This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 - EUROfusion). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

Simple and portable methods to extract performance metrics from software running on HPC clusters

F. Cipolletta

BSC-ACH

12/02/2026

EUROfusion E-TASC General Meeting #2

Preliminary warnings

- The methods presented here have **several** (even better) **alternatives**
- The focus is on what has been used within the BSC-ACH for work related to CARIDDI
- Some **details** and **issues** to be faced to port among different hardware will be discussed

Outline

- **Context**
  - JOREK couplings with STARWALL and CARIDDI
  - The generalized eigenvalue problem
  - Problems, ideas, and possibilities
- **Solutions adopted**
  - Time of computation
  - Parallelization efficiency
  - Memory usage and overhead
  - Notes on portability (in each aspects)
- **Conclusions**
  - Summary, takeaways and perspectives

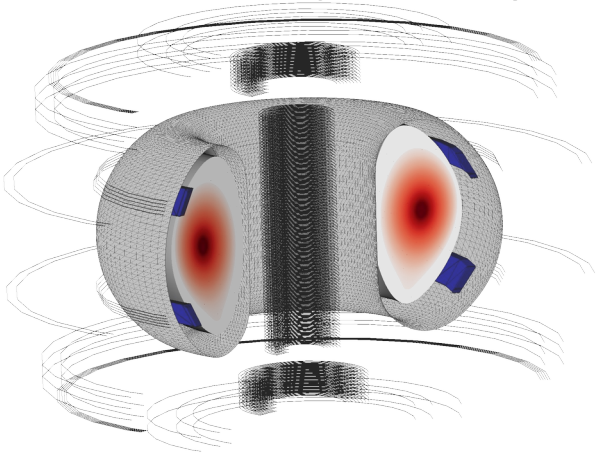
20 BSC YEARS

Context

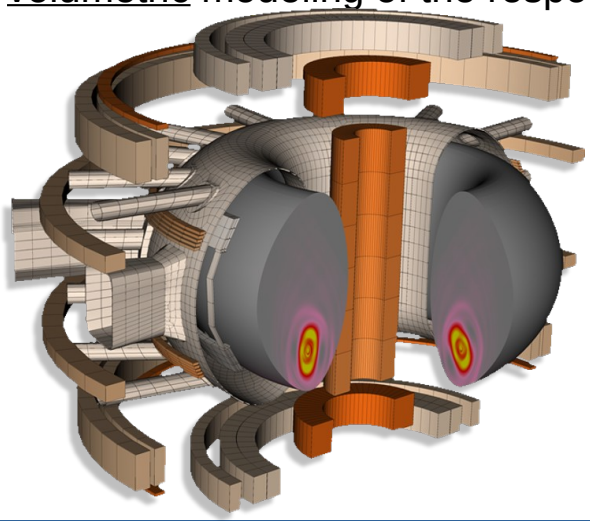
JOREK couplings with STARWALL and CARIDDI

- The *free-boundary and resistive wall extension* considers the **interactions** between the conducting structures and the plasma
- Calculations are done via coupling of JOREK to STARWALL (**J-S**) or CARIDDI (**J-C**) by means of response matrices
- Those couplings consist of memory-intensive works to be performed, directly related to the accuracy adopted in 3D modeling
- Representing "big" geometries (like ITER) with high accuracy imposes restrictive memory requirements
- Matrix compression have revealed to be only **partially effective** (see Cipolletta et al, 2024)

Geometry used in **STARWALL** adopting a 3D thin wall modeling of the response



Geometry used in **CARIDDI** adopting a 3D volumetric modeling of the response



The generalized eigenvalue problem

- Both STARWALL and CARIDDI rely on the solution of a generalized eigenvalue problem (**gEP**)
- The **3D passive structures** (STARWALL/CARIDDI) are interfaced with the **plasma** (JOREK)
- JOREK adopts a 2D Bezier discretization on the poloidal cut and a Fourier expansion on the toroidal direction
- STARWALL/CARIDDI instead models the walls in full 3D

$$L_w^* \, S_\lambda = \lambda \, R_w \, S_\lambda \,,$$

L_w^* → Inductance (**dense**) matrix for the conductive structures

S → Basis of eigenvectors

R_w → Resistance (**dense**) matrix

λ → Eigenvalue

- The solution of the problem makes the system of equations **diagonal**
- Both the inductance and resistance matrices have a **size of the walls' DoF squared**
- No cut can be done, because a **complete basis of eigenvectors** is needed

See Isernia et al, 2023 for further details

Problems and ideas

- The gEP is **challenging** when considering **ITER-size** geometry, due to time of computation and memory
- The request was to evaluate **alternative solvers** for the gEP:
  - Test new solvers and try to reach the maximum treatable number of DoF
  - Test also the capabilities to exploit GPUs for the calculation
  - Evaluate the performance in terms of **Time** and **Memory Overhead**
- CARIDDI is a big and complex code → it is worth performing the **assessment outside**, e.g. via a toy code

Possibilities

1. ScaLAPACK
   - This is currently used in STARWALL/CARIDDI
   - The implementation starts to be a bit old (mid 90s) but it is robust
   - It is still considered the **state-of-the-art reference** for parallel linear algebra on **CPUs**
2. MAGMA
   - It is developed by (an extension of) the same group of LAPACK/ScaLAPACK
   - It offers CPUs and GPUs capabilities
   - The gEP solver is not available on GPUs → **NOT AN OPTION**
3. SLATE
   - Developed by (another extension of) the same group of LAPACK/ScaLAPACK
   - It offers CPUs and GPUs capabilities
   - The gEP solver can run on multi-GPUs but within a single MPI → **NOT AN OPTION**
4. NVIDIA cuSOLVER
   - Dense SVD, factorizations, and standard eigenvalue solver
   - Multi node and multi GPU via CUDA → Possible Portability Issues
   - No direct implementation of gEP → More involved Implementation → **BETTER TO AVOID**
5. ELPA
   - Developed at MPG
   - It offers CPUs (**ELPA2** solver) and GPUs (**ELPA1** solver) capabilities
   - **The results shown ahead are obtained with this library**

Solutions Adopted

The eigenprobsolver code

1. **Love for ugly composite names**

2. Toy Fortran code to solve the gEP calling ELPA or ScaLAPACK

3. The matrices can be defined

- Randomically (prescribed dimension)
- Analytically (prescribed complexity)

→ Test implementation

- Read from STARWALL
- Read from CARIDDI

→ Results

4. Hosted at the BSC internal GitLab website (different branches for different solvers)

5. Porting between **different machines** was necessary

Tests Performed

- **STARWALL** allows scaling the problem size easily with 2 parameters, nvu and nwv

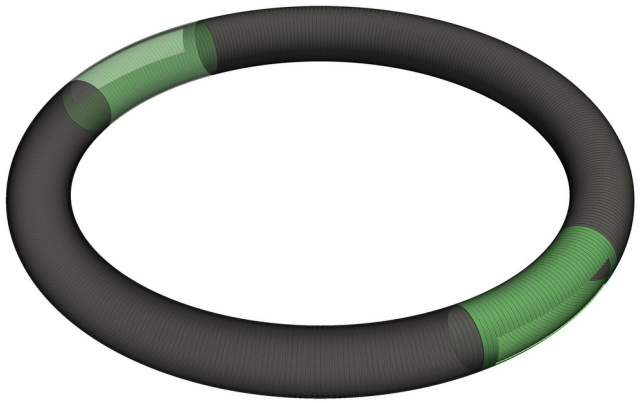
nvu x nwv	300 x 300	390 x 390	640 x 640
Matrix Size	$(90000)^2$ = 8.1 B	$(152100)^2 \approx$ 23.1 B	$(409600)^2 \approx$ 168 B

- With **CARIDDI** it is more difficult and requires explicit meshing of a geometry

NOTE: CARIDDI requires square process grids → the resources' selection respects this requirement

"CIRCULAR" test case

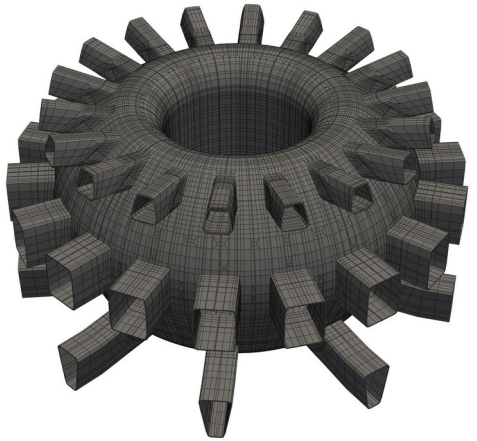
- Simple toroidal mesh
- 2 holes
- 2 walls resolutions tried
  - 71533 DoF
  - 161089 DoF



"ITER" test case

- ITER geometry
- Scaled adding components
  - 79220 DoF
  - …

...WORK IN PROGRESS...



Hardware 1 - MareNostrum5

- **MareNostrum5** for the initial steps and STARWALL tests (sudden loss of MARCONI)
- 2 partitions available:

  1. ACC

  - 80 cores per node
  - 512 GiB of RAM per node
  - 4 NVIDIA H100 (64GB) GPUs per node
  - 20 cores per GPU (prescription)

  2. GPP

  - 112 cores per node
  - 256 GiB of RAM per node (standard)
  - 1024 GiB of RAM per node (highmem)
  - Only CPUs (no GPUs)

Hardware 2 - PITAGORA

- To avoid moving too big matrices, the tests with CARIDDI ones were done on PITAGORA
- 2 partitions available:

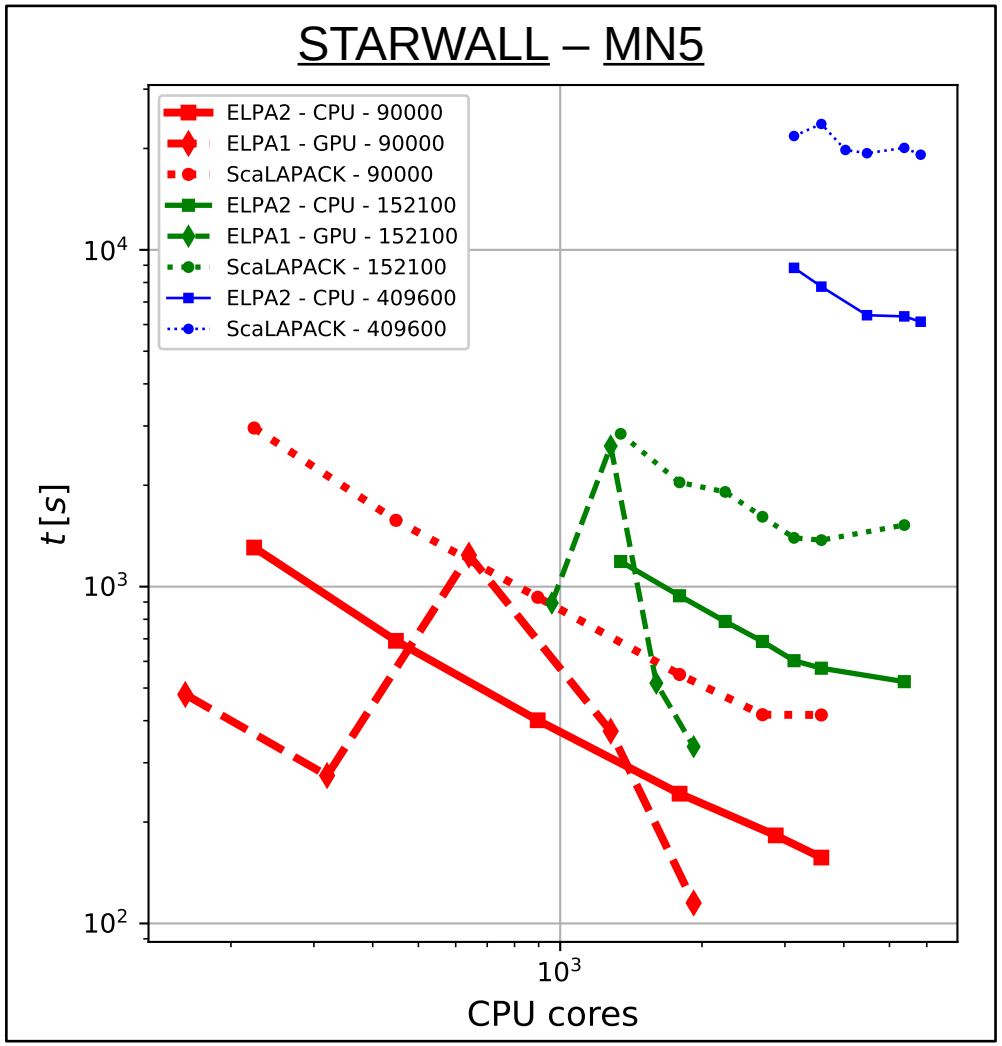
1. DCGP

- 256 cores per node
- 768 GiB of RAM per node
- Nodes based on AMD architecture

2. Booster

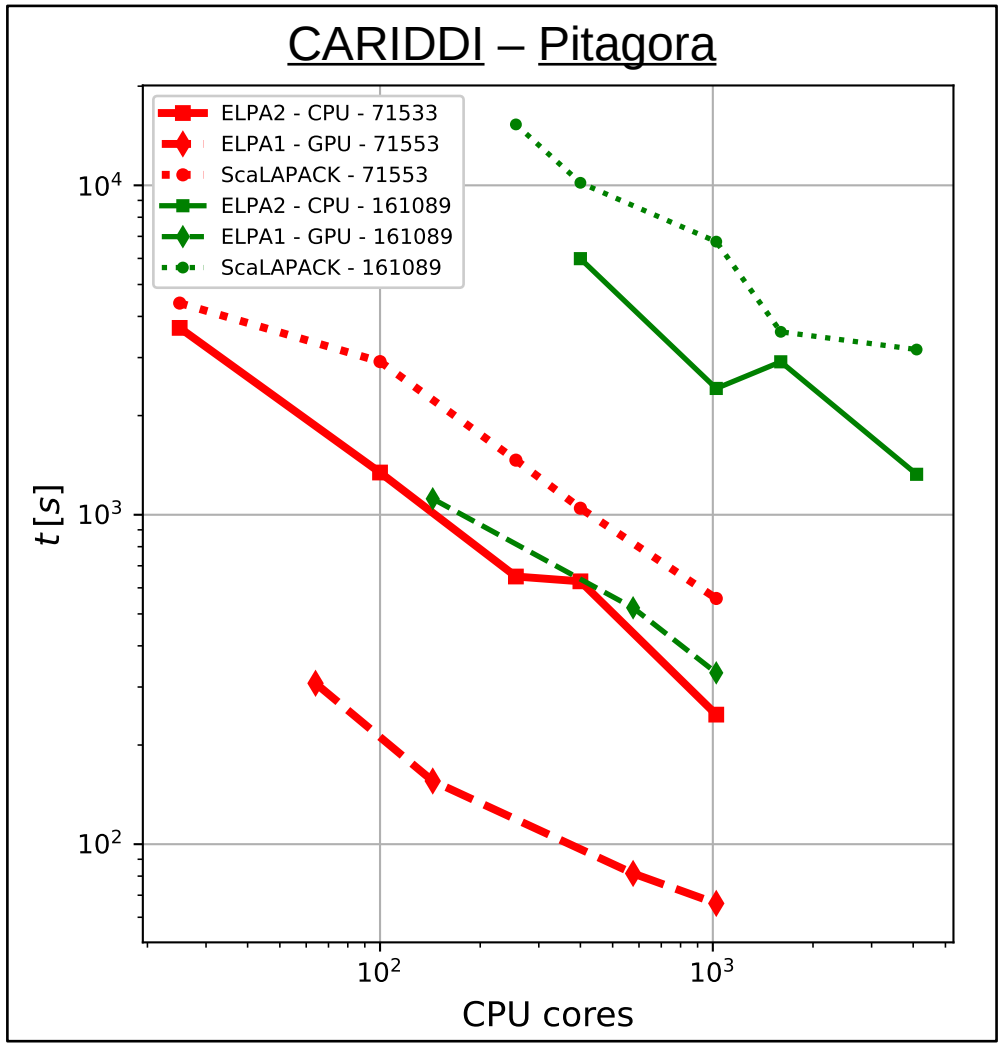
- 64 cores per node
- 512 GiB of RAM per node
- 4 NVIDIA H100 (80GB) GPUs per node
- 16 cores per GPU (to fill the node)

Computing the Execution Time



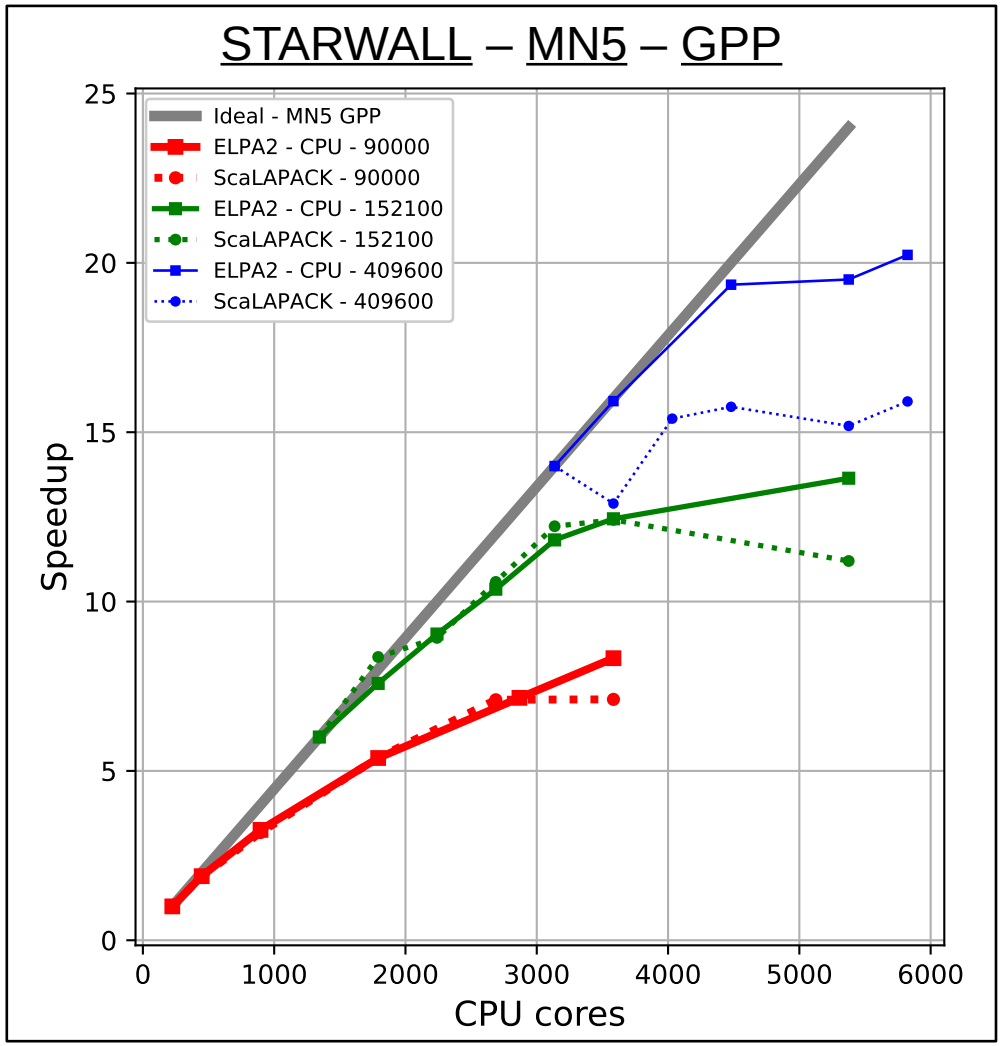
Measure the time to solve the gEP with calls to **WTIME**

The portability issues reside only in the compilation of the code because the subroutine is available in all the MPI major versions



20 YEARS BSC

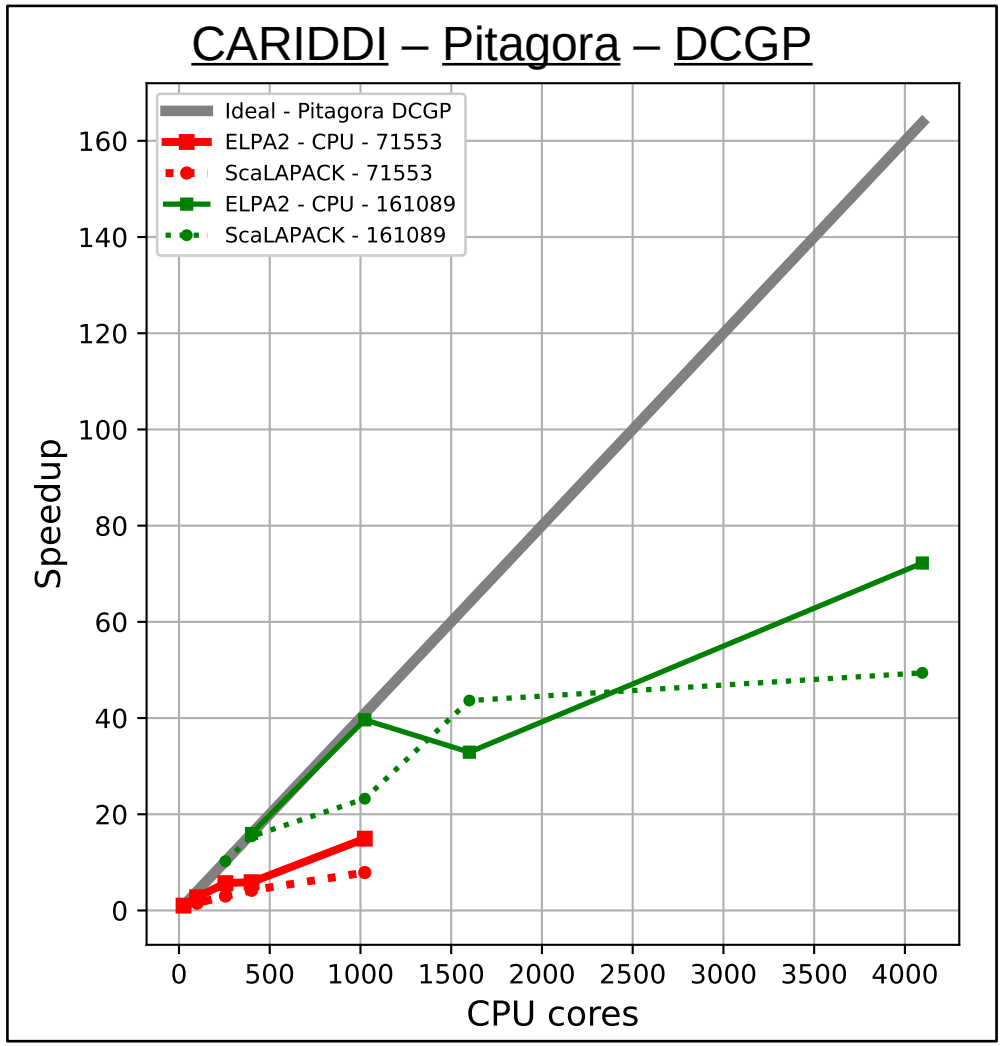
Computing the Speedup wrt CPUs



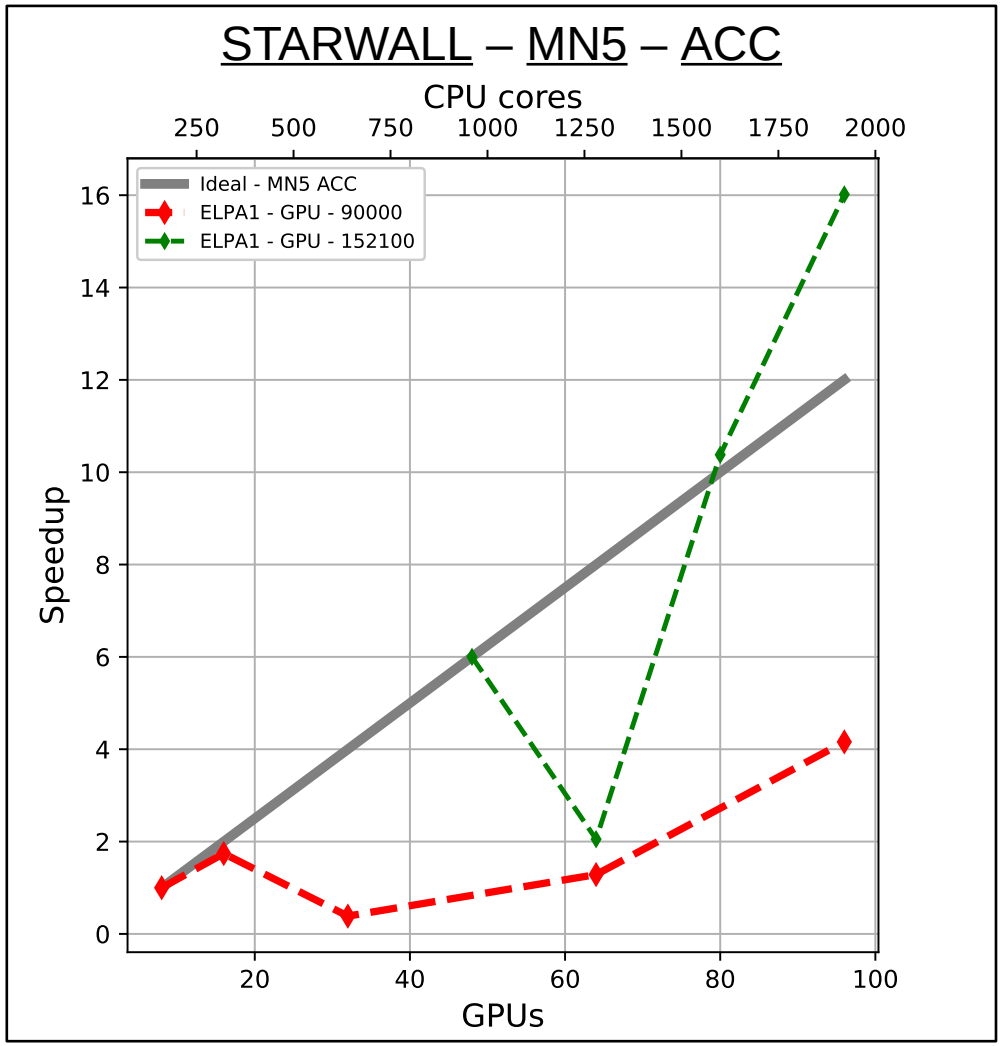
If **n** indicates the resource number

Speedup: ratio between the time with smallest resources and the time

Normalization can help the comparison: divide by the ratio between the n value of the first red point over the first n value of the curve

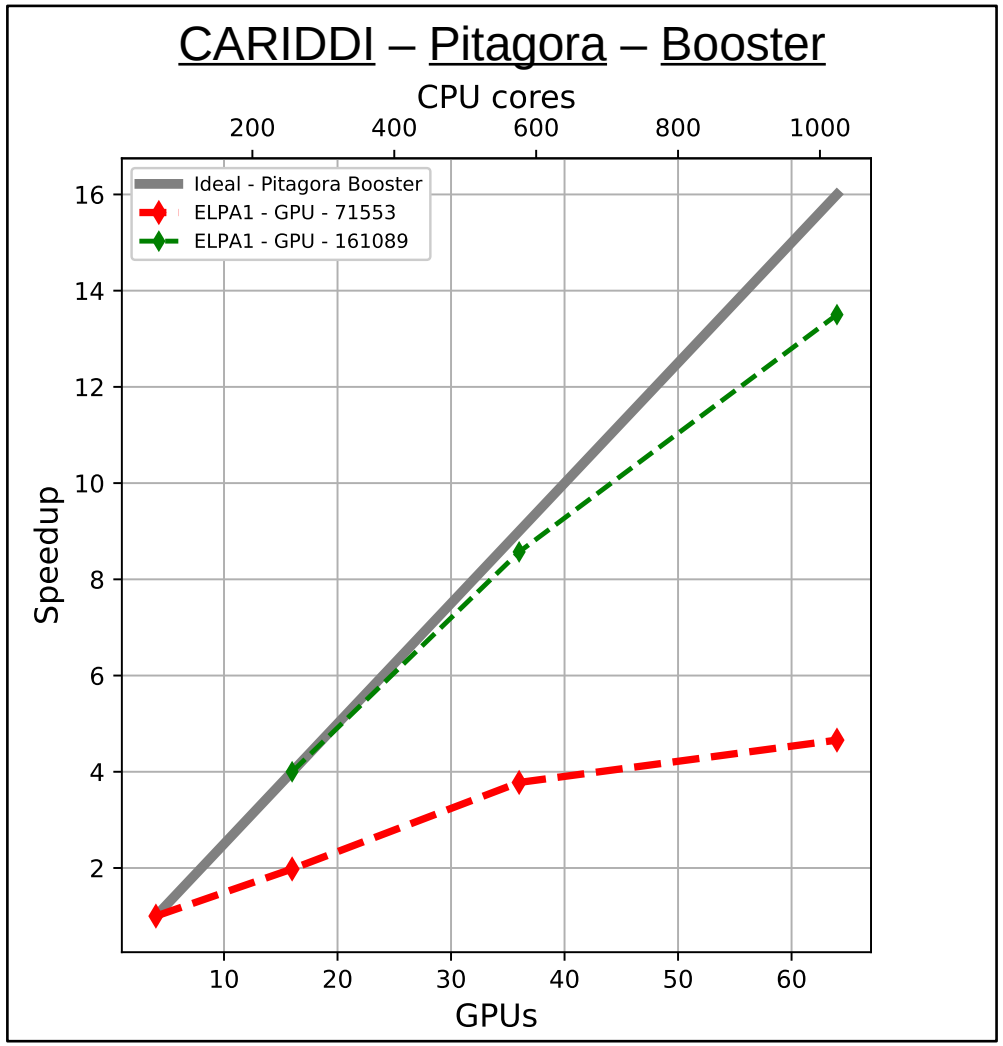


Computing the Speedup wrt GPUs



The **Speedup** can be computed and expressed vs GPU resources (ELPA1 runs)

Comparing against CPU runs would not be fair...(1 stage solver running on completely different hardware)



20 YEARS BSC

Evaluating the Parallelization Efficiency notes

Regarding DLB and TALP

- The **DLB** library is easy to install and well documented → this minimizes portability issues
- It offers analysis of work done via CPUs and GPUs
- To obtain hybrid CPU-GPU metrics, it must be computed with GPU plugins (**not done** here)
- If your installation is in `${DLB_HOME}` (e.g. set in a module) then the analysis can be performed
  - Bulkly on the whole program execution → export settings in the batch script and use the command

    `srun env LD_PRELOAD="${DLB_HOME}/lib/libdlb_mpi.so" ${PROGRAM} ${INPUT_FILE}`
  - In custom defined regions of the code (**not done** here – see the documentation)

Typical output (reported at the end of the execution)

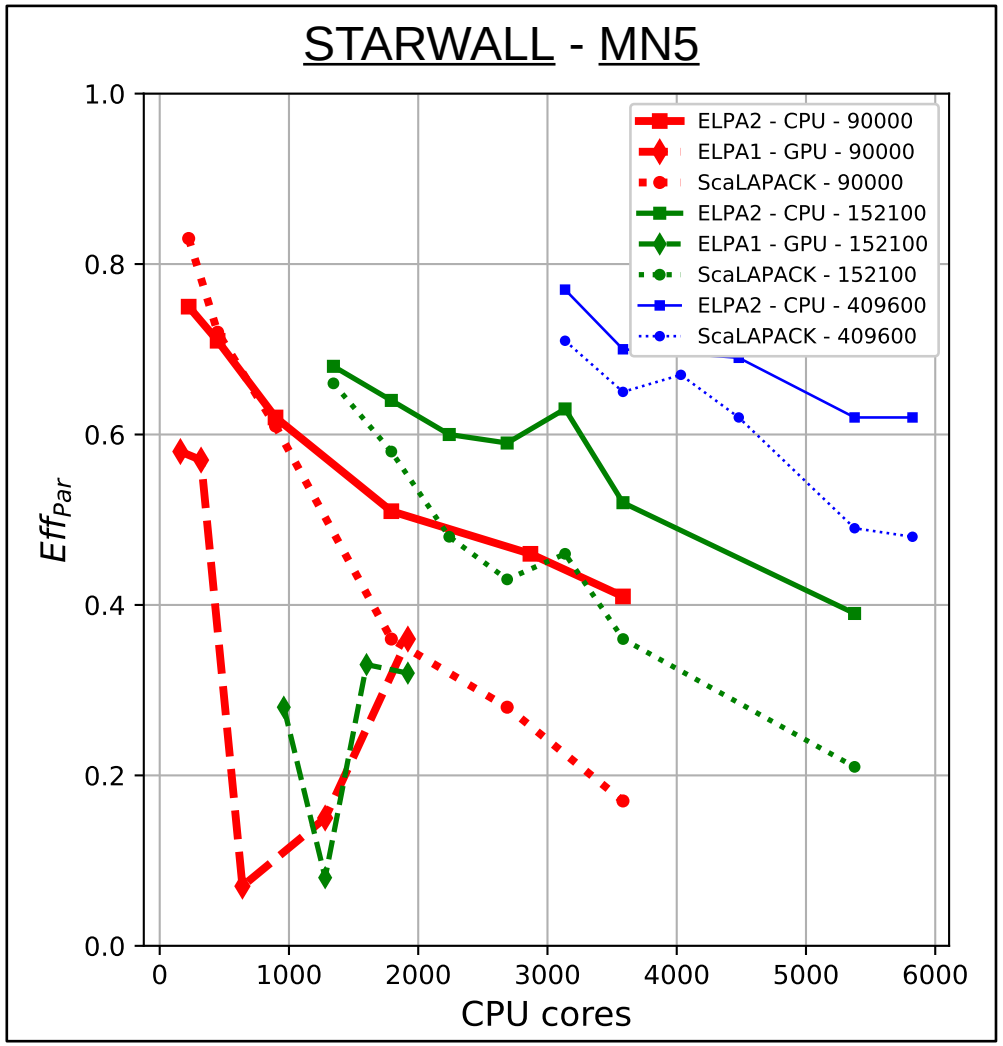
Settings →

```
...
### Set up talp
dlb_args=()
dlb_args+=("--talp")
dlb_args+=("--talp-summary=pop-metrics")
dlb_args+=("--ompt")
export IFS=" "
export DLB_ARGS="${dlb_args[@]}"
...
```

Output (stderr)

```
DLB[as03r4b02:1823769]: ############### Monitoring Region POP Metrics ###############
DLB[as03r4b02:1823769]: ### Name:                                          Global
DLB[as03r4b02:1823769]: ### Elapsed Time:                                  377.10 s
DLB[as03r4b02:1823769]: ### Parallel efficiency:                           0.15
DLB[as03r4b02:1823769]: ###   - MPI Parallel efficiency:                   0.15
DLB[as03r4b02:1823769]: ###     - Communication efficiency:               0.16
DLB[as03r4b02:1823769]: ###     - Load Balance:                           0.95
DLB[as03r4b02:1823769]: ###       - In:                                   0.96
DLB[as03r4b02:1823769]: ###       - Out:                                  0.99
```

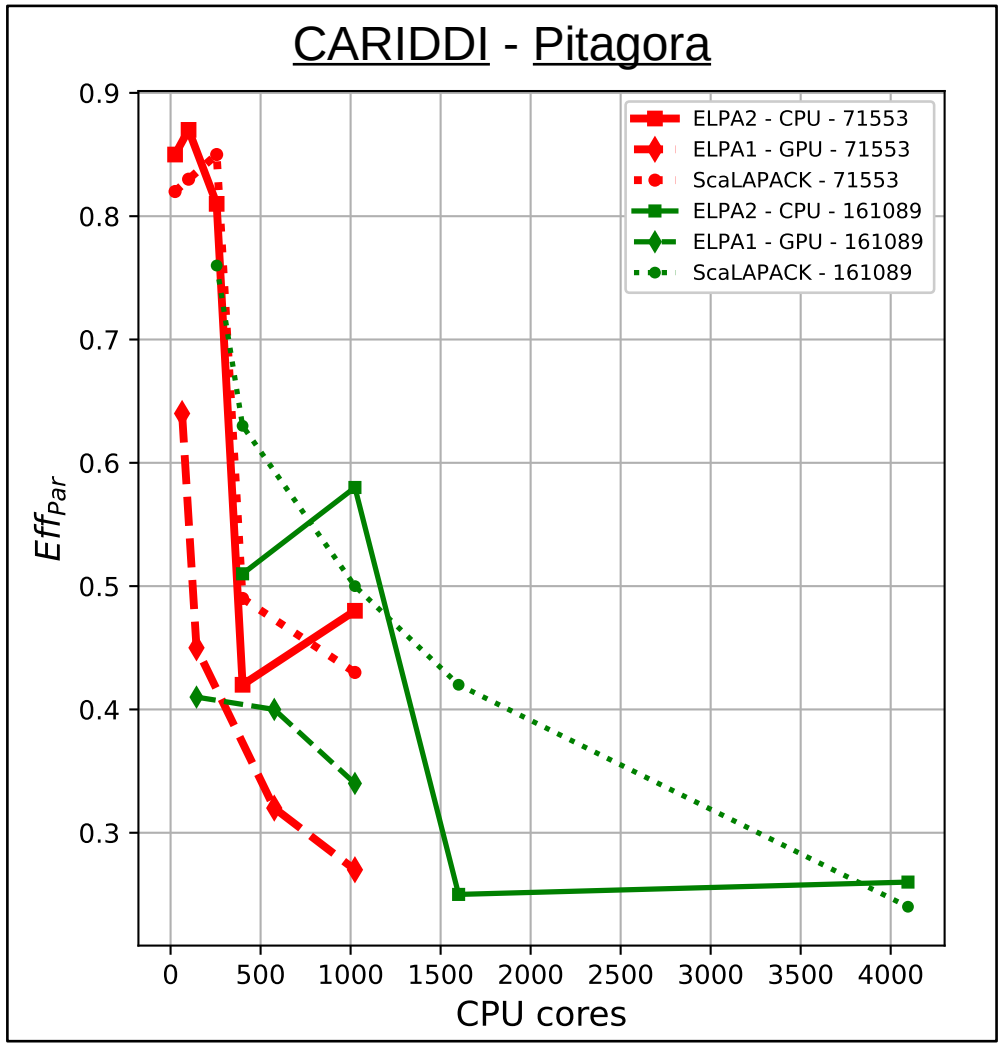
Evaluating the Parallelization Efficiency



Measure Eff_{Par} with TALP metrics from the DLB library (Pure MPI)

$$Eff_{Par} = Eff_{Comm} \times MPI_{lb}$$

From the data, the degradation of performance of GPU runs on the left is due to low Eff_{Comm} while MPI_{lb} is kept reasonable



Evaluating the Parallelization Efficiency comments

Regarding the performance issue observed for some GPU runs

- **ELPA1** and **ELPA2** algorithms are different (1-stage VS 2-stage)
- Host-Device communications are performed internally in the **ELPA** library
- Strong scaling is meant to test how well the code scales when (MPI) **communications increase**
- In hybrid CPU-GPU codes, the communication between host and device is a known possible bottleneck

Portability

- DLB can be installed **easily** → portability issues are **minimized**

Estimating the Memory Usage (CPU)

- Measure memory consumption per node with **free**, at a prescribed time interval
- Add a (background) subroutine to the batch script to log in to the compute nodes and launch free
- Small **differences** on different machines…

Useful variables

Nodes' list

Header of output

Subroutine
Read nodes' names

Get time stamp

Get mem usage

ssh to the node
dump to output

End Subroutine

Call the subroutine in background

MN5

```
# After setting OUTDIR and LOGNAME
interval=5
log_file=${OUTDIR}/01_log_${LOGNAME}
free_file=${OUTDIR}/02_free_${LOGNAME}
nodelist_file=${OUTDIR}/03_nodelist_${LOGNAME}

# Get the list of nodes where the job is running
scontrol show hostnames "$SLURM_JOB_NODELIST" | awk '{for(i=1;i<=NF;i++) print $i}' > $nodelist_file

# Header for the log file
echo "------------------------------------------------------------" >> $free_file
echo "Hostname   Timestamp            used [MB]    total [MB] " >> $free_file
echo "------------------------------------------------------------" >> $free_file

# Function to get memory usage per node
get_memory_usage_free() {

while ISF= read -r host
do {
  printf "%s\n" "$host"
  # Get the current timestamp
  timestamp=$(date "+%Y-%m-%d_%H:%M:%S")
  # Get total memory usage using `free` command
  # free -h for human-readable output, or `-m` for megabytes, '-g' for gigabytes
  node_memory=$(free -m | grep Mem | awk '{print $3 "    " $2 }')
  # Using " characters for the commands to pass through ssh, makes
  # each command to be evaluated within each ssh session
  # NOTE: Using " lets evaluating the variable at each ssh session, with the
  #       variable defined above
  ssh "$host" "
    echo "$host   $timestamp    $node_memory" >> $free_file
    "
  } 0<&3
done 3<&0 < $nodelist_file

}

# Run the memory monitoring in the background
while true; do
  get_memory_usage_free
  sleep $interval
done &
```

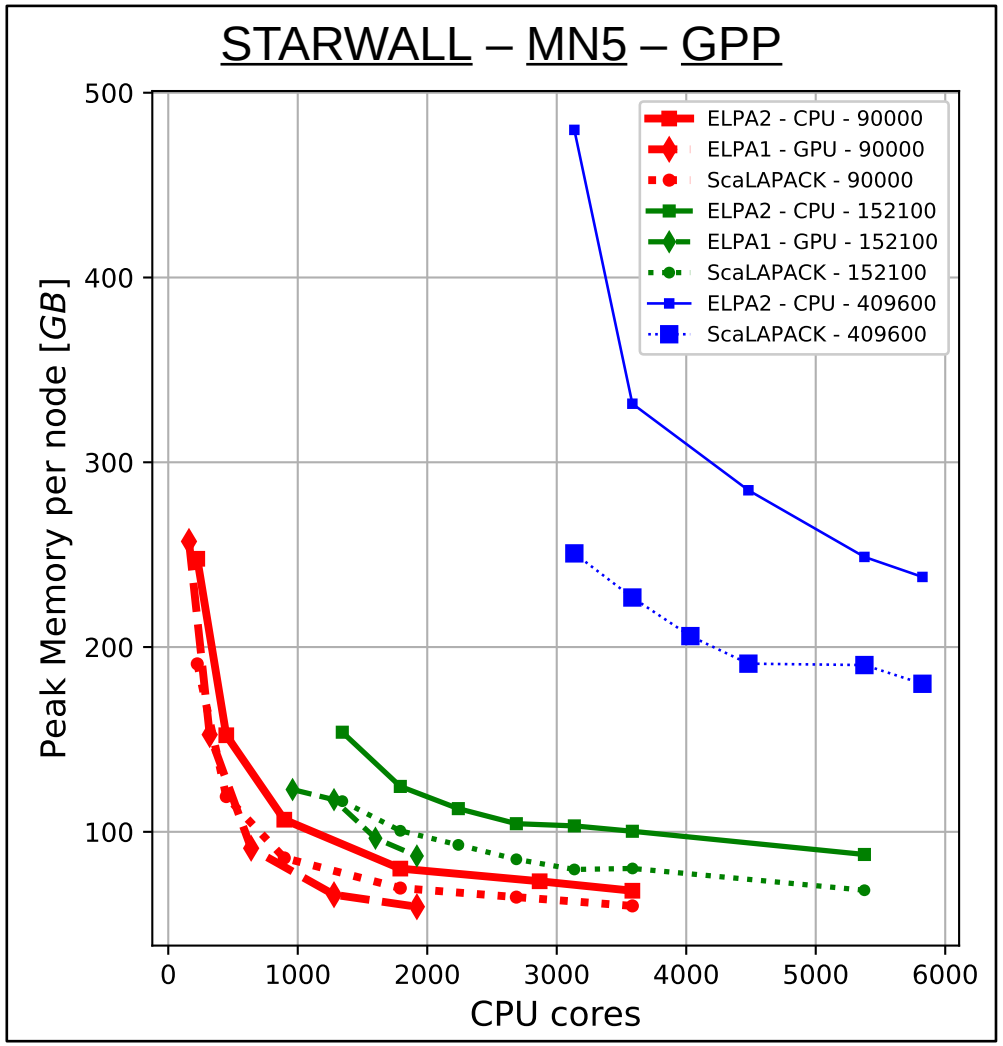
MN5/free_subroutine_MN5 [+] 43,1 All

Pitagora

```
# After setting OUTDIR and LOGNAME
interval=5
log_file=${OUTDIR}/01_log_${LOGNAME}
free_file=${OUTDIR}/02_free_${LOGNAME}
nodelist_file=${OUTDIR}/03_nodelist_${LOGNAME}

# Get the list of nodes where the job is running
srun hostname | sort | uniq  > $nodelist_file

# Header for the log file
echo "------------------------------------------------------------" >> $free_file
echo "Hostname   Timestamp            used [MB]    total [MB] " >> $free_file
echo "------------------------------------------------------------" >> $free_file

# Function to get memory usage per node
get_memory_usage_free() {

while ISF= read -r host
do {
  printf "%s\n" "$host"
  # Get the current timestamp
  timestamp=$(date "+%Y-%m-%d_%H:%M:%S")
  # Get total memory usage using `free` command
  # free -h for human-readable output, or `-m` for megabytes, '-g' for gigabytes
  node_memory=$(free -m | grep Mem | awk '{print $3 "    " $2 }')
  # Using " characters for the commands to pass through ssh, makes
  # each command to be evaluated within each ssh session
  # NOTE: Using " lets evaluating the variable at each ssh session, with the
  #       variable defined above
  ssh -o BatchMode=yes -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no $host "
    echo "$host   $timestamp    $node_memory" >> $free_file
    "
  } 0<&3
done 3<&0 < $nodelist_file

}

# Run the memory monitoring in the background
while true; do
  get_memory_usage_free
  sleep $interval
done &
```

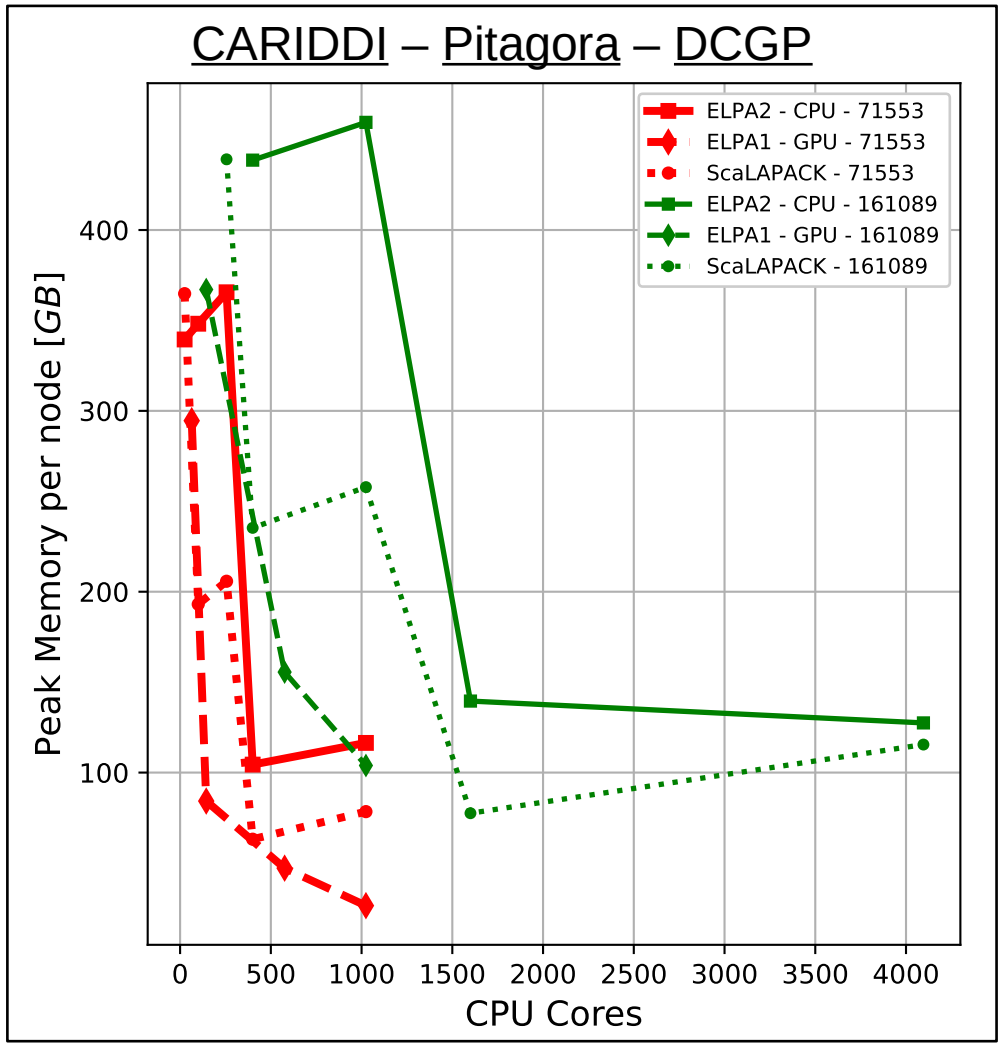
Pitagora/free_subroutine_Pitagora 43,1 All

Estimating the Memory Usage (CPU)

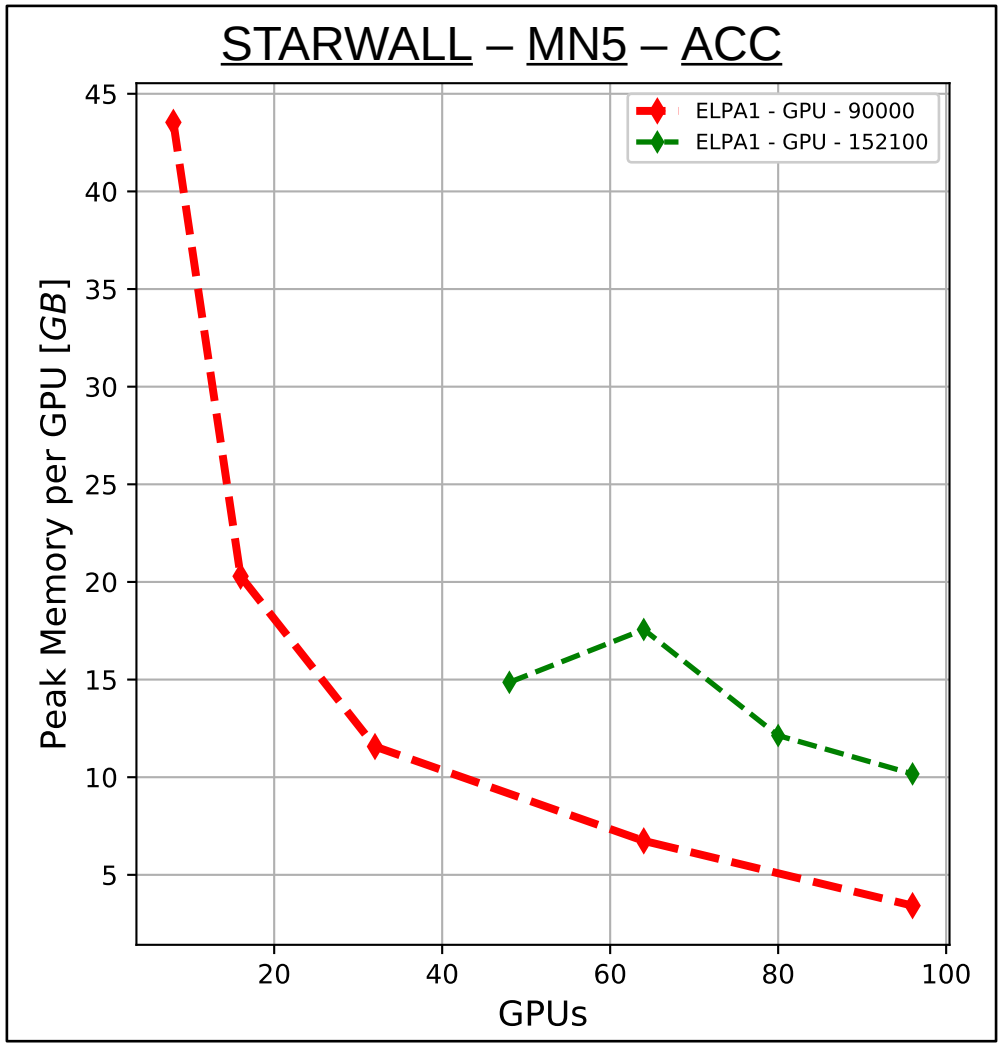


The Peak memory is simply the **maximum** of the numbers in the output file of the previous slide

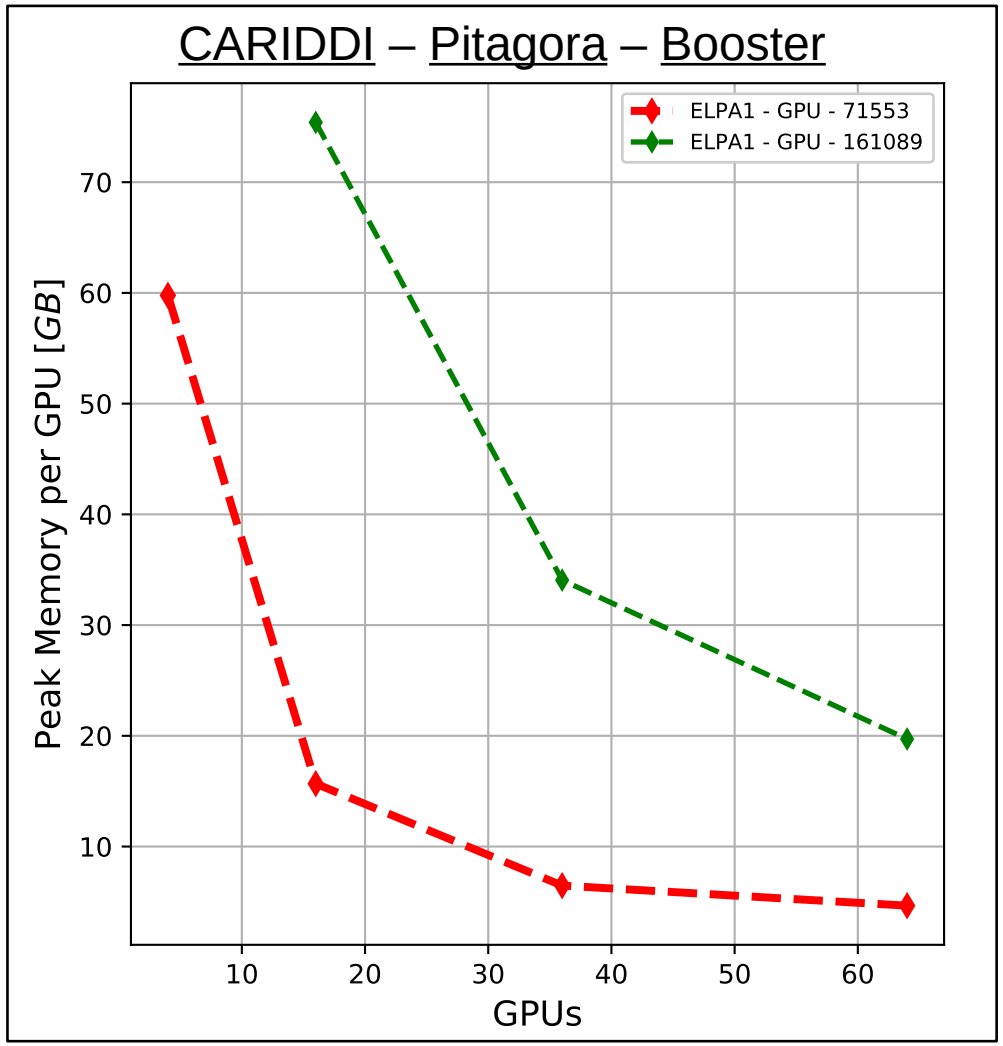
On the x the CPU cores are considered



Computing the Memory Usage (GPU)



Same approach as for CPU, but calling a query via the **`nvidia-smi`** command to get the memory used by GPUs



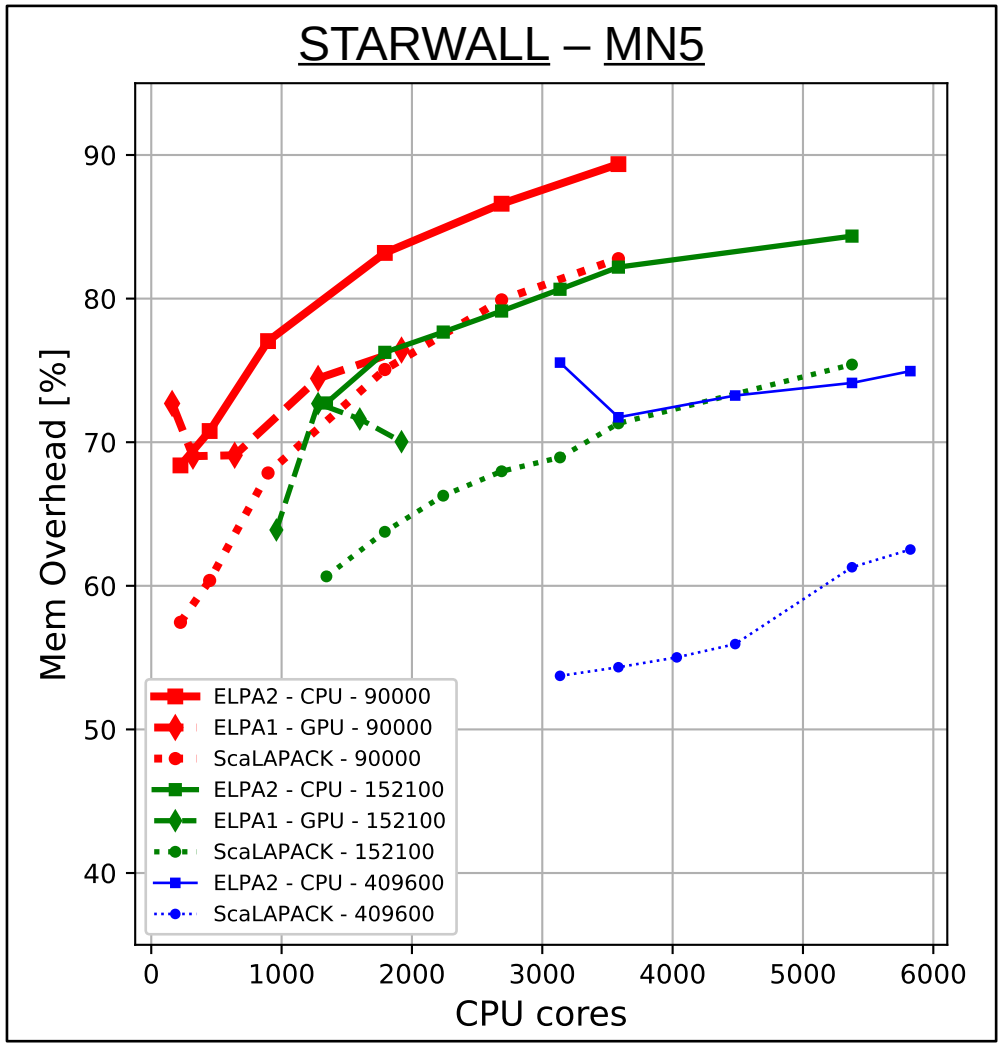
Estimating/Computing the Memory Overhead

- Using **`nvidia-smi`** is very precise for the GPUs
- Using **free** and taking the maximum can be less precise → it gives more than the memory for the job
- Luckily, using **SLURM**, gives you access to **`sstat`**
- Even more luckily, in **SLURM** you also have **`sacct`** for already ended jobs

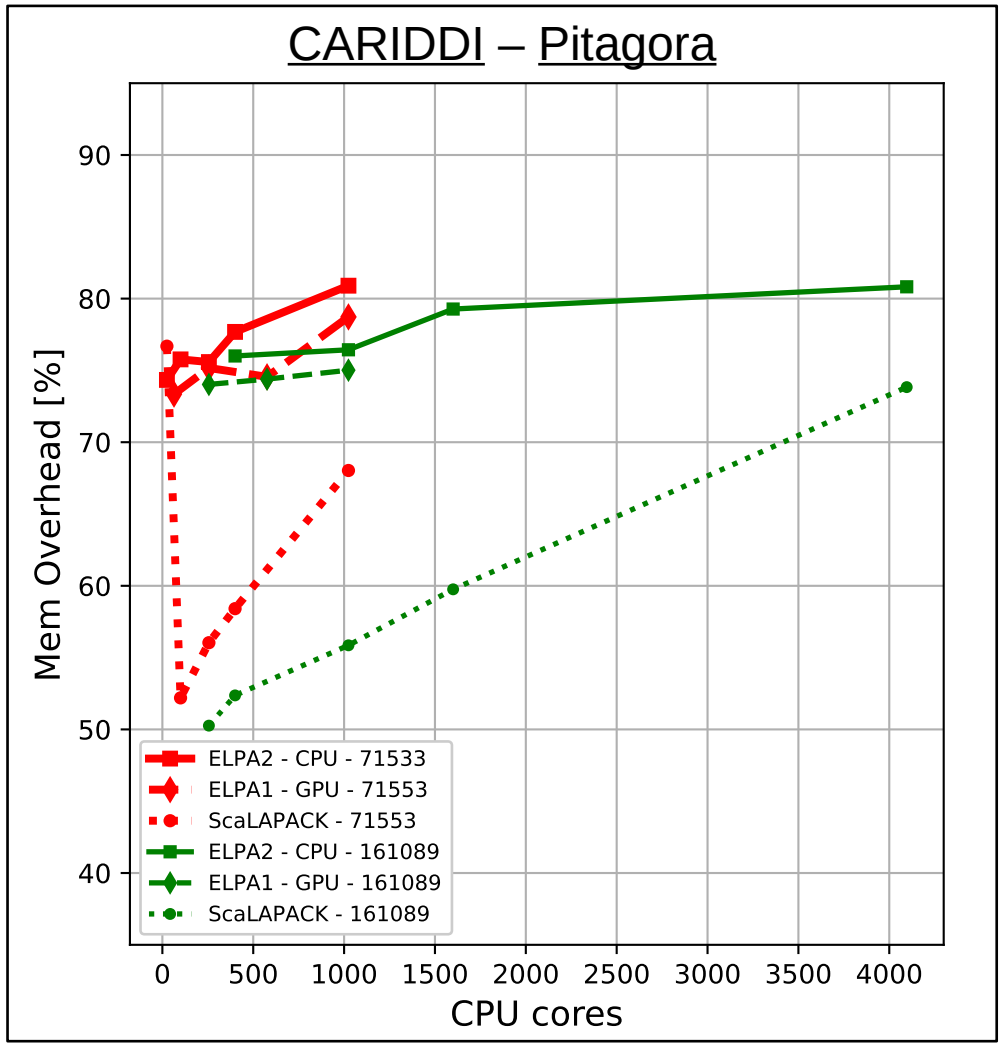
```
# Retrieving all the jobs from 01/01/1970 with a given Job Name
sacct -u $USER -S 1970-01-01 --
format=JobID,JobName,State,ExitCode,End,Elapsed,MaxRSS,ReqMem,NTasks,AllocTRES%100 \
--units=G --name={NAME}
# From a file with the job id list
sacct -j $(paste -sd, ./{file_jobid_list}) --format=JobID,JobName,Elapsed,End,MaxRSS -units=G
```

- `RSS` = **Resident Set Size** → it provides the information of the physical memory used
- **Portability**: if the `MaxRSS` is returning the single process data (and not per node), you may prefer `AveRSS`
- The **memory overhead** is given by the difference between the memory used and the useful memory
- It is better to normalize by the memory used
- Alternative tools are the **Intel profiler** or **Score-P** (not covered here)

Memory Overhead



The worst memory overhead is obtained for small matrices (this depends on the algorithms)



Conclusions

Summary

- The gEP problem in STARWALL and CARIDDI
- ELPA has been assessed as an alternative solver to ScaLAPACK, on the basis of
  - Computational time → **WTIME**
  - (Pure MPI) Parallel Efficiency (only on CPUs) → **DLB/TALP**
  - (Peak) Memory usage → **`free`** and **`nvidia-smi`**
  - Memory Overhead → post process info from **`nvidia-smi`** and **`sacct`**
- Results have been obtained using MN5 and Pitagora
- Ideas on portability issues have been provided

Takeaway and perspectives

- **ELPA** offers a very promising solvers for the gEP, in terms of time-to-solution (kind of 3X)
  - CARIDDI team is interested in implementing the library in the code
  - It also offers GPU capabilities → almost automatic introduction within CARIDDI
- The memory utilization of **ELPA** is slightly worse (in some cases 2X) than **ScaLAPACK**
  - It might **not** be the **best choice** to grow with treatable matrix dimensions
- Some more studies and trials might be needed for exploiting GPUs
- Even if the strong scaling of **ELPA1** on GPUs is not comparable with the others, its values of time to solution are very attractive

Thank you

fusion.bsc.es

@Fusion_BSC

fusion-bsc

This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 - EUROfusion). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Commission. Neither the European Union nor the European Commission can be held responsible for them.

EUROfusion E-TASC General Meeting #2