

Calculer sur GPU avec MATLAB



Eric ANTERRIEU
*Observatoire Midi-Pyrénées
 Centre d'Etudes Spatiales de la BIOSphère
 Equipe Systèmes d'Observation*

CESBIO — UMR5126







Eric.Anterrieu@cesbio.cnes.fr

Calculer sur GPU avec MATLAB

Tirer la pleine puissance de calcul des GPU n'est pas simple et la programmation massivement parallèle est un passage obligatoire. Si les professionnels du HPC n'éprouvent aucune difficulté pour les intégrer dans leurs développements, rien ou peu n'est fait pour attirer de nouveaux venus ni pour séduire ceux qui ne veulent pas consacrer de temps au HPC.

Nous montrerons comment les utilisateurs de MATLAB, pas nécessairement professionnels du HPC, peuvent très facilement intégrer les GPU dans leurs développements sans modifier leurs codes existants. Nous montrerons également comment les professionnels du HPC peuvent tout aussi facilement intégrer certains développements à MATLAB et en faire bénéficier ainsi le plus grand nombre. Dans les deux cas, nous illustrerons sur des exemples le bénéfice attendu et les investissements que cela nécessite de la part des uns et des autres.

Eric.Anterrieu@cesbio.cnes.fr

Calculer sur GPU avec MATLAB

- 1 Introduction
- 2 Identifier et sélectionner un GPU
- 3 Transférer des données
- 4 Exécuter des fonctions MATLAB
- 5 Exécuter des noyaux CUDA
- 6 Conclusion

Eric.Anterrieu@cesbio.cnes.fr

1 Introduction

MATLAB ?

Avec une communauté forte de plus de 2 millions d'utilisateurs répartis dans 180 pays sur plus de 80000 sites dans l'industrie, les organismes gouvernementaux et académiques, MATHWORKS est reconnu comme le principal éditeur de logiciels de calcul scientifique et technique.

www.mathworks.com
www.mathworks.com/matlabcentral

MATLAB est un langage de haut-niveau pour le calcul scientifique et un environnement de programmation pour le développement d'algorithmes, l'analyse des données, leur visualisation et le calcul numérique.

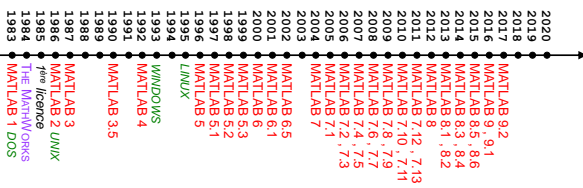
Eric.Anterrieu@cesbio.cnes.fr

1 Introduction

MATLAB ?

MATLAB est l'abréviation de MATrix LABoratory.

MATLAB est né au début des années 80 dans le milieu universitaire (Pr. Cleve Moler & Ing. Jack Little). Depuis 1984 MATLAB est développé et commercialisé par la société MATHWORKS.



Timeline showing MATLAB versions from 1983 to 2020. Key milestones include: 1984 (MATLAB 1 DOS), 1985 (MATLAB 2 UNIX), 1986 (MATLAB 3 UNIX), 1987 (MATLAB 3.5), 1988 (MATLAB 3.5), 1989 (MATLAB 3.5), 1991 (MATLAB 4), 1992 (MATLAB 4), 1993 (MATLAB 4), 1994 (MATLAB 4), 1995 (MATLAB 5), 1996 (MATLAB 5), 1997 (MATLAB 5.1), 1998 (MATLAB 5.2), 1999 (MATLAB 5.3), 2000 (MATLAB 6), 2001 (MATLAB 6.1), 2002 (MATLAB 6.5), 2003 (MATLAB 6.5), 2004 (MATLAB 7), 2005 (MATLAB 7.1), 2006 (MATLAB 7.2), 2007 (MATLAB 7.3), 2008 (MATLAB 7.4), 2009 (MATLAB 7.5), 2010 (MATLAB 7.6), 2011 (MATLAB 7.7), 2012 (MATLAB 7.8), 2013 (MATLAB 8.1), 2014 (MATLAB 8.2), 2015 (MATLAB 8.3), 2016 (MATLAB 8.4), 2017 (MATLAB 8.5), 2018 (MATLAB 9.1), 2019 (MATLAB 9.2), 2020 (MATLAB 9.3).

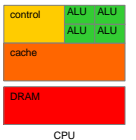
Eric.Anterrieu@cesbio.cnes.fr

1 Introduction

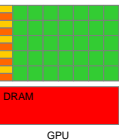
GPU ?

Un processeur graphique, ou GPU (*Graphics Processing Unit*), est un circuit intégré (sur carte graphique ou intégrée sur carte-mère) qui assure les fonctions de calcul liées à l'affichage des trames à l'écran. Un GPU a généralement une structure hautement parallèle qui le rend efficace pour une large palette de tâches graphiques. Si les premiers GPU étaient à fonctions fixes, ils ont évolué pour devenir programmables.

Le parallélisme massif des GPU les rend donc très intéressants comme processeurs de calcul matriciel sur des volumes de données importants (*General Purpose GPU*) mais ils ne sont pas la solution universelle!



CPU



GPU

CPU: ■+> ■ task parallelism
 GPU: ■+<■ data parallelism

Eric.Anterrieu@cesbio.cnes.fr

1 Introduction

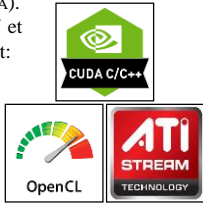
GPU ?

Historiquement il n'existait pas d'interface de programmation applicative, ou API (*Application Programming Interface*), spécifique pour programmer les GPU: les développements se faisaient par détournement des langages de programmation graphiques (GLSL pour OpenGL, HLSL pour DirectX, CG pour NVIDIA). Des API destinées au calcul parallèle massif et indépendantes des API graphiques apparaissent:

- CUDA pour les cartes NVIDIA,
- STREAM pour les cartes ATI/AMD,
- OpenCL langage dérivé du C + API,
- OpenACC directives à la openMP.

Inconvénient / conséquence:

- la programmation parallèle est obligatoire!
- l'accèsion à la pleine puissance de calcul du GPU n'est pas simple...



Calculer sur GPU avec MATLAB Eric.Anterieu@cesbio.cnes.fr

1 Introduction

MATLAB + GPU

Face à l'engouement pour le calcul parallèle et notamment pour les GPU, MATHWORKS a proposé une (pas la) solution à la difficulté de tirer la pleine puissance si on n'est pas un spécialiste de la programmation (massivement) parallèle.

A qui (ne) s'adresse (pas) cet exposé ? Quels sont les prérequis ?

- A ceux qui connaissent (bien/un peu) MATLAB et (pas/peu) les GPU: *comment tirer profit de la présence de GPU dans MATLAB sans passer des heures à comprendre l'architecture des GPU ni beaucoup modifier les fonctions MATLAB.*
- A ceux qui connaissent (bien/un peu) les GPU et (pas/peu) MATLAB: *comment des développements existants pour GPU peuvent être utilisés dans MATLAB sans modifier les noyaux CUDA.*

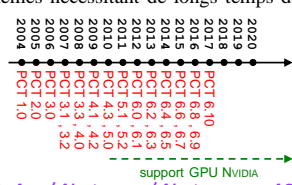
Calculer sur GPU avec MATLAB Eric.Anterieu@cesbio.cnes.fr

1 Introduction

MATLAB + GPU = PCT

En complément de la boîte de base, MATLAB, il est possible d'ajouter des boîtes à outils spécifiques. Ce sont de vastes collections de fonctions qui étendent les capacités de la boîte de base pour répondre à des besoins particuliers.

Parmi les domaines couverts, depuis 2004 la *Parallel Computing Toolbox* (PCT) permet de résoudre les problèmes nécessitant de longs temps de calculs et des volumes de données importants sur des ordinateurs multi-cœurs et multi-processeurs, sur des GPU (depuis 2010b) et aussi sur des *clusters / clouds / grids* d'ordinateurs (nécessite une licence du *Distributed Computing Server*).

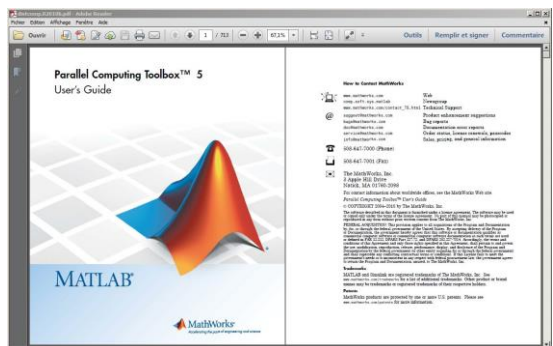


www.mathworks.com/help/pdf_doc/distcomp/distcomp.pdf

Calculer sur GPU avec MATLAB Eric.Anterieu@cesbio.cnes.fr

1 Introduction

MATLAB + GPU = PCT

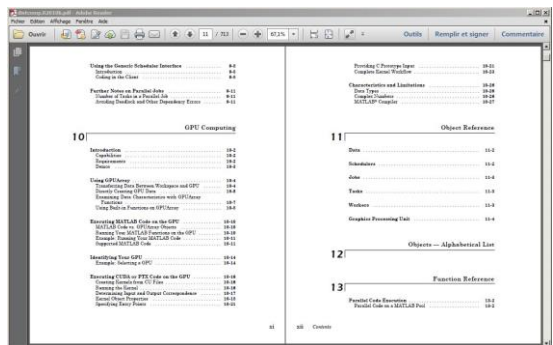


http://www.mathworks.com/help/release/R2010b/pdf_doc/distcomp/distcomp.pdf

Calculer sur GPU avec MATLAB Eric.Anterieu@cesbio.cnes.fr

1 Introduction

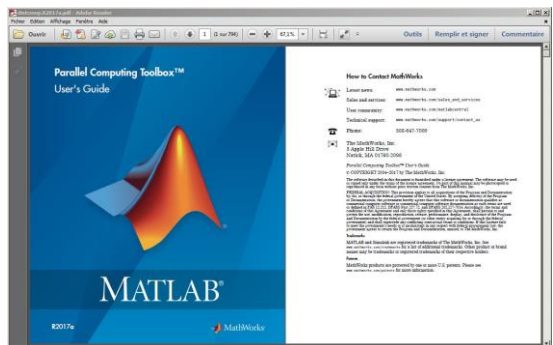
MATLAB + GPU = PCT



Calculer sur GPU avec MATLAB Eric.Anterieu@cesbio.cnes.fr

1 Introduction

MATLAB + GPU = PCT



http://www.mathworks.com/help/release/R2017a/pdf_doc/distcomp/distcomp.pdf

Calculer sur GPU avec MATLAB Eric.Anterieu@cesbio.cnes.fr

1 Introduction

MATLAB + GPU = PCT

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

1 Introduction

Ouvrages sur MATLAB/PCT et GPU

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

1 Introduction

Ouvrages sur MATLAB/PCT et GPU

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

1 Introduction

DELL R720

- 2 INTEL Xeon DecaCore E5-2680-V2 IvyBridge-EP with 128 GB
- 2 NVIDIA Tesla K40m with 12 GB

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

1 Introduction

NVIDIA GPGPU

	C1060	K20Xm	K40m
released	Apr. 9th, 2009	Nov. 12th, 2012	Nov. 18th, 2013
architecture / chips	Tesla / GT200	Kepler / GK110	Kepler / GK110B
CUDA compute ability	1.3	3.5	3.5
bus interface	PCIe 2.0 x16	PCIe 2.0 x16	PCIe 3.0 x16
CUDA cores	240 @ 1296 MHz	2688 @ 732 MHz	2880 @ 745 MHz
peak SP flops	933 Gflop/s	3950 Gflop/s	4290 Gflop/s
peak DP flops	78 Gflop/s	1310 Gflop/s	1430 Gflop/s
GDDR memory	4 GB @ 800 MHz	6 GB @ 2.6 GHz	12 GB @ 3 GHz
peak memory bandwidth	102 GB/s	250 GB/s	288 GB/s

www.nvidia.fr/page/tesla_product_literature.html

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

1 Introduction

PCIexpress

	PCIe 1.0	PCIe 2.0	PCIe 3.0	PCIe 4.0	PCIe 5.0
released	May 2003	Jan., 2007	Nov., 2010	Jun., 2017	Jun., 2019
x1	250 MB/s	500 MB/s	1 GB/s	2 GB/s	4 GB/s
x4	1 GB/s	2 GB/s	4 GB/s	8 GB/s	16 GB/s
x8	2 GB/s	4 GB/s	8 GB/s	16 GB/s	32 GB/s
x16	4 GB/s	8 GB/s	16 GB/s	32 GB/s	64 GB/s

en.wikipedia.org/wiki/PCI_Express

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

1 Introduction

Les étapes

- 1) Identifier et sélectionner un GPU
- 2) Transférer les données vers le GPU via le PCIe: *host* → *device*
- 3) Traiter les données sur le GPU
- 4) Transférer les résultats vers le CPU via le PCIe: *device* → *host*

- PCT se charge de l'identification, la sélection et des transferts
- Goulot: les transferts de données sur PCIe entre CPU et GPU
- Attention à la dissymétrie des performances des transferts via PCIe
- Privilégier la création des données directement sur GPU

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

1 Introduction

Les étapes

- 1) Identifier et sélectionner un GPU
- 2) Transférer les données vers le GPU via le PCIe: *host* → *device*
- 3) Traiter les données sur le GPU:
 - a) *built-in functions / array operators*
 - b) *custom functions / element-wise operators*
 - c) *user-defined CUDA kernels*
 - d) *user-defined CUDA-enabled MEX files*
→ *cuBLAS / cuFFT / cuRAND* routines
→ *ARRAYFIRE / CULA / MAGMA ...* routines
- 3) Transférer les résultats vers le CPU via le PCIe: *device* → *host*

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

1 Introduction

Les étapes

- 1) Identifier et sélectionner un GPU
- 2) Transférer les données vers le GPU via le PCIe: *host* → *device*
- 3) Traiter les données sur le GPU:
 - a) *built-in functions / array operators*
 - b) *custom functions / element-wise operators*
 - c) *user-defined CUDA kernels*
 - d) *user-defined CUDA-enabled MEX files*
→ *cuBLAS / cuFFT / cuRAND* routines
→ *ARRAYFIRE / CULA / MAGMA ...* routines
- 3) Transférer les résultats vers le CPU via le PCIe: *device* → *host*

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

1 Introduction

Les étapes

- 1) Identifier et sélectionner un GPU
- 2) Transférer les données vers le GPU via le PCIe: *host* → *device*
- 3) Traiter les données sur le GPU:
 - a) *built-in functions / array operators*
 - b) *custom functions / element-wise operators*
 - c) *user-defined CUDA kernels*
 - d) *user-defined CUDA-enabled MEX files*
→ *cuBLAS / cuFFT / cuRAND* routines
→ *ARRAYFIRE / CULA / MAGMA ...* routines
- 3) Transférer les résultats vers le CPU via le PCIe: *device* → *host*

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

2 Identifier / sélectionner un GPU

GPUDevice

La fonction `gpuDeviceCount` de la PCT retourne le nombre de GPU présente sur le bus PCIe.

La fonction `gpuDevice` sélectionne/réinitialise un GPU:

```
>> gpuDev = gpuDevice(gpuId);
```

Elle retourne des informations de la commande `nvidia-smi` dans une structure de type `GPUDevice` qui a évolué avec les versions de la PCT.

Depuis la version R2012a, la fonction `reset` réinitialise un GPU:

```
>> reset(gpuDev);
```

Depuis la version R2014b, la fonction `wait` bloque toute exécution sur CPU jusqu'à ce que toutes les opérations en suspens sur le GPU soient terminées:

```
>> wait(gpuDev);
```

Lors des transferts de données entre CPU et GPU, MATLAB effectue ce blocage automatiquement jusqu'à la fin du transfert.

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

2 Identifier / sélectionner un GPU

GPUDevice

Propriétés de la structure `GPUDevice`:

- **Name** Name of the device.
- **Index** Index by which the device can be selected.
- **ComputeCapability** Computational capability of the device.
- **SupportsDouble** Indicates if this device can support double precision operations.
- **DriverVersion** Version of the CUDA device driver currently in use.
- **ToolkitVersion** Version of the CUDA toolkit used by the current release of MATLAB.
- **MaxThreadsPerBlock** Maximum supported number of threads per block during execution of a CUDA Kernel.
- **MaxShmemPerBlock** Maximum supported amount of shared memory that can be used by a thread block during execution of a CUDA Kernel.
- **MaxThreadBlockSize** Maximum size in each dimension for thread block. Each dimension of a thread block must not exceed these dimensions. Also, the product of the thread block size must not exceed value of **MaxThreadsPerBlock**.
- **MaxGridSize** Maximum size of grid of thread blocks.

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

2 Identifier / sélectionner un GPU

GPUDevice

Thread (Warp) / 3d-Block / 3d-Grid:

Dans un kernel, thread Id unique:
`int idx=blockIdx.x*blockDim.x+threadIdx.x;`

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

2 Identifier / sélectionner un GPU

GPUDevice

Propriétés de la structure **GPUDevice**:

- > **SIMDWidth** Number of simultaneously executing threads.
- > **TotalMemory** Total memory (in bytes) on the device.
- > **AvailableMemory** Total amount of memory (in bytes) available for data.
- > **MultiprocessorCount** Number of vector processors present on the device.
- > **ClockRateKHz** Peak clock rate of the device in kHz.
- > **ComputeMode** Compute mode of the device, according to the following values:
 - 'Default' — The device is not restricted and can be used by multiple applications simultaneously. MATLAB can share the device with other applications, including other MATLAB sessions or workers.
 - 'Exclusive thread' or 'Exclusive process' — The device can be used by only one application at a time. While the device is selected in MATLAB, it cannot be used by other applications, including other MATLAB sessions or workers.
 - 'Prohibited' — The device cannot be used.

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

2 Identifier / sélectionner un GPU

GPUDevice

Propriétés de la structure **GPUDevice**:

- > **GPUOverlapsTransfers** Indicates if the device supports overlapped transfers.
- > **KernelExecutionTimeout** Indicates if the device can abort long-running kernels. If true, the operating system places an upper bound on the time allowed for the CUDA kernel to execute, after which the CUDA driver times out the kernel and returns an error.
- > **CanMapHostMemory** Indicates if the device supports mapping host memory into the CUDA address space.
- > **DeviceSupported** Indicates if PCT can use this device. Not all devices are supported; for example, if their **ComputeCapability** is insufficient, the toolbox cannot use them.
- > **DeviceSelected** Indicates if this is the currently selected device.

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

2 Identifier / sélectionner un GPU

Exemple

```

MATLAB
> gpuDevice(1)
Properties:
      Name: 'Tesla C1060'
      Index: 1
      ComputeCapability: '1.3'
      SupportsDouble: 1
      DriverVersion: 3.1
      MaxThreadsPerBlock: 512
      MaxShmemPerBlock: 16384
      MaxThreadBlockSize: [512 512 64]
      MaxGridSize: [65535 65535]
      SIMDWidth: 32
      TotalMemory: 4.2948e+09
      FreeMemory: 4.2563e+09
      MultiprocessorCount: 30
      GPUOverlapsTransfers: 1
      KernelExecutionTimeout: 0
      DeviceSupported: 1
      DeviceSelected: 1
    >>
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

2 Identifier / sélectionner un GPU

Exemple

```

MATLAB
> gpuDevice(1)
Properties:
      Name: 'Tesla C1060'
      Index: 1
      ComputeCapability: '1.3'
      SupportsDouble: 1
      DriverVersion: 3.2
      MaxThreadsPerBlock: 512
      MaxShmemPerBlock: 16384
      MaxThreadBlockSize: [512 512 64]
      MaxGridSize: [65535 65535]
      SIMDWidth: 32
      TotalMemory: 4.2948e+09
      FreeMemory: 4.2563e+09
      MultiprocessorCount: 30
      ComputeMode: 'Default' ← R2011a
      GPUOverlapsTransfers: 1
      KernelExecutionTimeout: 0
      CanMapHostMemory: 1 ← R2011a
      DeviceSupported: 1
      DeviceSelected: 1
    >>
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

2 Identifier / sélectionner un GPU

Exemple

```

MATLAB
> gpuDevice(1)
Properties:
      Name: 'Tesla C1060'
      Index: 1
      ComputeCapability: '1.3'
      SupportsDouble: 1
      DriverVersion: 4
      MaxThreadsPerBlock: 512
      MaxShmemPerBlock: 16384
      MaxThreadBlockSize: [512 512 64]
      MaxGridSize: [65535 65535]
      SIMDWidth: 32
      TotalMemory: 4.2948e+09
      FreeMemory: 4.2563e+09
      MultiprocessorCount: 30
      ClockRateKHz: 1296000 ← R2011b
      ComputeMode: 'Default'
      GPUOverlapsTransfers: 1
      KernelExecutionTimeout: 0
      CanMapHostMemory: 1
      DeviceSupported: 1
      DeviceSelected: 1
    >>
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

② Identifier / sélectionner un GPU

Exemple

```

MATLAB
> gpuDevice(1)
Properties:
    Name: 'Tesla C1060'
    Index: 1
    ComputeCapability: '1.3'
    SupportsDouble: 1
    DriverVersion: 5
    ToolkitVersion: 5
    MaxThreadsPerBlock: 512
    MaxShmemPerBlock: 16384
    MaxThreadBlockSize: [512 512 64]
    MaxGridSize: [65535 65535]
    SIMDWidth: 32
    TotalMemory: 4.2948e+09
    FreeMemory: 4.2563e+09
    MultiprocessorCount: 30
    ClockRateKHz: 1296000
    ComputeMode: 'Default'
    GPUOverlapsTransfers: 1
    KernelExecutionTimeout: 0
    CanMapHostMemory: 1
    DeviceSupported: 1
    DeviceSelected: 1
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@cesbio.cnes.fr

② Identifier / sélectionner un GPU

Exemple

```

MATLAB
> gpuDevice(1)
Properties:
    Name: 'Tesla C1060'
    Index: 1
    ComputeCapability: '1.3'
    SupportsDouble: 1
    DriverVersion: 5.5
    ToolkitVersion: 5.5
    MaxThreadsPerBlock: 512
    MaxShmemPerBlock: 16384
    MaxThreadBlockSize: [512 512 64]
    MaxGridSize: [65535 65535]
    SIMDWidth: 32
    TotalMemory: 4.2948e+09
    AvailableMemory: 4.2563e+09
    MultiprocessorCount: 30
    ClockRateKHz: 1296000
    ComputeMode: 'Default'
    GPUOverlapsTransfers: 1
    KernelExecutionTimeout: 0
    CanMapHostMemory: 1
    DeviceSupported: 1
    DeviceSelected: 1
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@cesbio.cnes.fr

② Identifier / sélectionner un GPU

Exemple

```

MATLAB
> gpuDevice(1)
Properties:
    Name: 'Tesla K20Xm'
    Index: 1
    ComputeCapability: '3.5'
    SupportsDouble: 1
    DriverVersion: 6
    ToolkitVersion: 5
    MaxThreadsPerBlock: 1024
    MaxShmemPerBlock: 49152
    MaxThreadBlockSize: [1024 1024 64]
    MaxGridSize: [2.1475e+09 65535 65535]
    SIMDWidth: 32
    TotalMemory: 6.0393e+09
    AvailableMemory: 5.9048e+09
    MultiprocessorCount: 14
    ClockRateKHz: 732000
    ComputeMode: 'Default'
    GPUOverlapsTransfers: 1
    KernelExecutionTimeout: 0
    CanMapHostMemory: 1
    DeviceSupported: 1
    DeviceSelected: 1
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@cesbio.cnes.fr

② Identifier / sélectionner un GPU

Exemple

```

MATLAB
> gpuDevice(2)
Properties:
    Name: 'Tesla K40m'
    Index: 2
    ComputeCapability: '3.5'
    SupportsDouble: 1
    DriverVersion: 6
    ToolkitVersion: 5
    MaxThreadsPerBlock: 1024
    MaxShmemPerBlock: 49152
    MaxThreadBlockSize: [1024 1024 64]
    MaxGridSize: [2.1475e+09 65535 65535]
    SIMDWidth: 32
    TotalMemory: 1.2079e+10
    AvailableMemory: 1.1896e+10
    MultiprocessorCount: 15
    ClockRateKHz: 745000
    ComputeMode: 'Default'
    GPUOverlapsTransfers: 1
    KernelExecutionTimeout: 0
    CanMapHostMemory: 1
    DeviceSupported: 1
    DeviceSelected: 1
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@cesbio.cnes.fr

② Identifier / sélectionner un GPU

Exemple

```

$ SHELL C
[antierieu@cuda pct]$ /usr/local/cuda-5.5/samples/1_utilities/deviceQuery/deviceQuery
Detected 2 CUDA Capable device(s)

Device 0: "Tesla K20Xm"
  CUDA Driver Version / Runtime Version      6.0 / 5.5
  CUDA Capability Major/Minor version number: 3.5
  Total amount of global memory:             5760 Mbytes (6039339008 bytes)
  (14) Multiprocessors, (192) CUDA Cores/MP: 2496 CUDA Cores
  GPU clock rate:                            732 MHz (0.73 GHz)
  Memory Clock rate:                         2500 Mhz
  Memory Bus Width:                          384-bit
  L2 Cache Size:                             1572864 bytes
  Maximum Texture Size (x,y,z):              1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers 1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers 2D=(16384, 16384), 2048 layers
  Total amount of constant memory:           65536 bytes
  Total amount of shared memory per block:   49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                 32
  Maximum number of threads per multiprocessor: 2048
  Maximum number of threads per block:      1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Maximum memory pitch:                      2147483647 bytes
  Texture alignment:                         512 bytes
  Concurrent copy and kernel execution:     Yes with 2 copy engine(s)
  Run time limit on kernels:                 No
  Integrated GPU sharing Host Memory:        No
  Support host page-locked memory mapping:   Yes
  Alignment requirement for Surfaces:        Yes
  Device has ECC support:                     Enabled
  Device supports Unified Addressing (UVA):  Yes
  Device PCI Bus ID / PCI location ID:       66 / 0
  Compute Mode:                              < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@cesbio.cnes.fr

② Identifier / sélectionner un GPU

Exemple

```

$ SHELL C
Device 1: "Tesla K40m"
  CUDA Driver Version / Runtime Version      6.0 / 5.5
  CUDA Capability Major/Minor version number: 3.5
  Total amount of global memory:             11520 Mbytes (12079136768 bytes)
  (15) Multiprocessors, (192) CUDA Cores/MP: 2880 CUDA Cores
  GPU clock rate:                            745 MHz (0.75 GHz)
  Memory Clock rate:                         3004 Mhz
  Memory Bus Width:                          384-bit
  L2 Cache Size:                             1572864 bytes
  Maximum Texture Dimension Size (x,y,z)    1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers 1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers 2D=(16384, 16384), 2048 layers
  Total amount of constant memory:           65536 bytes
  Total amount of shared memory per block:   49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                 32
  Maximum number of threads per multiprocessor: 2048
  Maximum number of threads per block:      1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Maximum memory pitch:                      2147483647 bytes
  Texture alignment:                         512 bytes
  Concurrent copy and kernel execution:     Yes with 2 copy engine(s)
  Run time limit on kernels:                 No
  Integrated GPU sharing Host Memory:        No
  Support host page-locked memory mapping:   Yes
  Alignment requirement for Surfaces:        Yes
  Device has ECC support:                     Enabled
  Device supports Unified Addressing (UVA):  Yes
  Device PCI Bus ID / PCI location ID:       66 / 0
  Compute Mode:                              < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@cesbio.cnes.fr

2 Identifier / sélectionner un GPU

Travailler avec plusieurs GPU

Exécuter le même code en // avec **spmd**:

```

GOpnu_spmd.m
% open pool of workers
matlabpool('open','local',gpuDeviceCount);
% execute code in parallel on workers
spmd(gpuDeviceCount)
% GPU selection
gpuId = labindex;
gpuDev = gpuDevice(gpuId);
% work on GPU
fprintf(1, 'worker %d connected to gpu %d\n', labindex, gpuDev.Index);
% GPU disconnect
reset(gpuDev);
end
% close pool of workers
matlabpool('close');
    
```

```

MATLAB
> GOpnu_spmd
Starting matlabpool using the 'local' profile ... connected to 2 workers.
worker #1 connected to gpu 1
worker #2 connected to gpu 2
Sending a stop signal to all the workers ... stopped.
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

2 Identifier / sélectionner un GPU

Travailler avec plusieurs GPU

Exécuter les itérations d'une boucle en // avec **parfor**:

```

GOpnu_parfor.m
% open pool of workers
matlabpool('open','local',gpuDeviceCount);
% execute loop iterations in parallel on workers
parfor(index = 1:gpuDeviceCount, gpuDeviceCount)
% GPU selection
gpuId = index;
gpuDev = gpuDevice(gpuId);
% work on GPU
fprintf(1, 'iteration %d using gpu %d\n', index, gpuDev.Index);
% GPU disconnect
reset(gpuDev);
end
% close pool of workers
matlabpool('close');
    
```

```

MATLAB
> GOpnu_parfor
Starting matlabpool using the 'local' profile ... connected to 2 workers.
iteration #1 using gpu 1
iteration #2 using gpu 2
Sending a stop signal to all the workers ... stopped.
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

2 Identifier / sélectionner un GPU

Travailler avec plusieurs GPU

Lorsqu'on fait abstraction des données, les deux approches semblent identiques alors qu'elles sont radicalement différentes:

- > l'approche **spmd** s'adresse aux tableaux *distributed*: la durée de vie des variables de type Composite est celle du matlabpool.
- > l'approche **parfor** s'adresse aux tableaux *sliced*: chaque itération ne doit adresser qu'une partie, une tranche ou *slice*, des tableaux.

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

3 Transférer des données

Transférer des données du CPU vers le GPU

La fonction **gpuArray** de la PCT permet de transférer un tableau de l'espace de travail sur la CPU vers la mémoire d'un GPU:

```
>> varGPU = gpuArray(varCPU);
```

La syntaxe n'a pas évolué avec les versions de la PCT.

```

MATLAB
>> A = eye(100,100,'double');
>> B = gpuArray(A);
>> whos
Name      Size      Bytes  Class
A         100x100  80000  double
B         100x100   108    parallel.gpu.GPUArray ← R2010b → R2012a
    
```

Tous les tableaux (quelque soit leurs dimensions) de la classe **GPUArray** occupent 108 octets dans l'espace de travail de la CPU: ils ne servent qu'à des fins d'adressage/indexation.

Le contenu du tableau **B** est bel et bien dans la mémoire du GPU et occupe 80000 octets, comme le tableau **A** dans la mémoire du CPU.

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

3 Transférer des données

Transférer des données du CPU vers le GPU

La fonction **gpuArray** de la PCT permet de transférer un tableau de l'espace de travail sur la CPU vers la mémoire d'un GPU:

```
>> varGPU = gpuArray(varCPU);
```

La syntaxe n'a pas évolué avec les versions de la PCT.

```

MATLAB
>> A = eye(100,100,'double');
>> B = gpuArray(A);
>> whos
Name      Size      Bytes  Class
A         100x100  80000  double
B         100x100   108    gpuArray ← R2012b → R2017a
    
```

Tous les tableaux (quelque soit leurs dimensions) de la classe **gpuArray** occupent 108 octets dans l'espace de travail de la CPU: ils ne servent qu'à des fins d'adressage/indexation.

Le contenu du tableau **B** est bel et bien dans la mémoire du GPU et occupe 80000 octets, comme le tableau **A** dans la mémoire du CPU.

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

3 Transférer des données

Transférer des données du CPU vers le GPU

La fonction **gpuArray** de la PCT permet de transférer un tableau de l'espace de travail sur la CPU vers la mémoire d'un GPU:

```
>> varGPU = gpuArray(varCPU);
```

La syntaxe n'a pas évolué avec les versions de la PCT.

```

MATLAB
>> A = eye(100,100,'double');
>> B = gpuArray(A);
>> whos
Name      Size      Bytes  Class
A         100x100  80000  double
B         100x100   108    gpuArray ← R2012b → R2017a
    
```

Depuis la version **R2013b**, la fonction **classUnderlying** retourne le type (la classe) des éléments d'un tableau de la classe **gpuArray**.

```

MATLAB
>> classUnderlying(B)
ans = double
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

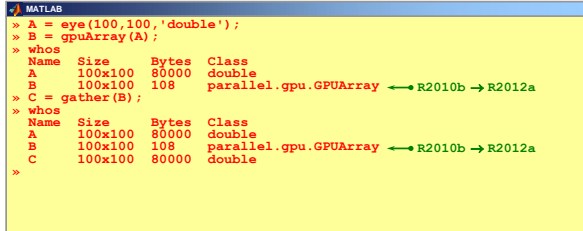
③ Transférer des données

Transférer des données du GPU vers la CPU

La fonction **gather** de la PCT permet de transférer un tableau de la mémoire d'un GPU vers l'espace de travail sur la CPU:

```
>> varCPU = gather(varGPU);
```

La syntaxe n'a pas évolué avec les versions de la PCT.



```

>> MATLAB
>> A = eye(100,100,'double');
>> B = gpuArray(A);
>> whos
Name Size Bytes Class
A 100x100 80000 double
B 100x100 108 parallel.gpu.GPUArray ← R2010b → R2012a
>> C = gather(B);
>> whos
Name Size Bytes Class
A 100x100 80000 double
B 100x100 108 parallel.gpu.GPUArray ← R2010b → R2012a
C 100x100 80000 double
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

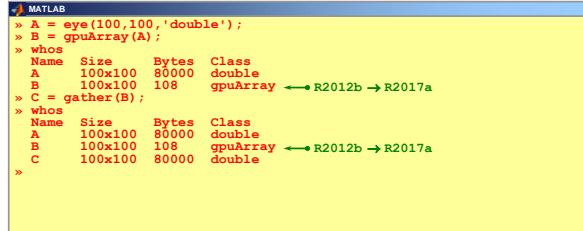
③ Transférer des données

Transférer des données du GPU vers la CPU

La fonction **gather** de la PCT permet de transférer un tableau de la mémoire d'un GPU vers l'espace de travail sur la CPU:

```
>> varCPU = gather(varGPU);
```

La syntaxe n'a pas évolué avec les versions de la PCT.



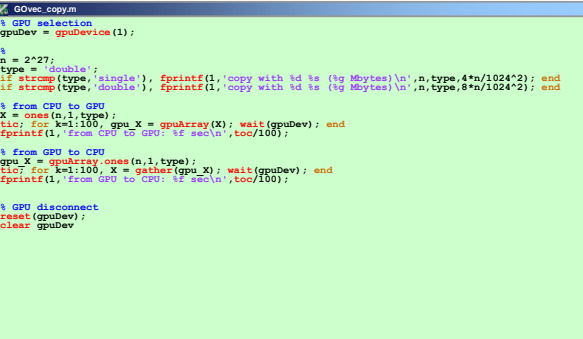
```

>> MATLAB
>> A = eye(100,100,'double');
>> B = gpuArray(A);
>> whos
Name Size Bytes Class
A 100x100 80000 double
B 100x100 108 gpuArray ← R2012b → R2017a
>> C = gather(B);
>> whos
Name Size Bytes Class
A 100x100 80000 double
B 100x100 108 gpuArray ← R2012b → R2017a
C 100x100 80000 double
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

③ Transférer des données

Exemple: CPU ↔ GPU



```

% GPU selection
gpuDev = gpuDevice(1);

n = 2^27;
type = 'double';
if strcmp(type,'single'), fprintf(1,'copy with %d %s (%g Mbytes)\n',n,type,4*n/1024^2); end
if strcmp(type,'double'), fprintf(1,'copy with %d %s (%g Mbytes)\n',n,type,8*n/1024^2); end

% from CPU to GPU
X = ones(n,1,type);
tic; for k=1:100, gpu_X = gpuArray(X); wait(gpuDev); end
fprintf(1,'from CPU to GPU: %d sec\n',toc/100);

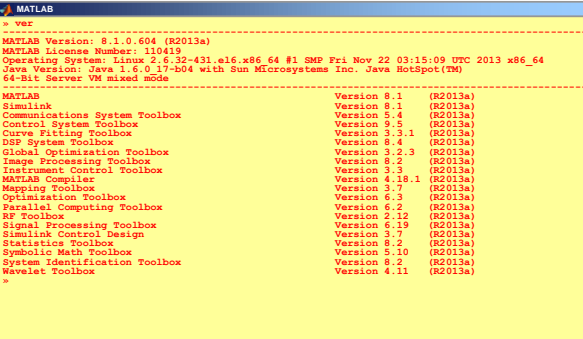
% from GPU to CPU
gpu_X = gpuArray(ones(n,1,type));
tic; for k=1:100, X = gather(gpu_X); wait(gpuDev); end
fprintf(1,'from GPU to CPU: %d sec\n',toc/100);

% GPU disconnect
reset(gpuDev);
clear gpuDev
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

③ Transférer des données

Exemple: CPU ↔ GPU



```

MATLAB
-----
MATLAB Version: 8.1.0.604 (R2013a)
MATLAB License Number: 110419
Operating System: Linux 2.6.32-431.el6.x86_64 #1 SMP Fri Nov 22 03:15:09 UTC 2013 x86_64
Java Version: Java 1.6.0.17-b04 with Sun Microsystems Inc. Java HotSpot(TM)
64-Bit Server VM mixed mode
-----
MATLAB Version 8.1 (R2013a)
Simulink Version 8.1 (R2013a)
Communications System Toolbox Version 5.4 (R2013a)
Control System Toolbox Version 9.5 (R2013a)
Curve Fitting Toolbox Version 3.3.1 (R2013a)
DSP System Toolbox Version 8.4 (R2013a)
Global Optimization Toolbox Version 3.2.3 (R2013a)
Image Processing Toolbox Version 8.2 (R2013a)
Instrument Control Toolbox Version 3.3 (R2013a)
MATLAB Compiler Version 4.18.1 (R2013a)
Mapping Toolbox Version 3.7 (R2013a)
Optimization Toolbox Version 6.3 (R2013a)
Parallel Computing Toolbox Version 6.2 (R2013a)
RF Toolbox Version 2.12 (R2013a)
Signal Processing Toolbox Version 6.19 (R2013a)
Simulink Control Design Version 3.7 (R2013a)
Statistics Toolbox Version 8.2 (R2013a)
Symbolic Math Toolbox Version 5.10 (R2013a)
System Identification Toolbox Version 8.2 (R2013a)
Wavelet Toolbox Version 4.11 (R2013a)
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

③ Transférer des données

Exemple: CPU ↔ GPU



```

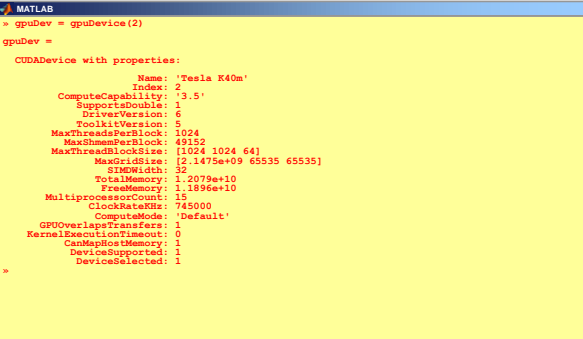
MATLAB
>> gpuDev = gpuDevice(1)
gpuDev =

CUDADevice with properties:
    Name: 'Tesla K20xm'
    Index: 1
    ComputeCapability: '3.5'
    SupportDouble: 1
    DriverVersion: 6
    ToolkitVersion: 5
    MaxThreadsPerBlock: 1024
    MaxShmemPerBlock: 49152
    MaxThreadBlockSize: [1024 1024 64]
    MaxGridSize: [2.1475e+09 65535 65535]
    SIMDWidth: 32
    TotalMemory: 6.0393e+09
    FreeMemory: 5.9048e+09
    MultiprocessorCount: 14
    ClockRateKHz: 732000
    ComputeMode: 'Default'
    GPUOverlapsTransfers: 1
    KernelExecutionTimeout: 0
    CanMapHostMemory: 1
    DeviceSupported: 1
    DeviceSelected: 1
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

③ Transférer des données

Exemple: CPU ↔ GPU



```

MATLAB
>> gpuDev = gpuDevice(2)
gpuDev =

CUDADevice with properties:
    Name: 'Tesla K40m'
    Index: 2
    ComputeCapability: '3.5'
    SupportDouble: 1
    DriverVersion: 6
    ToolkitVersion: 5
    MaxThreadsPerBlock: 1024
    MaxShmemPerBlock: 49152
    MaxThreadBlockSize: [1024 1024 64]
    MaxGridSize: [2.1475e+09 65535 65535]
    SIMDWidth: 32
    TotalMemory: 1.2079e+10
    FreeMemory: 1.1896e+10
    MultiprocessorCount: 15
    ClockRateKHz: 745000
    ComputeMode: 'Default'
    GPUOverlapsTransfers: 1
    KernelExecutionTimeout: 0
    CanMapHostMemory: 1
    DeviceSupported: 1
    DeviceSelected: 1
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

3 Transférer des données

Créer des données directement sur GPU

Un grand nombre de fonctions (**zeros**, **ones**, **eye**, **rand**...) permettent de créer des données directement dans la mémoire d'un GPU, sans avoir à les transférer depuis l'espace de travail sur la CPU.
! La syntaxe a évolué avec les versions de la PCT.

```

MATLAB
>> A = eye(100,100,'double');
>> B = gpuArray.eye(100,100,'double'); ← R2012b → R2013b
>> whos
Name      Size      Bytes    Class
A         100x100  80000    double
B         100x100  108      gpuArray ← R2012b → R2017a
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@cesbio.cnes.fr

3 Transférer des données

Créer des données directement sur GPU

Un grand nombre de fonctions (**zeros**, **ones**, **eye**, **rand**...) permettent de créer des données directement dans la mémoire d'un GPU, sans avoir à les transférer depuis l'espace de travail sur la CPU.
! La syntaxe a évolué avec les versions de la PCT.

```

MATLAB
>> A = eye(100,100,'double');
>> B = eye(100,100,'double','gpuArray'); ← R2014a → R2017a
>> whos
Name      Size      Bytes    Class
A         100x100  80000    double
B         100x100  108      gpuArray ← R2012b → R2017a
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@cesbio.cnes.fr

4 Exécuter des fonctions MATLAB

Les fonctions MATLAB compatibles GPU

La fonction **methods** liste les fonctions membres d'une classe:
>> methods('classname');
! Le nom et les membres/méthodes de la classe ont évolué avec les versions de la PCT:
R2010b → R2012a classname = parallel.gpu.GPUArray
R2012b → R2017a classname = gpuArray

L'aide d'une méthode d'une classe est obtenu via la commande **help**:
>> help classname/methodname

Dès lors qu'au moins un argument d'entrée d'une de ces méthodes est un **GPUArray** (resp. **gpuArray**), le calcul s'exécute sur le GPU et le résultat est un **GPUArray** (resp. **gpuArray**) dans la mémoire du GPU. Attention aux temps de transfert!

Calculer sur GPU avec MATLAB Eric.Antierieu@cesbio.cnes.fr

4 Exécuter des fonctions MATLAB

Les fonctions MATLAB compatibles GPU

abs acos acosd acosh acot acotd acoth acsc acscd acsch accumarray
all and angle any arrayfun asac asecd asech asin asind asinh
assert atan atan2 atan2d atand atanh bandwidth besselj bessely
beta betainc betaincinv betaln biog bicgstab bitand bitcmp bitget
bitor bitset bitshift bitxor blkdiag bsxfun cart2pol cart2sph
cast cat cdf2rdf ceil chol circshift classUnderlying colon compan
complex cond conj conv conv2 convn corrcoeff cos cosd cosh cot
cotd coth cov cross csc csd csch ctranpose cummax cummin
cumprod cumsum deg2rad del2 det detrend diag diff discretize disp
display dot double eig eps eq erf erfc erfcinv erfcx erfinv exp
expint expm expm1 eye factorial false fft fft2 fftn fftshift
filter filter2 find fix flip fliplr flipud floor fprintf full
gamma gammainc gammaincinv gammaln gather ge gmres gradient gt
hankel head histcounts horzcat hsv2rgb hypot idivide ifft ifft2
ifftn ifftshift imag ind2sub Inf inpolygon int16 int2str int32
int64 int8 interp1 interp2 interp3 interpn intersect inv ipermute
isaUnderlying isbanded iscolumn isdiag isempty isequal isequaln
isfinite isfloat ishermitian isinf isinteger islogical ismatrix

Calculer sur GPU avec MATLAB Eric.Antierieu@cesbio.cnes.fr

4 Exécuter des fonctions MATLAB

Les fonctions MATLAB compatibles GPU

ismember ismembertol isnan isnumeric isreal isrow issorted
issparse issymmetric istril istriu isvector kron ldivide le
legendre length log log10 log1p log2 logical lsqr lt lu mat2str
max median mean meshgrid min minus mldivide mod mode movmean
movstd movsum movvar mpower mrdivide mtimes NaN ndgrid ndims ne
nextpow2 nnz nonzeros norm normest not nthroot null num2str numel
ones or orth pagefun pcg perms permute pinv planerot plot plus
pol2cart poly polyarea polyder polyfit polyint polyval polyvalm
pow2 power prod psi qmr qr rad2deg rand randi randn randperm rank
rdivide real reallog realpow realsqrt rectint rem repelem repmat
reshape rgb2hsv roots rot90 round sec secd sech setdiff setxor
shiftdim sign sin sind single sinh size sort sortrows svds
spconvert sph2cart sprand sprandn sprandsym spconvert sph2cart
sprand sprandn sprandsym sprintf sqrt squeeze std sub2ind
subsasgn subsindex subspace subref sum superiorfloat svd svds
swapbytes tail tan tand tanh times toeplitz trace transpose trapz
tril triu true typecast uint16 uint32 uint64 uint8 uminus union
unique uniquetol unwrap uplus vander var vertcat xor zeros

Calculer sur GPU avec MATLAB Eric.Antierieu@cesbio.cnes.fr

4 Exécuter des fonctions MATLAB

Exemple: la fonction rand

```

C0vec_randm
% GPU selection
gpuDev = gpuDevice(1);
n = 2^26;
type = 'double';
if strcmp(type,'single'), fprintf(1,'rand with %d %s (%g Mbytes)\n',n,type,4*n/1024^2); end
if strcmp(type,'double'), fprintf(1,'rand with %d %s (%g Mbytes)\n',n,type,8*n/1024^2); end
% on CPU
X = zeros(n,1,type);
rng(0,'Combsuccessive');
tic; for k=1:100, X = rand(n,1,type); end
fprintf(1,'on CPU (direct): %f sec\n',toc/100);
% on GPU
gpu X = gpuArray.zeros(n,1,type);
parallel.gpu.rng(0,'Combsuccessive');
tic; for k=1:100, gpu X = gpuArray.rand(n,1,type); wait(gpuDev); end
fprintf(1,'on GPU (direct): %f sec\n',toc/100);
% GPU disconnect
reset(gpuDev);
clear gpuDev
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@cesbio.cnes.fr

4 Exécuter des fonctions MATLAB

Exemple: la fonction rand

	rand	float	double	single
R720 CPU (direct)	1.001	0.999	1.001	1.001
K20X GPU (direct)	0.023	0.023	0.023	0.023
K40 GPU (direct)	0.021	0.021	0.021	0.021

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

4 Exécuter des fonctions MATLAB

Exemple: la fonction rand

	rand	float	double	single
R720 CPU (direct)	1.001	0.999	1.001	1.001
K20X GPU (direct)	0.131	0.238	0.131	0.131
K40 GPU (direct)	0.128	0.236	0.128	0.128

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

4 Exécuter des fonctions MATLAB

Exemple: la fonction randn

	randn	float	double	single
R720 CPU (direct)	2.355	2.351	2.355	2.355
K20X GPU (direct)	0.070	0.070	0.070	0.070

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

4 Exécuter des fonctions MATLAB

Exemple: la fonction randn

	randn	float	double	single
R720 CPU (direct)	2.355	2.351	2.355	2.355
K20X GPU (direct)	0.070	0.070	0.070	0.070
K40 GPU (direct)	0.063	0.063	0.063	0.063

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

4 Exécuter des fonctions MATLAB

Exemple: la fonction randn

	randn	float	double	single
R720 CPU (direct)	2.355	2.351	2.355	2.355
K20X GPU (direct)	0.178	0.285	0.178	0.178
K40 GPU (direct)	0.171	0.278	0.171	0.171

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

4 Exécuter des fonctions MATLAB

Exemple: la fonction fft

	fft	float	double	single
R720 CPU (direct)	0.759	2.656	0.759	0.759
K20X GPU (direct)	0.033	0.064	0.033	0.033

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

4 Exécuter des fonctions MATLAB

Exemple: la fonction arrayfun

	float	double
R720 CPU (direct)	1.323	1.332
K20X GPU (direct)	0.316	0.612
K20X GPU (arrayfun)	0.222	0.434

Calculer sur GPU avec MATLAB Eric.Antierrieu@ceshio.cnes.fr

5 Exécuter des noyaux CUDA

CUDA ?

CUDA est l'acronyme de *Compute Unified Device Architecture*. C'est à la fois une architecture parallèle et un modèle de programmation parallèle développé par NVIDIA pour exploiter la puissance de calcul des GPU.

CUDA supporte plusieurs langages (notamment C/C++). L'API CUDA est constituée d'un pilote, d'un *runtime* et de bibliothèques. CUDA est aussi un langage dérivé du C qui étend les possibilités de ce dernier:

- 9 nouveaux mots clés,
- 24 nouveaux types,
- 62 nouvelles fonctions.

Ces extensions nécessitent un nouveau compilateur: les fichiers .cu sont appelés à contenir du code CUDA.

Calculer sur GPU avec MATLAB Eric.Antierrieu@ceshio.cnes.fr

5 Exécuter des noyaux CUDA

Les étapes

- 1) Ecrire un noyau .cu
- 2) Traduire le code du noyau .cu en .ptx
- 3) Créer une instance d'un **CUDAKernel** associé au code .ptx
- 4) Configurer les propriétés qui influent sur l'exécution
- 5) Exécuter le noyau sur le GPU à l'aide de la fonction **feval**

- PCT se charge de l'identification, la sélection et des transferts
- De nombreuses fonctions de MATLAB ont été étendues aux GPU
- Goulot: les transferts de données sur PCIe entre CPU et GPU
- Attention à la dissymétrie des performances des transferts via PCIe
- Privilégier la création des données directement sur GPU

Calculer sur GPU avec MATLAB Eric.Antierrieu@ceshio.cnes.fr

5 Exécuter des noyaux CUDA

Ecrire un noyau CUDA

CUDA étend le C/C++ en donnant la possibilité de définir des noyaux.

Une fonction C (classique) invoquée une fois est exécutée une seule fois. Un noyau CUDA invoqué une fois est exécuté plusieurs fois.

Un noyau est défini en utilisant la déclaration **__global__**. Un noyau est toujours de type **void**. Un noyau ne peut pas allouer/libérer de mémoire. Aucune information de taille/dimension des zones de mémoire pointées par des variables de type pointeur n'est transmise au noyau. Les pointeurs avec la spécification **const** sont protégés en écriture et sont considérés comme des **variables d'entrée**. Les autres peuvent être modifiés et sont considérés comme des **variables de sortie**.

Calculer sur GPU avec MATLAB Eric.Antierrieu@ceshio.cnes.fr

5 Exécuter des noyaux CUDA

Ecrire un noyau CUDA

```

vec_sppy.cu
#include <stdio.h>
#include <malloc.h>

// function that executes on the host
void MyFunction(const double a, const double *X, double *Y, const int n)
{
    int idx = 0;
    while (idx<n) {Y[idx] = a*X[idx]; idx += 1;}
}

// main routine that executes on the host
int main(void)
{
    const int n = 8;
    size_t size = n*sizeof(double);
    double *X_h, *Y_h;

    // allocate arrays on host
    X_h = (double *)malloc(size);
    Y_h = (double *)malloc(size);
    // initialize host arrays
    for (int i=0; i<n; i++) X_h[i]=1.0;
    // call kernel
    MyFunction(2.5,X_h,Y_h,n);

    // cleanup host memory
    free(X_h);
    free(Y_h);
}
    
```

Calculer sur GPU avec MATLAB Eric.Antierrieu@ceshio.cnes.fr

5 Exécuter des noyaux CUDA

Ecrire un noyau CUDA

```

vec_sppy.cu
#include <stdio.h>
#include <cuda.h>

// kernel that executes on the device
__global__ void MyKernel(const double a, const double *X, double *Y, const int n)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx<n) {Y[idx] = a*X[idx];}
}

// main routine that executes on the host
int main(void)
{
    const int n = 8;
    size_t size = n*sizeof(double);
    double *X_h, *Y_h;
    double *X_d, *Y_d;

    // allocate arrays on host
    X_h = (double *)malloc(size);
    Y_h = (double *)malloc(size);
    // initialize host arrays
    for (int i=0; i<n; i++) X_h[i]=1.0;

    // allocate arrays on device
    cudaMalloc((void **)&X_d,size);
    cudaMalloc((void **)&Y_d,size);
    // copy host arrays to device
    cudaMemcpy(X_h,X_d,X_h,size,cudaMemcpyHostToDevice);
    // call kernel
    MyKernel<<<<1,1>>>(2.5,X_d,Y_d,n);
    // copy device array to host
    cudaMemcpy(Y_h,Y_d,Y_d,size,cudaMemcpyDeviceToHost);

    // cleanup host memory
    free(X_h);
    free(Y_h);

    // cleanup device memory
    cudaFree(X_d);
    cudaFree(Y_d);
}
    
```

Calculer sur GPU avec MATLAB Eric.Antierrieu@ceshio.cnes.fr

5 Exécuter des noyaux CUDA

Compiler/Traduire un noyau CUDA

Le compilateur **nvcc**, est un compilateur propriétaire disponible avec le SDK CUDA de NVIDIA.

developer.nvidia.com/cuda-toolkit
docs.nvidia.com/cuda/cuda-compiler-driver-nvcc

Comme le code CUDA est amené à s'exécuter à la fois sur des CPU et sur des GPU, le premier rôle de **nvcc** est de séparer le code en deux parties et d'envoyer aux CPU le code destiné aux CPU et aux GPU le code destiné aux GPU:

```
$ nvcc -o MyMain MyMain.cu
```

ou:

```
$ nvcc -o MyMain.o -c MyMain.cu
$ g++ -o MyMain MyMain.o -lcudart
```

devblogs.nvidia.com/parallelforall/separate-compilation-linking-cuda-device-code

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

5 Exécuter des noyaux CUDA

Compiler/Traduire un noyau CUDA

```
MyKernel.cu
#include <cuda.h>
// kernel that executes on the device
__global__ void MyKernel(const double a, const double *X, double *Y, const int n)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx<n) Y[idx] = a*X[idx];
}
```

Avec l'option **-ptx**, le compilateur CUDA de NVIDIA traduit le code en un pseudo-code assembleur:

```
$ nvcc -ptx MyKernel.cu
```

Il est possible de cibler une architecture en particulier à l'aide l'option: **-generate arch=compute_nm,code=sm_nm** où **nm** est (lié à) la valeur de **ComputeCapability** de la structure **GPUDevice**.

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

5 Exécuter des noyaux CUDA

Compiler/Traduire un noyau CUDA

```
MyKernel.ptx
Generated by NVIDIA NVVM Compiler
Cuda compilation tools, release 5.5, V5.5.0
.version 3.2
.target sm_35
.address_size 64
.file 1 "/home/anterrieu/pct/MyKernel.cu", 1503589154, 215
.file 2 "/usr/local/cuda-5.5/bin/./include/cuda_device_runtime_api.h", 1391706351, 7655
.weak .func (.param .b32 func_retval0) cudaMalloc(
    .param .b64 cudaMalloc_param_0,
    .param .b64 cudaMalloc_param_1
)
{
    .reg s32 %rc2>
    mov.u32 %r1, 30;
    st.param.b32 [func_retval0+0], %r1;
    .loc 2 66 3
    ret;
}
.weak .func (.param .b32 func_retval0) cudaFuncGetAttributes(
    .param .b64 cudaFuncGetAttributes_param_0,
    .param .b64 cudaFuncGetAttributes_param_1
)
{
    .reg s32 %rc2>
    mov.u32 %r1, 30;
    st.param.b32 [func_retval0+0], %r1;
    .loc 2 71 3
    ret;
}
.visible .entry Z8MyKernelPKdPdi(
    .param .f64 Z8MyKernelPKdPdi_param_0,
    .param .u64 Z8MyKernelPKdPdi_param_1,

```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

5 Exécuter des noyaux CUDA

Compiler/Traduire un noyau CUDA

```
MyKernel.ptx
.param .u64 Z8MyKernelPKdPdi_param_2
.param .u32 Z8MyKernelPKdPdi_param_3
{
    .reg .pred %p<2>;
    .reg s32 %rc2>;
    .reg s64 %rd4>;
    .reg f64 %fd1>;
    ld.param.f64 %fd1, [Z8MyKernelPKdPdi_param_0];
    ld.param.u64 %rd3, [Z8MyKernelPKdPdi_param_1];
    ld.param.u64 %rd4, [Z8MyKernelPKdPdi_param_2];
    ld.param.u32 %r2, [Z8MyKernelPKdPdi_param_3];
    cvta.to.global.u64 %rd1, %rd4;
    cvta.to.global.u64 %rd2, %rd3;
    .loc 1 6 1
    mov.u32 %r3, %ntid.x;
    mov.u32 %r4, %ctaid.x;
    mov.u32 %r5, %tid.x;
    mad.lo.s32 %r1, %r3, %r4, %r5;
    .loc 1 8 1
    setp.ge.s32 %pl, %r1, %r2;
    %np1 br: %B2 2;
    mul.wide.s32 %rd5, %r1, 8;
    add.s64 %rd6, %rd2, %rd5;
    .loc 1 8 1
    ld.global.f64 %fd2, [%rd6];
    mul.f64 %fd5, %fd2, %fd1;
    add.s64 %rd7, %rd1, %rd5;
    .loc 1 8 1
    st.global.f64 [%rd7], %fd3;
    B2 2:
    .loc 1 9 2
    ret;
}
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

5 Exécuter des noyaux CUDA

Créer un noyau CUDA

La fonction **parallèle.gpu.CUDAKernel** de la PCT retourne un objet de type **CUDAKernel**:

```
>> gpuKern = parallèle.gpu.CUDAKernel (
    'kernel.ptx', 'kernel.cu');
```

Ce noyau peut être invoqué depuis le CPU pour être exécuté sur un GPU à l'aide de la fonction **feval**.

```
>> [out1, ..., outP] = feval (gpuKern, in1, ..., inN);
```

Depuis la version **R2012a**, la fonction **setConstantMemory** permet de donner une valeur aux variables dans la mémoire constante du noyau:

```
>> setConstantMemory (gpuKern, 'var', val, ...);
```

Ces variables ont été déclarées avec la directive **__constant__** (à ne pas confondre avec la spécification **const**).

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

5 Exécuter des noyaux CUDA

Créer un noyau CUDA

Propriétés de la structure **CUDAKernel**:

- > **GridSize** Number of blocks in the grid for this CUDA kernel. None of the elements of this vector can exceed the corresponding element in the vector of the **MaxGridSize** property of the **GPUDevice**.
- > **ThreadBlockSize** Number of threads in a block. None of the elements of this vector can exceed the corresponding element in the vector of the **MaxThreadBlockSize** property of the **GPUDevice**.
- > **MaxThreadsPerBlock** Maximum supported number of threads per block during execution of the CUDA Kernel. The product of the elements of **ThreadBlockSize** must not exceed this value.
- > **SharedMemorySize** Amount of dynamic shared memory (in bytes) that each thread block can use.

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

5 Exécuter des noyaux CUDA

Créer un noyau CUDA

Propriétés de la structure **CUDAKernel**:

- > **EntryPoint** (read-only) A character vector containing the actual entry point in the PTX code that the kernel is going to call.
- > **MaxNumLHSArguments** (read-only) The maximum number of left hand side arguments that this kernel supports.
- > **NumRHSArguments** (read-only) The required number of right hand side arguments needed to call this kernel.
- > **ArgumentTypes** (read-only) Cell array of character vectors, the same length as **NumRHSArguments**. Each of the character vectors indicates what the expected MATLAB type for that input is.

Calculer sur GPU avec MATLAB Eric.Anterieu@cbsio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: la fonction axpy de BLAS-1

```

vec_axpy.cu
#include <cuda.h>
// kernels that execute on the CUDA device
global void daxpy(const double A, const double *X, const double *Y, double *Z, const int N)
int idx = blockIdx.x * blockDim.x + threadIdx.x;
if (idx < N) Z[idx] = A*X[idx] + Y[idx];
}
global void saxpy(const float A, const float *X, const float *Y, float *Z, const int N)
int idx = blockIdx.x * blockDim.x + threadIdx.x;
if (idx < N) Z[idx] = A*X[idx] + Y[idx];
}
    
```

Calculer sur GPU avec MATLAB Eric.Anterieu@cbsio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: la fonction axpy de BLAS-1

```

GVec_axpy.cu
#include <cuda.h>
#include <sys/time.h>
#include <cuda.h>
#define DPRINTFMSG(tstart, tend) ((double)(tend.tv_sec - tstart.tv_sec) + 1E-06*(double)(tend.tv_usec - tstart.tv_usec))

// kernels that execute on the CUDA device
#include "vec_axpy.cu"

// main routine that executes on the host
int main(void)
{
    const int n = 2<<(26-1); // number of elements in arrays
    size_t size = n*sizeof(double); // size of arrays
    double *X_h, *Y_h, *Z_h; // pointers to host arrays
    double *X_d, *Y_d, *Z_d; // pointers to device arrays
    struct timeval tic, toc; // timers

    printf("axpy with %d double (%d Mbytes)\n", n, size/(2<<(20-1)));

    // allocate arrays on host
    X_h = (double *)malloc(size);
    Y_h = (double *)malloc(size);
    Z_h = (double *)malloc(size);
    // allocate arrays on device
    cudaMalloc((void **)&X_d, size);
    cudaMalloc((void **)&Y_d, size);
    cudaMalloc((void **)&Z_d, size);
    // initialize host arrays and copy them to device
    for (int i=0; i<n; i++) X_h[i] = 2.0;
    cudaMemset(X_d, X_h, size, cudaMemcpyHostToDevice);
    for (int i=0; i<n; i++) Y_h[i] = 0.5;
    cudaMemset(Y_d, Y_h, size, cudaMemcpyHostToDevice);
    // do calculation on device
    }
    
```

Calculer sur GPU avec MATLAB Eric.Anterieu@cbsio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: la fonction axpy de BLAS-1

```

GVec_axpy.cu
// do calculation on device
int blockSize = 1024;
int gridSize = n/blockSize + (n%blockSize == 0 ? 0 : 1); // threads in a block // blocks in the grid
gettimeofday(&tic, 0);
for (int k=0; k<gridSize; k++)
{
    daxpy<<gridSize, blockSize>>(3.0, X_d, Y_d, Z_d, n);
    cudaDeviceSynchronize();
    gettimeofday(&toc, 0);
    printf("on GPU (cuda/c++): %f sec\n", DIFFTIMESEC(tic, toc)/100.0);
    // retrieve result from device and copy it to host array
    cudaMemcpy(Z_h, Z_d, size, cudaMemcpyDeviceToHost);
    // use results
    for (int i=0; i<n; i++) printf("%d %f\n", i, Z_h[i]);
    for (int i=0; i<n; i++) printf("%d %f\n", i, Z_h[i]);
    // cleanup host memory
    free(X_h);
    free(Y_h);
    free(Z_h);
    // cleanup device memory
    cudaFree(X_d);
    cudaFree(Y_d);
    cudaFree(Z_d);
}
    
```

Calculer sur GPU avec MATLAB Eric.Anterieu@cbsio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: la fonction axpy de BLAS-1

```

$SHELL
[anterieu@cuda pctl]$ /usr/local/cuda-5.5/bin/nvcc -m64 -I/usr/local/cuda-5.5/include -I/usr/local/cuda-5.5/samples/common/inc -gencode arch=compute_35,code=sm_35 -o GVec_axpy.o -c GVec_axpy.cu
[anterieu@cuda pctl]$ /usr/bin/g++ -m64 -o GVec_axpy GVec_axpy.o -L/usr/local/cuda-5.5/lib64 -lcudart
[anterieu@cuda pctl]$ ./GVec_axpy
axpy with 71108864 float (256 Mbytes)
kernel<<<65536,1024>>
on GPU (cuda/c++): 0.004956 sec
[anterieu@cuda pctl]$ ./GVec_axpy
axpy with 71108864 float (256 Mbytes)
kernel<<<65536,1024>>
on GPU (cuda/c++): 0.004952 sec
[anterieu@cuda pctl]$ ./GVec_axpy
axpy with 71108864 float (256 Mbytes)
kernel<<<65536,1024>>
on GPU (cuda/c++): 0.004950 sec
[anterieu@cuda pctl]$ ./GVec_axpy
axpy with 71108864 float (256 Mbytes)
kernel<<<65536,1024>>
on GPU (cuda/c++): 0.004903 sec
[anterieu@cuda pctl]$
    
```

	axpy	float
GPU (cuda/c++)	0.00495	

Calculer sur GPU avec MATLAB Eric.Anterieu@cbsio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: la fonction axpy de BLAS-1

```

$SHELL
[anterieu@cuda pctl]$ /usr/local/cuda-5.5/bin/nvcc -m64 -I/usr/local/cuda-5.5/include -I/usr/local/cuda-5.5/samples/common/inc -gencode arch=compute_35,code=sm_35 -o GVec_axpy.o -c GVec_axpy.cu
[anterieu@cuda pctl]$ /usr/bin/g++ -m64 -o GVec_axpy GVec_axpy.o -L/usr/local/cuda-5.5/lib64 -lcudart
[anterieu@cuda pctl]$ ./GVec_axpy
axpy with 71108864 double (512 Mbytes)
kernel<<<65536,1024>>
on GPU (cuda/c++): 0.008966 sec
[anterieu@cuda pctl]$ ./GVec_axpy
axpy with 71108864 double (512 Mbytes)
kernel<<<65536,1024>>
on GPU (cuda/c++): 0.008965 sec
[anterieu@cuda pctl]$ ./GVec_axpy
axpy with 71108864 double (512 Mbytes)
kernel<<<65536,1024>>
on GPU (cuda/c++): 0.008964 sec
[anterieu@cuda pctl]$ ./GVec_axpy
axpy with 71108864 double (512 Mbytes)
kernel<<<65536,1024>>
on GPU (cuda/c++): 0.008965 sec
[anterieu@cuda pctl]$
    
```

	axpy	float	double
GPU (cuda/c++)	0.00495		0.00896

Calculer sur GPU avec MATLAB Eric.Anterieu@cbsio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: la fonction axpy de BLAS-1

```

vec_axpy.cu
#include <cuda.h>
// kernels that execute on the CUDA device
_global_ void daxpy(const double *A, const double *X, const double *Y, double *Z, const int N)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N) Z[idx] = A*X[idx] + Y[idx];
}
_global_ void saxpy(const float *A, const float *X, const float *Y, float *Z, const int N)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    if (idx < N) Z[idx] = A*X[idx] + Y[idx];
}
[
int n = 2*26;
type = 'double';
if strcmp(type, 'single'), fprintf(1, 'axpy with %d %s (%g Mbytes)\n', n, type, 4*n/1024*2); end
if strcmp(type, 'double'), fprintf(1, 'axpy with %d %s (%g Mbytes)\n', n, type, 8*n/1024*2); end
% on CPU
a = 3.0;
X = ones(n,1,type);
Y = ones(n,1,type);
Z = zeros(n,1,type);
tic; for k=1:100, Z = a*X + Y; end
fprintf(1, 'on CPU (direct): %f sec\n', toc/100);
% on GPU
gpu_a = gpuArray(a);
gpu_X = gpuArray(X);
gpu_Y = gpuArray(Y);
gpu_Z = gpuArray.zeros(n,1,type);
tic; for k=1:100, gpu_Z = gpu_a*gpu_X + gpu_Y; wait(gpuDev); end
fprintf(1, 'on GPU (direct): %f sec\n', toc/100);
gpuKern = parallel.gpu.CUDAKernel('vec_axpy.ptx', 'vec_axpy.cu', [type(1), 'axpy']);
gpuKern.ThreadBlockSize = min([numel(X) gpuDev.MaxThreadsPerBlock]); % threads in a block
gpuKern.GridSize = ceil(numel(X)/prod(gpuKern.ThreadBlockSize)); % blocks in the grid
fprintf(1, 'kernel <<<id>>>\n', prod(gpuKern.GridSize), prod(gpuKern.ThreadBlockSize));
gpu_Z = gpuArray.zeros(n,1,type);
tic; for k=1:100, gpu_Z = feval(gpuKern, gpu_a, gpu_X, gpu_Y, gpu_Z, n); wait(gpuDev); end
fprintf(1, 'on GPU (kernel): %f sec\n', toc/100);
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@ceshio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: la fonction axpy de BLAS-1

```

SHELL
[antierieu@cuda pct]$ /usr/local/cuda-5.5/bin/nvcc -m64 -I/usr/local/cuda-5.5/include -I/usr/local/cuda-5.5/samples/common/inc -gencode arch=compute_35,code=sm_35 -o GVec_axpy.o -c GVec_axpy.cu
[antierieu@cuda pct]$ /usr/bin/g++ -m64 -o GVec_axpy GVec_axpy.o -L/usr/local/cuda-5.5/lib64 -lcudart
[antierieu@cuda pct]$
[antierieu@cuda pct]$ /usr/local/cuda-5.5/bin/nvcc -gencode arch=compute_35,code=sm_35 -ptx vec_axpy.cu
[antierieu@cuda pct]$
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@ceshio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: la fonction axpy de BLAS-1

```

GVec_axpy.m
% GPU selection
gpuDev = gpuDevice(1);
n = 2*26;
type = 'double';
if strcmp(type, 'single'), fprintf(1, 'axpy with %d %s (%g Mbytes)\n', n, type, 4*n/1024*2); end
if strcmp(type, 'double'), fprintf(1, 'axpy with %d %s (%g Mbytes)\n', n, type, 8*n/1024*2); end
% on CPU
a = 3.0;
X = ones(n,1,type);
Y = ones(n,1,type);
Z = zeros(n,1,type);
tic; for k=1:100, Z = a*X + Y; end
fprintf(1, 'on CPU (direct): %f sec\n', toc/100);
% on GPU
gpu_a = gpuArray(a);
gpu_X = gpuArray(X);
gpu_Y = gpuArray(Y);
gpu_Z = gpuArray.zeros(n,1,type);
tic; for k=1:100, gpu_Z = gpu_a*gpu_X + gpu_Y; wait(gpuDev); end
fprintf(1, 'on GPU (direct): %f sec\n', toc/100);
gpuKern = parallel.gpu.CUDAKernel('vec_axpy.ptx', 'vec_axpy.cu', [type(1), 'axpy']);
gpuKern.ThreadBlockSize = min([numel(X) gpuDev.MaxThreadsPerBlock]); % threads in a block
gpuKern.GridSize = ceil(numel(X)/prod(gpuKern.ThreadBlockSize)); % blocks in the grid
fprintf(1, 'kernel <<<id>>>\n', prod(gpuKern.GridSize), prod(gpuKern.ThreadBlockSize));
gpu_Z = gpuArray.zeros(n,1,type);
tic; for k=1:100, gpu_Z = feval(gpuKern, gpu_a, gpu_X, gpu_Y, gpu_Z, n); wait(gpuDev); end
fprintf(1, 'on GPU (kernel): %f sec\n', toc/100);
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@ceshio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: la fonction axpy de BLAS-1

```

GVec_axpy.m
% on CPU
a = 3.0;
X = ones(n,1,type);
Y = ones(n,1,type);
Z = zeros(n,1,type);
tic; for k=1:100, Z = a*X + Y; end
fprintf(1, 'on CPU (direct): %f sec\n', toc/100);
% on GPU
gpu_a = gpuArray(a);
gpu_X = gpuArray(X);
gpu_Y = gpuArray(Y);
gpu_Z = gpuArray.zeros(n,1,type);
tic; for k=1:100, gpu_Z = gpu_a*gpu_X + gpu_Y; wait(gpuDev); end
fprintf(1, 'on GPU (direct): %f sec\n', toc/100);
gpuKern = parallel.gpu.CUDAKernel('vec_axpy.ptx', 'vec_axpy.cu', [type(1), 'axpy']);
gpuKern.ThreadBlockSize = min([numel(X) gpuDev.MaxThreadsPerBlock]); % threads in a block
gpuKern.GridSize = ceil(numel(X)/prod(gpuKern.ThreadBlockSize)); % blocks in the grid
fprintf(1, 'kernel <<<id>>>\n', prod(gpuKern.GridSize), prod(gpuKern.ThreadBlockSize));
gpu_Z = gpuArray.zeros(n,1,type);
tic; for k=1:100, gpu_Z = feval(gpuKern, gpu_a, gpu_X, gpu_Y, gpu_Z, n); wait(gpuDev); end
fprintf(1, 'on GPU (kernel): %f sec\n', toc/100);
% GPU disconnect
reset(gpuDev);
clear gpuDev
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@ceshio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: la fonction axpy de BLAS-1

```

MATLAB
> ver
-----
MATLAB Version: 8.1.0 (R2013a)
MATLAB License Number: 110415
Operating System: Linux 2.6.32-431.el6.x86_64 #1 SMP Fri Nov 22 03:15:09 UTC 2013 x86_64
Java Version: Java 1.6.0_17-b04 with Sun Microsystems Inc. Java HotSpot(TM)
64-Bit Server VM mixed mode
-----
Toolbox Name          Version      (R2013a)
Simulink              Version 8.1
Communications System Toolbox Version 5.4
Control System Toolbox Version 9.5
Curve Fitting Toolbox Version 3.1.1
DSP System Toolbox    Version 8.4
Global Optimization Toolbox Version 3.2.3
Image Processing Toolbox Version 8.2
Instrument Control Toolbox Version 3.3
MATLAB Compiler       Version 4.18.1
Mapping Toolbox       Version 3.7
Optimization Toolbox  Version 6.3
Parallel Computing Toolbox Version 6.2
RF Toolbox            Version 2.12
Signal Processing Toolbox Version 6.19
Simulink Control Design Version 3.7
Statistics Toolbox    Version 8.2
Symbolic Math Toolbox Version 5.10
System Identification Toolbox Version 9.2
Wavelet Toolbox       Version 4.11
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@ceshio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: la fonction axpy de BLAS-1

```

MATLAB
> gpuDev = gpuDevice(1)
gpuDev =
    CUDADevice with properties:
        Name: 'Tesla K20Xm'
        Index: 1
        ComputeCapability: '3.5'
        SupportDouble: 1
        DriverVersion: 6
        ToolkitVersion: 5
        MaxThreadsPerBlock: 1024
        MaxShmemPerBlock: 49152
        MaxThreadBlocksize: [1024 1024 64]
        MaxGridSize: [2.1475e+09 65535 65535]
        SIMDWidth: 32
        TotalMemory: 6.0393e+09
        FreeMemory: 5.9048e+09
        MultiprocessorCount: 14
        ClockRateKHz: 732000
        ComputeMode: 'Default'
        GPUOverlapsTransfers: 1
        KernelExecutionTimeout: 0
        CanMapHostMemory: 1
        DeviceSupported: 1
        DeviceSelected: 1
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@ceshio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: la fonction axpy de BLAS-1

```

MATLAB
> GVec axpy
axpy with 67108864 single (256 Mbytes)
on CPU (direct): 0.027249 sec
on GPU (direct): 0.008085 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.007988 sec
> GVec axpy
axpy with 67108864 single (256 Mbytes)
on CPU (direct): 0.026802 sec
on GPU (direct): 0.008086 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.007921 sec
> GVec axpy
axpy with 67108864 single (256 Mbytes)
on CPU (direct): 0.028471 sec
on GPU (direct): 0.008098 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.007968 sec
> GVec axpy
axpy with 67108864 single (256 Mbytes)
on CPU (direct): 0.027536 sec
on GPU (direct): 0.008085 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.007921 sec
> GVec axpy
axpy with 67108864 single (256 Mbytes)
on CPU (direct): 0.028810 sec
on GPU (direct): 0.008101 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.007901 sec
    
```

axpy	float
CPU (direct)	0.02699
GPU (direct)	0.00809
GPU (kernel)	0.00794
GPU (cuda/c++)	0.00495

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: la fonction axpy de BLAS-1

```

MATLAB
> gpuKern
gpuKern =
CUDAKernel with properties:
  ThreadBlockSize: [1024 1 1]
  MaxThreadsPerBlock: 1024
  GridSize: [65536 1 1]
  SharedMemorySize: 0
  EntryPoint: '_25axpydPKF50_Pdi'
  MaxNumArguments: 1
  NumArguments: 5
  ArgumentTypes: {'in single scalar' 'in single vector' 'in single vector'
                  'inout single vector' 'in int32 scalar'}
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: la fonction axpy de BLAS-1

```

MATLAB
> GVec axpy
axpy with 67108864 double (512 Mbytes)
on CPU (direct): 0.042894 sec
on GPU (direct): 0.014897 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.014988 sec
> GVec axpy
axpy with 67108864 double (512 Mbytes)
on CPU (direct): 0.042774 sec
on GPU (direct): 0.014892 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.014991 sec
> GVec axpy
axpy with 67108864 double (512 Mbytes)
on CPU (direct): 0.041961 sec
on GPU (direct): 0.014891 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.014991 sec
> GVec axpy
axpy with 67108864 double (512 Mbytes)
on CPU (direct): 0.042258 sec
on GPU (direct): 0.014892 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.015012 sec
> GVec axpy
axpy with 67108864 double (512 Mbytes)
on CPU (direct): 0.043364 sec
on GPU (direct): 0.014885 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.014988 sec
    
```

axpy	float	double
CPU (direct)	0.02699	0.04265
GPU (direct)	0.00809	0.01489
GPU (kernel)	0.00794	0.01499
GPU (cuda/c++)	0.00495	0.00896

MATLAB: direct / kernel = 1
 KERNEL: matlab / cuda = 1.6
 GPU: double / float = 2
 (KERN: 3680/1300=3)

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: la fonction axpy de BLAS-1

```

MATLAB
> gpuKern
gpuKern =
CUDAKernel with properties:
  ThreadBlockSize: [1024 1 1]
  MaxThreadsPerBlock: 1024
  GridSize: [65536 1 1]
  SharedMemorySize: 0
  EntryPoint: '_25axpydPKF50_Pdi'
  MaxNumArguments: 1
  NumArguments: 5
  ArgumentTypes: {'in double scalar' 'in double vector' 'in double vector'
                  'inout double vector' 'in int32 scalar'}
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: une fonction ax2pbxpc

```

vec_ax2pbxpc.cu
#include <cuda.h>
// kernels that execute on the CUDA device
__global__ void dax2pbxpc(const double A, const double B, const double C, const double *X,
                        double *Z, const int N)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx < N) Z[idx] = A*X[idx]*X[idx] + B*X[idx] + C;
}
__global__ void sax2pbxpc(const float A, const float B, const float C, const float *X,
                        float *Z, const int N)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx < N) Z[idx] = A*X[idx]*X[idx] + B*X[idx] + C;
}
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: une fonction ax2pbxpc

```

GVec_ax2pbxpc.cu
#include <cuda.h>
#include <sys/time.h>
#include <cuda.h>
#define DIFFTIMESEC(Tstart,Tend) (((double)(Tend.tv_usec - Tstart.tv_usec)
+ 1E-06*((double)(Tend.tv_nsec - Tstart.tv_nsec)))

// kernels that execute on the CUDA device
// main routine that executes on the host
int main(void)
{
  const int n = 2<<(26-1); // number of elements in arrays
  size_t size = n*sizeof(double); // size of arrays
  double *X_h, *Z_h; // pointers to host arrays
  double *X_d, *Z_d; // pointers to device arrays
  struct timeval tic, toc; // timers

  printf("ax2pbxpc with %d double (%d Mbytes)\n",n,size/(2<<(20-1)));

  // allocate arrays on host
  X_h = (double *)malloc(size);
  Z_h = (double *)malloc(size);
  // allocate arrays on device
  cudaMalloc((void **)X_d,size);
  cudaMalloc((void **)Z_d,size);
  // initialize host arrays and copy them to device
  for (int i=0; i<n; i++) X_h[i] = 2.0;
  cudaMemcpy(X_d,X_h,size, cudaMemcpyHostToDevice);
  // do calculation on device
  // initialize host arrays and copy them to device
  for (int i=0; i<n; i++) Z_h[i] = 2.0;
  cudaMemcpy(Z_d,Z_h,size, cudaMemcpyHostToDevice);
  // do calculation on device
  int blockSize = 1024;
  int gridSize = n/blockSize + (n%blockSize == 0 ? 0 : 1); // threads in a block
  printf("ax2pbxpc with %d double (%d Mbytes)\n",n,gridSize*blockSize);
  gettimeofday(&tic,0);
}
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cesbio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: une fonction ax2pbxpc

```

% GVec_ax2pbxpc.m
% on CPU
a = 3.0;
b = 2.0;
c = 0.5;
X = ones(n,1,type);
Z = zeros(m,1,type);
tic; for k=1:100, Z = a*X.*X + b*X + c; end
fprintf(1,'on CPU (direct): %f sec\n',toc/100);

% on GPU
gpu_a = gpuArray(a);
gpu_b = gpuArray(b);
gpu_c = gpuArray(c);
gpu_X = gpuArray(X);
gpu_Z = gpuArray(zeros(m,1,type));
tic7; for k=1:100, gpu_Z = gpu_a*gpu_X.*gpu_X + gpu_b*gpu_X + gpuDev_c; wait(gpuDev); end
fprintf(1,'on GPU (direct): %f sec\n',toc/100);

gpuKern = parallel_gpu_CUDAKernel('vec_ax2pbxpc.m','vec_ax2pbxpc.m',[type(1),'ax2pbxpc']);
gpuKern.ThreadBlockSize = min([numel(X)/gpuDev_MaxThreadsPerBlock]); % threads in a block
gpuKern.GridSize = ceil(numel(X)/prod(gpuKern.ThreadBlockSize)); % blocks in the grid
fprintf(1,'kernel<<<65536,1024>> on GPU (kernel): %f sec\n',toc/100);

gpu_Z = gpuArray(zeros(m,1,type));
tic7; for k=1:100, gpu_Z = feval(gpuKern,gpu_a,gpu_b,gpu_c,gpu_X,gpu_Z,n); wait(gpuDev); end
fprintf(1,'on GPU (kernel): %f sec\n',toc/100);

% GPU disconnect
reset(gpuDev);
clear gpuDev
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@cesbio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: une fonction ax2pbxpc

```

MATLAB
> ver
MATLAB Version: 8.1.0.604 (R2013a)
MATLAB License Number: 110419
Operating System: Linux 2.6.32-431.el6.x86_64 #1 SMP Fri Nov 22 03:15:09 UTC 2013 x86_64
Java Version: Java 1.6: 217-204 with Sun Microsystems Inc. Java HotSpot(TM)
64-Bit Server VM mixed mode

MATLAB
Simulation Version 8.1 (R2013a)
Communications System Toolbox Version 5.4 (R2013a)
Control System Toolbox Version 9.5 (R2013a)
Curve Fitting Toolbox Version 3.3.1 (R2013a)
DSP System Toolbox Version 8.4 (R2013a)
Global Optimization Toolbox Version 3.2.3 (R2013a)
Image Processing Toolbox Version 8.2 (R2013a)
Instrument Control Toolbox Version 3.3 (R2013a)
MATLAB Compiler Version 4.2.1 (R2013a)
Mapping Toolbox Version 3.7 (R2013a)
Optimization Toolbox Version 6.3 (R2013a)
Parallel Computing Toolbox Version 6.2 (R2013a)
RF Toolbox Version 2.12 (R2013a)
Signal Processing Toolbox Version 6.3 (R2013a)
Simulink Control Design Version 3.7 (R2013a)
Statistics Toolbox Version 8.2 (R2013a)
Symbolic Math Toolbox Version 5.10 (R2013a)
Wavelet Toolbox Version 8.2 (R2013a)
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@cesbio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: une fonction ax2pbxpc

```

MATLAB
> gpuDev = gpuDevice(1)
gpuDev =
    CUDAdevice with properties:
        Index: 1
        Name: 'Tesla K20xm'
        ComputeCapability: '3.5'
        SupportsDouble: 1
        DriversVersion: 5
        ToolkitVersion: 5
        MaxThreadsPerBlock: 1024
        MaxShmemPerBlock: 49152
        MaxThreadBlockSize: [1024 1024 64]
        MaxGridSize: [2.1475e+09 65535 65535]
        SIMDWidth: 32
        TotalMemory: 6.0393e+09
        FreeMemory: 5.9048e+09
        MultiprocessorCount: 14
        ClockRateKHz: 732000
        ComputeMode: 'Default'
        GPUOverlapsTransfers: 1
        KernelExecutionTimeout: 0
        CanMapHostMemory: 1
        DeviceSupported: 1
        DeviceSelected: 1
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@cesbio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: une fonction ax2pbxpc

```

MATLAB
> GVec_ax2pbxpc
ax2pbxpc with 67108864 single (256 Mbytes)
on CPU (direct): 0.045039 sec
on GPU (direct): 0.019349 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.006506 sec
> GVec_ax2pbxpc
ax2pbxpc with 67108864 single (256 Mbytes)
on CPU (direct): 0.042121 sec
on GPU (direct): 0.019357 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.006604 sec
> GVec_ax2pbxpc
ax2pbxpc with 67108864 single (256 Mbytes)
on CPU (direct): 0.047805 sec
on GPU (direct): 0.019352 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.006501 sec
> GVec_ax2pbxpc
ax2pbxpc with 67108864 single (256 Mbytes)
on CPU (direct): 0.040222 sec
on GPU (direct): 0.019359 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.006500 sec
> GVec_ax2pbxpc
ax2pbxpc with 67108864 single (256 Mbytes)
on CPU (direct): 0.041947 sec
on GPU (direct): 0.019341 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.006500 sec
    
```

ax2pbxpc	float
CPU (direct)	0.04329
GPU (direct)	0.01935
GPU (kernel)	0.00652
GPU (cuda/c++)	0.00344

Calculer sur GPU avec MATLAB Eric.Antierieu@cesbio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: une fonction ax2pbxpc

```

MATLAB
> gpuKern
gpuKern =
    CUDAKernel with properties:
        ThreadBlockSize: [1024 1 1]
        MaxThreadsPerBlock: 1024
        GridSize: [65536 1 1]
        ShareMemorySize: 0
        EntryPoint: 'E9ax2pbxpcffEPKPEPL'
        MaxNumLHSArguments: 1
        NumRHSArguments: 6
        ArgumentTypes: {'in single scalar' 'in single scalar' 'in single scalar' 'in int32 scalar' 'in single vector' 'inout single vector' 'in int32 scalar'}
    
```

Calculer sur GPU avec MATLAB Eric.Antierieu@cesbio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: une fonction ax2pbxpc

```

MATLAB
> GVec_ax2pbxpc
ax2pbxpc with 67108864 double (512 Mbytes)
on CPU (direct): 0.062908 sec
on GPU (direct): 0.036049 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.012029 sec
> GVec_ax2pbxpc
ax2pbxpc with 67108864 double (512 Mbytes)
on CPU (direct): 0.061884 sec
on GPU (direct): 0.036039 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.012035 sec
> GVec_ax2pbxpc
ax2pbxpc with 67108864 double (512 Mbytes)
on CPU (direct): 0.063711 sec
on GPU (direct): 0.036036 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.012035 sec
> GVec_ax2pbxpc
ax2pbxpc with 67108864 double (512 Mbytes)
on CPU (direct): 0.062717 sec
on GPU (direct): 0.036028 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.012035 sec
> GVec_ax2pbxpc
ax2pbxpc with 67108864 double (512 Mbytes)
on CPU (direct): 0.062717 sec
on GPU (direct): 0.036028 sec
kernel<<<65536,1024>>
on GPU (kernel): 0.012035 sec
    
```

ax2pbxpc	float	double
CPU (direct)	0.04329	0.06235
GPU (direct)	0.01935	0.03603
GPU (kernel)	0.00652	0.01205
GPU (cuda/c++)	0.00344	0.00599

MATLAB: direct / kernel = 3
 KERNEL: matlab / cuda = 2
 GPU: double / float = 2
 (KERN: 3*100 / (100-3))

Calculer sur GPU avec MATLAB Eric.Antierieu@cesbio.cnes.fr

5 Exécuter des noyaux CUDA

Exemple: une fonction ax2pbxpc

```

MATLAB
> gpuKern
gpuKern =
    CUDAKernel with properties:
    ThreadBlockSize: [1024 1 1]
    MaxThreadsPerBlock: 1024
    GridSize: [6536 1 1]
    SharedMemorySize: 0
    EntryPoints: 'Z9dax2pbxpcoddPRkPdI'
    MaxNumLHSArguments: 1
    NumRHSArguments: 6
    ArgumentTypes: {'in double scalar' 'in double scalar' 'in double scalar'
                    'inout double vector' 'inout double vector' 'in int32 scalar'}
    >
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cnesio.cnes.fr

6 Conclusion

Les étapes

- 1) Identifier et sélectionner un GPU
- 2) Transférer les données vers le GPU via le PCIe: *host* → *device*
- 3) Traiter les données sur le GPU:
 - a) *built-in functions / array operators*
 - b) *custom functions / element-wise operators*
 - c) *user-defined CUDA kernels*
 - d) *user-defined CUDA-enabled MEX files*
 → cuBLAS / cuFFT / cuRAND routines
 → ARRAYFIRE / CULA / MAGMA ... routines
- 3) Transférer les résultats vers le CPU via le PCIe: *device* → *host*

Calculer sur GPU avec MATLAB Eric.Anterrieu@cnesio.cnes.fr

6 Conclusion

Exemple: Mandelbrot

```

GMandelbrot.m
%% https://fr.mathworks.com/help/distcomp/examples/
Niter = 500; // maximum number of iterations
Npxl = 1024; // grid size
xlim = [-0.748766713922161, -0.748766707771757];
ylim = [ 0.123640844894862, 0.123640851045266];

%% CPU
% Setup
tic;
x = linspace(xlim(1),xlim(2),Npxl);
y = linspace(ylim(1),ylim(2),Npxl);
[xGrid,yGrid] = meshgrid(x,y);
z0 = complex(xGrid,yGrid);
count = ones(size(z0));
% Calculate
z = z0;
for n = 0:Niter
    z = z.*z + x0;
    count = count + (abs(z) <= 2);
end
count = log(count);
% Show
CPUtime = toc;
fprintf(1, '%.2f secs (CPU)\n', CPUtime);
figure(2); imagesc(x,y,count); colormap(jet); colorbar('west'); axis('off');
title(sprintf('%.2f secs (CPU)'));
clear xGrid yGrid x y z z0 count
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cnesio.cnes.fr

6 Conclusion

Exemple: Mandelbrot

```

GMandelbrot.m
%% https://fr.mathworks.com/help/distcomp/examples/
Niter = 500; // maximum number of iterations
Npxl = 1024; // grid size
xlim = [-0.748766713922161, -0.748766707771757];
ylim = [ 0.123640844894862, 0.123640851045266];

%% GPU
% Setup
tic;
x = gpuArray.linspace(xlim(1),xlim(2),Npxl);
y = gpuArray.linspace(ylim(1),ylim(2),Npxl);
[xGrid,yGrid] = meshgrid(x,y);
z0 = complex(xGrid,yGrid);
count = gpuArray.ones(size(z0));
% Calculate
z = z0;
for n = 0:Niter
    z = z.*z + x0;
    count = count + (abs(z) <= 2);
end
count = gather(log(count));
% Show
GPUtime = toc;
fprintf(1, '%.2f secs (GPU direct), %s if faster\n', GPUtime, CPUtime/GPUtime);
figure(2); imagesc(x,y,count); colormap(jet); colorbar('west'); axis('off');
title(sprintf('%.2f secs (GPU direct), %s if faster', GPUtime, CPUtime/GPUtime));
clear xGrid yGrid x y z z0 count
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cnesio.cnes.fr

6 Conclusion

Exemple: Mandelbrot

```

GMandelbrot.m
%% https://fr.mathworks.com/help/distcomp/examples/
Niter = 500; // maximum number of iterations
Npxl = 1024; // grid size
xlim = [-0.748766713922161, -0.748766707771757];
ylim = [ 0.123640844894862, 0.123640851045266];

%% GPU
% Setup
tic;
x = gpuArray.linspace(xlim(1),xlim(2),Npxl);
y = gpuArray.linspace(ylim(1),ylim(2),Npxl);
[xGrid,yGrid] = meshgrid(x,y);
% Calculate
count = arrayfun(@MandelbrotKernel, xGrid, yGrid, Niter);
count = gather(count);
% Show
GPUarraytime = toc;
fprintf(1, '%.2f secs (GPU arrayfun), %s if faster\n', GPUarraytime, CPUtime/GPUarraytime);
figure(2); imagesc(x,y,count); colormap(jet); colorbar('west'); axis('off');
title(sprintf('%.2f secs (GPU arrayfun), %s if faster', GPUarraytime, CPUtime/GPUarraytime));
clear xGrid yGrid x y count

function count = MandelbrotKernel(x0,y0,Niter)
% Initialize: z = z0
z0 = complex(x0,y0);
z = z0;
% Loop: until escape
iter = 1;
while (iter <= Niter) && (abs(z) <= 2)
    iter = iter + 1;
    % Update: z = z.*z + x0
    z = z.*z + x0;
end
count = log(iter);
end
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cnesio.cnes.fr

6 Conclusion

Exemple: Mandelbrot

```

GMandelbrot.m
%% https://fr.mathworks.com/help/distcomp/examples/
Niter = 500; // maximum number of iterations
Npxl = 1024; // grid size
xlim = [-0.748766713922161, -0.748766707771757];
ylim = [ 0.123640844894862, 0.123640851045266];

%% GPU
% Setup
tic;
x = gpuArray.linspace(xlim(1),xlim(2),Npxl);
y = gpuArray.linspace(ylim(1),ylim(2),Npxl);
[xGrid,yGrid] = meshgrid(x,y);
% Kernel
gpuKern = parallel.gpu.CUDAkernel('MandelbrotKernel.ptx', 'MandelbrotKernel.cu');
gpuKern.ThreadBlockSize = gpuKern.MaxThreadsPerBlock;
gpuKern.GridSize = ceil(Npxl.^2/gpuKern.MaxThreadsPerBlock);
% Calculate
count = gpuArray.zeros(size(xGrid));
count = feval(gpuKern, xGrid, yGrid, Niter, count, Npxl.^2);
count = gather(count);
% Show
GPUkerneltime = toc;
fprintf(1, '%.2f secs (GPU kernel), %s if faster\n', GPUkerneltime, CPUtime/GPUkerneltime);
figure(4); imagesc(x,y,count); colormap(jet); colorbar('west'); axis('off');
title(sprintf('%.2f secs (GPU kernel), %s if faster', GPUkerneltime, CPUtime/GPUkerneltime));
clear xGrid yGrid x y count gpuKern
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@cnesio.cnes.fr

6 Conclusion

Exemple: Mandelbrot

```

GOMandelbrot.m
%% https://fr.mathworks.com/help/distcomp/examples/
Niter = 500; // maximum number of iterations
Npxl = 1024; // grid size
xlim = [-0.74876713922361, -0.748767077771757];
ylim = [ 0.123640844894862, 0.123640851045266];

%% GPU
tic;
x = gpuArray(1:Npxl);
y = gpuArray(1:Npxl);
[xGrid,yGrid] = meshgrid(x,y);
% kernel
[iter,idx] = blockIdx.* blockDim.x + threadIdx.x;
% kernel
void MandelbrotKernel(const double *Z_re, const double *Z_im,
const int Niter, double *count, const int N2)
{
    double Z_re = *Z_re;
    double Z_im = *Z_im;
    // Initialize: z = z0
    double Z_re = Z_re;
    double Z_im = Z_im;
    // Loop: until escape
    double iter = 0;
    while ((iter <= Niter) && (abs(Z_re*Z_re + Z_im*Z_im) <= 4.0))
        iter++;
    // Update: z = z^2 + z0
    const double tmp = Z_re;
    Z_re = Z_re*Z_re - Z_im*Z_im + Z_re[idx];
    Z_im = 2*tmp*Z_im + Z0_im[idx];
    count[idx] = log(iter);
}
    
```

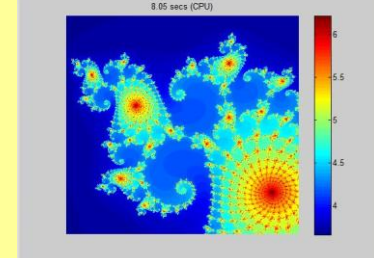
Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

6 Conclusion

Exemple: Mandelbrot

```

MATLAB
GOMandelbrot
8.05 secs (GPU)
0.96 secs (GPU direct), x8.4 faster
0.47 secs (GPU arrayfun), x17.2 faster
0.04 secs (GPU kernel), x202.0 faster
    
```



Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

6 Conclusion

Exemple: RadioSignal

```

RadioSignal.m
function [E,state]=RadioSignal(gpuId,Npxl,Xh,Yh,Tsrc,Gant,FlyingTime,Opkl,Fs,Ns,Q,seed,state)
% RADIOIGNAL returns the electric field [V/m] emitted by a scene with a given
% brightness temperature distribution over the Earth and captured by an antenna.
% The power spectral density of the emission of every pixels of the scene follows
% Planck's law of a black body in thermal equilibrium.
[E,state]=RadioSignal(gpuId,Npxl,Xh,Yh,Tsrc,Gant,FlyingTime,Opkl,Fs,Ns,Q,seed,state)
% gpuId = GPU device (in the range [1,gpuDeviceCount], otherwise CPU is selected)
% Npxl = number of pixels in field of view
% Ns = number of samples to simulate
% Q = number of samples to add to account for flying time delay
% seed = seed for random numbers generator
% state = state of random numbers generator
% E = electric field kept by antenna [V/m]
% state = state of random numbers generator
% See also: FIRPLANKLAW
% (c) Eric Anterrieu (2016)

%% GPU vs. CPU
if (gpuId >= 1)
    gpuDev = gpuDevice(gpuId);
    fprintf(1,'Calculations will be performed on GPU %s %d\n',gpuDev.Name,gpuDev.Index);
else
    fprintf(1,'Calculations will be performed on CPU\n');
end

% no complex arithmetic on GPU, only real arithmetic to save memory and time!
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

6 Conclusion

Exemple: RadioSignal

```

RadioSignal.m
% Planck's law filter in [0 Fs/2]
M = 2048;
Tavg = mean(Tsrc(:));
fprintf(1,'Computing Planck filter at %f K.\n',Tavg); tic;
H = firplanklaw(Tavg,1,[0 Fs/2],1,M);
Hr = real(H);
Hi = imag(H);
if (gpuId >= 1), Hr = gpuArray(Hr); end
if (gpuId >= 1), Hi = gpuArray(Hi); end
clear H;
fprintf(1,'done in %f sec.\n',toc);

% random numbers generator
if isempty(state)
    if (gpuId >= 1), rng(seed,'CMBRecursive'); end
    if (gpuId >= 1), parallel.gpu.rng(seed,'CMBRecursive'); end
else
    if (gpuId == 0), rng(state); end
    if (gpuId >= 1), parallel.gpu.rng(state); end
end

% radio signal in [0 Fs/2]
fprintf(1,'Simulating radio signal over %g sec with %d Planck distributed random\n',
(Ns-1)/Fs,Npxl);
fprintf(1,'sequences of %d samples each out of %d normally distributed samples.\n',
Ns,Ns+Q*M-1);

tic;
t = (0:Ns-1)/Fs;
OP = 1./sqrt(1 - Xh.^2 - Yh.^2);
sigma = sqrt(Tsrc/Tavg).*OP*Opkl; clear OP;
if (gpuId == 0), Er = zeros(1,Ns); E1 = zeros(1,Ns); end
if (gpuId >= 1), Er = gpuArray(zeros(1,Ns)); E1 = gpuArray(zeros(1,Ns)); end
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

6 Conclusion

Exemple: RadioSignal

```

RadioSignal.m
for pxl=1:Npxl
    % normally distributed random sequences
    if (gpuId == 0), N01 = sigma(pxl)*randn(1,Ns+Q*M-1); end
    if (gpuId >= 1), N01 = sigma(pxl)*gpuArray(randn(1,Ns+Q*M-1)); end
    % Planck distributed random sequences
    P01r = conv(N01,Hr,'valid');
    P01i = conv(N01,Hi,'valid');
    clear N01;
    % flying time
    Tfly = FlyingTime(pxl,2) + FlyingTime(pxl,1)*t;
    q = round((t - Tfly)/Fs); clear Tfly;
    P01r = P01r(q+q);
    P01i = P01i(q+q);
    clear q;
    % contribution to radio signal captured by antenna
    Er = Er + real(Gant(pxl))*P01r - imag(Gant(pxl))*P01i;
    E1 = E1 + real(Gant(pxl))*P01i + imag(Gant(pxl))*P01r;
    clear P01r P01i;
    % verbose part
    if rem(pxl,1000) == 0
        fprintf(1,'%d pxl out of %d processed [average %f %f sec/pxl]...\n',pxl,Npxl,toc/pxl);
    end
end
clear Hr Hi pxl t sigma
if (gpuId == 0), E = complex(Er,E1); end
if (gpuId >= 1), E = complex(gather(Er),gather(E1)); end
clear Er E1;
fprintf(1,'done in %f sec.\n',toc);

% random numbers generator
if (gpuId == 0), state = rng(); end
if (gpuId >= 1), state = parallel.gpu.rng(); end

% GPU disconnect
if (gpuId >= 1), reset(gpuDev); clear gpuDev; end
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

6 Conclusion

Exemple: RadioSignal

```

MATLAB
gpuId = 0;
[E,-] = RadioSignal(gpuId,Nearth,Xearth,Yearth,Tearth,Gearth,Tfly,Opkl,Fs,Nd,Q,seed,1);
Computing Planck filter at 186.0 K... done in 0.2 sec.
Radio signal is simulated over 0.02 sec with 22980 Planck distributed random
sequences of 16000000 samples each out of 243002047 normally distributed samples...
1000 pxl out of 22980 processed [average 56.6 sec/pxl]...
2000 pxl out of 22980 processed [average 56.6 sec/pxl]...
3000 pxl out of 22980 processed [average 56.6 sec/pxl]...
4000 pxl out of 22980 processed [average 56.6 sec/pxl]...
5000 pxl out of 22980 processed [average 56.6 sec/pxl]...
6000 pxl out of 22980 processed [average 56.6 sec/pxl]...
7000 pxl out of 22980 processed [average 56.6 sec/pxl]...
8000 pxl out of 22980 processed [average 56.6 sec/pxl]...
9000 pxl out of 22980 processed [average 56.6 sec/pxl]...
10000 pxl out of 22980 processed [average 56.6 sec/pxl]...
11000 pxl out of 22980 processed [average 56.6 sec/pxl]...
12000 pxl out of 22980 processed [average 56.6 sec/pxl]...
13000 pxl out of 22980 processed [average 56.6 sec/pxl]...
14000 pxl out of 22980 processed [average 56.6 sec/pxl]...
15000 pxl out of 22980 processed [average 56.6 sec/pxl]...
16000 pxl out of 22980 processed [average 56.6 sec/pxl]...
17000 pxl out of 22980 processed [average 56.6 sec/pxl]...
18000 pxl out of 22980 processed [average 56.6 sec/pxl]...
19000 pxl out of 22980 processed [average 56.6 sec/pxl]...
20000 pxl out of 22980 processed [average 56.6 sec/pxl]...
21000 pxl out of 22980 processed [average 56.6 sec/pxl]...
22000 pxl out of 22980 processed [average 56.6 sec/pxl]...
done in 1301672.3 sec.
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

6 Conclusion

Exemple: RadioSignal

```

MATLAB
> gpuId = 0;
> [E, I] = RadioSignal(gpuId, Nearth, Xearth, Yearth, Tearth, Gearth, Tfly, Opnl, Fs, Nd, Q, seed, []);
Computing Planck filter at 186.0 K... done in 0.2 sec.
Radio signal is simulated over 0.02 sec with 22980 Planck distributed random
sequences of 16000000 samples each out of 243002047 normally distributed samples...
1000 pxl out of 22980 processed [average 56.6 sec/pxl]...
2000 pxl out of 22980 processed [average 56.6 sec/pxl]...
3000 pxl out of 22980 processed [average 56.6 sec/pxl]...
4000 pxl out of 22980 processed [average 56.6 sec/pxl]...
5000 pxl out of 22980 processed [average 56.6 sec/pxl]...
6000 pxl out of 22980 processed [average 56.6 sec/pxl]...
7000 pxl out of 22980 processed [average 56.6 sec/pxl]...
8000 pxl out of 22980 processed [average 56.6 sec/pxl]...
9000 pxl out of 22980 processed [average 56.6 sec/pxl]...
10000 pxl out of 22980 processed [average 56.6 sec/pxl]...
11000 pxl out of 22980 processed [average 56.6 sec/pxl]...
12000 pxl out of 22980 processed [average 56.6 sec/pxl]...
13000 pxl out of 22980 processed [average 56.6 sec/pxl]...
14000 pxl out of 22980 processed [average 56.6 sec/pxl]...
15000 pxl out of 22980 processed [average 56.6 sec/pxl]...
16000 pxl out of 22980 processed [average 56.6 sec/pxl]...
17000 pxl out of 22980 processed [average 56.6 sec/pxl]...
18000 pxl out of 22980 processed [average 56.6 sec/pxl]...
19000 pxl out of 22980 processed [average 56.6 sec/pxl]...
20000 pxl out of 22980 processed [average 56.6 sec/pxl]...
21000 pxl out of 22980 processed [average 56.6 sec/pxl]...
22000 pxl out of 22980 processed [average 56.6 sec/pxl]...
23000 pxl out of 22980 processed [average 56.6 sec/pxl]...
done in 1301672.3 sec.
    
```

361 heures

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

6 Conclusion

Exemple: RadioSignal

```

MATLAB
> gpuId = 1;
> [E, I] = RadioSignal(gpuId, Nearth, Xearth, Yearth, Tearth, Gearth, Tfly, Opnl, Fs, Nd, Q, seed, []);
Computing Planck filter at 186.0 K... done in 0.2 sec.
Radio signal is simulated over 0.02 sec with 22980 Planck distributed random
sequences of 16000000 samples each out of 243002047 normally distributed samples...
1000 pxl out of 22980 processed [average 8.7 sec/pxl]...
2000 pxl out of 22980 processed [average 8.7 sec/pxl]...
3000 pxl out of 22980 processed [average 8.7 sec/pxl]...
4000 pxl out of 22980 processed [average 8.7 sec/pxl]...
5000 pxl out of 22980 processed [average 8.7 sec/pxl]...
6000 pxl out of 22980 processed [average 8.7 sec/pxl]...
7000 pxl out of 22980 processed [average 8.7 sec/pxl]...
8000 pxl out of 22980 processed [average 8.7 sec/pxl]...
9000 pxl out of 22980 processed [average 8.7 sec/pxl]...
10000 pxl out of 22980 processed [average 8.7 sec/pxl]...
11000 pxl out of 22980 processed [average 8.7 sec/pxl]...
12000 pxl out of 22980 processed [average 8.7 sec/pxl]...
13000 pxl out of 22980 processed [average 8.7 sec/pxl]...
14000 pxl out of 22980 processed [average 8.7 sec/pxl]...
15000 pxl out of 22980 processed [average 8.7 sec/pxl]...
16000 pxl out of 22980 processed [average 8.7 sec/pxl]...
17000 pxl out of 22980 processed [average 8.7 sec/pxl]...
18000 pxl out of 22980 processed [average 8.7 sec/pxl]...
19000 pxl out of 22980 processed [average 8.7 sec/pxl]...
20000 pxl out of 22980 processed [average 8.7 sec/pxl]...
21000 pxl out of 22980 processed [average 8.7 sec/pxl]...
22000 pxl out of 22980 processed [average 8.7 sec/pxl]...
done in 200224.8 sec.
    
```

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

6 Conclusion

Exemple: RadioSignal

```

MATLAB
> gpuId = 1;
> [E, I] = RadioSignal(gpuId, Nearth, Xearth, Yearth, Tearth, Gearth, Tfly, Opnl, Fs, Nd, Q, seed, []);
Computing Planck filter at 186.0 K... done in 0.2 sec.
Radio signal is simulated over 0.02 sec with 22980 Planck distributed random
sequences of 16000000 samples each out of 243002047 normally distributed samples...
1000 pxl out of 22980 processed [average 8.7 sec/pxl]...
2000 pxl out of 22980 processed [average 8.7 sec/pxl]...
3000 pxl out of 22980 processed [average 8.7 sec/pxl]...
4000 pxl out of 22980 processed [average 8.7 sec/pxl]...
5000 pxl out of 22980 processed [average 8.7 sec/pxl]...
6000 pxl out of 22980 processed [average 8.7 sec/pxl]...
7000 pxl out of 22980 processed [average 8.7 sec/pxl]...
8000 pxl out of 22980 processed [average 8.7 sec/pxl]...
9000 pxl out of 22980 processed [average 8.7 sec/pxl]...
10000 pxl out of 22980 processed [average 8.7 sec/pxl]...
11000 pxl out of 22980 processed [average 8.7 sec/pxl]...
12000 pxl out of 22980 processed [average 8.7 sec/pxl]...
13000 pxl out of 22980 processed [average 8.7 sec/pxl]...
14000 pxl out of 22980 processed [average 8.7 sec/pxl]...
15000 pxl out of 22980 processed [average 8.7 sec/pxl]...
16000 pxl out of 22980 processed [average 8.7 sec/pxl]...
17000 pxl out of 22980 processed [average 8.7 sec/pxl]...
18000 pxl out of 22980 processed [average 8.7 sec/pxl]...
19000 pxl out of 22980 processed [average 8.7 sec/pxl]...
20000 pxl out of 22980 processed [average 8.7 sec/pxl]...
21000 pxl out of 22980 processed [average 8.7 sec/pxl]...
22000 pxl out of 22980 processed [average 8.7 sec/pxl]...
done in 200224.8 sec.
    
```

56 heures (+ 6.5)

Calculer sur GPU avec MATLAB Eric.Anterrieu@ceshio.cnes.fr

Calculer sur GPU avec MATLAB



Eric ANTERRIEU
*Observatoire Midi-Pyrénées
 Centre d'Etudes Spatiales de la Biosphère
 Equipe Systèmes d'Observation*

CESBIO — UMR5126






Eric.Anterrieu@ceshio.cnes.fr