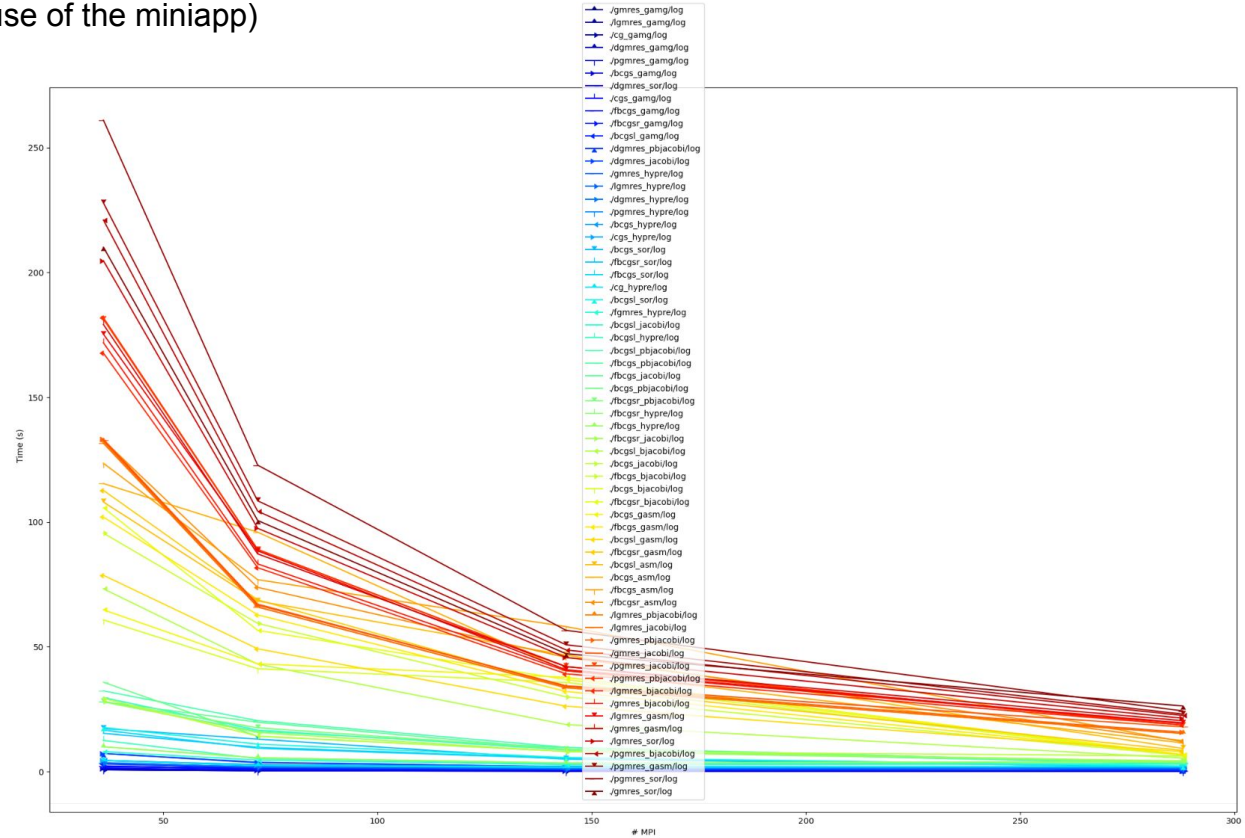
Profiling and optimization of Solvers in Soledge3X

Miniapps for linear system

- Use of Miniapps
  - New routine in Soledge3X for dumping matrices in PETSC format for all implicit solvers
  - Miniapps load matrices and solve linear system with PETSC and AMGX (see Nicola’s talk on solvers)
  - We study performance for different couples solver/preconditionner
  - New python script to plot scaling according to #MPI processes

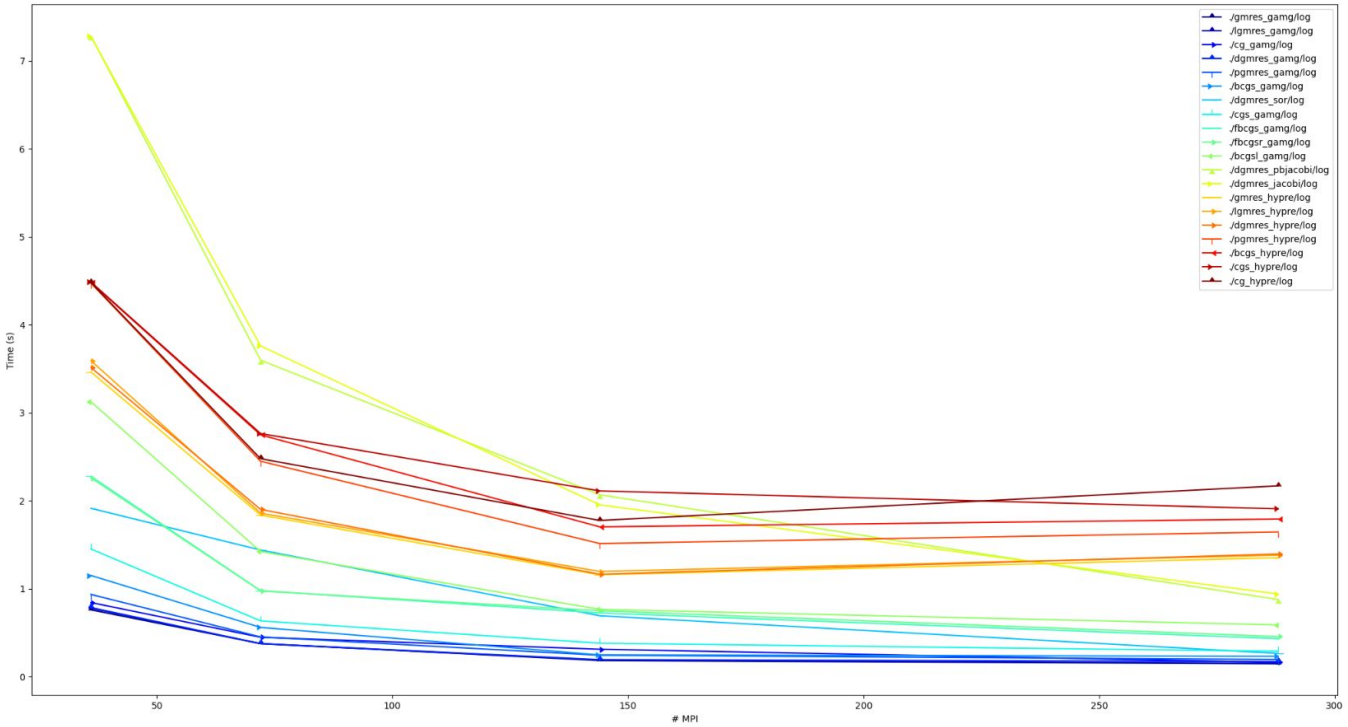
TCV-Timing solver 3D with PETSC

- TCV test-case: timing for the 3d linear solver (reusing preconditioning) using the matrix dumped from the TCV case (use of the miniapp)



TCV-Timing solver 3D with PETSC-zoom

- TCV test-case: timing for the 3d linear solver (reusing preconditioning) using the matrix dumped from the TCV case (use of the miniapp) - zoom



TCV-Timing solver 3D with PETSC

TCV-3D-36-MPI

Norm	solver_precond	time reusing preconditionning
0,04488280278	gmres_gamg	0,7525878432
0,04488280284	lgmres_gamg	0,7535700276
0,04488280283	dgmres_gamg	0,7565472894
0,04575610405	cg_gamg	0,8222253188
0,04488280279	pgmres_gamg	0,9235707652
0,01940921561	bcgs_gamg	1,133472991
0,01439650238	cgs_gamg	1,443112343
0,0267549819	fgmres_gamg	1,926239386
0,03671255467	dgmres_sor	1,932614055
0,01401087505	fbcgsr_gamg	2,253752024
0,01401087505	fbcgs_gamg	2,278856417
2,22E-05	bcgsl_gamg	3,117366321
0,0157930183	gmres_hypre	3,457721147
0,0157930183	dgmres_hypre	3,486946244
0,0157930183	lgmres_hypre	3,489860036
0,01975034769	bcgs_hypre	4,499845477
0,0157930183	pgmres_hypre	4,50615459
0,02350953298	cgs_hypre	4,514745466
0,04159422673	cg_hypre	4,519228674

TCV-Timing solver 3D with PETSC

TCV-3D-288-MPI

Norm	solver_precond	time reusing preconditionning
0,04588284819	gmres_gamg	0,1396856951
0,04749503765	cg_gamg	0,1405559438
0,04588284823	lgmres_gamg	0,1328434579
0,04588284824	dgmres_gamg	0,1449191147
0,04588284822	pgmres_gamg	0,1809122302
0,01965201655	bcgs_gamg	0,2143931612
0,02812369939	dgmres_sor	0,2688934538
0,01596058082	cgs_gamg	0,2779521719
0,02492266819	fgmres_gamg	0,3731893594
0,02634344349	fbcgs_gamg	0,4592761695
0,02634344349	fbcgsr_gamg	0,4427626932
1,68E-05	bcgsl_gamg	0,5735744894
0,0123953568	dgmres_pbjacobi	0,7672227672
0,01223723629	dgmres_jacobi	0,842321082
0,01592096196	lgmres_hypre	1,34849206
0,01592096196	dgmres_hypre	1,371061352
0,01592096196	gmres_hypre	1,312337736
0,02675850511	cgs_hypre	1,690713092
0,01592096196	pgmres_hypre	1,724306964

Miniapp: use of AMGX

- Miniapp routine

```
..... PETSC INIT .....

! solving with AMGX
call allocate_amgx_struct(amgx_struct)
call init_amgx(amgx_struct, MPI_COMM_WORLD, 2)
mataddr = A%v
rhs_addr = rhs_petsc%v
lhs_addr = lhs%v
call set_amgx(amgx_struct, mataddr, rhs_addr, lhs_addr, MPI_COMM_WORLD)
call solve_amgx(amgx_struct)
! solving with PETSC
call KSPCreate(PETSC_COMM_WORLD,ksp,ierr)
call KSPSetFromOptions(ksp,ierr)
call KSPSetOperators(ksp,A,A,ierr)
call KSPSolve(ksp,rhs_petsc,lhs_petsc,ierr)
```

Miniapp: use of AMGX

- Matrix dumped from circ_3D_onlyD_noNeutr case (50x500x50)
- Miniapp allows to compare PETSC and AMGX solver
- Miniapp compiled with gnu-cuda
- Result with PETSC miniapp ok with result Soledge checked
- Results for Phi 3D matrix: (1MPI process Vs 1GPU)

```
 Solving with AMGX .....
("solver": "PBICGSTAB", "preconditioner":AMG)
AMGX version 2.2.0.132-opensource
        ---------------------------------------------------------
        Total Iterations: 110
        Avg Convergence Rate:               0.8813
        Final Residual:                 2.826928e-01
        Total Reduction in Residual:        9.221377e-07
        Maximum Memory Usage:               2.249 GB
        ---------------------------------------------------------
 TIME AMGX =        1.3683695793151855
 AMGX solution norm L2  3125.2008633342521
 Solving with Petsc .....
 TIME PETSC =       40.264363288879395
 PETSc solution norm L2  3123.7779412178043
 ----------         PETSC          ----------------------
 Krylov solver: bcgs
 Preconditioner: gamg
  Number of iterations:          8
 Residual norm from solver and check:   2.9148633099987314E-002
```

Miniapp: use of AMGX

- Matrix dumped from circ_3D_onlyD_noNeutr case (50x500x50)
- miniapp allows to compare PETSC and AMGX solver
- miniapp compiled with gnu-cuda
- result with PETSC miniapp ok with result Soledge checked
- Results for Phi matrix: (20MPI process Vs 1GPU)

```
 Solving with AMGX .....
("solver": "PBICGSTAB", "preconditioner":AMG)
AMGX version 2.2.0.132-opensource
        ---------------------------------------------------------
        Total Iterations: 110
        Avg Convergence Rate:               0.8813
        Final Residual:                 2.826928e-01
        Total Reduction in Residual:        9.221377e-07
        Maximum Memory Usage:               2.249 GB
        ---------------------------------------------------------
 TIME AMGX =        1.3683695793151855
AMGX solution norm L2  3125.2008633342521
 Solving with Petsc .....
 TIME PETSC =       4.6425805091857910
PETSc solution norm L2  3123.9884891099045
----------         PETSC          ----------------------
 Krylov solver: bcgs
 Preconditioner: gamg
  Number of iterations:          9
Residual norm from solver and check:   2.0763482339195199E-002
```

Miniapp: use of AMGX

- Matrix dumped from circ_3D_onlyD_noNeutr case (50x500x50)
- miniapp allows to compare PETSC and AMGX solver
- miniapp compiled with gnu-cuda
- result with PETSC miniapp ok with result Soledge checked
- Results for Phi matrix: (40MPI process Vs 2GPU)

```
 Solving with AMGX .....
("solver": "PBICGSTAB", "preconditioner":AMG)
AMGX version 2.2.0.132-opensource
        ---------------------------------------------------------
        Total Iterations: 110
        Avg Convergence Rate:               0.8813
        Final Residual:                 2.826928e-01
        Total Reduction in Residual:        9.221377e-07
        Maximum Memory Usage:               2.249 GB
        ---------------------------------------------------------
 TIME AMGX =        2.6621108055114746
AMGX solution norm L2  3125.2008627290916
 Solving with Petsc .....
 TIME PETSC =       3.5797915458679199
PETSc solution norm L2  3124.0291715355133
----------          PETSC          ----------------------
 Krylov solver: bcgs
 Preconditioner: gamg
  Number of iterations:          9
Residual norm from solver and check:   2.1662928432653411E-002
```

Hypre

- HYPRE
  - Hypre allows to exploit threads and GPUs
  - Hypre has been installed on Helvetios SCITAS cluser with openmp option
  - https://hypre.readthedocs.io/_/downloads/en/latest/pdf/ : "*Configuration of hypre with threads requires an implementation of OpenMP. Currently, only a subset of hypre is threaded.*"
  - Soledge3X has been linked with the Hypre library
  - OpenMP coarse-grain parallelism is implemented in Soledge3X:

    —> Need to manage nested OpenMP regions when calling Hypre in Soledge3X

Hypre - OpenMP

- Use of a Fortran miniapp to test OpenMP nested loops and cores pinning

```
31! Jacobi on CPU with OpenMP
32  call init_array(A_omp, N, 7.0_dp)
33  call init_array(Anew_omp, N, 7.0_dp)
34!  print *, "hello from thread: ", OMP_GET_THREAD_NUM()
35  do iterk=1,iter_max
36     call jacobi_openmp(A_omp, Anew_omp, N)
37  enddo
38  call test_array(A_omp,A,N)
39
40! Jacobi on CPU with OpenMP nested
41  call init_array(A_omp, N, 7.0_dp)
42  call init_array(Anew_omp, N, 7.0_dp)
43  !$OMP PARALLEL
44!  print *, "hello from thread: ", OMP_GET_THREAD_NUM()
45  do iterk=1,iter_max
46     !$OMP MASTER
47     call jacobi_openmp_nested(A_omp, Anew_omp, N)
48     !$OMP END MASTER
49   enddo
50  !$OMP END PARALLEL
51  call test_array(A_omp,A,N)
```

```
105  subroutine jacobi_openmp(Tab, Tabnew, N)
106    real(dp), intent(in)      :: Tab(N,N)
107    real(dp), intent(out)     :: TabNew(N,N)
108    integer,  intent(in)      :: N
109    integer i, j
110    !$omp parallel do
111    do j = 2, N-1
112       do i = 2, N-1
113          TabNew(i, j) = 0.25 * (Tab(i, j+1) + Tab(i, j-1) + Tab(i+1, j) + Tab(i-1, j))
114       end do
115    end do
116    !$omp end parallel do
117  end subroutine jacobi_openmp
118
119  subroutine jacobi_openmp_nested(Tab, Tabnew, N)
120    real(dp), intent(in)      :: Tab(N,N)
121    real(dp), intent(out)      :: TabNew(N,N)
122    integer,  intent(in)                :: N
123    integer i, j
124    !$omp parallel
125    !$omp do
126    do j = 2, N-1
127       do i = 2, N-1
128          TabNew(i, j) = 0.25 * (Tab(i, j+1) + Tab(i, j-1) + Tab(i+1, j) + Tab(i-1, j))
129       end do
130    end do
131    !$omp end do
132    !$omp end parallel
133  end subroutine jacobi_openmp_nested
```

Hypre - OpenMP

- With intel, don't use cores pinning and let free threads (don't use for instance *KMP_AFFINITY="compact,1,0* and set OMP_NESTED=TRUE.
- Example with 4 threads for 4 cores with nested regions:
  —> First, 4 threads are created and bound to 4 cores
  —> 3 cores idle in the OMP MASTER region
  —> Output (*KMP_AFFINITY=verbose,none*):

```
OMP: Info #154: KMP_AFFINITY: Initial OS proc set respected: 0-3
OMP: Info #191: KMP_AFFINITY: 1 socket x 4 cores/socket x 1 thread/core (4 total cores)
OMP: Info #251: KMP_AFFINITY: pid 27199 tid 27199 thread 0 bound to OS proc set 0-3
OMP: Info #251: KMP_AFFINITY: pid 27199 tid 27241 thread 1 bound to OS proc set 0-3
OMP: Info #251: KMP_AFFINITY: pid 27199 tid 27242 thread 2 bound to OS proc set 0-3
OMP: Info #251: KMP_AFFINITY: pid 27199 tid 27243 thread 3 bound to OS proc set 0-3
```

```
43  !$OMP PARALLEL
44!   print *, "hello from thread: ", OMP_GET_THREAD_NUM()
45  do iterk=1,iter_max
46     !$OMP MASTER
47     call jacobi_openmp_nested(A_omp, Anew_omp, N)
48     !$OMP END MASTER
49   enddo
50  !$OMP END PARALLEL
```

Hypre - OpenMP

- With intel, don't use cores pinning and let free threads (don't use for instance *KMP_AFFINITY="compact,1,0* and set OMP_NESTED=TRUE.
- Example with 4 threads for 4 cores with nested regions:
  —> Then Master thread creates 3 new threads (in red) at the time when it encounters a new nested OpenMP region
  —> These new threads are binded to cores let idle by the 3 other threads created initially and waiting for the master thread at the end of the nested region.
  —> Output (*KMP_AFFINITY=verbose,none*):

OMP: Info #154: KMP_AFFINITY: Initial OS proc set respected: 0-3
OMP: Info #191: KMP_AFFINITY: 1 socket x 4 cores/socket x 1 thread/core (4 total cores)
OMP: Info #251: KMP_AFFINITY: pid 27199 tid 27199 thread 0 bound to OS proc set 0-3
OMP: Info #251: KMP_AFFINITY: pid 27199 tid 27241 thread 1 bound to OS proc set 0-3
OMP: Info #251: KMP_AFFINITY: pid 27199 tid 27242 thread 2 bound to OS proc set 0-3
OMP: Info #251: KMP_AFFINITY: pid 27199 tid 27243 thread 3 bound to OS proc set 0-3
OMP: Info #251: KMP_AFFINITY: pid 27199 tid 27428 thread 5 bound to OS proc set 0-3
OMP: Info #251: KMP_AFFINITY: pid 27199 tid 27427 thread 4 bound to OS proc set 0-3
OMP: Info #251: KMP_AFFINITY: pid 27199 tid 27429 thread 6 bound to OS proc set 0-3

```
43  !$OMP PARALLEL
44!   print *, "hello from thread: ", OMP_GET_THREAD_NUM()
45  do iterk=1,iter_max
46     !$OMP MASTER
47     call jacobi_openmp_nested(A_omp, Anew_omp, N)
48     !$OMP END MASTER
49   enddo
50  !$OMP END PARALLEL
```

```
119  subroutine jacobi_openmp_nested(Tab, Tabnew, N)
120    real(dp), intent(in)       :: Tab(N,N)
121    real(dp), intent(out)      :: TabNew(N,N)
122    integer,  intent(in)               :: N
123    integer i, j
124    !$omp parallel
125    !$omp do
126    do j = 2, N-1
127       do i = 2, N-1
128          TabNew(i, j) = 0.25 * (Tab(i, j+1) + Tab(i, j-1) + Tab(i+1, j) + Tab(i-1, j))
129       end do
130    end do
131    !$omp end do
132    !$omp end parallel
133  end subroutine jacobi_openmp_nested
```

Hypre - OpenMP

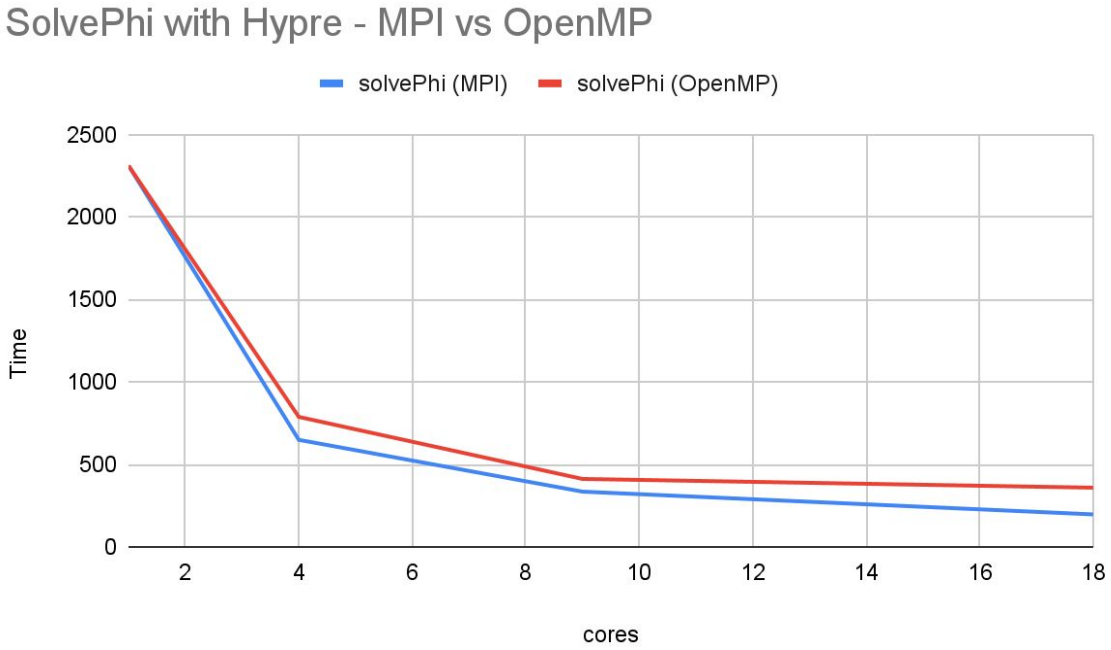
- First promising tests with Soledge3X:
  - To use Hypre with OpenMP, just set OMP_NESTED option
  - OpenMP parallel regions where Hypre routines are called, OMP MASTER is required
  - To have a first estimation of performance, OMP MASTER clause has been added in Soledge3X in some of these regions
- In next slides, a timing of the following region in Soledge3X is displayed (named *solvePhi with Hypre*):

```
!$OMP MASTER
!BoomerAMG: Because we are using a ParCSR solver, we need to get the object of the matrix and vectors to pass in to the ParCSR solvers

call HYPRE_SStructMatrixGetObject(hypreA, parA, ierr)
call HYPRE_SStructVectorGetObject(hypreb, parb, ierr)
call HYPRE_SStructVectorGetObject(hyprex, parx, ierr
call HYPRE_ParCSRBiCGSTABSetup(solver, parA, parb, parx, ierr)
call HYPRE_ParCSRBiCGSTABSolve(solver, parA, parb, parx, ierr)
!$OMP END MASTER
```

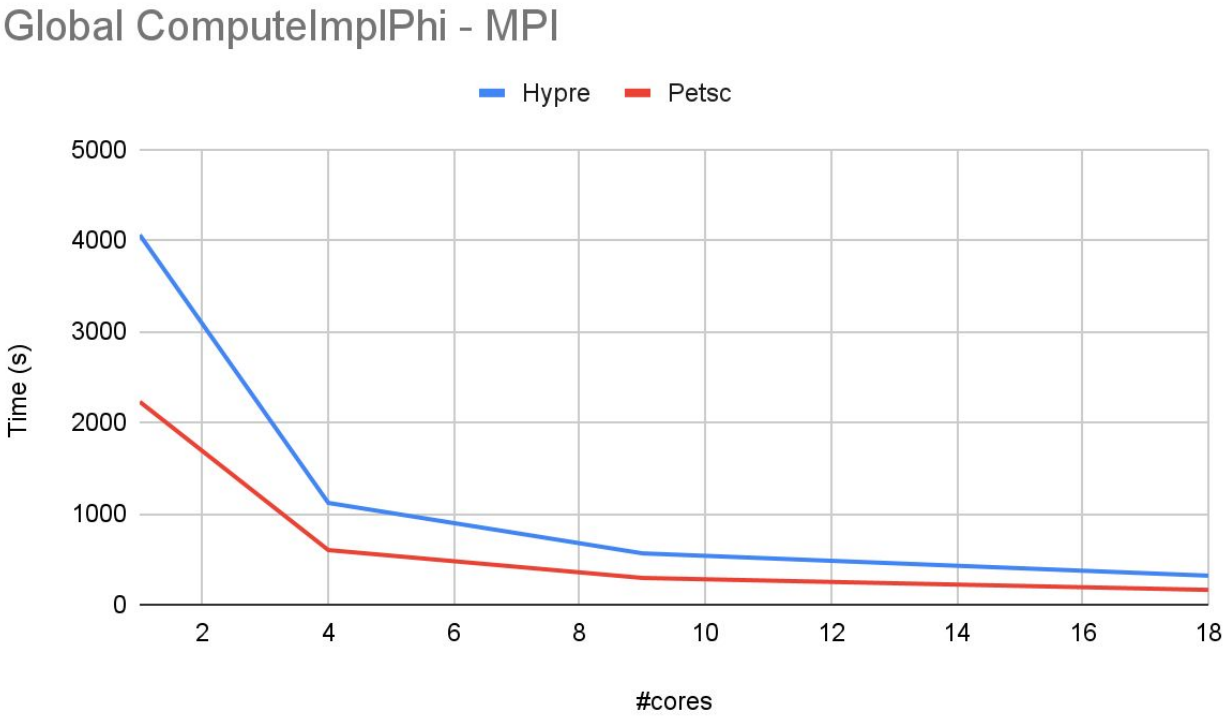
Hypre - OpenMP

- Time To Solution for solvePhi with Hypre (BiCGSTAB + BoomerAMG precond)



Hypre - MPI

- Time To Solution for Vorticity solver with Hypre (with matrix building)

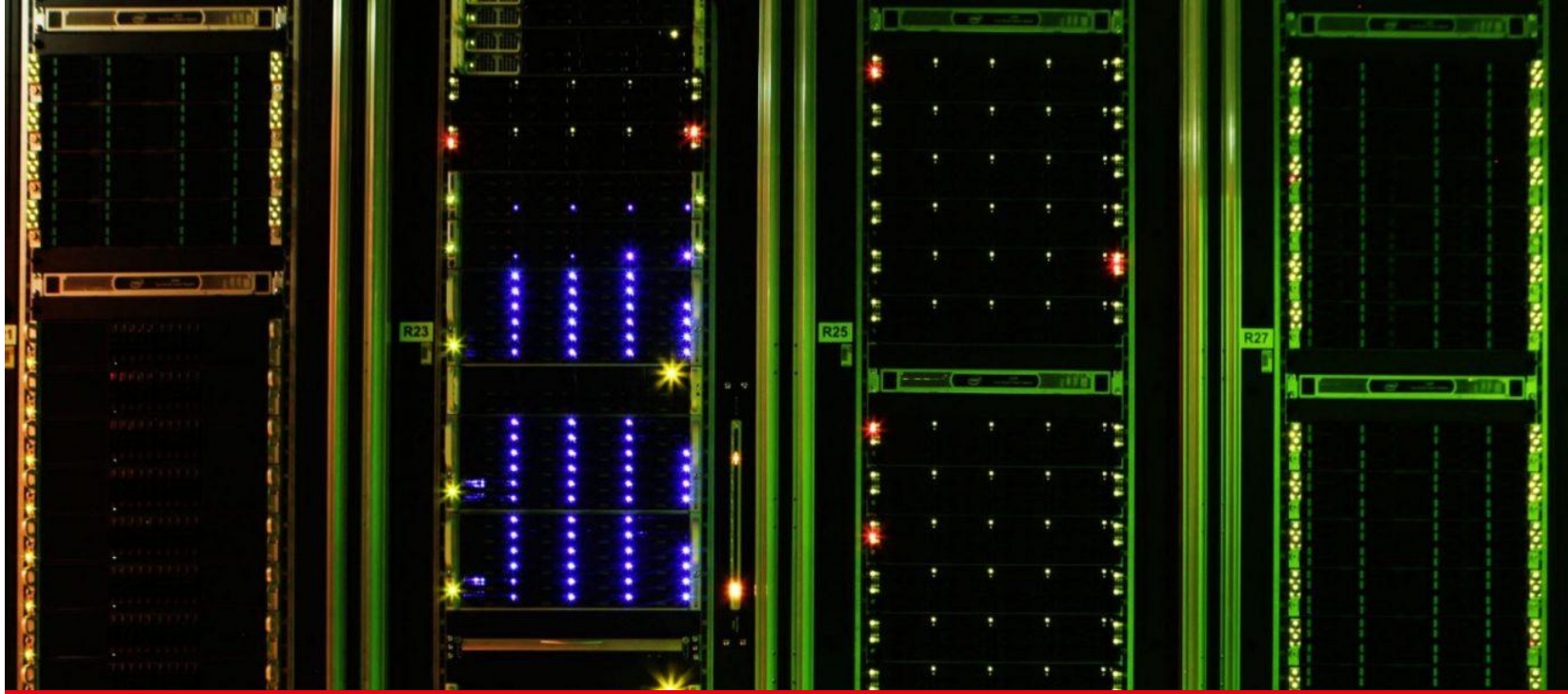


Hypre

- HYPRE
  - To use Openmp with Hypre in Soledge3X:
    - export OMP_NESTED="TRUE"
    - use OMP MASTER rather than single in OMP regions calling Hypre
    - replace OMP DO by OMP SINGLE in regions calling hypre:
      --> to do: need to refactor theses regions to exploit threads

--> Therefore some of these OpenMP // regions have to be revisited to re-introduce OpenMP work sharing by putting outside OpenMP MASTER some work

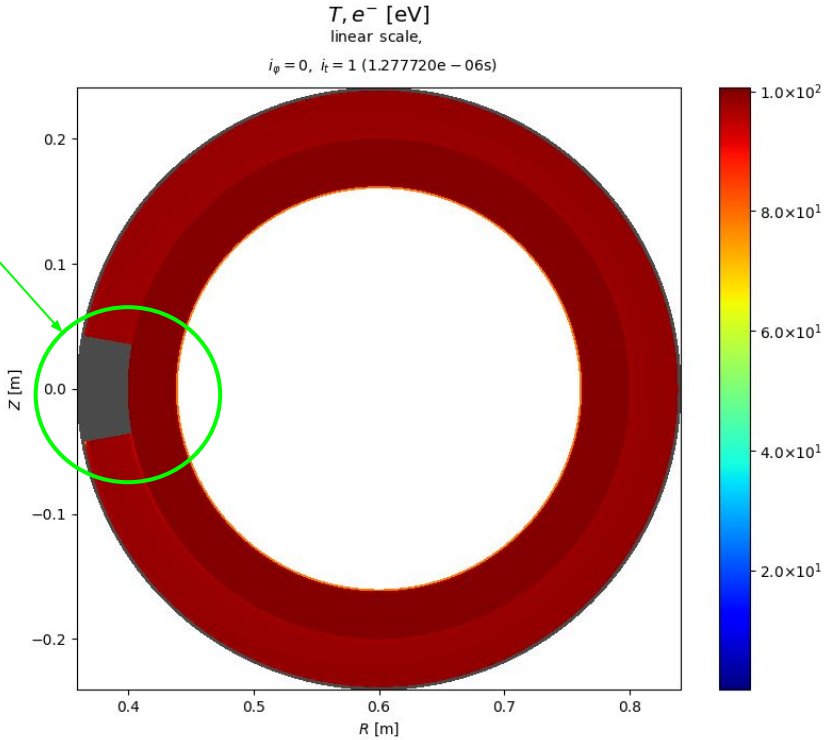
EPFL



MPI Load Balancing in Soledge3X

LoadBalancing

- Presence of a wall in usual configurations
- Cells in the wall are treated using a mask
- Currently, try to get same number of cells per MPI process
- Implicit Solvers don't solve cells in the wall
  —> can lead to a non-optimal load balancing between MPI processes
- New development to improve MPI load balancing taking into account the mask
- A weight factor is introduced for each cell with a value:
  - = 1 for a cell outside the wall
  - < 1 for a cell in the wall
- The new MPI decomposition takes into account theses cell weights to share workload between MPI processes



LoadBalancing

- Performance results for Circle test-case

180 degres - 32x256x64 -10it					
3 MPI					
	MainLoop	ImplPhi	ImplE	ImplG	Expl
Initial version	105	32	24	25	18
new version(coef=0.01)	95	25	18	20	20

270 degres - 9x1024x128-10it					
21MPI					
	MainLoop	ImplPhi	ImplE	ImplG	Expl
Initial version	54	13	11	11	13
new version(coef=0.0001)	64	10	7	8	21

LoadBalancing

- Circle test-case: *ScoreP* analysis
  - installation of *ScoreP.7.0* with intel toolchain
  - *export PATH=~/profiling/scalasca_intel/scorep-7.0/ScoreP-7.0/bin:${PATH}*
  - compilation with *scorep mpiifort -O2 -qopenmp …*

- ScoreP allows to analyse:

  - Communication efficiency (*maximum across all processes of the ratio between useful computation time and total run-time*):

    CommE = maximum across processes (ComputationTime / TotalRuntime) = 0.95

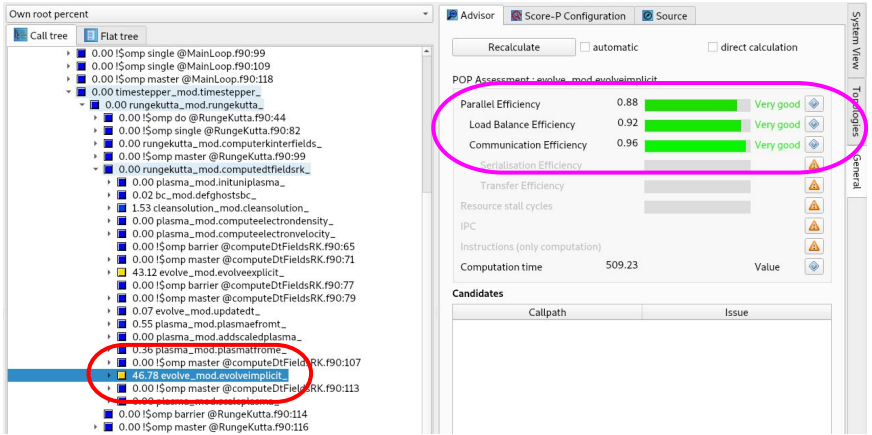
  - Load balance efficiency (*ratio between average useful computation time - across all processes - and maximum useful computation time - also across all processes -* :

    LB=avg(ComputationTime) / max(ComputationTime) = 0.66

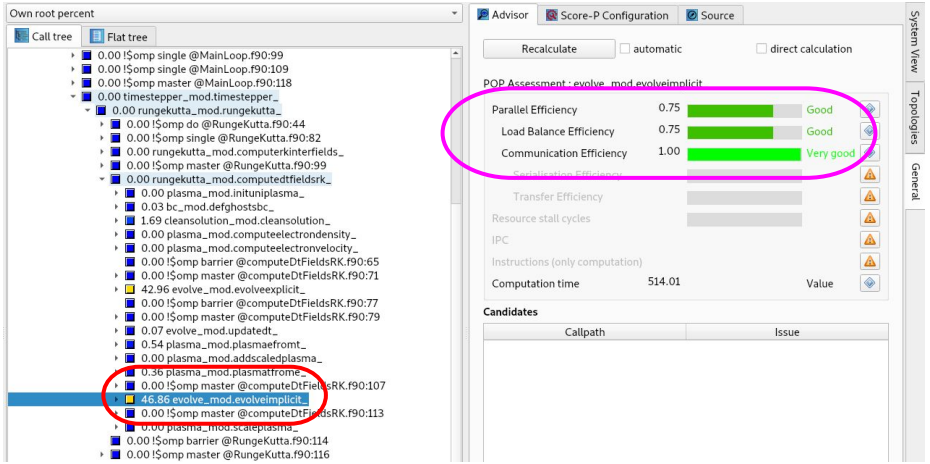
Profiling with Scorep

- Load balancing in *Implicit* & *Explicit* modules

Implicit module
New version - coef=0.01)



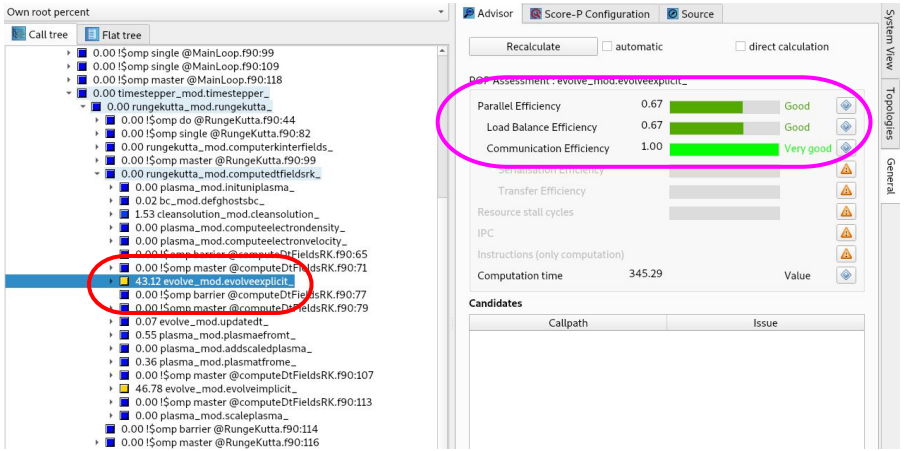
Implicit module
(Initial Version)



Profiling with Scorep

- Load balancing in *Implicit* & *Explicit* modules

Explicit module
New version - coef=0.01)



Explicit module
(Initial Version)