pyAT and Matlab AT with GPU

Accelerator Middle Layer Workshop

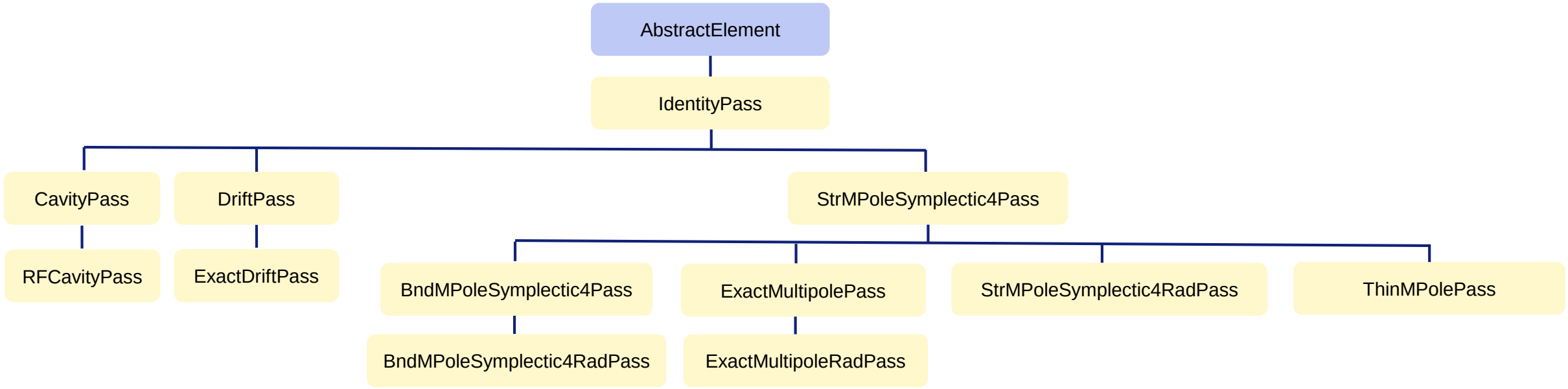
JUNE 19-21, DESY Hamburg

Jean-Luc PONS
E.S.R.F.

STATUS

- Standard passMethods and exact passMethods implementation done (available from github.com/atcollab/at branch gpu_tracking)
- GPU tracking available from both python and matlab
- Build and setup scripts available for Windows/Linux, python/matlab
- MacOS development in progress
- Evaluation of Metal (MSL) foreseen for MacOS (OpenCL deprecated on MacOS)
- OpenCL and Cuda implemented
- Parallel tracking starting from several references development in progress
- Parallel tracking in multiple RING foreseen
- Multiple GPU foreseen

C++ IMPLEMENTATION OVERVIEW (PASS METHODS CODE GENERATION)

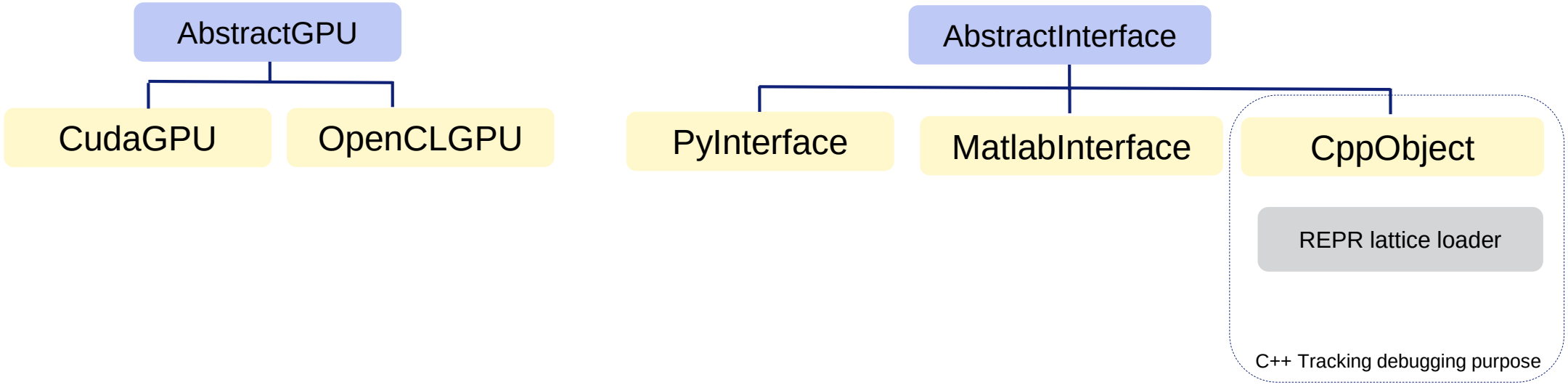


- AbstractElement: GPU memory transfer, access to ring data from python/matlab (method declarations)
- IdentityPass: Translation, Rotation, Aperture check
- StrMPoleSymplectic4Pass: Polynomial handling

- PassMethodFactory: Singleton class that creates elements and generates GPU code
- Lattice: Main class to run GPU kernel

- Each class implemented only its specificities.

C++ IMPLEMENTATION (INTERNAL)



- AbstractGPU: GPU compilation, running, loading (method declarations)
- CudaGPU: CUDA implementation
- OpenCLGPU: OpenCL implementation

- AbstractInterface: Access to ring data method declarations (used by pass methods)
- PyInterface : Interface to python implementation
- MatlabInterface : Interface to Matlab implementation

CODE GENERATION

- GPU code for integrator loops are automatically generated for all possible configurations

```
int nb = elem->mpole.NumIntSteps;
switch(elem->SubType) {
case 1:{
  for(int m = 0; m < nb; m++) {
    p_norm=PNORM(r6[4]);fastdrift(r6,SL*(AT_FLOAT)0.6756035959798289,p_norm);
    quadthinkickrad(r6,elem->mpole.PolynomA[0],elem->mpole.PolynomB[0],elem->mpole.K,SL*(AT_FLOAT)1.3512071919596578,elem->mpole.CRAD,p_norm);
    p_norm=PNORM(r6[4]);fastdrift(r6,SL*(AT_FLOAT)-0.1756035959798289,p_norm);
    quadthinkickrad(r6,elem->mpole.PolynomA[0],elem->mpole.PolynomB[0],elem->mpole.K,SL*(AT_FLOAT)-1.7024143839193155,elem->mpole.CRAD,p_norm);
    p_norm=PNORM(r6[4]);fastdrift(r6,SL*(AT_FLOAT)-0.1756035959798289,p_norm);
    quadthinkickrad(r6,elem->mpole.PolynomA[0],elem->mpole.PolynomB[0],elem->mpole.K,SL*(AT_FLOAT)1.3512071919596578,elem->mpole.CRAD,p_norm);
    p_norm=PNORM(r6[4]);fastdrift(r6,SL*(AT_FLOAT)0.6756035959798289,p_norm);
  }
```

- Code generation overhead

```
Ring parsing: 77.288ms (REPR Loader on EBS)
GPU Code generation: 0.158ms
GPU Code compilation: 17.279ms
GPU lattice loading: 1.98ms
```

SYMPLECTIC INTEGRATOR

- Symplectic integrator loops are automatically generated (6 integrator types supported)

1 Euler 1st
2 Optimal 2nd order from "The Accuracy of Symplectic Integrators", R. Mclachlan P Atela, 1991
3 Ruth 3rd
4 Forest/Ruth 4th
5 Optimal 4th order from "The Accuracy of Symplectic Integrators", R. Mclachlan P Atela, 1991
6 H. Yoshida 6th, "Construction of higher order symplectic integrators", 1990

Symplecticity check (6D tracking, misalignment of 5 microns RMS, EBS lattice)

```
J6 = jmat(3)                                    # block matrix, from at.physics.amat import jmat
M66 = ring.find_m66()[0]
VS = (M66.T.dot(J6).dot(M66) - J6).flatten()**2
RMS66 = np.sqrt(np.sum(VS))                     # should be 0 when 100% symplectic
```

RFCavity OFF, Radiation OFF, SL is the slice length

```
CPU(SL=0.010) RMS66=3.020310206081857e-09
GPU(SL=0.010,$Eul_{1st}) RMS66=1.7971693218050667e-09
GPU(SL=0.010,$McL_{2nd}) RMS66=3.5984645974252943e-09
GPU(SL=0.010,$Rut_{3rd}) RMS66=1.6426534063683773e-09
GPU(SL=0.010,$F/R_{4th}) RMS66=5.136143080445193e-09
GPU(SL=0.010,$McL_{4th}) RMS66=3.857426258406334e-09
GPU(SL=0.010,$Yos_{6th}) RMS66=5.755868260401188e-09
```

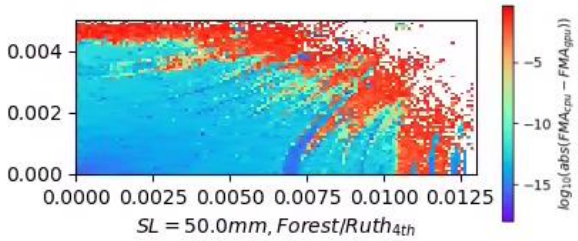
With Radiation ON, the impact of the damping is clearly visible

```
CPU(SL=0.010) RMS66=0.013576939942364072
GPU(SL=0.010,$Eul_{1st}) RMS66=0.012741108407377816
GPU(SL=0.010,$McL_{2nd}) RMS66=0.013472996199817283
GPU(SL=0.010,$Rut_{3rd}) RMS66=0.013577271012208037
GPU(SL=0.010,$F/R_{4th}) RMS66=0.013576940544098255
GPU(SL=0.010,$McL_{4th}) RMS66=0.013577264165442464
GPU(SL=0.010,$Yos_{6th}) RMS66=0.013576865317977516
```

- Polynomial are optimized for basic type (KickAngle, Quad, DQ, Sextu, Octu)

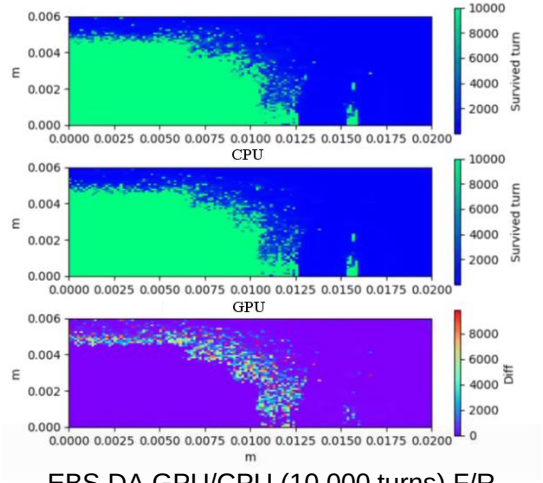
SYMPLECTIC INTEGRATOR / ACCURACY / PERFORMANCE

- FMA (EBS 1024 turns)
  SL = slice length

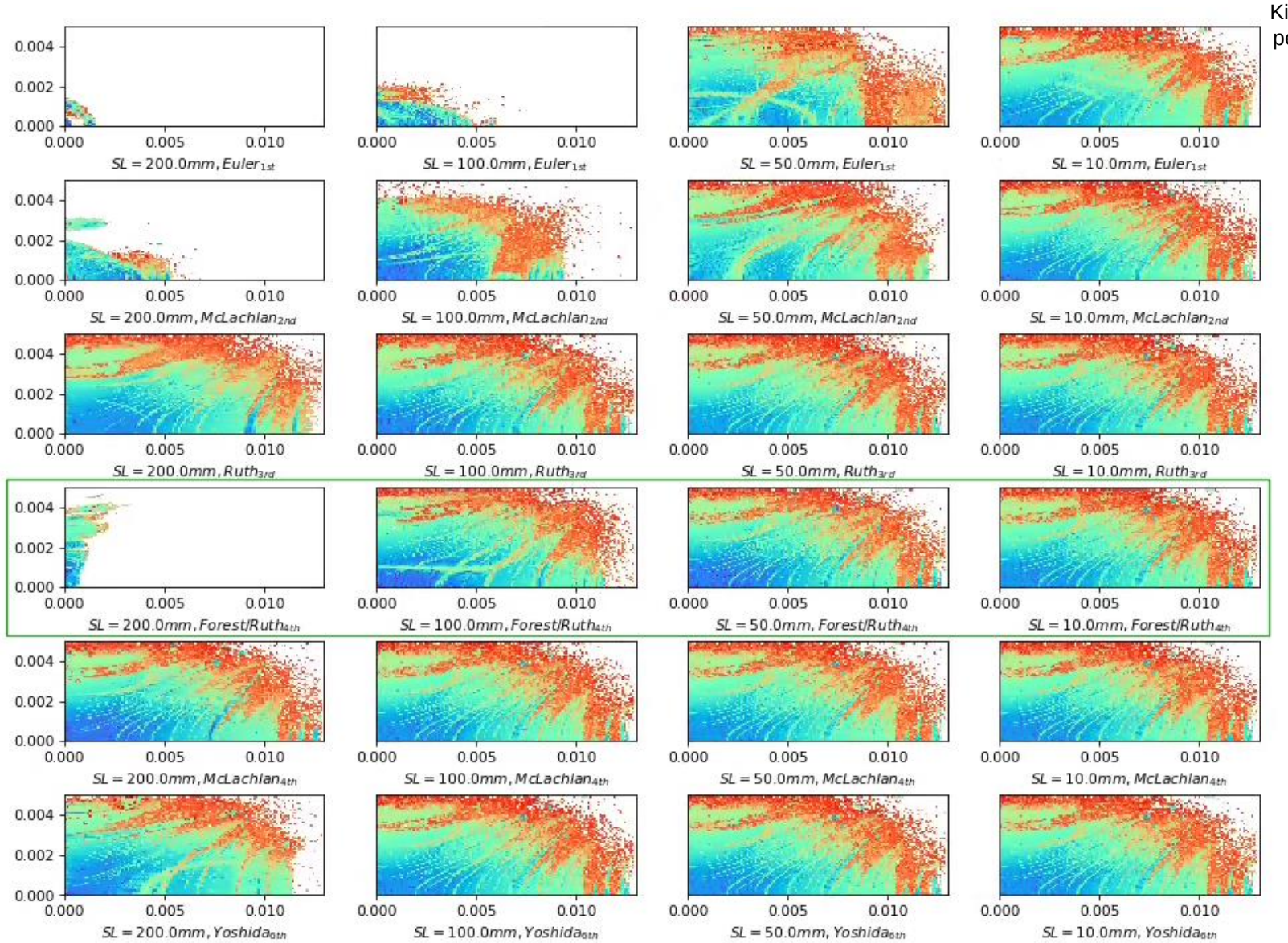


Standard passMethod
CPU 14 cores (Xeon@4.1GHz) **199.4 sec**
GPU (RTX A4500) **36.4 sec**
GPU/Single CPU: **77.5**

Exact passMethod
CPU 14 cores (Xeon@4.1GHz) **265.1 sec**
GPU (RTX A4500) **46.6 sec**
GPU/Single CPU: **79.6**

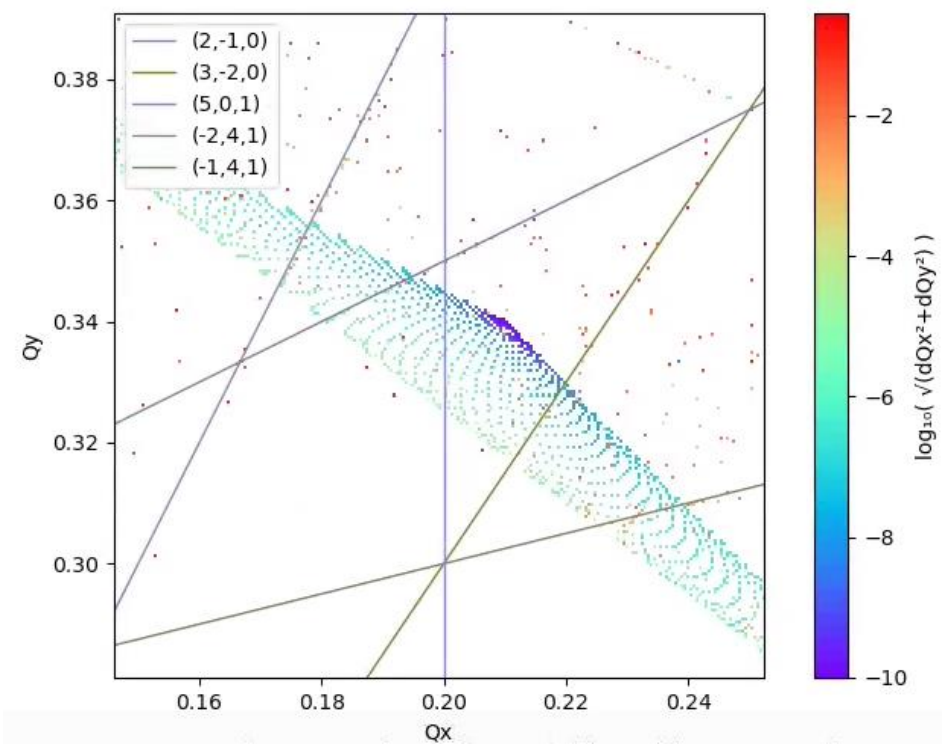


EBS DA GPU/CPU (10.000 turns) F/R_{4th}

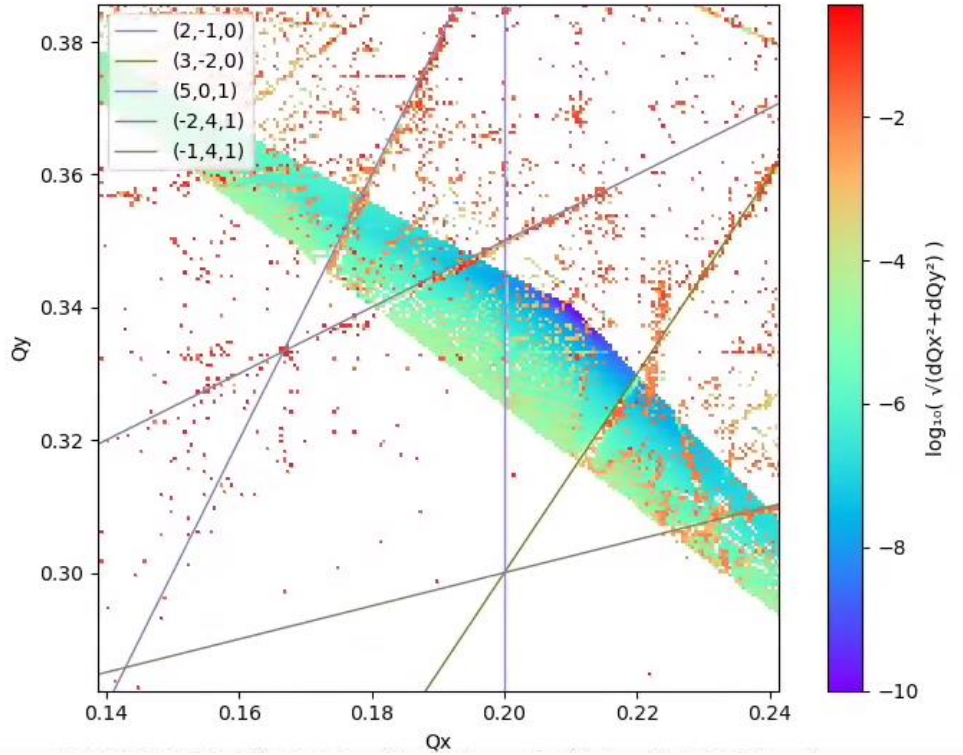


SYMPLECTIC INTEGRATOR / ACCURACY / PERFORMANCE

- FMA in tune space using F/R_{4th}



CPU



GPU

the fitting of the killing resonance lines is good

PRICE / PERFORMANCE

- 1 x **RTX A4500** ≈ ~90 x CPU@4.1GHz (no loss particle, ~1MB lattice of ~3300 elements, GPU at 100%)

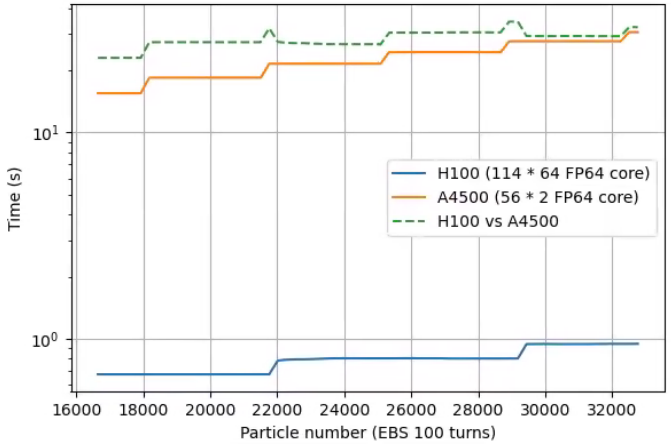
SPECIFICATIONS	
GPU memory	20GB GDDR6
Memory interface	320-bit
Memory bandwidth	640 GB/s
Error-correcting code (ECC)	Yes
NVIDIA Ampere architecture-based CUDA Cores	7,168
NVIDIA third-generation Tensor Cores	224
NVIDIA second-generation RT Cores	56
Single-precision performance	23.7 TFLOPS[5]
RT Core performance	46.2 TFLOPS[5]
Tensor performance	189.2 TFLOPS[4]
NVIDIA NVLink	Low profile bridges connect two NVIDIA RTX A4500 GPUs[1]
NVIDIA NVLink bandwidth	112.5 GB/s (bidirectional)
System interface	PCIe 4.0 x16
Power consumption	Total board power: 200 W
Thermal solution	Active
Form factor	4.4" H x 10.5" L, dual slot, full height
Display connectors	4x DisplayPort 1.4
Max simultaneous displays	4x 4096 x 2160 @ 120 Hz, 4x 5120 x 2880 @ 60 Hz, 2x 7680 x 4320 @ 60 Hz
Power connector	1x 8-pin PCIe
Encode/decode engines	1x encode, 1x decode (+AV1 decode)
VR ready	Yes
Graphics APIs	DirectX 12 Ultimate, Shader Model 6.6, OpenGL 4.6[7], Vulkan 1.3[7]
Compute APIs	CUDA 11.6, DirectCompute, OpenCL 3.0

RTX A4500 has 8.6 compute capabilities:
- 56*128 FP32 units,
- 56*2 FP64 units,
- 56*64 INT32 units

H100 has 9.0 computes capabilities:
- 114*128 FP32 units,
- 114*64 FP64 units,
- 114*64 INT32 units

In 2023 (ESRF price):
H100 PICe ~34.000 euros
14 cores/28 threads 3.3GHz (4.8 GHz) Desktop + 2xRTX A4500 ~4800 euros

In 2024 (ESRF price):
96 cores/192 threads 2.5GHz (5.1GHz) Desktop + RTX A4000 ~17.736 euros

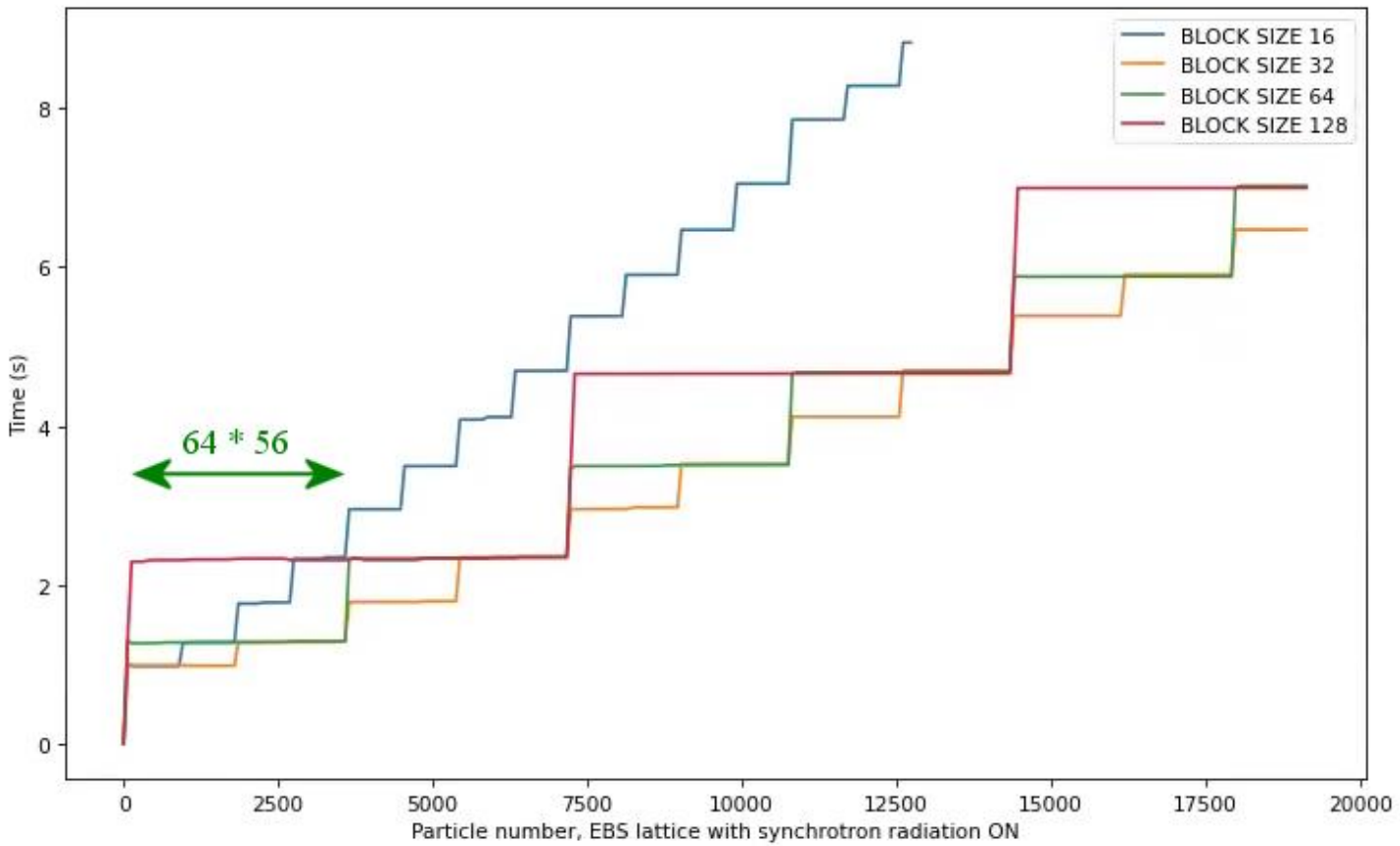


https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#compute-capabilities

```
➢ ./clpeak --compute-sp --compute-dp

Platform: NVIDIA CUDA
  Device: NVIDIA RTX A4500
    Driver version   : 535.146.02 (Linux x64)
    Compute units    : 56
    Clock frequency : 1650 MHz

    Single-precision compute (GFLOPS)
      float    : 23074.39

    Double-precision compute (GFLOPS)
      double   : 426.19
```

```
Platform: NVIDIA CUDA
  Device: NVIDIA H100 PCIe
    Driver version   : 535.129.03 (Linux x64)
    Compute units    : 114
    Clock frequency : 1755 MHz

    Single-precision compute (GFLOPS)
      float    : 49331.30

    Double-precision compute (GFLOPS)
      double   : 25186.19
```

PERFORMANCE MEASUREMENT AND LIMITATION



In CUDA, A warp is an “elementary” subset of **32 threads** within a thread block such that all the threads in a warp can execute a same instruction (SIMT) at the time in a single compute unit.

The RTX A4500 (8.6) has 4 warp schedulers. A compute unit can execute up to 128 threads in parallel.

56*128 (7168) threads can really be executed in parallel without serialization if required underlying units (FP32,FP64,INT32) are available.

To load a GPU at 100%, it is recommended to track a number of particles which is multiple of 64 times the number of compute units. AT use BLOCK_SIZE 64.

PERFORMANCE MEASUREMENT AND LIMITATION

- Do not rely on nvidia-smi for GPU usage !

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 535.146.02             Driver Version: 535.146.02   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA RTX A4500               On  | 00000000:17:00.0 Off |                  Off |
| 30%   35C    P2             73W / 200W |    198MiB / 20470MiB |    100%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   1  NVIDIA RTX A4500               On  | 00000000:65:00.0 Off |                  Off |
| 30%   32C    P8             24W / 200W |     79MiB / 20470MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
```

Single particle tracking

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 535.146.02             Driver Version: 535.146.02   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA RTX A4500               On  | 00000000:17:00.0 Off |                  Off |
| 30%   56C    P2            111W / 200W |    948MiB / 20470MiB |    100%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
|   1  NVIDIA RTX A4500               On  | 00000000:65:00.0 Off |                  Off |
| 30%   34C    P8             25W / 200W |     79MiB / 20470MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+
```

16384 particles tracking

USAGE (PYTHON)

- Main tracking function

```
rout, *_ = ring.track(rin, nturns=nturns, use_gpu=True, gpu_pool=[0], integrator=4)
rout, *_ = ring.track(rin, nturns=nturns, use_mp=True, pool_size=16)
```

- Acceptance

```
b2, s2, g2 =
ring.get_acceptance(planes,npoints,amplitudes,
grid_mode=at.GridMode.CARTESIAN,
nturns=nturns,
use_gpu=True,
offset=np.array([1e-7,0,1e-7,0,0,0]))
```

- GPU Info

```
Help on function gpu_info in module at.tracking.track:

gpu_info()
    :py:func:`gpu_info` returns list of GPU present on the system and their corresponding information. If GPU
    support is not enabled or if no capable device are present on the system, an empty list is returned.

    Returns:
        gpu: [gpu name,hardware version (CUDA device),compute unit number,platform]. The number of compute
        units is not the so-called number of "CUDA cores". The number of threads that can be executed simultaneously
        depends on the hardware and on the type of used instructions. For best performance, it is recommended to
        track a number of particles which is multiple of 64 (or 128) times the number of compute units.

>>> at.tracking.gpu_info()
[['NVIDIA RTX A4500', '8.6', 56, 'CUDA 12.2'], ['NVIDIA RTX A4500', '8.6', 56, 'CUDA 12.2']]
```

USAGE (MATLAB)

- Main tracking function

```
>> ring = atloadlattice("test/lattice/simple_ebs.mat");
>> rin = [0.001,0,0.001,0,0,0];
>> % rout = ringpass(ring,rin',100); % CPU Call
>> rout = ringpass(ring,rin',100,'gpuId',0);
>> rout(99*6:100*6-1)

ans =

   1.0e-03 *

   -0.5059    0.9495   -0.0113    0.7300    0.2937         0
```

- GPU Info

```
>> out = gpuinfo();
>> out{1}

ans =

  struct with fields:

          Name: 'NVIDIA RTX A4500'
       Version: '8.6'
    CoreNumber: 56
      Platform: 'CUDA 12.2'
```

USAGE (PYTON EVALUTATION)

- Parallel tracking starting from several refpts (development in progress)

```
Help on built-in function gpupass in module at.tracking.gpu:

gpupass(...)
    gpupass(line: Sequence[Element], r_in, n_turns: int, refpts: Uint32_refs = [], reuse:
Optional[bool] = False, omp_num_threads: Optional[int] = 0)

    Track input particles r_in along line for nturns turns.
    Record 6D phase space at elements corresponding to refpts for each turn.

    Parameters:
…
        tracking_starts: numpy array of indices of elements where tracking should start.
           len(tracking_start) must divide the number of particle and it gives the stride size.
           The i-th particle of rin starts at elem tracking_start[i/stride].
           The behavior is similar to lattice.rotate(tracking_starts[i/stride]).
           The stride size should be multiple of 64 for best performance.
…
```

```
startingElem = np.array([0,100,200,300,400,500,600,700],dtype=np.uint32)
rout, *_ = ring.track(rin, nturns=nturns, use_gpu=True, tracking_starts=startingElem)
```

INSTALLATION

- Python (https://github.com/atcollab/at/blob/gpu_tracking/docs/p/howto/multiprocessing.rst)

CUDA Installation

NVidia CUDA toolkit must be preliminary installed on the system from NVidia. Set the environment variable `CUDA_PATH`:

```
export CUDA_PATH=/cvmfs/hpc.esrf.fr/software/packages/ubuntu20.04/x86_64/cuda/12.3.1
```

or on Windows:

```
set CUDA_PATH=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.3
```

Install PyAT using the `cuda` flag:

```
pip install --config-settings cuda=1 .
```

You can check the install using the method `at.tracking.gpu_info()` as described above.

- MATLAB (R2021b, Ubuntu 20)

```
>>cd /home/esrf/pons/at
>>addpath(genpath(['./atmat']))
>>addpath(genpath(['./atintegrators']))
>>atmexall -cuda /cvmfs/hpc.esrf.fr/software/packages/ubuntu20.04/x86_64/cuda/12.3.1
```

THANKS

Accelerator Middle Layer Workshop

JUNE 19-21, DESY Hamburg

Jean-Luc PONS
E.S.R.F.