TSVV - ACH meeting Feb 2023

Nicola Varini

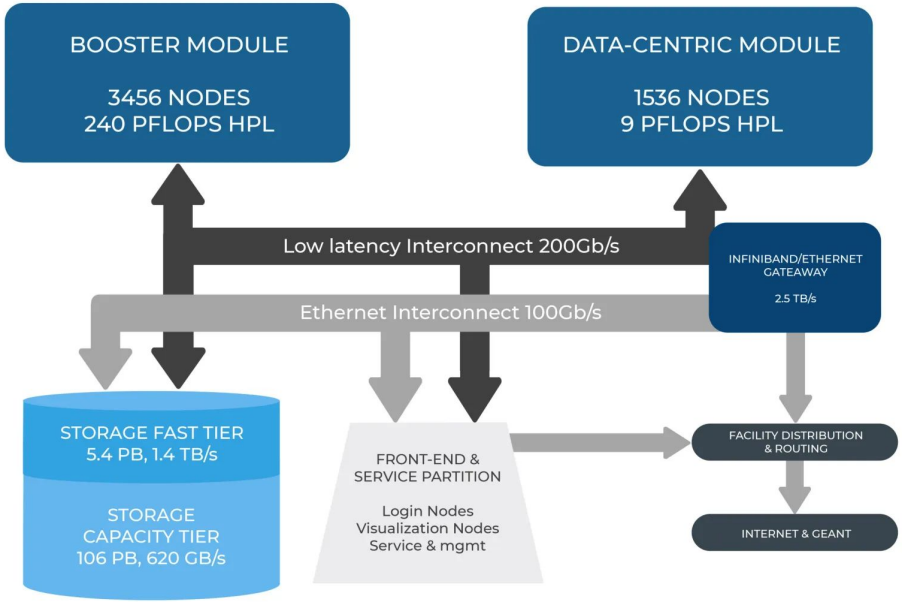
EUROfusion

EPFL

ACH 2023 goals

- Marconi and Marconi100 will be decommissioned.
- Leonardo will be the replacement.
- Ideally all the codes should be ported to NVIDIA GPUs.
- Computational patterns:
  - Solver: PETSc offers different flavors of **GMRES** for NVIDIA GPUs.
  - Preconditioner: Through PETSC we can have **HYPRE**, **AMGX**, **GAMG**.
  - RHS: Stencil operations
    - CUDA - requires explicit coding but it is reliable
    - OpenMP/OpenACC offload - strong compiler dependency
  - MPI: if data are copied to/from CPU/GPU will be very slow.
    - Technical point: It’s important to use CUDA-aware MPI and RDMA.

Running on Leonardo Booster



Features a **custom BullSequana X2135 “Da Vinci” blade**, composed of:

- 1 x CPU Intel Xeon 8358 32 cores, 2,6 GHz
- 512 (8 x 64) GB RAM DDR4 3200 MHz
- 4 x NVidia custom Ampere GPU 64GB HBM2
- 2 x NVidia HDR 2×100 Gb/s cards

- **Network:** 2xNvidia HDR cards 2x100Gb/s

EuroHPC regular access

System	Architecture	Site (Country)	Total Core Hours	Minimum request core hours
Vega CPU	BullSequana XH2000	IZUM Maribor (SI)	75 million	10 million
Vega GPU	BullSequana XH2000	IZUM Maribor (SI)	1.5 million	0.5 million
MeluXina CPU	BullSequana XH2000	LuxProvide (LU)	65.5 million	10 million
MeluXina GPU	BullSequana XH2000	LuxProvide (LU)	11.1 million	2 million
Karolina CPU	HPE Apollo 2000Gen10 Plus and HPE Apollo 6500	VSB-TUO, IT4Innovations, (CZ)	60 million	10 million
Karolina GPU	HPE Apollo 2000Gen10 Plus and HPE Apollo 6500	VSB-TUO, IT4Innovations, (CZ)	6 million	1 million
Discoverer CPU	BullSequana XH2000	Sofiatech, (BG)	104 million	10 million

- EuroHPC replaced PRACE
- Either NVIDIA GPU or AMD based processors

GRILLIX - recent progress in HPC development

- Scaling
- Poisson solver

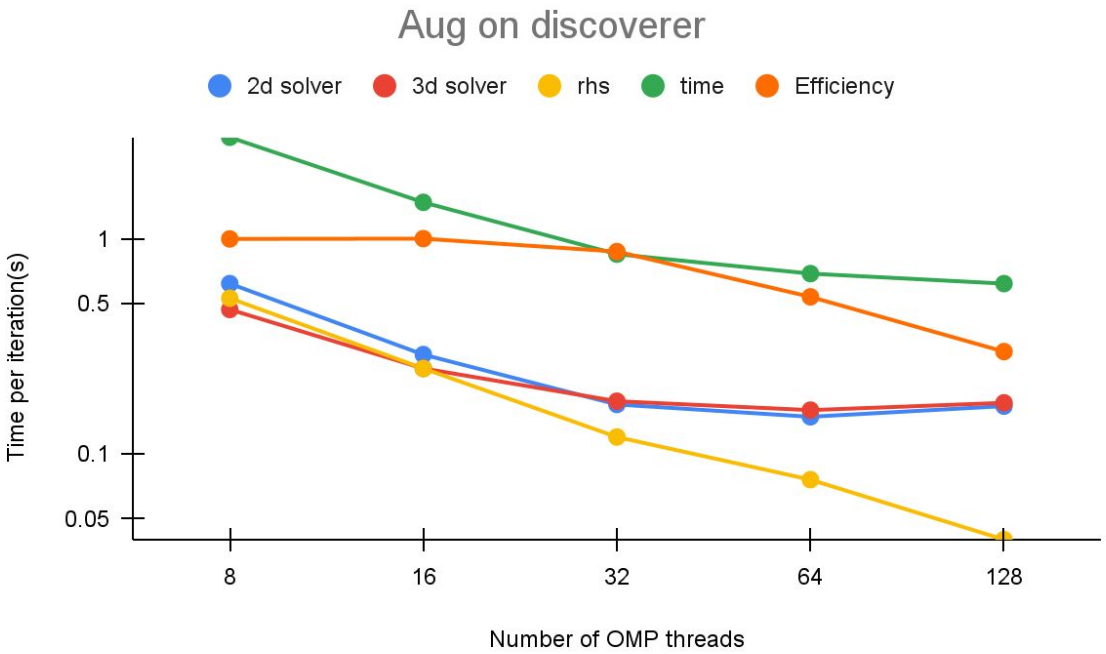
GRILLIX - STATUS

- Run wells up to 16 nodes on Intel architectures.
- No working GPU implementation.
- The CMAKE need to be refactored.
  - C++ optional dependency(done by ACH).
  - Hardcoded MKL dependency(problem on Marconi100)
- Solvers: PARALLAX GMRES, PETSc(ACH), DIRECT

GRILLIX - GOALS 2023

- Introduce ENABLE_CXX as CMAKE option - Done
- Benchmarks on AMD based supercomputers.
- Runs on GPU supercomputers(Leonardo):
  - RHS: stencil ported to GPU with CUDA(done in GBS)
  - Solver(PETSc GPU or parallax).

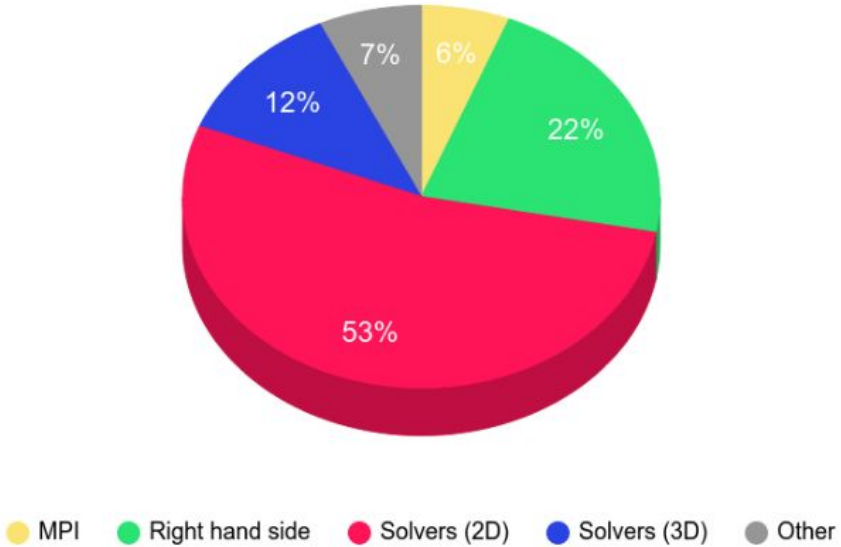
Benchmark on AMD CPU - discoverer

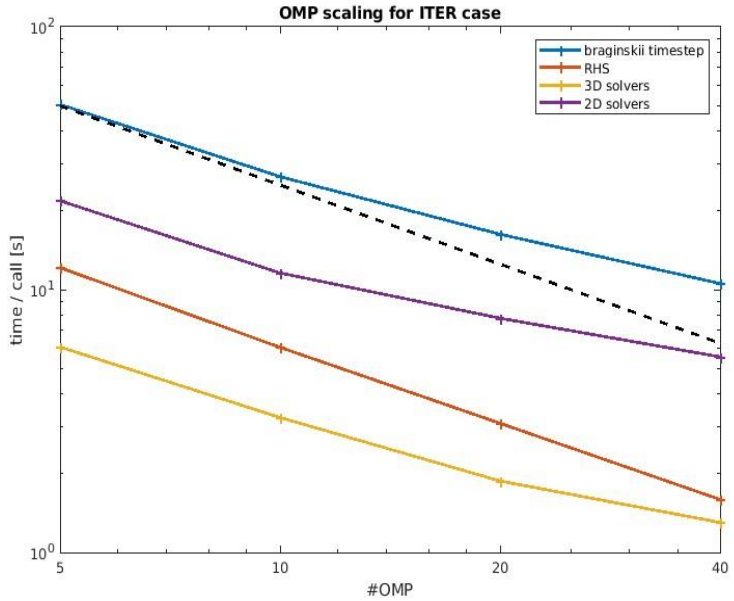


- Discoverer is one of the supercomputer available through EuroHPC.
- Dual 64 AMD Epyc
- 1128 nodes

- Good scalability up to 32 cores.
- 2d and 3d solver limit the scalability

Profiling results - ITER on Intel CPU





The 2D solver is the main bottleneck

- Perfect scaling of RHS
- Non-ideal scaling especially for 2D elliptic solver
- Some performance gain even across socket (20 → 40)

GRILLIX - Comparison of different solvers (ITER neumann)

Algorithm:	time per solve [s]
MGMRES (MKL on coarsest) on 20 cores	7.36
DIRECT (MKL)	62.19
DIRECT (CUSPARSE), gpu:v100:2	> 900
PETSc/HYPRE AMGX	not converging
PETSC (gmres + mg)	converges in miniapp, if matrices provided on each level

- For Neumann BC AMG preconditioner do not converge.
- Direct solver is slow bot on CPU and GPU.
- It is necessary to provide restriction, prolongation, elliptic matrices to make gmres converge.

GRILLIX - Comparison of different solvers (ITER dirichlet)

Algorithm:	time per solve [s]
MGMRES (MKL on coarsest)	3.99
DIRECT (MKL)	72.80
PETSC -ksp_type gmres -pc_type hypre	57.26
AMGX (Miniapp on Marconi v100 GPU)	1.08

- For Dirichlet BC AMGX is fast!

Parallax + PETSc for Numann BC - python miniapp

```
pc1.setType("mg")
pc1.setFromOptions()
pc1.setMGLevels(nlevels)

for i in range(1,nlevels):
    pc1.setMGInterpolation(i,mat_interpolation[i-1])
    pc1.setMGRestriction(i,mat_restriction[i-1])
    local_ksp = pc1.getMGSmoother(i)
    local_ksp.setOperators(mat_level[i],mat_level[i])
    local_ksp.setUp()

ksp1.solve(b, guess)
```

Multigrid preconditioner - requires you provide additional information about the coarser grid matrices and restriction/interpolation operators.

Necessary to explicitly set the operators at each level

petscrc

```
-ksp_initial_guess_nonzero true
-ksp_rtol 1e-8
-ksp_norm_type unpreconditioned
-ksp_monitor_true_residual
-pc_mg_galerkin both
```

```
aug  python3.10 solve.py
 0 KSP unpreconditioned resid norm 3.610986680466e-07 true resid norm 3.610986680466e-07 ||r(i)||/||b|| 2.555560576964e-08
 1 KSP unpreconditioned resid norm 3.610417719737e-07 true resid norm 3.610417595971e-07 ||r(i)||/||b|| 2.555157825575e-08
 2 KSP unpreconditioned resid norm 3.609282526214e-07 true resid norm 3.609281884341e-07 ||r(i)||/||b|| 2.554354061916e-08
 3 KSP unpreconditioned resid norm 3.608856947201e-07 true resid norm 3.608856529350e-07 ||r(i)||/||b|| 2.554053030496e-08
 4 KSP unpreconditioned resid norm 3.549461025393e-07 true resid norm 3.549460736618e-07 ||r(i)||/||b|| 2.512017553831e-08
 5 KSP unpreconditioned resid norm 2.703489769946e-07 true resid norm 2.703489176311e-07 ||r(i)||/||b|| 1.913308181557e-08
 6 KSP unpreconditioned resid norm 1.868843796150e-07 true resid norm 1.868842785487e-07 ||r(i)||/||b|| 1.322613836536e-08
 7 KSP unpreconditioned resid norm 1.802563980435e-07 true resid norm 1.802565601019e-07 ||r(i)||/||b|| 1.275708274493e-08
 8 KSP unpreconditioned resid norm 1.802127671659e-07 true resid norm 1.802130793818e-07 ||r(i)||/||b|| 1.275400553574e-08
 9 KSP unpreconditioned resid norm 1.801540438828e-07 true resid norm 1.801542401071e-07 ||r(i)||/||b|| 1.274984137386e-08
10 KSP unpreconditioned resid norm 1.780247312436e-07 true resid norm 1.780247451981e-07 ||r(i)||/||b|| 1.259913316805e-08
11 KSP unpreconditioned resid norm 1.600517875415e-07 true resid norm 1.600516244517e-07 ||r(i)||/||b|| 1.132714290918e-08
12 KSP unpreconditioned resid norm 1.477388465260e-07 true resid norm 1.477387546203e-07 ||r(i)||/||b|| 1.045573884389e-08
13 KSP unpreconditioned resid norm 1.466611242851e-07 true resid norm 1.466609434732e-07 ||r(i)||/||b|| 1.037946019984e-08
14 KSP unpreconditioned resid norm 1.462808160648e-07 true resid norm 1.462807077803e-07 ||r(i)||/||b|| 1.035255023221e-08
15 KSP unpreconditioned resid norm 1.441139387650e-07 true resid norm 1.441141329897e-07 ||r(i)||/||b|| 1.019921781612e-08
16 KSP unpreconditioned resid norm 1.401376138544e-07 true resid norm 1.401380844258e-07 ||r(i)||/||b|| 9.917825668737e-09
```

Next steps:

- Implementation in parallax
- Parameter tuning

Parallax + PETSc for Neumann BC

Numerical and algorithmic tools for Flux-Coordinate Independent approach
(Equilibra, meshes, solvers, operators)

- Parallax generates the multigrid data.
- The elliptic matrices at each fine grid will be passed to PETSc.
- We can then use the GPU implementation available in PETSc for the solvers.
- This will allow runs on Leonardo.

Running PETSc on GPU

- The following steps will allow to run PETSc on GPU
- PETSc configuration
  - **NVIDIA GPU** ./configure **--with-cuda --download-hypre –download-amgx --download-hypre-arguments='--enable-unified-memory'**
  - **AMD GPU** ./configure **--with-hip --download-hypre --download-hypre-arguments='--enable-unified-memory'**
- Petscrc:
  - -ksp_type gmres
  - -pc_type hypre, amgx
  - -mat_type aijcusparse
  - -vec_type cuda
- Downside: if hypre is compiled on GPU it cannot run on CPU, it has to be recompiled.

Solvers - GRILLIX

- Neumann BC:
  - the PARALLAX implementation works well on CPU but NO GPU porting available(yet) for production runs.
  - Next: implement the python miniapp in Fortran/PARALLAX.
- Dirichlet BC:
  - AMGX gives good performance.
  - AMGX is now available through PETSc.
- Goals for 2023:
  - Allow PARALLAX to run on many architectures.
  - Improve the current PETSc implementation of PARALLAX to allow runs with Neumann BC.

GBS - recent progress in HPC development

- Poisson solver
  - AMGX vs PETSC on GPU

- Scaling on Marcon100

GBS - Solver current status

- Let's consider 16 poloidal planes of TCV@0.9T to fit on 1 GPU

Architecture	Solver Time(s)	MPI	Mode	Aggressive Coarse	% TTS
SkyLake	15.8	36	single	y	20
Icy Lake	8.9	72	single	y	41
1 V100	21.39	1	many	y	60
1 V100	CRASH	1	many	n	
1 V100	55.97	1	single	y	74
1 V100	47.15	1	single	n	61
1 A100	57.2	1	single	n	68

- Times measured from 10 steps of GBS simulations.
- CPU -> PETSc/HYPRE
- GPU -> AMGX
- Mode many: 16 planes assembled in one matrix, rhs and guesses. Solved together
- Mode single: one solve per poloidal plane. 16 in total.

- At TCV scale the CPU is faster. Why?
- On V100 many/aggressive_corse gives a 2.5x performance boost
- On A100 the aggressive_coarse is bugged (github issue).

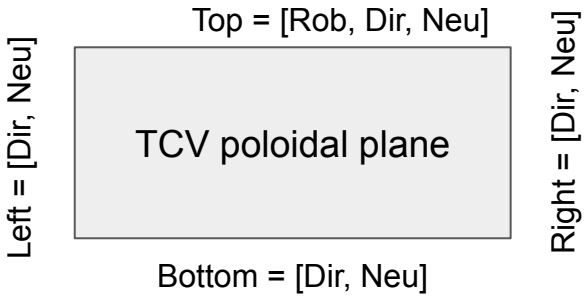
Solvers - GBS

- The elliptic solver is the main bottleneck for all the edge codes.
- **Assumption:** the solution of the different poloidal planes will take approximately the same number of iterations.
- **Miniapp for solver:**
  - It loads matrix, rhs and guess and solve the system with the external libraries(AMGX, PETSc).
  - It can be tested on many different architectures.
  - Quick exploration of different parameters.
- Questions:
  - Current status
  - What are the differences between CPU and GPUs?
  - What is the impact of the boundary conditions?

HYPRE BoomerAMG paramters(in Bold the GPU supported options)

- CycleType = {"", "V", "W"};
- CoarsenType = {"CLJP", "Ruge-Stueben", "", "modifiedRuge-Stueben", "", "", "Falgout", "", "**PMIS**", "", "HMIS"};
- Smoother = {"**Jacobi**", "sequential-Gauss-Seidel", "seqboundary-Gauss-Seidel", "**SOR/Jacobi**", "**backward-SOR/Jacobi**",, "symmetric-SOR/Jacobi", "l1scaled-SOR/Jacobi", "Gaussian-elimination", "**l1-Gauss-Seidel**" , "**backward-l1-Gauss-Seidel**" , "CG" , "Chebyshev", "FCF-Jacobi", "**l1scaled-Jacobi**"};
- Interpolation = {"classical", "**direct**", "multipass", "multipass-wts", "**ext+i**", "ext+i-cc", "standard", "standard-wts", "block", "block-wtd", "FF", "FF1", "**ext**", "ad-wts", "ext-mm", "ext+i-mm", "**ext+e-mm**", “**ad-wts**”};
- **Aggressive coarsening**

Poisson solver - GBS - miniapp on GPU with AMGX



- 1 solve - 8 planes
- Let's test different BC, 3 cases:
  - Bottom, Left = Neu Right,Top = Dir
  - Bottom, Top, Right, Left = Dir
- Presweep, postsweep = 4
- V cycle
- FGMRES

- **For complex BC the number of iterations increase.**
- **Aggressive coarsening is important**

GPU type	#planes	BC type	#iterations AMGX	solver mode	aggressive coarsening	time(s)
A100	8	NeuNeuDirDir	90	many	no	1.13
A100	1	NeuNeuDirDir	25	single	no	0.11
V100	**8**	**NeuNeuDirDir**	**57**	**many**	**2**	**0.45**
V100	1	NeuNeuDirDir	25	single	no	0.11
V100	*8*	*NeuNeuDirDir*	*90*	*many*	*no*	*1.59*
A100	8	DirDirDirDir	4	many	no	0.12
A100	1	DirDirDirDir	4	single	no	0.04
V100	8	DirDirDirDir	21	many	2	0.3
V100	1	DirDirDirDir	4	single	no	0.04
V100	8	DirDirDirDir	4	many	no	0.14

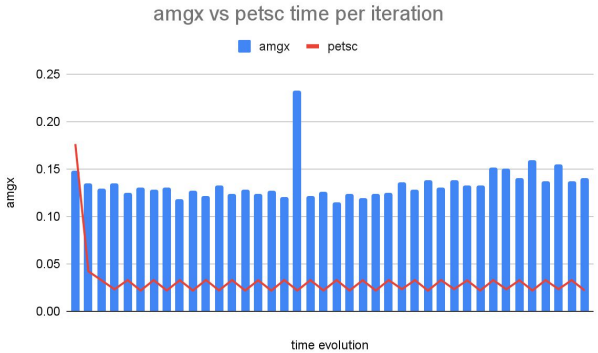
Poisson solver - GBS - miniapp on GPU with PETSc

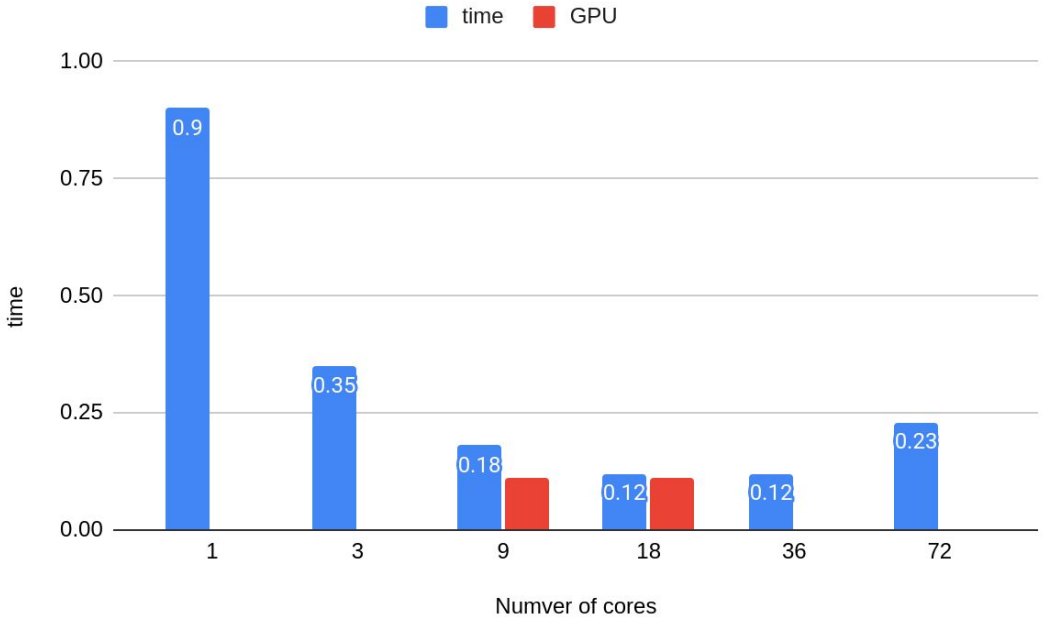
GPU type	#planes	BC type	#iterations	solver mode	aggressive coarsening	time(s)
A100	8	MagMagTarTar	112	many	0	0.97
A100	8	MagMagTarTar	48	many	1	0.29
A100	8	MagMagTarTar	120	many	2	0.7
A100	1	MagMagTarTar	55	single	0	0.16
A100	8	MagMagTarTar	22	single	1	0.07
A100	1	MagMagTarTar	27	single	2	0.1

- Aggressive coarsening doesn’t work for AMGX on A100.
- HYPRE on GPU through PETSc is good.
- Aggressive coarsening is important, especially when we solve many planes at once.

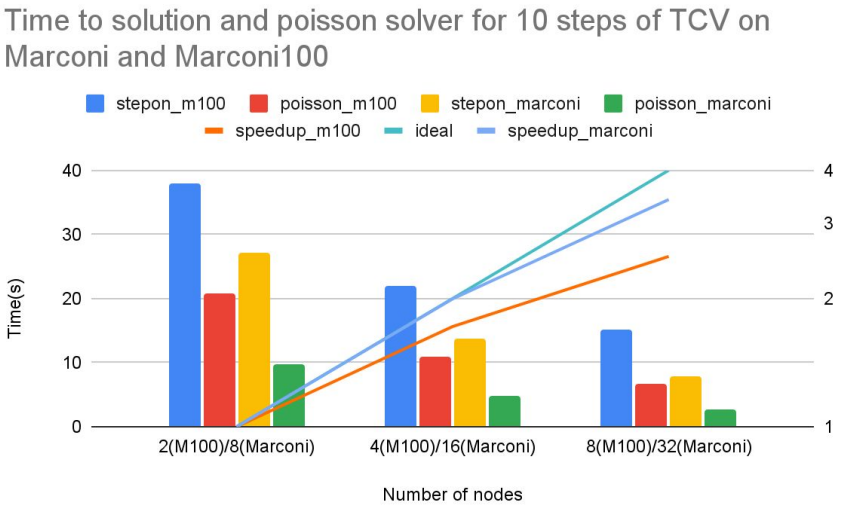
Poisson solver - GBS - comparison CPU GPU 1 plane - miniapp drawback

- The solver scales up to 18 cores on the CPU.
- The performance of the solver on 1 iterations is comparable.
- Somehow PETSc is more efficient after the first iteration.





GBS - TCV 0.9T



- At 0.9 T the runs on Marconi are faster compared to Marconi100.
- The strong scaling curves are qualitatively similar.
- The strong scaling is better on Marconi.

GBS - TCV 1.45T

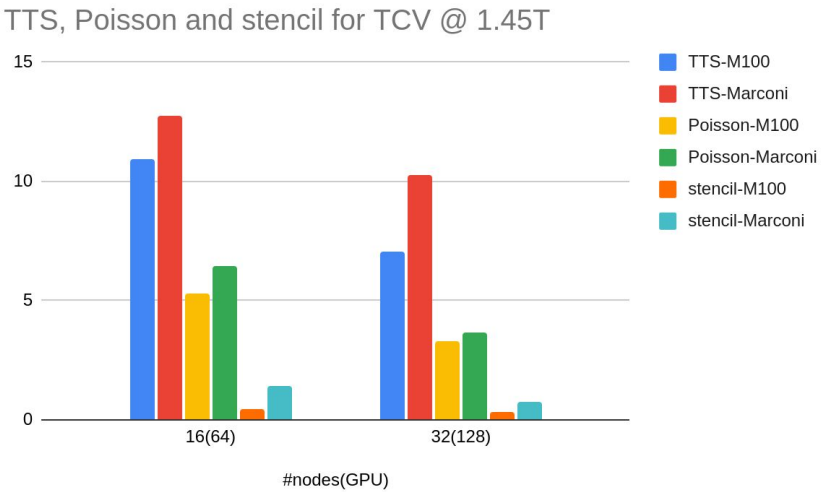
Marconi 100

#nodes(GPU)	TTS	Poisson	stencil
8(32)	832.97	63.39	127.83
16(64)	10.96	5.27	0.48
32(128)	7.08	3.28	0.35

Marconi

#nodes	TTS	Poisson	stencil
8	62.92	27.31	10.34
16	32.68	14.25	5.33
32	18.14	7.33	2.7
64	12.73	6.46	1.44
128	10.27	3.66	0.74

- On 8 nodes the GPU memory is not sufficient.
- The stencil operations are fast on GPU.
- The solver on GPU is competitive.



Running on Leonardo

- Solver:
  - GBS and Soledge rely on external libraries:
    - Pros: Existing GPU implementation
    - Cons: Not all the parameters are ported to GPU
  - Grillix:
    - AMG preconditioner available in parallax: GPU porting in progress.
    - PETSc implementation in progress.
  - Feltor:
    - In-house solver implementation.
- RHS:
  - Using CUDA will guarantee good performance for the stencil operations.
  - Offload directives are effective as long as the compiler works properly.