Golang et Programmation GPU

Plan

1. Introduction à GO
    a. Histoire
    b. Bases du langage
    c. GoRoutines et Channels
2. GPU
    a. Architecture et fonctionnement
    b. GPU vs CPU
3. GPU et GO
    a. Librairies
    b. GoRoutines et GPU

Introduction à Go

« Ils ne sont pas capables de comprendre un langage brillant, mais nous voulons les amener à réaliser de bons programmes. Ainsi, le langage que nous leur donnons doit être facile à comprendre et facile à adopter »

Rob Pike

Source : Wikipedia

Langage compilé impératif, programmation procédurale

Développé par Google, concept initial par Griesemer, Pike et Thompson.

Première version en 2012/Dernière version 1.11.5 en 2019

Support pour les systèmes de communication en réseau, la concurrence et le parallélisme.

Inspiré du langage C et du Pascal

La conception du langage lui permet d’atteindre une rapidité élevée, équivalentes ou supérieur à celles des langages de même catégorie.

Write Less, Do More : Écrivez-moins, Faites-en plus

Bases du langage
Typage, variables et fonctions

Typage fort, types prédéfinis byte, int, float, bool, string(immutable) ...etc

Structures de plus haut niveau Slices, List, Map ...etc

```
i := 10          // déclaration + affectation (pas besoin de spécifier le type)

func plus(a, b int) (int, error)

_ , error := plus(a, b)
```

Structures de contrôle

Une seule syntaxe de boucle: Le for

for i:=0; i < 10; i++ for i < 10

if <déclaration/affectation> {...}

else {...}

switch <déclaration/affectation> { // switch sur les types possibles

case value: do something} // pas besoin de break

defer reporte l'exécution d'une fonction jusqu'à ce que la fonction qui la contient retourne.

Structure et prog “orientée objet”

Pas de concept de « classe » mais possibilité d’écriture de code dans un style orienté-objet.

```
type donnee struct {
    a    int
    b    int
}

func (d *donnee) total() int {
    return d.a * d.b
}
```

Réflexivité

Go permet à un programme d’examiner sa propre structure à travers des types :

```
type T struct {
    A int
    B string
}
t := T{23, "kikoulol"}
s := reflect.ValueOf(&t).Elem()
typeOfT := s.Type()
```

```
for i := 0; i < s.NumField(); i++ {
    f := s.Field(i)
    fmt.Printf("%d: %s %s = %v\n", i,
        typeOfT.Field(i).Name, f.Type(), f.Interface())
}
```

```
0: A int = 23
1: B string = kikoulol
```

GoRoutines et Channels

Une goroutine est un thread d'exécution léger :

go f ()

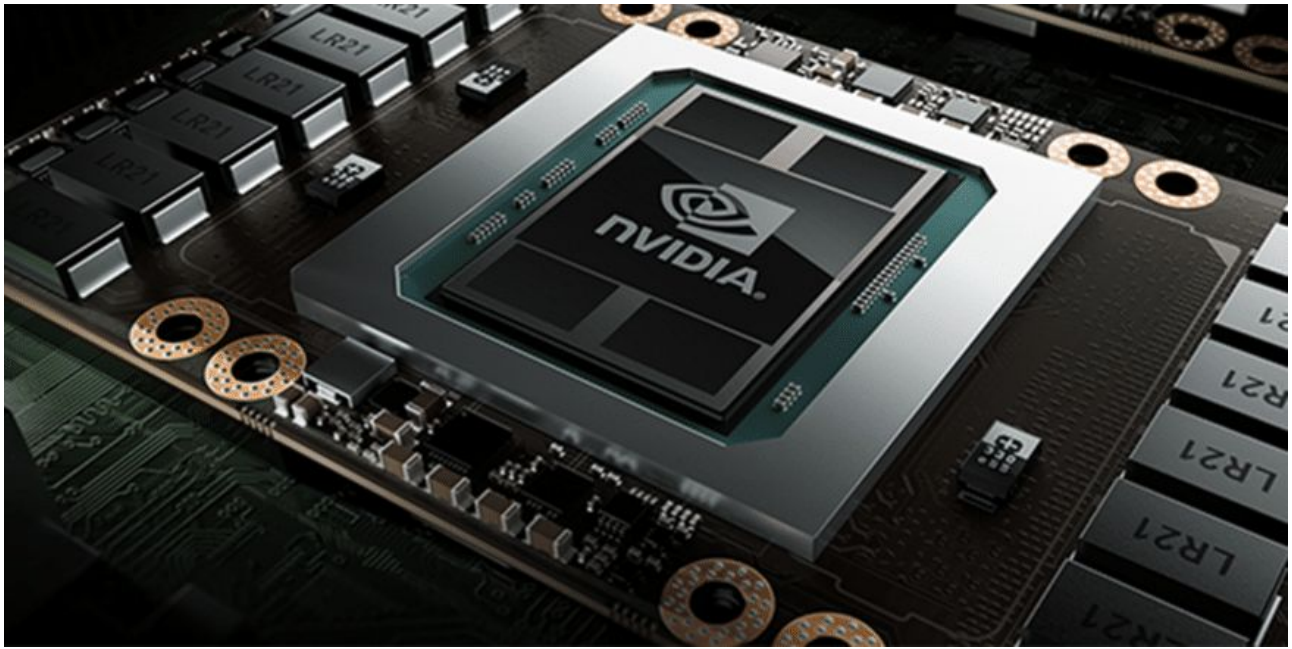
Cette nouvelle goroutine va s'exécuter de manière concurrente avec la fonction appelée.

```
messages := make(chan string)

func worker(done chan bool) {
    fmt.Print("working...")
    time.Sleep(time.Second)
    fmt.Println("done")
    done <- true
}

func main() {
    done := make(chan bool, 1)
    go worker(done)
    <-done
}
```

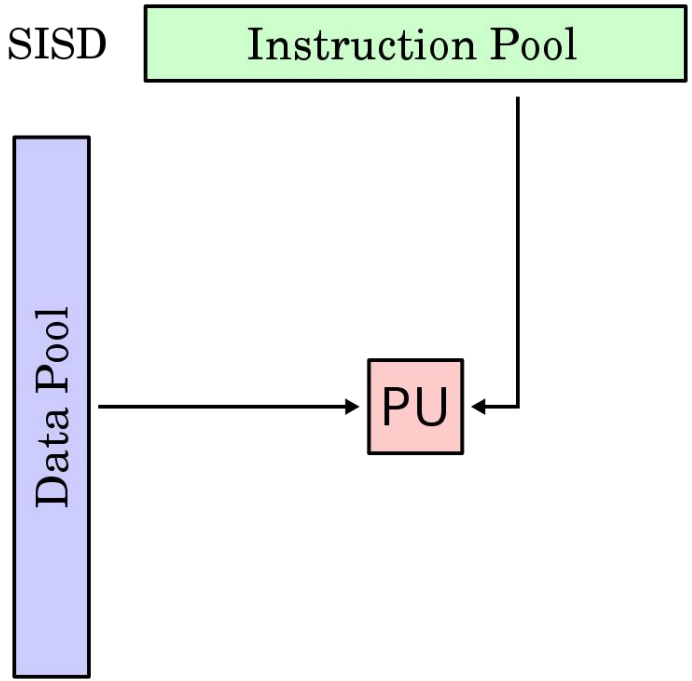
GPU
Graphics processing unit



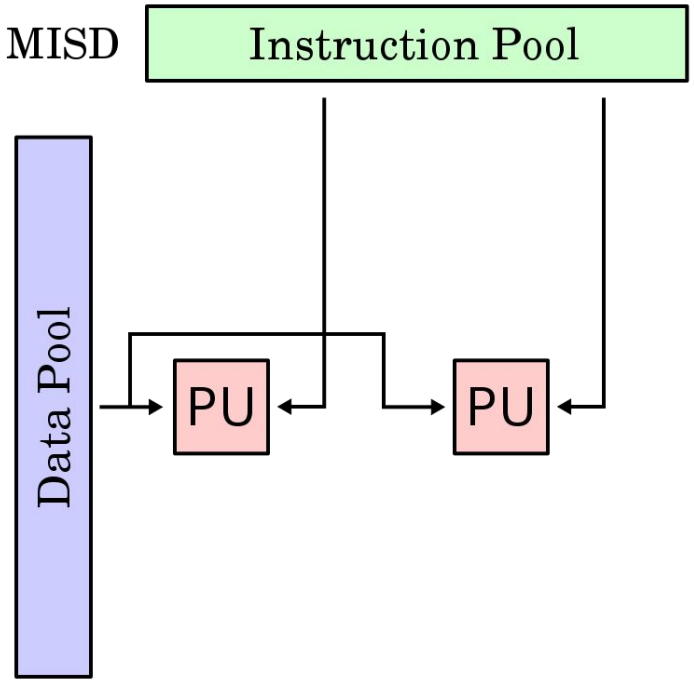
Taxonomie de Flynn
Michael J. Flynn en 1966

	SIMPLE INSTRUCTIONS	MULTIPLE INSTRUCTIONS
SIMPLE DATA	SISD	MISD
MULTIPLE DATA	SIMD	MIMD

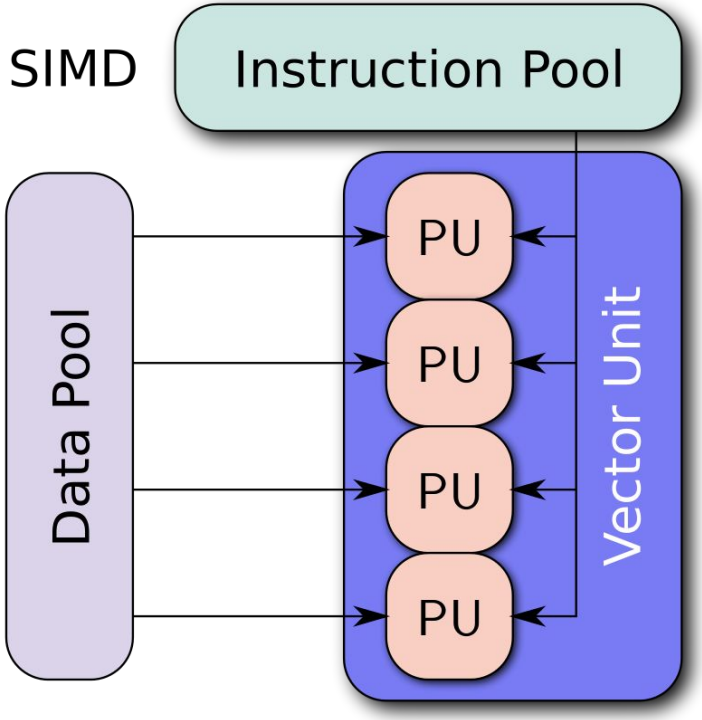
SISD



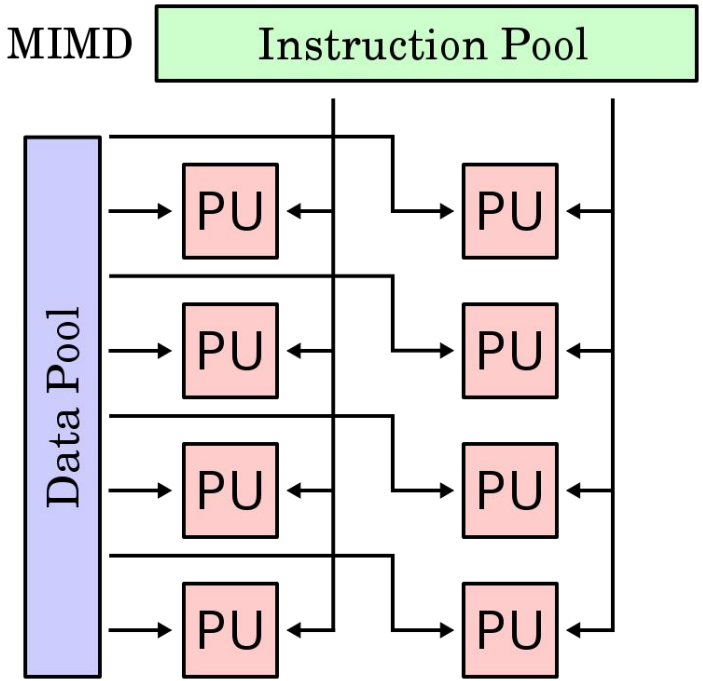
MISD



SIMD



MIMD

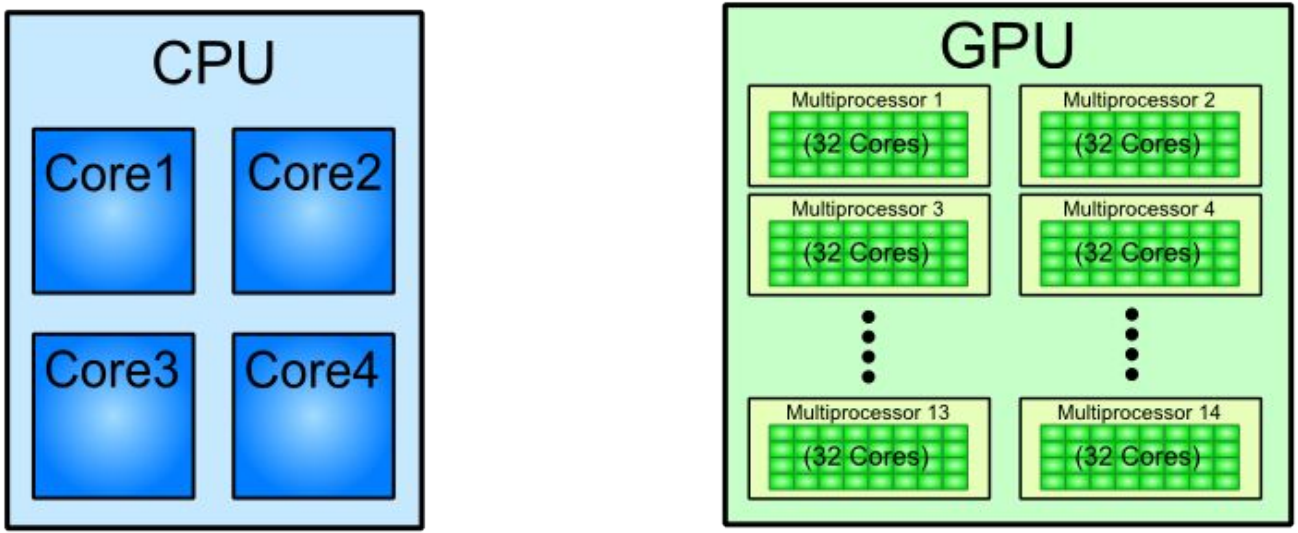


Architecture GPU

- Un grand nombre d’unités de calcul
- Graphics Pipeline
- Utilisation de caches

CPU vs GPU

CPU/GPU Architecture Comparison



Exemple
opération sur un vecteur

Supposons A un vecteur de 100 d'éléments, que l'on souhaite multiplier par 2

CPU	GPU
for i:=0 to 99 A[i] = 2 * A[i]	A = 2 * A

GPGPU: General purpose graphics processing units

GPUs pour la programmation à but plus général que le traitements d'éléments graphiques.

- Recherche
- Tri
- Algorithmes fortement parallélisable

Golang et programmation GPU

Peu répandue mais avec un intérêt certain

Le Go est un langage qui se veut être léger et rapide à la fois avec une aisance d'apprentissage. Ces caractéristiques sont visibles dans le volume de code des projets Go en comparaison de ce qui se fait en C++ par exemple.

Version Go (GPU + GUI): 11.000 lignes

Version C++ (sans code GPU et avec GUI): 100.000 lignes

Version C++ (avec GPU et sans GUI): 30.000 lignes

(GPU-accelerated micromagnetic simulation program)

Pourquoi le Go en GPU ?

Go présente plusieurs avantages tels que :

- Une gestion élégante de la concurrence
- Considéré comme Memory-Safe.. ou au moins plus que ne le sont C et C++..
- On peut appeler du code C !

Et des désavantages :

- Plus lent que du code C optimisé
- Des librairies manquantes (ex: matrices..)

Comment ?

Deux approches sont disponibles en Go.

1. La plus easy-to-use sans doute: Écrire du code CUDA, utiliser la commande cuda2go pour générer le code en Go.
2. Ecrire le code en Go et appeler du code C grace a “cgo”.

Hello World !

```go
package main

import (
    "fmt"

    "github.com/mumax/3/cuda/cu"
)

func main() {
    name := cu.Device(0).Name()
    fmt.Printf("Hello !\n your GPU's name is %v\n", name)
}
```

C Bindings

```go
package main
//#include <cuda.h>
//#cgo LDFLAGS: -lcuda
import "C"
import "fmt"

func Random() int {
    return int(C.random())
}

func Seed(i int) {
    C.srandom(C.uint(i))
}

func main() {
    Seed(100)
    fmt.Println("Hello, your GPU is:", Random())
    buf := C.CString(string(make([]byte, 256)))
    fmt.Println(buf)
    C.cuDeviceGetName(buf, 256, C.CUdevice(0))
    fmt.Println("Hello, your GPU is:", C.GoString(buf))
}
```

```
#include "../common/book.h"
#define N 10

int main( void ) {
    int a[N], b[N], c[N];
    int *dev_a, *dev_b, *dev_c;
    // allocate the memory on the GPU
    HANDLE_ERROR( cudaMalloc( (void**)&dev_a, N * sizeof(int) ) );
    HANDLE_ERROR( cudaMalloc( (void**)&dev_b, N * sizeof(int) ) );
    HANDLE_ERROR( cudaMalloc( (void**)&dev_c, N * sizeof(int) ) );

    // fill the arrays 'a' and 'b' on the CPU
    for (int i=0; i<N; i++) {
        a[i] = -i;
        b[i] = i * i;
    }
    // copy the arrays 'a' and 'b' to the GPU
    HANDLE_ERROR( cudaMemcpy( dev_a, a, N * sizeof(int), cudaMemcpyHostToDevice ) );
    HANDLE_ERROR( cudaMemcpy( dev_b, b, N * sizeof(int), cudaMemcpyHostToDevice ) );
    add<<<N,1>>>( dev_a, dev_b, dev_c ); // copy the array 'c' back from the GPU to the CPU
    HANDLE_ERROR( cudaMemcpy( c, dev_c, N * sizeof(int), cudaMemcpyDeviceToHost ) );
```

```
package main

import "github.com/mumax/3/cuda"

func main(){

    N := 3
    a := cuda.NewSlice(N)
    b := cuda.NewSlice(N)
    c := cuda.NewSlice(N)
    defer a.Free()
    defer b.Free()
    defer c.Free()

    a.CopyHtoD([]float32{0, -1, -2})
    b.CopyHtoD([]float32{0,  1,  4})
```

Petit exercice

```cpp
#include <iostream>
#include <math.h>


// Kernel function, will be run on the GPU
__global__ void add(int n, float *x, float *y)
{
  int index = threadIdx.x;
  int stride = blockDim.x;
  for (int i = index; i < n; i += stride)
      y[i] = x[i] + y[i];
}

int main(void)
{
  int N = 1<<20;
  float *x, *y;

  // Why do we use this functions ?
  cudaMallocManaged(&x, N*sizeof(float));
  cudaMallocManaged(&y, N*sizeof(float));

  // initialize x and y arrays on the host
  for (int i = 0; i < N; i++) {
    x[i] = 1.0f;
    y[i] = 2.0f;
  }


  add<<<1, 256>>>(N, x, y);

  // Free memory
  cudaFree(x);
  cudaFree(y);

  return 0;

```

Sources

An introduction to GPU Programming with CUDA:
https://www.youtube.com/watch?v=1cHx1baKqq0

Scientific GPU Computing with Go:

https://archive.fosdem.org/2014/schedule/event/hpc_devroom_go/attachments/slides/486/export/events/attachments/hpc_devroom_go/slides/486/FOSDEM14_HPC_devroom_14_GoCUDA.pdf

[..]