

# High Performance Combinatorics

**Florent Hivert**

Mél : Florent.Hivert@lri.fr

Adresse universelle : <http://www.lri.fr/~hivert>

# Objectifs

Montrer deux technologies qui permettent de rendre les calculs beaucoup plus efficaces :

- Instructions vectorielles entières pour les petits tableaux de nombres
- Parcours d'arbre récursif parallèle avec Cilk++

# Objectifs

Montrer deux technologies qui permettent de rendre les calculs beaucoup plus efficaces :

- Instructions vectorielles entières pour les petits tableaux de nombres
- Parcours d'arbre récursif parallèle avec Cilk++

# Objectifs

Montrer deux technologies qui permettent de rendre les calculs beaucoup plus efficaces :

- Instructions vectorielles entières pour les petits tableaux de nombres
- Parcours d'arbre récursif parallèle avec Cilk++

# Avertissement !!!

Même si l'on obtient des gains très importants (x50, x500...), rien de tout cela n'est de la **Bonne Informatique**...

- Technologie expérimentale, pas mure, bugs dans les compilateurs...
- Pas portable, y compris d'une version à l'autre d'un même compilateur...
- Tentatives en cours de normaliser les syntaxes...

## Avertissement !!!

Même si l'on obtient des gains très importants (x50, x500...), rien de tout cela n'est de la **Bonne Informatique**...

- Technologie expérimentale, pas mure, bugs dans les compilateurs. . .
- Pas portable, y compris d'une version à l'autre d'un même compilateur. . .
- Tentatives en cours de normaliser les syntaxes...

## Avertissement !!!

Même si l'on obtient des gains très importants (x50, x500...), rien de tout cela n'est de la **Bonne Informatique**...

- Technologie expérimentale, pas mure, bugs dans les compilateurs...
- Pas portable, y compris d'une version à l'autre d'un même compilateur...
- Tentatives en cours de normaliser les syntaxes...

## Que retenir ?

### Retenir

#### *Les principes de parallélisation*

- *Vectorisation automatique ou manuelle des objets combinatoires*
- *Parallélisation d'une exploration récursive*

La syntaxe n'est **PAS** stable.



## Instruction vectorielles entières

Registre : `epi8`, `eui8` : 128 bits = 16 octets

Encore plus : `AVX`, `AVX2`, `AVX512`

- Opérations Arithmetiques/logiques : `and`, `or`, `add`, `sub`, `min`, `max`, `abs`, `cmp`
- Recherche de bits : `popcount`, `bfsd`

Encore plus intéressant pour nous :

- Manipulation de tableau : `blend`, `broadcast`, `shuffle`
- Comparaison de chaînes de caractères : `cmpistr` (`lex`, `find`).

Manipulations très efficaces !

## Instruction vectorielles entières

Registre : `epi8`, `eui8` : 128 bits = 16 octets

Encore plus : `AVX`, `AVX2`, `AVX512`

- Opérations Arithmetiques/logiques : `and`, `or`, `add`, `sub`, `min`, `max`, `abs`, `cmp`
- Recherche de bits : `popcount`, `bfsd`

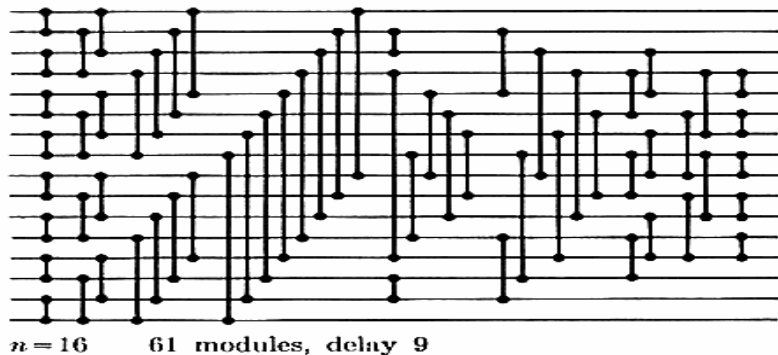
Encore plus intéressant pour nous :

- Manipulation de tableau : `blend`, `broadcast`, `shuffle`
- Comparaison de chaînes de caractères : `cmpistr` (`lex`, `find`).

**Manipulations très efficaces !**

## Exemple : les réseaux de tris

Knuth AoCP3 Fig. 51 p. 229 :



*// Sorting network Knuth AoCP3 Fig. 51 p 229.*

```
static const array<perm, 9> rounds = {{  
    { 1, 0, 3, 2, 5, 4, 7, 6, 9, 8,11,10,13,12,15,14},  
    { 2, 3, 0, 1, 6, 7, 4, 5,10,11, 8, 9,14,15,12,13},  
    ...  
}};
```

```
perm sort(perm a) {  
    for (perm round : rounds) {  
        perm minab, maxab, mask;  
        perm b = _mm_shuffle_epi8(a, round);  
        mask = _mm_cmplt_epi8(round, permid);  
        minab = _mm_min_epi8(a, b);  
        maxab = _mm_max_epi8(a, b);  
        a = _mm_blendv_epi8(minab, maxab, mask);  
    }  
    return a;  
}
```

## Exemple 2 : les cycles dans les permutations

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 6 & 9 & 7 & 8 & 2 & 4 & 3 & 5 \end{pmatrix} = (1)(2, 6)(3, 9, 5, 8, 3)(4, 7)$$

$O(n)$  algorithme utilisant  $O(n)$  memoire.

```
uint8_t nb_cycles_ref(Perm16 p) {  
    Vect16 v {};  
    int i, j, c = 0;  
    for (i = 0; i < 16; i++) {  
        if (v[i] == 0) {  
            for (j=i; v[j] == 0; j = p[j]) v[j] = 1;  
            c++;  
        }  
    }  
    return c;  
}
```

## Exemple 2 : les cycles dans les permutations

$O(\log_2(n))$  algorithme utilisant  $O(n)$  memoire.

Idée : propager le minimum le long des cycles :

[illegible]

# Parallelisation avec Cilk++

## Retenir

- *Programmation parallèle multifils en mémoire partagé.*
- *Extension du C et C++*
- *Ne pas écrire le parallélisme, mais l'exposer quand il est possible.*
- *Modèle Fork-Join :*

*cilk\_spawn*

*cilk\_sync*

# Parallelisation avec Cilk++

## Retenir

- *Programmation parallèle multifils en mémoire partagé.*
- *Extension du C et C++*
- *Ne pas écrire le parallélisme, mais l'exposer quand il est possible.*
- *Modèle Fork-Join :*

*cilk\_spawn*

*cilk\_sync*



# Parallelisation avec Cilk++

## Retenir

- *Programmation parallèle multifils en mémoire partagé.*
- *Extension du C et C++*
- *Ne pas écrire le parallélisme, mais l'exposer quand il est possible.*
- *Modèle Fork-Join :*

*cilk\_spawn*

*cilk\_sync*

# Parallelisation avec Cilk++

## Retenir

- *Programmation parallèle multifils en mémoire partagé.*
- *Extension du C et C++*
- *Ne pas écrire le parallélisme, mais l'exposer quand il est possible.*
- *Modèle Fork-Join :*

*cilk\_spawn            cilk\_sync*

# Fibonacci

```
int fib(int n) {  
    if (n < 2) {  
        return n;  
    }  
    else {  
        int x, y;  
        x = cilk_spawn fib(n - 1);  
        y = cilk_spawn fib(n - 2);  
        sync;  
        return x + y;  
    }  
}
```