

Introduction à la recherche

[Systèmes Numériques]

Récherche numérique

Bertrand LE GAL

{prenom.nom}@ims-bordeaux.fr

Laboratoire IMS - UMR CNRS 5218
Institut Polytechnique de Bordeaux
Université de Bordeaux
France



ENSEIRB
MATMECA
BORDEAUX

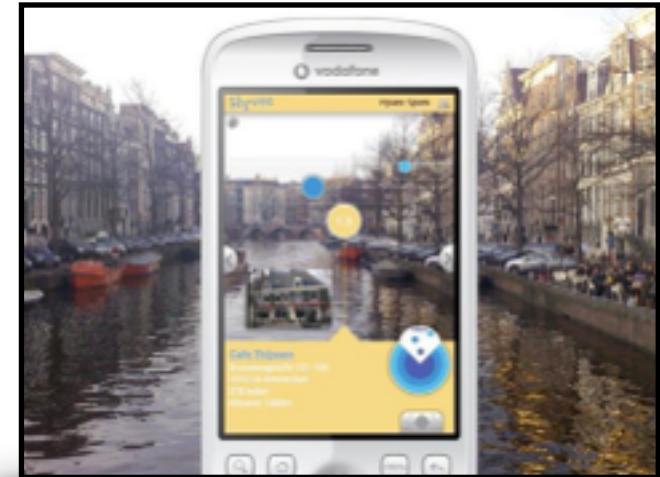


Evolution des domaines applicatifs - la téléphonie mobile

The next evolution !



Augmented Reality (Ar)



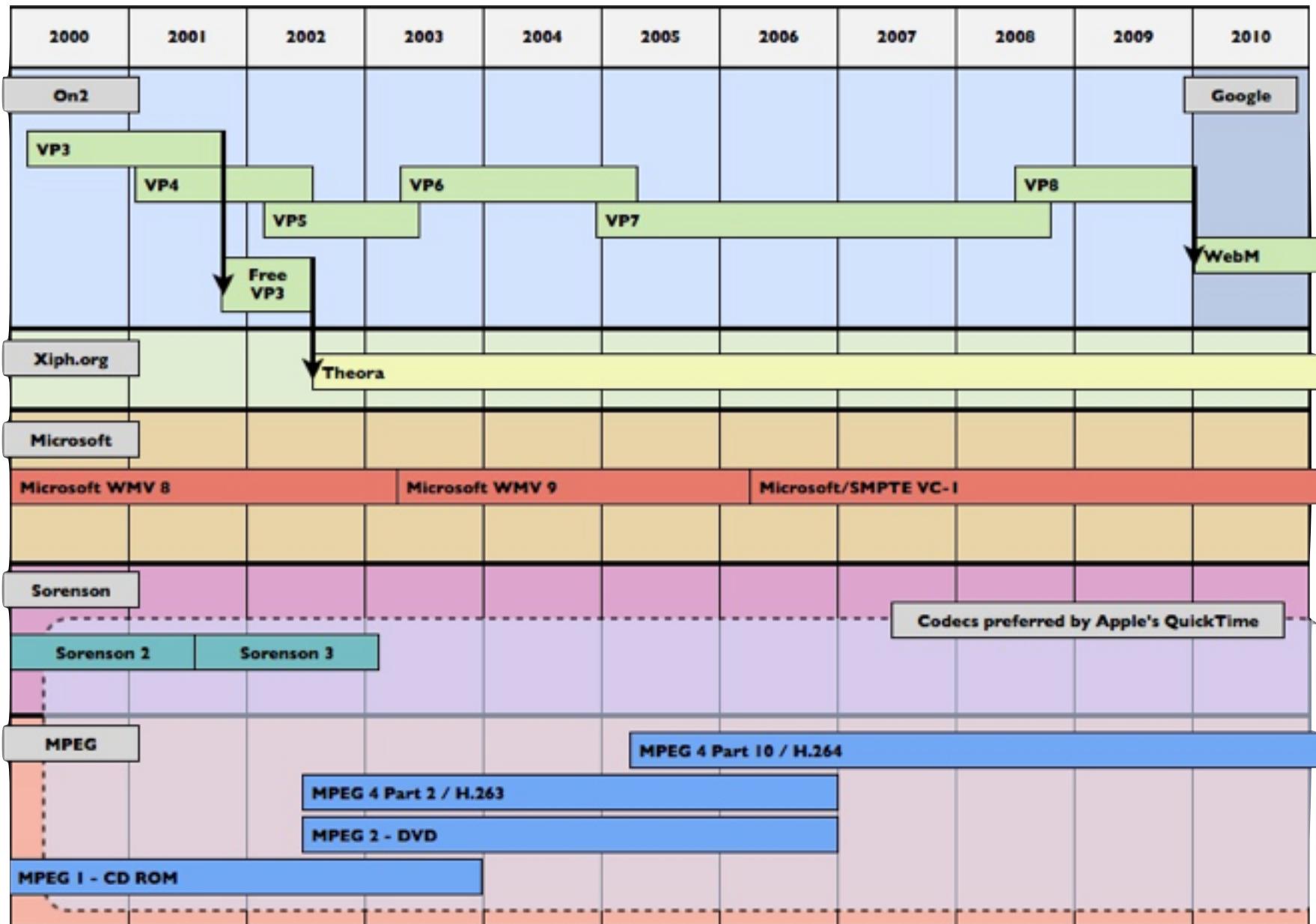
Seamless Voice Control



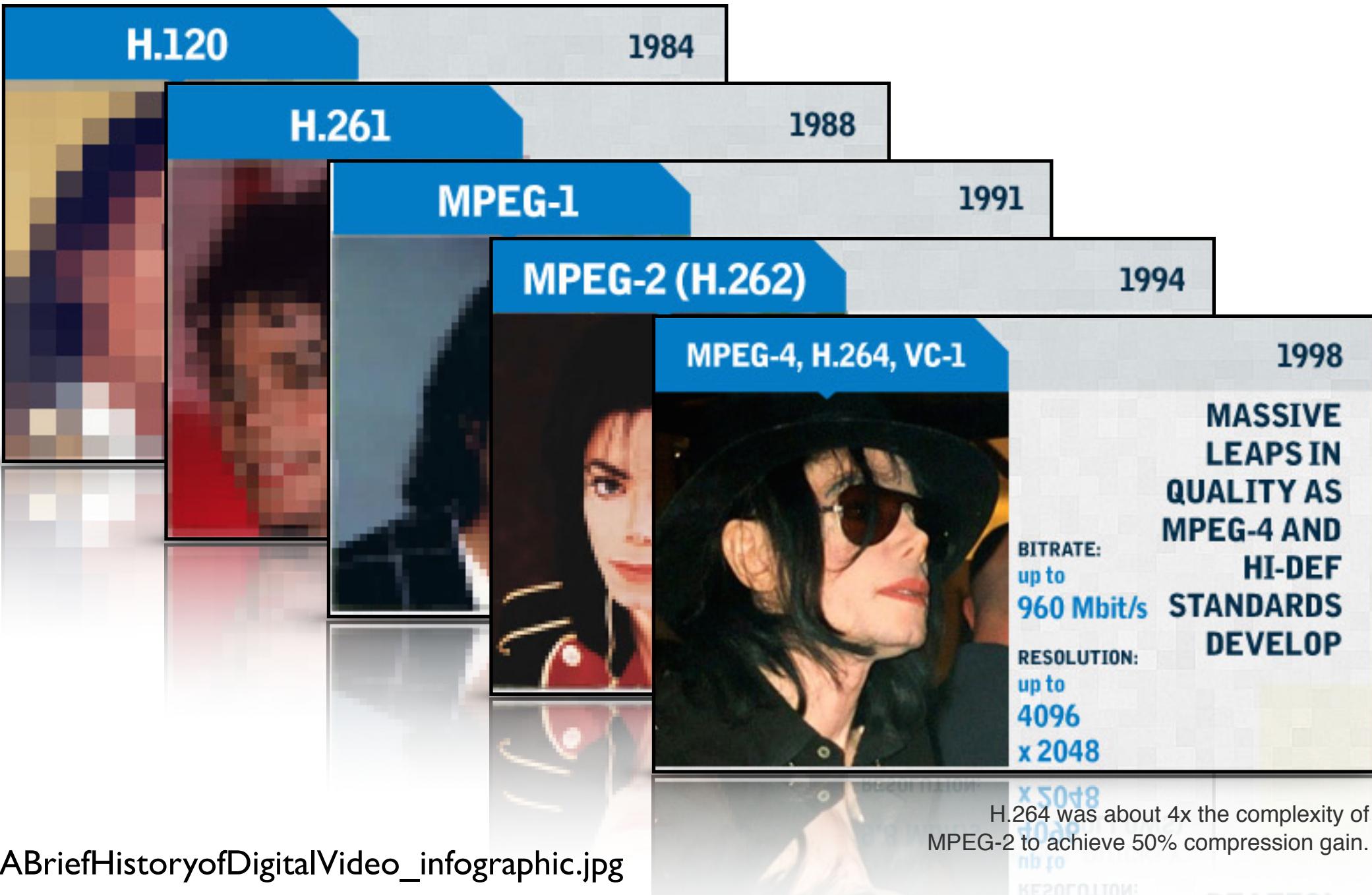
3D Screens & Holograms



Evolution des CODECs de compression vidéo



Exemples de ce qu'était la vidéo (autrefois)



HEVC - le CODEC vidéo des prochaines années

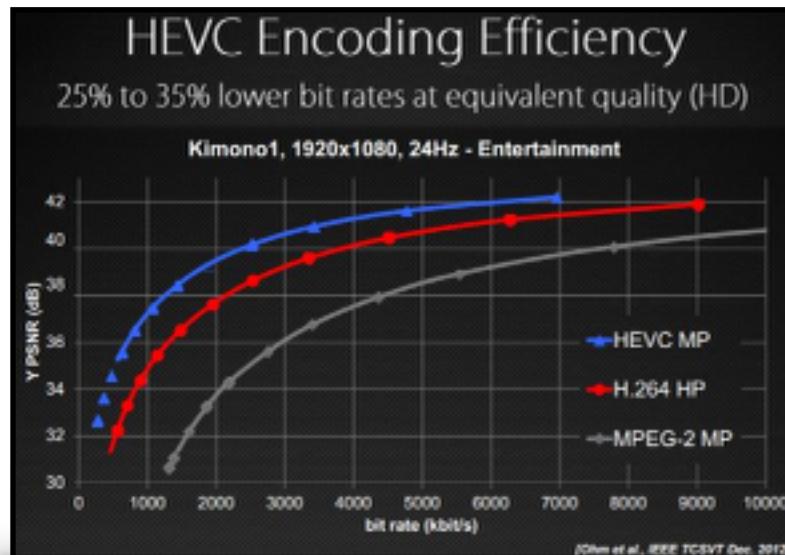
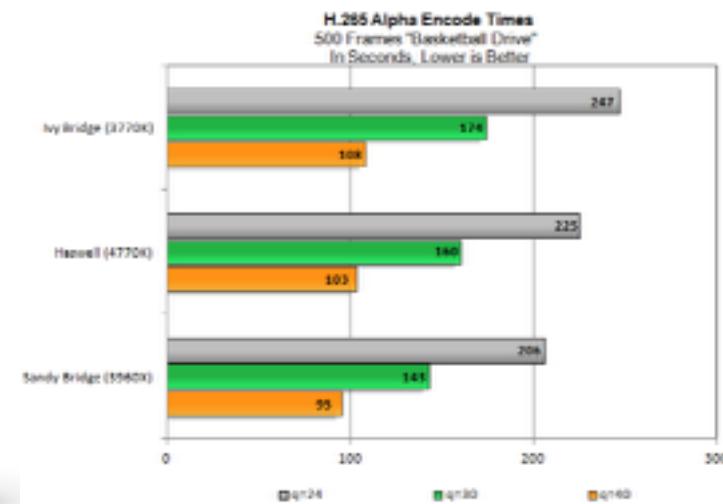


Table 1 shows the CPU required to play back H.264 and HEVC video at 720p and 1080p

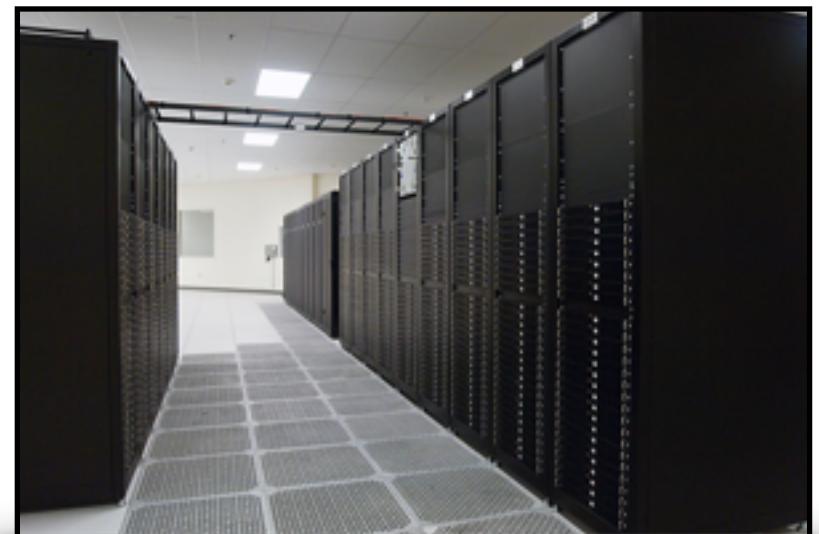
	H264 720p	HEVC 720p	H264 1080p	HEVC 1080p
Dell Precision 390 2.93 GHz Core 2	24%	40%	30%	Fail
Mac 3.06 GHz Core 2 Duo	18%	39%	35%	Fail
HP Elitebook 8760 i7-2820 (4/8 core)	10%	18%	20%	21%

Compared to x264, even on the very slow preset, x265 encodes take noticeably longer — our Ivy Bridge 3770K encoded the same file in H.264 in 129 seconds as compared to 247 seconds for H.265. Keep in mind, however, that this is very, very early software.

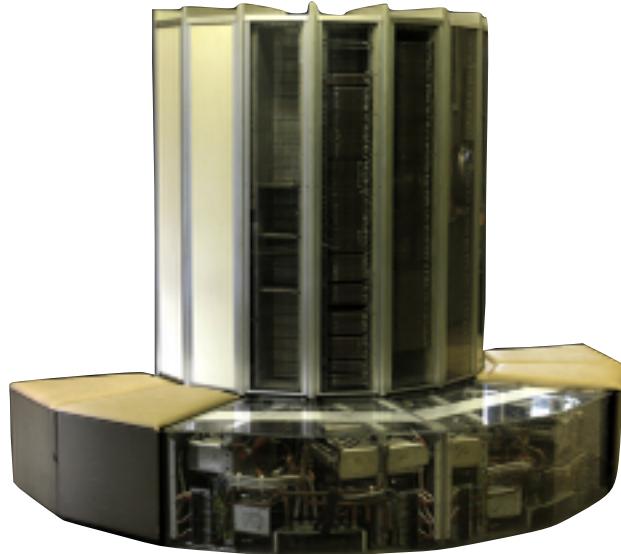


Evolution des systèmes informatiques

ENIAC, the world's first electronic digital computer (1946). It could consume 150 kilowatts of electricity.



Evolution de la puissance de calcul



1979 Cray 1 supercomputer
80 MFLOP/s



1988 Cray Y-MP
2.7 Gigaflops



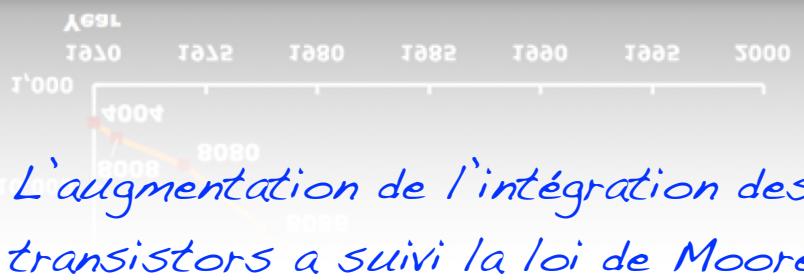
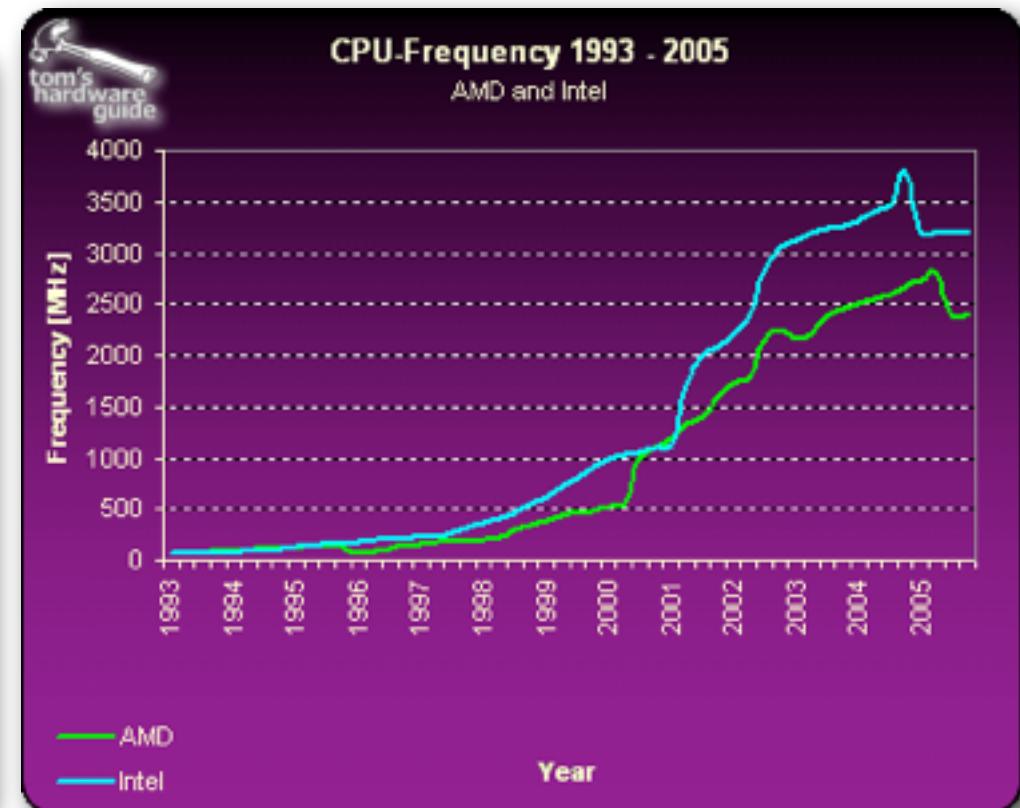
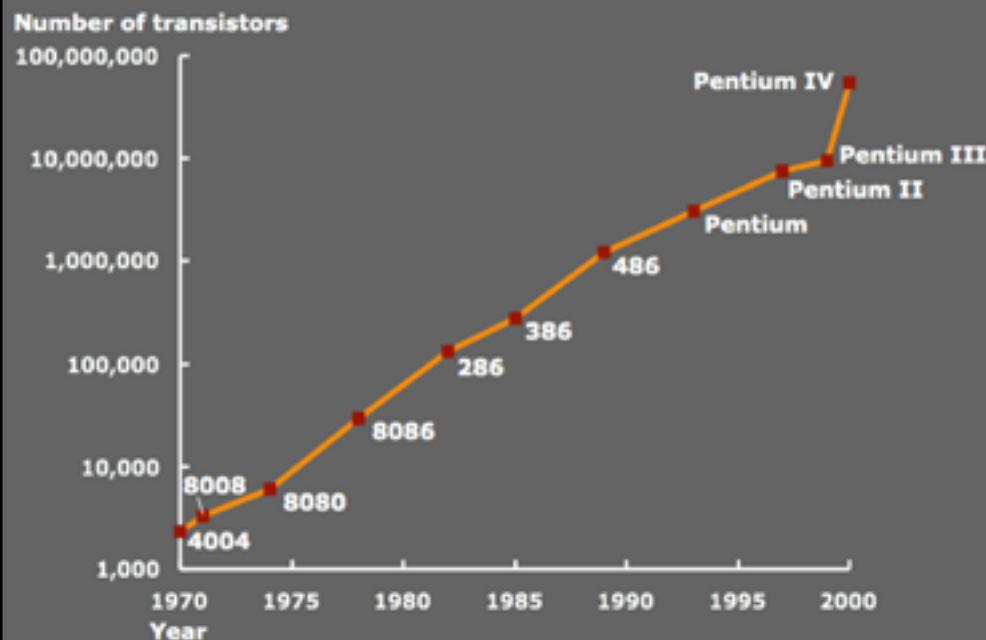
iPhone 5 - 76.8GFlops

Evolution de la puissance de calcul des processeurs

● Intel Core i7 4770k	124,850 Dhrystone MIPS	at 3.9 GHz	2013
● Qualcomm Krait	9,900 MIPS	at 1.5 GHz	2011
● Q-core Cortex-A9	13,800 MIPS	at 1.5 GHz	2011
● ARM Cortex A7	2,850 MIPS	at 1.5 GHz	2011
● ARM Cortex-M0	45 MIPS	at 50 MHz	2009
● MIPS32 24K	604 MIPS	at 400 MHz	2006
● Intel Core 2 X6800	27,079 Dhrystone MIPS	at 2.93 GHz	2006
● PS3 Cell BE (PPE only)	10,240 MIPS	at 3.2 GHz	2006
● Nios II	190 MIPS	at 165 MHz	2004
● Pentium 4 Extreme	9,726 MIPS	at 3.2 GHz	2003
● Intel Pentium III	2,054 MIPS	at 600 MHz	1999
● Intel Pentium Pro	541 MIPS	at 200 MHz	1996
● Intel Pentium	188 MIPS	at 100 MHz	1994
● Intel 486DX2	54 MIPS	at 66 MHz	1992
● Intel 386DX	9.9 MIPS	at 33 MHz	1985

Evolution des caractéristiques des processeurs généralistes

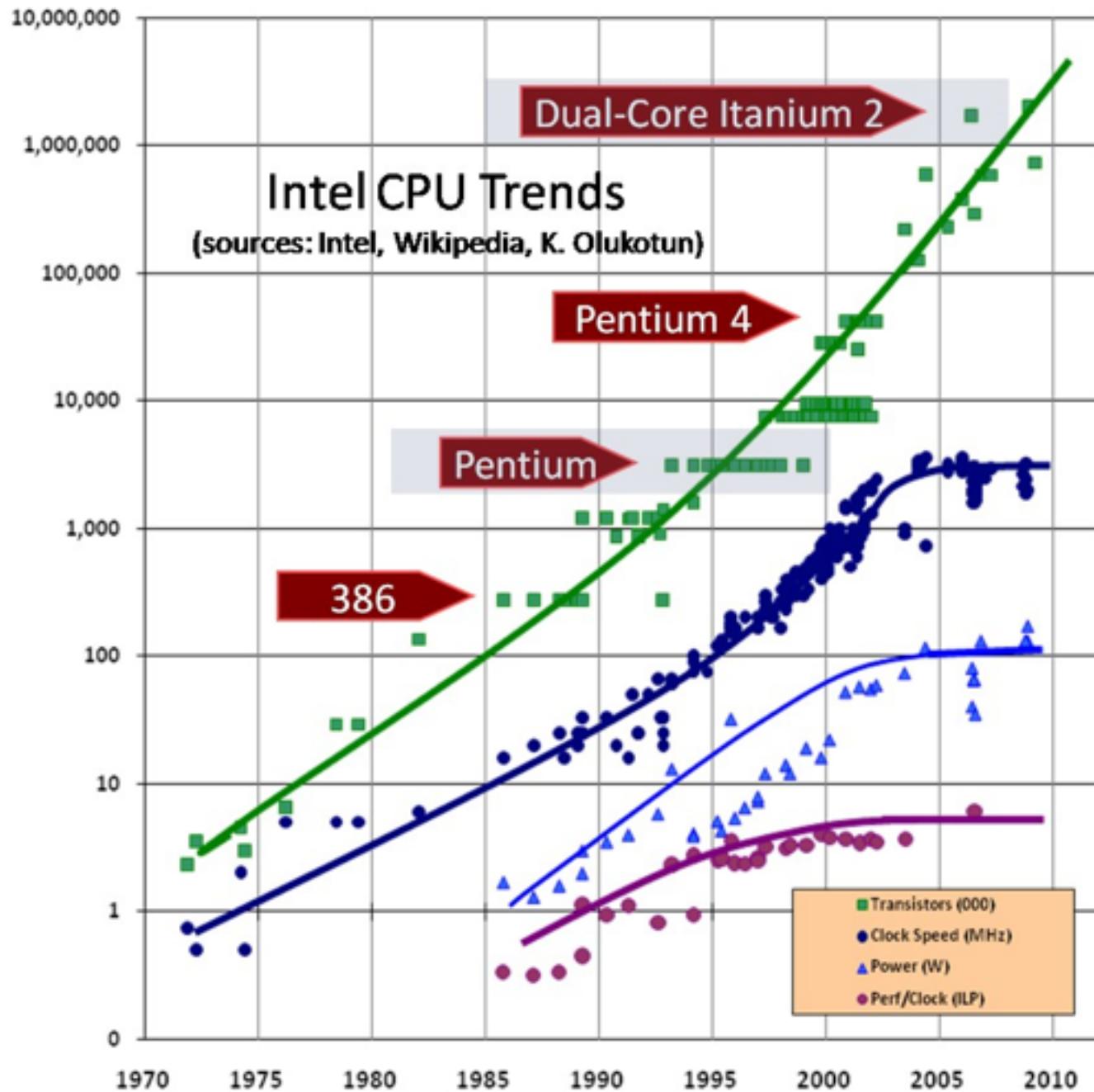
Transistor growth



L'augmentation de l'intégration des transistors a suivi la loi de Moore

Du point de vue de la fréquence, une stabilisation est apparue ces dernières années aux alentours de 3 GHz
(consommation et dissipation...)

Evolution des caractéristiques des processeurs généralistes



Evolution des caractéristiques des processeurs généralistes

Evolution of Intel Platforms

Floating point peak performance [Mflop/s]

CPU frequency [MHz]

100,000

work required

10,000

1,000

100

Pentium

Pentium Pro

Pentium II

Pentium III

free speedup

single precision

double precision

CPU frequency

era of parallelism

1993

1995

1997

1999

2001

2003

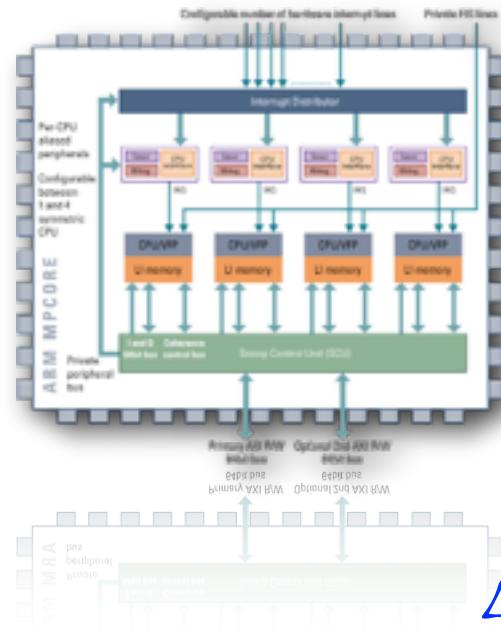
2005

2007

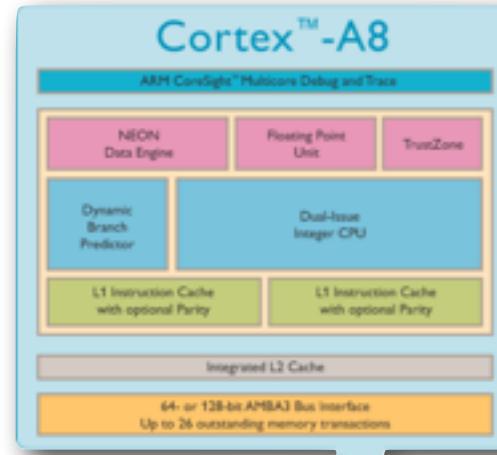
Year

data: www.sandpile.org

Augmentation de la complexité des architectures matérielles



ARM11
90nm



ARM Cortex A8
65nm

La technologie diminue mais la taille
des circuits augmente fortement !

ARM11
65nm

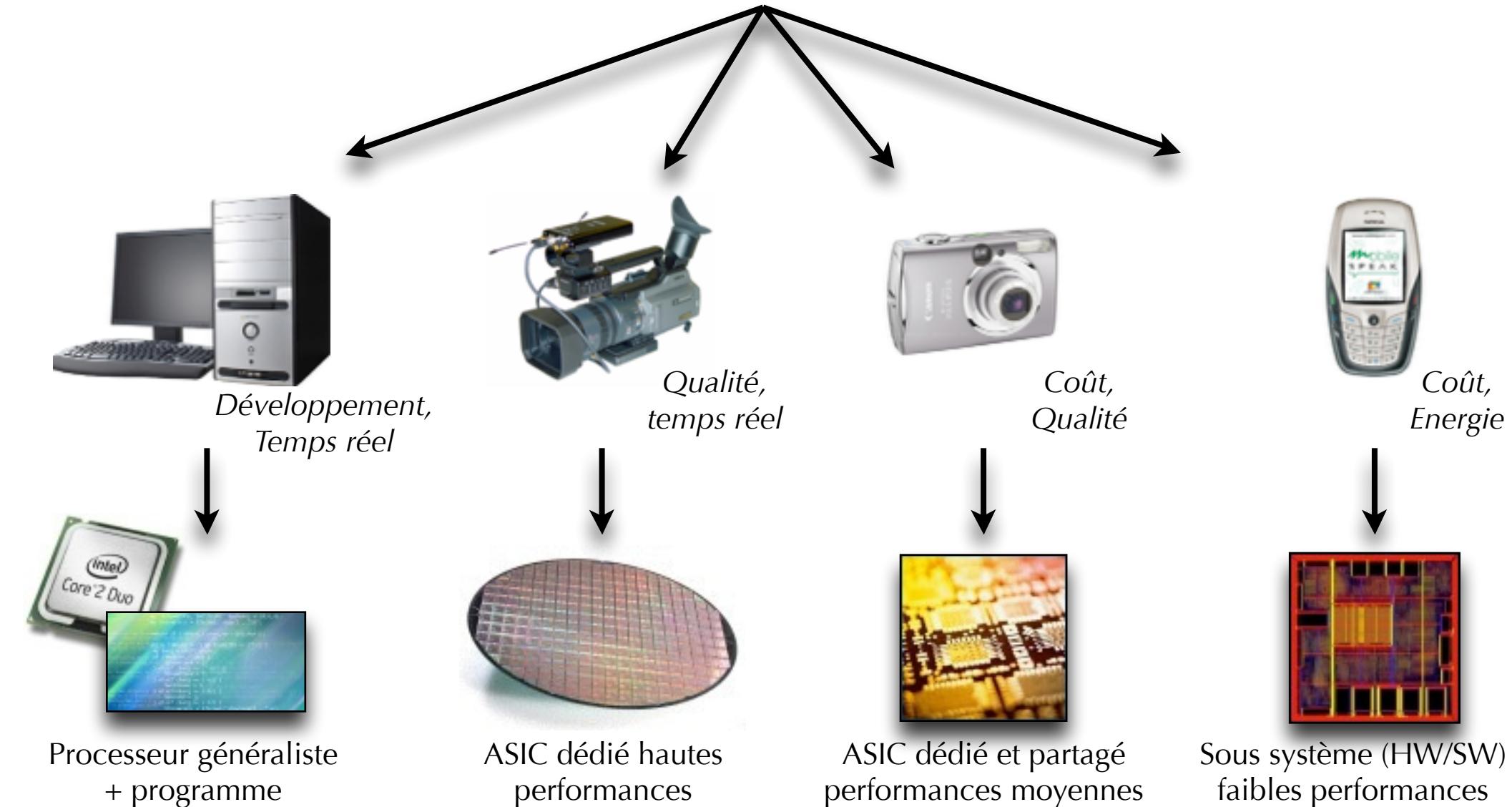
ARM11
90nm

ARM Cortex A8
65nm

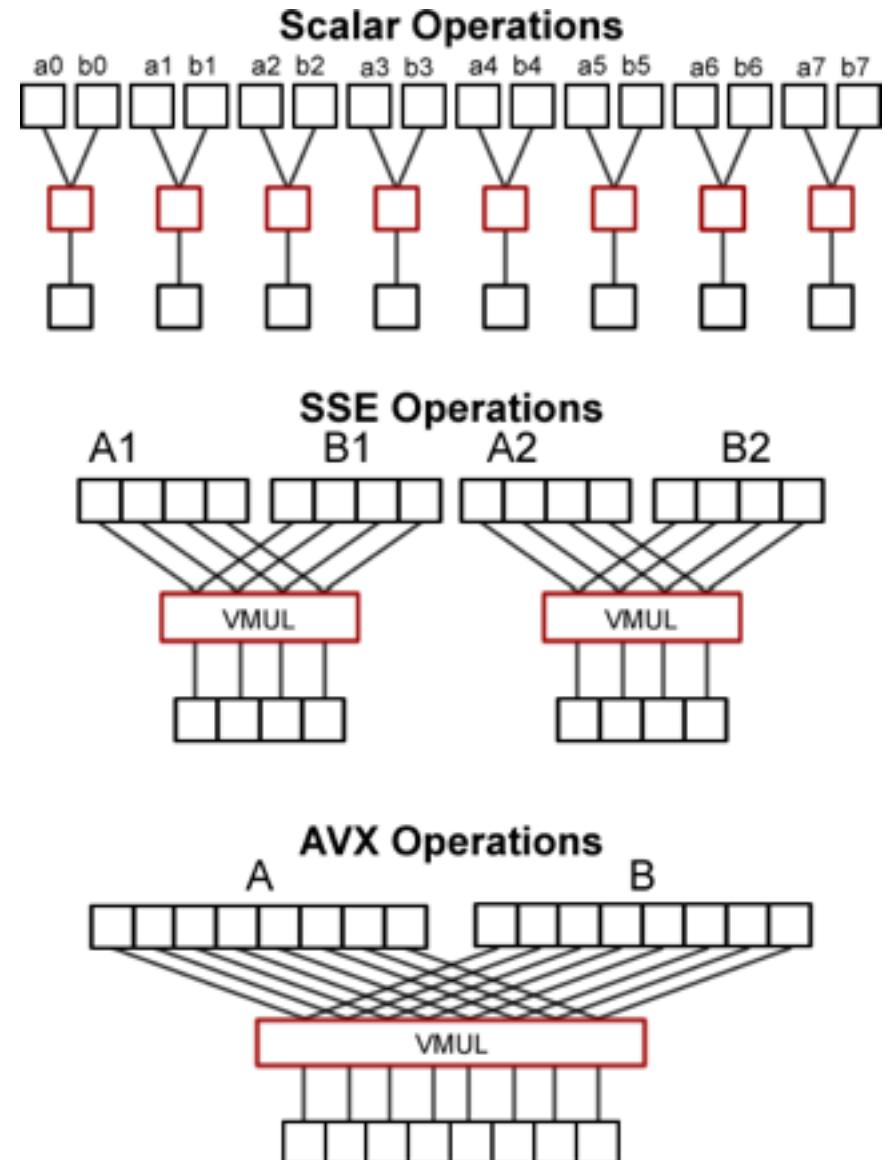
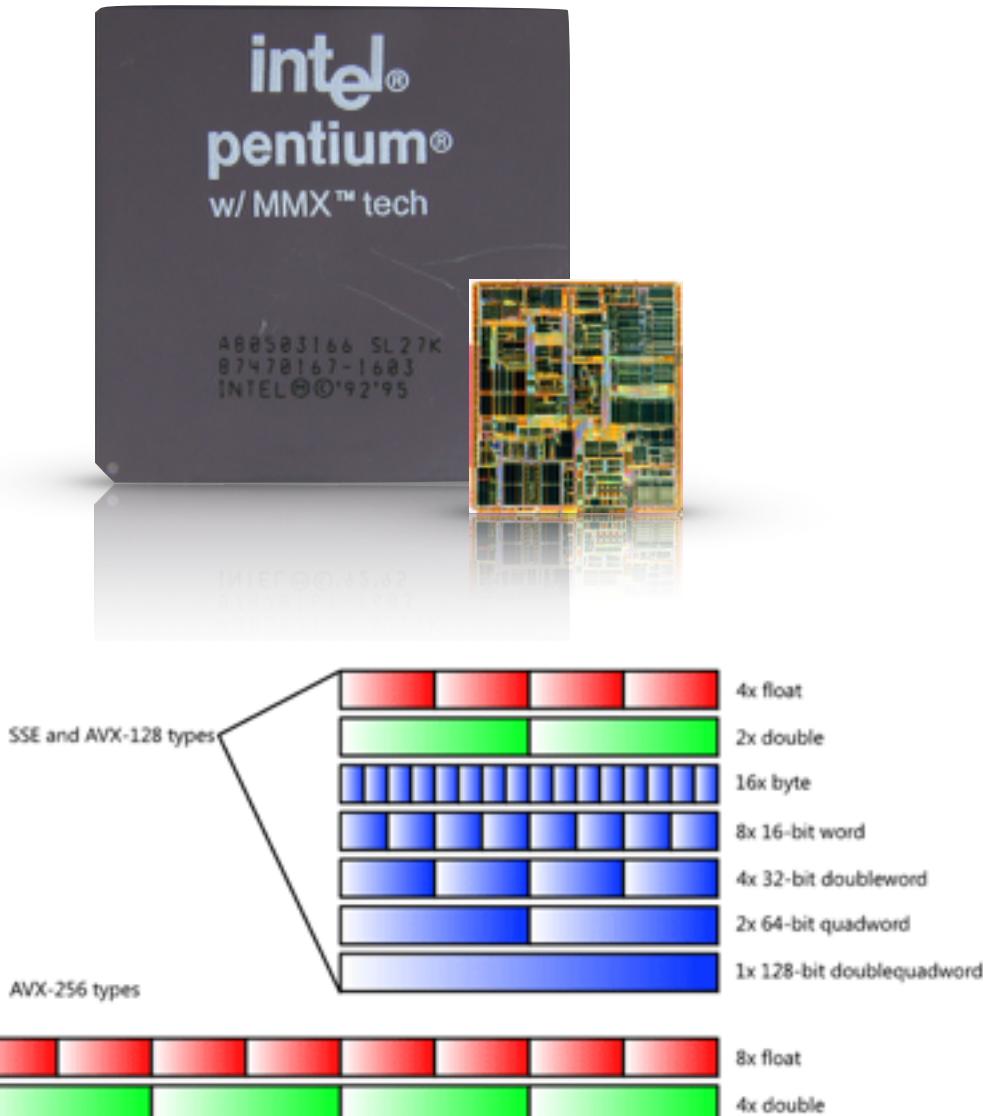
<http://www.engadget.com/2009/10/14/core-values-the-silicon-behind-android/>

Exemple d'étude pour une application

Compression vidéo h265



L'évolution des architectures matérielles (processeurs)



Exemples de gains obtenus grâce à la vectorisation

Intel® AVX-intrinsic Mandelbrot Implementation

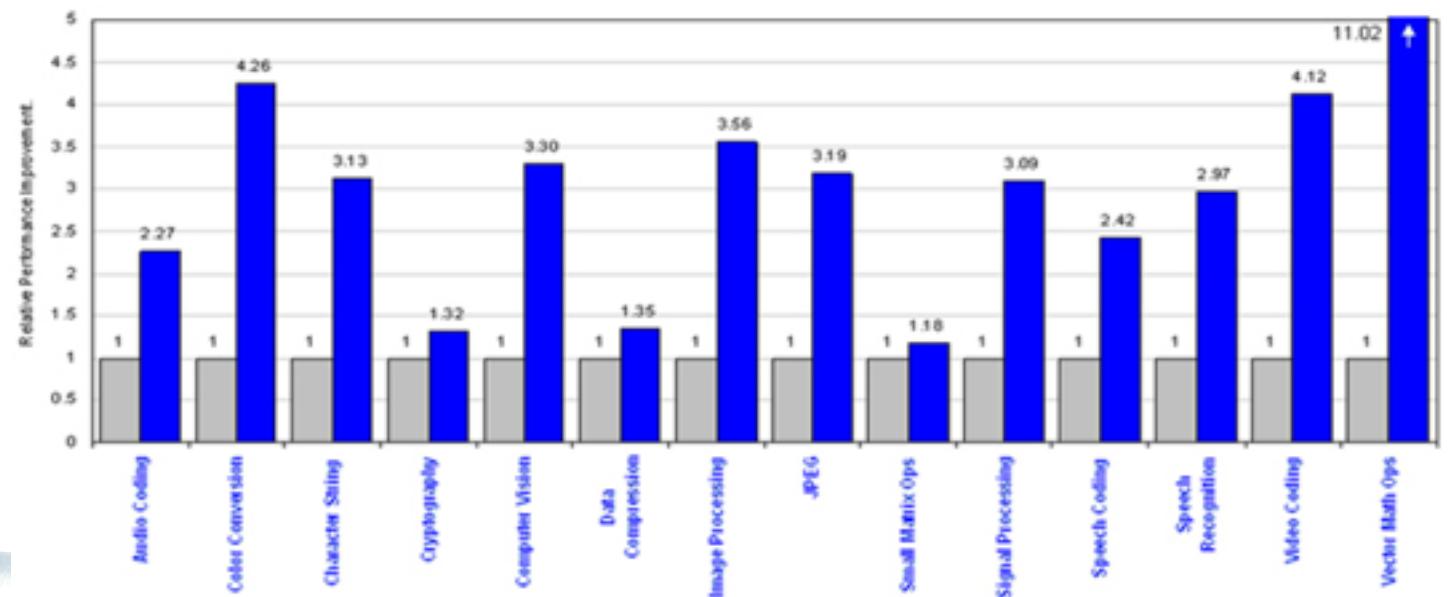
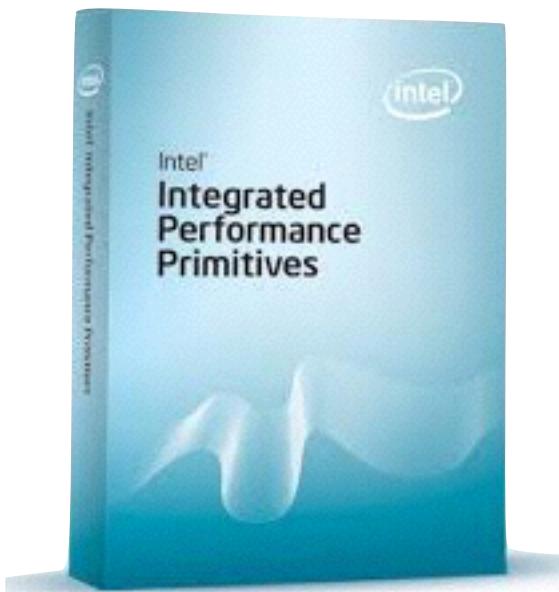
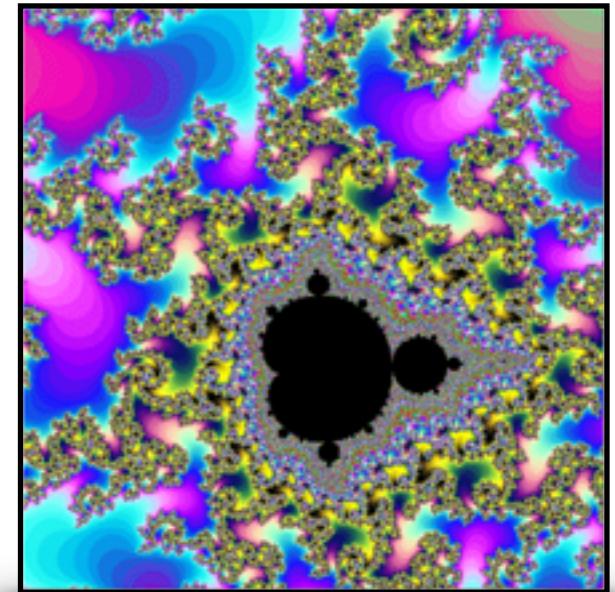
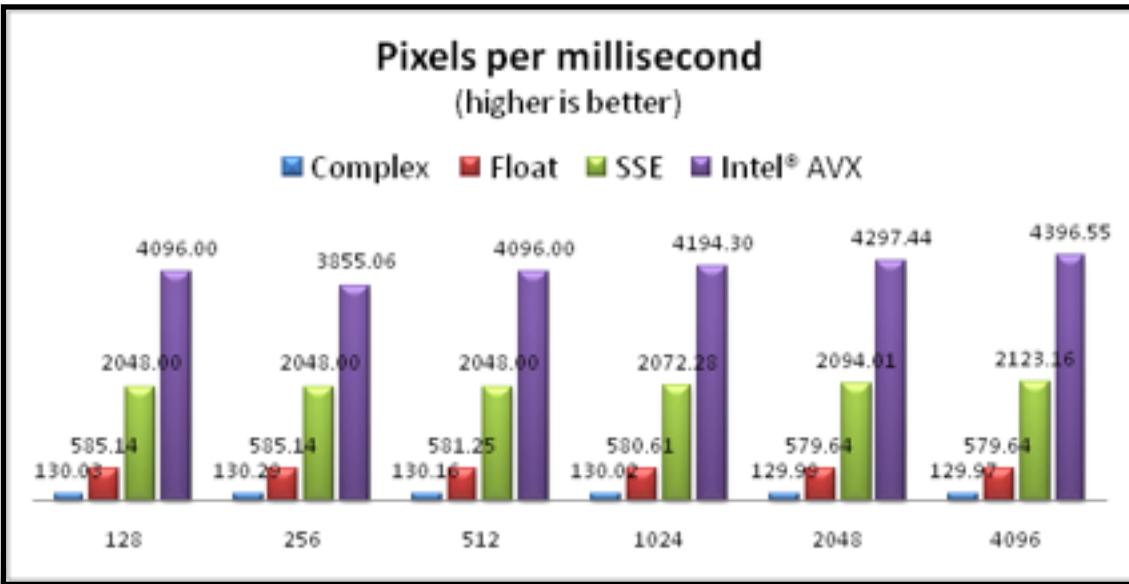


Figure shows relative average performance improvements measured for the various Intel IPP (SSE).

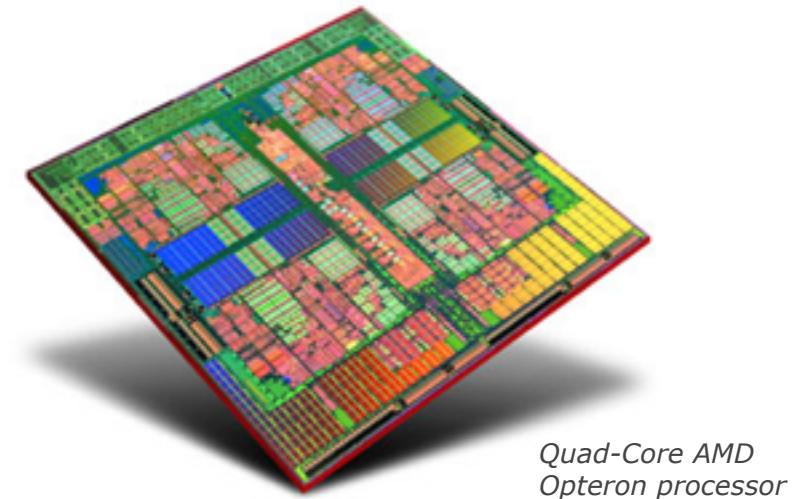
L'évolution des architectures matérielles (processeurs)

● Les architectures multi-coeurs

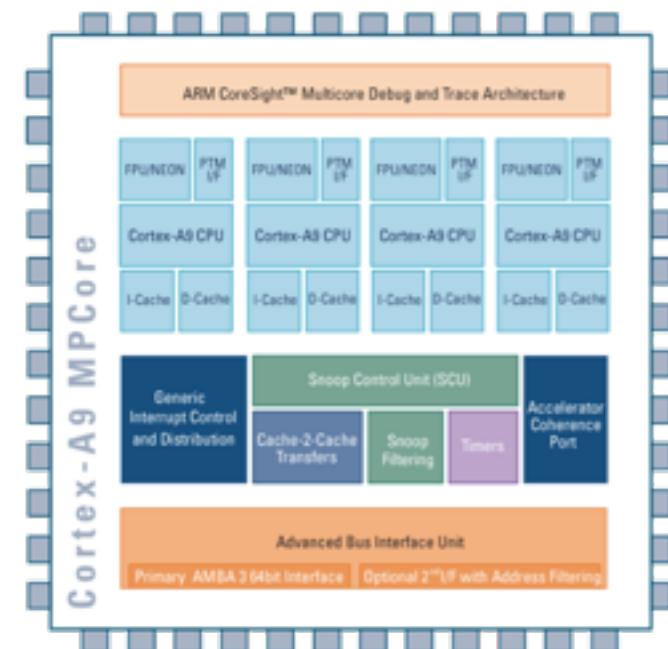
- Pour augmenter la puissance de calcul, on place N coeurs homogènes,
 - Processeurs généralistes,
 - Processeurs embarqués,
- N fois la puissance de calcul en théorie,

● Requiert des algorithmes adaptés,

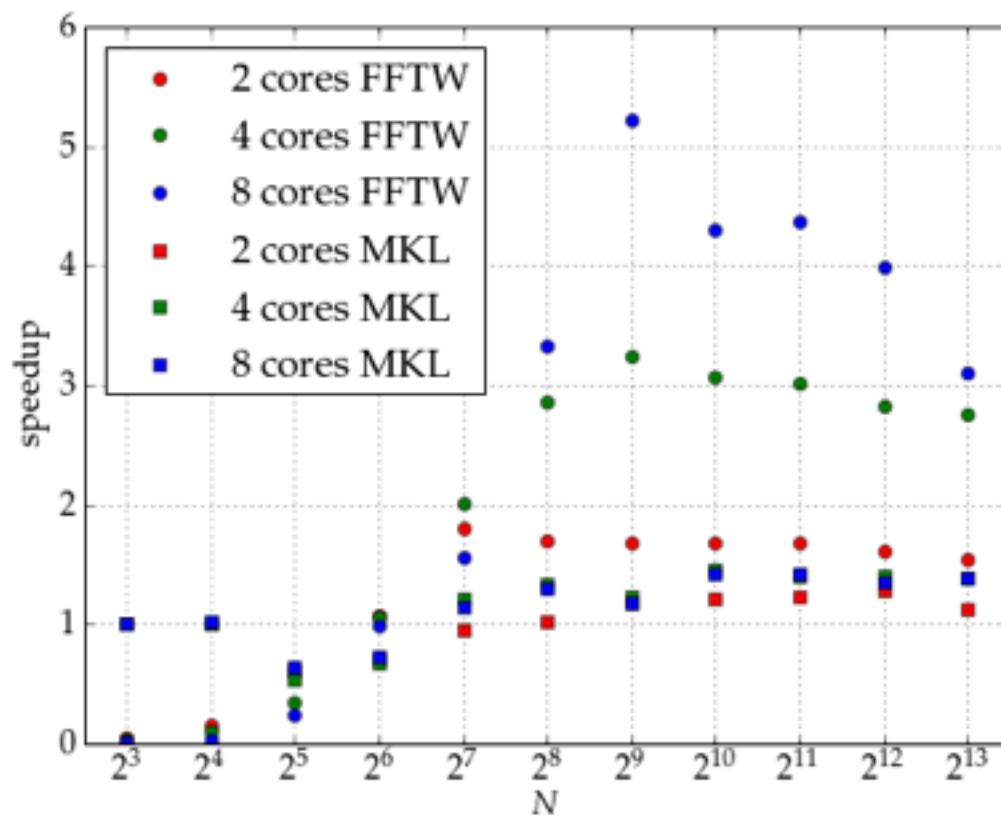
- La programmation efficace de tels systèmes est complexe,
- Charger tous les processeurs,



Quad-Core AMD
Opteron processor

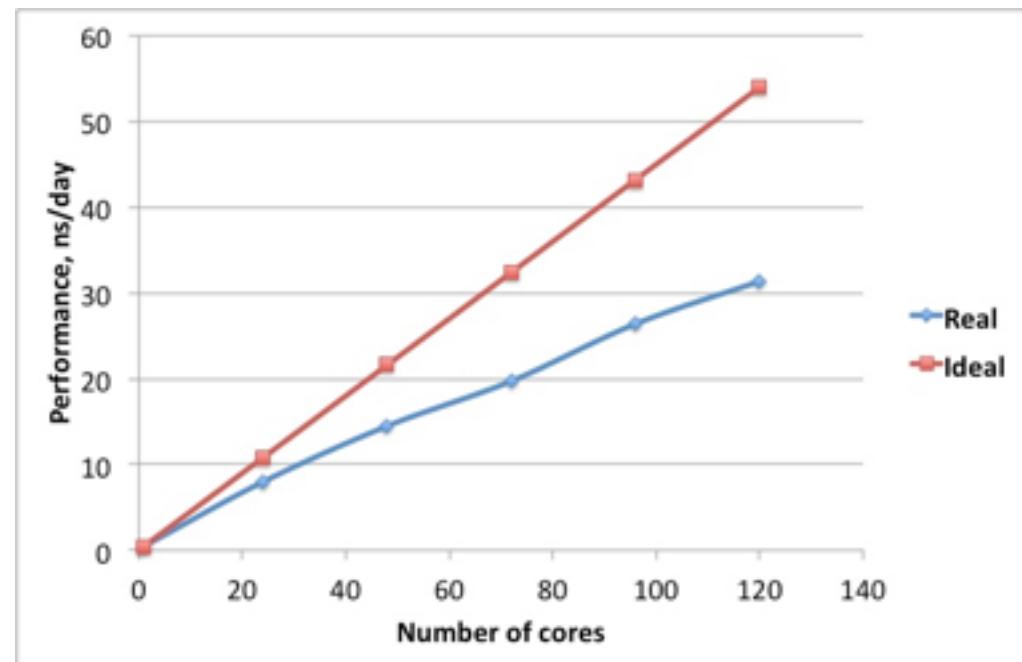


Exemples de gains monocoeur versus multicoeurs



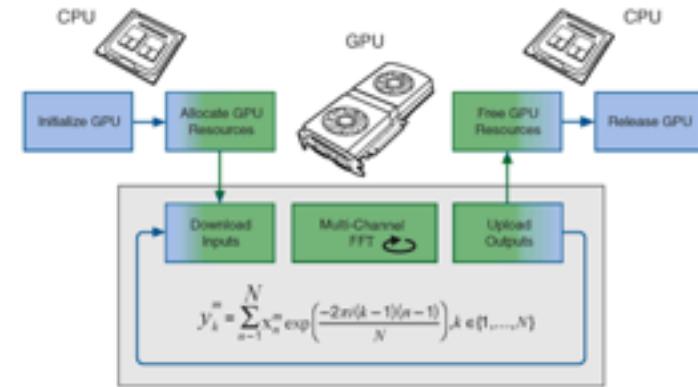
Parallel speed up of parallel FFT of four interwoven two-dimensional matrices of size $N \times N$.

Gromacs on Lindgren up to 120 cores using a voltage-gated ion channel system with a system size of ~28,000 atoms. The test runs are done on 24 core nodes of which 6 are using OpenMP threads for PME and 18 are using MPI for the rest of the calculations.

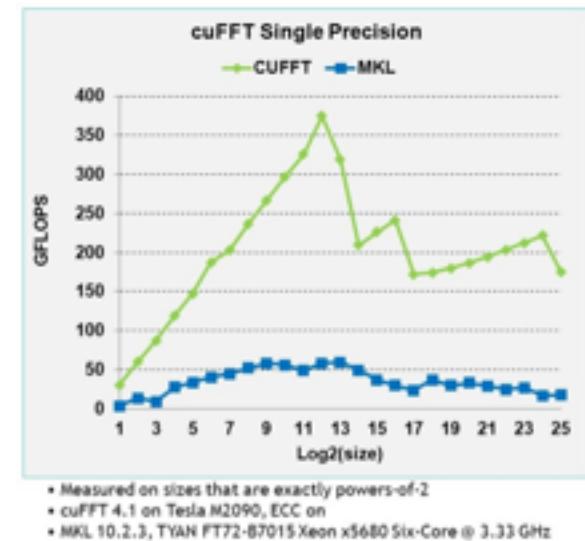
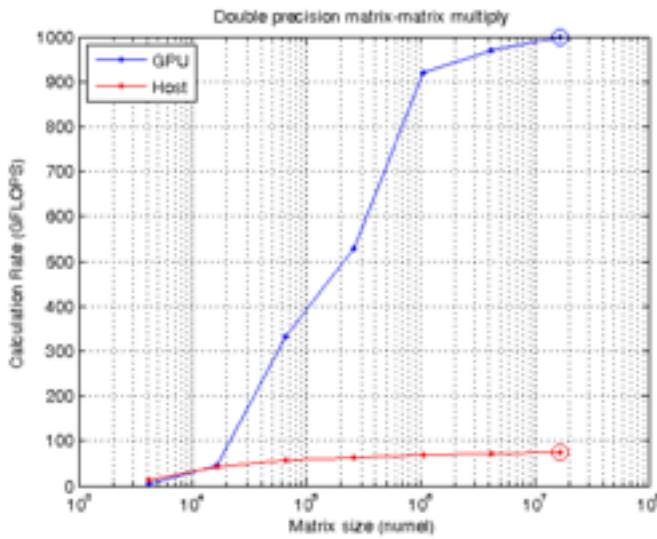
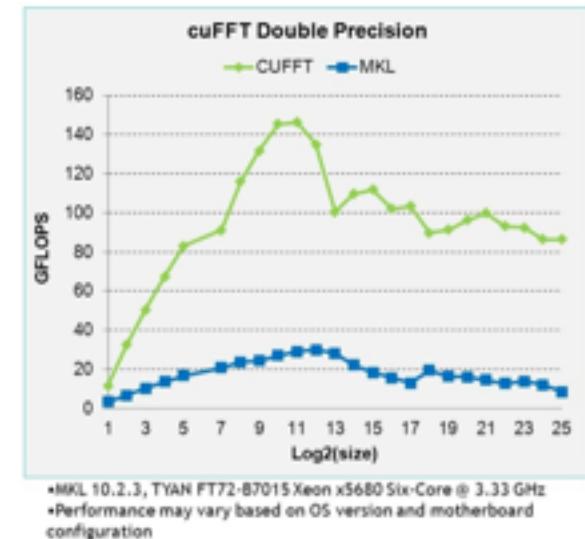
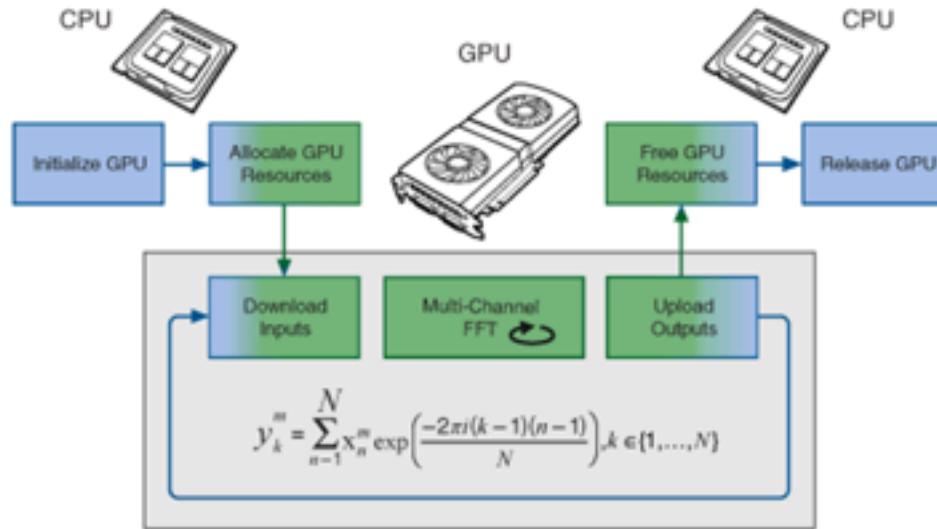


L'évolution des architectures matérielles (GPU)

- Les GPUs sont des processeurs « multicoeurs » avec des coeurs élémentaires (ALU)
- Parallelisme de calcul poussé à l'extrême: >2000 coeurs de calculs par GPU
- Fréquence de fonctionnement des coeurs env. 800MHz
- Les GPUs
 - Adaptés aux calculs massivement parallèles,
 - Inadaptés aux faibles lots d'information.



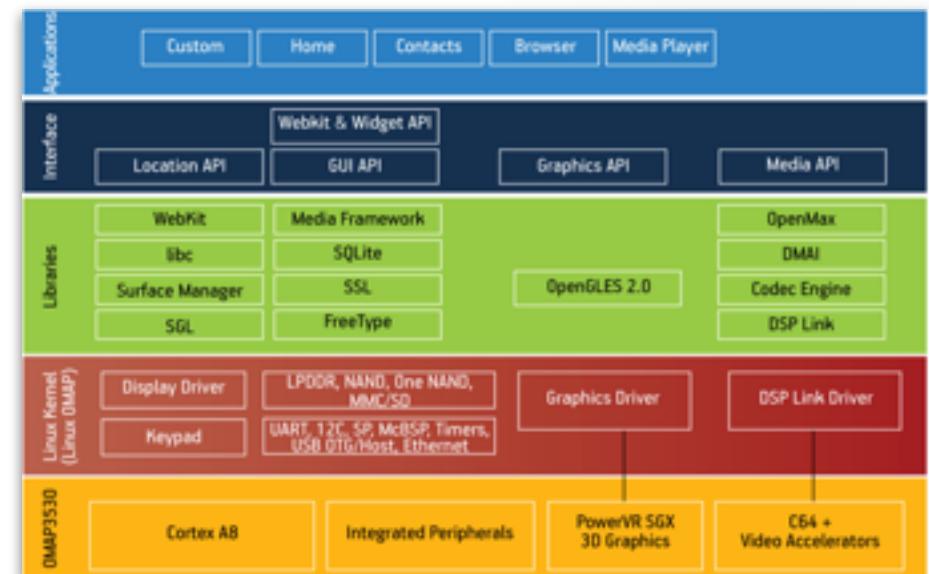
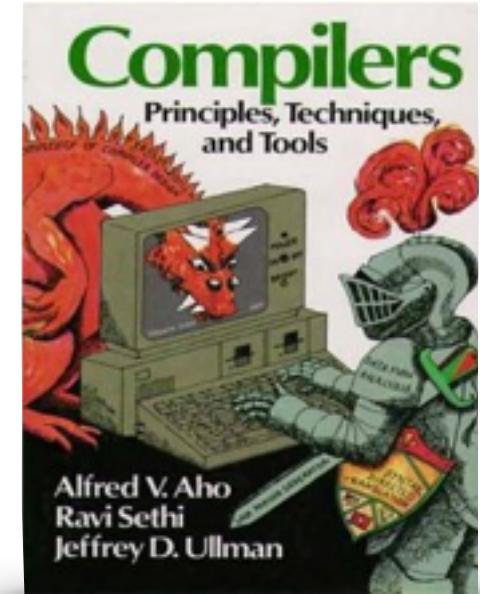
Exemples de gains CPU versus GPU



<http://www.mathworks.fr/fr/help/distcomp/examples/measuring-gpu-performance.html>

Sans maîtrise, la puissance n'est rien...

- De « belles » architectures, c'est intéressant, mais il faut pouvoir les exploiter !
- Coder en « assembleur » ne permet pas de développer et d'optimiser des applications réelles !
- Nécessité d'avoir à disposition des langages et des outils pour exploiter les architectures !
 - Compilateurs;
 - Frameworks de compilation;
 - Bibliothèques optimisées.



Vectorisation automatique à l'aide de GCC

example1:

```
int a[256], b[256], c[256];
foo () {
    int i;

    for (i=0; i<256; i++){
        a[i] = b[i] + c[i];
    }
}
```

example14: double reduction:

```
for (k = 0; k < K; k++) {
    sum = 0;
    for (j = 0; j < M; j++)
        for (i = 0; i < N; i++)
            sum += in[i+k][j] * coeff[i][j];

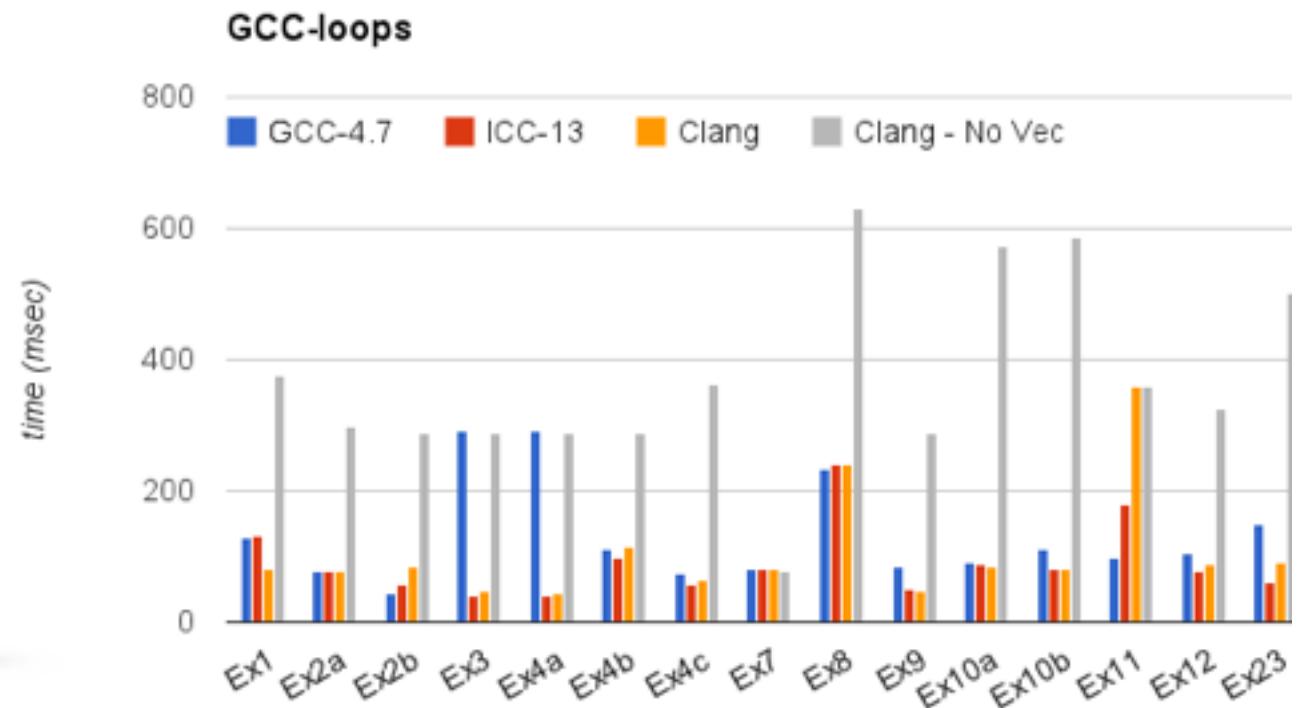
    out[k] = sum;
}
```

Unvectorizable Loops

Examples of loops that currently cannot be vectorized:

example1: uncountable loop:

```
while (*p != NULL) {
    *q++ = *p++;
}
```



Exploitation des arch. multicoeurs à l'aide d'OpenMP

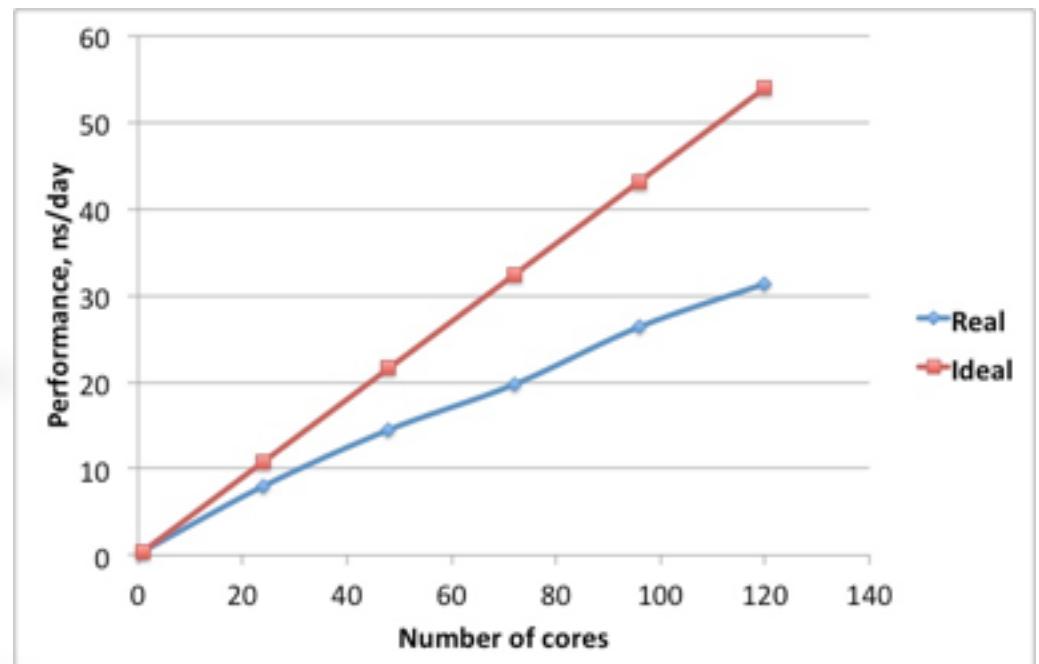
```
#pragma omp sections nowait
{
    #pragma omp section
    {
        for (i=0; i<N; i++){
            c[i] = a[i] + b[i];
        }
    }

    #pragma omp section
    {
        for (i=0; i<N; i++){
            d[i] = a[i] * b[i];
        }
    }
}
```

```
#pragma omp parallel for reduction(+:sum)
for (i=0; i < n; i++)
    sum = sum + (a[i] * b[i]);
```

```
#pragma omp parallel for
for (i=0; i < n; i++){
    int x = data[i];
    result[i] = the_big_function( x );
}
```

```
#pragma omp for
for (i=0; i<NRA; i++)
{
    for(j=0; j<NCB; j++)
        for (k=0; k<NCA; k++)
            c[i][j] += a[i][k] * b[k][j];
}
```



Description d'un kernel de calcul en CUDA

```
void main()
{
    ...
    inc_cpu(a, N);
}
void inc_cpu(int *a, int N)
{
    int idx;
    for (idx = 0; idx < N; idx++)
        a[idx] = a[idx] + 1;
}
```



```
void main()
{
    ...
    dim3 dimBlock (blocksize);
    dim3 dimGrid (ceil(N/(float)blocksize));
    inc_gpu<<<dimGrid, dimBlock>>>(a, N);
}
__global__ void inc_gpu(int *a, int N)
{
    int idx = blockIdx.x * blockDim.x +
              threadIdx.x;
    if (idx < N)
        a[idx] = a[idx] + 1;
}
```

```
__global__ void vecAdd(float * A, float * B, float * C)
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    const int nThreadsPerBlocks = 4;
    const int nBlocks = (arraySize/nThreadsPerBlocks) + ((arraySize%nThreadsPerBlocks)==0?0:1);
    vecAdd<<<nBlocks, nThreadsPerBlocks>>>(A, B, C);
}
```

Description d'un kernel de calcul en CUDA

```
1 // example1.cpp : Defines the entry point for the console application.
2 //
3
4 #include "stdafx.h"
5
6 #include <stdio.h>
7 #include <cuda.h>
8
9 // Kernel that executes on the CUDA device
10 __global__ void square_array(float *a, int N)
11 {
12     int idx = blockIdx.x * blockDim.x + threadIdx.x;
13     if (idx<N) a[idx] = a[idx] * a[idx];
14 }
15
16 // main routine that executes on the host
17 int main(void)
18 {
19     float *a_h, *a_d; // Pointer to host & device arrays
20     const int N = 10; // Number of elements in arrays
21     size_t size = N * sizeof(float);
22     a_h = (float *)malloc(size); // Allocate array on host
23     cudaMalloc((void **)&a_d, size); // Allocate array on device
24     // Initialize host array and copy it to CUDA device
25     for (int i=0; i<N; i++) a_h[i] = (float)i;
26     cudaMemcpy(a_d, a_h, size, cudaMemcpyHostToDevice);
27     // Do calculation on device:
28     int block_size = 4;
29     int n_blocks = N/block_size + (N%block_size == 0 ? 0:1);
30     square_array <<< n_blocks, block_size >>> (a_d, N);
31     // Retrieve result from device and store it in host array
32     cudaMemcpy(a_h, a_d, sizeof(float)*N, cudaMemcpyDeviceToHost);
33     // Print results
34     for (int i=0; i<N; i++) printf("%d %f\n", i, a_h[i]);
35     // Cleanup
36     free(a_h); cudaFree(a_d);
37 }
```

Les processeurs ne sont pas la seule solution !

- Au lieu d'utiliser des circuits sur étagère (processeurs), autant concevoir le circuit,
- Conception de circuits en VHDL de niveau RTL
 - Hautes performances,
 - Efficacité silicium,
 - Efficacité énergétique,
- Inconvénients
 - le temps de développement;
 - le niveau d'expertise.

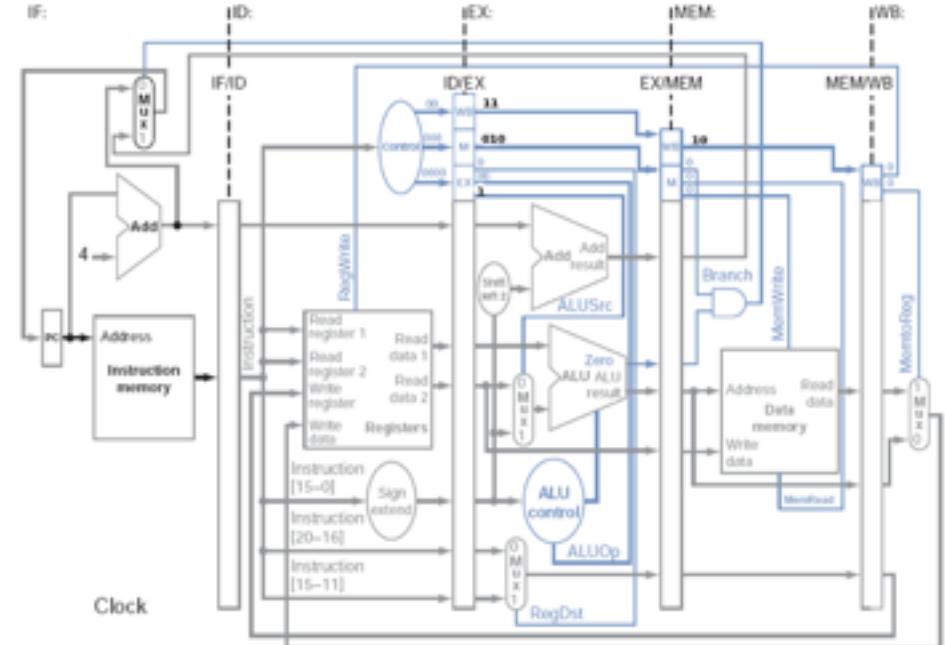
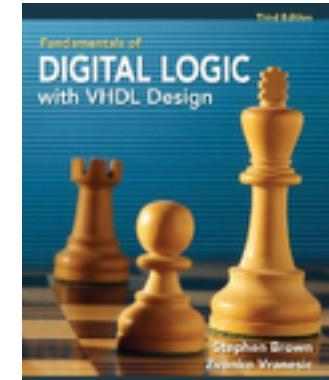
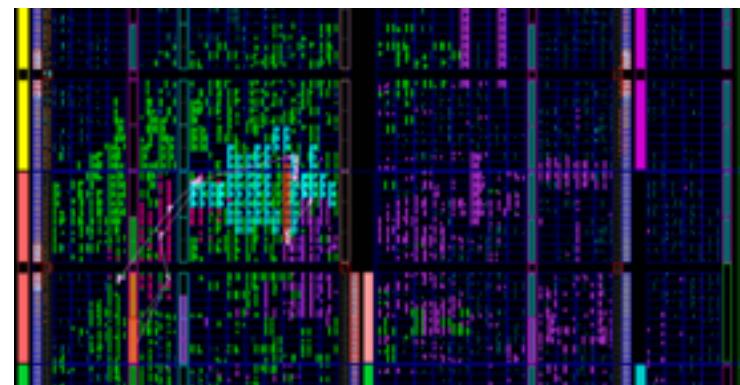


FIGURE 6.14.10 A blank single-clock-cycle pipeline diagram with control. (With Corrections)



Et si on automatisait tout cela...

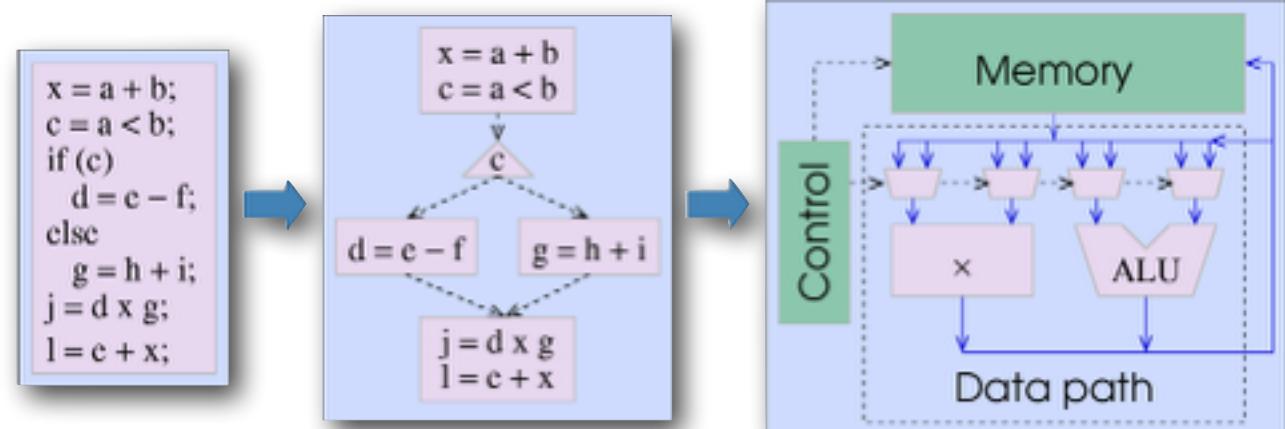
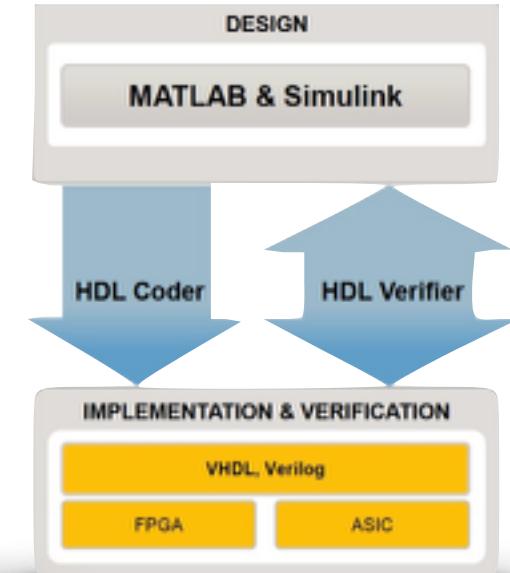
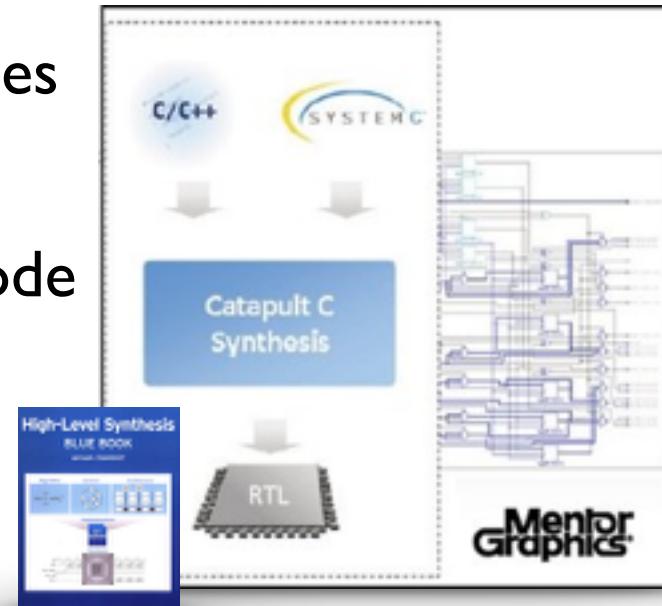
- Objectif, éviter l'écriture des architectures VHDL,
- Outils de génération de code VHDL à partir de:

- Spécification en C,
- Contraintes,

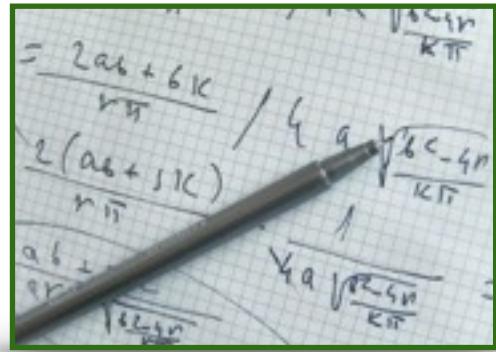
- Outils:

- Onéreux;
- Difficiles à maîtriser;
- Sous optimal.

- Solution d'avenir pour les systèmes complexes.



Vers la conception de systèmes complexes (ASIP)



Il existe des solutions intermédiaires apportant performances et flexibilité !

ISA Instruction Options	
<input type="checkbox"/>	HAC32 instruction family
<input type="checkbox"/>	MUL32
<input type="checkbox"/>	MUL16
<input type="checkbox"/>	Floating Point (coprocessor id 0)
<input checked="" type="checkbox"/>	MIN/MAX and MAXU/MAXU
<input type="checkbox"/>	Enable density instructions
<input type="checkbox"/>	Enable Processor ID
<input type="checkbox"/>	Synchronization instruction
<input type="checkbox"/>	TIE arbitrary byte enables
<input checked="" type="checkbox"/>	Number of Coprocessors (NCP)
<input type="checkbox"/>	Thread Pointer
<input type="checkbox"/>	CLAMPS
<input type="checkbox"/>	Add MUL32 options MULH and MULSH
<input checked="" type="checkbox"/>	NSA/NSAU
<input checked="" type="checkbox"/>	Sign Extend to 32 bits
<input type="checkbox"/>	Enable Boolean Registers
<input checked="" type="checkbox"/>	Zero-overhead loop instructions
<input type="checkbox"/>	Conditional store synchronize instruction
<input type="checkbox"/>	Miscellaneous Special Register count



Pourquoi vouloir rajouter de nouvelles instructions ?

```
int bit_swap(int a){  
    int r = 0;  
    for(int i=0; i<32; i++){  
        r = r | (a & 0x01);  
        a = a >> 1;  
        r = r << 1;  
    }  
    return r;  
}
```

```
}
```

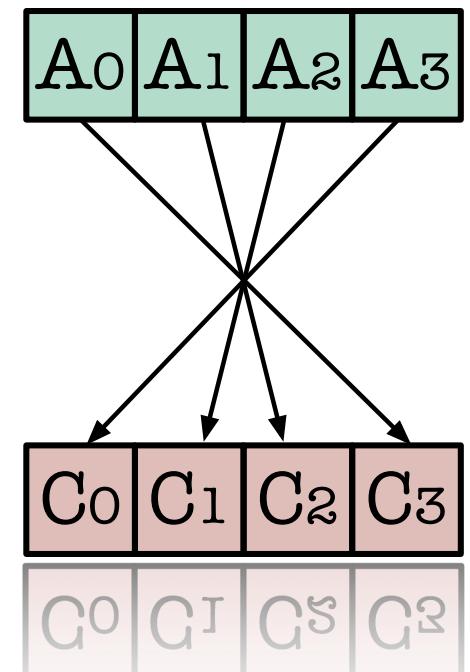
REFRANU n:

Le cout matériel pour ajouter une instruction permettant de réaliser cette opération est nul (quasi-nul) car il s'agit de fils.

- Cout matériel de l'opérateur : 0 LUT
- chemin critique : 0 ns
- facteur d'accélération max. : x250

Les opérations contenues dans un algorithme ne sont pas toujours efficacement implantée dans le processeur (RISC : Reduced Instruction Set).

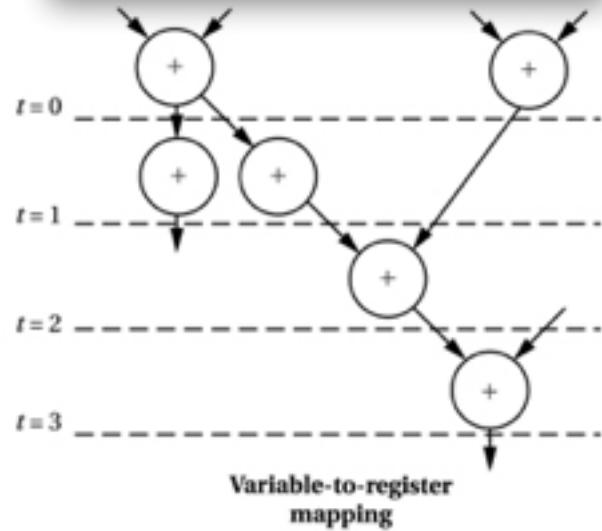
Exemple de la permuttation de bits dans un mot de 32 bits (Little Endian => Big Endian).



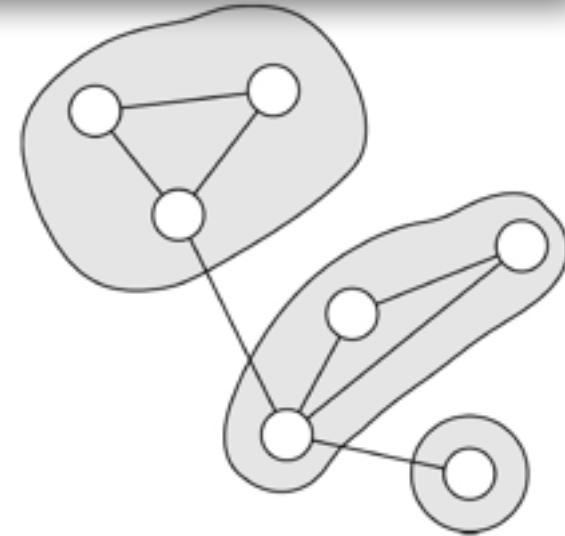
* Chiffres donnés pour une cible Virtex-4 (XC4VSX53-10)

Conception automatisée d'un ASIP, une tache complexe !

Modélisation des calculs



Identification des instructions

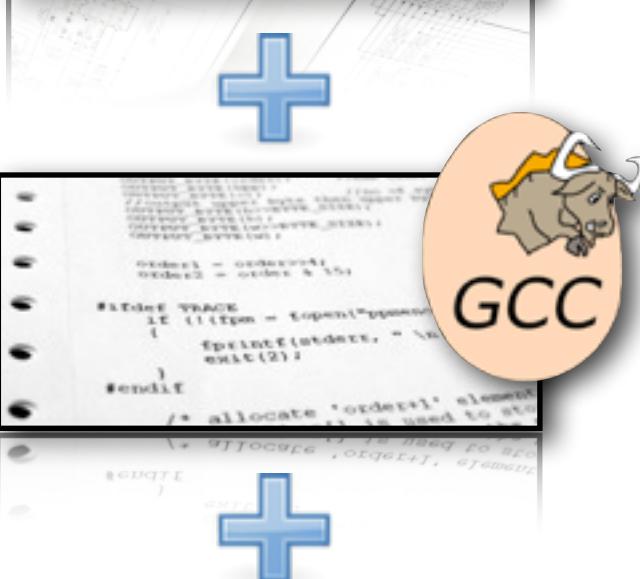
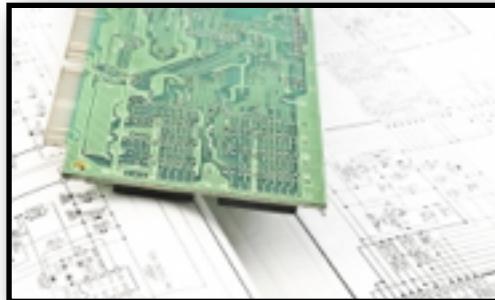


Estimation des performances & sélection des instructions



Génération du processeur

Les pré-requis nécessaires pour concevoir un ASIP



Un cœur de processeur modifiable
=> Décrit en VHDL (ou en Verilog)
=> Un jeu d'instructions extensible

Une chaîne de compilation modifiable
=> Les codes source du compilateur

Les connaissances nécessaires
=> Applications à intégrer
=> Architecture des processeurs
=> Assembleur, code C
=> Langages matériels (VHDL, Verilog)

Pourquoi vouloir rajouter de nouvelles instructions ?

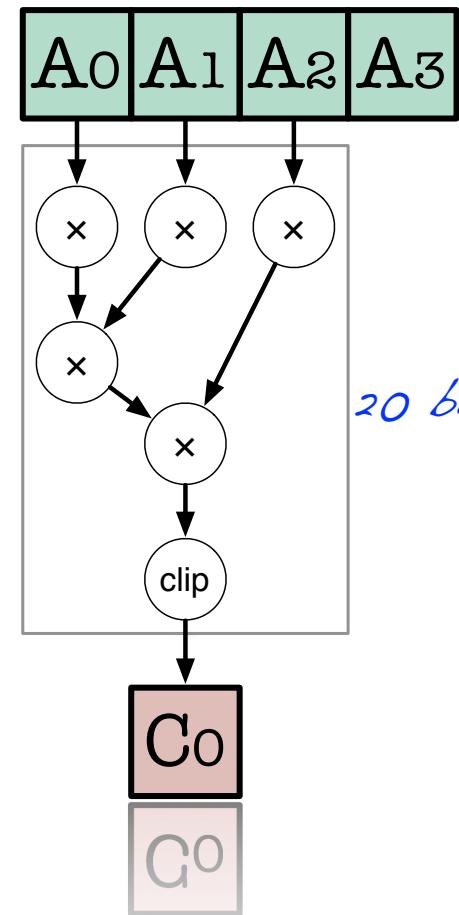
```
#define N 12
#define CLIP(a,b,c) (min(max(a,b)),c))

inline int Y_from_RGB(unsigned char arvb[3]){
    int V1 = (int)( 0.299 * (1 << N)) * (int)arvb[0];
    int V2 = (int)( 0.587 * (1 << N)) * (int)arvb[1];
    int V3 = (int)( 0.114 * (1 << N)) * (int)arvb[2];
    int Y = (v1 + v2 + v3) >> N;
    return CLIP(Y, 0, 255);
}
```

Le gain en terme de performance est significatif sur ce calcul
plus gain en terme d'occupation des registres du UP).

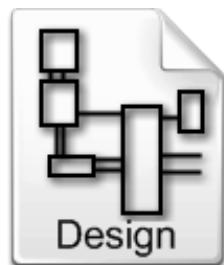
- Cout matériel de l'opérateur : 135 LUTs
- chemin critique : 6,805 ns
- facteur d'accélération max. : [x15 ~ x19]

Cette approche est aussi efficace pour les calculs arithmétiques.



* Chiffres donnés pour une cible Virtex-4 (XC4VSX53-10)

Les différentes manières d'enrichir son ASIP (extension)



Processor
+ specific
instructions
+ dedicated ALUs

Lower processor integration



One clock cycle instructions
=> limited instruction complexity



Pipeline instructions
=> 1 clock cycle < latency < 2 clock cycles
=> Limited instruction complexity

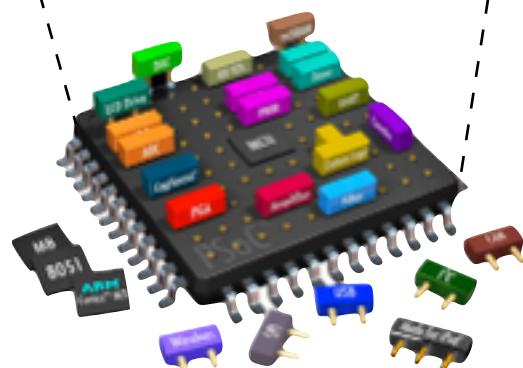


Sequential instructions
=> Not pipelined (latency > 1 clock cycle)
=> Freeze computation datapath

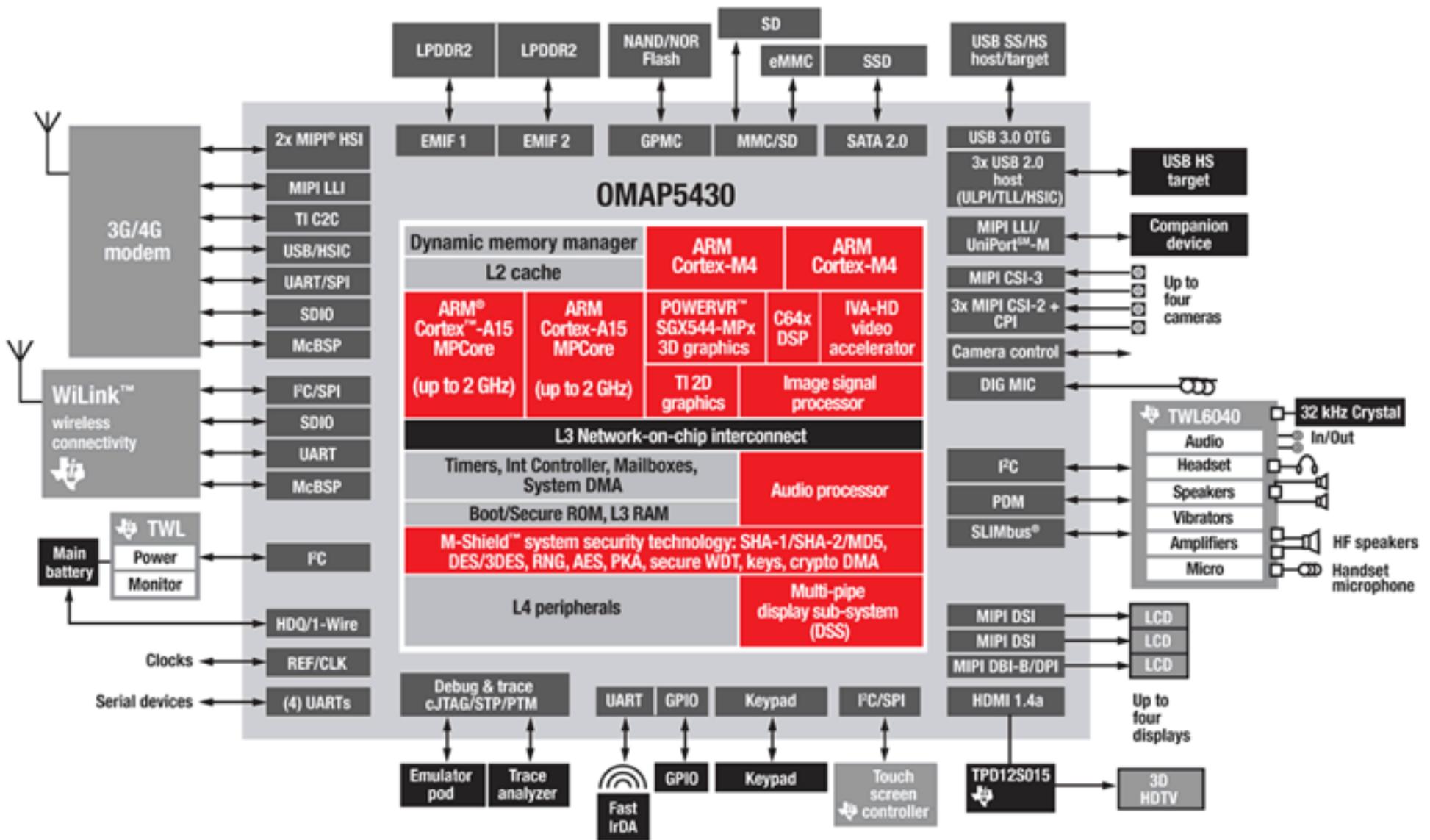


Complex hardware co-processors
=> Concurrent execution.
=> Processor can process other data.
=> Data transfer using MOVE like instructions

Vers la conception de systèmes complexes (SoC)

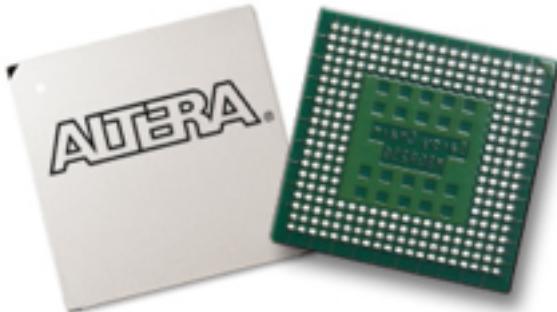
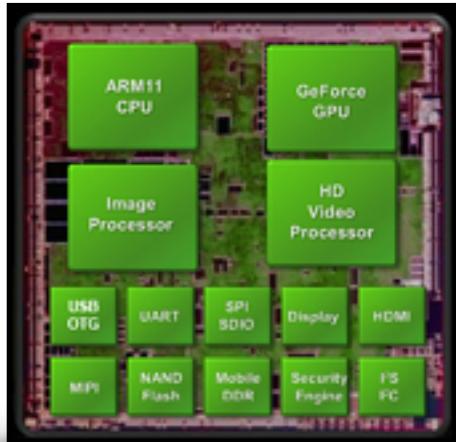


Quelques exemples de SoC provenant du monde réel (1/2)

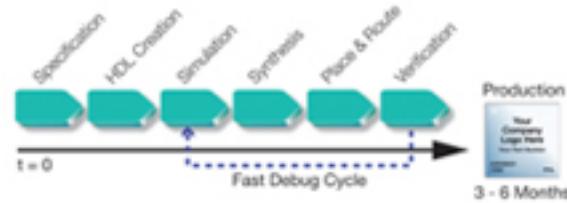


The 28 nanometer OMAP 5 applications processors feature up to 3x processing performance and five-fold 3D graphics improvement compared to OMAP 4 platform. In addition, OMAP 5 reduces average power reduction by almost 60%. TI's OMAP 5 platform will sample in the second half of 2011, with devices on the market in the second half of 2012. The OMAP5430 processor is offered in a 14x14mm Package-on-Package (PoP) with LPDDR2 memory support. The OMAP5432 processor is offered in a 17x17mm BGA package with DDR3/DDR3L memory support

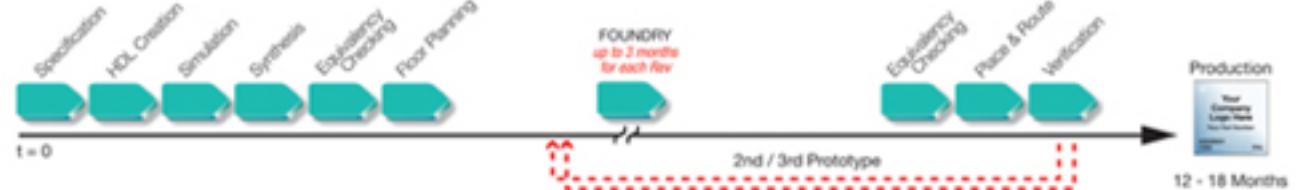
Vers la conception de systèmes complexes flexibles (SoPC)



FPGA & cSoC Time to Market



ASSP / ASIC Time to Market

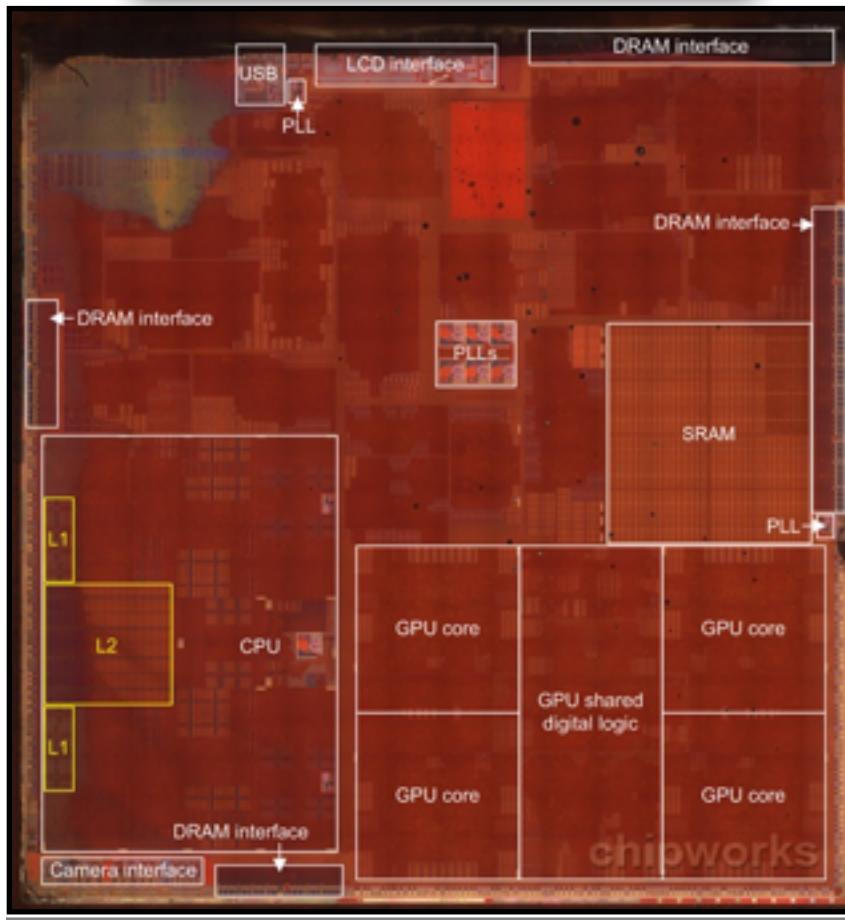


Un SoPC est un SoC implanté dans un FPGA

Intéressant pour le prototypage et les petites séries
Nouvelles opportunités: reconf. dynamique partielle

Quelques exemples de SoC provenant du monde réel (2/2)

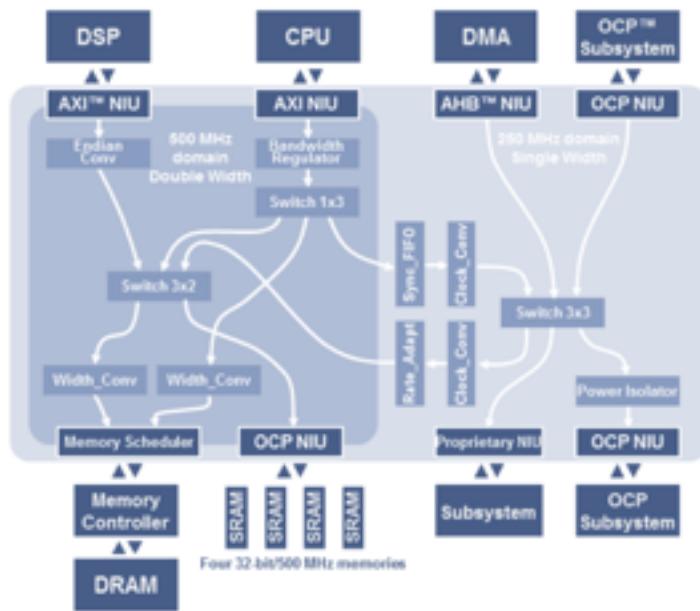
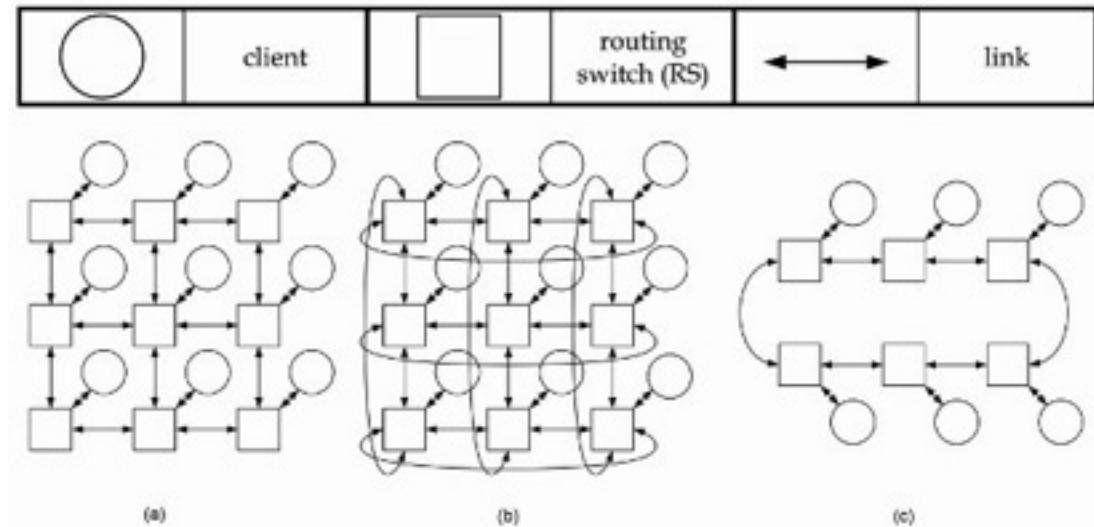
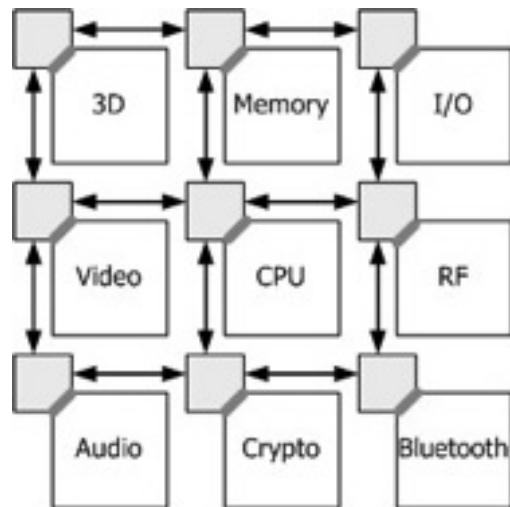
SoC A7 - iPhone 5S



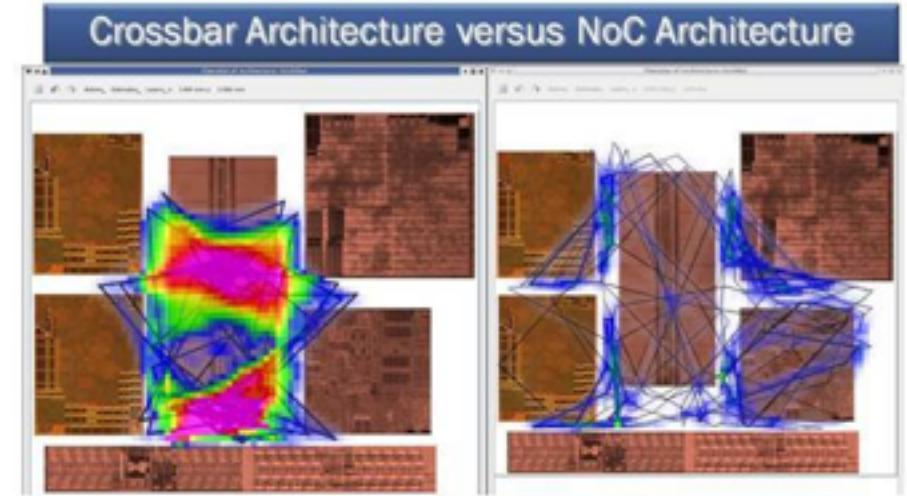
Intégrer tant d'IPs introduit
inévitablement des problèmes...



Faire transiter les données est une tâche complexe !



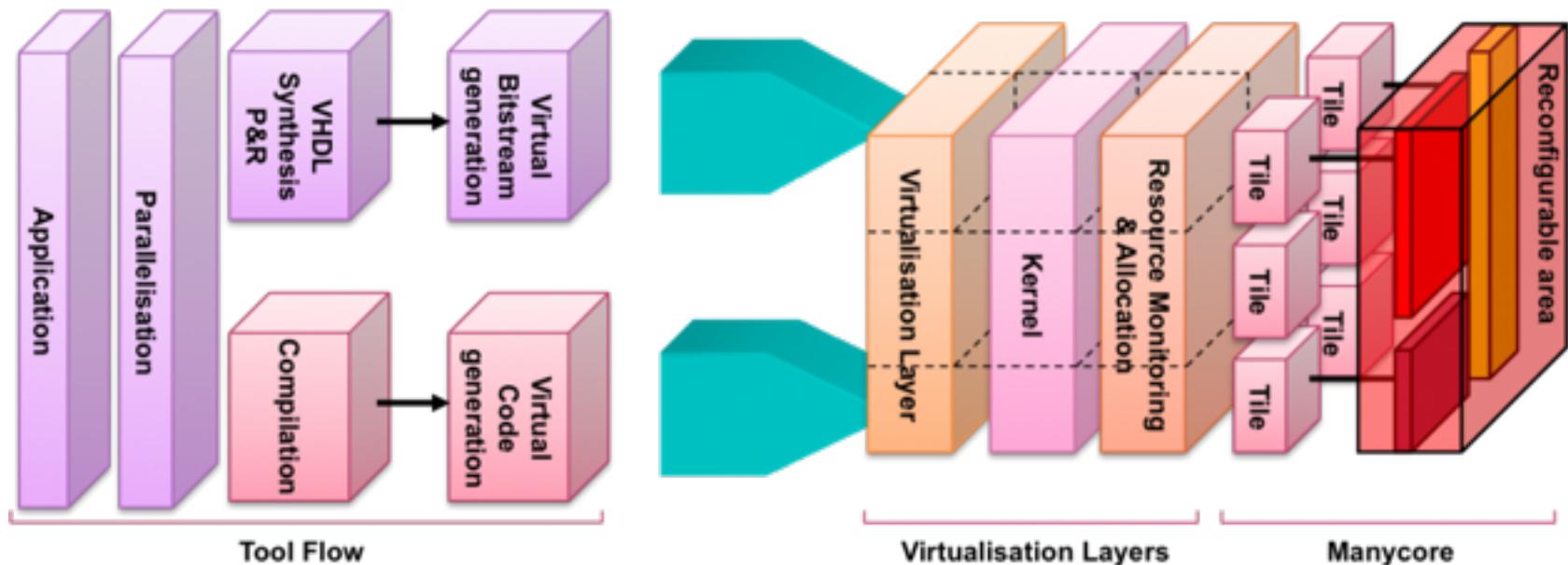
Reduced Wire Routing Congestion with Arteris FlexNoC



FlexNoC Datapath and Wire Density View in action

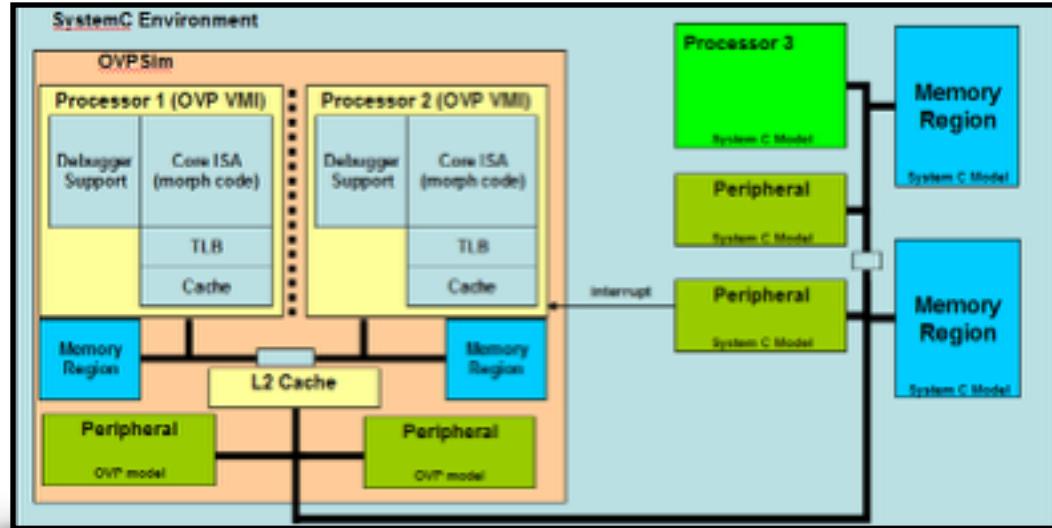
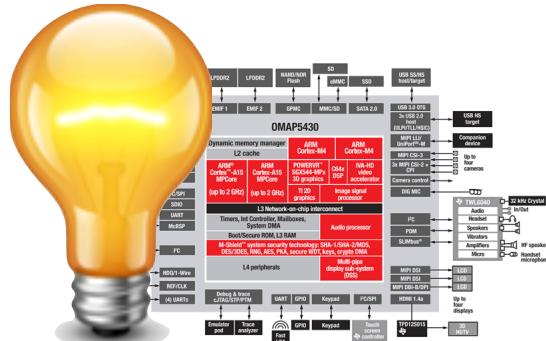
Méthodologie de conception de systèmes complexes

A major challenge in computing is to leverage multi-core technology to develop energy-efficient high performance systems. This is critical for embedded systems with a very limited energy budget as well as for supercomputers in terms of sustainability. Moreover the efficient programming of multi-core architectures, as we move towards manycores with more than a thousand cores predicted by 2020, remains an unresolved issue. The FlexTiles project will define and develop an energy-efficient yet programmable heterogeneous manycore platform with self-adaptive capabilities.



Modélisation & plateformes virtuelles

- Les SoCs sont complexes,
- Nécessité de modéliser les SoC avant de les concevoir,
 - Validation fonctionnelle;
 - Estimation/validation des perfs.
 - Exploration architecturale;
 - Raffinements;
- Hardware + software + OS ...



Benchmark	Altera Nios II		ARM32		Imagination MIPS32		
	Simulated Instructions	Run time	Simulated MIPS	Simulated Instructions	Run time	Simulated MIPS	Run time
Impact	3,075,857,171	2.52s	1225	6,105,795,956	4.79s	1277	9,114,621,292
Dhrystone	1,810,082,387	1.10s	1547	2,259,075,259	2.32s	974	1,795,088,667
Whetstone	5,850,087,389	3.20s	1789	1,185,353,551	1.04s	1140	1,890,420,882
peakSpeed2	22,000,013,458	3.11s	7097	22,400,000,766	4.67s	4807	22,800,003,853
Xilinx MicroBlaze							
Benchmark	Simulated Instructions	Run time	Simulated MIPS	Simulated Instructions	Run time	Simulated MIPS	Run time
Impact	6,306,275,159	3.77s	1699	69,945,059	1.01s	594*	1,558,856,686
Dhrystone	3,770,116,748	2.61s	1450	3,038,061,475	2.79s	1086	1,598,084,345
Whetstone	27,109,532,666	13.23s	2054	488,724,420	0.64s	759*	2,113,926,542
peakSpeed2	20,000,013,433	5.76s	3826	81,300,003,854	3.73s	3011	17,100,018,875
ARM AArch64							
Benchmark	Simulated Instructions	Run time	Simulated MIPS	Simulated Instructions	Run time	Simulated MIPS	Run time
Impact	6,306,275,159	3.77s	1699	69,945,059	1.01s	594*	1,558,856,686
Dhrystone	3,770,116,748	2.61s	1450	3,038,061,475	2.79s	1086	1,598,084,345
Whetstone	27,109,532,666	13.23s	2054	488,724,420	0.64s	759*	2,113,926,542
peakSpeed2	20,000,013,433	5.76s	3826	81,300,003,854	3.73s	3011	17,100,018,875
PowerPC							
Benchmark	Simulated Instructions	Run time	Simulated MIPS	Simulated Instructions	Run time	Simulated MIPS	Run time
Impact	3,163,996,113	2.95s	1076	4,991,344,769	4.79s	1051	4,184,762,664
Dhrystone	2,205,058,239	1.71s	1260	6,470,733,701	4.01s	1603	3,755,062,476
Whetstone	6,424,865,755	3.97s	1622	10,296,940,531	7.41s	1393	7,883,567,047
peakSpeed2	22,400,002,937	5.6s	4007	22,400,007,589	3.53s	6364	22,000,002,100
Renesas v850							
Benchmark	Simulated Instructions	Run time	Simulated MIPS	Simulated Instructions	Run time	Simulated MIPS	Run time
Impact	3,163,996,113	2.95s	1076	4,991,344,769	4.79s	1051	4,184,762,664
Dhrystone	2,205,058,239	1.71s	1260	6,470,733,701	4.01s	1603	3,755,062,476
Whetstone	6,424,865,755	3.97s	1622	10,296,940,531	7.41s	1393	7,883,567,047
peakSpeed2	22,400,002,937	5.6s	4007	22,400,007,589	3.53s	6364	22,000,002,100
Synopsys ARC							
Benchmark	Simulated Instructions	Run time	Simulated MIPS	Simulated Instructions	Run time	Simulated MIPS	Run time
Impact	3,163,996,113	2.95s	1076	4,991,344,769	4.79s	1051	4,184,762,664
Dhrystone	2,205,058,239	1.71s	1260	6,470,733,701	4.01s	1603	3,755,062,476
Whetstone	6,424,865,755	3.97s	1622	10,296,940,531	7.41s	1393	7,883,567,047
peakSpeed2	22,400,002,937	5.6s	4007	22,400,007,589	3.53s	6364	22,000,002,100

All measurements on 3.60GHz Intel i7-3770, Linux, OVPsim 20140427.0

* hardware floating point instructions

Sans maîtrise, la puissance n'est rien... [bis]

- Du point de vue du concepteur, changer de système nécessite la réécriture du code...

→ Problème de flexibilité, maintenabilité !

- Développement de couches d'abstraction permettant l'exploitation de la puissance disponible:

→ Le langage OpenCL

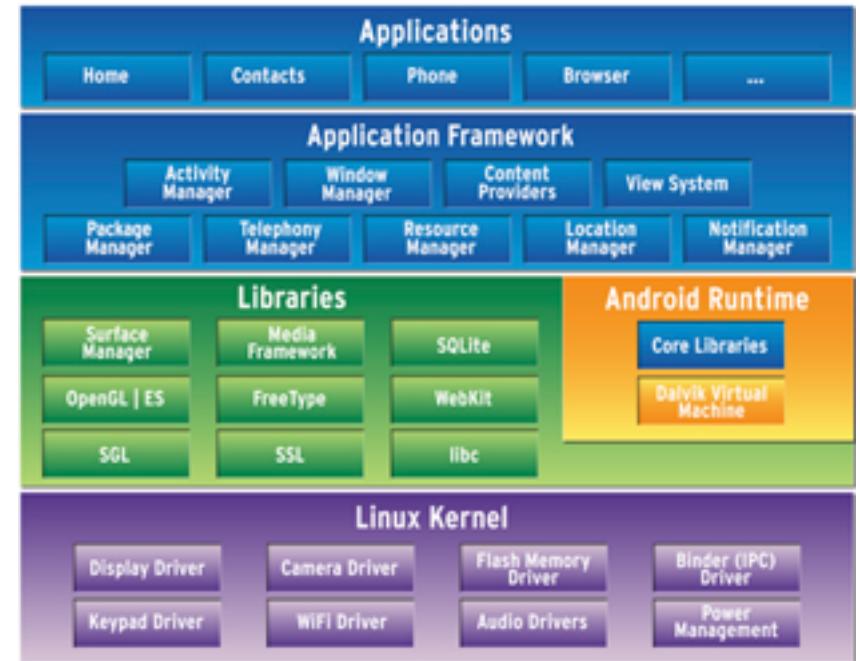


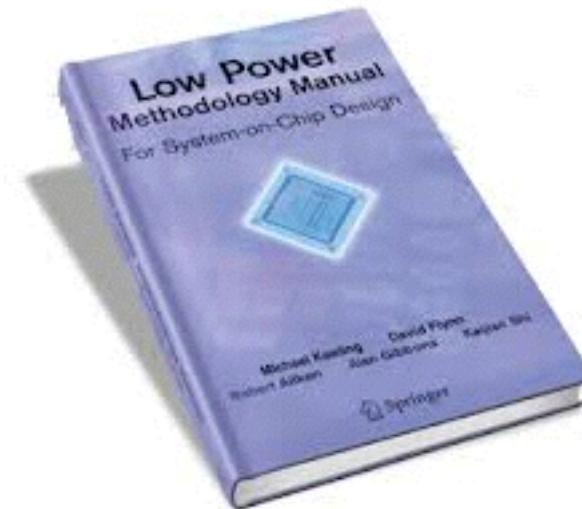
Fig 1. The Android system architecture uses a standard software-stack approach with Linux as its base.



La consommation d'énergie dans les systèmes embarqués

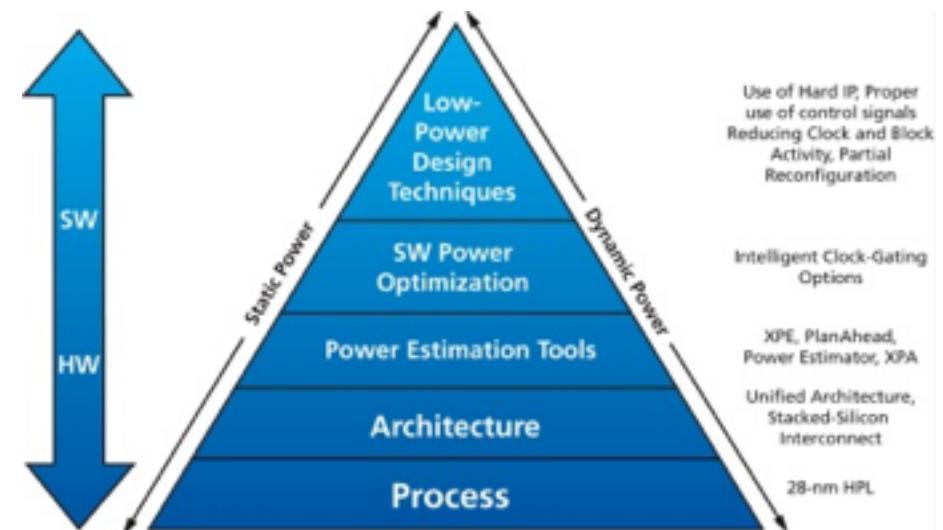
● Deux composantes:

- Consommation statique,
 - ▶ Proportionnel à la taille du circuit,
- Consommation dynamique
 - ▶ Liée à la commutation des « bits »,



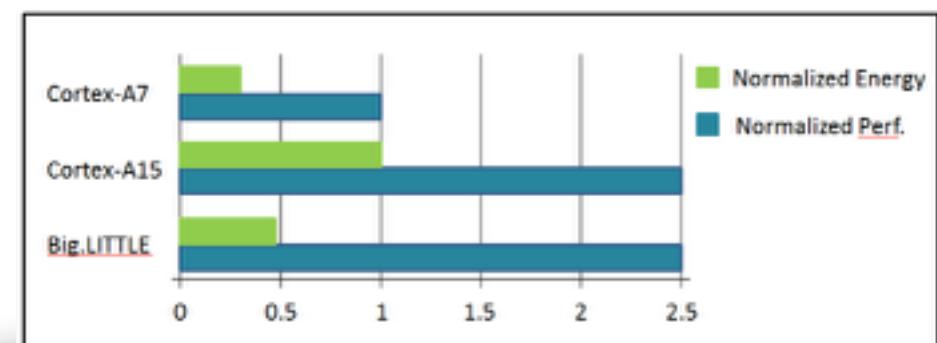
● L'efficacité énergétique s'obtient par un travail à tous les niveaux,

- Choix des algorithmes,
- Accès aux données (software),
- Organisation du circuit (VHDL),
- Technologie d'implantation,



Low-power, exemple de l'architecture ARM (big.LITTLE)

- Approche originale développée par ARM,
 - 2 jeux de processeurs dans le même SoC,
 - Execution mutuellement exclusive sur ces 2 jeux de processeurs;
- En fonction du workload les taches s'exécutent sur l'un ou l'autre des processor set.
- La difficulté vient de:
 - la gestion des coeurs,
 - la migration des taches.
- Perte de contrôle sur le fonctionnement du système !



Le codage CSD pour minimiser la consommation

The canonical signed digit (CSD) representation [II] assigns a separate sign to each digit: 0, 1 and $\bar{1}$ ($= -1$). Its goal is to minimise the number of non-zero digits.

By encoding the filter coefficients with CSD, the filter output can be computed using a reduced amount of hardware, since multiplications by zero are simply not implemented.

As an example, consider the multiplication by 15: since $15 = 2^3 + 2^2 + 2^1 + 2^0 = (00111)_2$, this operation in binary arithmetics would require three shifts and three additions; while using CSD we can write $15 = 24 - 20 = (01000\bar{1})_2$, and we implement the same operation using only one shifter and one subtractor (Fig. 3).

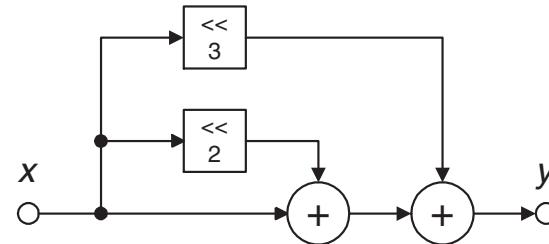


Fig. 2. Multiplication by 13 implemented with shifters and adders

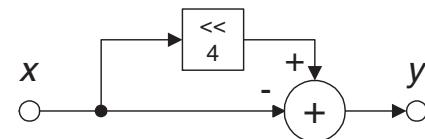
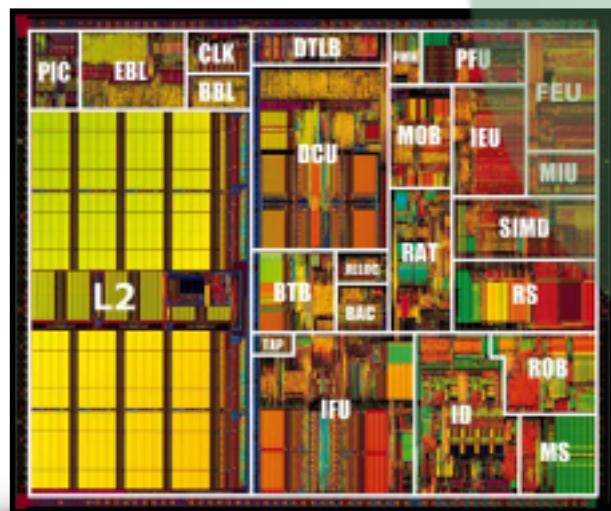
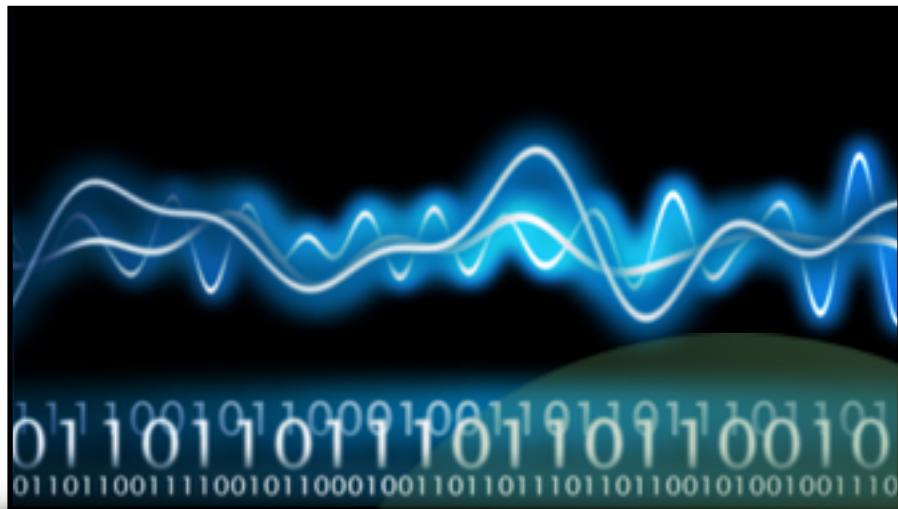


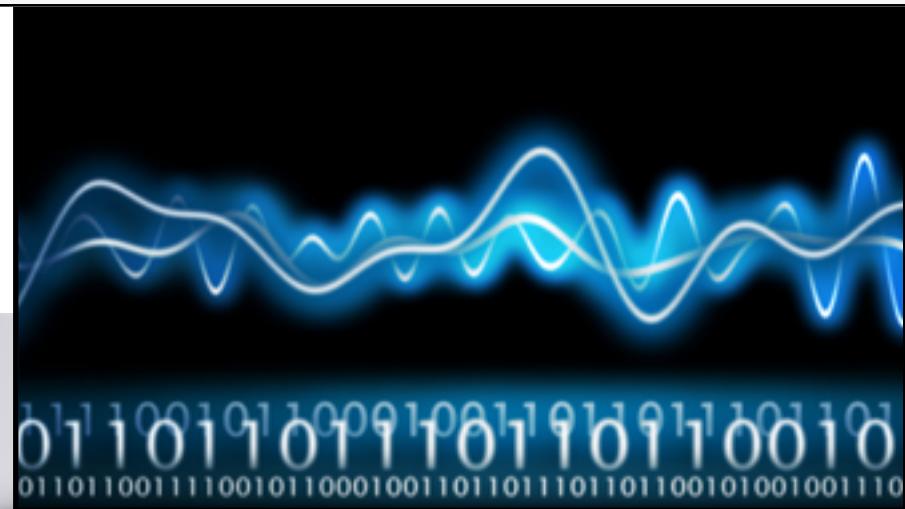
Fig. 3. Multiplication by 15 implemented with one shifter and one subtractor

Des applications aux architectures...

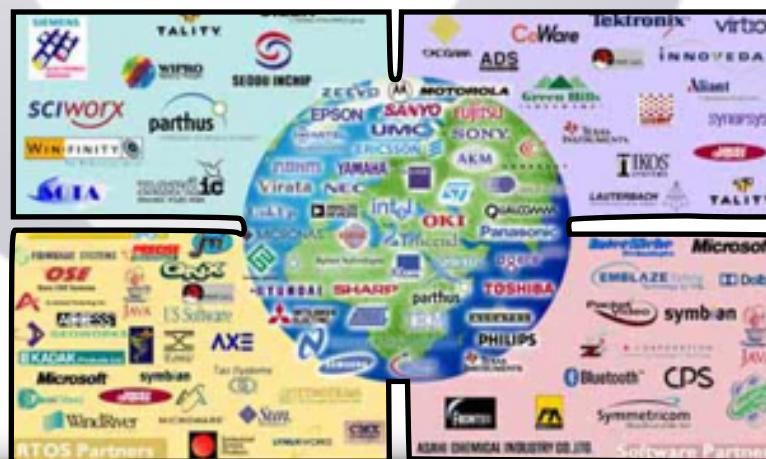
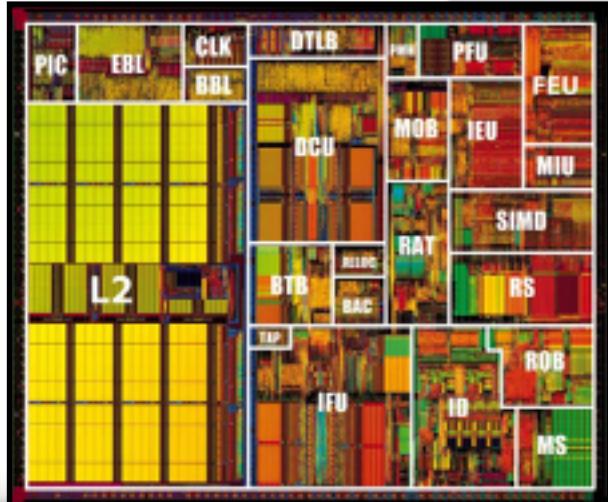


On part de l'algorithme et on essaye d'optimiser son implantation sur l'architecture matérielle choisie (à l'aide d'outils et de bibliothèques).

Des architectures aux applications...



On conçoit l'architecture matérielle
(ou des outils) vis à vis d'une application
ou d'un domaine applicatif



The
End