How EUROfusion Advanced Computing Hubs leverage high-performance computing to accelerate research and engineering in nuclear fusion

Dr. Gilles Fourestey

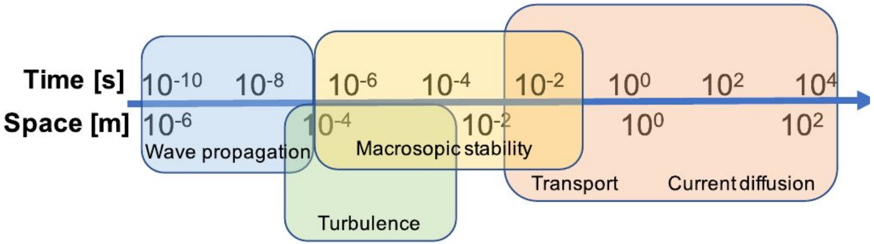
Simulations/HPC to model the hot plasma

- Fusion reactors are **extremely complex to build**
- **Numerical simulations** are an ***essential* tool** to help their design
- But are also **extremely demanding** both in terms of models and resources

Coupling between different fields...

- Electromagnetic
- Plasma physics kinetic, gyrokinetic,
- Two-fluids,
- MHD models
- Material science plasma-wall interactic
- Wave physics heating systems
- Engineering not included!

with different space/time scales...



... and HPC motifs and their hardware implementations

Science areas	Multi-physics, Multi-scale	Dense linear algebra (DLA)	Sparse linear algebra (SLA)	Spectral Methods (FFT)s (SM-FFT)	N-Body Methods (N-Body)	Structured Grids (S-Grids)	Unstructured Grids (U-Grids)	Data Intensive
Nanoscience	X	X	X	X	X	X		
Chemistry	X	X	X	X	X			
Fusion	X	X	X			X	X	X
Climate	X		X	X		X	X	X
Combustion	X		X			X	X	X
Astrophysics	X	X	X	X	X	X	X	X
Biology	X	X					X	X
Nuclear		X	X		X			X
System Balance Implications	General Purpose balanced System	High Speed CPU, High Flop/s rate	High Performance Memory	High Interconnect Bisection bandwidth	High Performance Memory	High Speed CPU, High Flop/s rate	Irregular Data and Control Flow	High Storage and Network bandwidth

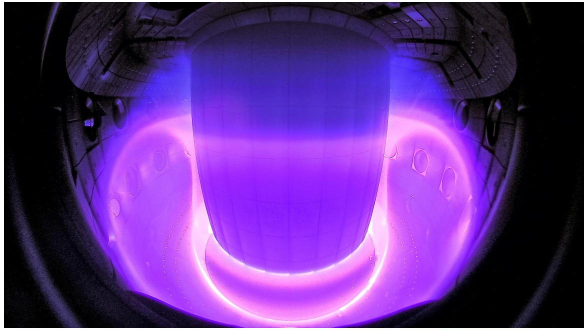
Fusion Timeline

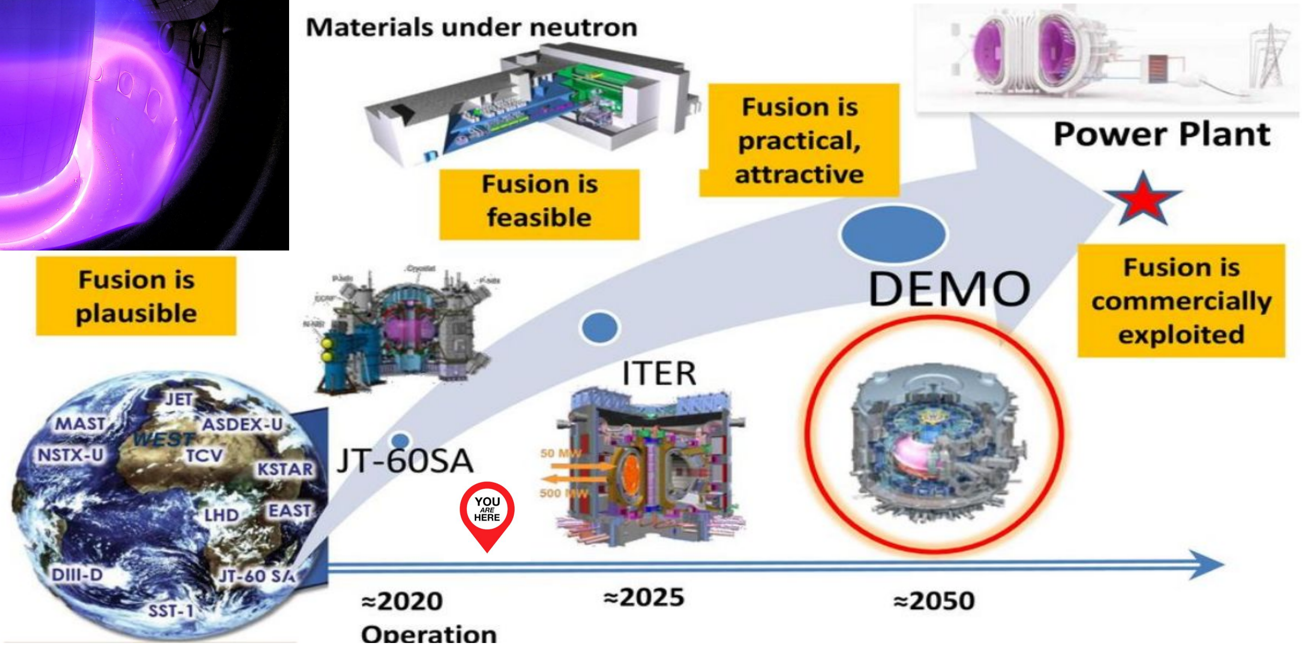
TCV@EPFL

JT-60: **100x TCV**
Q = 1.25

ITER: **500x TCV**
Q = 10, 500MW

DEMO: **5000x TCV**
Q = 25, 2000MW





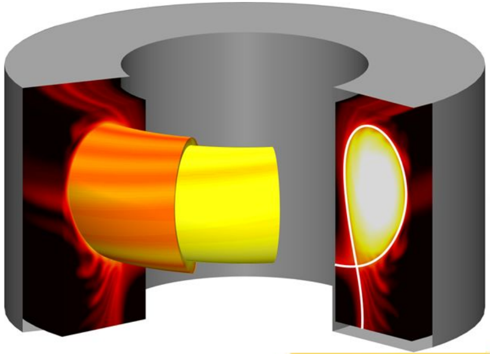
Fusion Timeline

TCV@EPFL
100 Teraflops

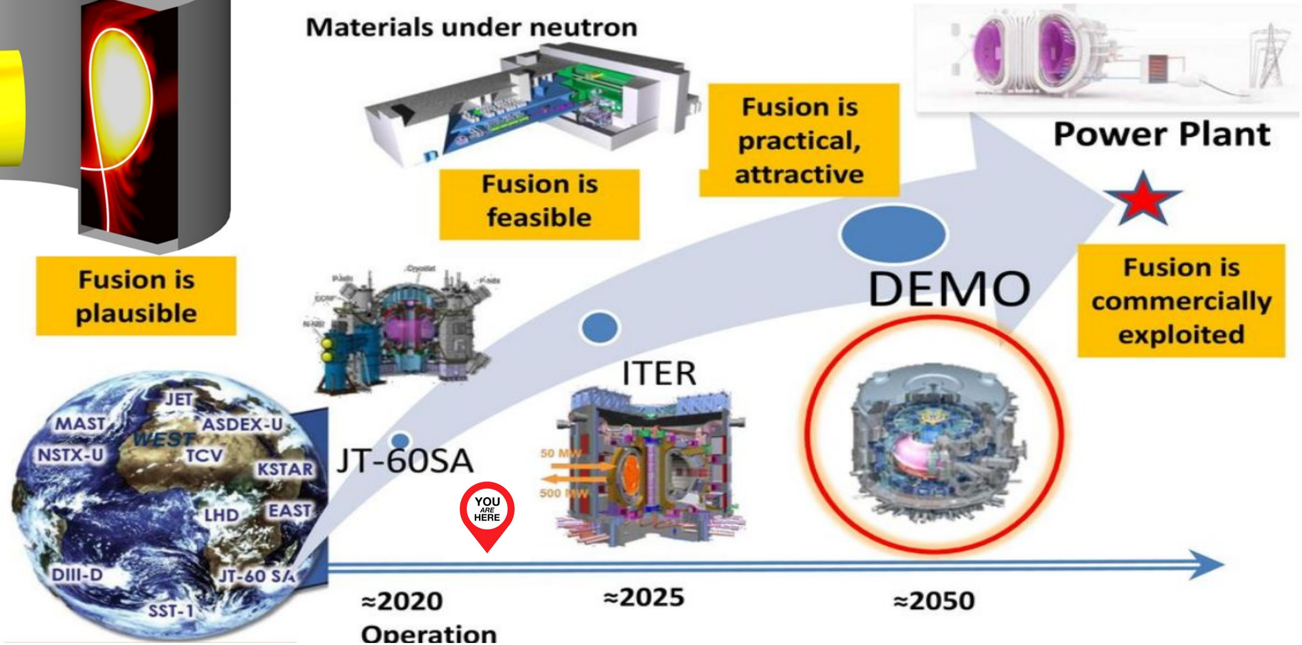
JT-60: 100x TCV
Q = 1.25
10 Petaflops

ITER: 500x TCV
Q = 10, 500MW
100 Petaflops

DEMO: 5000x TCV
Q = 25, 2000MW
1 Exaflops



Ricci at al.
(SPC-EPFL)



Unprecedented level of heterogeneity

EPFL

- **GPUs** are dominating the **Top500**
- but the **CPU/GPU combo is rarely the same** vendor-wise
- And there's more to come:

intel. intel.

arm

LUMI

AMD AMD

LEONARDO CINECA

intel. NVIDIA.

arm NVIDIA.

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8,699,904	1,194.00	1,679.82	22,703
2	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	4,742,808	585.34	1,059.33	24,687
3	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Microsoft Azure United States	1,123,200	561.20	846.84	
4	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7,630,848	442.01	537.21	29,899
5	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2,752,704	379.70	531.51	7,107
6	**Leonardo** - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, EVIDEN EuroHPC/CINECA Italy	1,824,768	238.70	304.47	7,404
7	**Summit** - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2,414,592	148.60	200.79	10,096
8	**MareNostrum 5 ACC** - BullSequana XH3000, Xeon Platinum 8460Y+ 40C 2.3GHz, NVIDIA H100 64GB, Infiniband NDR200, EVIDEN EuroHPC/BSC Spain	680,960	138.20	265.57	2,560
9	**Eos NVIDIA DGX SuperPOD** - NVIDIA DGX H100, Xeon Platinum 8480C 56C 3.8GHz, NVIDIA H100, Infiniband NDR400, Nvidia NVIDIA Corporation United States	485,888	121.40	188.65	
10	**Sierra** - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1,572,480	94.64	125.71	7,438

EUROfusion Advanced Computing Hubs

EUROfusion is a consortium of national fusion research institutes

- 30 research groups (25 in Europe)
- Founded in 2014, HQ at IPP

Aiming at helping **deliver fusion energy.**

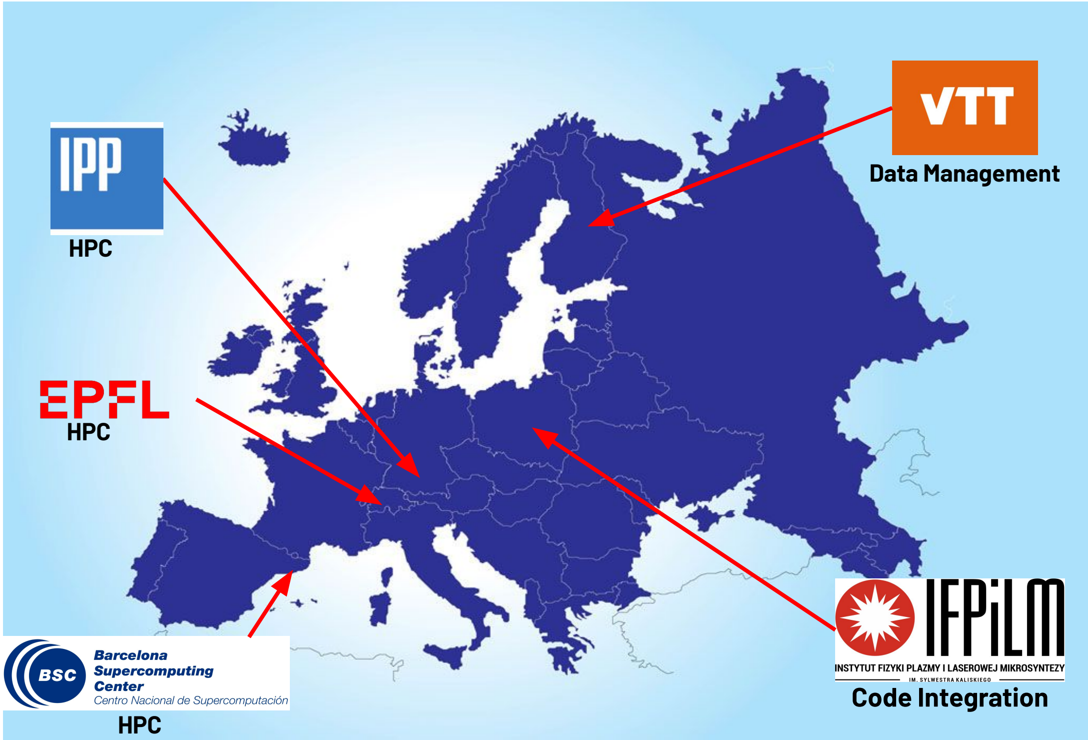
E-TASC – Theory and Advanced Simulation Coordination between:

- **14 TSVV** (Theory, Simulation, Validation and Verification) projects
- **5 ACH** (Advanced computing Hubs)

In particular, **3 HPC ACHs** were created in order **to help building power plants through numerical simulations**

- Extension of **HLST** (Roman Hatsky/IPP)
- Develop **efficient, reliable** tools
- **Modernize and industrialize research codes**

In order to **gain insight and predict fusion experiments** (ITER, JT60-SA, DEMO…)



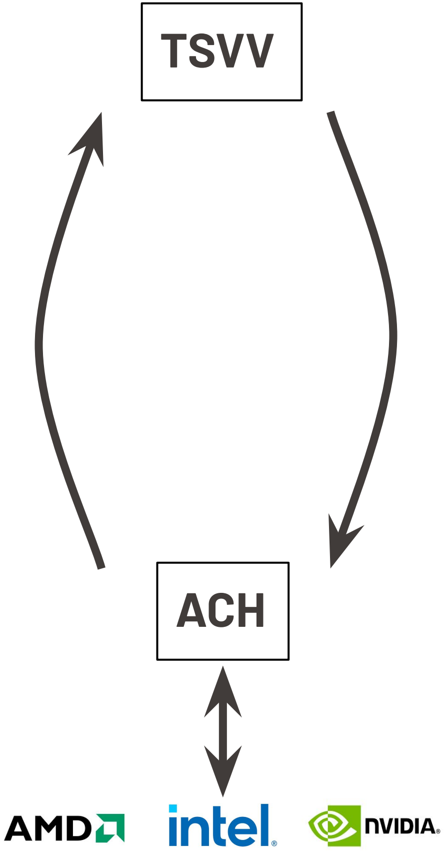
EUROfusion Advanced Computing Hubs

E-TASC – Theory and Advanced Simulation: coordination between

- **TSVV**: perform research and channel science and engineering into scientific codes
- **ACH**: modernize and industrialize research codes into HPC standards

According to EUROfusion's Software Standards:

- **SOFTWARE ENGINEERING**: Version control, coding standards, test-driven.
- **CODE INTERFACES**: Graphical User Interface, post-processing and visualisation tools, interfaces to the IMAS Data Dictionary.
- **VVUQ**: Code Verification and Validation, reports/papers available, validation against experimental results.
- **CODE DISSEMINATION**: Up-to-date release version of the source code available, trainings provided.
- **CODE DOCUMENTATION**: High-quality technical documentation and user manual available.
- **USER SUPPORT**: Responsive support team in place, tools for managing support requests.



Plasma Physics Codes

Plama simulation codes are research codes:

- mostly **CPU-only**,
- written in **C,C++ and Fortran**,
- **MPI** and/or **OpenMP**,
- **under active development** by physicists, mathematicians, etc…

General porting rules:

- **least modifications** of the code/no rewrite
- “maximum” **performance**
- **portability**/no specific target

- GRILLIX
- GyselaX
- ORB5
- Soledge3X
- ASCOT5
- CAS3D
- FELTOR
- GBS
- GENE

How to transition towards GPU codes?

There are **3 main approaches**:

- **Library encapsulation** (e.g. Kokkos, PETSc, AmgX, FFTW, BLAS/Lapack...):
  - easy to use, good out-of-the-box performance
  - not applicable to all codes and might necessitate some rewrite (e.g. data structures)
- **Cuda/ROCm/SYCL**: low level high performance
  - best performance
  - not portable, necessitate heavy rewrite
- **Pragma directives** (OpenMP offload / OpenACC)
  - portable, relatively easy to use
  - some code rewrite and possible algorithmic modifications

How to transition towards GPU codes

There are 3 main approaches:

Library encapsulation (e.g. Kokkos, PETSc, AmgX, BLAS/Lapack...)

Pragma directives (OpenMP offload / OpenACC)

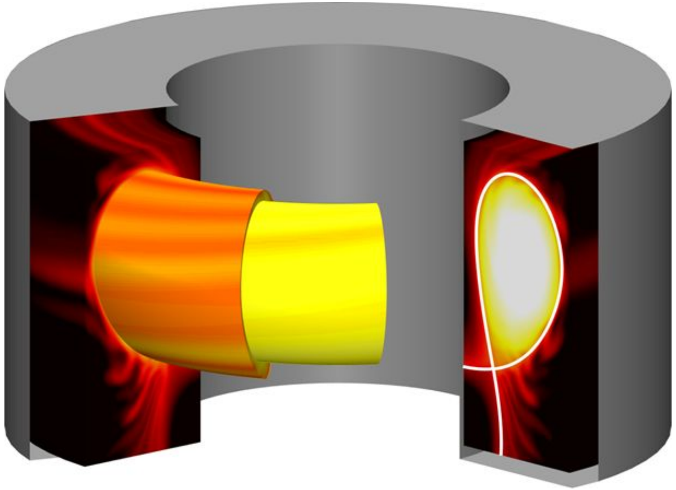
Cuda/ROCm/SYCL

- ASCOT5 — Aalto University
- CAS3D — IPP
- FELTOR — DTU
- GBS — EPFL
- GENE — IPP
- GRILLIX — IPP
- GyselaX — cea irfm
- ORB5 — EPFL
- Soledge3X — cea irfm

GBS - Global Braginskii Solver (Ricci et al. EPFL)

GBS is used to study **plasma turbulence** in the tokamak boundary

- Plasma model based on drift-reduced Braginskii equations
- Single species kinetic neutral model
- Time evolution: **4th order Runge-Kutta algorithm**
- Spatial discretisation: **4th order finite centered differences**



HPC in GBS:

- Plasma model based on drift-reduced Braginskii equations
- Written in **Fortran90 + MPI, CUDA for NVIDIA GPU**
- Dependencies: **MPI, HDF5, PETSc, CUDA**

Main bottlenecks:

- RHS computation (stencil operations)
- Poisson and Ampere solvers

GBS: TCV on Piz Daint@CSCS (Cray XC40)

1st step: sparse linear solver library

- Moving from UMFpack to PETSc

65X performance increase using 32 nodes

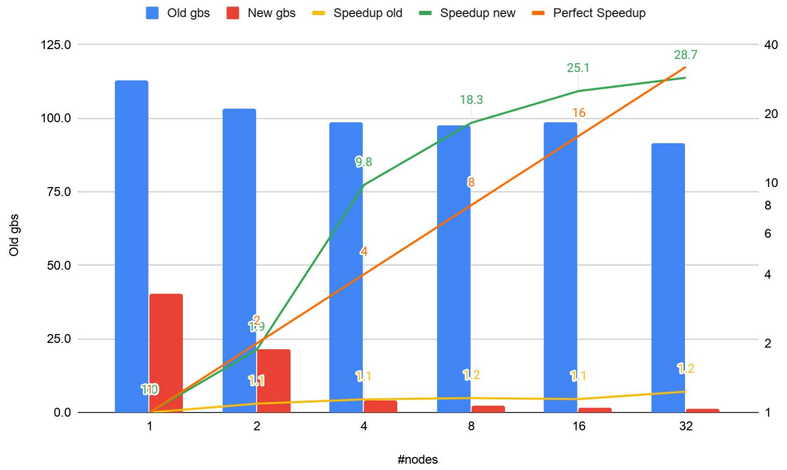
2ns step: GPU implementation

- **RHS:** stencil operations using **CUDA kernels**
- **Solver: PETSc or AMGX on GPU**

Outcome: currently, allocation of

- 500M core-hours
- 5M GPU-hours

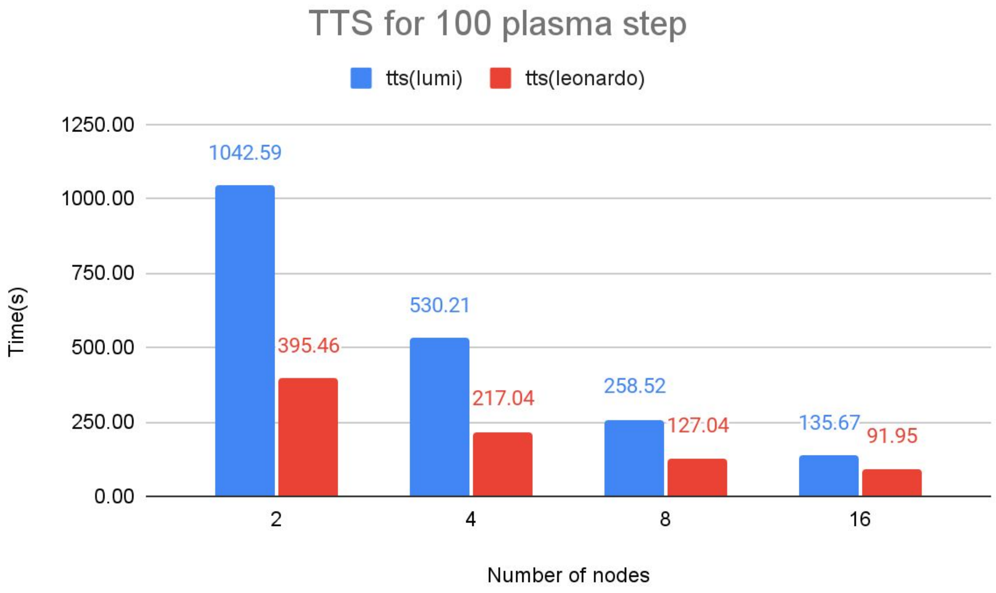
on multiple Tier-0 HPC systems



Setup: **TCV**, 2D poisson, 1 timestep

- Nx = 720, ny = 960, nz = 1
- Solver: BiCGStab
- Preconditioner: jacobi
- Xeon E5-2690V3@2.6Ghz, 64GB ram,
- Aries interconnect

Leonardo vs LUMI-C – TCV@0.9T



- Grid size:
  - Nx=300
  - Ny=600
  - Nz=128
- 100 plasma steps
- No neutrals

nodes	Px	Py	Pz
2	8 1	16 1	2
4	8 1	16 1	4
8	8 1	16 1	8
16	8 1	16 1	16

Leonardo is 2x to 3x faster than LUMI-C

ASCOT5 J. Varje, S. Äkäslompolo (ASCOT group - Aalto University)

- ASCOT5 is a test **particle orbit-following** code for toroidal magnetically confined fusion devices
- The code uses the **Monte Carlo method** to solve the distribution of particles by following their trajectories.
  - The **evolution of the distribution function** for a test particle species *a* is described by the **Fokker-Planck equation**

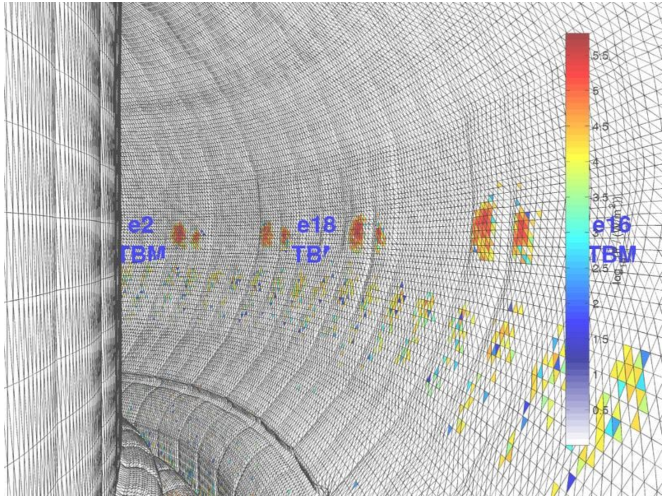
$$\frac{\partial f_a}{\partial t} + \mathbf{v} \cdot \nabla f_a + \frac{q_a}{m_a}(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \nabla_{\mathbf{v}} f_a = \sum_b -\nabla_{\mathbf{v}} \cdot [\mathbf{a}_{ab} f_a - \nabla_{\mathbf{v}} \cdot (\mathbf{D}_{ab} f_a)]$$

and **approximated by the Langevin equation** for a large number of markers that represent the distributed function:

$$d\mathbf{z} = [\dot{\mathbf{z}} + \mathbf{a}(\mathbf{z}, t)]\, dt + \boldsymbol{\sigma}(\mathbf{z}, t) \cdot d\mathcal{W}$$

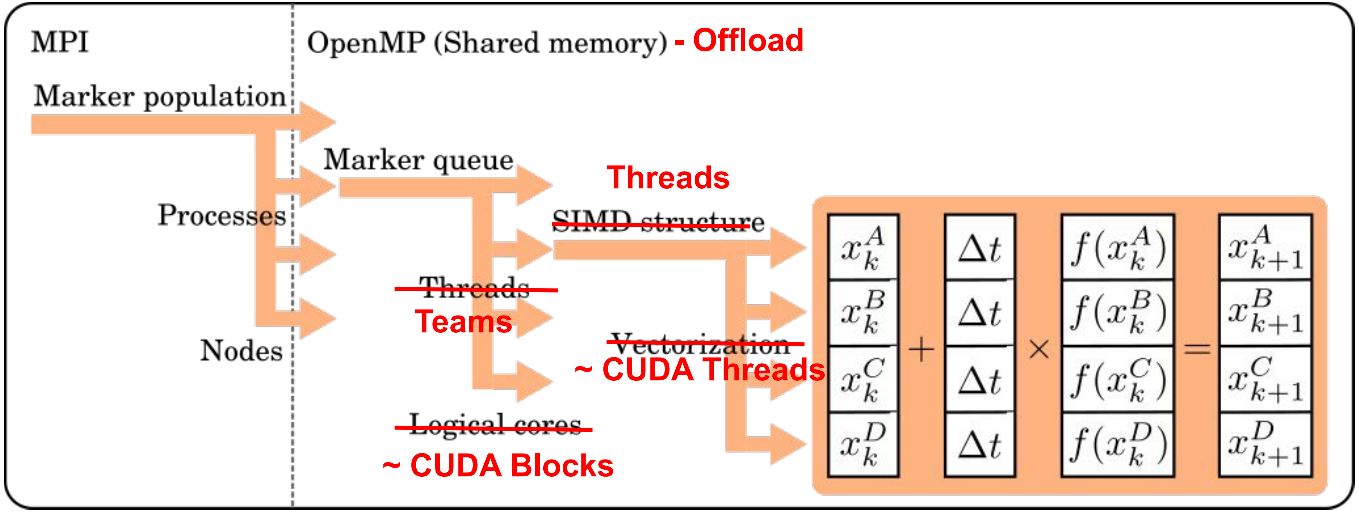
- The particles undergo **collisions with a static Maxwellian background plasma**
- The detailed magnetic fields and the first wall can be **fully 3D**

- **MPI + OpenMP** (task-based) and **highly vectorized**



Moving to GPUs: OpenMP-Offload hardware mapping

- **MPI+GPU** levels of parallelism:
  - Message Passing: particles distributed among MPI tasks, fields replicated
  - GPU OpenMP-offload based – **2 levels of parallelism** map to:
    - Marker queues distributed over **OpenMP teams**
    - Each marker is distributed over **OpenMP team threads**

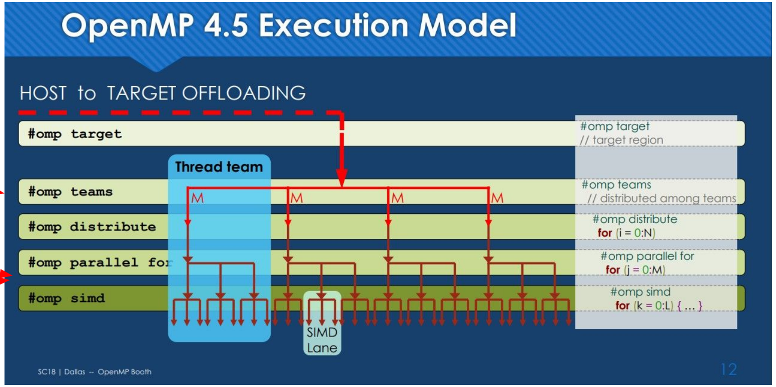


ASCOT5 – GPU version

- **MPI+GPU** levels of parallelism:
  - Message Passing: particles distributed among MPI tasks, fields replicated
  - GPU OpenMP-offload based – **2 levels of parallelism** map to:
    - Marker queues distributed over **OpenMP teams**
    - Each marker is distributed over **OpenMP team threads**

L1
L2
L2
GPU

```
#pragma omp target teams distribute
for(int iprt = 0; iprt < NbParticules ; iprt += NSIMD) {
        …some work…
        particle_simd_fo p; //new set of NSIMD particles
        #pragma omp parallel for simd
        for(int i=0; i< NSIMD; i++) {
                p.running[i] = 0;
                …some work…
        #pragma omp parallel for simd
        for(int i=0; i< NSIMD; i++) {
                …some work…
```



ASCOT5 Benchmarks

"May2022" Benchmark, comparison with different compilers/platforms

- gcc11 on x86 + v100 (Phoenix@EPFL)
- XL compilers + v100 (m100@Cineca)
- intel compilers on skylake and icelake (Jed@EPFL, ASCOT5 cpu-only)
- gcc11 with OpenACC on x86 + v100 (Phoenix@EPFL)

ASCOT5	TTS [s]	may2022_2dwall_go_analyticB			
	markers:	10000	100000		
				Platform	Compiler
m100@CINECA	**OMP Offload**	**46**	**473**	Power9 + v100	XL compilers
Phoenix@EPFL	**OMP Offload**	**232**	**2143**	6138 gold + v100	**gcc 11**
Phoenix@EPFL	**OpenACC**	**48**	**261**	6138 gold + v100	**gcc 11**
Helvetios@EPFL	**OMP (cpu-only)**	87	860	2x Gold 6140	intel compilers
Jed@EPFL	**OMP (cpu-only)**	31	318	2x Platinum 8360Y	intel compilers

ASCOT5-GPU with OpenACC/OpenMP offload interop

EPFL

Aalto University

Moving to OpenACC

- OpenACC is more mature than OpenMP offload
- gcc supports it along OpenMP offload (-fopenacc or -fopenmp)
- OpenACC and OpenMP offload are very similar

```
OMP_L1
for(int iprt = 0; iprt < NbParticules ; iprt += NSIMD) {
        …some work…
        particle_simd_fo p; //new set of NSIMD particles
        OMP_L2
        for(int i=0; i< NSIMD; i++) {
                p.running[i] = 0;
                …some work…
        OMP_L2
        for(int i=0; i< NSIMD; i++) {
                …some work…
```

GPU

```
#pragma once
#define STRINGIFY(X) #X
#define MY_PRAGMA(X) _Pragma(STRINGIFY(X))
#if !defined(_OPENMP) && !defined(_OPENACC)
#warning "No Openmp or OpenACC"
#define OMP_L1
#define OMP_L2
#define DECLARE_TARGET
#define DECLARE_TARGET_END
#endif
#ifdef _OPENMP
#warning "OpenMP"
#define OMP_L1 MY_PRAGMA (omp distribute parallel for)
#define OMP_L2 MY_PRAGMA (omp simd)
#define DECLARE_TARGET         MY_PRAGMA(omp declare target)
#define DECLARE_TARGET_END MY_PRAGMA(omp end declare target
#endif
#ifdef _OPENACC
#warning "OpenACC"
#define OMP_L1 MY_PRAGMA (acc loop gang worker)
#define OMP_L2 MY_PRAGMA (acc vector)
#define DECLARE_TARGET         MY_PRAGMA(acc routine vector)
#define DECLARE_TARGET_END MY_PRAGMA(warning "ACC")
#endif
```

New algorithmic approach

- The original implementation is not GPU-friendly:
  - **one very large kernel**
  - events depend on the previous event

- Implement a new version by **splitting the initial kernel**:
  - **Parallelize over events** instead of execute all particles
  - small kernels independent of each other

```
#pragma acc loop
for each particle
   while particle still running
      step_forward();
      end_condition_time();
      diag();
      …
   end while
end for
```

Initial method

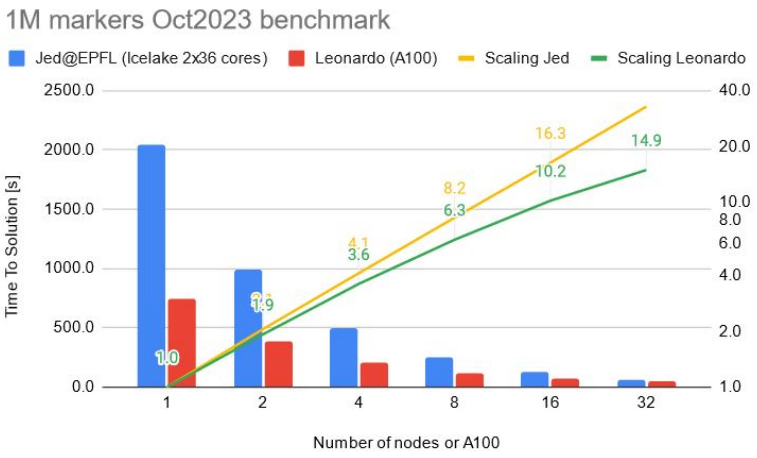
→

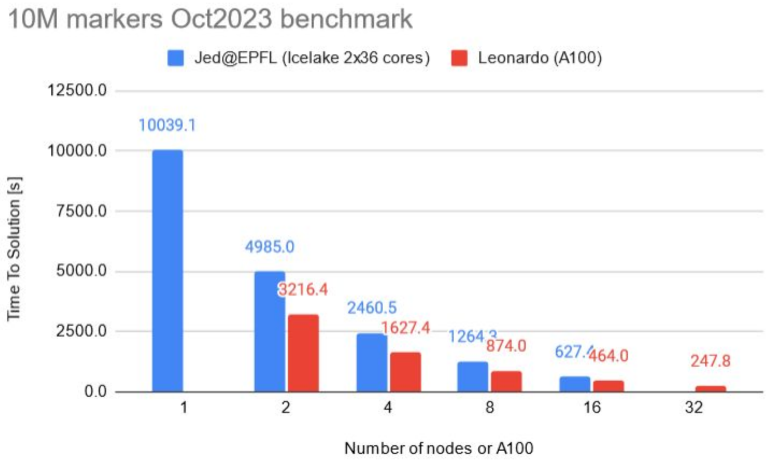
```
while particles are still running
   #pragma acc loop
   for each particle still running
      step_forward();
   #pragma acc loop
   for each particle still running
      end_condition_time();
   #pragma acc loop
   for each particle still running
      diag();
      …
end while
```

Event-Based

Benchmarks

- "Sept2023" Benchmark:
  - **Jed**: 2x Platinum 8360Y, intel/2021.6.0, -Ofast -qopt-zmm-usage=high -march=native
  - **Leonardo**: A100, nvhpc/23.1, -O3 -acc -Minfo=accel -gpu=managed
  - **Time-to-Solution**, lower is better





Conclusions

- EUROfusion has created an framework to improve high performance scientific software development in order to help build fusion reactors
  - **TSVV** to develop research codes and validate them on current reactors
  - **ACH** to industrialize and accelerate those codes to help build future reactors
- There are several ways to do that:
  - **Optimized librairies** are the best way to get performance out-of-the box
  - **CUDA/ROCm/SYCL** will give best performance but not portable
  - **OpenMP offload/OpenACC** directives
- Directive-based approaches are sound, but
  - OpenMP Offload is probably **the best choice but it is not mature**
  - OpenACC is **very fast but is not supported by all**
- Whatever the choice, **expect at least some rewrite**
  - the highest quality the code is, the easiest it will be

Thank you!