Particle tracking on a GPU

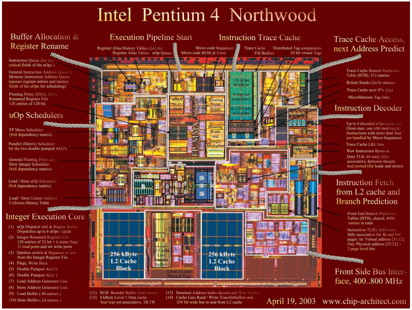
3 Oct 2023, AT workshop

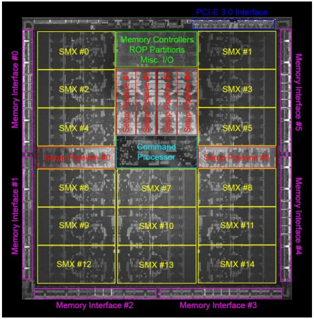
Dynamic aperture optimization takes forever!!!

- ALBA2 has many sextupoles.
- Dynamic aperture optimization is a minimization problem with high dimensionality.
- At each step of the minimization you need to compute the dynamic aperture
- **Computing the dynamic aperture takes time!!**

What can you do with a GPU?

CPU vs GPU (Part 1)

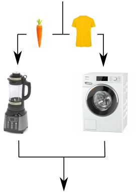




- CPU: A lot of effort into optimizing program flow execution (Efficient code branching is hard!!)
- CPU: Each core is almost an independent CPU: duplicated hardware
- CPU: Peripherals interfaces take a lot of space
- **GPU:** a **small number** of instruction fetch/decode units are **driving many cores in parallel**
- **GPU: no branch-prediction/out of order execution.**

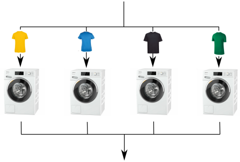
Tracking follows a simple execution flow:
all electrons fly through the same elements in the same order **No branching is required**
A tracking code does not require (almost) any "if" instruction

CPU vs GPU (Part 2)



- In a CPU cores can run **different programs**
  Cool! But we don't need it: we do only tracking!
- A CPU can have a few tens of cores

We can track a few **tens of particles** in parallel

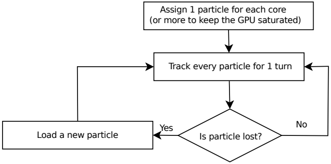


- In a GPU the **same instructions feed a group of cores**
  That's fine: every core is running the same tracking
- However, each core operates on **different data**
  Good: each particle has different coordinates and machine settings
- Each group includes typically 64 to 128 cores
- A GPU can have a few to 100 groups of cores

We can track a few **thousands of particles** in parallel

The old GPU in my office computer (Nvidia Quadro P600) is worth around **100$ has 3 groups of cores, each with 128 cores** → **384 total cores**. Big GPUs (such as the Nvidia Tesla or AMD Instinct series) cost around **7000$ and have 20 times more cores.** Big GPUs also **handle some instructions more efficiently** (e.g. transcendental functions) → faster execution.

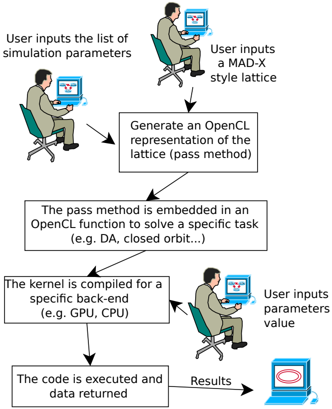
(dirty) Tricks and magic to make tracking fast!



- **merge** together **consecutive pass-methods**
  e.g. a straight section followed by a quadrupole can be represented as one single linear transformation. In UFO this simplification task is demanded to the OpenCL compiler: An **intermediate OpenCL representation of the 1-turn pass method** is generated and passed to the compiler for optimization
- **Pre-compute expressions** whenever possible
  The OpenCL compiler is smart enough to evaluate functions (e.g. sin/cos/sqrt) whenever possible.
- **32 bit** variables are most of the times **good enough**
  Watch out: the ieee754 CPU standard prescribes 80 bit for the internal representation of double variables. GPUs instead use 64/32 bit for double and float variables $\rightarrow$ CPU/GPU can behave differently
- High order **relativistic effects** can be **neglected** (sometimes)
  Relativistic effects are quite marginal in multi-bend achromat lattices, at least in the one I have tried

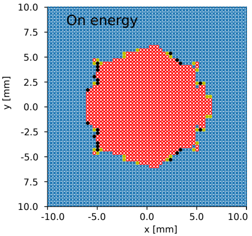
Watch out:
This considerations depend on the specific case
Do not apply this optimization without testing!!!

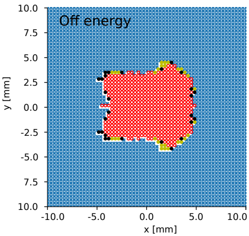
How does it work?



- **Optics parameters** (field strength, element lengths...) can be **specified per-particle**
- The list of per-particle parameters must be specified before compiling the pass-method
- Compiling takes time, but once done the simulation can be repeated for different parameters
- UFO **uses OpenCL**, a **standard promoted by** the Khronos group (**AMD, Amazon, Apple, ARM, Google, Intel, Microsoft, NEC, Nokia, NVIDIA, Samsung, Sony, Texas Instruments, Xilinx...**)
  CUDA is also a popular choice but is not standard, is proprietary and only for NVIDIA

Relativistic exact integrator vs classical integrator

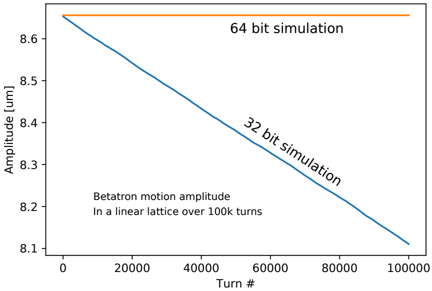




- The effect of using a purely classical integrator has been characterized by comparing a DA simulation against MAD-X/PTC with the exact relativistic Hamiltonian option switched on.
- Particle are tracked for 1000 turns using the ALBA2 lattice.
- Matching results are shown in blue (unstable) and red (stable)
- Differing results are yellow (stable in MAD and unstable in UFO) and black (unstable in MAD and stable in UFO)

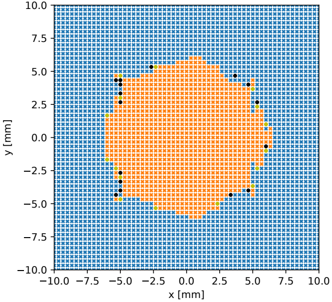
$\sim$ **5%** of the stable particles are **incorrectly tracked** for the on-energy case
$\sim$ **10%** in the **off-energy** case.

32 vs 64 bit variables representation



- **UFO can run using 32 or 64 bit** variables.
- On GPU **32** bit operations are substantially **faster** respect to 64 bit but **less precise**.
- In a 10^5 turns tracking the **loss of symplecticity** is clearly visible for the 32 bit integrator.
- **Dynamic aperture** simulations for electron rings require usually $\sim 10^3$ **turns**.

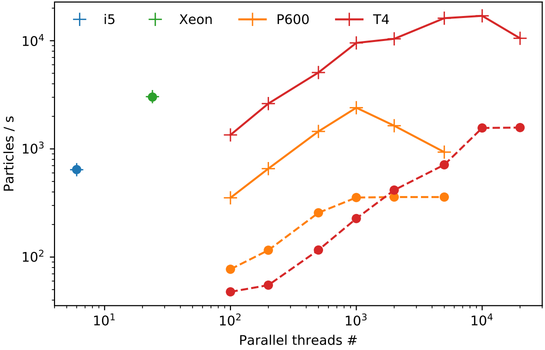
32 vs 64 bit variables representation



- Particle are tracked for **1000 turns** using the ALBA2 lattice.
- Matching results are shown in blue (unstable) and red (stable).
- Differing results are yellow (stable in 32 and unstable in 64) and black (unstable in 32 and stable in 64).

$\sim$ **3%** of the stable particles are **incorrectly tracked** when switching to 32 bit.

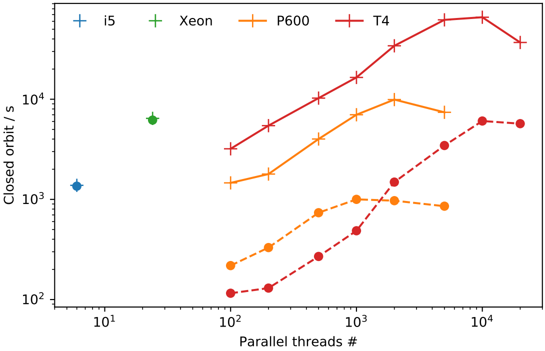
How fast?



	Base clock	Cores
Intel i5-8400	2.8 GHz	6
Intel Xeon Gold 6136	3.0 GHz	24
Nvidia Quadro P600	1329 MHz	384
Nvidia Tesla T4	585 MHz	2560

- Tracking is repeated for different number of particles on **4 different hardware configurations.**
- UFO can run also on CPU.
- The test is repeated using **32 (solid lines) and 64 (dashed lines)** bit variables.
- **Performance improvement for 32 bit variables is remarkable for the GPUs, no difference is observed for CPUs.**

UFO is not only dynamic aperture: closed orbit



	Base clock	Cores
Intel i5-8400	2.8 GHz	6
Intel Xeon Gold 6136	3.0 GHz	24
Nvidia Quadro P600	1329 MHz	384
Nvidia Tesla T4	585 MHz	2560

- Closed orbit is that orbit that repeat itself after one turn.
- The tracking code is encapsulated in a simple minimization routine to find the closed orbit.
- This could be used for **fast orbit response matrix computation.**

Time dependent elements

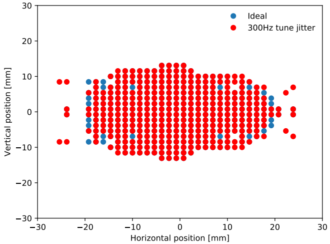
- The OpenCL intermediate lattice representation opens for some interesting possibility
- When the OpenCL representation of the pass method is generated, the **value of parameter is expanded as a text string** and embedded in the OpenCL source code
- UFO does not care if the value of a parameter is a number or a text string
- We can take advantage of this to **"inject" an expression in place of a numerical value!**

For example, the strength of a quadrupole is usually defined as:

alba.QH1.k = 1.57

If we want to include a time dependent sinusoidal ripple on top of it:

alba.QH1.k = "(1.571 + 1.0e-3 * sin((float)turn * 2.68e-4))"



This feature allows to **simulate** for example: **noise, kickers, stripline, rf fields...**

What's next?

- Applying some tricks and proper programming strategies it is possible to **speed up tracking substantially**
- Some of the used **tricks** work only for electron rings and **short term tracking**, for example the 32 bit approximation is very dangerous for long term tracking without damping!!!
- Nevertheless, with a clear idea of the aforementioned limits it is possible to carry out complex optics optimization using only limited resources.
- UFO implements **4D and 5D tracking, 6D** is also supported but there are **no pass methods for RF or radiation yet**
- **Higher order pass methods** are on the "todo" list
- UFO is my every day tool for non-linear optics optimization and allows me to compute $\sim 10^6$ optics per day on a $\sim$2500$ GPU
- **UFO is not friendly, has an unusual structure and requires some effort to make proper use of it**
- **UFO is free software: https://github.com/mcarla/ufo**

UFO is an experiment

I learn many things writing this code and probably made some questionable design decision

...what would you change?

Lattice/Beam:

Lattice*(path=None)*

Beam*(energy=3e9, mass=electron_mass, bunch_charge=1e-9, beam_current=0.25, ex=1e-9, ey=1e-9, bunch_length=6e-3, energy_spread=1e-3)*

Flags:

LINEAR	Replace every non-linear element with a drift (useful for optics computation)
FIVED	Fixed energy simulation (5D)
EXACT	Use exact Hamiltonian for tracking through drifts
KICK	Replace every thick-element with a thin-element/drift approximation using the 'Tea pot' expansion
DOUBLE_PRECISION	Use double precision (64 bit) instead of single precision (32 bit)
ACHROMATIC	Suppress the chromatic aberration of quadrupoles (useful to compute dispersion)

Physics/Simulation commands:

Track*(line, flags=0, turns=1000, particles=1000, parameters=[], where=[], dp=0., context=None, options=None)*

StableAperture*(line, flags=0, turns=1000, particles=1000, parameters=[], dp=0., context=None, options=None)*

Optics*(line, where=[], periodic=True, parameters=[], flags=LINEAR | ACHROMATIC, context=None, options=None)*

Elements: (Parameters follow the definition from MAD-X)

	label	*slices*	*length*	*angle*	*k1*	*e1*	*e2*	*dx*	*dy*	*knl*	*ksl*	*k2*	*k2s*	*k3*	*k3s*
Marker	×														
Drift	×		×								×				
Multipole	×		×					×	×	×	×				
Quadrupole	×	×	×		×			×	×						
Sbend	×	×	×	×	×	×	×	×	×						
Rbend	×	×	×	×	×	×	×	×	×						
Sextupole	×	×	×					×	×			×	×		
Octupole	×	×	×					×	×					×	×