Programmation parallèle

Sujet pedagogique

Augustin Thiercelin

EPITA

2 juillet 2020

Table des matières

Introduction

Parallelisme : Traiter des données de facon simultané.

Precisions sur les termes

- Programmation Asynchrone
  Requêtes IO non bloquantes

- Programmation Parallèle
  Facon de programmer pour amener du parallelisme

- Concurrence
  Partage de ressources entre plusieurs systèmes

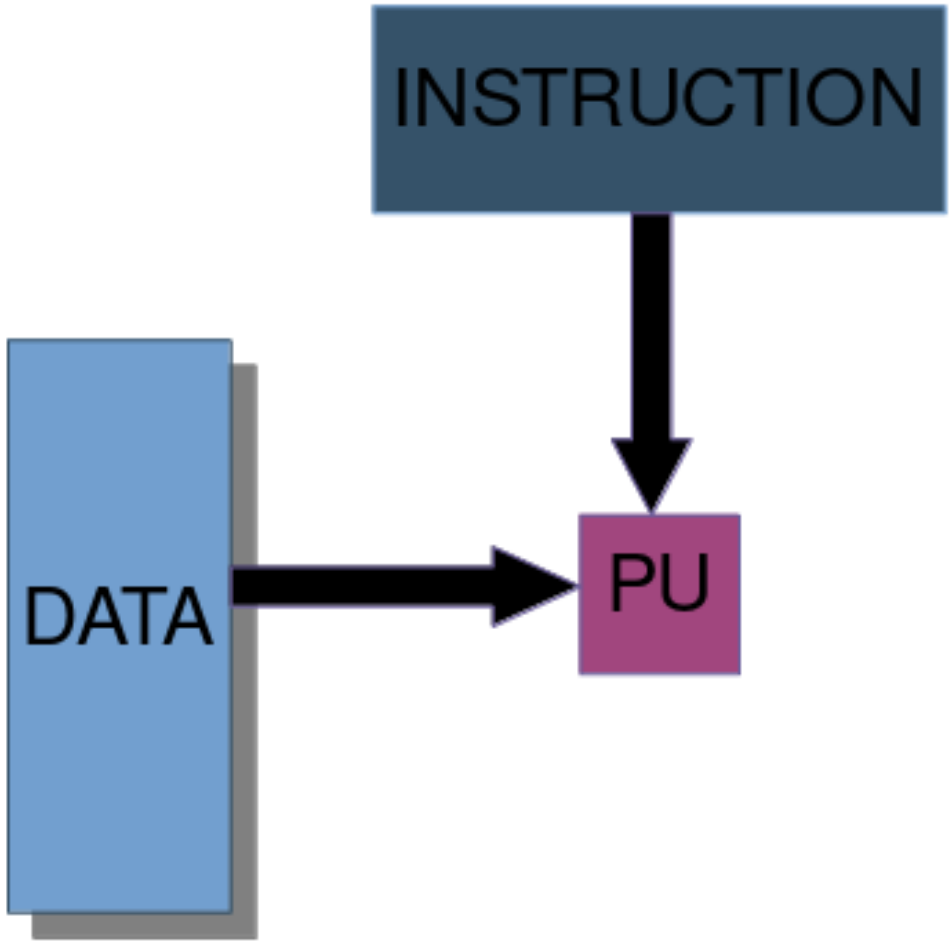
La taxonomie de Flynn

Classification des types d'architecture (notamment processeurs).

	Single Instruction	**Many instructions**
Single data	SISD	MISD
Multiple data	SIMD	MIMD

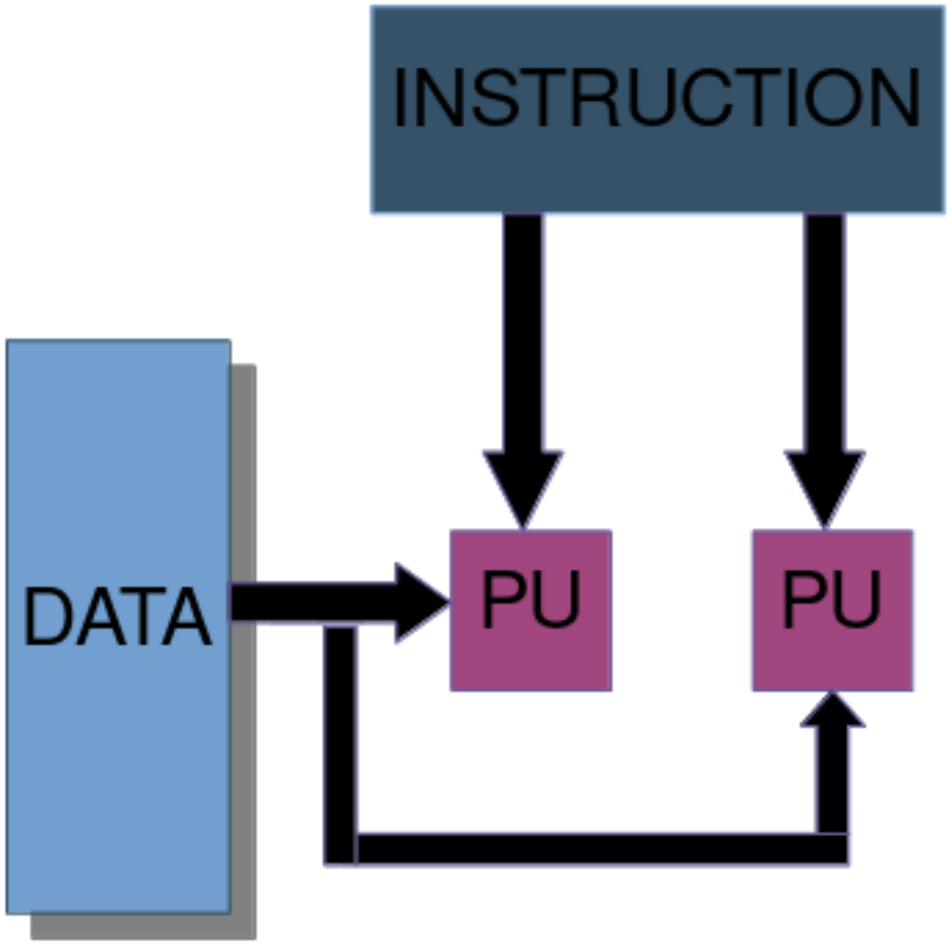
SISD

Pas de parallelisme, sequentiel



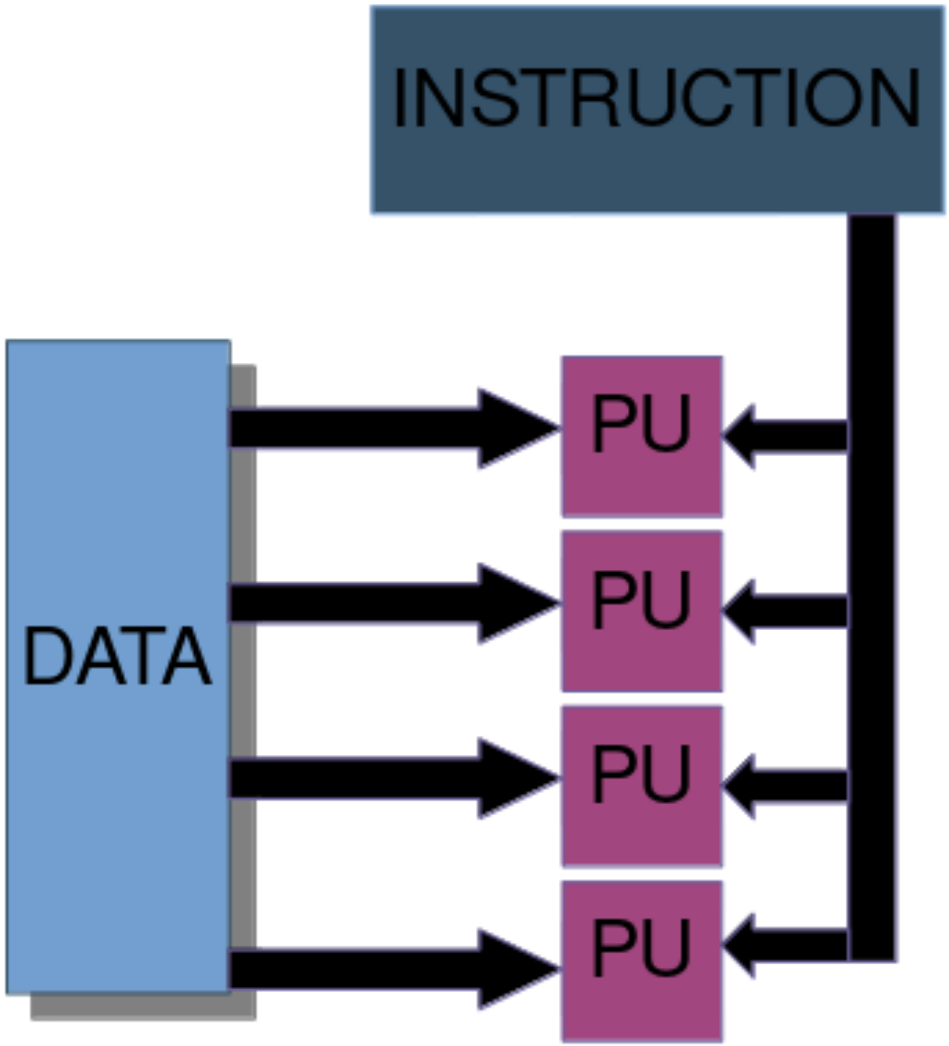
MISD

Peu utilisé, usage pour la détection d'erreur notamment.



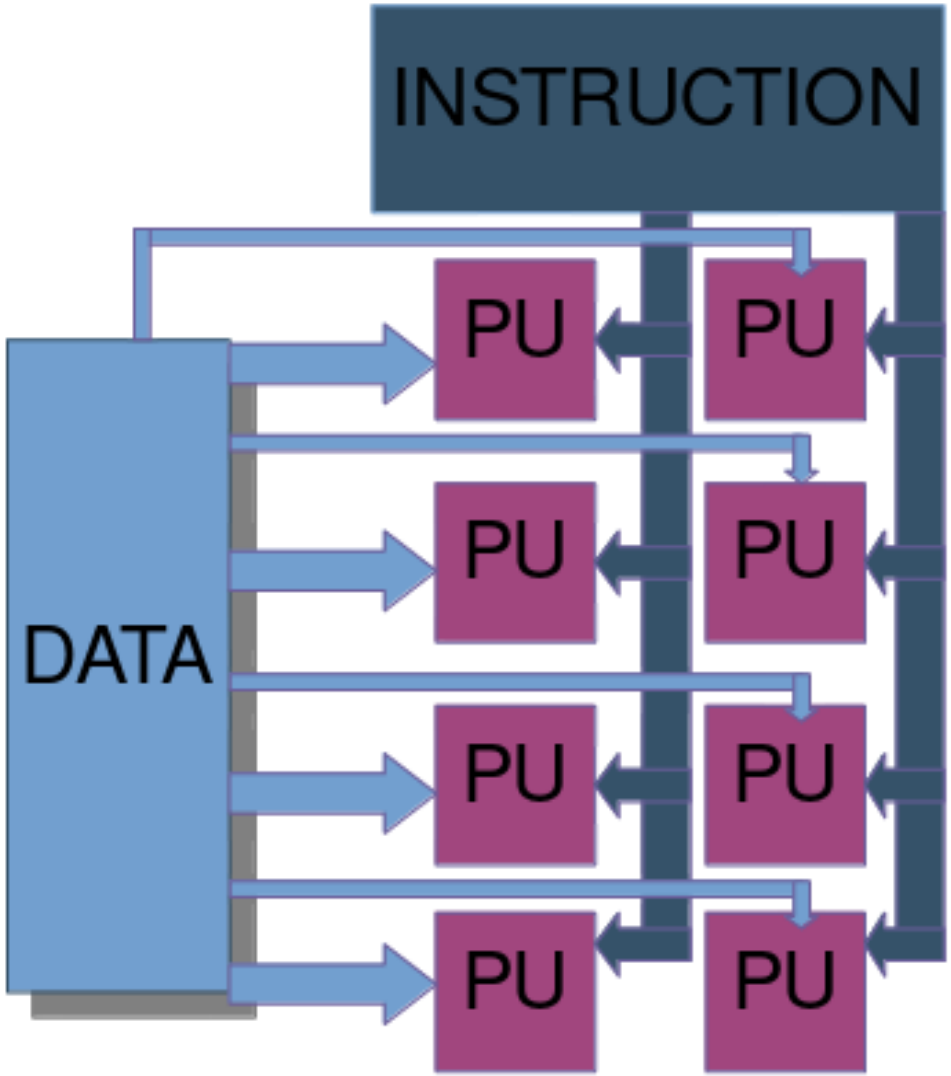
SIMD

Permet la vectorisation



MIMD

Permet le multithreading



Les differents types de parallelismes

- Parallelisme des instructions
- Vectorisation
- Multithreading

Parallelisme des instructions

Une instruction CPU se decompose en plusieurs etapes

Exemple pour les processeurs RISC :

- instruction fetch (IF)
- instruction decode (ID)
- execute (EX)
- memory access (MEM)
- Register write back (WB)

Execution sequentiel

Instruction 1	IF	ID	EX	MEM	WB					
Instruction 2						IF	ID	EX	MEM	WB
CPU Cycle	1	2	3	4	5	6	7	8	9	10

Pipelining

Instruction 1	IF	ID	EX	MEM	WB	
Instruction 2		IF	ID	EX	MEM	WB
CPU Cycle	1	2	3	4	5	6

Hazard

Pseudo code d'instructions

1. **R2** = R5 + R3
2. R4 = **R2** + R3

Instruction 1	IF	ID	EX	MEM	WB	
Instruction 2		IF	ID	EX	MEM	WB
CPU Cycle	1	2	3	4	5	6

Bulle (Pipeline stall)

Instruction 1	IF	ID	EX	MEM	WB				
Instruction 2					IF	ID	EX	MEM	WB
CPU Cycle	1	2	3	4	5	6	7	8	9

Vectorisation

- Architecture SIMD
- Instructions speciales du processeur
- Effectue une operation pour tout un block de donnée
- -ftree-vectorize (enabled with -O3)

OpenMP

Macro de la bibliotheque OpenMD pour aider a la vectorisation

```
1 float * a = ...;
2 float * b = ...;
3 float s;
4 ...
5 #pragma omp simd
6 for (int i = 0; i < N; i++) {
7   a[i] += s * b[i];
8 }
```

Types vecteur

Directement definir les types en temps que vecteurs

```
1  typedef float Vec8f __attribute__ ((vector_size (32)));
2  typedef int Vec8i __attribute__ ((vector_size (32)));
3
4  Vec8i kernel(Vec8f ax, Vec8f ay) {
5    Vec8f x{0};
6    Vec8f y{0};
7    for (int n = 1; n <= 100; n++) {
8      Vec8f newx = x*x - y*y + ax;
9      Vec8f newy = 2*x*y + ay;
10     Vec8i cmpmask = (4 < newx*newx + newy*newy);
11   }
12 }
```

Multithreading

Un thread est un peu similaire a un process mais avec plusieurs differences.

- Les threads sont liés a un process parent
- Les threads sont plus légers
- Les threads partagent la meme zone memoire

Thread

Un exemple d'utilisation en C++

```
1
2  #include <thread>
3  void foo()
4  {
5    // Calcul
6  }
7
8  void bar(int x)
9  {
10   // Calcul
11 }
12
13 int main()
14 {
15   std::thread first (foo);
16   std::thread second (bar,0);
17
18   // Les threads tournent de maniere parallele
19
20   first.join();
21   second.join();
22
23   return 0;
24 }
```

Bibliotheque TBB

Un exemple d'utilisation en C++

```
1   #include "tbb/task_group.h"
2
3   using namespace tbb;
4
5   int Fib(int n) {
6       if( n<2 ) {
7           return n;
8       } else {
9           int x, y;
10          task_group g;
11          g.run([&]{x=Fib(n-1);});
12          g.run([&]{y=Fib(n-2);});
13          g.wait();
14          return x+y;
15      }
16  }
```

Concurrence

- Data races
- Race conditions
- Solutions

Data races

- Deux threads accedent de facon concurrente au meme espace memoire.
- Au moins une des deux actions est une ecriture.
- Au moins une des deux n'est pas synchronisé avec les autres threads.

```
1  #include <thread>
2
3  void function(int& value) {
4      value = 20 + value;
5  }
6
7  int main()
8  {
9      int value = 10;
10     while (1) {
11         std::thread t1(function, std::ref(value));
12         std::thread t2(function, std::ref(value));
13
14         t1.join();
15         t2.join();
16     }
17
18     return 42;
19 }
```

Race conditions

```
1  #include <thread>
2
3  void function(int& value) {
4    if (value == 10) {
5      // Changement de thread ici
6      value = 0;
7    }
8    else {
9      // Changement de thread ici
10     value = 20;
11   }
12 }
13
14 int main()
15 {
16     int value = 10;
17     while (1) {
18         std::thread t1(function, std::ref(value));
19         std::thread t2(function, std::ref(value));
20
21         t1.join();
22         t2.join();
23     }
24
25     return 42;
26 }
```

Solutions

- Atomic
- Mutex
- Lock
- Memory Fences

References

- https://www.lrde.epita.fr/~carlinet/cours/parallel/
- https://dev.to/thibmaek/explain-coroutines-like-im-five-2d9
- http://sametmax.com/la-difference-entre-la-programmation-asynchrone-parallele
- https://www.geeksforgeeks.org/computer-architecture-flynns-taxonomy/
- https://en.wikipedia.org/wiki/Parallel_computing
- https://en.wikipedia.org/wiki/Automatic_vectorization
- https://en.wikipedia.org/wiki/Hazard_(computer_architecture)#Eliminating_hazards
- https://en.wikipedia.org/wiki/Pipeline_stall