Utilisation des GPUs avec YoGA
Introduction

Arnaud Sevin / Damien Gratadour / Julien Brulé

Plan

- **Le *Plug-in* YoGA**
  - Un *binding* original : YoGA - *Yorick with GPU accélération*
  - Tendre vers l'universel : extensions dans d'autres langages
- **Fonctionnalités & performances**
- **Développements futurs**
- **Plate-forme matérielle à Meudon**

Environnement logiciel

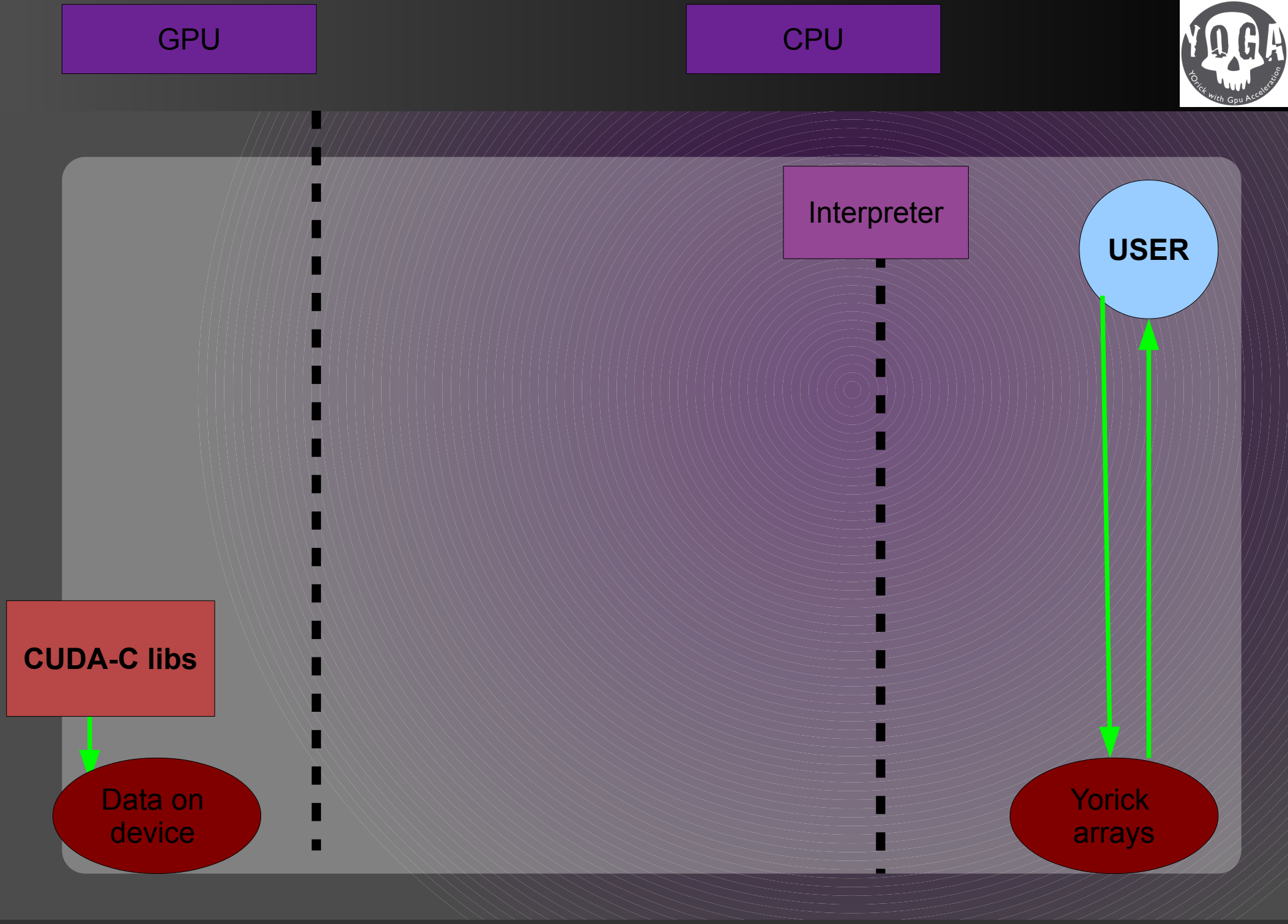
- **Pourquoi un langage interprété ?**
  - Les simulations de systèmes complexes bénéficient grandement de l'utilisation d'un langage interprété (interface simplifiée pour le design et l'utilisation du code)
- **Pourquoi Yorick ?**
  - Yorick est un langage interprété pour les calculs scientifiques et les simulations
  - Écrit en ANSI-C et tourne sur la plupart des OS
  - Syntaxe compacte (C-like) + opérations sur les tableaux + capacités graphiques étendue
  - Aussi simple et puissant qu'IDL ou Matlab mais libre de droit !
- **Facilement extensible**
  - *Dynamic linking* librairies C
  - Interaction stdin/out et *process* "*spawnés*" (ex : yorick-python, a.k.a pyk)
- **Communauté active**
  - Développé par Dave Munro (@ Lawrence Livermore)
  - Principaux contributeurs : Éric Thiébaut & François Rigaut ...
  - Beaucoup de *plugins* / extensions disponibles (yeti, yao, spydr, etc..)
- **Libre, ouvert, liscence BSD**
  - Sur github : http://github.com/yorick/yorick.github.com/wiki

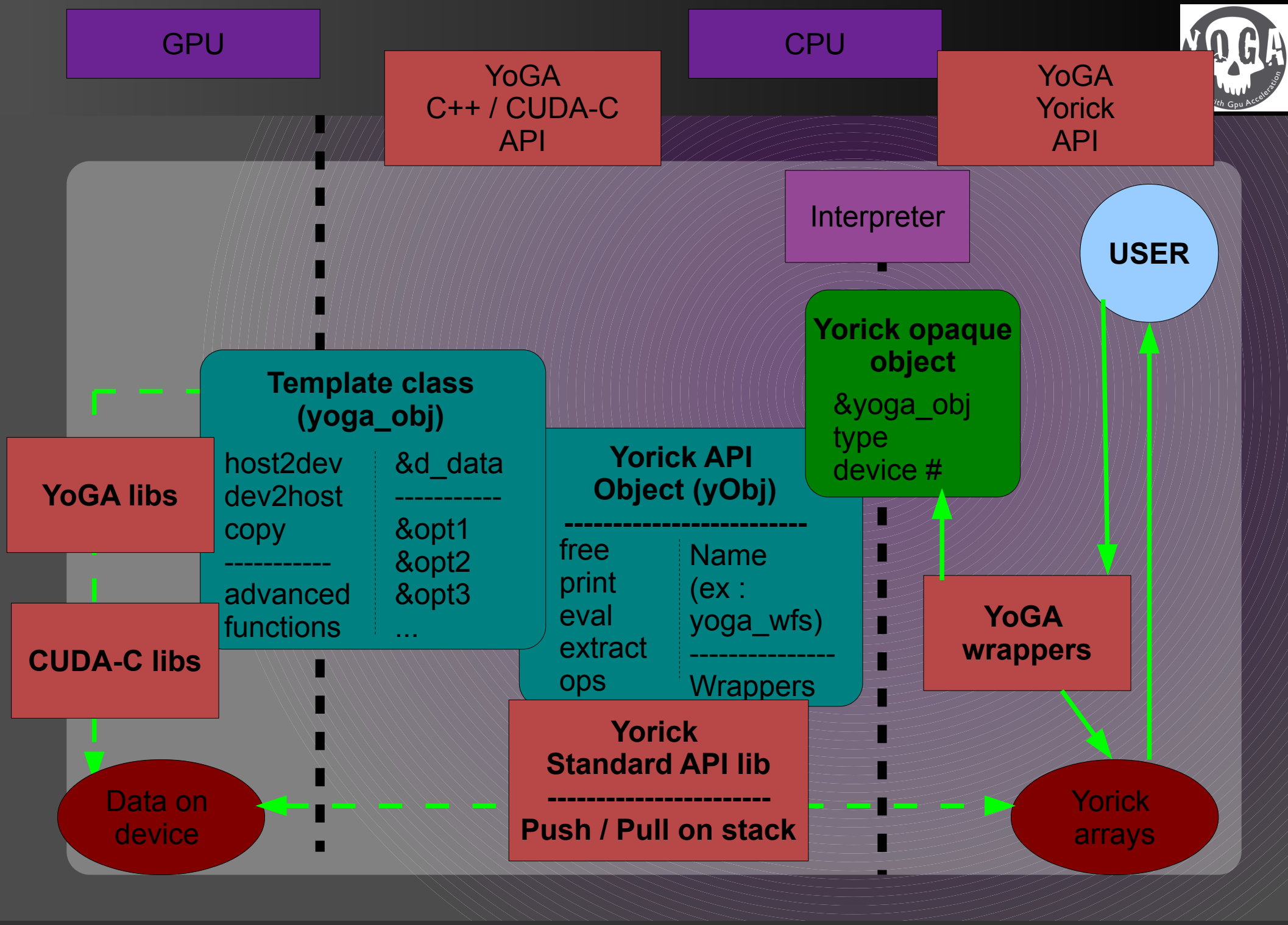
Librairie YoGA : *binding* original à CUDA

- **Travailler sur le GPU avec Yorick**
  - Manipuler des tableaux sur le GPU
  - Lancer des calculs intensifs sur ces objets à travers un environnement interprété
  - Écrire et debugger facilement des applications haut-niveau sur le GPU
  - Minimiser l'impact de la copie mémoire entre l'hôte et le GPU
- ***Dynamic linking* de librairies CUDA-C**
  - *Wrappers* vers des librairies CUDA optimisées
  - Un objet Yorick qui pointe vers une adresse sur la mémoire du GPU
- **Librairie à 2 niveaux**
  - API C++
  - API Yorick
- **Disponible sur github**
  - https://github.com/yorick-yoga/yorick-yoga/wiki

Extension vers d'autres langages

- **La même idée peut être reproduite**
  - Créer une librairie partagée
  - La pré-compiler => cross-platform
  - Le langage interprété est toujours utile car tout coder sur le GPU est lourd et le plus souvent pas nécessaire
- **Développements actuels vers Python**
  - Même idée d'un *binding* Cuda – Python
  - Similaire à pyCuda mais plus simple à étendre / optimiser
  - Garder la main => des nouvelles fonctionnalités peuvent être incluses sans dépendre d'autres équipes
- **Librairie à 2 étages**
  - API C++ (YoGA-C++)
  - API Python
- **Bientôt disponible github**


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

YoGA object

- **Template class**
- **Principale composante de l'API YoGA C++**
- **Inclus des *wrappers* vers les principales librairies CUDA : CUFFT, CUBLAS, CURAND, CUDPP, CULA**
- **Intégration simplifiée de ces fonctionnalités dans du code haut niveau**
- **Gestion de la mémoire transparente**

```
template<class T_data> class yoga_obj {

 protected:
  T_data *d_data;///< Input data
  T_data *o_data;///< optional data (used for scan / reduction)
  long *dims_data;///< dimensions of the array
  int nb_elem;///< number of elements in the array
  int device; ///< device where the yoga_obj is allocate
  yoga_context *current_context;

  curandGenerator_t gen;
  curandState *d_states;

  int nThreads;
  int nBlocks;

#ifdef _USE_CUDPP
  CUDPPHandle   mScanPlan;        // CUDPP plan handle for prefix sum
#endif
  bool keysOnly; //< optional flag (used for sort)
  unsigned int *values;///< optional data (used for sort)
  size_t *d_numValid;///< used for compact

  cufftHandle plan;///< FFT plan
  cufftType tPlan;///< FFT plan type

  yoga_streams *streams;

  void init(yoga_context *current_context, long *dims_data, T_data *data, bool fromHost, int nb_streams);

 public:
  yoga_obj(yoga_obj<T_data> *obj);
  yoga_obj(yoga_context *current_context, long *dims_data);
  yoga_obj(yoga_context *current_context, yoga_obj<T_data> *obj);
  yoga_obj(yoga_context *current_context, long *dims_data, T_data *data);
  yoga_obj(yoga_context *current_context, long *dims_data, int nb_streams);
  yoga_obj(yoga_context *current_context, yoga_obj<T_data> *obj, int nb_streams);
  yoga_obj(yoga_context *current_context, long *dims_data, T_data *data, int nb_streams);
  ~yoga_obj();

  int get_nbStreams();
  int add_stream();
  int add_stream(int nb);
  int del_stream();
  int del_stream(int nb);
  cudaStream_t get_cudaStream_t(int stream);
  int wait_stream(int stream);
  int wait_all_streams();

  /**< General Utilities */
  T_data* getData() { return d_data; }
  T_data* getOData() { return o_data; }
  long * getDims() { return dims_data; }
  long getDims(int i) { return dims_data[i]; }
  int getNbElem() { return nb_elem; }
  yoga_context* getContext() { return current_context; }

  bool is_rng_init() { return (gen!=NULL); }


  /**< Memory transfers both ways */
  int host2device(T_data *data);
  int device2host(T_data *data);
```

YoGA object

- **Intégration du multi-GPU**
- **yoga_device & yoga_context**
- **Prise en compte des propriétés du GPU (optimisation des *kernels*)**
- **Prise en charge des calculs multi-GPU**
- **Prise en charge du P2P & des fonctionnalités GPUdirect**

```
class yoga_device {
 protected:
  int            id;
  cudaDeviceProp properties;
  float          compute_perf;
  float          sm_per_multiproc;
  bool                  p2p_activate;

 public:
  yoga_device(int devid);
  yoga_device(const yoga_device& device);
  ~yoga_device();

  int get_id(){ return id; }
  cudaDeviceProp get_properties(){ return properties; }
  float get_compute_perf(){ return compute_perf; }
  float get_sm_per_multiproc(){ return sm_per_multiproc; }
  bool isGPUCapableP2P()
  {
      return (bool)(properties.major >= 2);
  }

  bool isP2P_active()
  {
      return p2p_activate;
  }

};

class yoga_context {
 protected:
  int                    ndevice;
  vector<yoga_device *>  devices;
  int                                  activeDevice;
  int**                          can_access_peer;

 public:
  yoga_context();
  yoga_context(const yoga_context& cntxt);
  ~yoga_context();

  int get_ndevice() {return ndevice;}
  yoga_device *get_device(int dev) {return devices[dev];}
  int get_activeDevice() {return activeDevice;}
  string get_activeDeviceStr();
  int set_activeDevice(int newDevice, int silent=1);
  int set_activeDeviceForCpy(int newDevice, int silent=1);

  int get_maxGflopsDeviceId();

};
```

YoGA object

- **Interlaçage rapide copie / calcul : *stream processing***
- **yoga_stream & yoga_host_obj**

```
class yoga_streams {
protected:
        vector<cudaStream_t> streams;
        vector<cudaEvent_t> events;
        int eventflags;

public:
        yoga_streams();
        yoga_streams(unsigned int nbStreams);
        //yoga_stream(const yoga_stream& src_yoga_stream);
        ~yoga_streams();

        cudaStream_t get_stream(int stream);
        cudaEvent_t get_event(int stream);
        cudaStream_t operator[](int idx) {
                return get_stream(idx);
        };

        int get_nbStreams();
        int add_stream();
        int add_stream(int nb);
        int del_stream();
        int del_stream(int nb);
        int del_all_streams();
        int wait_event(int stream);
        int wait_stream(int stream);
        int wait_all_streams();

};
```

```
template<class T_data>
class yoga_host_obj {

 protected:
  T_data *h_data;///< Input data
  T_data *data_UA;///< unpadded input dara for generic pinned mem
  long *dims_data;///< dimensions of the array
  int nb_elem;///< number of elments in the array
  MemAlloc mallocType;///< type of host alloc
  yoga_streams *streams;

  void init(long *dims_data, T_data *data, MemAlloc mallocType, int nb_streams);

 public:
  yoga_host_obj(long *dims_data);
  yoga_host_obj(long *dims_data, MemAlloc mallocType);
  yoga_host_obj(yoga_host_obj<T_data> *obj);
  yoga_host_obj(yoga_host_obj<T_data> *obj, MemAlloc mallocType);
  yoga_host_obj(long *dims_data, T_data *data);
  yoga_host_obj(long *dims_data, T_data *data, MemAlloc mallocType);
  yoga_host_obj(long *dims_data, int nb_streams);
  yoga_host_obj(long *dims_data, MemAlloc mallocType, int nb_streams);
  yoga_host_obj(yoga_host_obj<T_data> *obj, int nb_streams);
  yoga_host_obj(yoga_host_obj<T_data> *obj, MemAlloc mallocType, int nb_streams);
  yoga_host_obj(long *dims_data, T_data *data, int nb_streams);
  yoga_host_obj(long *dims_data, T_data *data, MemAlloc mallocType, int nb_streams);
  ~yoga_host_obj();

  void get_devpntr(void **pntr_dev);

  int get_nbStreams();
  int add_stream();
  int add_stream(int nb);
  int del_stream();
  int del_stream(int nb);
  cudaStream_t get_cudaStream_t(int stream);
  int wait_stream(int stream);
  int wait_all_streams();

  int cpy_obj(yoga_obj<T_data>* yObj, cudaMemcpyKind flag);
  int cpy_obj(yoga_obj<T_data>* yObj, cudaMemcpyKind flag, unsigned int stream);

  /**< General Utilities */
  T_data* getData() { return h_data; }
  long * getDims() { return dims_data; }
  long getDims(int i) { return dims_data[i]; }
  int getNbElem() { return nb_elem; }

  /**< Memory transfer */
  int fill_from(T_data *data);
  int fill_into(T_data *data);

  string getMetAlloc (){
        switch(mallocType){
        case MA_MALLOC: return "MA_MALLOC";
        case MA_PAGELOCK: return "MA_PAGELOCK";
        case MA_ZEROCPY: return "MA_ZEROCPY";
        case MA_PORTABLE: return "MA_PORTABLE";
        case MA_WRICOMB: return "MA_WRICOMB";
        case MA_GENEPIN: return "MA_GENEPIN";
        default: return "MA_UNKNOWN";
        }
  }
```

Plan

- **Le *Plug-in* YoGA**
  - Un *binding* original : YoGA - *Yorick with GPU accélération*
  - Tendre vers l'universel : extensions dans d'autres langages
- **Fonctionnalités & performances**
- **Développements futurs**
- **Plate-forme matérielle à Meudon**

Fonctionnalités de YoGA

- **Fonctionnalités générales**
  - **Yoga_context** : définit l'environnement de travail et permet d'optimiser la répartition des tâches en fonction du hardware utilisé
  - **Yoga_obj** : représentation d'un objet dans la mémoire du GPU. Objet persistent
  - **Yoga_host_obj** : représentation d'un objet dans la mémoire de l'hôte (type de mémoire optimisé pour un transfert rapide : mémoire "*pinnée*", "*zero-copy*", etc ..)
  - **Yoga_streams** : gestion de l'interlaçage copie / calcul (support du multi-GPU)

Fonctionnalités de YoGA

- **Fonctionnalités avancées :**
  - **FFT** : utilisation de la librairie cuFFT d'nVIDIA
    - « jusqu'à 10 fois plus rapide »
    - « utilise une centaine de cœurs »
  - **BLAS** : utilisation de librairie cuBLAS d'nVIDIA
    - Basic Linear Algebra Subprograms
    - « de 6 à 17 fois plus rapide que la dernière lib MKL BLAS »
  - **LAPACK dense** : utilisation de CULA dense
    - Essentiellement pour la SVD

Fonctionnalités de YoGA

- **Fonctionnalités avancées optionnelles :**
  - **CUDPP** : CUDA Data Parallel Primitives
    - parallèle prefix-sum (”scan”)
    - parallèle sort
    - parallèle reduction
  - **MAGMA** : Matrix Algebra on GPU and Multicore Architectures
    - Librairie d'algèbre linéaire pour des architecture hybride
    - Testé pour sa SVD mais moins performante que celle de CULA

Performance de YoGA

- **FFT : yoga_fft vs yorick FFT**

taille	32	64	128	256	512	1024	2048	4096	8192
avec cpy	0.7ms (x0.3)	0.9ms (x0.5)	0.9ms (x0.6)	3.4ms (x0.6)	12.7ms (x0.8)	69ms (x0.9)	285ms (x1.1)	1.2s (x1.3)	4.8s (x1.7)
sans cpy	0.4ms (x0.5)	0.3ms (x1.4)	0.1ms (x3.7)	0.1ms (x15.1)	0.4ms (x32.5)	0.5ms (x134)	0.5ms (x761)	0.7ms (x2017)	1.8ms (x4452)
yorick	0.19ms	0.4ms	0.5ms	2.1ms	12.7ms	65ms	313ms	1.54s	8.24s

- **MultiFFT 32x32 : yoga_fft vs yorick FFT**

taille	32	64	128	256	512	1024	2048	4096	8192
avec cpy	5.8ms (x0.4)	5.2ms (x0.6)	5.8ms (x1.0)	7.0ms (x1.7)	9.1ms (x2.6)	13.5ms (x3.6)	20.1ms (x4.7)	40ms (x4.8)	76.6ms (x5.0)
sans cpy	0.2ms (x8.8)	0.3ms (x11.6)	0.3ms (x23.5)	0.3ms (x46.6)	0.3ms (x93.7)	0.3ms (x185)	0.3ms (x370)	0.3ms (x730)	0.6ms (x641)
yorick	2.2ms	2.9ms	5.9ms	11.8ms	24.1ms	48.2ms	96.6ms	194ms	386ms

Performance de YoGA

- **Matrice – Vecteur multiplication CUBLAS2 ($V_n * M_{n,n}$)**

taille	32	64	128	256	512	1024	2048	4096	8192
avec cpy	0.4ms (x0.1)	0.3ms (x0.1)	0.3ms (x0.3)	0.4ms (x0.8)	0.7ms (x0.6)	0.8ms (x2.6)	0.9ms (x11.8)	1.6ms (x28)	3.7ms (x48)
sans cpy	76µs (x0.3)	27µs (x1.3)	27µs (x3.3)	27µs (x11.1)	12µs (x33.1)	20µs (x120)	20µs (x580)	20µs (x2182)	20µs (x9251)
yorick	25µs	36µs	90µs	301µs	434µs	2.3ms	11ms	44s	176ms

- **Matrice – Matrice multiplication :**

taille	32	64	128	256	512	1024	2048	4096	8192
avec cpy	1.8ms (x0.0)	0.4ms (x1.4)	0.4ms (x8.3)	0.8ms (x12.1)	2.7ms (x30.3)	9ms (x71.2)	45ms (x124)	289ms (x154)	2.0s (x182)
sans cpy	82µs (x1.0)	27µs (x18.5)	36µs (x110)	54µs (x18)	30µs (x2731)	41µs (x1.6e4)	46µs (x1.2e5)	47µs (x9e5)	47µs (x8e6)
yorick	84µs	535µs	3.9ms	10.2ms	81ms	641ms	5.6s	44s	367s

Performance de YoGA

- **SVD avec CULA :**

taille	32	64	128	256	512	1024	2048	4096	8096
Device	6ms (x0.1)	7.8ms (x0.2)	21.9ms (x0.9)	67ms (x2.6)	269ms (x4.6)	1.0s (x10.9)	4s (x48.3)	17.2s (x112)	168.4s
Host	2.5ms (x0.2)	6.7ms (x0.2)	23.9ms (x0.9)	68ms (x2.6)	274ms (x4.6)	1.0s (x10.8)	4s (x48.3)	17.6s (x110)	169.3s
yorick	563µs	1.8ms	20ms	176ms	1.25s	11.16s	195s	1935s	-

Plan

- **Le *plug-in* YoGA**
  - Un *binding* original : YoGA - *Yorick with GPU accélération*
  - Tendre vers l'universel : extensions dans d'autres langages
- **Fonctionnalités & performances**
- **Développements futurs**
- **Plate-forme matérielle à Meudon**

Développements futurs

- **Développement de l'intégration du multi-GPU :**
  - Stream processing n'est pas utilisé partout,
  - Mise en œuvre de pipelines indépendants consacrés à un GPU spécifique.
- **Développement basé sur les matrices creuses :**
  - Dans notre domaine, nous utilisons souvent des matrices creuses (loi de commande intégrateur, ...).
  - Pour augmenter les performances, nous devons mettre en œuvre cette fonctionnalité à l'intérieur de YOGA.
- **Ajout de méthodes optimisées dans Yoga_context :**
  - L'arrivée des nouveaux GPU Keper, certains paramètres doivent être ajustés en fonction du GPU utilisé : #threads, #blocks, size of the SHM, …
  - Yoga_context a besoin de savoir comment configurer l'environnement pour optimiser les performances.
- **Template de fonction :**
  - Création d'un kernel générique permettant d'utiliser les fonctions de base sans avoir à coder une ligne de code.

Plan

- **Le *plug-in* YoGA**
  - Un *binding* original : YoGA - *Yorick with GPU accélération*
  - Tendre vers l'universel : extensions dans d'autres langages
- **Fonctionnalités & performances**
- **Développements futurs**
- **Plate-forme matérielle à Meudon**

Plateforme disponible à Meudon

plate-forme de production

Yogi: PC industriel avec un processeur Intel Core i7 + 2 GPU NVIDIA: GeForce GTX 480 et Tesla C2070

- Carte mère simple IOH (2 ports PCIe avec une bande passante complète)
- Utilisé pour tester les nouveaux développements et le code de référence

Bientôt : Guru (TBD, mais sans doute à carte mère double IOH + Geforce GTX 690)

- Sera utilisé pour tester des applications temps réel

plate-forme de production

HiPPO: cluster de 4 nœuds avec 2 6-core Xeon 3.3GHz et 2 Tesla M2090 par nœud

- Interconnexion Infiniband
- Cartes mères double IOH => jusqu'à 4 ports PCIe par nœud avec une bande passante complète
- Chaque nœud peut être reconfiguré en fonction des applications (GPUdirect ou Peer-to-host-to-peer)
- Utilisé pour effectuer des simulations à grande échelle + de tests de bande passante dans diverses configurations (en cours)

Conclusions

- **Il y a beaucoup de choses de faites pour faciliter le développement**
  - API Yorick :
    - Pratique pour les tests
    - Optimisation de gros calculs
    - Performance dégradée due au transfert host<->device
  - API C++ :
    - base pour construire des codes plus évolues
    - Permet une gestion simplifiée des spécificités du GPU et surtout une gestion transparente de la mémoire
    - Seule façon d'écrire du code vraiment optimisé (cf yoga_ao).
    - Possibilité de réfléchir à d'autres extensions : traitement d'image.
  - Support assuré :)
- **Le code est libre et disponible sur github**
  - Possibilité d'évolution pour mieux répondre aux besoins
- **Manque de visibilité et donc d'utilisateurs**
  - Développement des moyens de commutations entre les développeurs et les utilisateurs