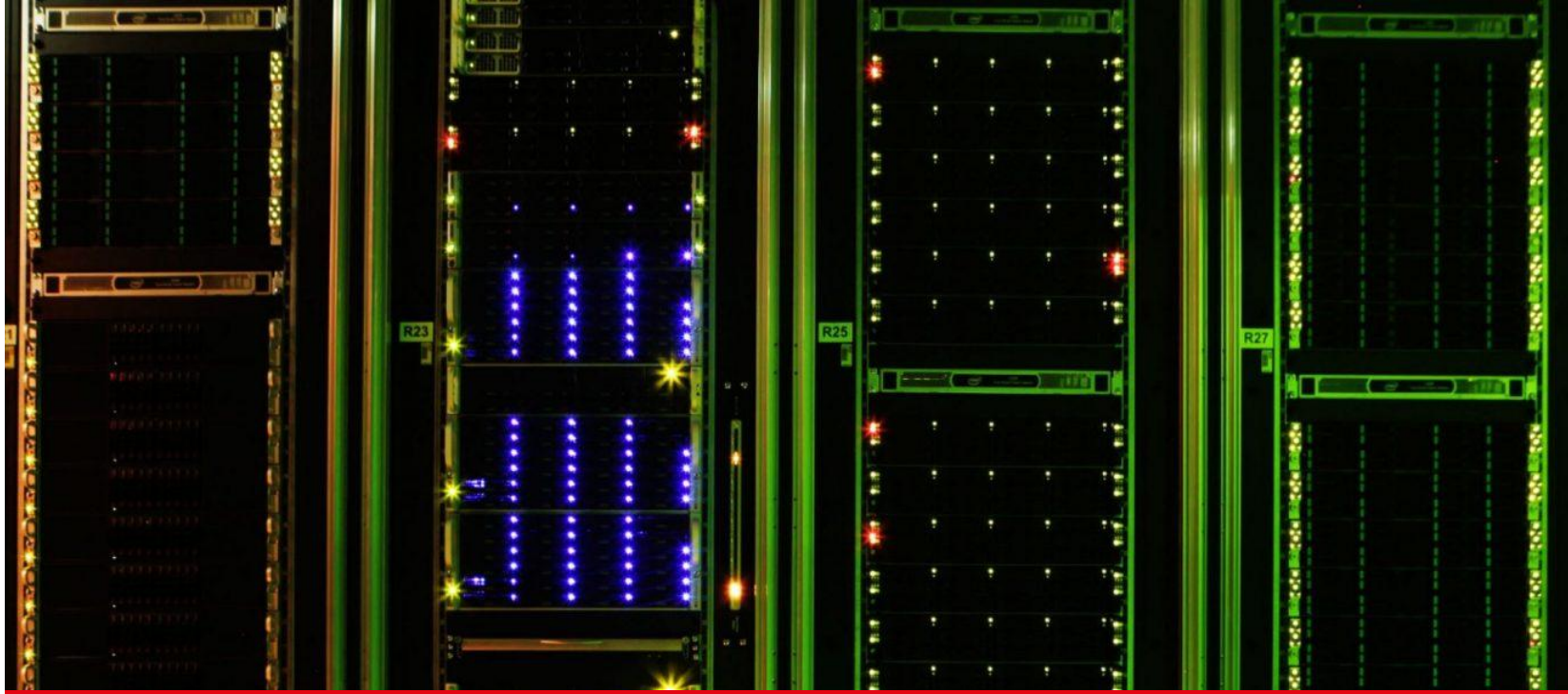
EPFL

EUROfusion



EPFL-ACH: Instrumentation and profiling for efficient GPU porting

SCientific IT and Application Support Platform (SCITAS)
Swiss Plasma Center (SPC)
EUROfusion Advanced Computing Hub

EUROfusion E-TASC meeting #2

SWISS PLASMA CENTER

EPFL SCITAS
Scientific IT and Application Support

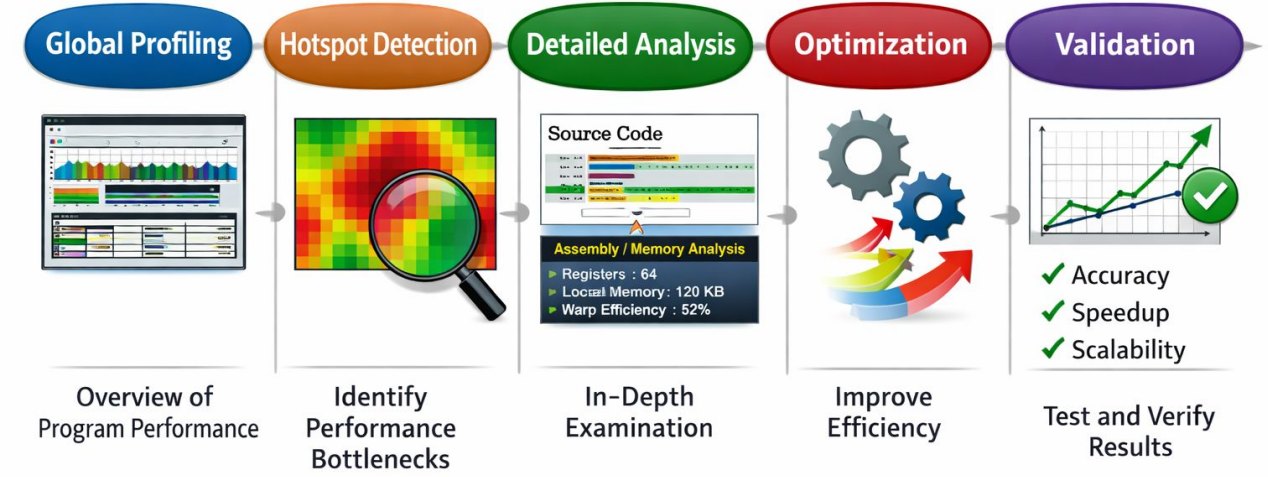
Agenda

- Introduction
- HPC Profiling Tools
- ASCOT5 Case Study
- SOLEDGE3X Case Study
- Mini-Apps
- Lessons Learned
- Conclusions

Introduction

- HPC performance challenges
- Motivation for profiling
- Objectives of this work

HPC Profiling Methodology



Overview of HPC Profiling Tools

- Intel VTune
- Intel Advisor
- NVIDIA Nsight Systems
- NVIDIA Nsight Compute
- AMD Rocprof
- Perf / Score-P
- TAU
- Extrae / Paraver

Profiling and Optimization Flags: Examples

- Intel oneAPI (icx / ifx / icpx):
  - -g, -O2 : Debug + profiling
  - -qopt-report=5 : Optimization report
  - -qopt-report-phase=vec,loop : Vectorization info
  - -qopt-zmm-usage=high : AVX-512 usage
  - -march=native / -xHost : CPU tuning

- NVIDIA NVHPC (nvc / nvc++ / nvfortran):
  - -g, -O2 : Debug + profiling
  - -Minfo=vec : Vectorization report
  - -Minfo=all : Full optimization info
  - -gpu=lineinfo : Nsight source mapping
  - -fast : Aggressive optimizations

- Recommended workflow:
- - Debug: -g -O0
- - Profile: -g -O2
- - Analyze SIMD: reports
- - Production: tuned flags

```
197, Generating implicit firstprivate(order,nz,ny,nx)
     Generating NVIDIA GPU code
    198, !$acc loop gang collapse(3) ! blockidx%x
    199,   ! blockidx%x collapsed
    200,   ! blockidx%x collapsed
    202, !$acc loop vector(128) ! threadidx%x
    204, !$acc loop seq
    208, !$acc loop vector(128) ! threadidx%x
    210, !$acc loop seq
    230, !$acc loop vector(128) ! threadidx%x
         Generating implicit reduction(+:alphapsum,alphamsum)
    238, !$acc loop seq
197, Generating implicit copyout(ur(order-1:nz-order,:ny,:nx)) [if not already present]
     Generating implicit copyin(var(:,:ny,:nx),dminus(:order-1),crj(:order-1,:order-1))
     Generating implicit copyout(ul(order:nz-order+1,:ny,:nx)) [if not already present]
     Generating implicit copyin(dplus(:order-1)) [if not already present]
198, Loop not fused: no successor loop
     Loop not vectorized/parallelized: too deeply nested
199, Loop not vectorized/parallelized: too deeply nested
200, Generating implicit firstprivate(alphapsum,alphamsum)
     Invariant if transformation
     Loop not fused: no successor loop
     FMA (fused multiply-add) instruction(s) generated
202, Loop is parallelizable
     Generating implicit firstprivate(j)
     Loop not fused: enclosed in different number of nests
     2 loops fused
     Loop not fused: enclosed in different number of nests
     2 loops fused
     Loop not fused: enclosed in different number of nests
     2 loops fused
204, Complex loop carried dependence of reconstm prevents parallelization
     Loop carried reuse of reconstm prevents parallelization
     Loop not fused: possible intervening definition
     Generated vector simd code for the loop containing reductions
```

ASCOT5 Overview

- Plasma simulation code
- CPU and GPU kernels
- Main performance bottlenecks

ASCOT5

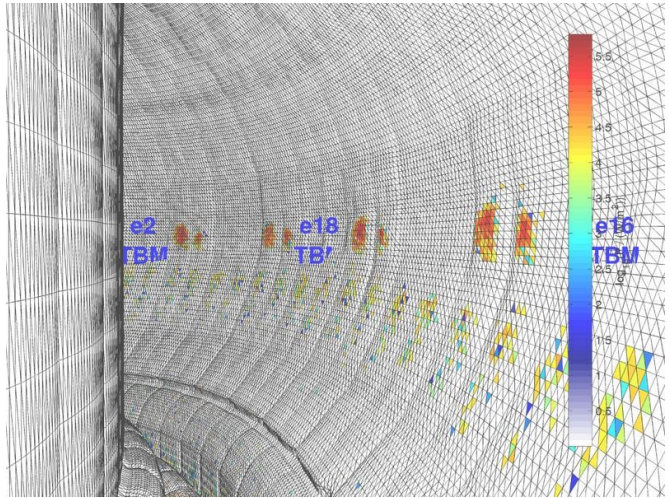
- ASCOT5 is a test **particle orbit-following** code for toroidal magnetically confined fusion devices
- The code uses the **Monte Carlo method** to solve the distribution of particles by following their trajectories.
  - The **evolution of the distribution function** for a test particle species *a*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

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

- 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**



ASCOT5 – CPU version

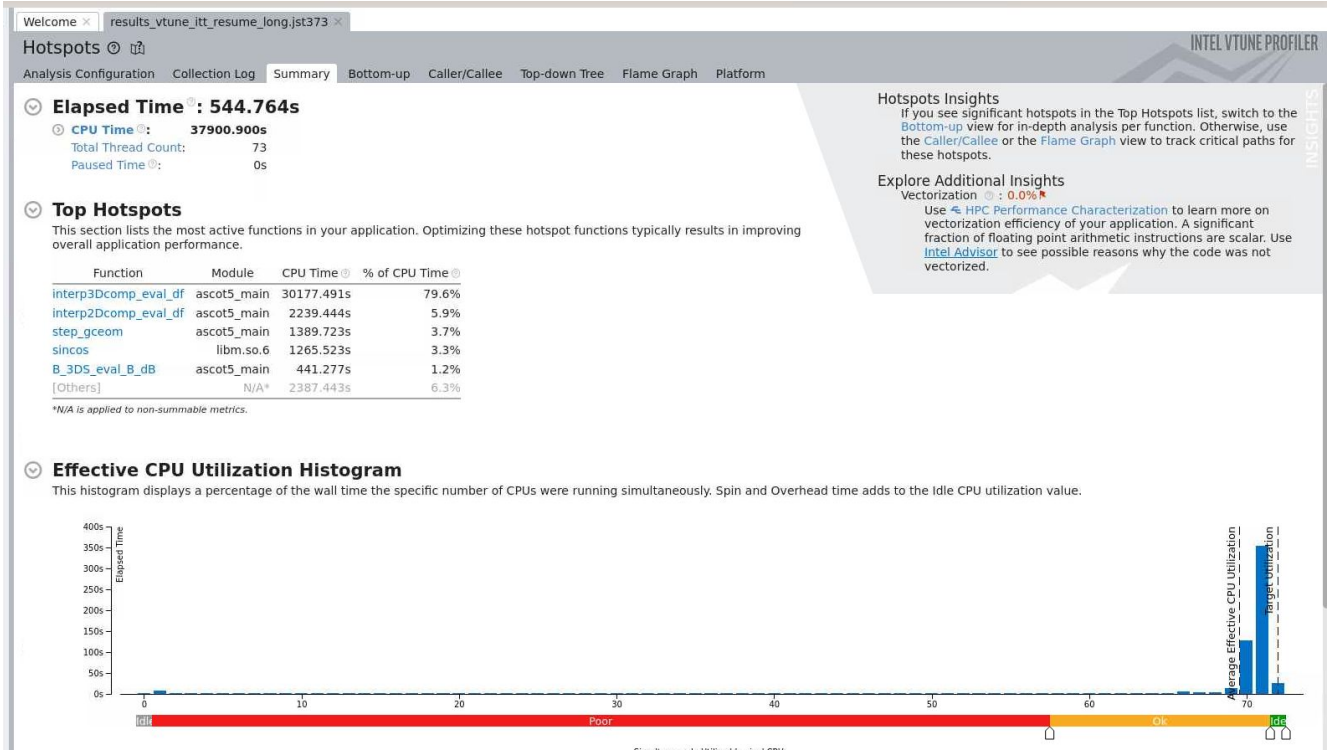
- CPU: MPI - OpenMP - Vectorized implementation:
  - The time evolutions of each particle are independent from each other, particles having different lifetimes
  - One + two levels of parallelism:
  - MPI: Particles distributed among tasks, fields replicated
  - OpenMP: queue based approach
  - highly vectorized using the SIMD, originally developed for KNL manycore systems as target
  - to enable multithreading, a number of worker threads, each operating on a single set of N_{SIMD} arrays, are launched and allowed to perform their simulation independently
  - swapping mechanism
    - after each iteration, particles that have reached their end condition are stored in an array for completed particles
    - a fresh particle is retrieved from a queue to continue simulation in the particular slot in the N_{SIMD} arrays

Algorithm 1: CPU multithread vectorized algorithm

```
initialization;
#pragma omp parallel
while particles are alive in pack_{N_SIMD} do
    #pragma omp simd
    for particles ∈ pack_{N_SIMD} do
        move_particle;
    end
    #pragma omp simd
    for particles ∈ pack_{N_SIMD} do
        collisions;
    end
    #pragma omp simd
    for particles ∈ pack_{N_SIMD} do
        end_condition;
    end
    #pragma omp simd
    for particles ∈ pack_{N_SIMD} do
        diagnostics;
    end
    for particles ∈ pack_{N_SIMD} do
        if particle reached end condition then
            store particle and replace it by new one
        end
    end
end
```

ASCOT5 - CPU profiling

- Intel VTune profiling
  - srun vtune -collect hotspots -result-dir ./results_vtune3 -- ./ascot5

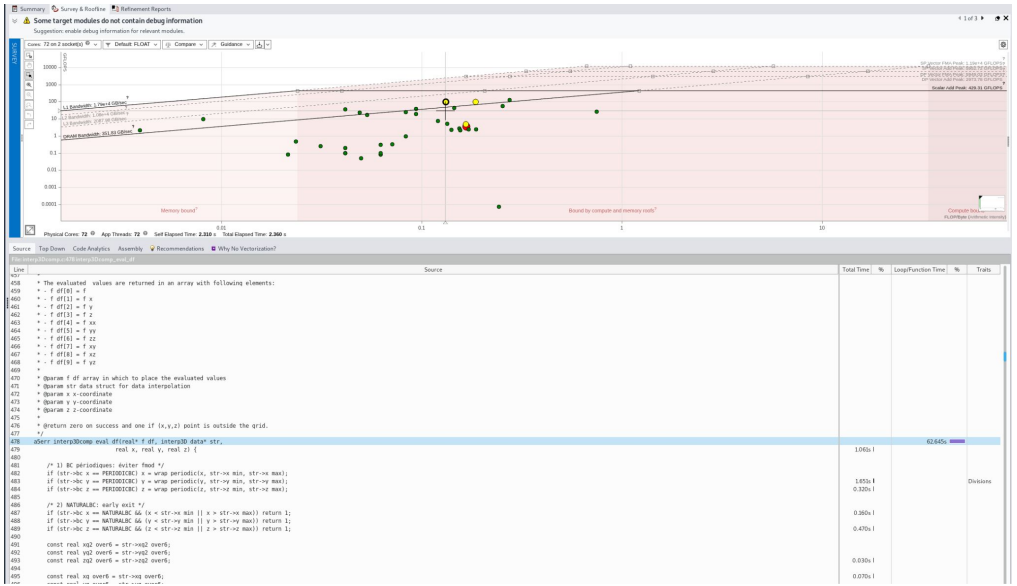


ASCOT5 - CPU profiling

- Use of Intel-Advisor on ASCOT5:

srun advisor --collect=survey --project-dir=./advi_results -- ./ascot5

srun advisor --collect=tripcounts --flop --project-dir=./advi_results -- ./ascot5



ASCOT5 - GPU version

- GPU porting strategy
  - Maintain a single version of the code
  - Ensure code portability and readability
  - Generic pragma for OpenMP/OpenACC

```
#ifndef gpu_commands
#define gpu_commands
/**
 * @brief Applies parallel execution to loops
 */
#if defined(GPU) && defined(OPENMP)
#define GPU_PARALLEL_LOOP_ALL_LEVELS\
        str_pragma(omp target teams distribute parallel for simd
#elif defined(GPU) && defined(OPENACC)
#define GPU_PARALLEL_LOOP_ALL_LEVELSstr_pragma(acc parallel loop)
#else
#define GPU_PARALLEL_LOOP_ALL_LEVELSstr_pragma(omp simd)
#endif


/**
 * @brief Maps variables to the target device
 */
#if defined(GPU) && defined(OPENMP)
#define GPU_MAP_TO_DEVICE(...) \
        str_pragma(omp target enter data map(to: __VA_ARGS__))
#elif defined(GPU) && defined(OPENACC)
#define GPU_MAP_TO_DEVICE(...) str_pragma(acc enter data copyin
(__VA_ARGS__))
#else
#define GPU_MAP_TO_DEVICE(...)
#endif
............

#endif
#endif
```

```
  GPU_LOOP_ALL_LEVELS
  for(i = 0; i < n_queue_size; i++) {
    if(p->running[i]) {
        posxyz[0] = posxyz0[0] + pxyz[0] * h[i] / (2.0 * gamma *
mass);
        posxyz[1] = posxyz0[1] + pxyz[1] * h[i] / (2.0 * gamma *
mass);
        posxyz[2] = posxyz0[2] + pxyz[2] * h[i] / (2.0 * gamma *
mass);
  }
  GPU_END_LOOP_ALL_LEVELS
```

ASCOT5 - GPU version

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

Algorithm 2: GPU algorithm - History-based

```
initialization;
#pragma acc parallel loop
for all particles ∈ {1...N_tot} do
    while particle is alive do
        move_particle;
        collisions;
        end_condition;
        diagnostics;
    end
end
```

Algorithm 3: GPU algorithm - Event-based

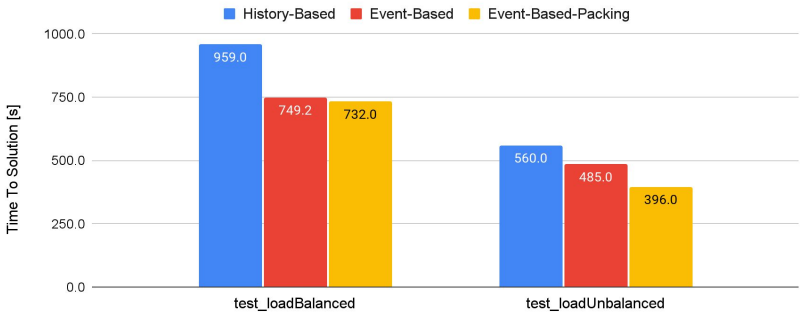
```
initialization;
while number of particles alive > 0 do
    #pragma acc parallel loop
    for all particles ∈ {1...N_tot} do
        if particle alive then
            move_particle;
        end
    end
    #pragma acc parallel loop
    for all particles ∈ {1...N_tot} do
        if particle alive then
            collisions;
        end
    end
    #pragma acc parallel loop
    for all particles ∈ {1...N_tot} do
        if particle alive then
            end_condition;
        end
    end
    #pragma acc parallel loop
    for all particles ∈ {1...N_tot} do
        if particle alive then
            diagnostics;
        end
    end
end
```

Algorithm 4: GPU algorithm - Event-based - packing

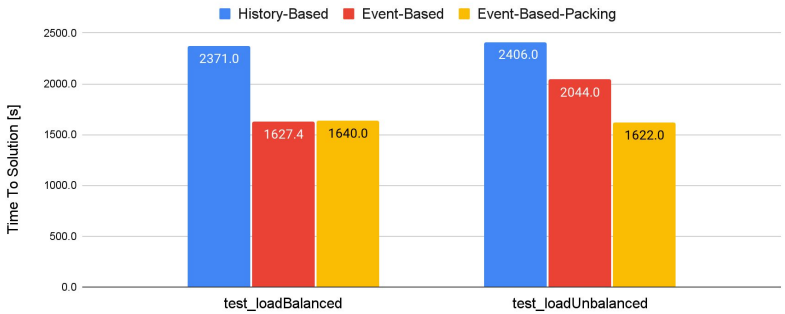
```
initialization;
N_pack ← N_tot;
while number of particles alive > 0 do
    #pragma acc parallel loop
    for packed particles still alive ∈ {1...N_pack} do
        move_particle;
    end
    #pragma acc parallel loop
    for packed particles still alive ∈ {1...N_pack} do
        collisions;
    end
    #pragma acc parallel loop
    for packed particles still alive ∈ {1...N_pack} do
        end_condition;
    end
    #pragma acc parallel loop
    for packed particles still alive ∈ {1...N_pack} do
        diagnostics;
    end
    if (N_pack − N_running > α · N_tot) then
        pack particles;
        N_pack ← N_running;
    end
end
```

Benchmarks

- Benchmark:
  - Spline interpolation of a analytical ITER circular equilibrium bfield
  - 2D wall rectangular, No coulomb collisions, gyro orbit, simulation time = 0.0001s, fixed time step
  - **Leonardo**: A100
  - Comparison of three GPU implementations on GPU A100
    - Event-based packing algorithm is most efficient in all cases
    - Impact of Packing:
      - test_loadBalanced: Minimal impact due to majority of particles reaching end of simulation
      - test_loadUnbalanced: Significant impact with speedup of up to 1.41 compared to history-based algorithm and up to 1.22 compared to event-based one.



Comparison of the 3 particle-following GPU implementations - 1 Millions markers - 1 A100



Comparison of the 3 particle-following GPU implementations - 10 Millions markers - 4 A100

Profiling

- **srun nsys profile --stats=true -t cuda,openacc**

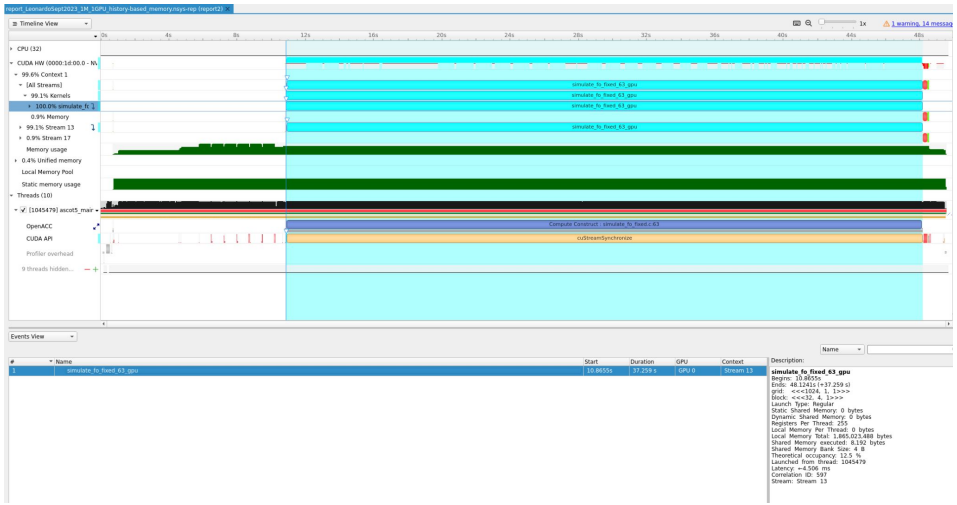
Time (%)	Total Time (ns)	Instances	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
69.6	293,966,172,960	102,042	**2,880,835.1**	3,059,920.0	48,880	3,237,080	534,642.4	**step_fo_vpa_38_gpu**
7.4	31,226,396,320	102,042	306,015.1	322,440.0	10,280	473,640	52,712.6	simulate_fo_fixed_125_gpu
5.8	24,687,683,640	102,042	241,936.5	251,920.0	23,560	463,040	37,128.6	endcond_check_fo_88_gpu
3.6	15,089,905,240	102,042	147,879.4	154,200.0	16,960	171,000	23,098.7	dist_COM_update_fo_90_gpu
3.1	13,018,526,000	102,042	127,580.1	135,040.0	7,920	161,640	28,709.7	dist_COM_update_fo_127_gpu
2.1	8,935,938,360	102,042	87,571.2	92,200.0	7,960	95,440	15,142.1	dist_rho5D_update_fo_129_gpu
1.9	8,172,358,320	102,042	80,088.2	84,040.0	7,960	87,000	13,206.1	dist_5D_update_fo_126_gpu
1.8	7,472,436,880	102,042	73,229.0	76,920.0	8,080	80,760	12,332.8	dist_rho6D_update_fo_129_gpu
1.7	7,021,646,480	102,042	68,811.3	72,200.0	8,000	76,920	11,463.7	dist_6D_update_fo_127_gpu
0.7	2,986,112,440	102,042	29,263.6	30,360.0	7,920	47,800	4,277.9	simulate_fo_fixed_203_gpu
0.3	1,379,537,280	102,042	13,519.3	13,760.0	7,920	15,400	1,057.5	dist_rho5D_update_fo_191_gpu
0.3	1,361,297,040	102,042	13,340.6	13,600.0	7,880	15,040	1,061.3	dist_rho6D_update_fo_182_gpu
0.3	1,359,132,040	102,042	13,319.3	13,560.0	7,920	14,960	1,005.6	dist_5D_update_fo_180_gpu
0.3	1,342,167,360	102,042	13,153.1	13,440.0	7,880	15,000	1,046.7	dist_6D_update_fo_174_gpu
0.3	1,187,350,360	102,042	11,635.9	11,720.0	7,760	13,280	500.3	simulate_fo_fixed_133_gpu
0.3	1,156,788,280	102,042	11,336.4	11,440.0	7,400	13,280	455.5	simulate_fo_fixed_160_gpu
0.2	946,637,680	102,042	9,276.9	9,280.0	8,480	12,640	148.3	simulate_fo_fixed_283_gpu
0.2	927,622,400	102,042	9,090.6	9,080.0	8,520	11,960	136.8	simulate_fo_fixed_283_gpu__red
0.0	465,080	1	465,080.0	465,080.0	465,080	465,080	0.0	simulate_fo_fixed_321_gpu
0.0	8,560	1	8,560.0	8,560.0	8,560	8,560	0.0	simulate_fo_fixed_316_gpu

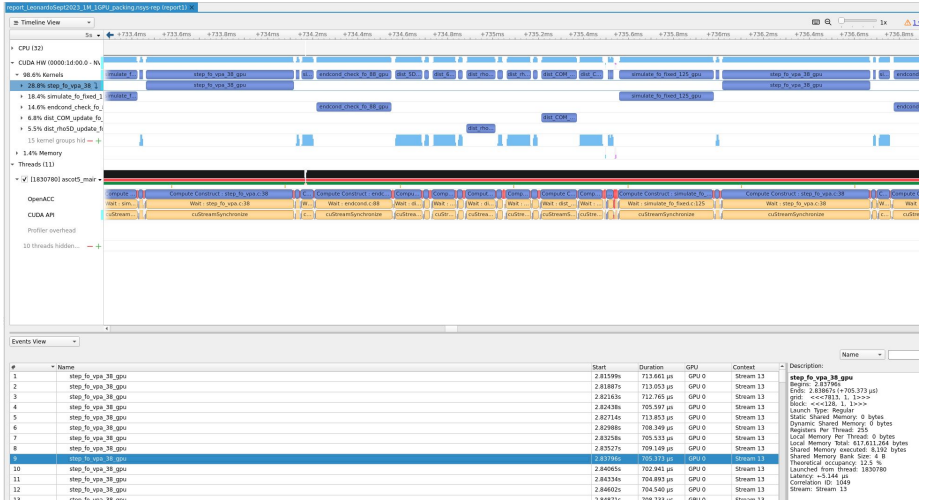
[7/8] Executing 'cuda_gpu_mem_time_sum' stats report

Time (%)	Total Time (ns)	Count	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Operation
45.7	195,716,120	102,148	1,916.0	1,520.0	1,080	639,600	13,689.2	[CUDA memcpy DtoH]
29.4	125,764,080	219	574,265.2	674,360.0	1,440	811,400	218,978.6	[CUDA memcpy HtoD]
24.9	106,789,160	102,043	1,046.5	1,040.0	960	2,200	27.4	[CUDA memset]

Profiling

- 1M Benchmark:
  - Spline interpolation of a analytical ITER circular equilibrium bfield
  - 2D wall rectangular, No coulomb collisions, gyro orbit, simulation time = 0.0001s, fixed time step
  - **Jed**: 2x Platinum 8360Y, intel/2021.6.0
  - **Pitagora**: H100
  - srun nsys profile -t cuda,openacc ...





Benchmarks

- Benchmark:
  - Spline interpolation of a analytical ITER circular equilibrium bfield
  - 2D wall rectangular, No coulomb collisions, gyro orbit, simulation time = 0.0001s, fixed time step
  - **Pitagora**: H100

History-Based

Description:

simulate_fo_fixed_63_gpu
Begins: 10.8655s
Ends: 48.1241s (+37.259 s)
grid: <<<1024, 1, 1>>>
block: <<<32, 4, 1>>>
Launch Type: Regular
Static Shared Memory: 0 bytes
Dynamic Shared Memory: 0 bytes
Registers Per Thread: 255
Local Memory Per Thread: 0 bytes
Local Memory Total: 1,865,023,488 bytes
Shared Memory executed: 8,192 bytes
Shared Memory Bank Size: 4 B
Theoretical occupancy: 12.5 %
Launched from thread: 1045479
Latency: ←4.506 ms
Correlation ID: 597
Stream: Stream 13

Event-Based

Description:

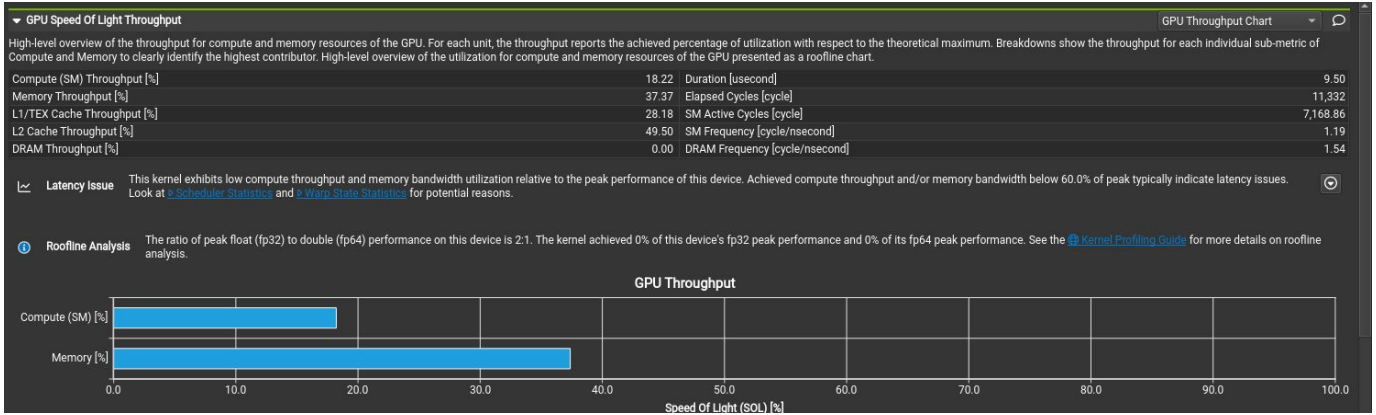
step_fo_vpa_38_gpu
Begins: 2.83796s
Ends: 2.83867s (+705.373 µs)
grid: <<<7813, 1, 1>>>
block: <<<128, 1, 1>>>
Launch Type: Regular
Static Shared Memory: 0 bytes
Dynamic Shared Memory: 0 bytes
Registers Per Thread: 255
Local Memory Per Thread: 0 bytes
Local Memory Total: 617,611,264 bytes
Shared Memory executed: 8,192 bytes
Shared Memory Bank Size: 4 B
Theoretical occupancy: 12.5 %
Launched from thread: 1830780
Latency: ←5.144 µs
Correlation ID: 1049
Stream: Stream 13

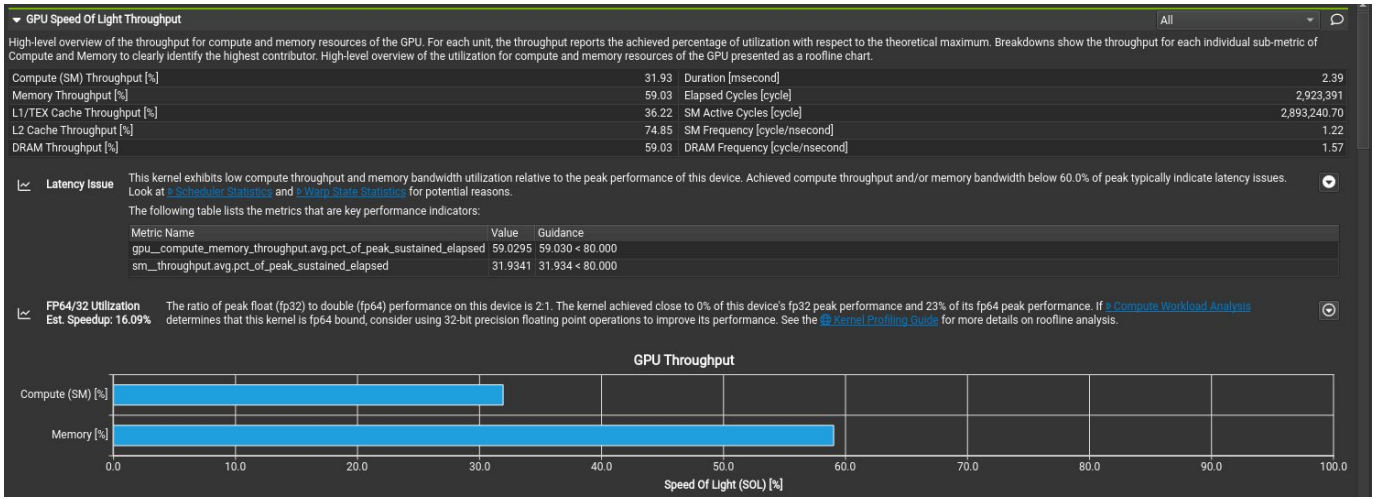
Profiling NCU:

- HistoryBased

srun ncu \
—set full \
—export report2
…

- EventBased





Profiling

- EventBased
  - kernels mostly memory-bound
  - multiple branch divergences in end_condition kernel involving lower Memory SOL due to thread divergence

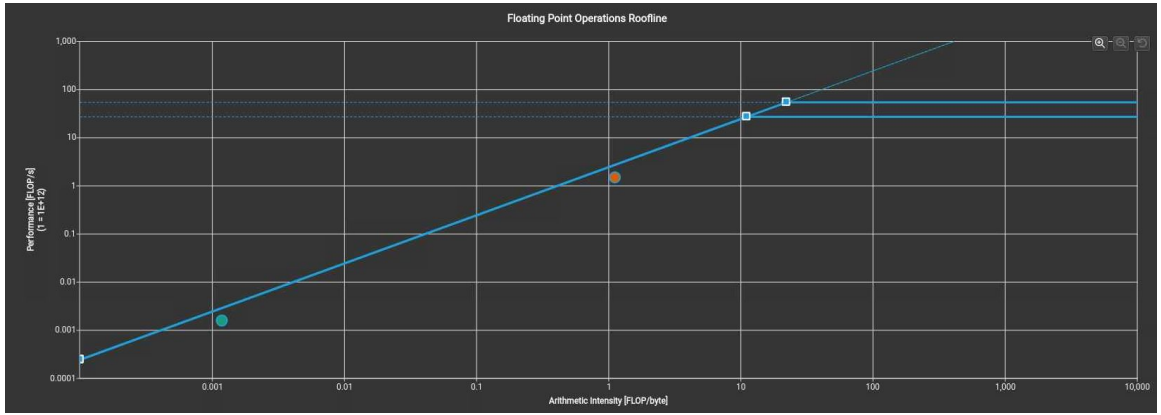
Main kernels	*%*
move_particle	64.8
end_condition	6.2
collisions	4.1
diagnostics	9.6
copy_P_to_P0	7.5
sorting	< 0.1
packing	< 0.1

kernel	*Memory SOL*	*Compute SOL*
move_particle	68	30
end_condition	36	12
collisions	40	56
diagnostics	80	26

EPFL SCITAS
Scientific IT and Application Support

ASCOT5 - GPU profiling

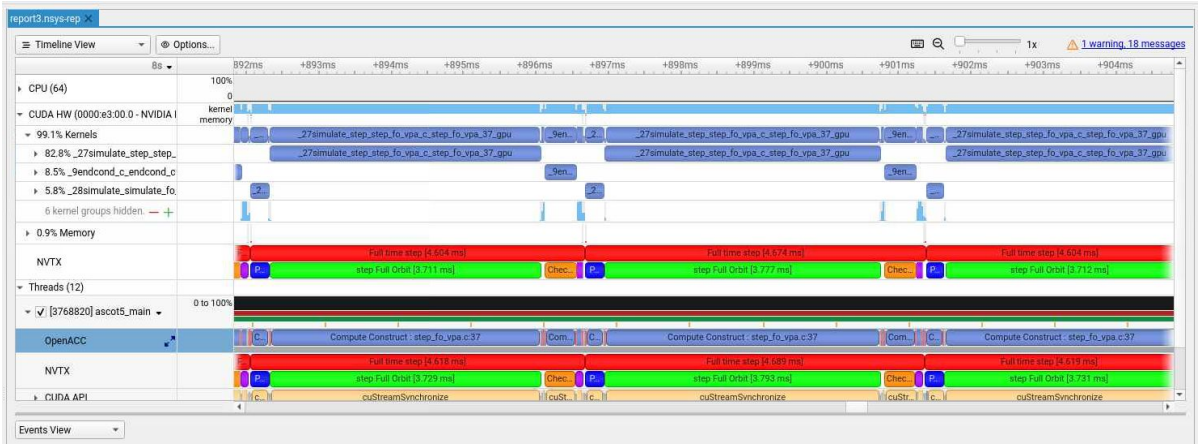
- Nsight Compute
- Kernel analysis
- GPU utilization
- srun ncu -k regex:move_particle --set full



NVTX instrumentation in ASCOT5

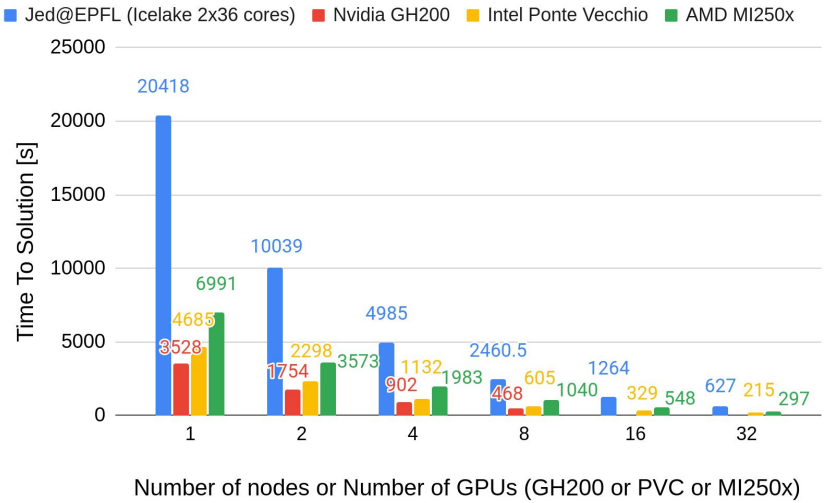
- Code annotations
- Phase identification
- Correlation with profilers

```
nvtxPushColorA("Full time step",NVTX_COLOR_RED);
nvtxPushColorA("Particle copy",NVTX_COLOR_BLUE);
GPU_PARALLEL_LOOP_ALL_LEVELS
for(int i = 0; i < p.n_mrk; i++) {
    particle_copy_fo(&p, i, &p0, i);
}
nvtxRangePop();
```



Benchmarks

- 10M markers Benchmark:
  - Collisional full-orbit simulation of prompt-losses of fusion alpha particles
  - 2D wall; ITER-like but circular equilibrium interpolated with cubic splines
  - 2D wall rectangular, coulomb collisions, gyro orbit, simulation time = 0.0001s, fixed time step
  - **Jed (@EPFL)**: 2x Platinum 8360Y, intel/2021.6.0
  - **NVIDIA Grace Hopper Superchip engineering sample early access courtesy of NVIDIA**
  - **Intel Ponte-Vecchio 600W engineering sample early access courtesy of INTEL**
  - **LUMI-G (GPU partition): AMD MI250x**



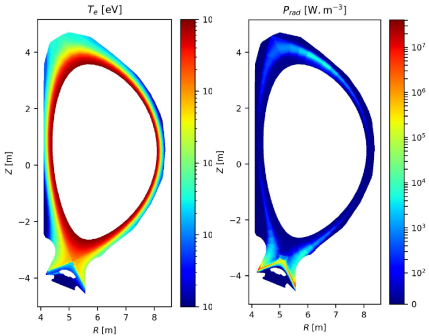
SOLEDGE3X: a versatile fluid code for the edge plasma

- **SOLEDGE3X**: multi-fluid modelling tool for the edge plasma

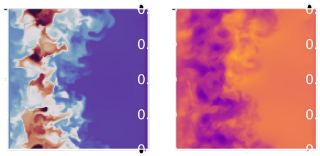
- Key features:
  - **Neutrals** either fluid (embedded) or kinetic (EIRENE)
  - Complete plasma **geometrical flexibility** (arbitrary number of X-points)
  - Usable in **2D or 3D**
  - Usable as **mean-field or self-consistent turbulence** code

- The numerical scheme uses:
  - mix explicit-implicit scheme
  - based on 2D or 3D finite volumes
  - WENO methods for the advection
  - following terms are treated implicitly
    - ➢ Parallel viscosity
    - ➢ Parallel heat conduction
    - ➢ vorticity

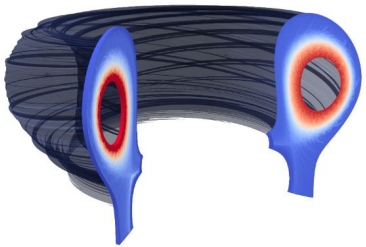
2D mean-field



2D turbulence



3D turbulence



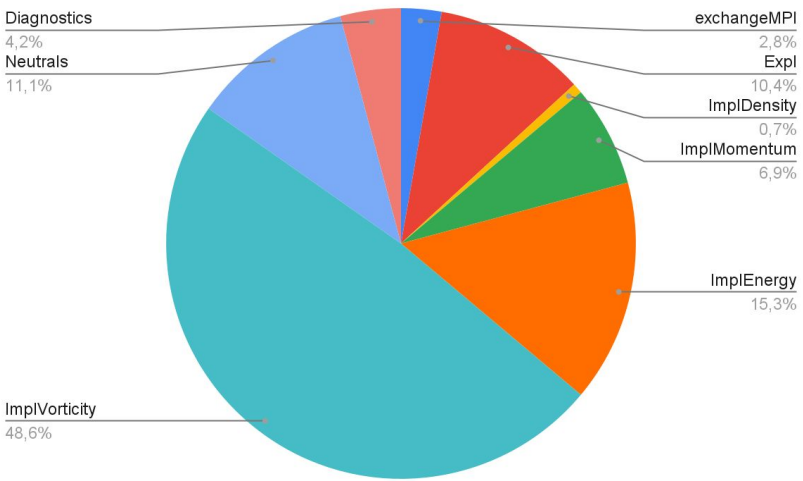
SOLEDGE3X: Domain Decomposition

- 3 level domain decomposition:
  1. Structured zones for magnetic topology
  2. **MPI blocks: prioritized by flux surface** across zones
     - If $N_{MPI} \leq N_{FS}$ each MPI process in charge of a set of FS
     - If $N_{MPI} > N_{FS}$ largest flux surfaces will be shared by a team of MPI processes
  3. **Thread chunks**: no direction priority, aiming at load balance between chunks
     - OpenMP loops are on chunks and species, not on mesh points inside chunks

```
!$OMP DO SCHEDULE(RUNTIME) COLLAPSE(2)
do ichunk = 1, split%Nchunks
   do ispec = 0, Nspecies
      do ipsi = ipsiminWG, ipsimaxWG
         do itheta = ithetaminWG, ithetamaxWG
            do iphi = iphiminWG, iphimaxWG
```

SOLEDGE3X: Profiling of CPU version

- CPU profiling



SOLEDGE3X: GPU Porting Strategy

- GPU porting
  - Maintain a single version of the code
  - Ensure code portability and readability
  - Generic pragma for OpenMP/OpenACC
    - Same approach for other codes such as ASCOT5 and CAS3D
  - Open ACC/MP for advection and matrix construction
  - PETSC (with CUDA/HIP) for linear solvers

```
GPU_LOOP_ALL_LEVELS
do ispec=1,Nspecies
   melt(specElt(ispec))=SpecMass(ispec)
end do
GPU_END_LOOP_ALL_LEVELS


GPU_LOOP_ALL_LEVELS collapse(3)
do ipsi = ipsimin, ipsimax
   do itheta = ithetamin, ithetamax
      do iphi = iphimin, iphimax
         …some work
      end do !iphi
   end do !itheta
end do !ipsi
GPU_END_LOOP_ALL_LEVELS
```

```
#ifndef gpu_commands
#define gpu_commands
#ifdef _OPENMP

#define GPU_MAP_TO_DEVICE !$omp target enter data map(to:
#define GPU_MAP_FROM_DEVICE !$omp target exit data map(from:

#define GPU_ALLOC_ON_DEVICE !$omp target enter data map(alloc:
#define GPU_DELETE_FROM_DEVICE !$omp target exit data map(delete:

#define GPU_LOOP_ALL_LEVELS !$omp target teams distribute parallel do simd
#define GPU_END_LOOP_ALL_LEVELS !$omp end target teams distribute parallel do simd

#define GPU_LOOP_LEVEL_1 !$omp target teams distribute
#define GPU_END_LOOP_LEVEL_1 !$omp end target teams distribute

#define GPU_LOOP_LEVEL_2 !$omp parallel do simd
#define GPU_END_LOOP_LEVEL_2 !$omp end parallel do simd

#elif _OPENACC

#define GPU_MAP_TO_DEVICE !$acc enter data copyin(
#define GPU_MAP_FROM_DEVICE !$acc exit data copyout(

#define GPU_ALLOC_ON_DEVICE !$acc enter data create(
#define GPU_DELETE_FROM_DEVICE !$acc exit data delete(

#define GPU_LOOP_ALL_LEVELS !$acc parallel loop
#define GPU_END_LOOP_ALL_LEVELS !$acc end parallel loop

#define GPU_LOOP_LEVEL_1 !$acc parallel loop gang
#define GPU_END_LOOP_LEVEL_1 !$acc end parallel loop

#define GPU_LOOP_LEVEL_2 !$acc loop worker vector
#define GPU_END_LOOP_LEVEL_2

#endif
#endif
```

SOLEDGE3X: Implicit Solvers

- 5 implicit solvers in SOLEDGE3X :
  - **Parallel viscosity** terms
  - **Parallel heat conduction** terms

  } 2D

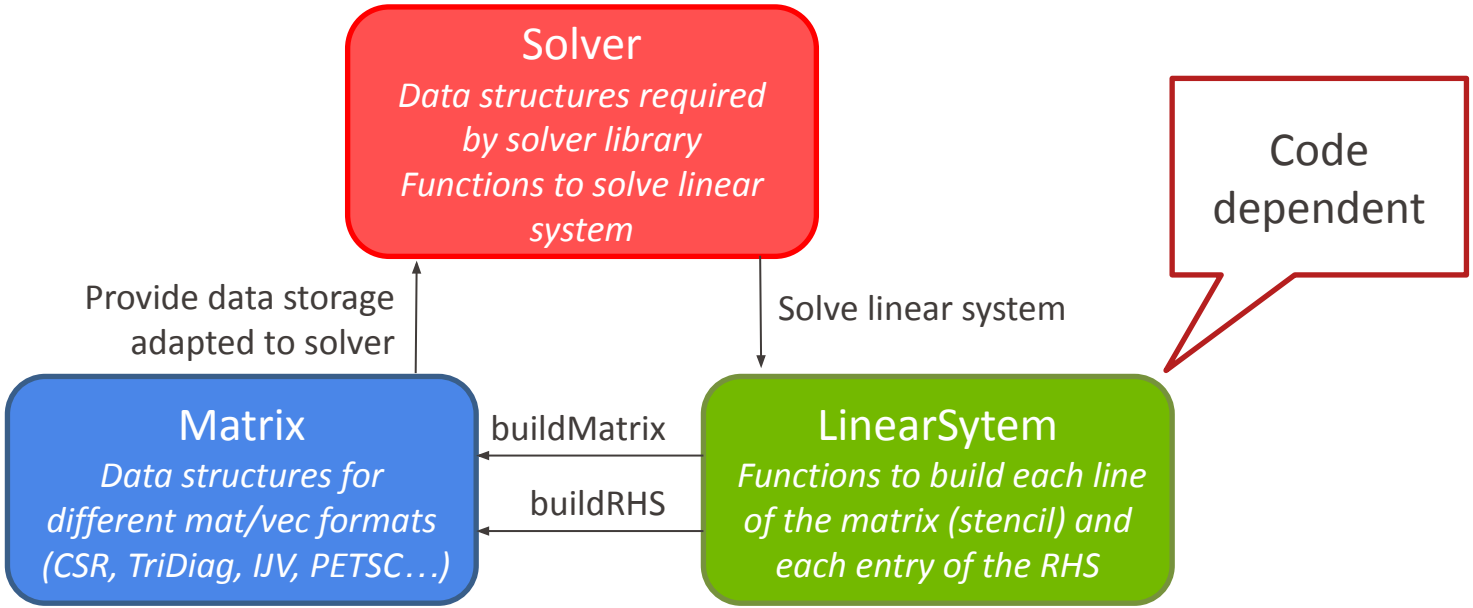
  - **Vorticity** equation
  - (optional) **fluid neutrals**
  - (optional) **potential filter**

  } 3D

  - **and more for EM …**

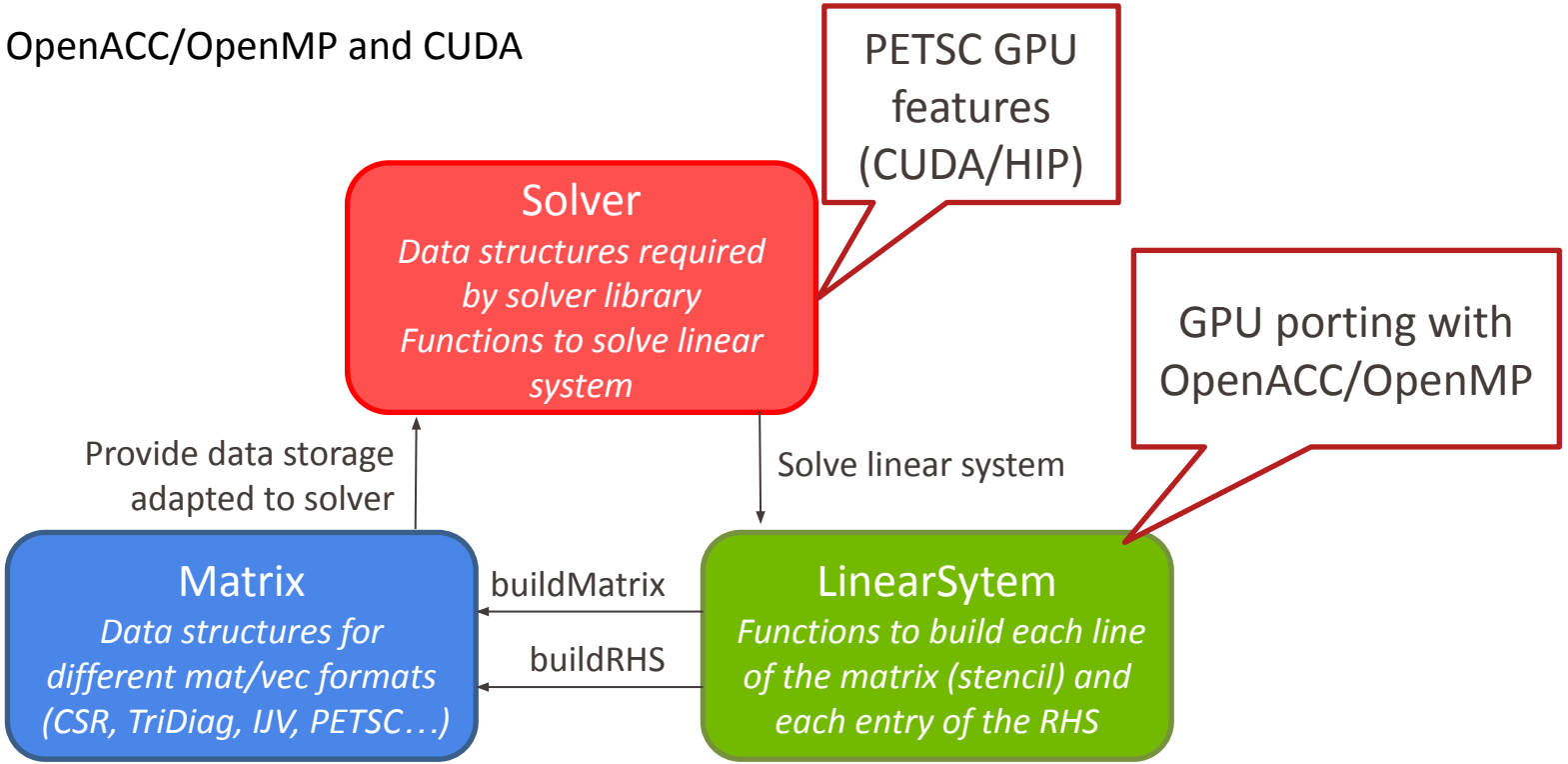
SOLEDGE3X: Implicit Solvers

- Implicit solvers
  - solvers management based on 3 Fortran classes



SOLEDGE3X: Implicit Solvers

- Mix OpenACC/OpenMP and CUDA

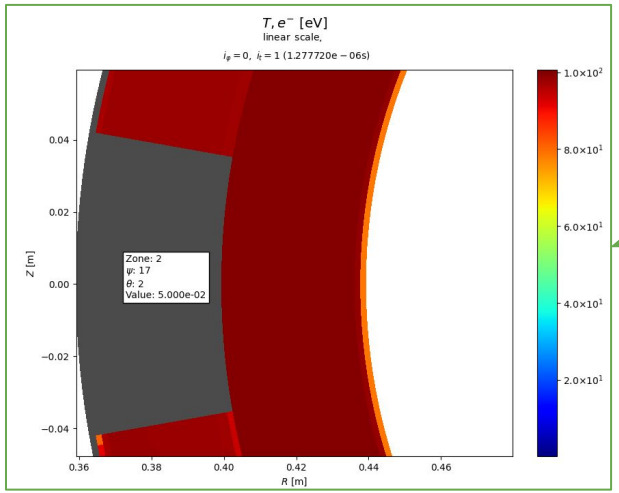


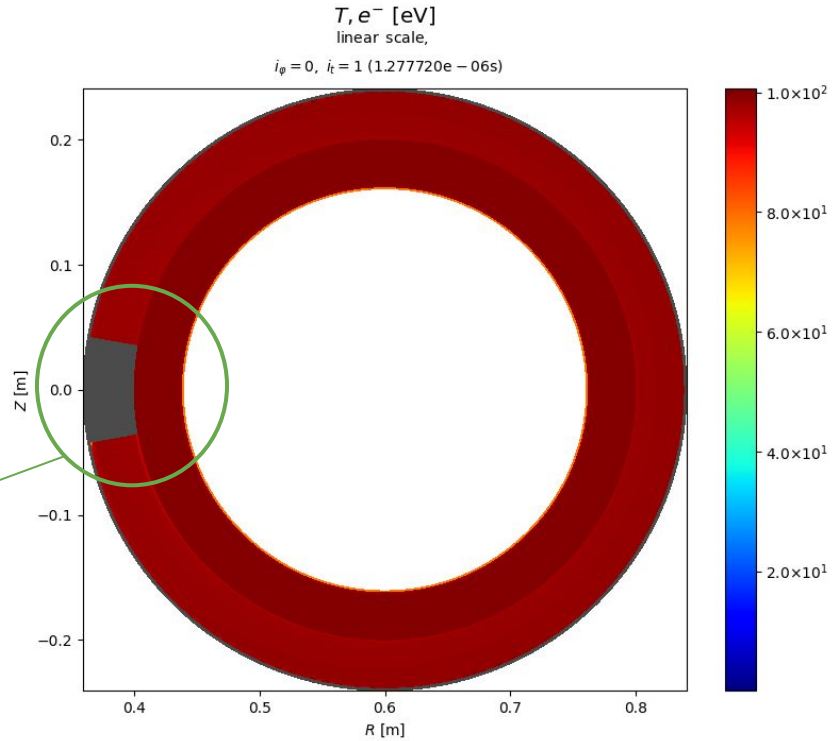
Profiling Setup

Test case:

Npsi = 64, Ntheta = 512, Nphi = 64

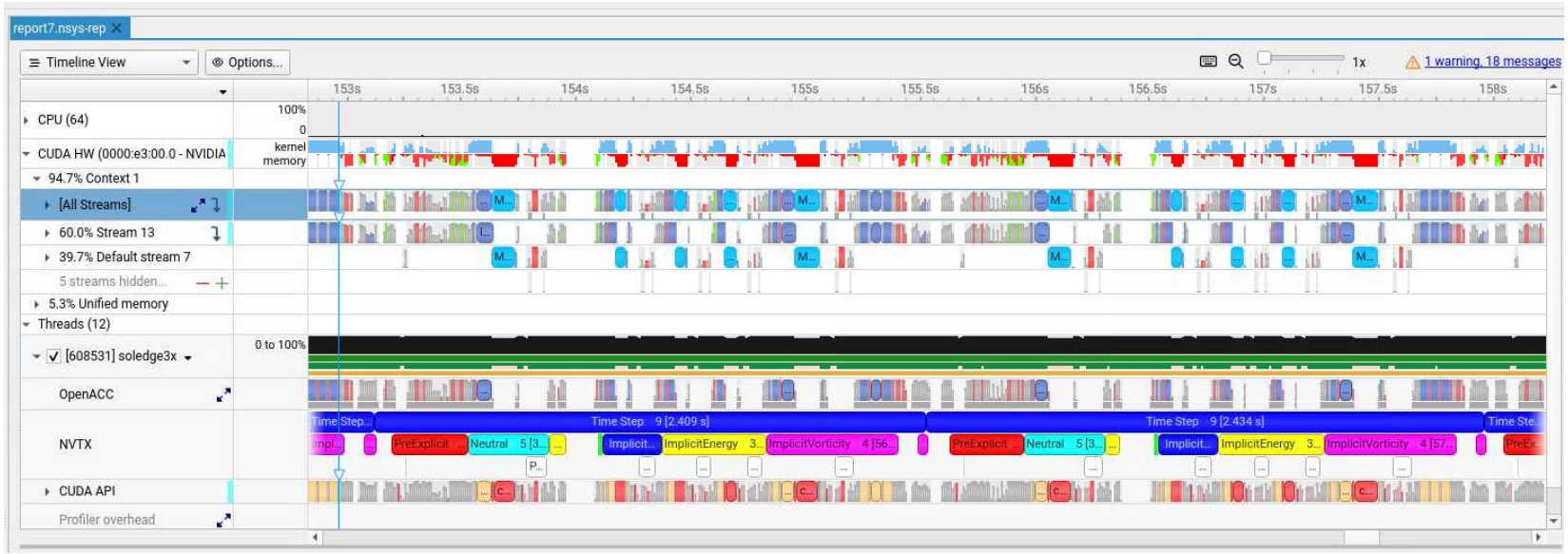
- presence of a wall
- Petsc for all implicit solvers
- Neutrals
- Vorticity filtering





SOLEDGE3X - GPU profiling

- - Nsight Systems
- - GPU activity
- - MPI + GPU interaction



NVTX Instrumentation in Fortran

- Fortran bindings
- Integration strategy
- Benefits for analysis

```
module nvtx_mod

  use iso_c_binding
  implicit none

  ! NVTX color is ARGB: 0xAARRGGBB (alpha must be FF for opaque colors)
  integer, private :: col(7) = [ int(Z'FF00FF00',8), int(Z'FF0000FF',8), int(Z'FFFFFF00',8), &
                                 int(Z'FFFF00FF',8), int(Z'FF00FFFF',8), int(Z'FFFF0000',8), int(Z'FFFFFFFF',8) ]

  ! C char buffer for the message (null-terminated). Must have TARGET for c_loc().
  character(kind=C_CHAR), private, target :: tempName(256)

  integer, parameter :: invtx_ImplicitDensities             = 1
  integer, parameter :: invtx_ImplicitMomentum              = 2
….
  integer, parameter :: invtx_weno                          = 13

  ! NVTX event attributes struct (NVTX v2-ish layout). Size must match the C struct.
  type, bind(C) :: nvtxEventAttributes
               integer(C_INT16_T) :: version = 1
               integer(C_INT16_T) :: size    = 48
….
               = C_NULL_PTR
  end type nvtxEventAttributes

  interface
               integer(C_INT) function nvtxRangePushA(name) bind(C, name='nvtxRangePushA')
               use iso_c_binding
               character(kind=C_CHAR), dimension(*) :: name
               end function nvtxRangePushA

               integer(C_INT) function nvtxRangePushEx(event) bind(C, name='nvtxRangePushEx')
               use iso_c_binding
               import :: nvtxEventAttributes
               type(nvtxEventAttributes), intent(in) :: event
               end function nvtxRangePushEx

               integer(C_INT) function nvtxRangePop() bind(C, name='nvtxRangePop')
               use iso_c_binding
               end function nvtxRangePop
  end interface

contains

  subroutine nvtxStartRange(name, id)
               use iso_c_binding
               character(kind=C_CHAR, len=*), intent(in) :: name
               integer, optional, intent(in) :: id

               type(nvtxEventAttributes) :: event
…..
  end subroutine nvtxStartRange


  subroutine nvtxEndRange()
               use iso_c_binding
               integer(C_INT) :: rid
               rid = nvtxRangePop()
  end subroutine nvtxEndRange

end module nvtx_mod
```

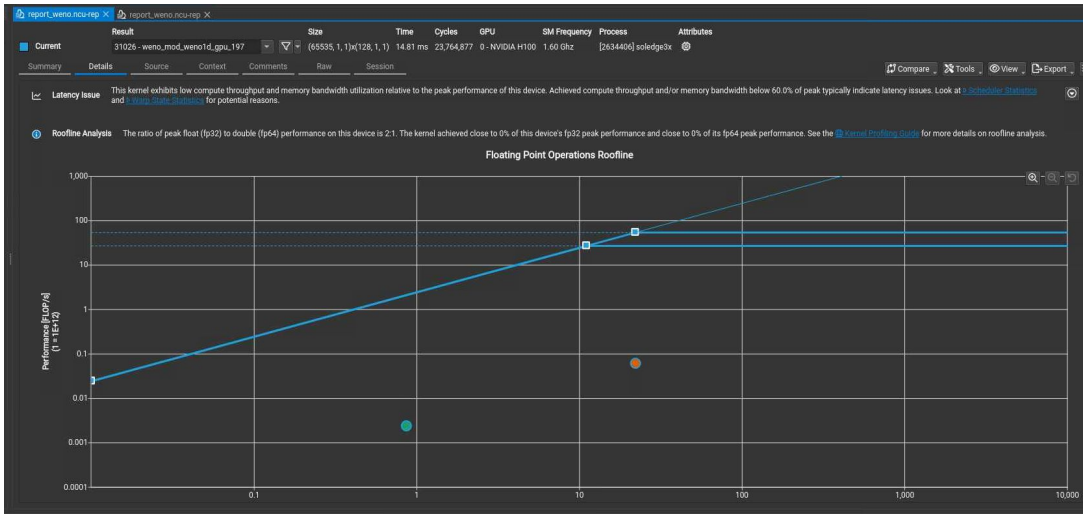
Nsys stats

- srun nsys profile --stats=true -t cuda,openacc
- *build_mat_loop* called by every implicit solver
- *weno* weirdly time consuming

Time (%)	Total Time (ns)	Instances	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
27.9	1,165,205,839	23	50,661,123.4	35,985,738.0	34,912,745	69,515,157	16,309,527.4	linsys_buildmat_loop_petsc_coo_1753_gpu
16.4	684,103,244	57	12,001,811.3	12,101,649.0	11,784,336	12,118,192	148,570.5	weno_mod_weno1d_gpu_197_gpu
12.0	501,134,901	648	773,356.3	100,368.0	5,984	4,350,149	1,277,420.6	void cub::CUB_200101_500_520_600_610_700_750_800_860_890_900_NS::DeviceRadixSortOnesweepKernel<cub:…
3.8	159,825,657	3,973	40,228.0	5,024.0	2,464	257,344	73,904.1	void cusparse::csrmv_v3_kernel<std::integral_constant<bool, (bool)0>, int, int, double, double, dou…
3.6	152,485,705	4	38,121,426.3	38,198,739.0	37,712,208	38,376,019	293,380.0	vorticity_getglobalchunkstencil_vorticity_getglobalchunkstencil_petsc_1782_gpu
3.4	140,923,351	8	17,615,418.9	17,628,310.5	17,358,549	17,923,190	156,265.6	diffparalt_getglobalchunkstencil_diffparalt_getglobalchunkstencil_petsc_446_gpu
2.3	97,915,691	2	48,957,845.5	48,957,845.5	48,794,885	49,120,806	230,460.9	vorticity_mod_computejpoladt_138_gpu
2.3	97,314,920	2	48,657,460.0	48,657,460.0	48,631,940	48,682,980	36,090.7	vorticity_mod_computejpoladt_90_gpu
2.3	94,718,885	2	47,359,442.5	47,359,442.5	47,084,771	47,634,114	388,444.2	vorticity_mod_computejpoladt_187_gpu
1.8	74,201,926	4	18,550,481.5	18,535,817.5	18,453,113	18,677,178	99,756.9	fn_getglobalchunkstencil_fn_getglobalchunkstencil_petsc_379_gpu
1.5	61,343,598	4	15,335,899.5	15,358,994.0	15,173,685	15,451,925	132,822.5	diffparalv_getglobalchunkstencil_diffparalv_getglobalchunkstencil_petsc_418_gpu
1.4	57,717,767	3	19,239,255.7	19,249,209.0	19,132,279	19,336,279	102,363.6	filteringphi_getglobalchunkstencil_filteringphi_getglobalchunkstencil_petsc_343_gpu
1.2	49,428,995	12	4,119,082.9	4,141,398.0	3,811,877	4,320,485	174,007.4	linsys_buildrhs_loop_1_gpu_1963_gpu
1.2	49,205,093	12	4,100,424.4	3,999,029.5	3,746,885	5,565,032	480,344.1	solver_class_mod_solveriterative_storesol_701_gpu
1.0	42,469,947	2	21,234,973.5	21,234,973.5	21,007,070	21,462,877	322,304.2	vorticity_getglobalchunkrhs_vorticity_getglobalchunkrhs_petsc_1402_gpu
0.9	37,420,666	178	210,228.5	9,136.0	2,720	4,804,007	658,464.0	void cusparse::load_balancing_kernel<(unsigned int)128, (unsigned int)8, (unsigned long)8192, int, …
0.9	36,980,339	96	385,211.9	15,872.0	2,463	6,260,552	1,127,467.0	void cusparse::load_balancing_kernel<(unsigned int)128, (unsigned int)8, (unsigned long)0, int, int…
0.7	30,395,497	2	15,197,748.5	15,197,748.5	15,193,941	15,201,556	5,384.6	plasmaevolve_mod_evolveimplicitp_72_gpu

NCU-Roofline

- *weno* weirdly time consuming




```
! Loop on the cells and performs interpolation
GPU_PARALLEL_LOOP_ALL_LEVELS collapse(3) private(reconstm, reconstp, beta, alpham, alphap)
do ix = 1,Nx
do iy = 1,Ny
do iz=1+order-1,Nz-order+1
! Polynomial reconstructions at the faces of the cell
do r = 0, order-1
reconstm(r) = 0._dp
do j = 0, order-1
                reconstm(r) = reconstm(r) + crj(r,j)*var(iz-r+j, iy, ix)
enddo
enddo
do r = 0, order-1
reconstp(r) = 0._dp
do j = 0, order-1
                reconstp(r) = reconstp(r) + crj(r-1,j)*var(iz-r+j, iy, ix)
enddo
enddo
! Calculation of beta (smoothness) factors
SELECT CASE (order)
CASE (2)
beta(0) = (var(iz+1, iy, ix)-var(iz, iy, ix))**2
beta(1) = (var(iz, iy, ix)-var(iz-1, iy, ix))**2
CASE (3)
beta(0) = 13./12.*(var(iz, iy, ix)-2.*var(iz+1, iy, ix)+var(iz+2, iy, ix))**2 &
                +1./4.*(3.*var(iz, iy, ix)-4.*var(iz+1, iy, ix)+var(iz+2, iy, ix))**2
beta(1) = 13./12.*(var(iz-1, iy, ix)-2.*var(iz, iy, ix)+var(iz+1, iy, ix))**2 &
                +1./4.*(var(iz-1, iy, ix)-var(iz+1, iy, ix))**2
beta(2) = 13./12.*(var(iz-2, iy, ix)-2.*var(iz-1, iy, ix)+var(iz, iy, ix))**2 &
                +1./4.*(var(iz-2, iy, ix)-4.*var(iz-1, iy, ix)+3.*var(iz, iy, ix))**2
END SELECT
! Calculation of WENO coefficients
alphapsum = 0.0
alphamsum = 0.0
do r = 0, order-1
alphap(r) = dplus(r)/(epsilon+beta(r))**2
alpham(r) = dminus(r)/(epsilon+beta(r))**2
alphapsum = alphapsum + alphap(r)
alphamsum = alphamsum + alpham(r)
end do
ur(iz-1, iy, ix) = 0.0
ul(iz, iy, ix) = 0.0
do r = 0, order-1
ur(iz-1, iy, ix) = ur(iz-1, iy, ix) + (alphap(r)/alphapsum)*reconstp(r)
ul(iz, iy, ix) = ul(iz, iy, ix) + (alpham(r)/alphamsum)*reconstm(r)
end do
! WENO reconstruction
enddo
end do
end do
GPU_END_PARALLEL_LOOP_ALL_LEVELS
```

GPU Compiler Optimization Reporting

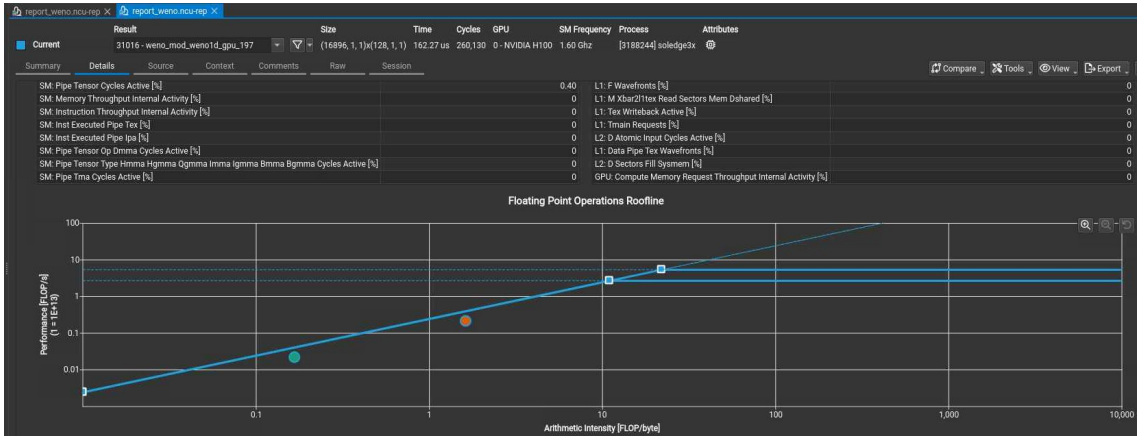
- *-Minfo=all*
- inner loops are parallelized by the compiler

```
197, Generating implicit firstprivate(order,nz,ny,nx)
     Generating NVIDIA GPU code
    198, !$acc loop gang collapse(3) ! blockidx%x
    199,   ! blockidx%x collapsed
    200,   ! blockidx%x collapsed
    202, !$acc loop vector(128) ! threadidx%x
    204, !$acc loop seq
    208, !$acc loop vector(128) ! threadidx%x
    210, !$acc loop seq
    230, !$acc loop vector(128) ! threadidx%x
         Generating implicit reduction(+:alphapsum,alphamsum)
    238, !$acc loop seq
197, Generating implicit copyout(ur(order-1:nz-order,:ny,:nx)) [if not already present]
     Generating implicit copyin(var(:,:ny,:nx),dminus(:order-1),crj(:order-1,:order-1))
     Generating implicit copyout(ul(order:nz-order+1,:ny,:nx)) [if not already present]
     Generating implicit copyin(dplus(:order-1)) [if not already present]
198, Loop not fused: no successor loop
     Loop not vectorized/parallelized: too deeply nested
199, Loop not vectorized/parallelized: too deeply nested
200, Generating implicit firstprivate(alphapsum,alphamsum)
     Invariant if transformation
     Loop not fused: no successor loop
     FMA (fused multiply-add) instruction(s) generated
202, Loop is parallelizable
     Generating implicit firstprivate(j)
     Loop not fused: enclosed in different number of nests
     2 loops fused
     Loop not fused: enclosed in different number of nests
     2 loops fused
     Loop not fused: enclosed in different number of nests
     2 loops fused
204, Complex loop carried dependence of reconstm prevents parallelization
     Loop carried reuse of reconstm prevents parallelization
     Loop not fused: possible intervening definition
     Generated vector simd code for the loop containing reductions
208, Loop is parallelizable
210, Complex loop carried dependence of reconstp prevents parallelization
     Loop carried reuse of reconstp prevents parallelization
     Generated vector simd code for the loop containing reductions
230, Loop is parallelizable
     Loop not fused: possible intervening definition
     Generated vector simd code for the loop containing reductions
238, Complex loop carried dependence of ur prevents parallelization
     Loop carried reuse of ur prevents parallelization
     Complex loop carried dependence of ul prevents parallelization
     Loop carried reuse of ul prevents parallelization
     Generated vector simd code for the loop containing reductions
```

```
196      ! Loop on the cells and performs interpolation
197      GPU_PARALLEL_LOOP_ALL_LEVELS collapse(3) private(reconstm, reconstp, beta, alpham, alphap)
198      do ix = 1,Nx
199         do iy = 1,Ny
200            do iz=1+order-1,Nz-order+1
201               ! Polynomial reconstructions at the faces of the cell
202               do r = 0, order-1
203                  reconstm(r) = 0._dp
204                  do j = 0, order-1
205                     reconstm(r) = reconstm(r) + crj(r,j)*var(iz-r+j, iy, ix)
206                  enddo
207               enddo
208               do r = 0, order-1
209                  reconstp(r) = 0._dp
210                  do j = 0, order-1
211                     reconstp(r) = reconstp(r) + crj(r-1,j)*var(iz-r+j, iy, ix)
212                  enddo
213               enddo
214               ! Calculation of beta (smoothness) factors
215               SELECT CASE (order)
216               CASE (2)
217                  beta(0) = (var(iz+1, iy, ix)-var(iz, iy, ix))**2
218                  beta(1) = (var(iz, iy, ix)-var(iz-1, iy, ix))**2
219               CASE (3)
220                  beta(0) = 13./12.*(var(iz, iy, ix)-2.*var(iz+1, iy, ix)+var(iz+2, iy, ix))**2 &
221                       +1./4.*(3.*var(iz, iy, ix)-4.*var(iz+1, iy, ix)+var(iz+2, iy, ix))**2
222                  beta(1) = 13./12.*(var(iz-1, iy, ix)-2.*var(iz, iy, ix)+var(iz+1, iy, ix))**2 &
223                       +1./4.*(var(iz-1, iy, ix)-var(iz+1, iy, ix))**2
224                  beta(2) = 13./12.*(var(iz-2, iy, ix)-2.*var(iz-1, iy, ix)+var(iz, iy, ix))**2 &
225                       +1./4.*(var(iz-2, iy, ix)-4.*var(iz-1, iy, ix)+3.*var(iz, iy, ix))**2
226               END SELECT
227               ! Calculation of WENO coefficients
228               alphapsum = 0.0
229               alphamsum = 0.0
230               do r = 0, order-1
231                  alphap(r) = dplus(r)/(epsilon+beta(r))**2
232                  alpham(r) = dminus(r)/(epsilon+beta(r))**2
233                  alphapsum = alphapsum + alphap(r)
234                  alphamsum = alphamsum + alpham(r)
235               end do
236               ur(iz-1, iy, ix) = 0.0
237               ul(iz, iy, ix) = 0.0
238               do r = 0, order-1
239                  ur(iz-1, iy, ix) = ur(iz-1, iy, ix) + (alphap(r)/alphapsum)*reconstp(r)
240                  ul(iz, iy, ix) = ul(iz, iy, ix) + (alpham(r)/alphamsum)*reconstm(r)
241               end do
242            enddo
243         end do
244      end do
245      GPU_END_PARALLEL_LOOP_ALL_LEVELS
```

NCU-Roofline

- *weno* weirdly time consuming:
  - inner loops are parallelized by the compiler
  - use of:
    #define GPU_LOOP_SEQ !$acc loop seq




```
! Loop on the cells and performs interpolation
GPU_PARALLEL_LOOP_ALL_LEVELS collapse(3) private(reconstm, reconstp, beta, alpham, alphap)
do ix = 1,Nx
do iy = 1,Ny
do iz=1+order-1,Nz-order+1
! Polynomial reconstructions at the faces of the cell
GPU_LOOP_SEQ
do r = 0, order-1
reconstm(r) = 0._dp
GPU_LOOP_SEQ
do j = 0, order-1
            reconstm(r) = reconstm(r) + crj(r,j)*var(iz-r+j, iy, ix)
enddo
enddo
GPU_LOOP_SEQ
do r = 0, order-1
reconstp(r) = 0._dp
GPU_LOOP_SEQ
do j = 0, order-1
            reconstp(r) = reconstp(r) + crj(r-1,j)*var(iz-r+j, iy, ix)
enddo
enddo
! Calculation of beta (smoothness) factors
SELECT CASE (order)
CASE (2)
beta(0) = (var(iz+1, iy, ix)-var(iz, iy, ix))**2
beta(1) = (var(iz, iy, ix)-var(iz-1, iy, ix))**2
CASE (3)
beta(0) = 13./12.*(var(iz, iy, ix)-2.*var(iz+1, iy, ix)+var(iz+2, iy, ix))**2 &
          +1./4.*(3.*var(iz, iy, ix)-4.*var(iz+1, iy, ix)+var(iz+2, iy, ix))**2
beta(1) = 13./12.*(var(iz-1, iy, ix)-2.*var(iz, iy, ix)+var(iz+1, iy, ix))**2 &
          +1./4.*(var(iz-1, iy, ix)-var(iz+1, iy, ix))**2
beta(2) = 13./12.*(var(iz-2, iy, ix)-2.*var(iz-1, iy, ix)+var(iz, iy, ix))**2 &
          +1./4.*(var(iz-2, iy, ix)-4.*var(iz-1, iy, ix)+3.*var(iz, iy, ix))**2
END SELECT
! Calculation of WENO coefficients
alphapsum = 0.0
alphamsum = 0.0
GPU_LOOP_SEQ
do r = 0, order-1
alphap(r) = dplus(r)/(epsilon+beta(r))**2
alpham(r) = dminus(r)/(epsilon+beta(r))**2
alphapsum = alphapsum + alphap(r)
alphamsum = alphamsum + alpham(r)
end do
ur(iz-1, iy, ix) = 0.0
ul(iz, iy, ix) = 0.0
GPU_LOOP_SEQ
do r = 0, order-1
ur(iz-1, iy, ix) = ur(iz-1, iy, ix) + (alphap(r)/alphapsum)*reconstp(r)
ul(iz, iy, ix) = ul(iz, iy, ix) + (alpham(r)/alphamsum)*reconstm(r)
end do
! WENO reconstruction
enddo
end do
end do
GPU_END_PARALLEL_LOOP_ALL_LEVELS
```

Nsys stats

- srun nsys profile --stats=true -t cuda,openacc

Time (%)	Total Time (ns)	Instances	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Name
33.2	1,162,820,005	23	50,557,391.5	35,931,171.0	34,766,213	69,225,484	16,211,812.5	linsys_buildmat_loop_petsc_coo_1753_gpu
14.3	501,125,429	648	773,341.7	100,704.0	5,951	4,350,913	1,277,685.8	void cub::CUB_200101_500_520_600_610_700_750_800_860_890_900_NS::DeviceRadixSortOnesweepKernel<cub:…
4.6	159,909,078	3,973	40,248.9	5,024.0	2,464	257,024	73,944.0	void cusparse::csrmv_v3_kernel<std::integral_constant<bool, (bool)0>, int, int, double, double, dou…
4.4	153,640,586	4	38,410,146.5	38,366,970.5	38,241,607	38,665,038	180,797.6	vorticity_getglobalchunkstencil_vorticity_getglobalchunkstencil_petsc_1782_gpu
4.0	141,136,354	8	17,642,044.3	17,574,292.5	17,438,150	17,950,946	181,514.3	diffparalt_getglobalchunkstencil_diffparalt_getglobalchunkstencil_petsc_446_gpu
2.8	97,649,667	2	48,824,833.5	48,824,833.5	48,742,577	48,907,090	116,328.3	vorticity_mod_computejpoladt_138_gpu
2.7	95,569,954	2	47,784,977.0	47,784,977.0	46,614,129	48,955,825	1,655,829.1	vorticity_mod_computejpoladt_90_gpu
2.7	94,845,251	2	47,422,625.5	47,422,625.5	47,372,593	47,472,658	70,756.6	vorticity_mod_computejpoladt_187_gpu
2.1	75,004,630	4	18,751,157.5	18,779,172.5	18,412,551	19,033,734	258,109.4	fn_getglobalchunkstencil_fn_getglobalchunkstencil_petsc_379_gpu
1.8	61,759,660	4	15,439,915.0	15,425,715.0	15,324,449	15,583,781	124,036.8	diffparalv_getglobalchunkstencil_diffparalv_getglobalchunkstencil_petsc_418_gpu
1.6	56,761,195	3	18,920,398.3	19,149,507.0	18,391,683	19,220,005	459,235.7	filteringphi_getglobalchunkstencil_filteringphi_getglobalchunkstencil_petsc_343_gpu
1.4	49,188,431	12	4,099,035.9	4,117,217.0	3,738,913	4,261,026	138,815.9	solver_class_mod_solveriterative_storesol_701_gpu
1.4	49,051,475	12	4,087,622.9	4,103,377.0	3,759,618	4,339,874	160,733.4	linsys_buildrhs_loop_1_gpu_1963_gpu
1.2	42,034,190	2	21,017,095.0	21,017,095.0	20,932,264	21,101,926	119,969.2	vorticity_getglobalchunkrhs_vorticity_getglobalchunkrhs_petsc_1402_gpu
1.1	37,426,758	178	210,262.7	9,344.0	2,751	4,805,282	658,524.7	void cusparse::load_balancing_kernel<(unsigned int)128, (unsigned int)8, (unsigned long)8192, int, …
1.1	36,979,024	96	385,198.2	15,760.0	2,432	6,260,642	1,127,506.7	void cusparse::load_balancing_kernel<(unsigned int)128, (unsigned int)8, (unsigned long)0, int, int…

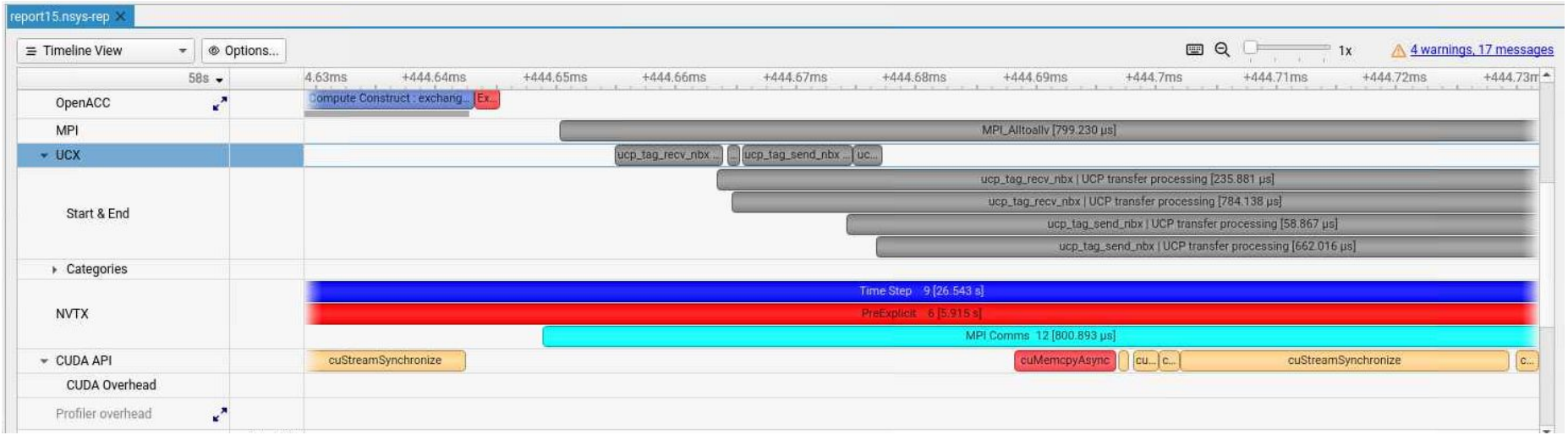
NVTX Instrumentation in Fortran

- Profiling specific regions

```
srun  ncu --nvtx --nvtx-include "WENO" --set full -f  -o report3
../../../build_gpu/soledge3x -ksp_type gmres -pc_type gamg -mat_type
mpiaijcusparse -vec_type mpicuda -log_nvtx 0
```

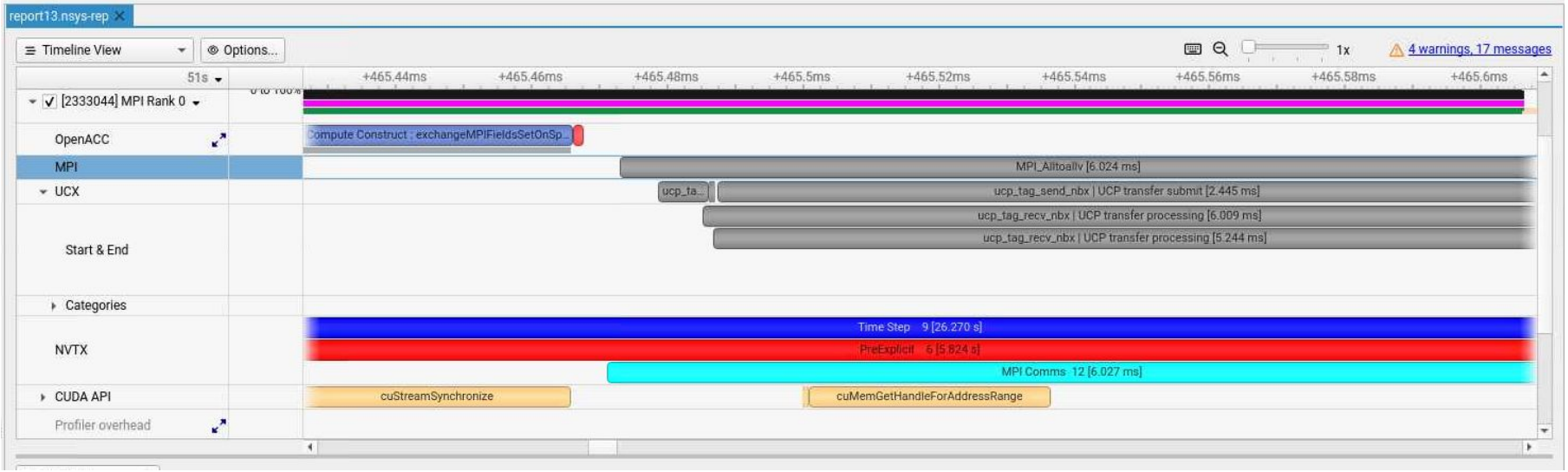
SOLEDGE3X - Multi-GPU profiling

- If RDMA is not used, cuda Mem copy appears



SOLEDGE3X - Multi-GPU profiling

- Nsight Systems
- If RDMA on, No cuda Mem copy



SOLEDGE3X - Multi-GPU profiling

- MPI + GPU interaction (GPU Direct)
  - *export UCX_LOG_LEVEL=debug*
  - **Enable UCX debug logs** (e.g. UCX_LOG_LEVEL=debug) and check that UCX selects an InfiniBand transport such as **rc_mlx5** (e.g. “created interface using rc_mlx5/mlx5_*”), which indicates an RDMA-capable network path.
  - Look for GPU-related capabilities on the selected IB device, e.g. **“cuda GPUDirect RDMA is enabled”** and **“dmabuf is supported”**, which show that UCX/IB can handle CUDA device memory for RDMA.
  - For additional confidence, check for **memory-registration messages** on mlx5/DevX (e.g. “memory registration status *Success*”), which indicates that buffers are being registered for high-performance RDMA transfers.

Mini-app: PETSc Linear Solver

- Solver hotspot
- KSP and preconditioners
- Parameter exploration
- https://github.com/peyberne/petsc-python-miniapp/tree/main

PETSc Python Mini-app: Overview

- Standalone benchmarking tool based on petsc4py
- Designed to evaluate PETSc KSP solvers and preconditioners
- Supports CPU and GPU executions
- Enables systematic exploration of solver parameters
- Provides reproducible performance measurements

Mimicking Soledge3x Linear Solvers

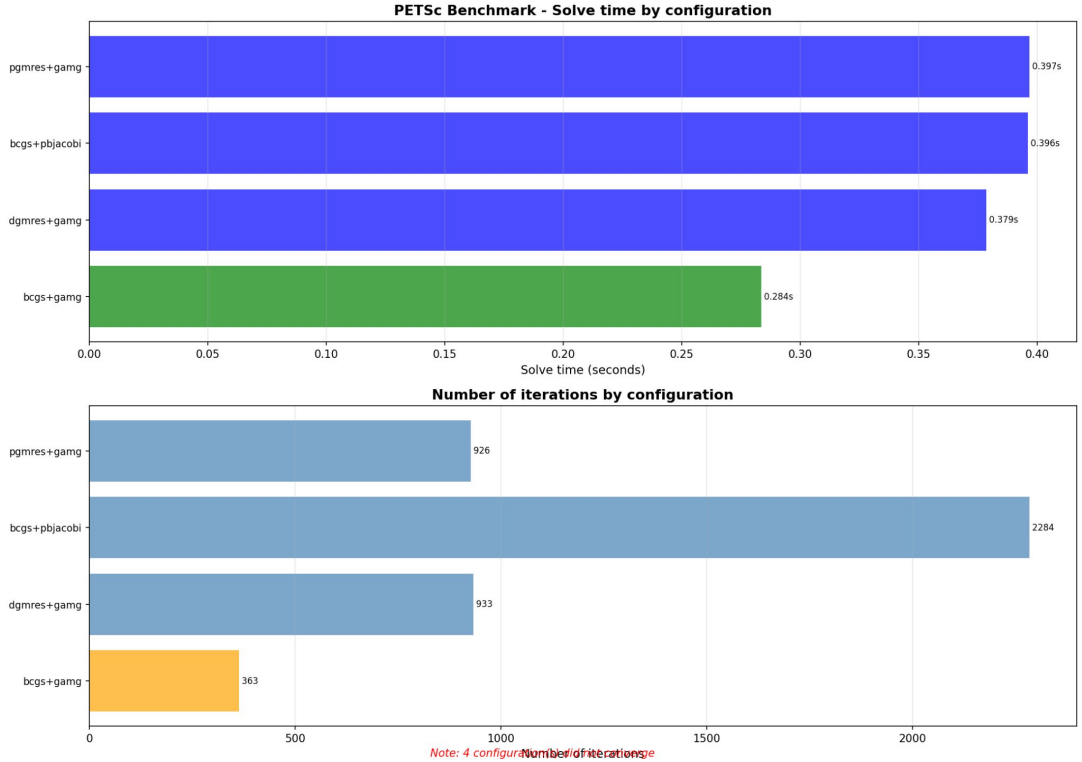
- Matrices and RHS vectors are dumped from SOLEDGE3X
- Realistic reproduction of production workloads
- Uses original problem structure and conditioning
- Allows offline tuning of solver and preconditioner options
- Optimal configurations can be reintegrated into SOLEDGE3X

Running Benchmark with Miniapp

```
$ cat data/options.json
{
  "ksp_rtol": [1e-13],
  "pc_type": ["gamg", "pbjacobi"],
  "ksp_type": ["gmres", "bcgs", "dgmres",
"pgmres"],
  "use_initial_guess": [true]
}
```

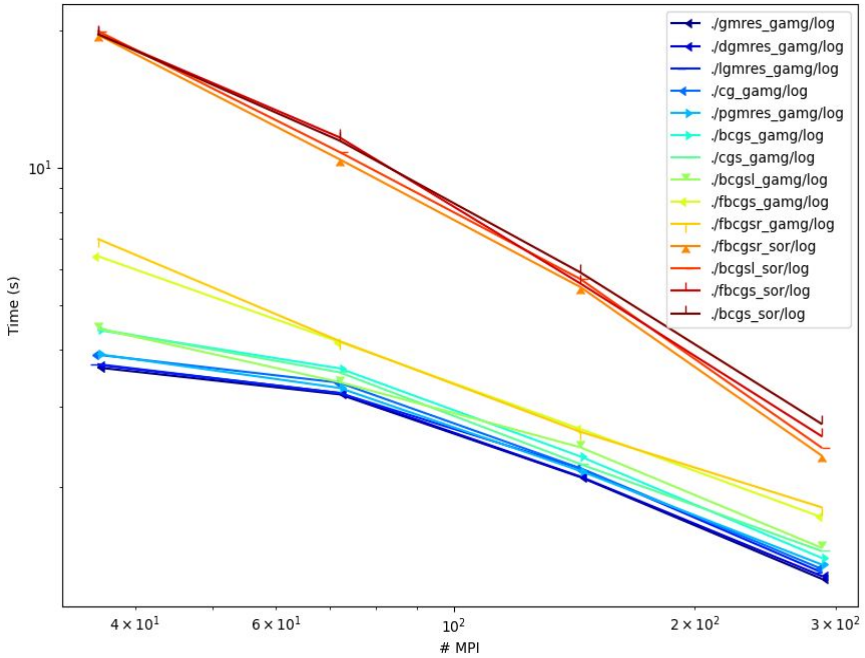
```
# Run benchmark
srun -n $SLURM_NTASKS python3
benchmark_petsc.py \
       --mat $MATRIX_FILE \
       --rhs $RHS_FILE \
       --guess $GUESS_FILE \
       --ref $REF_FILE \
       --config $CONFIG_FILE \
       --gpu
```

```
Benchmark completed!

Plot saved: results/benchmark_results.png

=== TOP 3 fastest configurations ===
1. bcgs+gamg: 0.2836s (363 iterations)
2. dgmres+gamg: 0.3787s (933 iterations)
3. bcgs+pbjacobi: 0.3962s (2284 iterations)
========================================
Benchmark completed
Date: mer 04 fév 2026 22:35:27 CET
========================================
```



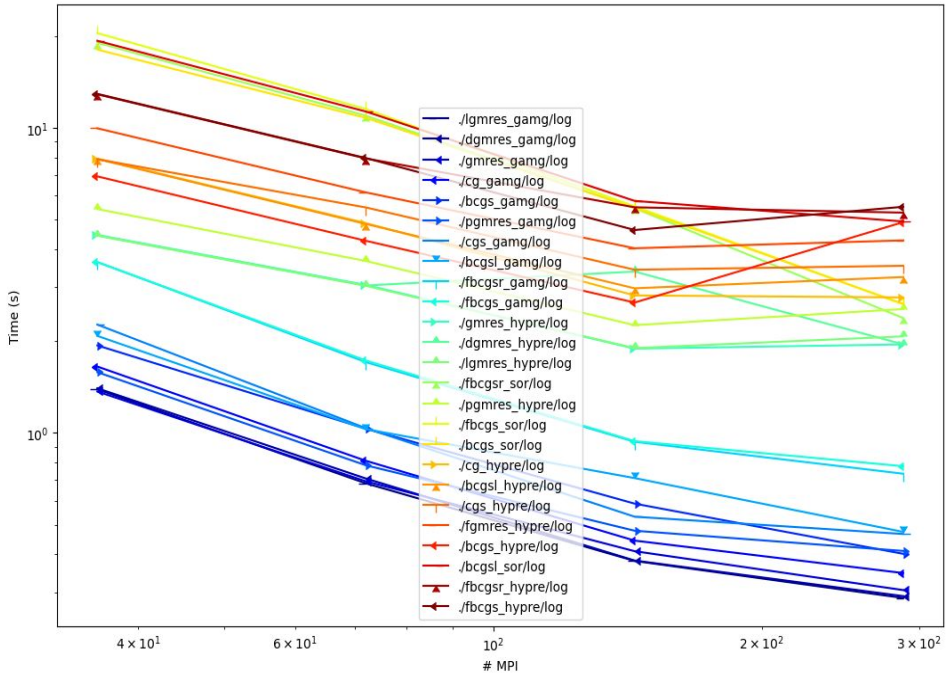
TCV- Timing solver 3D

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



TCV-Timing solver 3D

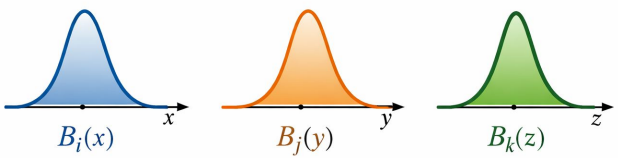
- TCV test-case: timing for the 3d linear solver (reusing preconditioning) using the matrix dumped from the TCV case (use of the miniapp)



Tricubic Splines in Ascot5

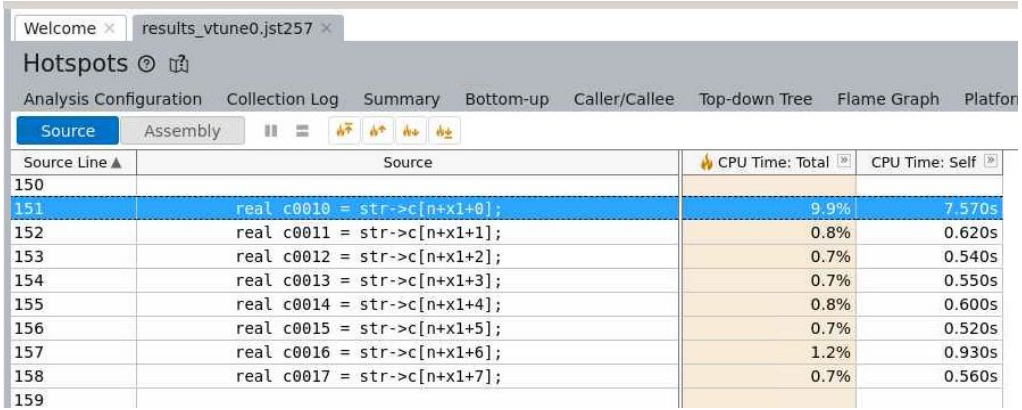
- Fields are discretized on a 3D grid
- Compact tricubic spline representation
- 8 corners × 8 coefficients
- 64 memory loads per evaluation
- C^2 continuity (smooth second derivatives)
- Formula:

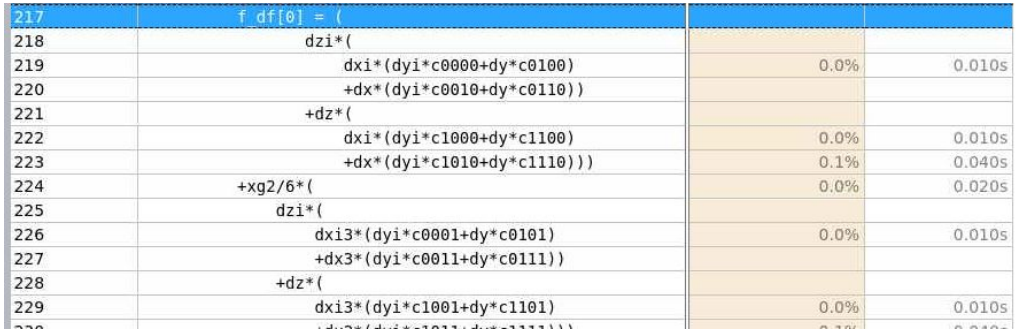
$$f(x,y,z) = \sum_{i,j,k} a_{ijk} \cdot B_i(x) \cdot B_j(y) \cdot B_k(z)$$



Performance Issues

- 64 loads per interpolation
- Random memory access
- Limited cache reuse
- Memory bandwidth bound
- Cost dominated by data movement
- If we skip only 2nd derivatives not very useful according to time spent in flops in
- ⟶ Interpolation becomes a bottleneck





Mini-app: 3D Spline Interpolation

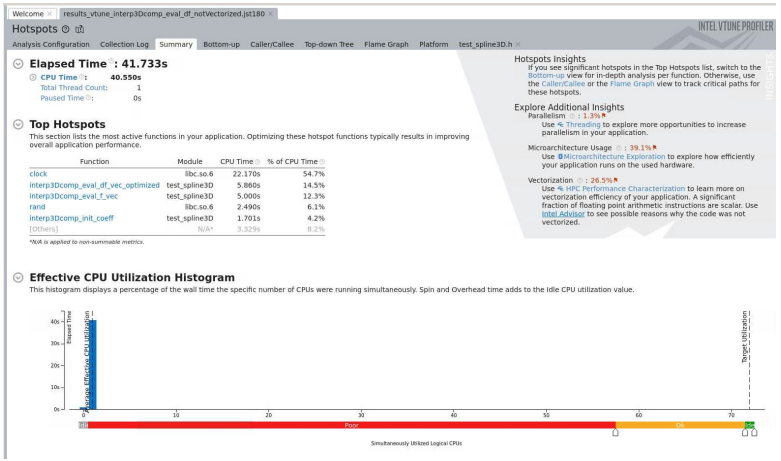
- Main hotspot
- Tricubic interpolation
- Standalone kernel extraction

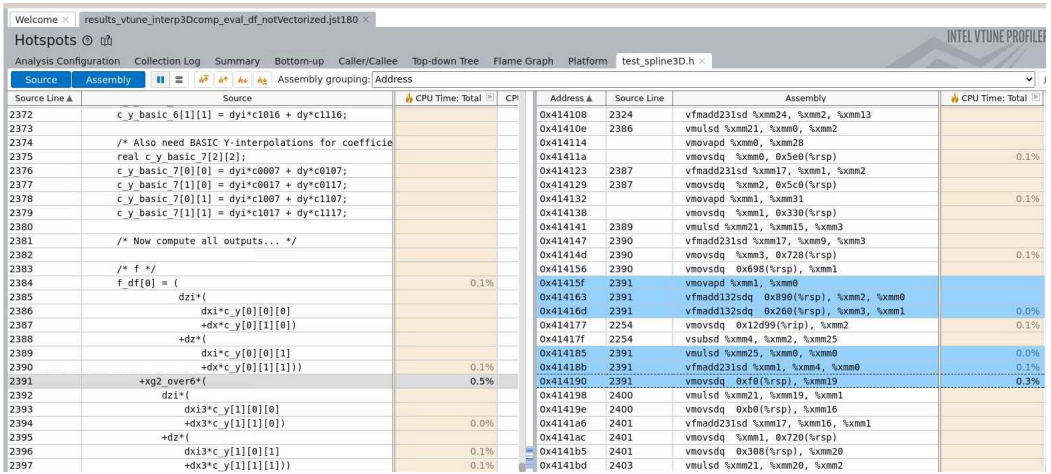
Spline Profiling Analysis

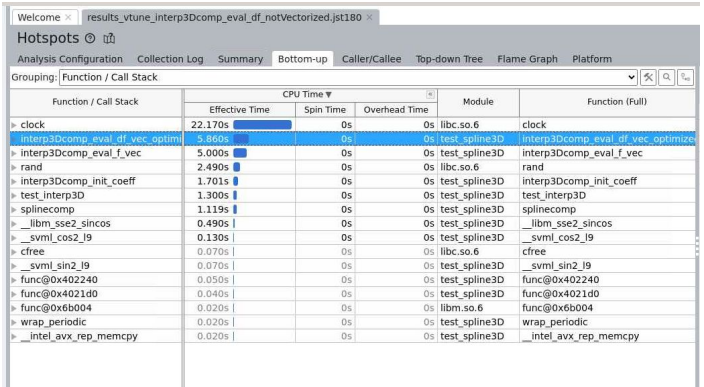
- Memory loads
- Instruction mix
- Cache reuse
- Bandwidth usage

Spline - CPU profiling

- Intel VTune
  - Single thread
  - No vectorization








```
LOOP BEGIN at test_spline3D.c (213, 1)
    remark #15571: simd loop was not vectorized: loop contains a recurrent c\
omputation that could not be identified as an induction or reduction.  Try u\
sing #pragma omp simd reduction/linear/private to clarify recurrence.
```

Spline - CPU profiling

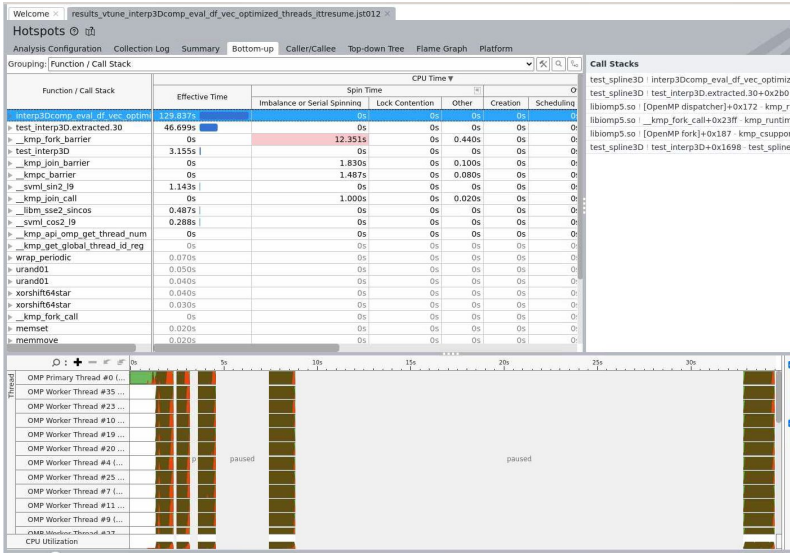
- Miniapp

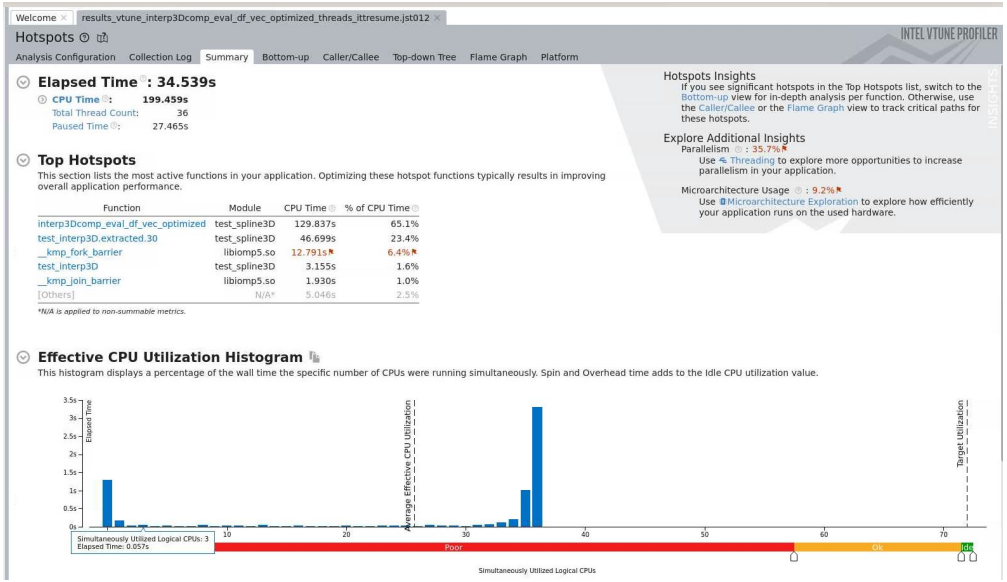
```
#pragma omp declare simd
__attribute__((always_inline)) inline
a5err interp3Dcomp_eval_df(real* f_df, interp3D_data* str,
                        real x, real y, real z) {

…
            real c0000 = str->c[n+0];
            real c0001 = str->c[n+1];
            real c0002 = str->c[n+2];
…
            /* Evaluate spline values */

            /* f */
            f_df[0] = (
            dzi*(
                        dxi*(dyi*c0000+dy*c0100)
                        +dx*(dyi*c0010+dy*c0110))
            +dz*(

…
   /* d2f/dydz */
            f_df[9] = ygi*zgi*(
            (
            dxi*(c0000  -c0100)
            +dx*(c0010-c0110))
```

```
#pragma omp simd
 for(int i = 0; i < n_rnd; i++)
 {
  /* Draw random point */
  x_rnd = x_min + (real)rand()/(real)RAND_MAX*(x_max - x_min);
  y_rnd = y_min + (real)rand()/(real)RAND_MAX*(y_max - y_min);
  z_rnd = z_min + (real)rand()/(real)RAND_MAX*(z_max - z_min);
  /* Calculate the analytical value and derivatives */
  if(anl_func == TRIG)
  {
              df_anl[0] =  sin(x_rnd)*sin(y_rnd)*sin(z_rnd);
              df_anl[1] =  cos(x_rnd)*sin(y_rnd)*sin(z_rnd);
……..
              df_anl[9] =  sin(x_rnd)*cos(y_rnd)*cos(z_rnd);
  }
  else if(anl_func == EXP)
  {
              df_anl[0] = exp(-pow(x_rnd-CONST_PI, 2)
                                   -pow(y_rnd-CONST_PI, 2)
                                   -pow(z_rnd-CONST_PI, 2));
……..

              df_anl[9] = 4*exp(-pow(x_rnd-CONST_PI, 2)
                                   -pow(y_rnd-CONST_PI, 2)
                                   -pow(z_rnd-CONST_PI, 2))
                          *(y_rnd-CONST_PI)*(z_rnd-CONST_PI);
  }
  /* Evaluate spline interpolant, and time it cumulatively */
  if(rep == EXPL)
  {
              start = clock();
              interp3Dexpl_eval_f(&f_spl, &str, x_rnd, y_rnd, z_rnd);
              interp3Dexpl_eval_df(df_spl, &str, x_rnd, y_rnd, z_rnd);
              end = clock();
  }
  else if(rep == COMP)
  {
              start = clock();
              interp3Dcomp_eval_f_vec(&f_spl, &str, x_rnd, y_rnd, z_rnd);
              interp3Dcomp_eval_df(df_spl, &str, x_rnd, y_rnd, z_rnd);
              end = clock();
  }
  cpu_time[1] = cpu_time[1] + ((double) (end-start))/CLOCKS_PER_SEC;
  /* Cumulate error */
  for(int j = 0; j < 10; j++)
  {
              err_df[j] += fabs(df_anl[j] - df_spl[j]);
  }
 }
```

Spline - CPU profiling

- Intel VTune
  - Multiple threads

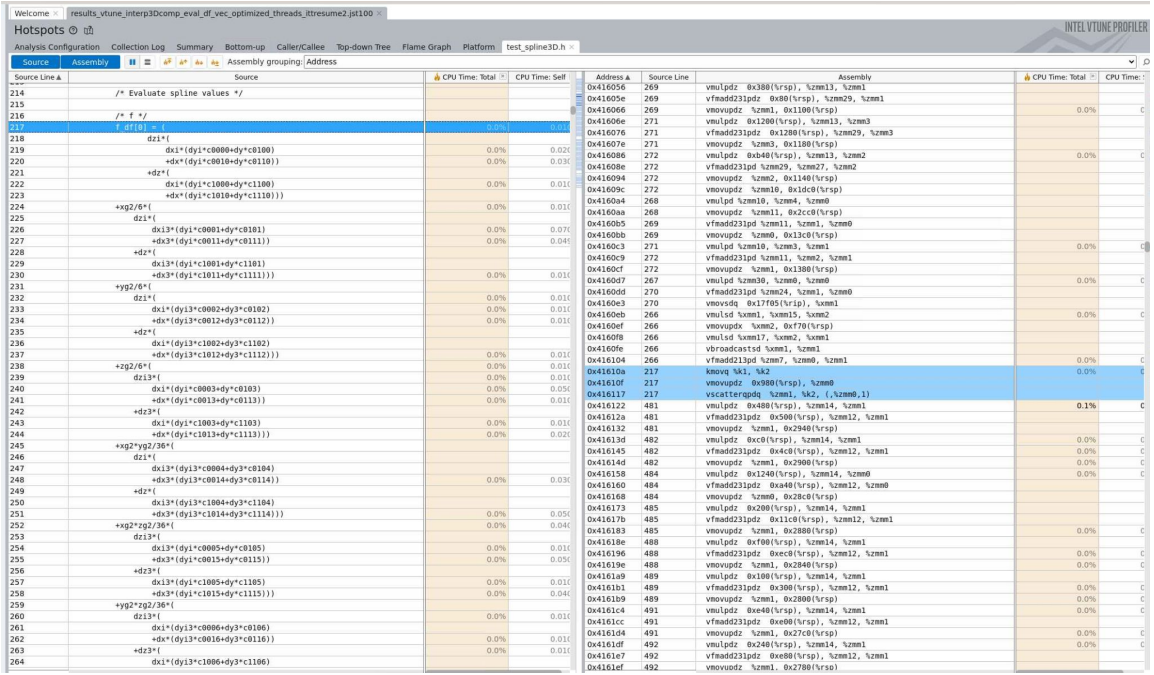




Spline - CPU profiling

- Intel VTune
  - Multiple threads
  - Vectorization: kernel splitting

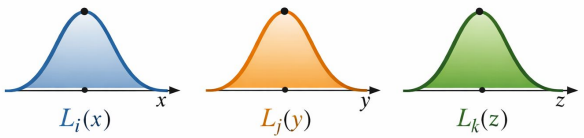
```
__itt_resume();   // <-- START profiling
#pragma omp parallel for simd
for(int i = 0; i < n_rnd; i++)
{
  int tid = omp_get_thread_num();
  real f_spl;
  real df_spl[10];
  real x_rnd = x_r[i];
  real y_rnd = y_r[i];
  real z_rnd = z_r[i];
  if(rep == EXPL)
  {
      interp3Dexpl_eval_f(&f_spl, &str, x_rnd, y_rnd, z_rnd);
      interp3Dexpl_eval_df(df_spl, &str, x_rnd, y_rnd, z_rnd);
  }
  else if(rep == COMP)
  {
          /* interp3Dcomp_eval_df_vec_optimized(df_spl, &str, x_rnd,
y_rnd, z_rnd); */
          interp3Dcomp_eval_df(df_spl, &str, x_rnd, y_rnd, z_rnd);
          f_spl = df_spl[0];

  }
  /* Cumulate error */
  for(int j = 0; j < 4; j++)
  {
      err_df_loc[tid*10+j] += fabs(df_anl[i][j] - df_spl[j]);
  }
}
__itt_pause();  // <-- STOP profiling
```



Triquadratic Alternative (27 points)

- Uses 3×3×3 = 27 grid points
- Quadratic Lagrange polynomials
- Only first derivatives required
- C^1 continuity
- Reduced memory traffic
- Trade-off: lower accuracy
- Formula:

$$f(x,y,z) = \sum_{i,j,k} f_{ijk} \cdot L_i(x) \cdot L_j(y) \cdot L_k(z)$$



Spline Performance Comparison - CPU

- **Performance:** Quadratic interpolation is about **1.8–2× faster** than tricubic in our benchmarks.
- **Accuracy:** Quadratic interpolation shows errors that are typically **2–3 orders of magnitude larger** than tricubic.
- **Precision trade-off:** This corresponds to a loss of accuracy of roughly **100–1000×** when switching from tricubic to quadratic.
- **Grid refinement behavior:** Typically, quadratic interpolation reduces the error as **~O(Δx³)**, while tricubic achieves **~O(Δx⁴)**, leading to much faster convergence for tricubic on refined meshes.

```
err3D_df = ...
[2.156403e-07 1.278646e-08 7.808901e-10 4.826328e-11 3.000212e-12
 1.415311e-06 1.650560e-07 2.002238e-08 2.468597e-09 3.066787e-10
 1.293181e-06 1.575842e-07 1.955249e-08 2.439873e-09 3.048827e-10
 1.415197e-06 1.650120e-07 2.002559e-08 2.468399e-09 3.066920e-10
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00];


cpu_time = ...
init: 3.268003e-09 comp: 1.076283e+01
init: 2.780294e-08 comp: 1.106228e+01
init: 2.774050e-07 comp: 1.586455e+01
init: 2.671721e-06 comp: 1.688363e+01
init: 2.377185e-05 comp: 1.756516e+01
Total time = 72.138474
```

Tricubic

```
err3D_df = ...
[7.602645e-05 1.504860e-05 3.396656e-06 8.221558e-07 2.049879e-07
 5.840462e-04 2.390699e-04 1.069041e-04 5.106332e-05 2.519882e-05
 2.101840e-04 5.195958e-05 1.291684e-05 3.222167e-06 8.063628e-07
 5.819221e-04 2.372244e-04 1.063061e-04 5.026960e-05 2.496501e-05
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00];


cpu_time = ...
init: 3.294945e-09 comp: 4.729121e+00
init: 2.806807e-08 comp: 5.518665e+00
init: 2.768760e-07 comp: 5.987247e+00
init: 2.658788e-06 comp: 1.064214e+01
init: 2.397984e-05 comp: 1.157540e+01
Total time = 38.452602
```

Quadratic

Spline Performance Comparison – ASCOT5 GPU

- **Performance:** Quadratic interpolation version is about **4 × faster** than tricubic one in our benchmarks.

```
Simulation complete.
Simulation finished in 228.595737 s
Endstate written.

Combining and writing diagnostics.

Writing diagnostics output.

Diagnostics output written.
Diagnostics written.

Summary of results:
   858084 markers had end condition Sim time limit
   141902 markers had end condition Wall collision
          14 markers had end condition Sim time limit and Wall
collision

           No markers were aborted.

Done.
```

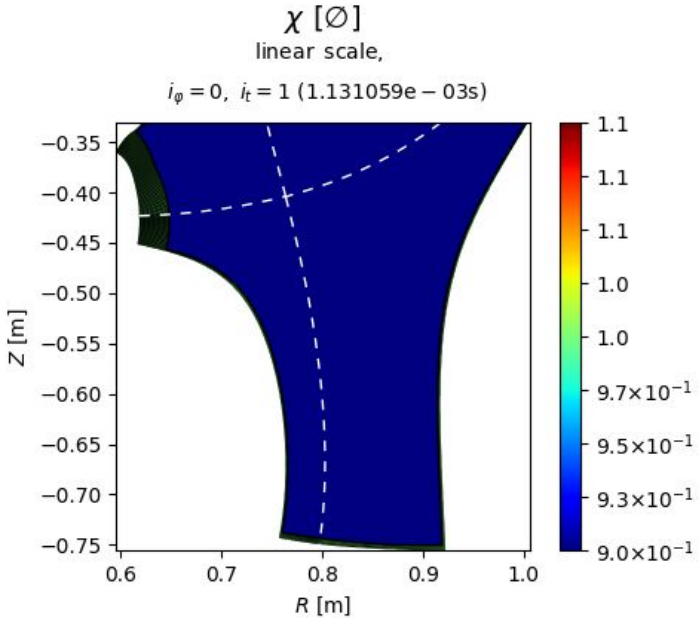
Tricubic

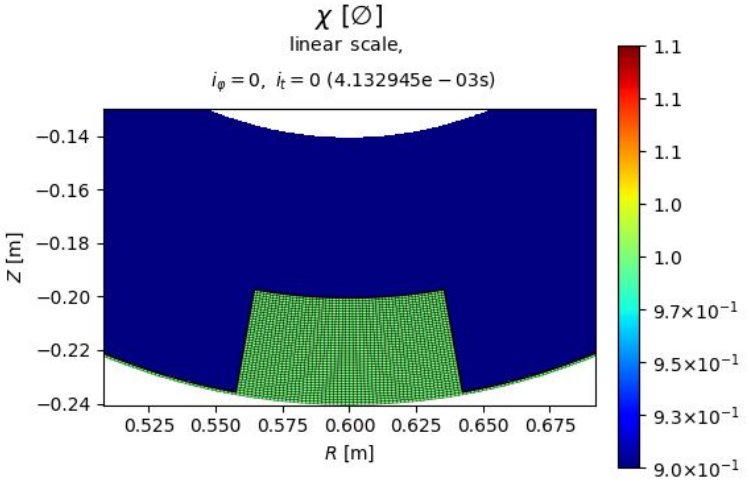
```
Simulation complete.
Simulation finished in 55.581174 s
Endstate written.

Combining and writing diagnostics.

Writing diagnostics output.

Diagnostics output written.
Diagnostics written.

Summary of results:
   858081 markers had end condition Sim time limit
   141905 markers had end condition Wall collision
          14 markers had end condition Sim time limit and Wall
collision

           No markers were aborted.
```

Quadratic

SOLEDGE3X: Boundary Conditions

- Wall shape defined through mask function χ
- 2 types of boundary conditions:
  - Edges of wall mask => imposed through fluxes for most of them
  - Edges of simulation domain, typically one coes as wall should surround the plasma





Mask Function and Load Imbalance

- Wall regions are handled using a mask function (chi)
  - MPI domain decomposition balances the total number of cells per subdomain, **without distinguishing masked and unmasked cells**
  - Each MPI process may contain a different number of masked cells
  - This leads to uneven computational workload
- Implicit Scheme:
  - Matrix and RHS are built using an if-condition on the mask
  - Only unmasked cells are solved
  - Processes with many masked cells do less work
- Explicit Scheme:
  - No conditional exclusion of masked cells
  - Computation performed on all cells
  - Mask applied via (1 - chi)
  - More uniform workload across MPI ranks

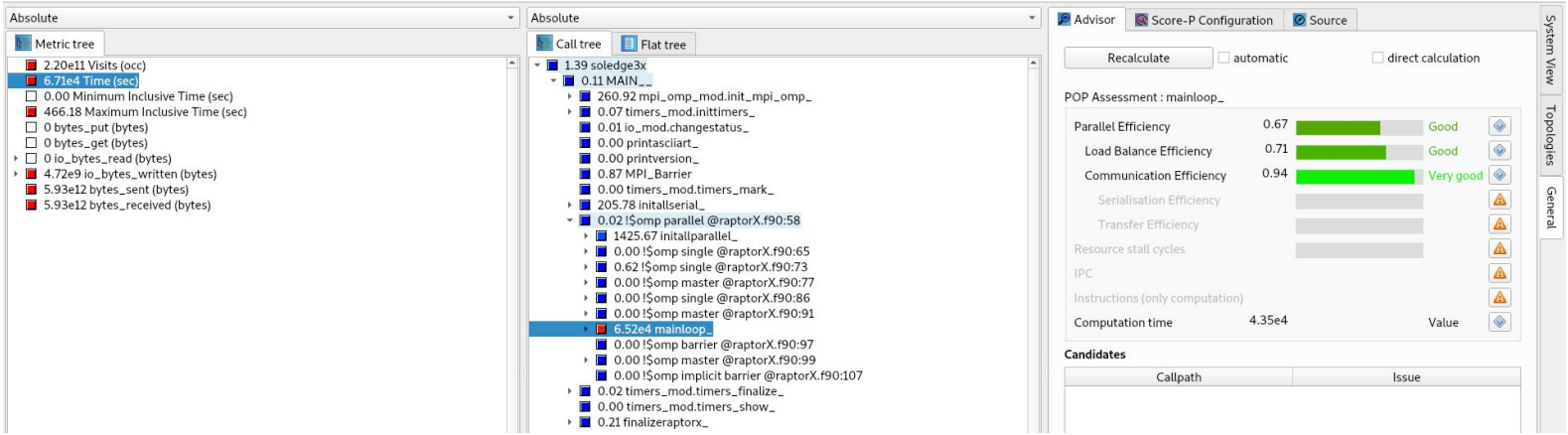
Profiling with Scorep

- Main loop: Scorep analysis for 144 MPI processes
  - 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

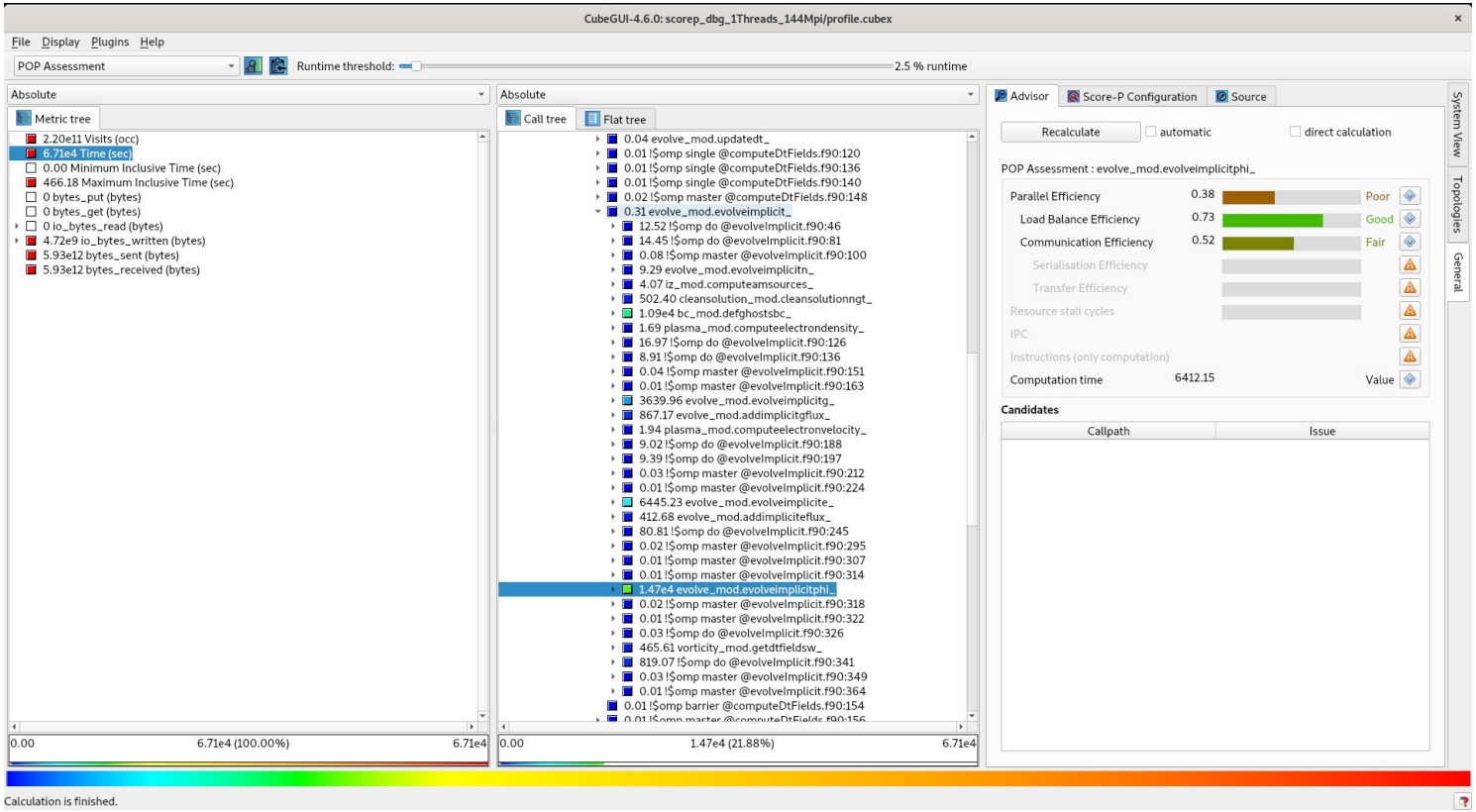
  - Load balance efficiency (*ratio between average useful computation time - across all processes - and maximum useful computation time - also across all processes -* :

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



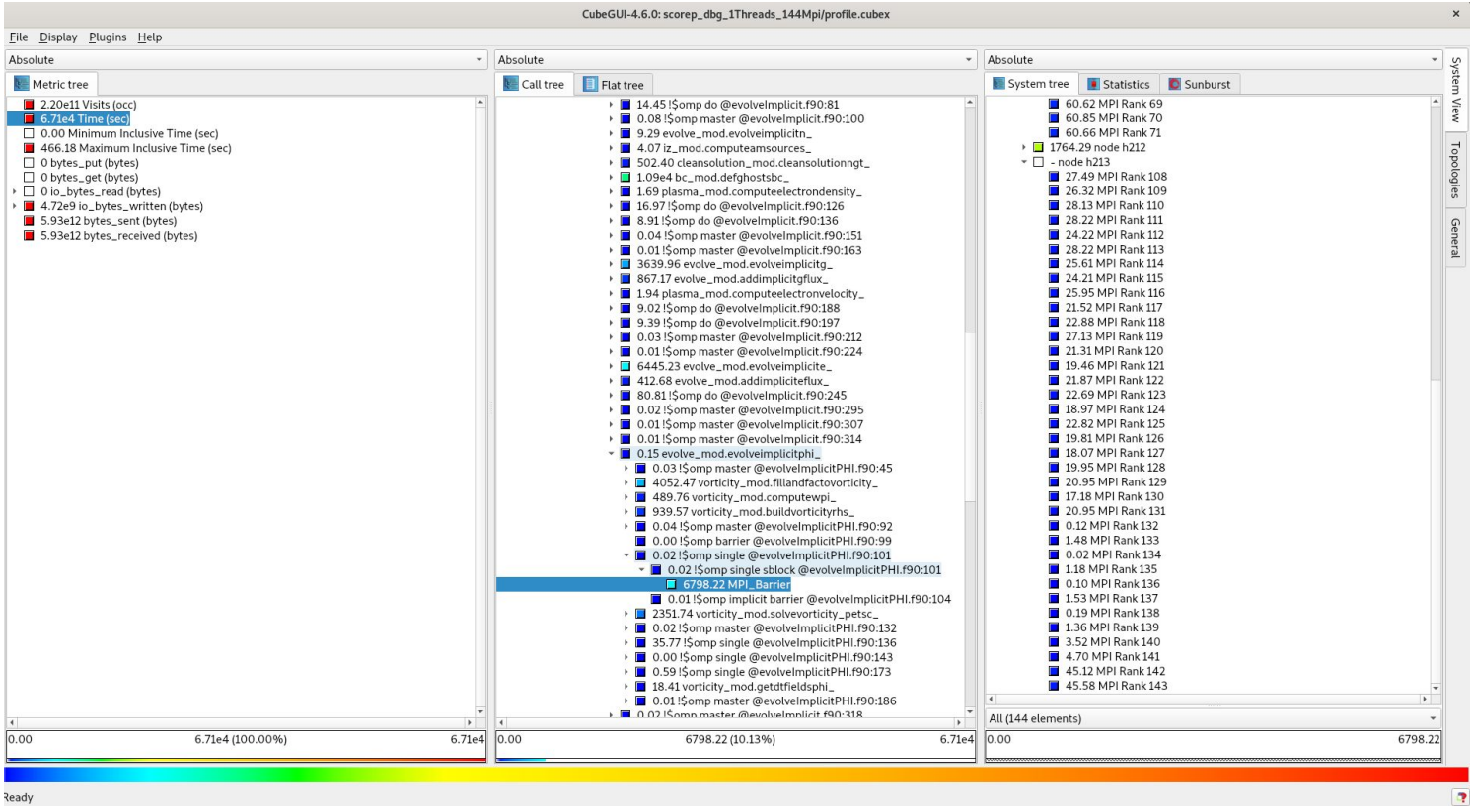
Profiling with Scorep

- evolveImplicitPHI routine: MPI barrier take most of the time



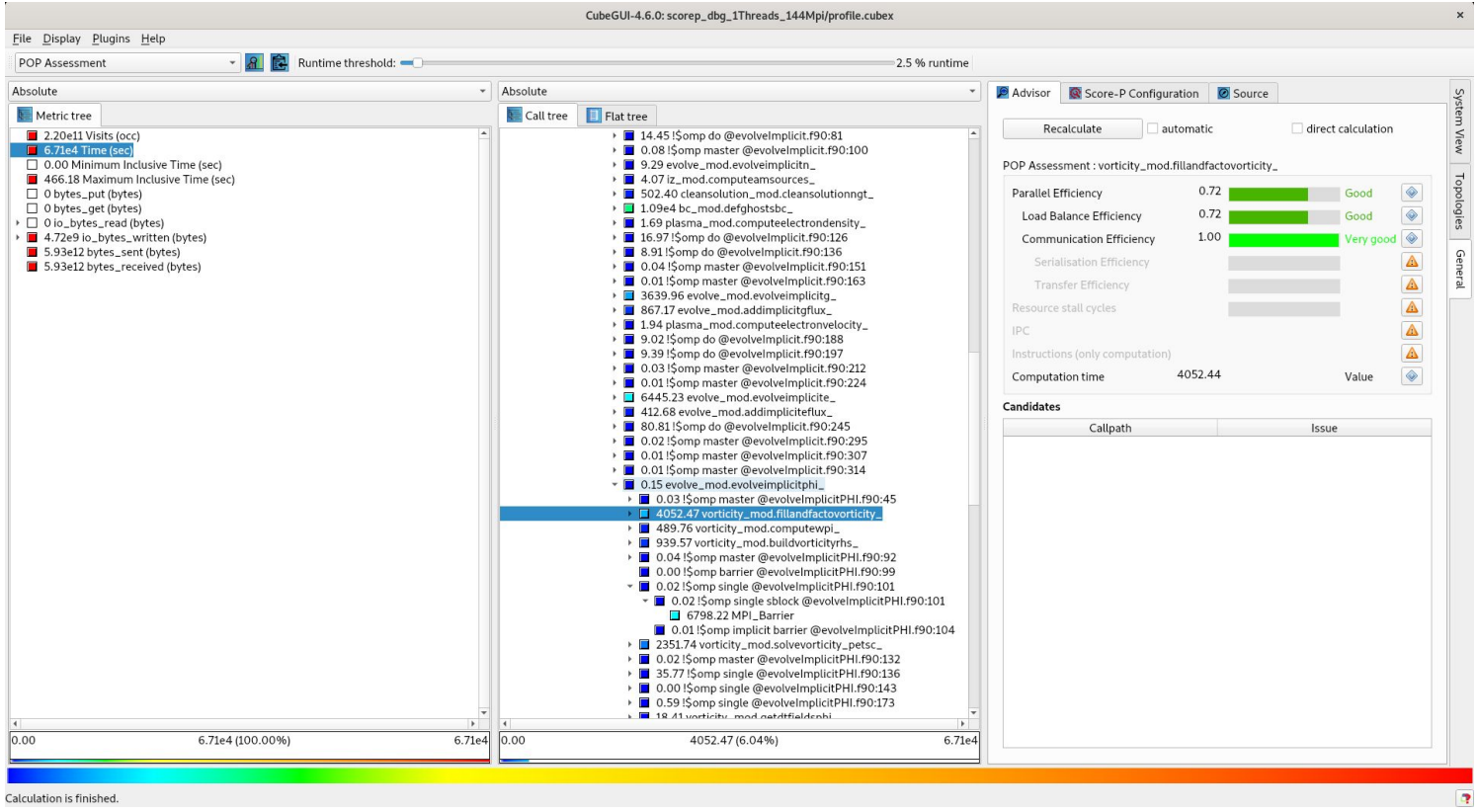
Profiling with Scorep

- evolveImplicitPHI routine: MPI barrier take most of the time



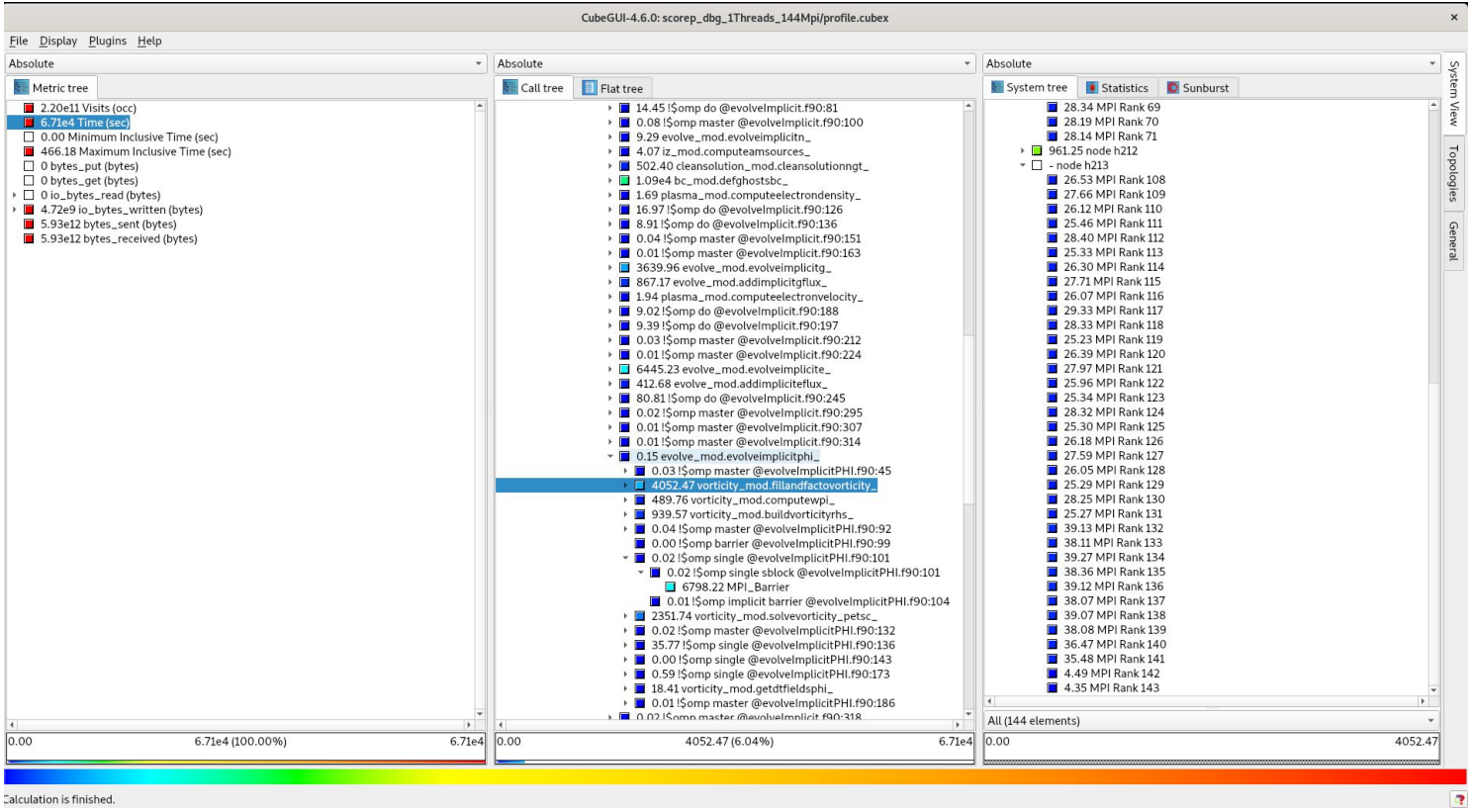
Profiling with Scorep

- evolveImplicitPHI routine: MPI barrier take most of the time
→ load imbalance between MPI processes (presence of penalization mask to take into account wall and workload imbalance between magnetic flux surfaces)



Profiling with Scorep

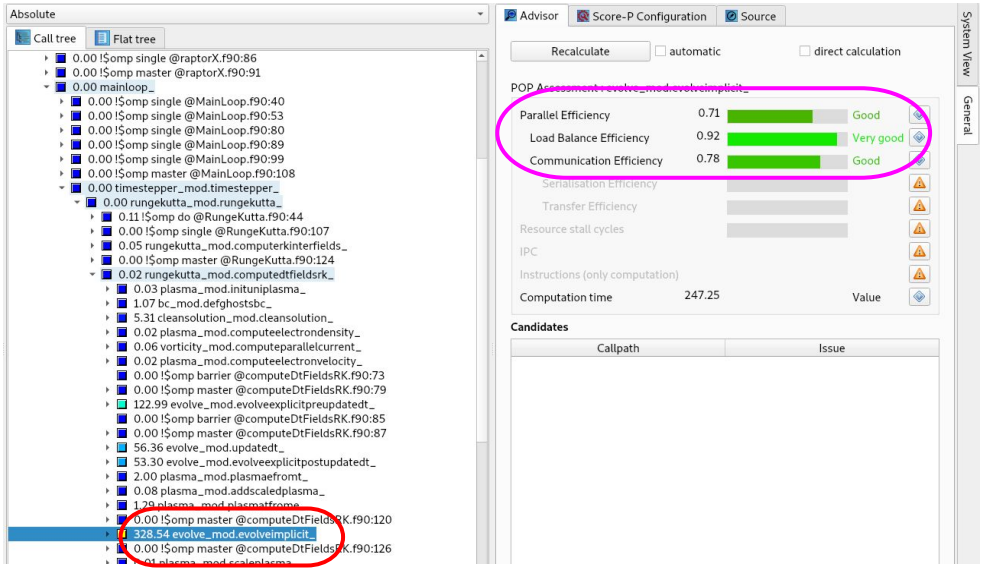
- evolveImplicitPHI routine: MPI barrier take most of the time
  → load imbalance between MPI processes (presence of penalization mask to take into account wall and workload imbalance between magnetic flux surfaces)



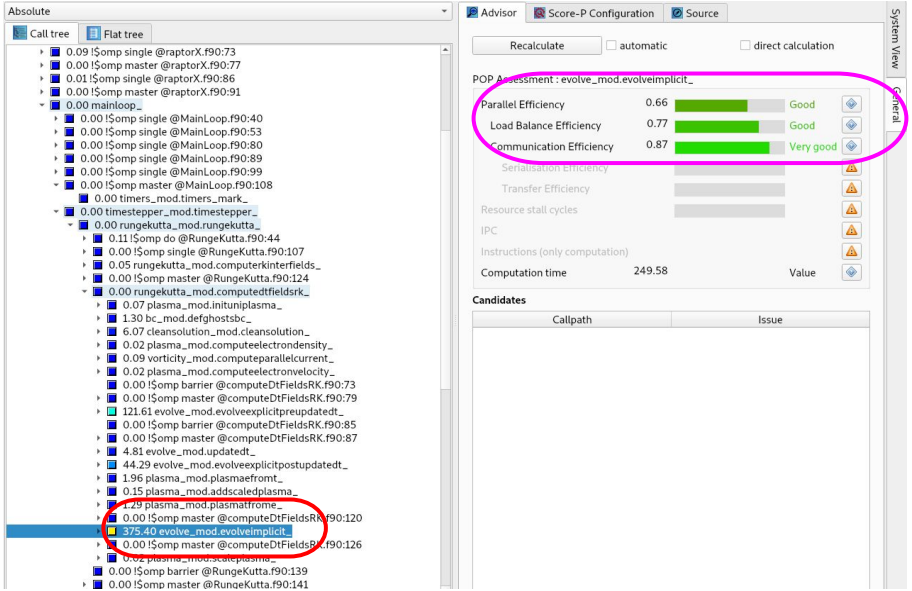
Profiling with Scorep

- New version giving a weight to “Penalized” cells in MPI domain decompostion:
  - Penalized (wall) cells are assigned a weight < 1
  - MPI decomposition accounts for their reduced computational cost
  - Better workload balance across processes

Implicit module
New version - Weight=0.1)

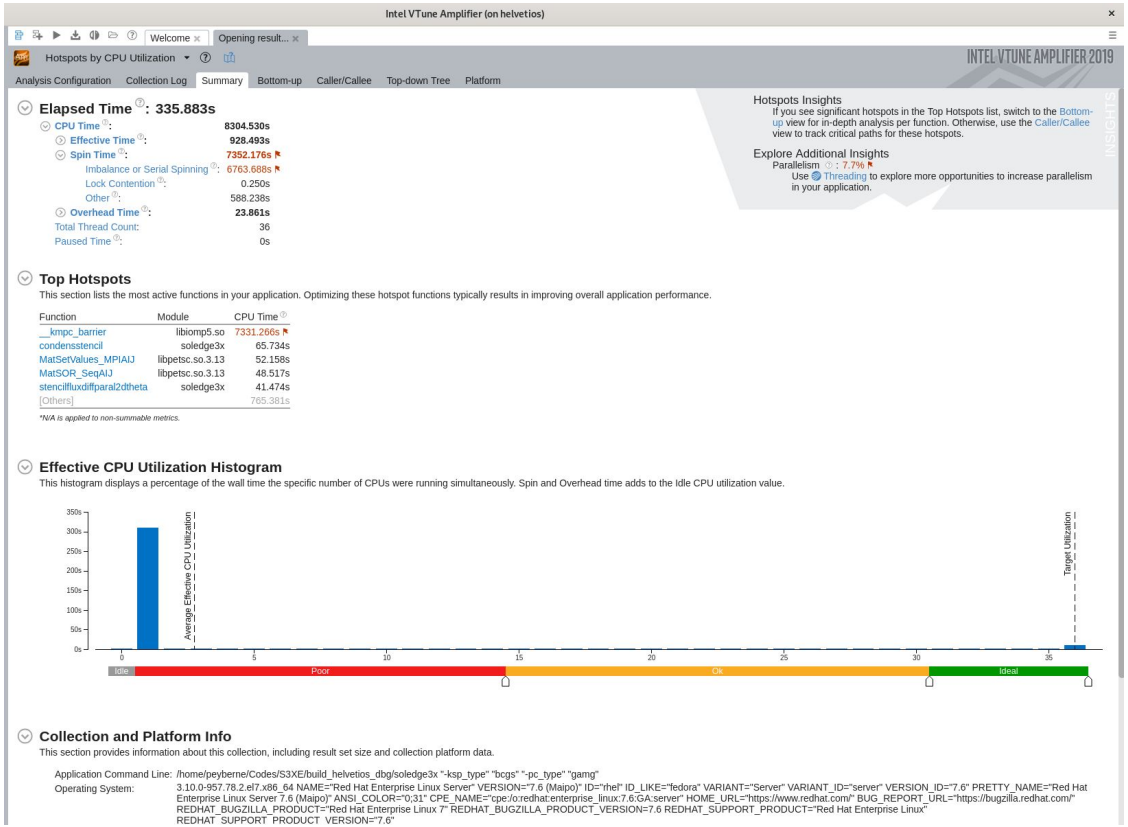


Implicit module
(Initial Version)



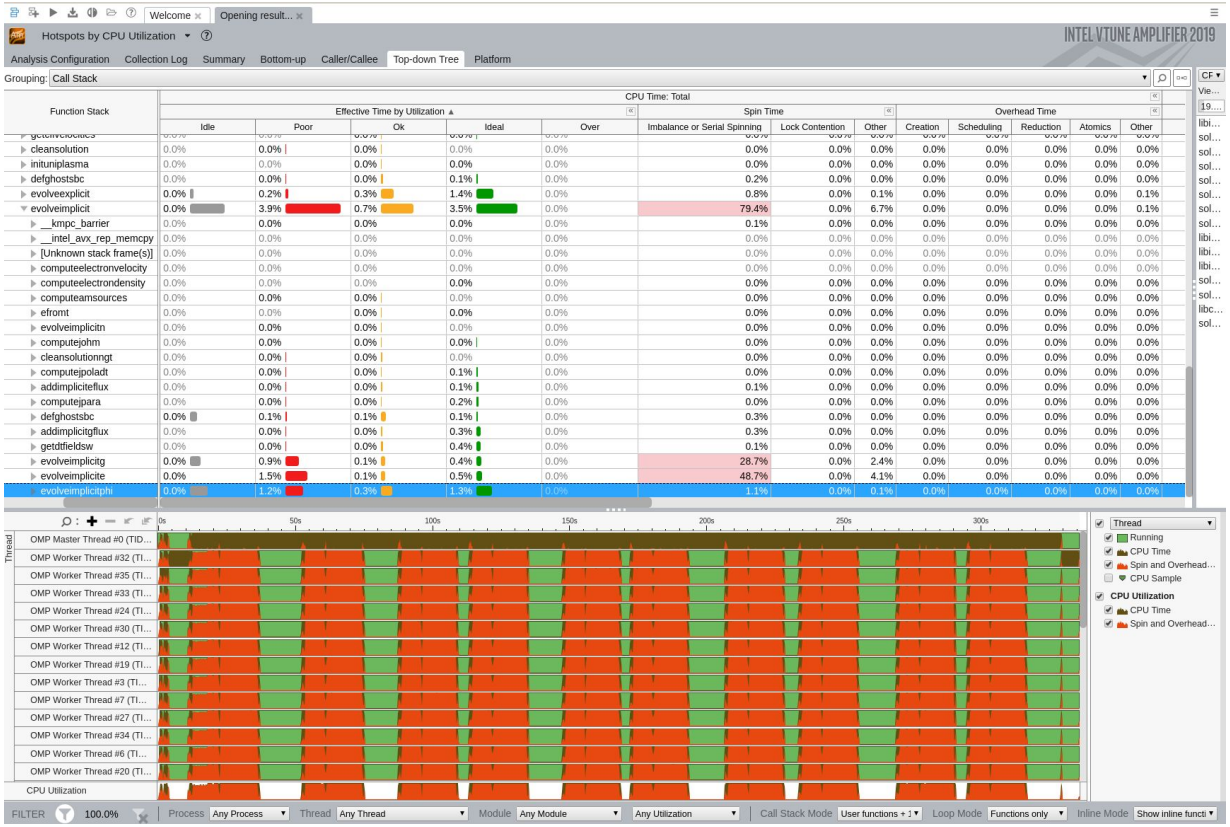
intra-node profiling

- Intel-Vtune



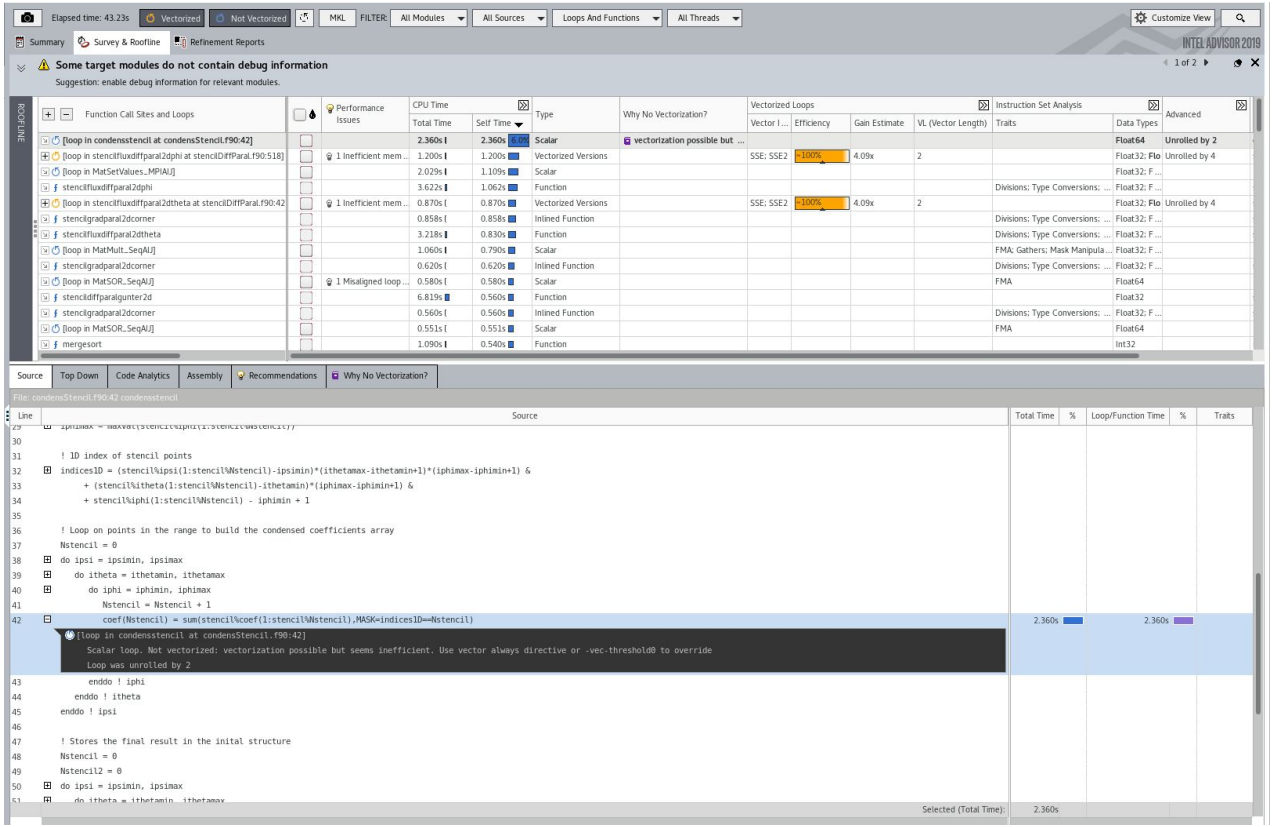
intra-node profiling

- Intel-Vtune



intra-node profiling

- Intel-Advisor



intra-node profiling

- Vectorization:
  - example of non vectorized loop (using *-qopt-report=5* compilation option) in *computeZhdanov* (15% of computeEpxl routine):

```
LOOP BEGIN at ./computeZhdanov.f90(107,12)
   remark #15388: vectorization support: reference NI(ispec) has aligned access   [ ./computeZhdanov.f90(108,15) ]
   remark #15388: vectorization support: reference UI(ispec) has aligned access   [ ./computeZhdanov.f90(109,15) ]
   remark #15388: vectorization support: reference TI(ispec) has aligned access   [ ./computeZhdanov.f90(111,15) ]
   remark #15335: loop was not vectorized: vectorization possible but seems inefficient. Use vector always directive or -vec-threshold0 to override
   remark #15328: vectorization support: non-unit strided load was emulated for the variable <fieldsloc(ichunk,ispec)>, stride is unknown to compiler   [\
 ./computeZhdanov.f90(108,25) ]
```

the data access in the loop on ispec using the array *fieldsLoc(ichunk)%spec(ispec)%n(iphi,itheta,ipsi)* is not stride1:

```
do ipsi = ipsimin, ipsimax
  do itheta = ithetamin, ithetamax
    do iphi = iphimin, iphimax
          do ispec=0,Nspecies
             ni(ispec)=fieldsLoc(ichunk)%spec(ispec)%n(iphi,itheta,ipsi)
             ui(ispec)=fieldsLoc(ichunk)%spec(ispec)%G(iphi,itheta,ipsi)&
             /fieldsLoc(ichunk)%spec(ispec)%n(iphi,itheta,ipsi)
             Ti(ispec)=fieldsLoc(ichunk)%spec(ispec)%T(iphi,itheta,ipsi)
          end do
```

Lessons Learned

- Importance of mini-apps
- Instrumentation value
- Memory vs compute limits

Best Practicies

- Start global
- Use roofline
- Instrument hotspots
- Validate optimizations

Kokkos Instrumentation

- - Kokkos Tools interface
- - Performance hooks
- - Portable instrumentation

Profiling a Kokkos application

Kokkos provides a large number of tools that can be used for profiling

- KernelFilter
- KernelSampler
- MemoryHighWater
- MemoryUsage
- MemoryEvents
- SimpleKernelTimer
- KernelLogger
- VTuneConnector
- VTuneFocusedConnector
- NVTXConnector
- Timemory

Questions

- Thank you for your attention
- Acknowledgements
  - Eurofusion
  - EPFL SPC/SCITAS
  - CEA/IRFM, VTT
  - SCITAS HPC group
  - NVIDIA/INTEL