Quasi-real-time adaptive optics simulations on GPUs

YOGA

YOrick with Gpu Acceleration

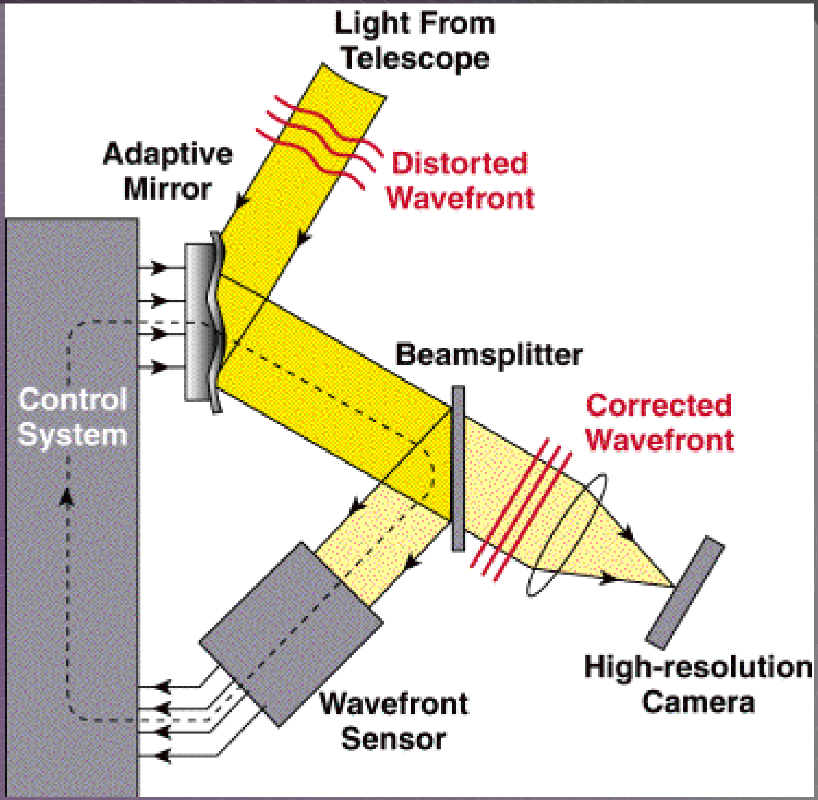
Damien Gratadour

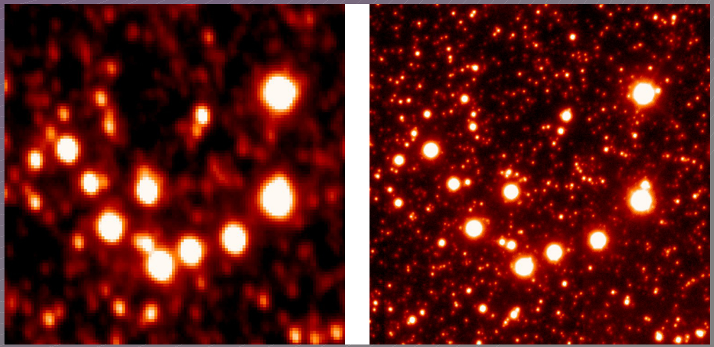
Outline

- **Adaptive Optics simulations**
  - Concept and model
  - The E-ELT scale and the need for massive parallelism
- **YoGA_AO software platform**
  - YoGA : Yorick with GPU acceleration
  - AO extension : data structures and algorithms
- **Features & performance**
- **Future work**
- **Live demo !** (if time allows)

Adaptive optics systems

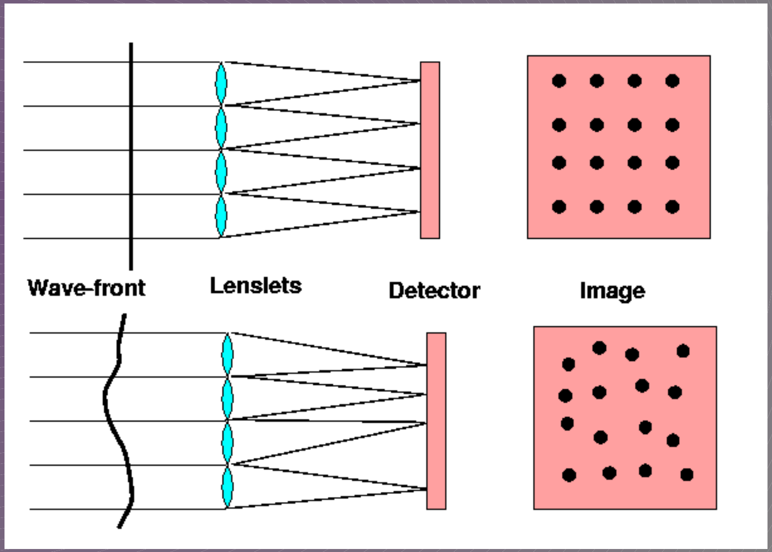
- **Compensate for atmospheric turbulence in real-time**
  - Turbulence measurement using wavefront sensors
    - Several concepts (Shack-Hartmann, curvature, etc ..)
  - Wavefront reconstruction using a real-time computer (analyze measurements and compute correction)
  - Turbulence compensation using a deformable mirror
- **Advanced AO concepts:**
  - Laser Guide stars (LGS) : increase sky coverage
  - Multiple guide stars / deformable mirrors (MCAO, MOAO, GLAO)
  - Very high contrast: XAO





AO systems simulations

- **Whole system and its environment**
  - Atmosphere : several layers of turbulence. Can be preloaded or computed on-the-fly on « rolling » phase screens.
  - Telescope : trivial for monolithic telescope
  - AO system : intensive computations, Monte-Carlo simulations
  - Image formation : large FFTs
- **Include several levels of parallelism**
  - Some computations are intrinsically parallel (matrix multiplies, FFTs, ray-tracing through turbulence)
  - Shack-Hartmann WFS : multiple sub-apertures = « low-level » parallelism
  - Evolved AO concepts : multiple WFS, multiple DMs = « high-level » parallelism (cluster of GPUs)



E-ELT scale

- **Fast AO simulation for 8m telescope**
  - Existing tools: YAO by François Rigaut : http://frigaut.github.com/yao/index.html
  - 60 iteration/s , i.e. 10x slower than real-time control (500Hz)
  - Dominated by wavefront sensing simulation
- **E-ELT : 40m telescope**
  - Need to simulate very large phase screens (2k x 2k). Unrealistic to preload 20k x 20k screens
  - 20 times more subapertures (5k) with sub-images 20x20 to 64x64
  - 20 times more DM actuators (5k)
  - Larger phase screens => larger FFTs to compute final images (4k x 4k)
- **Evolved AO concepts**
  - LGS AO : larger sub-images for WFS (up to 128x128)
  - Multiple DMs and WFSs (ATLAS : 6 LGS WFS, EAGLE : 9 WFS)
  - Very large control matrix (up to 30k x 30k)
- **Need a parallel platform to get realistic execution times (at least few tens of iterations/s)**

Parallel platform

- **Why GPUs ?**
  - Emergence of GPGPU (General Purpose Graphics Processing Units)
  - Provides stream processing capabilities over a large number of processors (NVIDIA : 512)
  - 2 solutions : NVIDIA + CUDA or ATI + Open-CL
  - Cheap solution to build a massively parallel cluster
- **Open-CL**
  - Open standard for parallel architectures
  - Not yet a standard (several distribution and compilers)
  - Few unified libraries available
  - Portability issues : intrinsic hardware properties lead to profound choices in software design (ATI : vector processors, NVIDIA / Intel : scalar processors)
- **NVIDIA + CUDA**
  - Rich development environment + optimized hardware
  - High-level maths library available free of charge
- Tesla series : few k€ versus GeForce series : few 100€ but no ECC, shorter lifetime, larger form factor, larger power consumption

Software environment

- **Why Yorick ?**
  - Complex systems simulations benefit from the use of an interpreted language (comprehensive interface to design / use the code)
  - Yorick is an interpreted programming language for scientific simulations or computations
  - Written in ANSI-C and runs on most OS
  - Compact syntax (C-like) + array operators + extensive graphics possibilities
- **Easily expandable**
  - Dynamic linking of C libraries
  - Spawned process and stdin/out interaction (ex : yorick-python, a.k.a pyk)
- **Active community**
  - Developped by Dave Munro (@ Lawrence Livermore)
  - Éric Thiébaut & François Rigaut main contributors, many more ...
  - Many plugins / extensions available (yeti, yao, spydr, etc..)

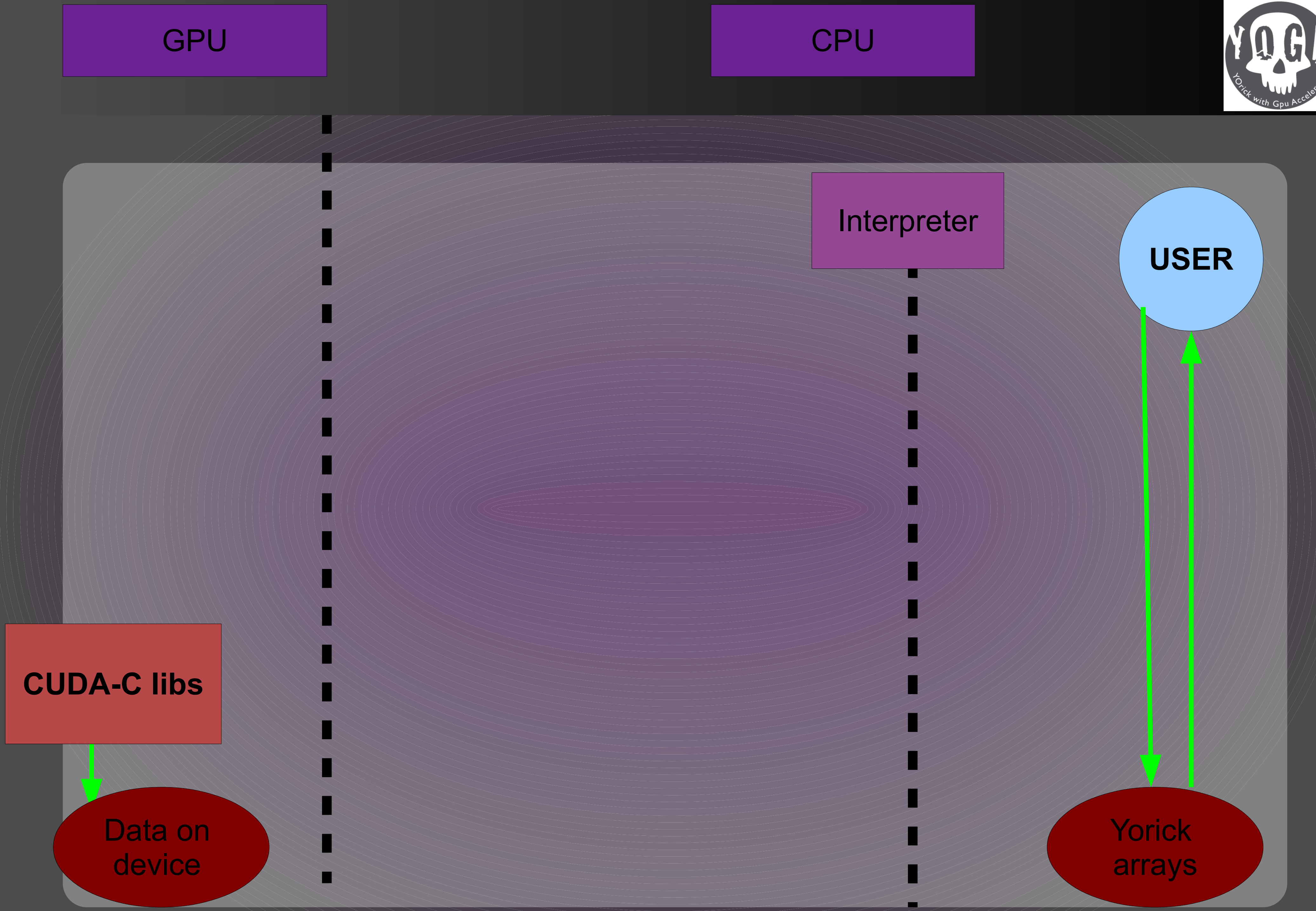
- **Open-source, BSD licence**
  - Available on github : http://github.com/yorick/yorick.github.com/wiki

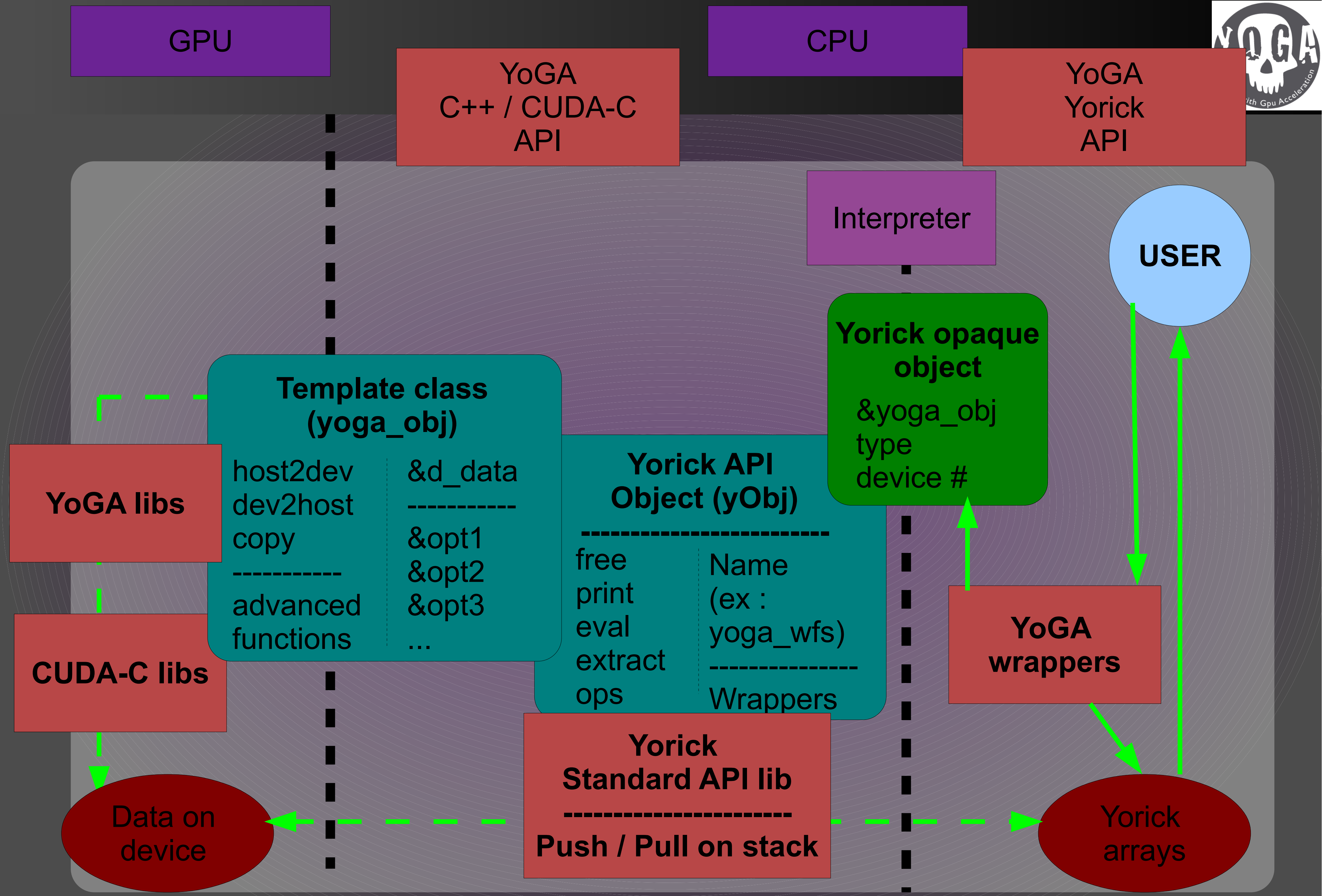
YoGA library : original binding to CUDA

- **Work on the GPU through Yorick**
  - Manipulate arrays on the GPU
  - Launch intensive computations on these objects through an interpreted environment
  - Writting and debugging high-level GPU applications made easy
  - Minimize impact of memcopy between host and device
- **Dynamic linking of CUDA-C libraries**
  - Wrappers to optimized CUDA libraries
  - Yorick object that points to an address on the GPU memory
- **Two-sided implementation**
  - C++ API
  - Yorick API
- **Available on github**
  - https://github.com/yorick-yoga/yorick-yoga/wiki

YoGA_AO

- **Adaptive Optics extension for YoGA**
  - Uses the yoga_object class for basic features
  - Custom classes for atmosphere, optics, WFS, guide sources, etc
  - Easy access to all parameters from within a Yorick session (useful for debug / diagnosis + displays)
- **Includes scripting capabilities + GUI**
  - Optimized template scripts for batch mode
  - GUI using yorick-python binding + GTK
- **Main AO features**
  - Multiple layers turbulence generation
  - Shack-Hartmann wavefront sensor (NGS + LGS)
  - Wavefront slopes computations using various algorithms
- **Available on github** :
  - https://github.com/dgratadour/yoga_ao/wiki

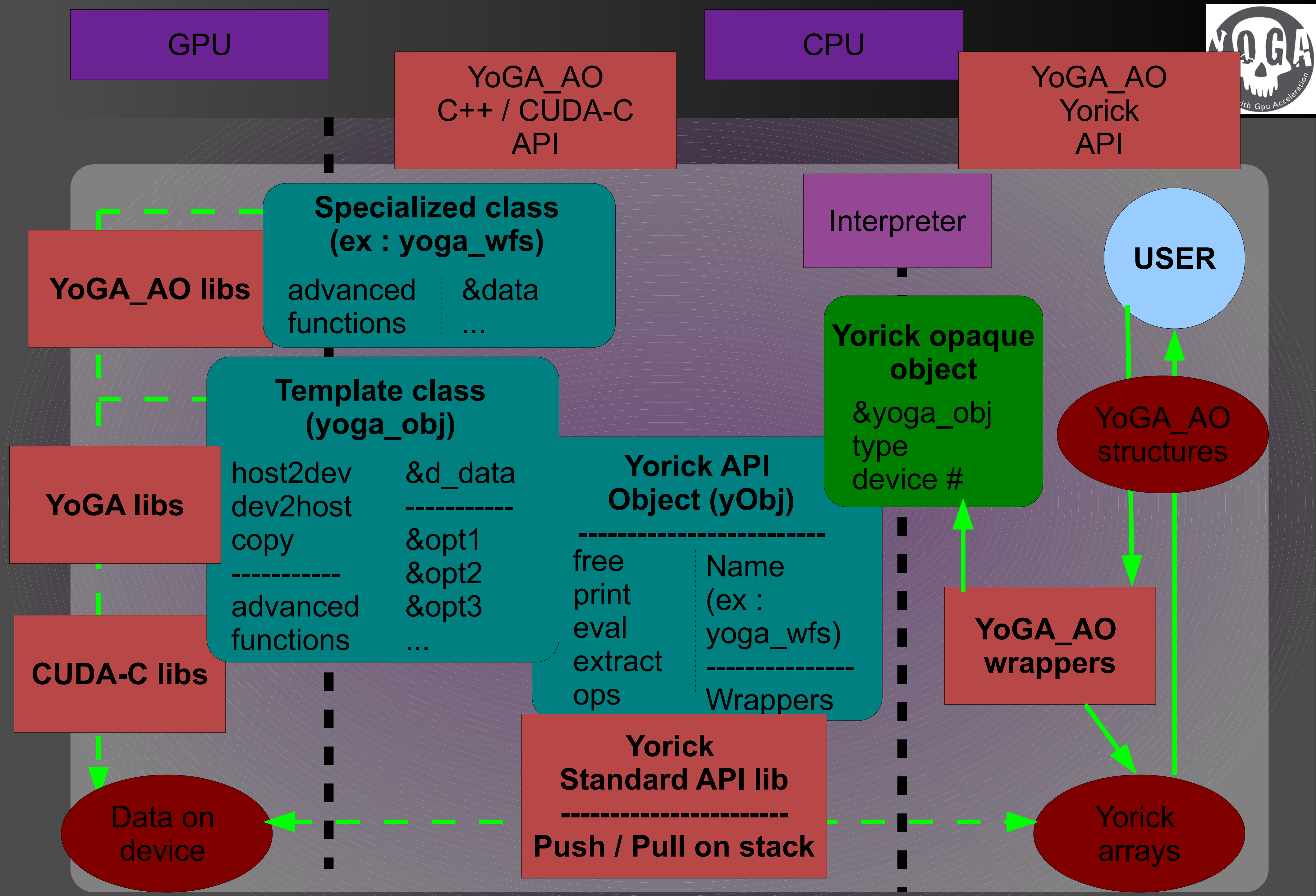

GPU
CPU
Interpreter
USER
CUDA-C libs
Data on device
Yorick arrays


GPU
CPU
YoGA
C++ / CUDA-C
API
YoGA
Yorick
API
Interpreter
USER
Yorick opaque object
&yoga_obj
type
device #
Template class (yoga_obj)
host2dev
dev2host
copy
----
advanced
functions
&d_data

&opt1
&opt2
&opt3
...
Yorick API Object (yObj)
----
free
print
eval
extract
ops
Name
(ex :
yoga_wfs)

Wrappers
YoGA libs
CUDA-C libs
YoGA wrappers
Yorick
Standard API lib

Push / Pull on stack
Data on device
Yorick arrays


GPU
CPU
YoGA_AO
C++ / CUDA-C
API
YoGA_AO
Yorick
API
Interpreter
USER
YoGA_AO libs
Specialized class
(ex : yoga_wfs)
advanced
functions
&data
...
Yorick opaque
object
&yoga_obj
type
device #
Template class
(yoga_obj)
host2dev
dev2host
copy
----
advanced
functions
&d_data

&opt1
&opt2
&opt3
...
YoGA libs
Yorick API
Object (yObj)
----
free
print
eval
extract
ops
Name
(ex :
yoga_wfs)

Wrappers
YoGA_AO
structures
YoGA_AO
wrappers
CUDA-C libs
Yorick
Standard API lib

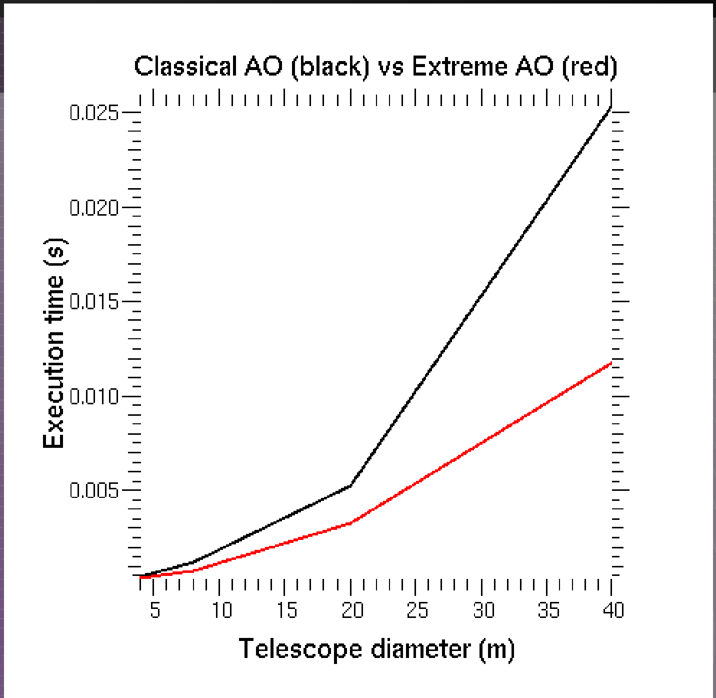
Push / Pull on stack
Data on
device
Yorick
arrays

YoGA_AO features

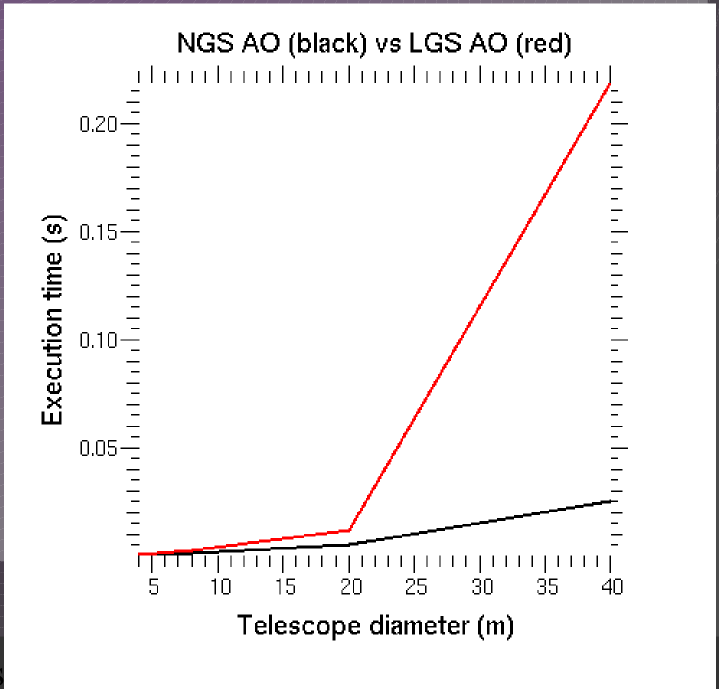
- **General features**
  - On-the-fly atmospheric turbulence generation on multiple layers at various altitude with various strength, speed, direction
  - Optimized ray-tracing in a given direction for image computation
  - Multiple targets
  - Optimized Shack-Hartmann wavefront sensor model
  - Laser guide star model
  - Various centroiding algorithms (COG, thresholded, weighted, correlation)
  - Multiple WFS in multiple directions (LGS or NGS)
  - Comprehensive interface through Yorick
  - **… more to come ! (deformable mirror model, various command algorithms, etc ..)**

YoGA_AO performance

- **Various system dimensioning**
  - Classical AO (SCAO) : Subaps Ø ~ 50cm => # subaps ~ 2 x telescope diam.
  - Extreme AO (XAO) : Subaps Ø ~ 20cm => # subaps ~ 5 x telescope diam.
  - Better performance for XAO.
  - XAO : smaller subaps hence less phase points / subaps even if way more subaps => our code takes full advantage of massive parallelism

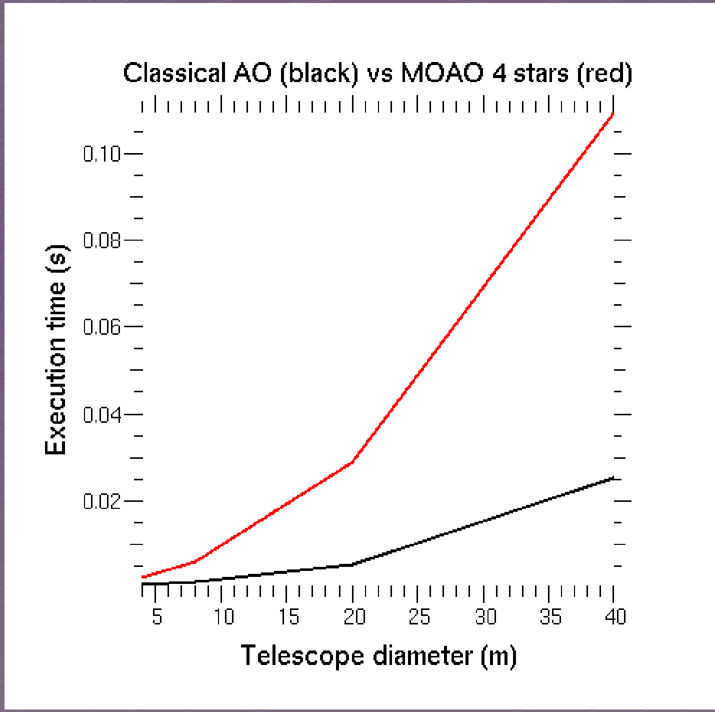


- **Case of Laser guide star**
  - LGS AO : subaps FoV is larger and increases with elongation hence telescope diameter
  - Unable to fit the whole computation for one subap in shared memory : no significant gain as compared to SCAO



YoGA_AO performance

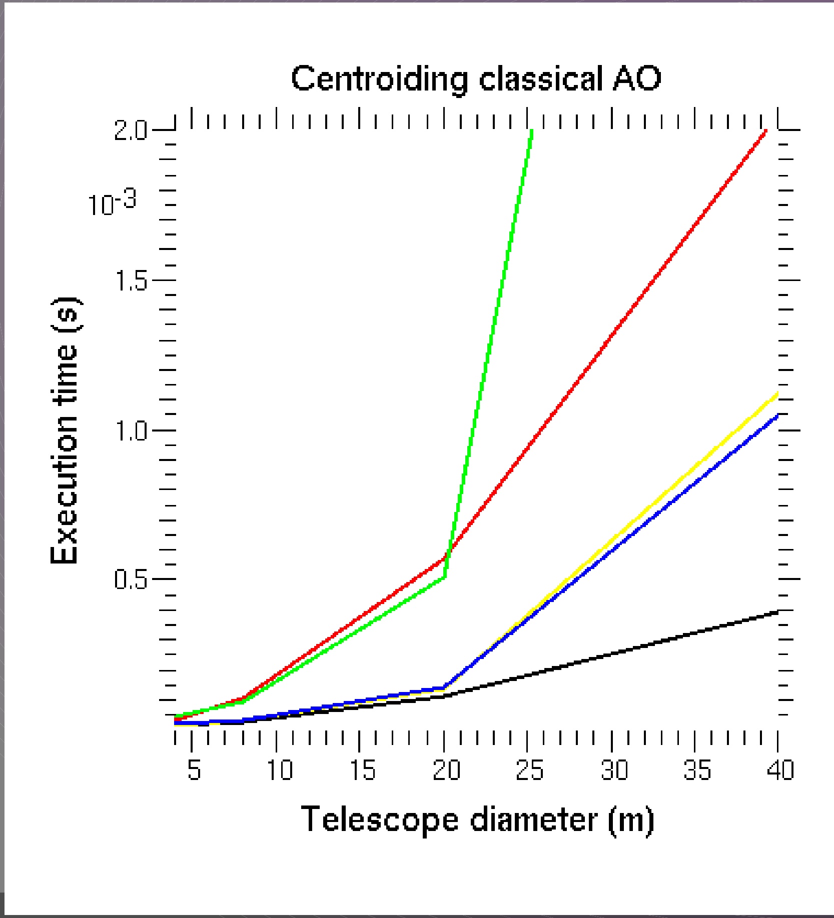
- **Case of multiple WFS**
  - For now, sequential for the multiple WFS so no gain in performance as compared to SCAO

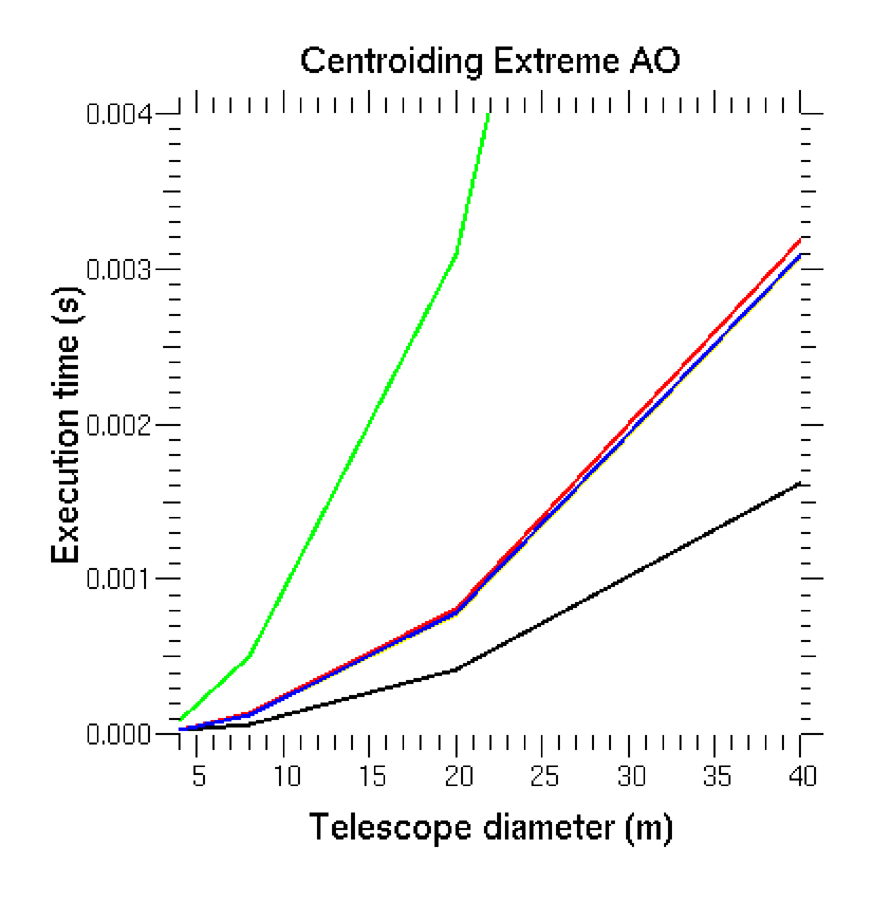


  - Need to work on a proper multi-GPU version to parallelize wavefront sensing across multiple GPUs (bandwidth issue ?)

YoGA_AO performance

- **Centroiding**
  - As fast for XAO as SCAO : again we take full advantage of parallelism for centroid computation





Conclusion & future works

- **Optimized AO simulation with a comprehensive interface running on GPUs**
  - Few 1000 iterations/s are reached for XAO systems on a 8m telescope : faster than « real-time » controlers for AO
  - Few 100 iterations/s are reached for SCAO & XAO systems at the E-ELT scale : realistic enough to start working
  - Code takes full advantage of GPU architecture for core computations
  - User-friendly interface to test various configurations
- **Missing some components**
  - Deformable optics (trivial using existing libs)
  - Control scheme : something that needs to be thought and optimized
- **Future works**
  - Need to properly integrate a multi-GPU approach for evolved AO concepts (multi-WFS systems)
  - Define an interface to the outside world so that the code could be used to characterise real-time controlers for AO

Demo time !