PLASMA PEPSC

https://plasma-pepsc.eu/

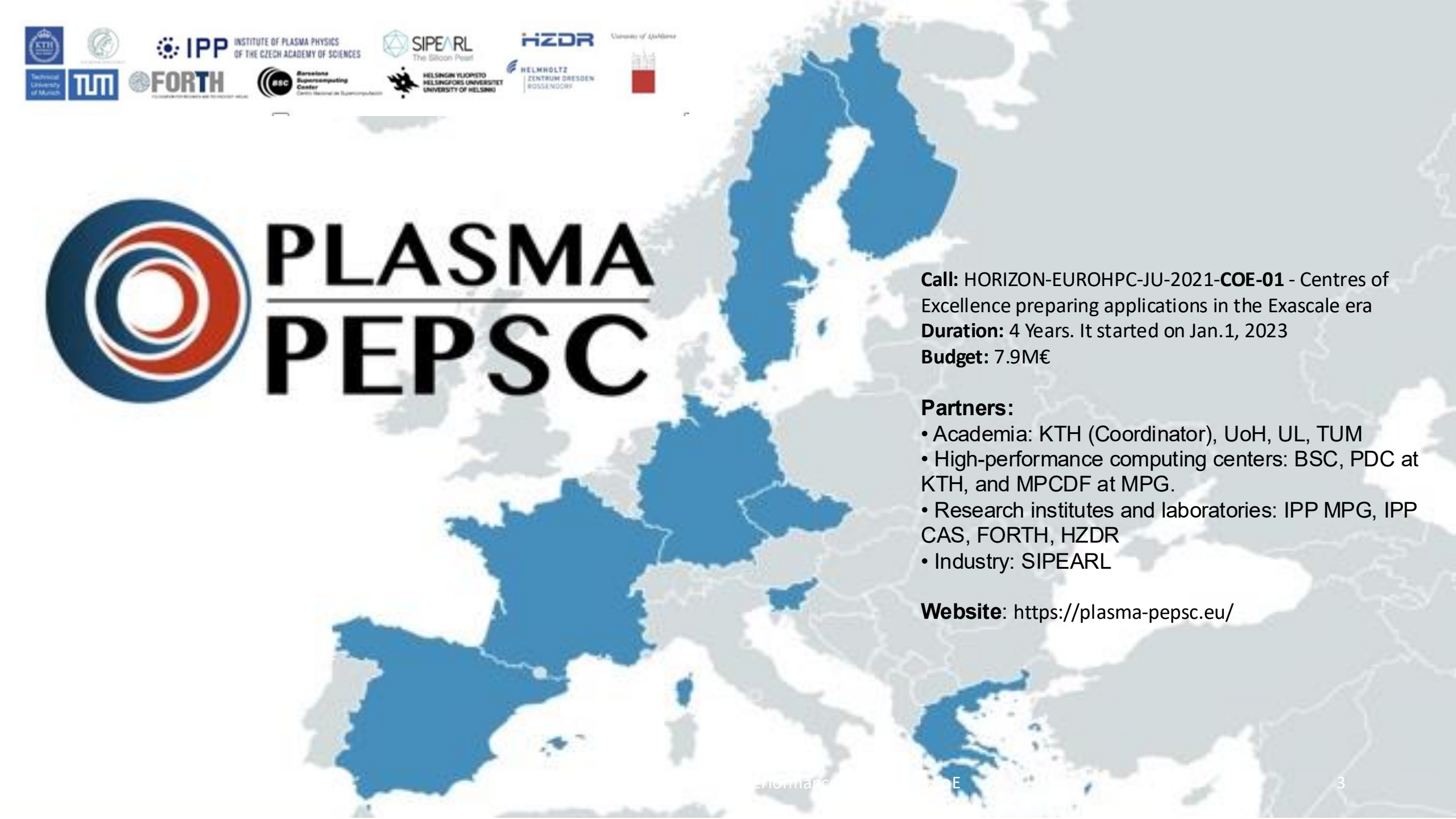
Plasma-PEPSC
Plasma Exascale-Performance Simulations CoE

EuroCC Greece - Training Event
September 29th 2025

Kallia Chronaki
ICS-FORTH
khronaki@ics.forth.gr

Outline

- The Plasma-PEPSC EuroHPC CoE
- Vision
- Four Plasma Simulation Codes
- Our Approach
- Conclusions

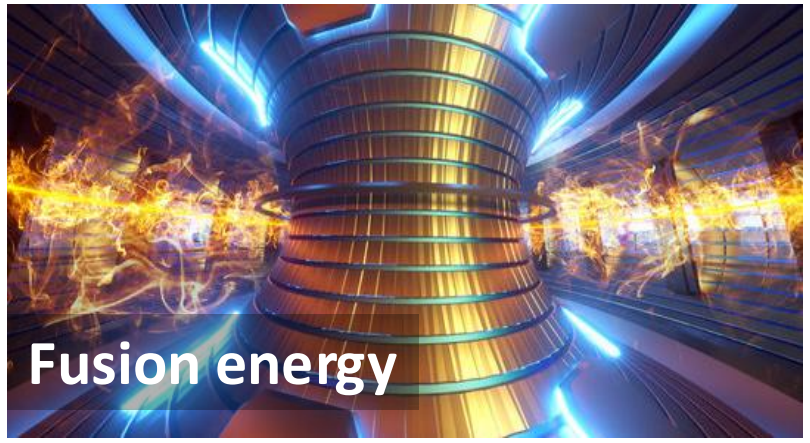


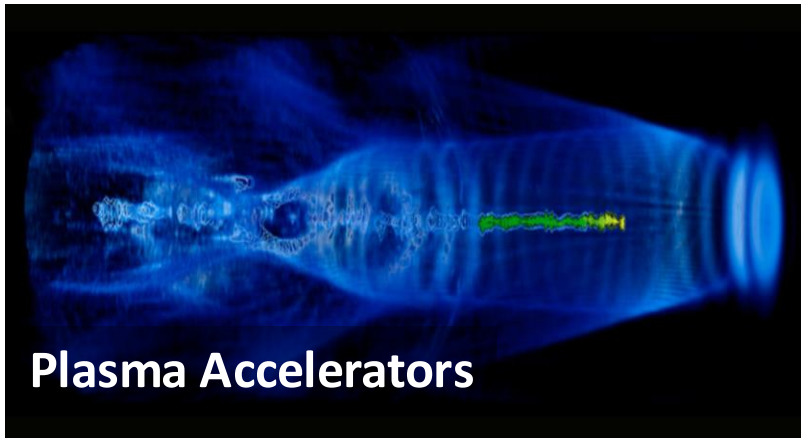
PLASMA PEPSC

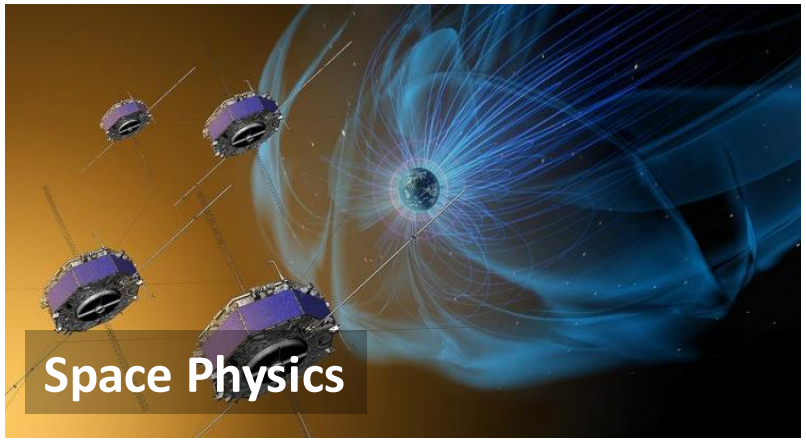
Call: HORIZON-EUROHPC-JU-2021-**COE-01** - Centres of Excellence preparing applications in the Exascale era
Duration: 4 Years. It started on Jan.1, 2023
Budget: 7.9M€

Partners:
- Academia: KTH (Coordinator), UoH, UL, TUM
- High-performance computing centers: BSC, PDC at KTH, and MPCDF at MPG.
- Research institutes and laboratories: IPP MPG, IPP CAS, FORTH, HZDR
- Industry: SIPEARL

Website: https://plasma-pepsc.eu/







Plasma-PEPSC Vision:

Pushing Flagship Plasma Simulation Codes to Tackle
...s via Performance
...esign

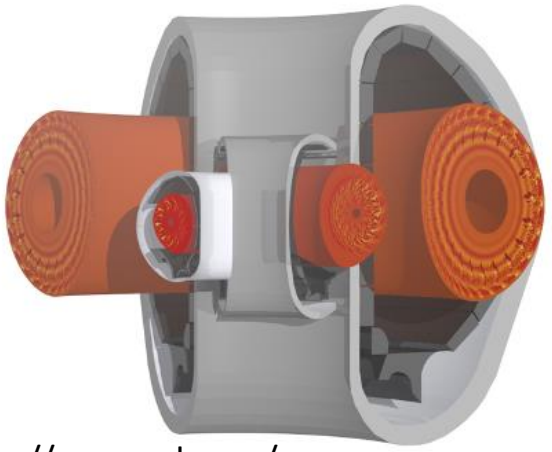
2 codes (BIT1, GENE) focus on simulations during the fusion energy production in the tokamaks

1 code (PIConGPU) focuses on the modelling of Plasma Accelerators → useful for medical applications (e.g. diagnostics, radiotherapy) instead of state of the art accelerators

1 code (Vlasiator) focuses on the modelling of the entire earth magnetosphere and interaction with plasma from the sun → useful for space weatherprediction and understanding plasma processes

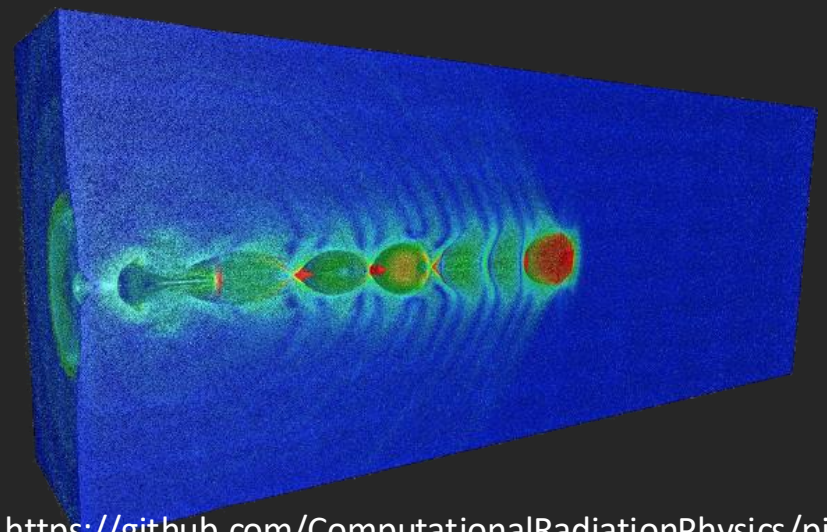
Plasma-PEPSC Flagship Codes

GENE / GENE-X



https://genecode.org/

PIConGPU

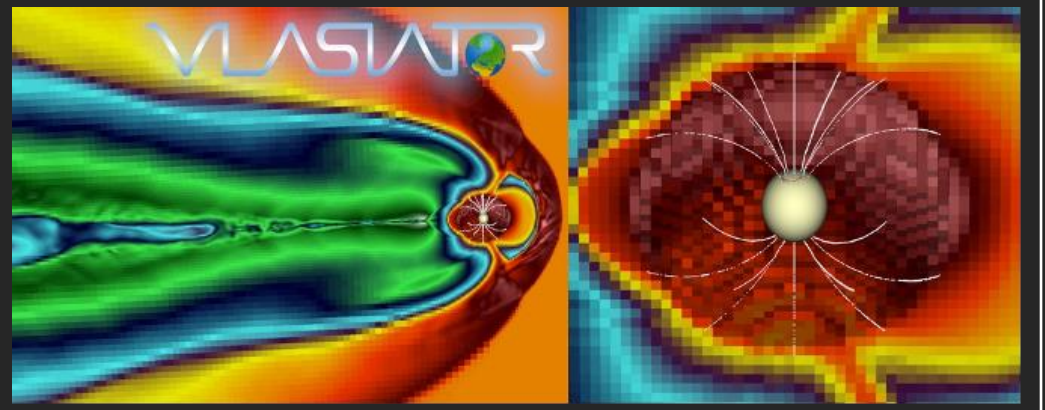


https://github.com/ComputationalRadiationPhysics/picongpu

BIT

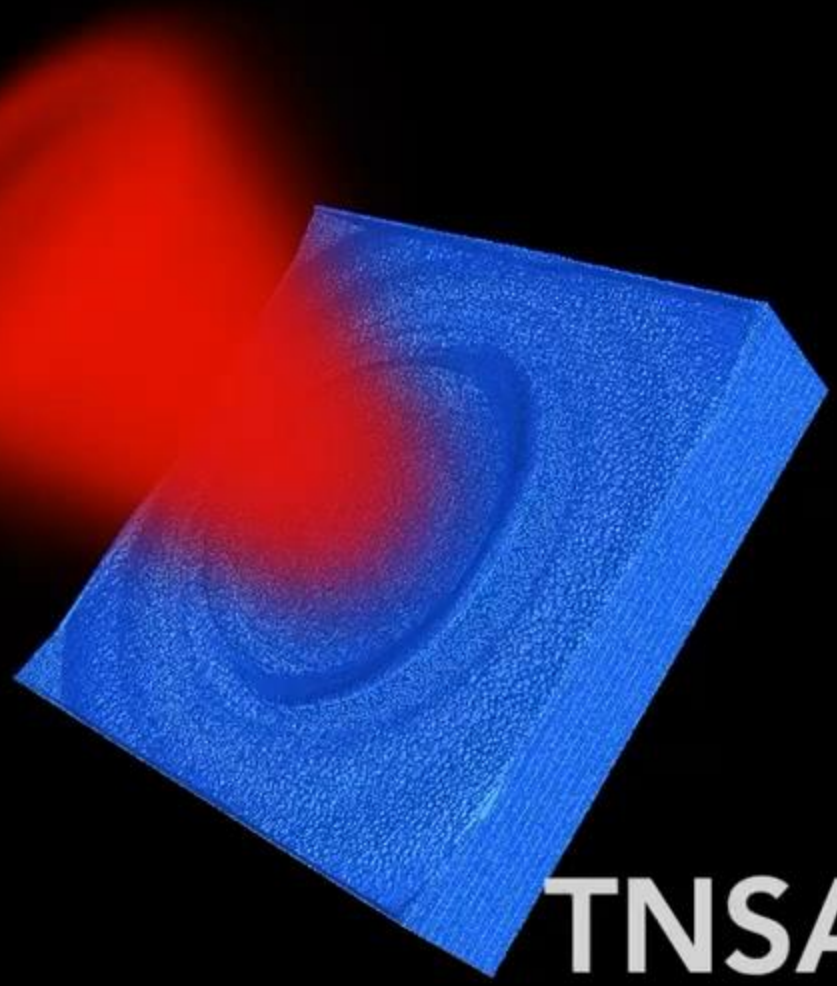
https://repo.tok.ipp.cas.cz/tskhakaya/bit1

Vlasiator



https://github.com/fmihpc/vlasiator

PIConGPU
ISAAC



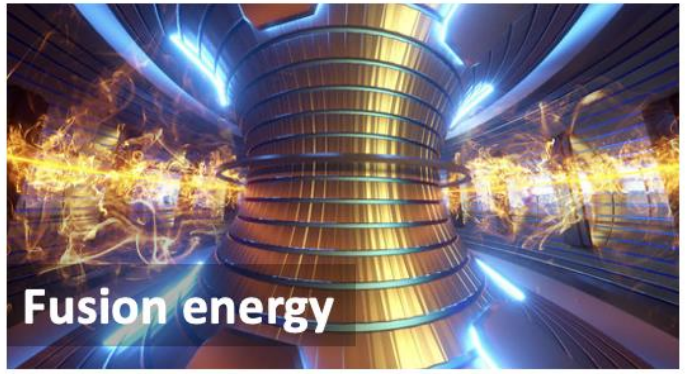
TNSA

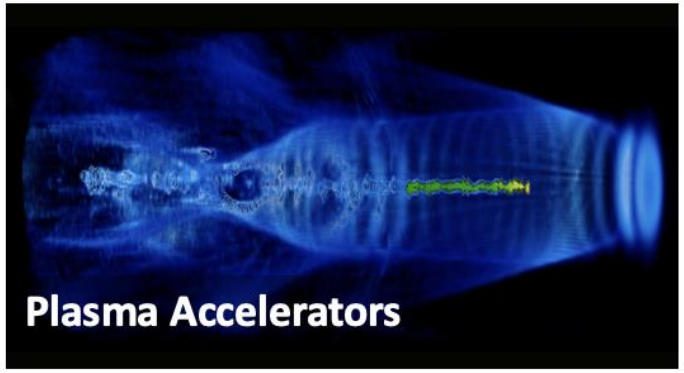
TARGET NORMAL SHEATH ACCELERATION

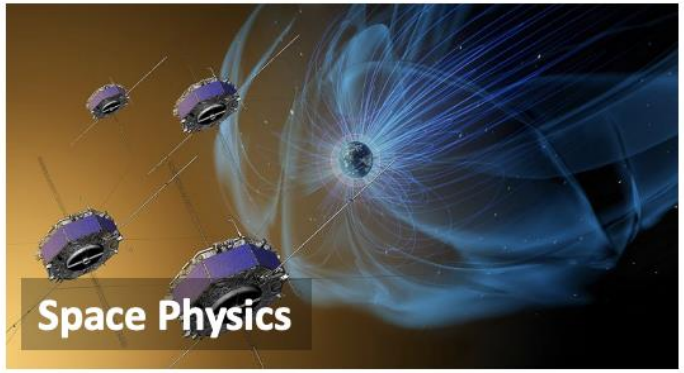
Name	Location	Peak Performance	More info
JUPITER	Forschungszentrum Jülich campus in Germany and operated by the Jülich Supercomputing Centre.	Expected 1 ExaFLOP. To become the first Europe's Exascale supercomputer	
LUMI	Kajaani, Finland, hosted by CSC – IT Center for Science	539.13 petaflops	https://lumi-supercomputer.eu/
LEONARDO	CINECA datacenter in Bologna, Italy	315.74 petaflops	https://leonardo-supercomputer.cineca.eu/
MARENOSTRUM 5	Hosted by BSC-CNS in Barcelona, Spain	314 petaflops	https://www.bsc.es/marenostrum/marenostrum-5
MELUXINA	Hosted by LuxProvide in Bissen, Luxembourg	18.29 petaflops	https://www.luxprovide.lu/meluxina/
KAROLINA	Hosted by IT4Innovations National Supercomputing Center located in Ostrava, Czechia	12.91 petaflops	https://docs.it4i.cz/karolina/hardware-overview/
DISCOVERER	Located in Sofia, Bulgaria hosted by hosted by Sofia Tech Park.	5.94 petaflops	https://docs.discoverer.bg/resource_overview.html
VEGA	Vega is a petascale EuroHPC supercomputer located in Maribor, Slovenia. It is supplied by Atos, based on the BullSequana XH2000 supercomputer and hosted by IZUM.	10.05 petaflops	https://doc.vega.izum.si/general-spec/
DEUCALION	Located in Guimarães, Portugal. Deucalion is hosted by FCT and managed by CNCA.	9.76 petaflops	https://docs.macc.fccn.pt/support

Five Technical WPs

WP1 - *Plasma Simulations – Codes and Grand Challenges*







WP2 - *Co-design of Plasma Simulation Codes with the European Processor and Accelerator*

- EPI Processor
- EPI Accelerator
- Quantum Computing

WP3 - *Algorithms and Libraries for Extreme-Scale Plasma Simulations*

- MPI
- Load-balancing
- Resilience & Fault-tolerance

WP4 - *Extreme Data Analytics for Plasma Simulations*

- Parallel I/O
- In-situ data analysis
- Compression

WP5 - *Accelerated Plasma Simulations on Heterogeneous Systems*

- Redesigning Algorithms, Porting, and Optimization for Accelerators
- Application Data Placement and Migration for Heterogeneous Memories

Methodology and BIT1 showcase

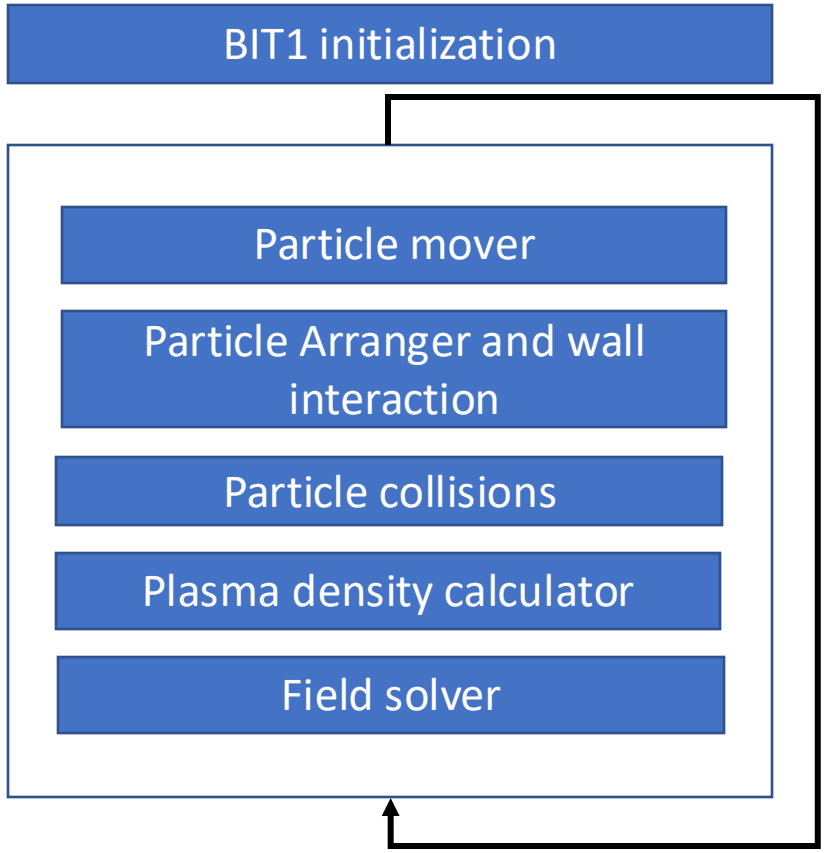
Methodology that we follow

- Analyse the code and figure out the hotspots:
  - Run the code with profiling tools and find the most time consuming parts
- Extract the most time consuming parts out of the application's code and optimize them
  - For example run one of the time consuming functions in isolation and optimize it in isolation
- Port the extracted parts back to the application and run the entire code

A case study: BIT1

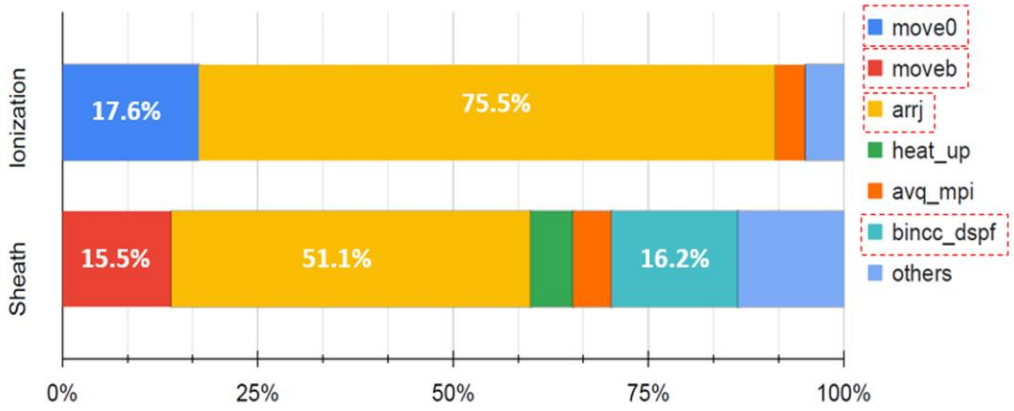
BIT1 overview

- BIT1 is an MPI Particle-In-Cell (PIC) application that simulates plasma behavior in the tokamak divertor such as in the ITER fusion device
  - PIC: a computational technique that models plasma
- After the BIT1 initialization there is an iterative process that performs the simulation the interactions between plasma and the walls within the tokamak



Step1: Analysis

- **gprof:** reports frequently used functions



arrj(): particle arrangement
move0()/moveb():particle mover
bincc_dspf(): binary collision operator

- **Maqao:** reports hotspots and their coverage in execution time

Code	Function	Coverage%
arranger.c: 33-134	arrj	36.93%
prof_mpi.c: 63-66	avq_0	22.36%
mover.c: 145-146	move0	20.24%
mover.c: 177-178	nmove	9.98%
pflmv_mpi.c: 19-20	accum_mpi	0.11%
prof_mpi.c: 43-44	avq_mpi	0.06%
arranger.c: 20-21	arrj	0.06%

[1] J. J. Williams et al. "Leveraging HPC Profiling & Tracing Tools to Understand the Performance of Particle-in-Cell Monte Carlo Simulations." Euro-Par 2023: Parallel Processing Workshops: Euro-Par 2023 International Workshops, Limassol, Cyprus, August 28–September 1, 2023, Revised Selected Papers, Part I, LNCS 14351

Step 2: Extraction and optimization (OpenMP)

OpenMP Tasks

```
#pragma omp parallel shared(chsp, sn2d, dinj, nstep, np, x, yp, yx, vy) \
    private(isp, i, j) firstprivate(nsp, nc)
{
#pragma omp single
 {
 for (isp = 0; isp < nsp; isp++) {
    ...
#pragma omp taskloop grainsize(500) nogroup
    for (j = 0; j < nc; j++) {
#pragma omp simd
      for (i = 0; i < np[isp][j]; i++)
        x[isp][j][i] += nstep[isp] * vx[isp][j][i];
        }
        ...
#pragma omp taskloop grainsize(500) nogroup
     for (j = 0; j < nc; j++)
#pragma omp simd
      for (i = 0; i < np[isp][j]; i++)
        x[isp][j][i] += nstep[isp] * vx[isp][j][i];
    }
  }
}
```

- Based on the analysis of the previous step we extract the mover function
- Parallelization of loops with OpenMP (work of KTH, CAS, UL)
- The loop iterations are split among the cores of each node to exploit the full node for each MPI rank

PLASMA PEPSC

[1] J. J. Williams, et al. “Optimizing BIT1, a Particle-in-Cell Monte Carlo Code, with OpenMP/OpenACC and GPU Acceleration.” 24th International Conference on Computational Science, Málaga, Spain, July 2-4, 2024, Part I, LNCS 14832. Springer Nature.

Step 2: Extraction and optimization (EPAC 1.5)

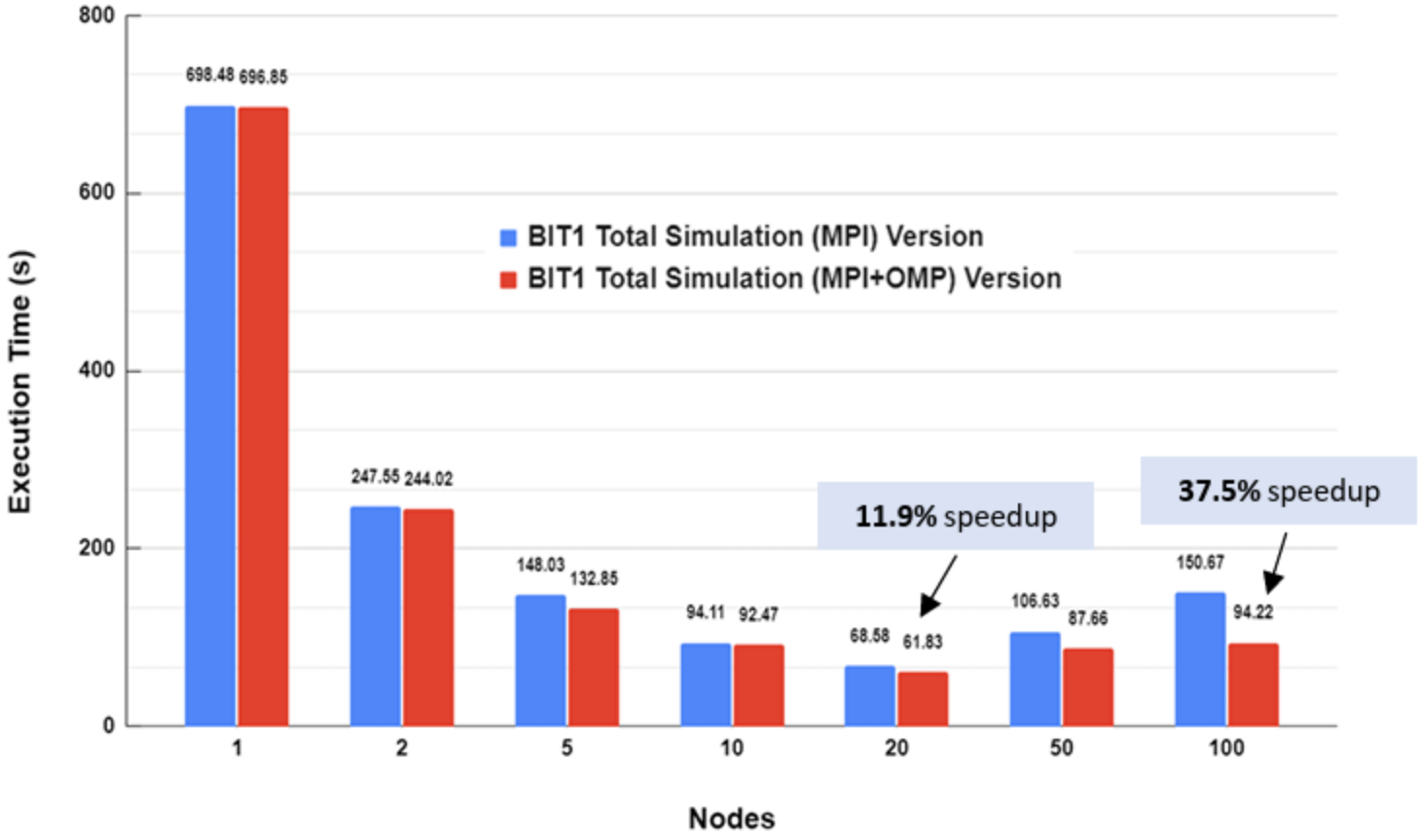
```
for(i = 0; i<np[j]; i++){
    sub_a = a[j+1] – a[j];
    atemp = a[j] + x[j][i] * sub_a;
    …
}
```

```
unsigned long gvl = __builtin_epi_vsetvlmax(__epi_e32, __epi_m1);
a_v = __builtin_epi_vbroadcast_2xf32(a[j], gvl);
a1_v = __builtin_epi_vbroadcast_2xf32(a[j+1], gvl);
for(i = 0; i<np[j] – gvl; i+=gvl)
{
    atemp_v = __builtin_epi_vload_2xf32(&x[j][i], gvl);
    a_v = __builtin_epi_vload_2xf32(&a[j], gvl);
    sub_a_v = __builtin_epi_vfsub_2xf32(a1_v, a_v, gvl);
    atemp_v = __builtin_epi_vfmul_2x_f32(atemp_v, sub_a_v, gvl);
    atemp_v = __builtin_epi_vfadd_2xf32(atemp_v, a_v, gvl);
    …
}
//The remaining vector length:
gvl_rem = __builtin_epi_vsetvli(np[j][i]-1, __epi_e32, __epi_m1);
atemp_v = __builtin_epi_vload_2x_f32(&x[j][i], gvl_rem);
sub_a_v = __builtin_epi_vfsub_2xf32(a1_v, a_v, gvl_rem);
atemp_v = __builtin_epi_vfmul_2x_f32(atemp_v, sub_a_v, gvl_rem);
atemp_v = __builtin_epi_vfadd_2xf32(atemp_v, a_v, gvl_rem);
…
```

Vector setup instructions

- **gvl**: granted vector length has to be set
- The loop runs with step **gvl** because it operates on **gvl** elements at a time
- For each instruction we need to load the data to vector registers first
- If the total loop iterations originally is not a multiple of **gvl,** we perform the rest computations on **gvl_rem** elements

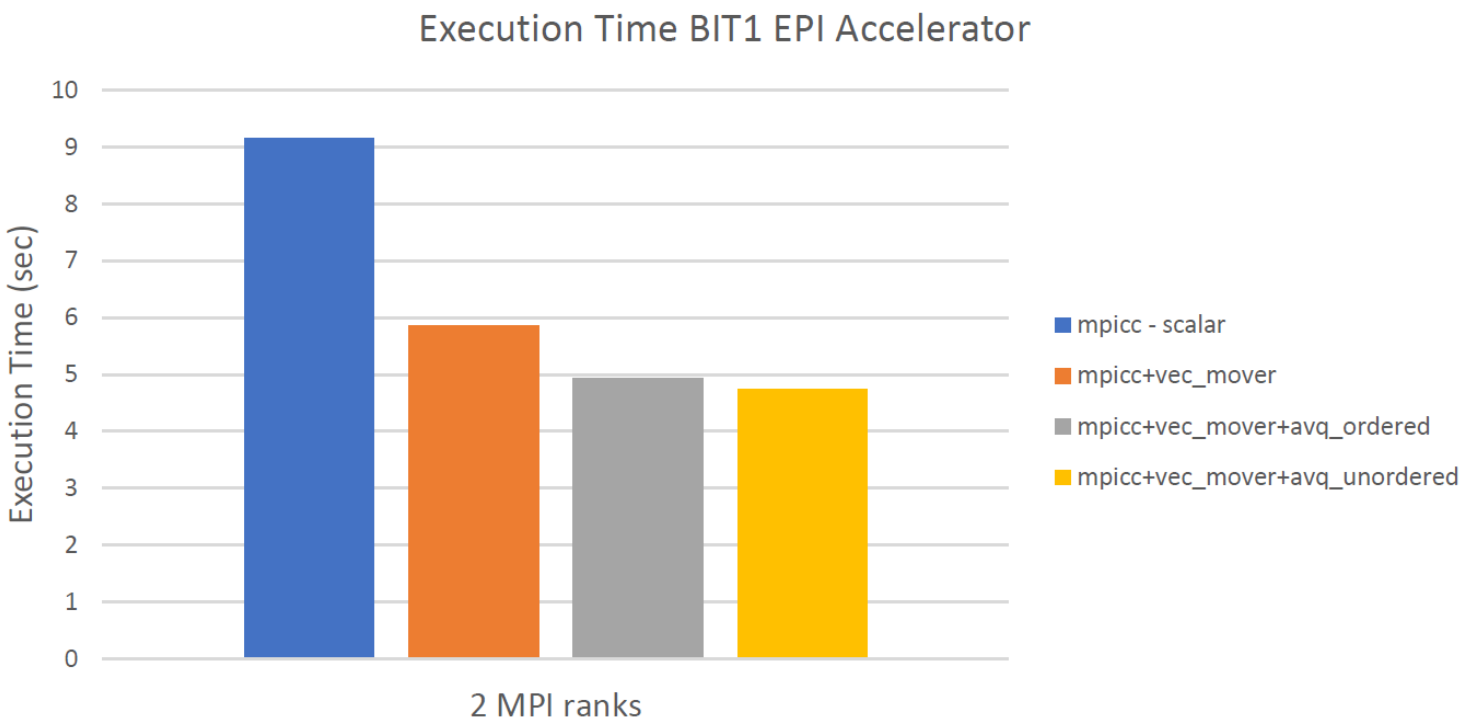
Step 3: Port back and evaluation (OpenMP)



- Improving total simulation performance by increasing intra-node parallelism (work of KTH, CAS, UL)
- Up to 37.5% speedup

[1] J. J. Williams et all. “Accelerating Particle-in-Cell Monte Carlo Simulations with MPI, OpenMP/OpenACC and Asynchronous Multi-GPU Programming”. Journal of Computational Science. Accepted For Publication.

Step 3: Port back and evaluation (EPAC 1.5)

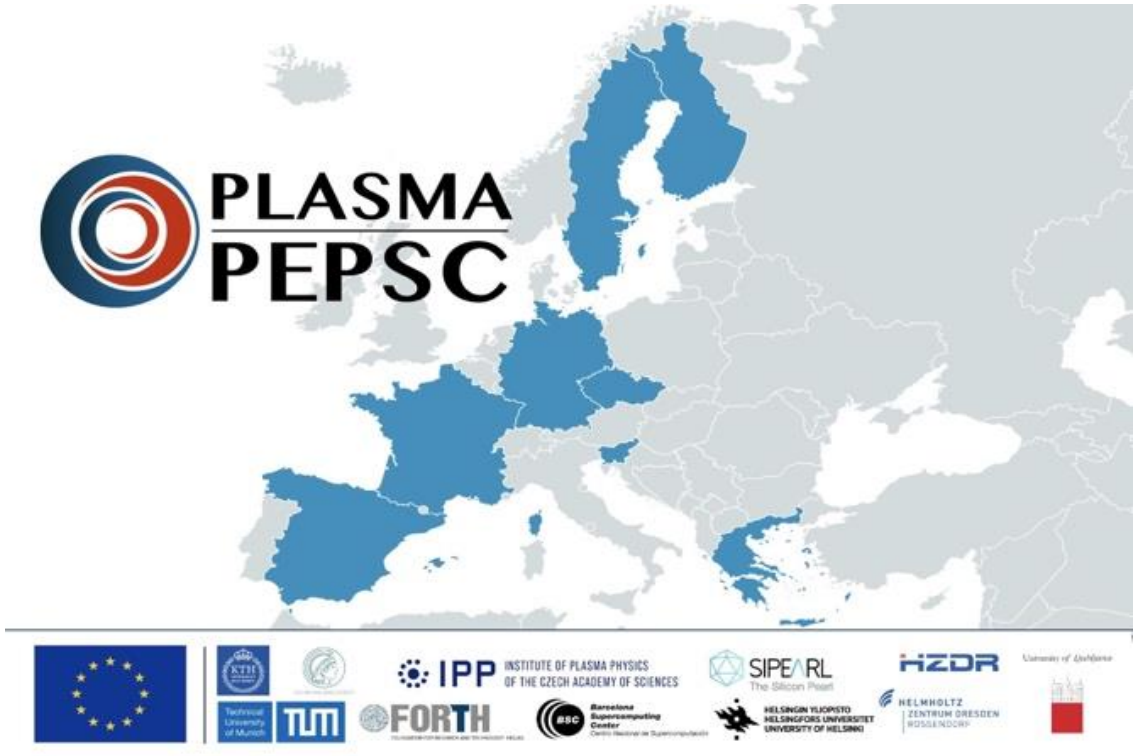


- Improving total simulation time by taking advantage of the vector unit of the EPAC 1.5 chip
- Achieved close to 2x speedup for 2 MPI ranks

Conclusions

- We are a EuroHPC center of excellence for Plasma simulations at Exascale
  - Four lighthouse plasma codes: BIT, GENE, PIConGPU, and Vlasiator
  - Addressing challenges at exascale: software engineering at scale, extreme heterogeneity, data deluge problem, algorithms and libraries

Acknowledgments



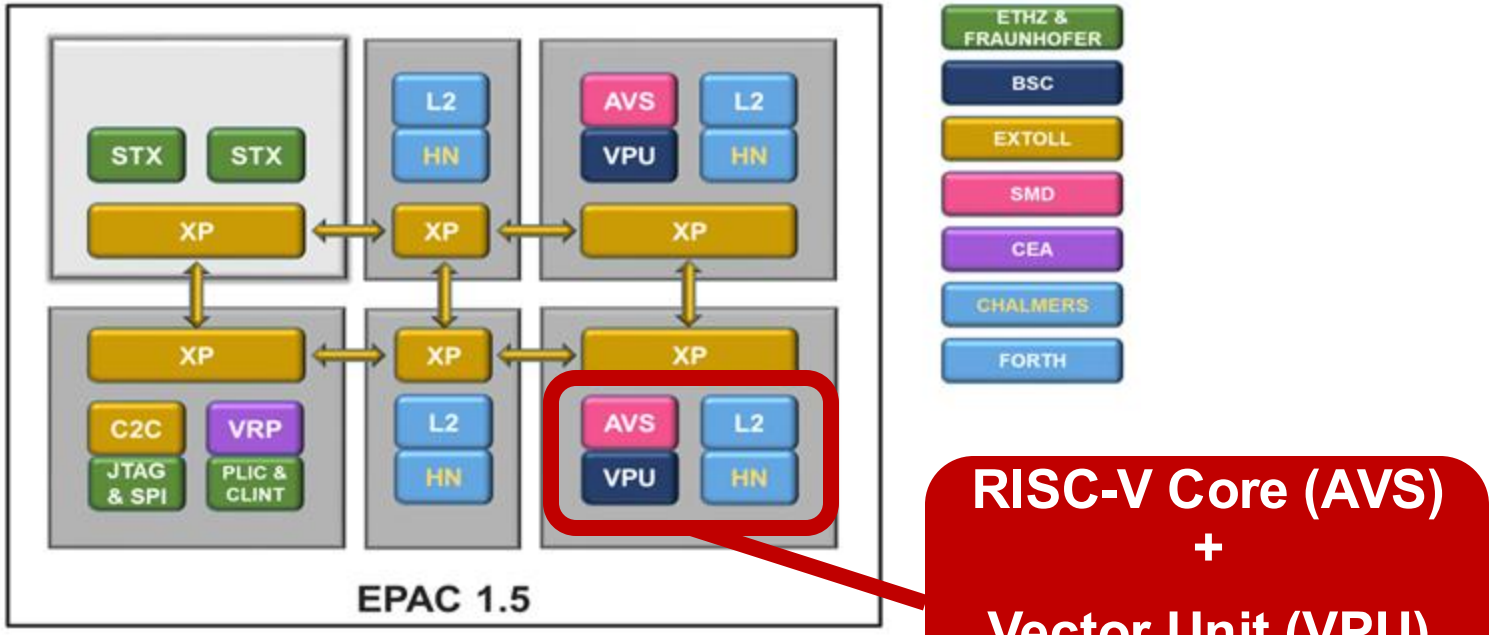
This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101093261. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Sweden, Germany, France, Spain, Finland, the Czech Republic, Slovenia, and Greece.

EuroHPC
Joint Undertaking

PLASMA
PEPSC

The Hardware: EPAC RISC-V Accelerator technology

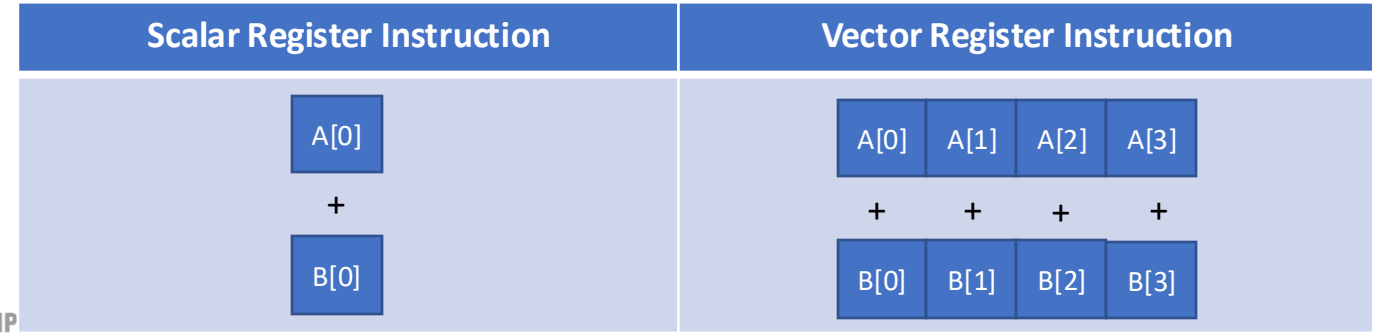
- Prototype chip implemented as a joint effort by different parties



RISC-V Core (AVS)
+
Vector Unit (VPU)
aka EPAC-VEC

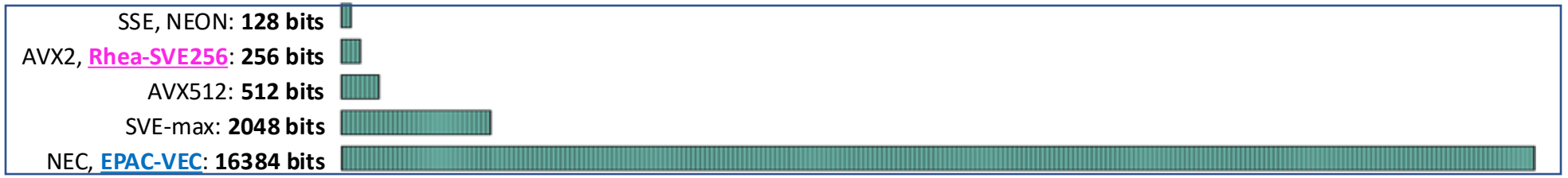
EPAC RISC-V Accelerator

- The EPAC RISC-V accelerator consists of a RISC-V core and a RISC-V Vector (RVV) unit
- **RISC-V core** (Avispado): a general purpose core that implements the RISC-V instruction set architecture (ISA), by Semidynamics
- **RISC-V vector**: a vector unit that implements the RISC-V vector extension (RVV), by BSC
  - The vector unit executes instructions on multiple data
  - One vector unit instruction corresponds to multiple similar instructions operated on several elements of the vector register
    - This is achieved by using the special **vector registers** that can store up to a number of elements each (instead of 1 element per register for the traditional scalar registers)

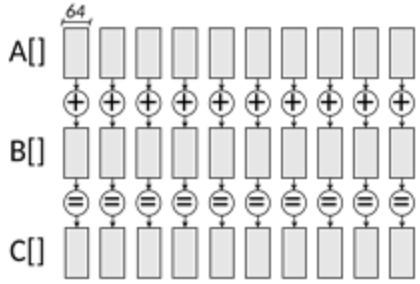


EPAC RISC-V Accelerator

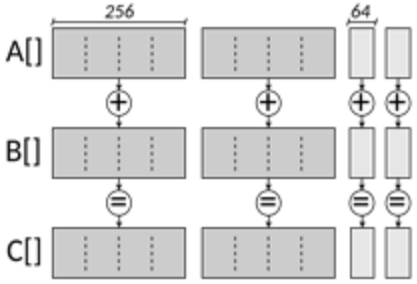
- Key features of the EPAC RISC-V Accelerator architecture:
  - Use of **very long vector registers** (**256 FP64** elements = 16,384 bits)
  - A **scalar** register (non vector) is 32 or 64 bit – depending on the system
  - Other **vector architectures** have the following vector register sizes:



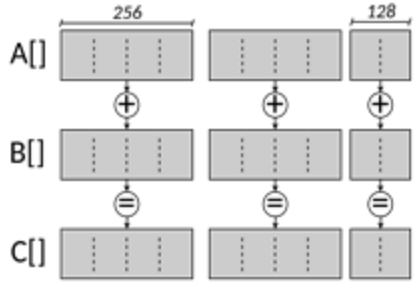
- Vector Length agnostic architecture: supports multiple sizes of vector length (can be set at runtime)



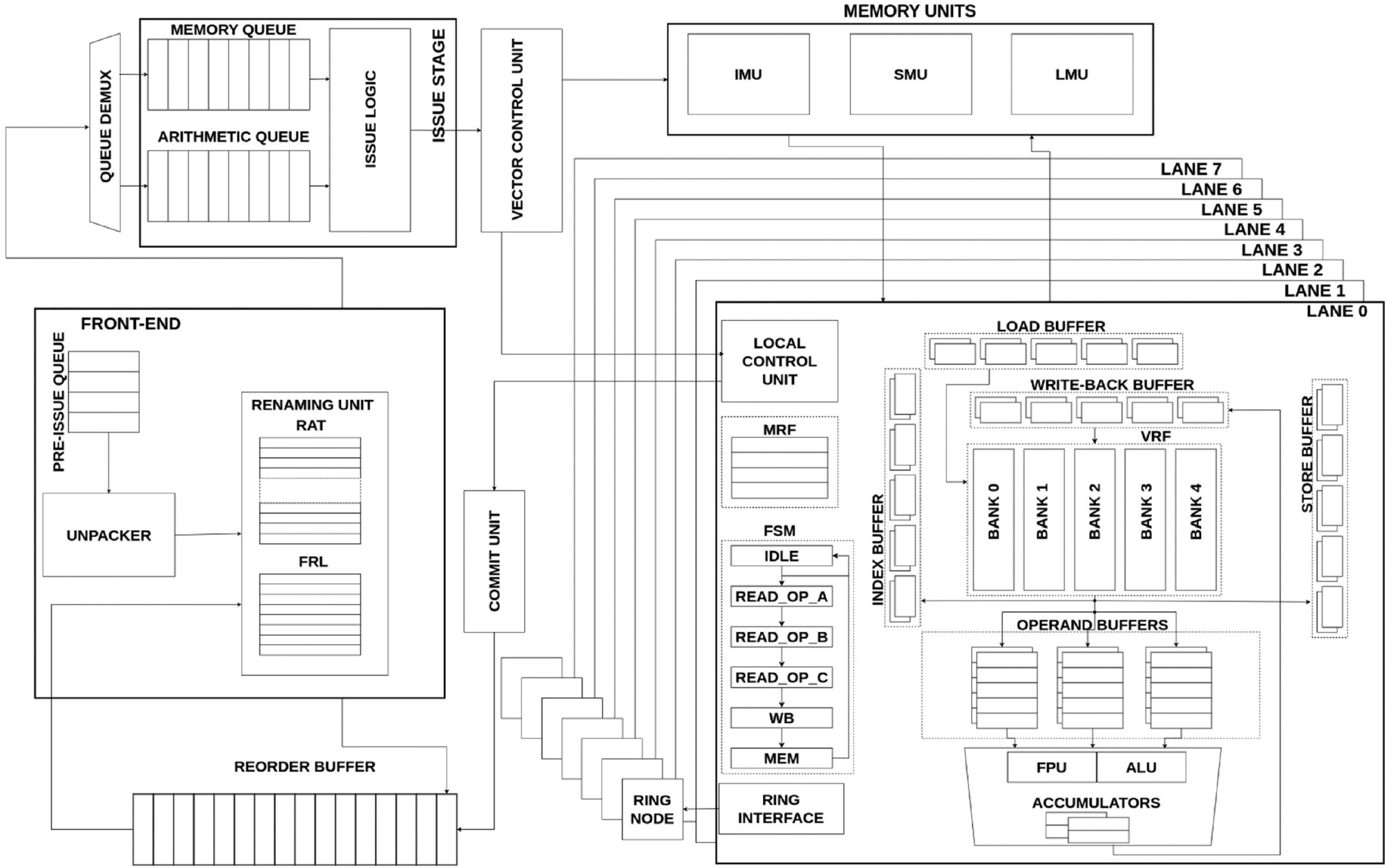
Scalar processor



SIMD (e.g., AVX2)



Variable VL (e.g., RVV)


MEMORY UNITS
IMU
SMU
LMU
ISSUE STAGE
QUEUE DEMUX
MEMORY QUEUE
ARITHMETIC QUEUE
ISSUE LOGIC
VECTOR CONTROL UNIT
LANE 7
LANE 6
LANE 5
LANE 4
LANE 3
LANE 2
LANE 1
LANE 0
FRONT-END
PRE-ISSUE QUEUE
UNPACKER
RENAMING UNIT
RAT
FRL
COMMIT UNIT
LOCAL CONTROL UNIT
MRF
FSM
IDLE
READ_OP_A
READ_OP_B
READ_OP_C
WB
MEM
RING NODE
RING INTERFACE
LOAD BUFFER
WRITE-BACK BUFFER
VRF
BANK 0
BANK 1
BANK 2
BANK 3
BANK 4
INDEX BUFFER
STORE BUFFER
OPERAND BUFFERS
FPU
ALU
ACCUMULATORS
REORDER BUFFER