Progress in software implementations of ECC decoders

Bertrand LE GAL, Camille LEROUX and Christophe JEGO

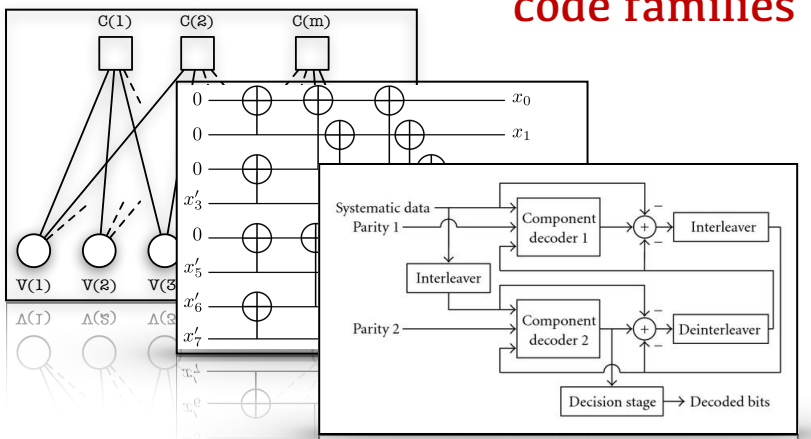
IMS laboratory, CNRS UMR 5218
Digital Circuits and Systems team
Bordeaux-INP, University of Bordeaux
France

GdR ISIS: Energy-efficiency in Error-Correction Coding
June 8th 2016 @ Télécom ParisTech

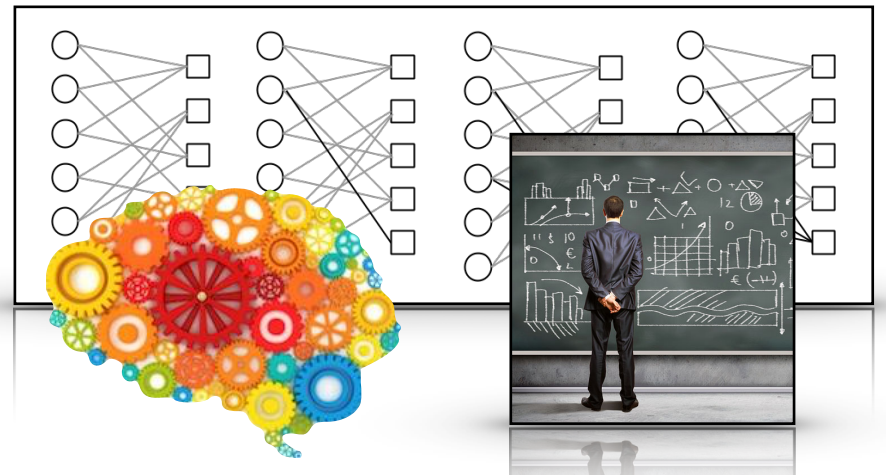
firstname.lastname@ims-bordeaux.fr

Historically, software implementations were used for…

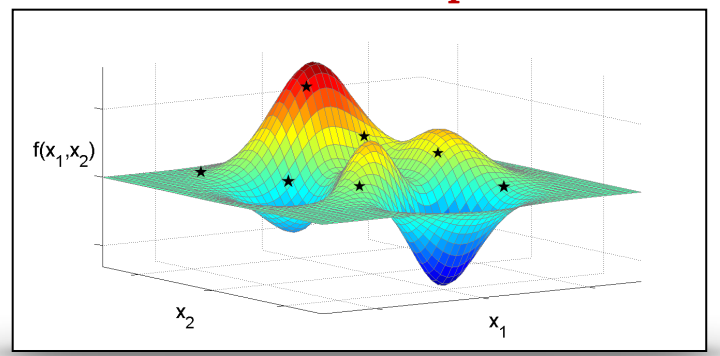
Validate and compare error correction code families



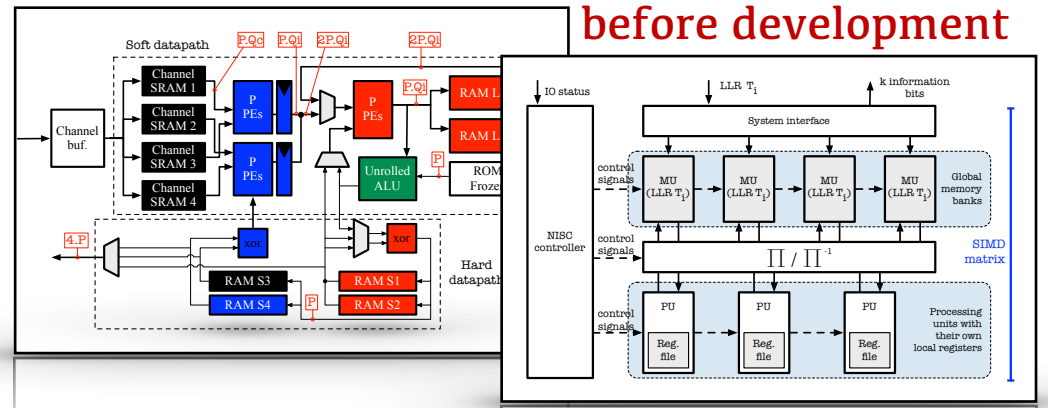
Benchmarking of decoding algorithms or code construction techniques



Parameter optimization



Estimation of hardware design performances before development



Currently they can fulfill others requirements

Current software decoders are at least as fast as many Hw circuits

802.11e (2304,1152), Layered, OMS, (6, 8, 6)
SNR = 2.30 | BER = 1.61e-07 | FER = 1.21e-06 | RUNTIME = 30’

Provide design and runtime flexibilities

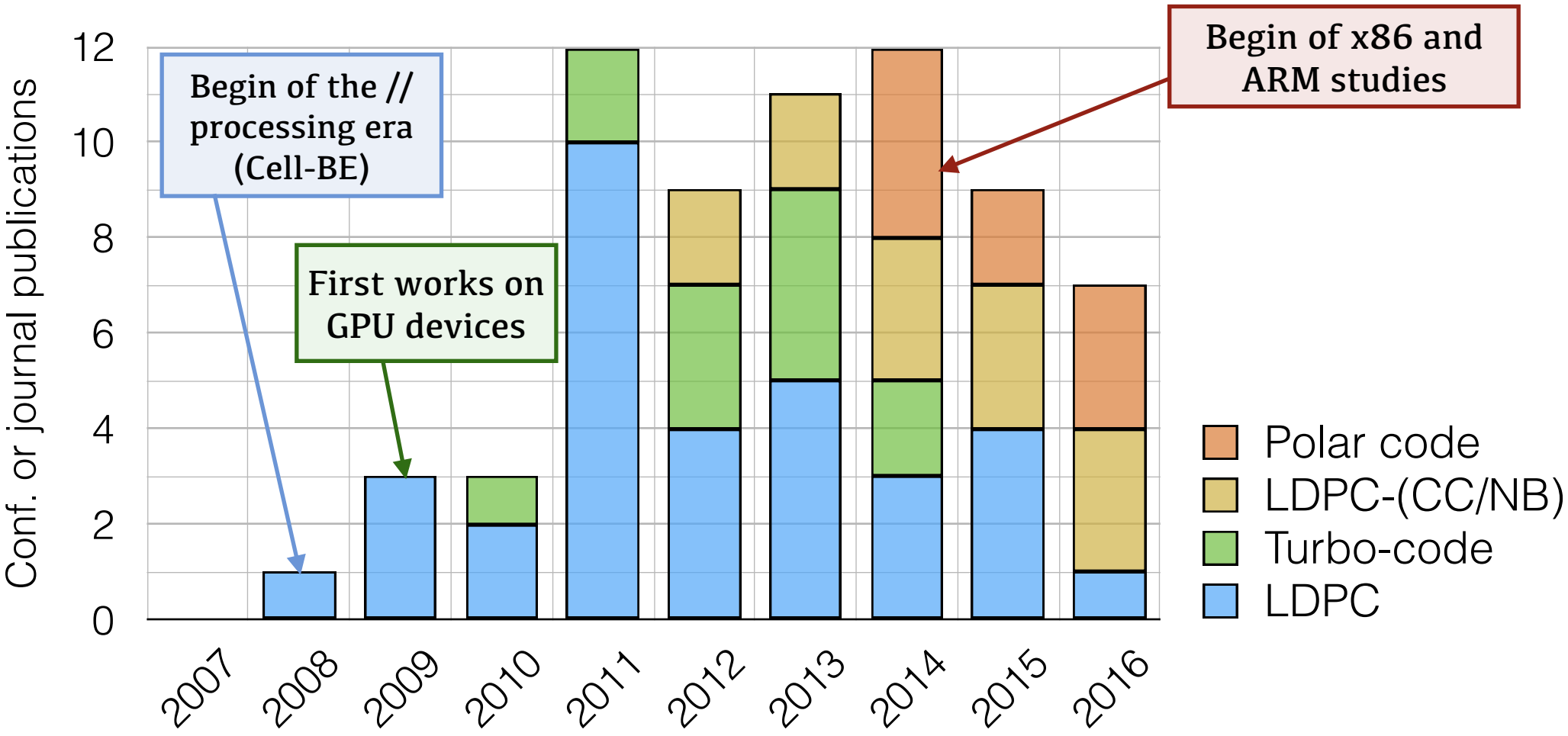
Energy consumption is challenge for industrials

Currently, compatible with some industrial use cases.

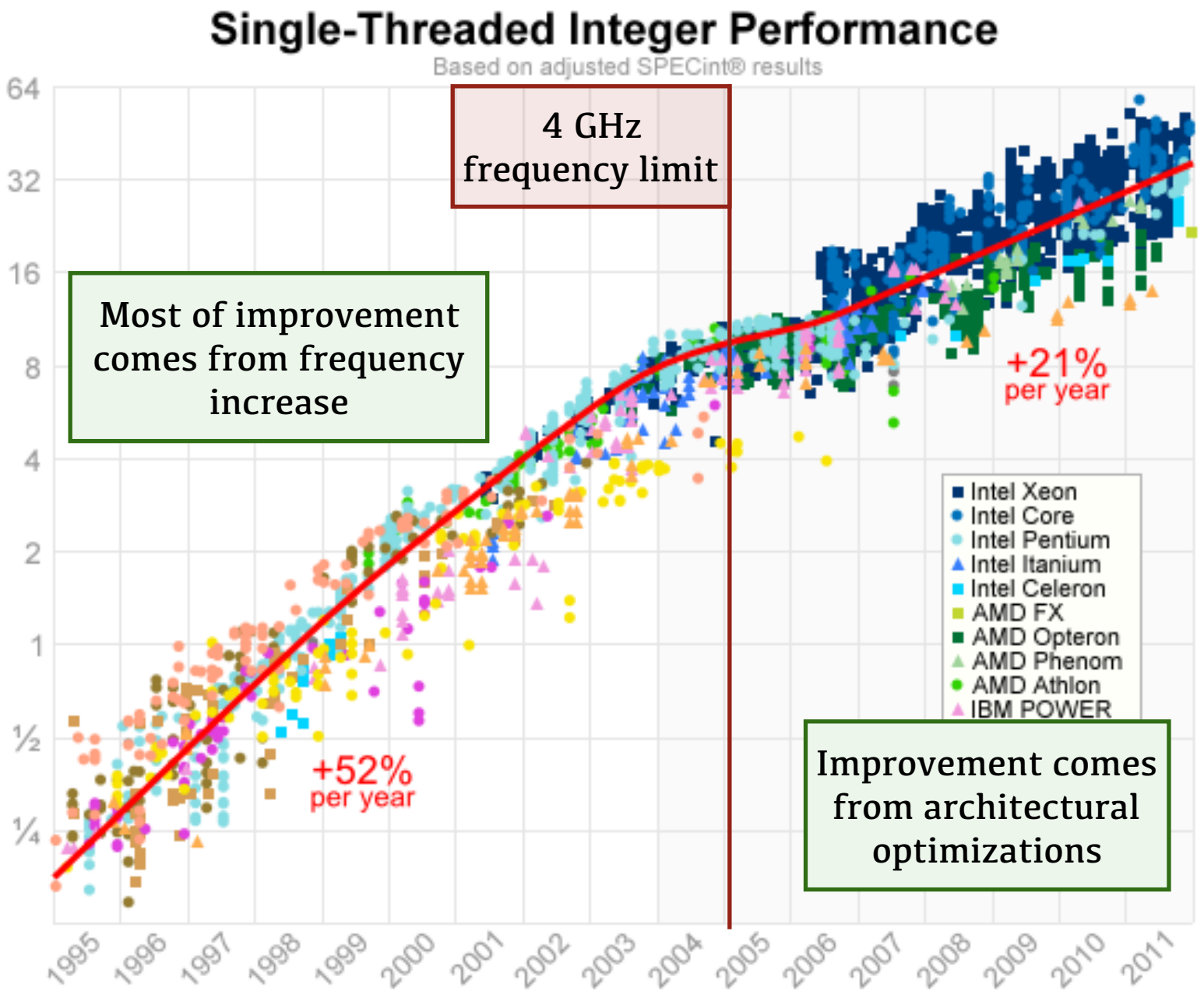
Software implementation is an active research topic

More than 70 research publications in the field in 8 years

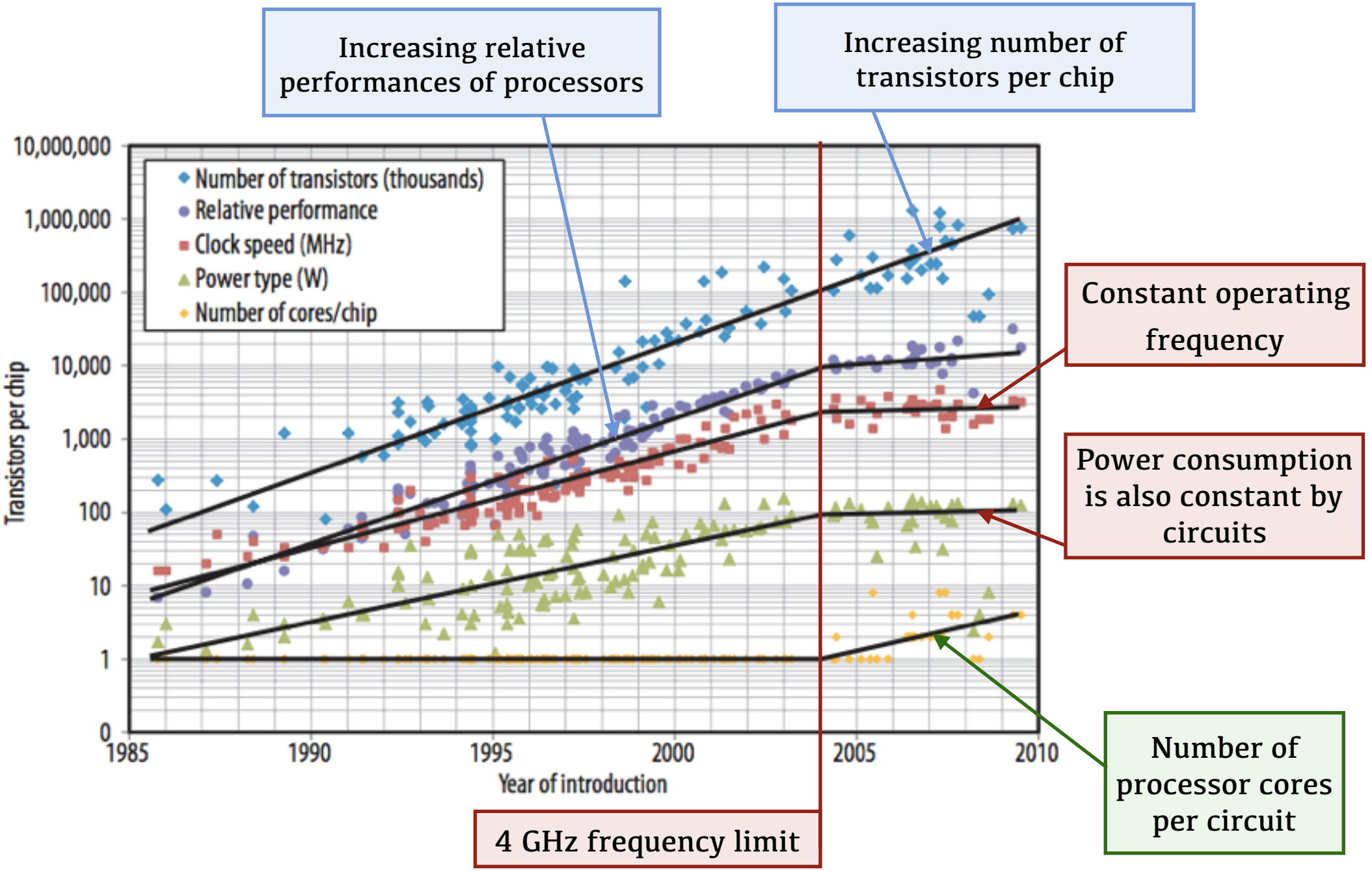
- Many ECC families were implemented on programmable targets during last decade with various performance levels.



Evolution of integer performance of processor cores

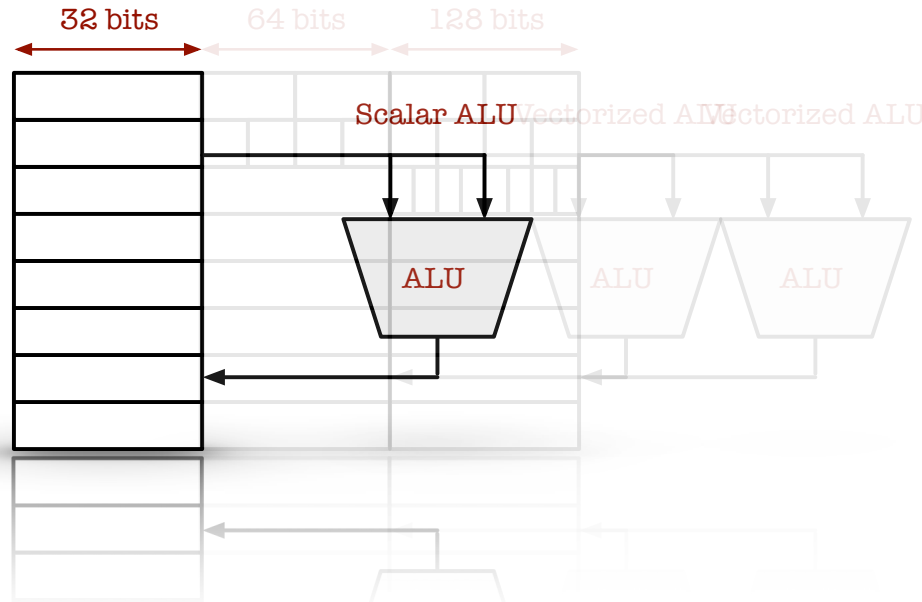


Evolution of processor target features



Evolution of the programmable architectures (CPUs)

- Evolution of INTEL CPUs
  - 1993: pipeline scalar processors,
  - 1995 : pipeline vector processor (SIMD=64b),
  - 2000-2006 pipeline superscalar vector processor (SIMD=128b),
  - 2006 : pipeline superscalar vector multi-processors
  - 2016 : pipeline superscalar vector multi-processors (SIMD=256b or 512b).
- Modern embedded processors (ARM, MIPS) offer same features,
- Power consumption,
  - Frequency and voltage scaling,
  - Switching is possible at runtime.



During 1 clock cycle, a Pentium [8 W => 15 W]
One integer or float **computation** or load/store

During 1 clock cycle, a Pentium-MMX [20 W]
2x32b or **8x8b** int or 2 float computations => **8** 8b-instr.

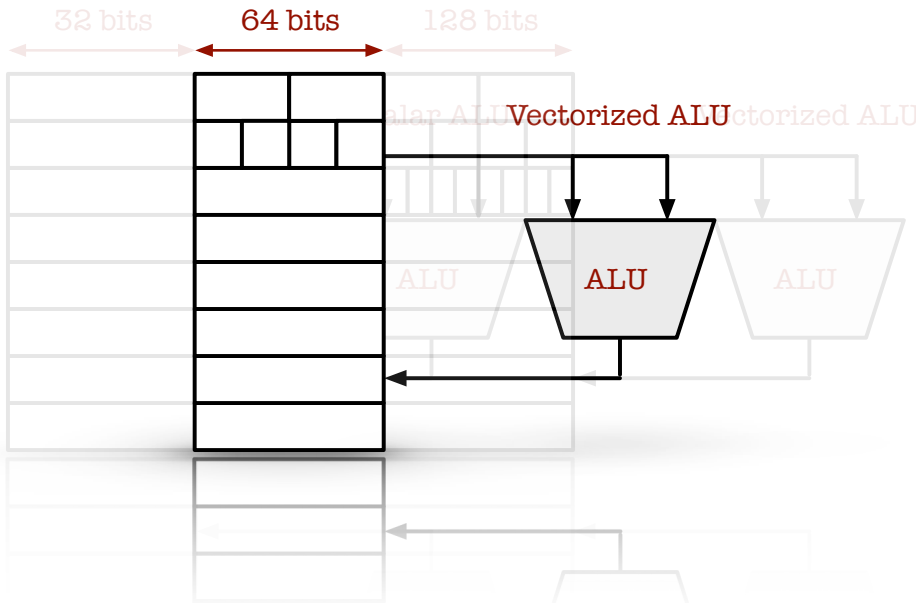
During 1 clock cycle, a Pentium-II,III,IV [200 W]:
up to 3 instr. on **128b** [**16x8b**] => **96** 8b-instr.

During 1 clock cycle, a Core 2/i3 … [< 70 W]
2 cores x up to **6 instr**. on 128b [16x8b] => **192** 8b-instr.

During 1 clock cycle a Core i7 (Haswell) …:
6 cores x up to **8 instr**. on **256b** [**32x8b**] => **1536** 8b-instr.

Evolution of the programmable architectures (CPUs)

- Evolution of INTEL CPUs
  - 1993: pipeline scalar processors,
  - 1995 : pipeline vector processor (SIMD=64b),
  - 2000-2006 pipeline superscalar vector processor (SIMD=128b),
  - 2006 : pipeline superscalar vector multi-processors
  - 2016 : pipeline superscalar vector multi-processors (SIMD=256b or 512b).
- Modern embedded processors (ARM, MIPS) offer same features,
- Power consumption,
  - Frequency and voltage scaling,
  - Switching is possible at runtime.



During 1 clock cycle, a Pentium [8 W => 15 W]
One integer or float **computation** or load/store

During 1 clock cycle, a Pentium-MMX [20 W]
2x32b or **8x8b** int or 2 float computations => **8** 8b-instr.

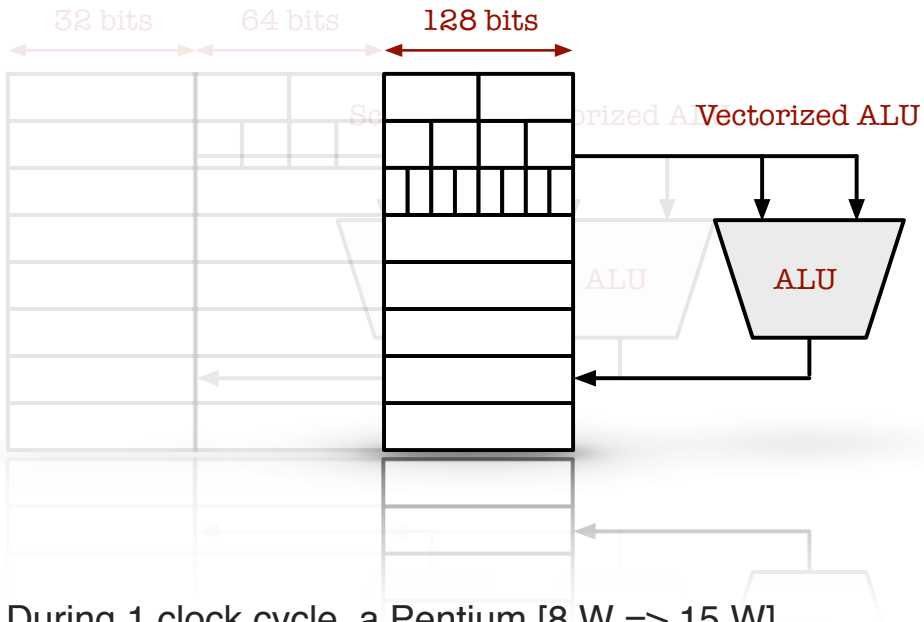
During 1 clock cycle, a Pentium-II,III,IV [200 W]:
up to 3 instr. on **128b** [**16x8b**] => **96** 8b-instr.

During 1 clock cycle, a Core 2/i3 … [< 70 W]
2 cores x up to **6 instr**. on 128b [16x8b] => **192** 8b-instr.

During 1 clock cycle a Core i7 (Haswell) …:
6 cores x up to **8 instr**. on **256b** [**32x8b**] => **1536** 8b-instr.

Evolution of the programmable architectures (CPUs)

- Evolution of INTEL CPUs
  - 1993: pipeline scalar processors,
  - 1995 : pipeline vector processor (SIMD=64b),
  - 2000-2006 pipeline superscalar vector processor (SIMD=128b),
  - 2006 : pipeline superscalar vector multi-processors
  - 2016 : pipeline superscalar vector multi-processors (SIMD=256b or 512b).
- Modern embedded processors (ARM, MIPS) offer same features,
- Power consumption,
  - Frequency and voltage scaling,
  - Switching is possible at runtime.



During 1 clock cycle, a Pentium [8 W => 15 W]
One integer or float **computation** or load/store

During 1 clock cycle, a Pentium-MMX [20 W]
2x32b or **8x8b** int or 2 float computations => **8** 8b-instr.

During 1 clock cycle, a Pentium-II,III,IV [200 W]:
up to 3 instr. on **128b** [**16x8b**] => **96** 8b-instr.

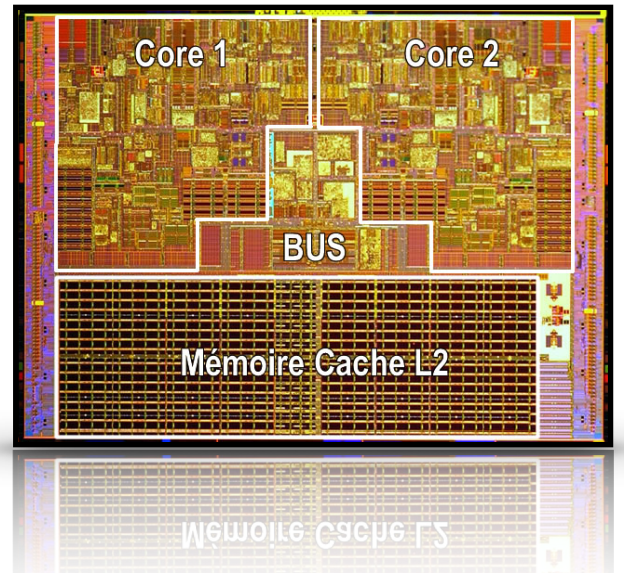
During 1 clock cycle, a Core 2/i3 … [< 70 W]
2 cores x up to **6 instr**. on 128b [16x8b] => **192** 8b-instr.

During 1 clock cycle a Core i7 (Haswell) …:
6 cores x up to **8 instr**. on **256b** [**32x8b**] => **1536** 8b-instr.

Evolution of the programmable architectures (CPUs)

- Evolution of INTEL CPUs
  - 1993: pipeline scalar processors,
  - 1995 : pipeline vector processor (SIMD=64b),
  - 2000-2006 pipeline superscalar vector processor (SIMD=128b),
  - 2006 : pipeline superscalar vector multi-processors
  - 2016 : pipeline superscalar vector multi-processors (SIMD=256b or 512b).
- Modern embedded processors (ARM, MIPS) offer same features,
- Power consumption,
  - Frequency and voltage scaling,
  - Switching is possible at runtime.

Core 2: power consumption < 70 Watts



During 1 clock cycle, a Pentium [8 W => 15 W]
One integer or float **computation** or load/store

During 1 clock cycle, a Pentium-MMX [20 W]
2x32b or **8x8b** int or 2 float computations => **8** 8b-instr.

During 1 clock cycle, a Pentium-II,III,IV [200 W]:
up to 3 instr. on **128b** [**16x8b**] => **96** 8b-instr.

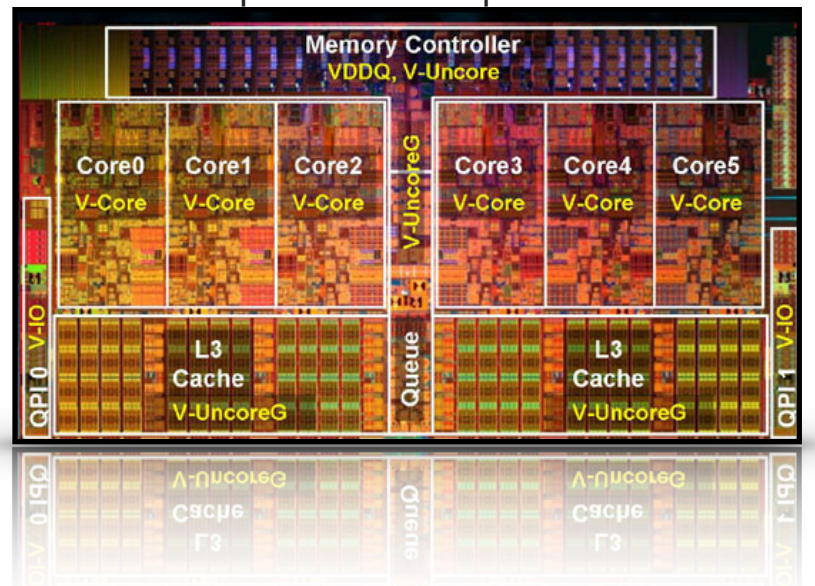
During 1 clock cycle, a Core 2/i3 … [< 70 W]
2 cores x up to **6 instr**. on 128b [16x8b] => **192** 8b-instr.

During 1 clock cycle a Core i7 (Haswell) …:
6 cores x up to **8 instr**. on **256b** [**32x8b**] => **1536** 8b-instr.

Evolution of the programmable architectures (CPUs)

- Evolution of INTEL CPUs
  - 1993: pipeline scalar processors,
  - 1995 : pipeline vector processor (SIMD=64b),
  - 2000-2006 pipeline superscalar vector processor (SIMD=128b),
  - 2006 : pipeline superscalar vector multi-processors
  - 2016 : pipeline superscalar vector multi-processors (SIMD=256b or 512b).
- Modern embedded processors (ARM, MIPS) offer same features,
- Power consumption,
  - Frequency and voltage scaling,
  - Switching is possible at runtime.

Core-i7 M: power consumption < 50 Watts



During 1 clock cycle, a Pentium [8 W => 15 W]
One integer or float **computation** or load/store

During 1 clock cycle, a Pentium-MMX [20 W]
2x32b or **8x8b** int or 2 float computations => **8** 8b-instr.

During 1 clock cycle, a Pentium-II,III,IV [200 W]:
up to 3 instr. on **128b** [**16x8b**] => **96** 8b-instr.

During 1 clock cycle, a Core 2/i3 … [< 70 W]
2 cores x up to **6 instr**. on 128b [16x8b] => **192** 8b-instr.

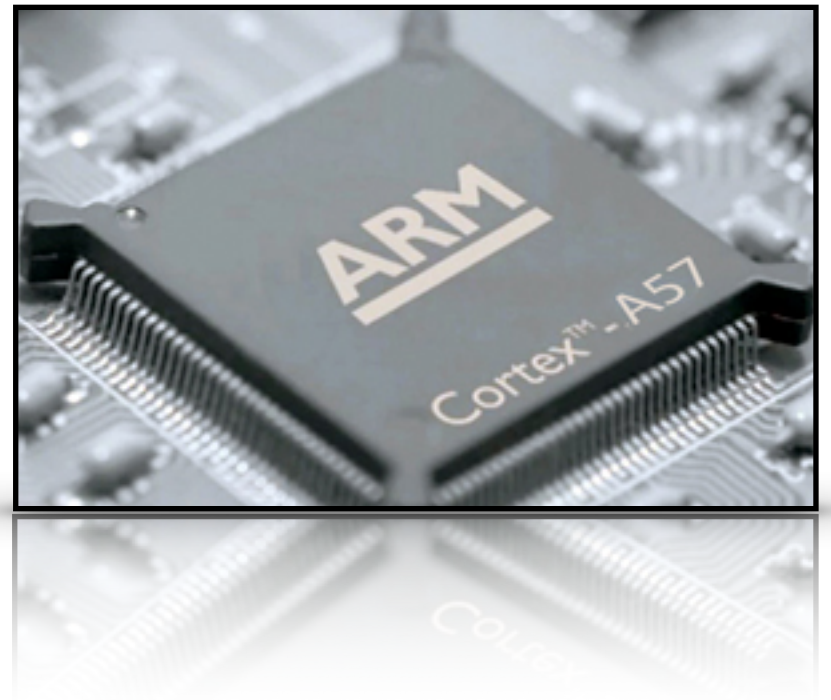
During 1 clock cycle a Core i7 (Haswell) …:
6 cores x up to **8 instr**. on **256b** [**32x8b**] => **1536** 8b-instr.

Evolution of the programmable architectures (CPUs)

- Evolution of INTEL CPUs
  - 1993: pipeline scalar processors,
  - 1995 : pipeline vector processor (SIMD=64b),
  - 2000-2006 pipeline superscalar vector processor (SIMD=128b),
  - 2006 : pipeline superscalar vector multi-processors
  - 2016 : pipeline superscalar vector multi-processors (SIMD=256b or 512b).
- Modern embedded processors (ARM, MIPS) offer same features,
- Power consumption,
  - Frequency and voltage scaling,
  - Switching is possible at runtime.

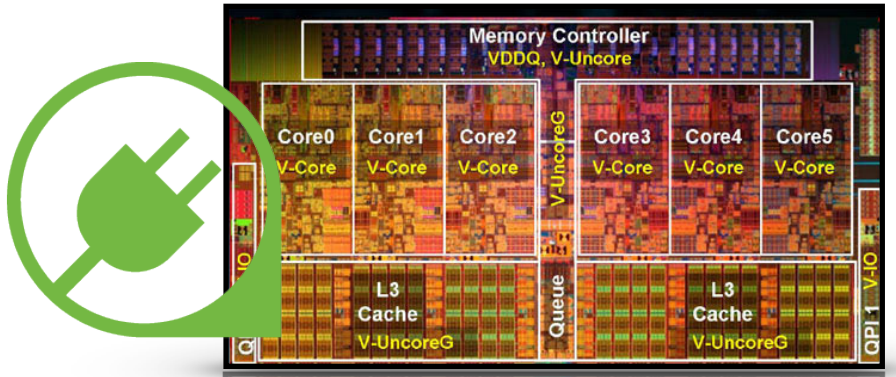
ARM A57 processor

- RISC 64 bits, up to 3 instr/cycle
- 4 Cores @ 2,3 GHz
- SIMD 128b NEON (32 registres)
- Power consumption < 4 Watts



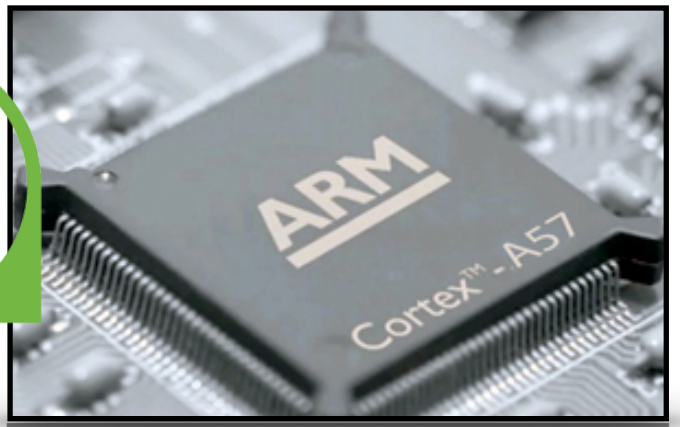
Evolution of the programmable architectures (CPUs)

- Evolution of INTEL CPUs
  - 1993: pipeline scalar processors,
  - 1995 : pipeline vector processor (SIMD=64b),
  - 2000-2006 pipeline superscalar vector processor (SIMD=128b),
  - 2006 : pipeline superscalar vector multi-processors
  - 2016 : pipeline superscalar vector multi-processors (SIMD=256b or 512b).
- Modern embedded processors (ARM, MIPS) offer same features,
- **Power consumption,**
  - Frequency and voltage scaling,
  - Switching is possible at runtime.



REDUCED ENERGY CONSUMPTION

Pentium 4, > 200 W for 1 core
Core-i7 M, < 50 W for 4 cores



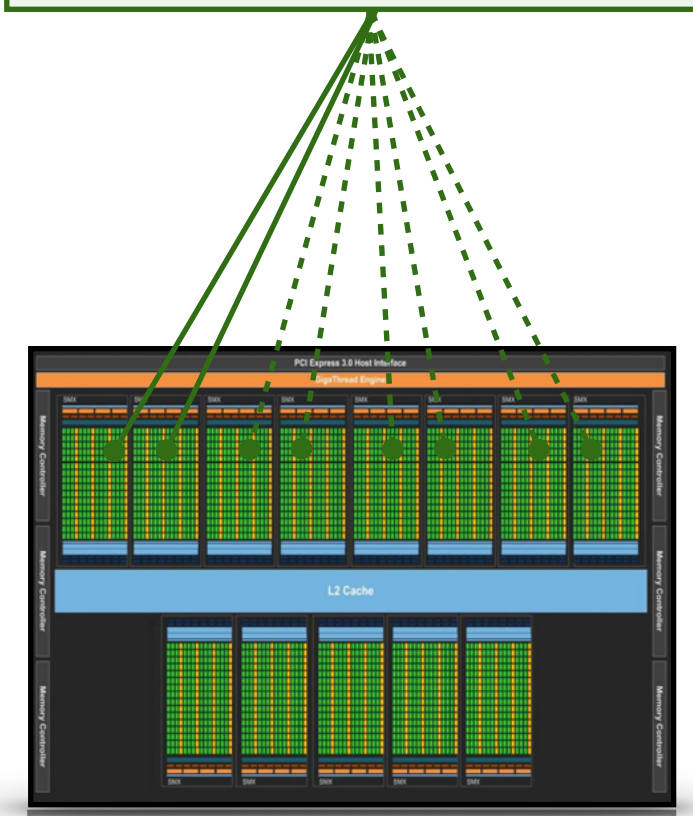
REDUCED ENERGY CONSUMPTION

Evolution of the programmable architectures (GPUs)

- GPU devices were initially developed for video applications,
- Hierarchical architecture,
  - GPU => Stream Processor => Cores
- Very large set of processing units,
  - One core = {ALU + FPU} unit
  - As or more more efficient for floating point computations than integer ones,
  - Efficient transcendental units,
- « Low » working frequency,
- Adapted to massively // applications without low latency constraints.

A GPU is composed of many processors named stream-processors (SMX) working in parallel

k40: 15 SMX

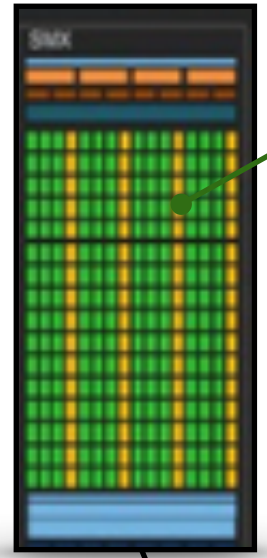


K40 : 245W - SM@875 MHz - 12GB GDDR5

[Lecture 1: an introduction to CUDA from Mike Giles]

Evolution of the programmable architectures (GPUs)

- GPU devices were initially developed for video applications,
- Hierarchical architecture,
  - GPU => Stream Processor => Cores
- Very large set of processing units,
  - One core = {ALU + FPU} unit
  - As or more more efficient for floating point computations than integer ones,
  - Efficient transcendental units,
- « Low » working frequency,
- Adapted to massively // applications without low latency constraints.



A stream processor contains 192 cores

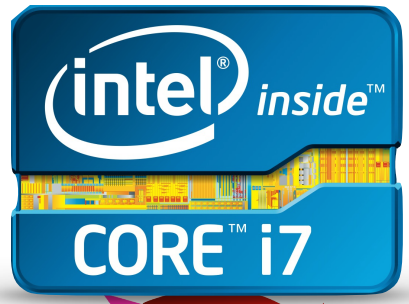
K40: 2880 cores

[192 cores / SMX]

K40 : 245W - SM@875 MHz - 12GB GDDR5

Evolution of the programmable architectures (GPUs)

- GPU devices were initially developed for video applications,
- Hierarchical architecture,
  - GPU => Stream Processor => Cores
- Very large set of processing units,
  - One core = {ALU + FPU} unit
  - As or more more efficient for floating point computations than integer ones,
  - Efficient transcendental units,
- « Low » working frequency,
- Adapted to massively // applications without low latency constraints.



Data and control transfer

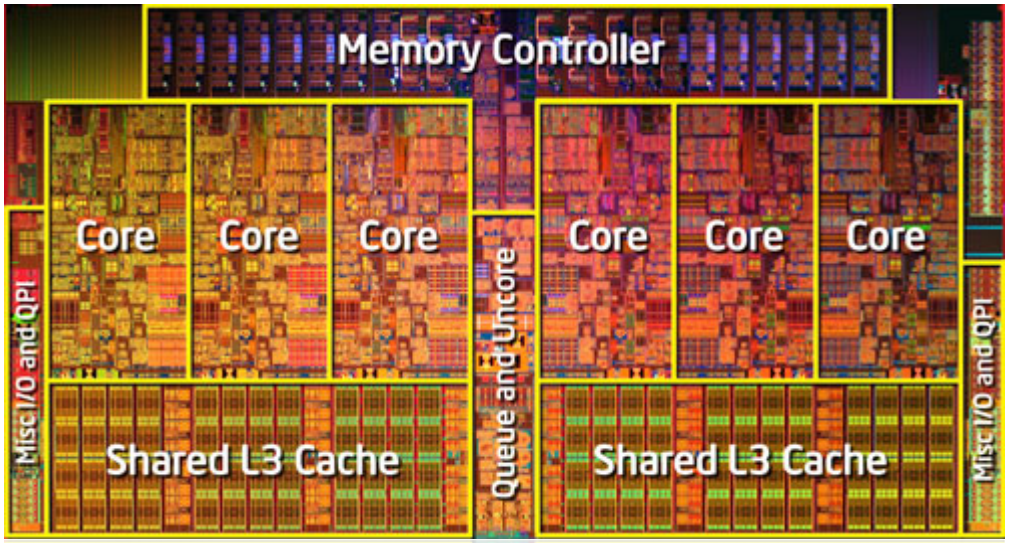
K40 : 245W - SM@875 MHz - 12GB GDDR5

The mapping challenges to reach high performances

- Efficiently implementing ECC on software programmable target is a challenging task,
- Less complexity than designing hardware circuit.
  - CPU and GPU have fixed architecture,
  - So, they add contraints to any implementation.
- Different programming models,
  - Difficult to transpose techniques GPU<=>CPU,
- Different languages and tools,
  - C/C++ for CPUs,
  - CUDA or Open-CL for GPUs.

The mapping challenges to reach high performances

- Efficiently implementing ECC on software programmable target is a challenging task,
- Less complexity than designing hardware circuit.
  - CPU and GPU have fixed architecture,
  - They add contraints to any implementation.
- Different programming models,
  - Difficult to transpose techniques GPU<=>CPU,
- Different languages and tools,
  - C/C++ for CPUs,
  - CUDA or Open-CL for GPUs.



Moderate computation parallelism

Efficient SIMD instructions

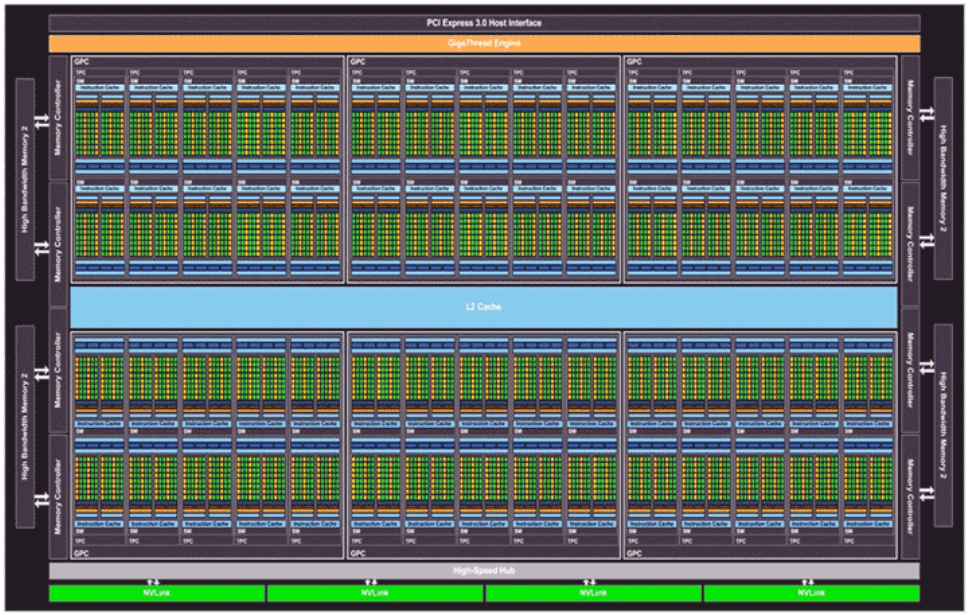
Moderate memory cache sizes

Difficult to manage conditional structures

Deep architecture knowledge required

The mapping challenges to reach high performances

- Efficiently implementing ECC on software programmable target is a challenging task,
- Less complexity than designing hardware circuit.
  - CPU and GPU have fixed architecture,
  - They add contraints to any implementation.
- Different programming models,
  - Difficult to transpose techniques GPU<=>CPU,
- Different languages and tools,
  - C/C++ for CPUs,
  - CUDA or Open-CL for GPUs.



High FPU computation parallelism

Can process conditional structures

Slow memory access (global memory)

Need aligned memory access patterns

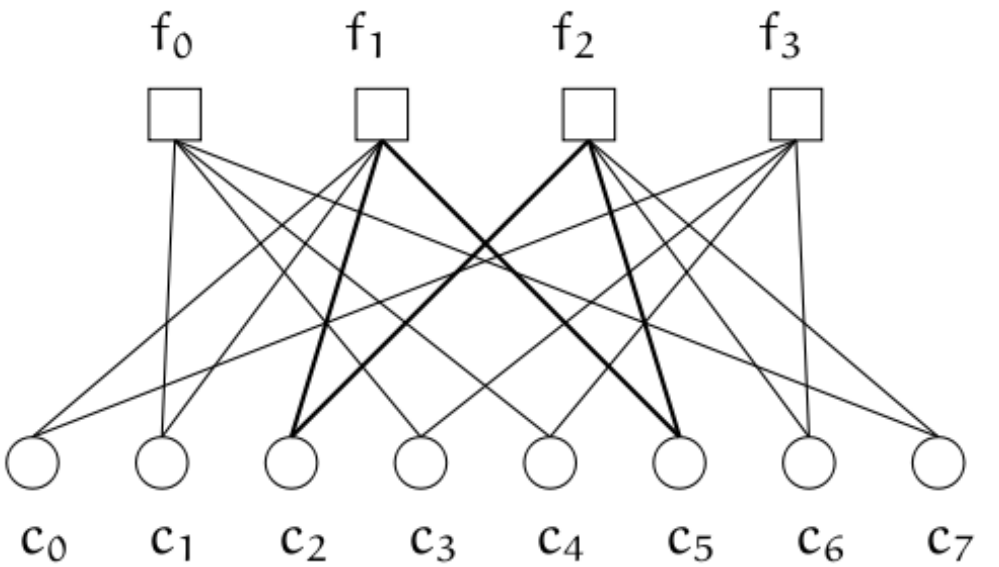
Quite slow transfer from/to host

The mapping challenges to reach high performances

- Efficiently implementing ECC on software programmable target is a challenging task,
- Less complexity than designing hardware circuit.
  - CPU and GPU have fixed architecture,
  - They add contraints to any implementation.
- Different programming models,
  - Difficult to transpose techniques GPU<=>CPU,
- Different languages and tools,
  - C/C++ for CPUs,
  - CUDA or Open-CL for GPUs.

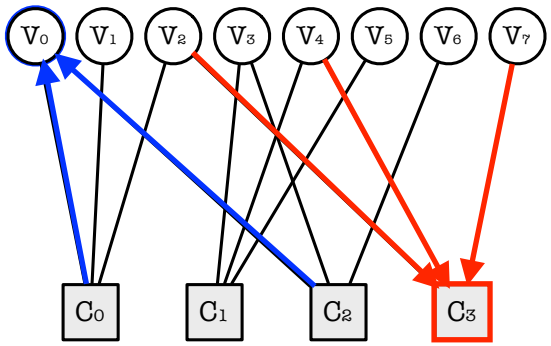
Case of study - The software LDPC decoders

- The most implemented ECC on multicore devices,
  - Loops in decoding algorithm are good candidates for parallelization,
  - The parallelism levels suitable for hardware design were explored,
  - Many decoding heuristics and computation scheduling (flooding, layered) were implemented,
- Computations are not the perf. bottleneck, msg interleaving is…
- Implementation characteristics
  - On GPU devices float SPA, were preferred to 32b/8b-NMS => equivalent throughput perf,
  - On CPU target it is different (8b-NMS).




```
0 1 0 1 1 0 0 1
1 1 1 0 0 1 0 0
0 0 1 0 0 1 1 1
1 0 0 1 1 0 1 0
```

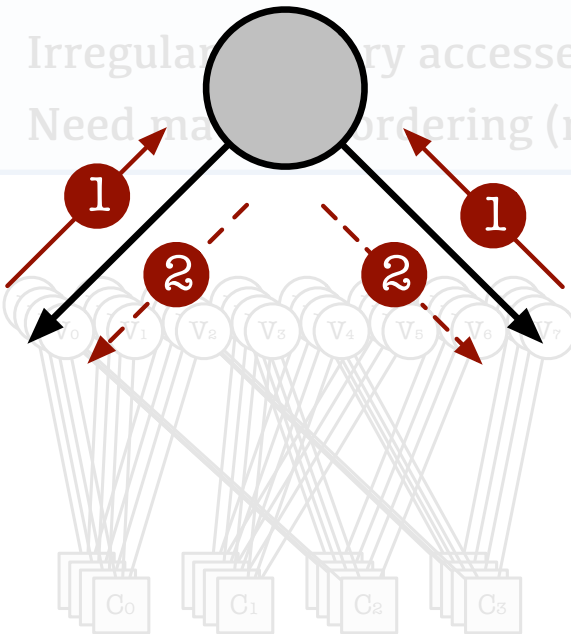
Parallelization in flooding-based decoder implementation

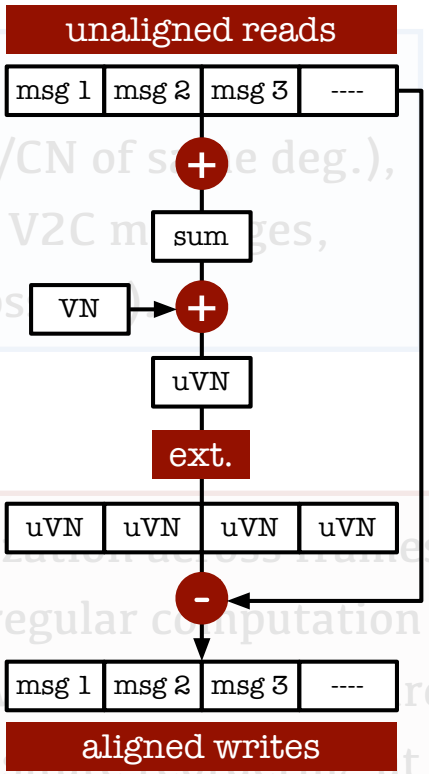


Parallelization inside **VN and CN kernels**

- Parallelization is limited due to VN and CN degrees,
- Horizontal SIMD processing (bad efficiency),
- Necessitate unaligned memory accesses to messages.

Parallel VN processing

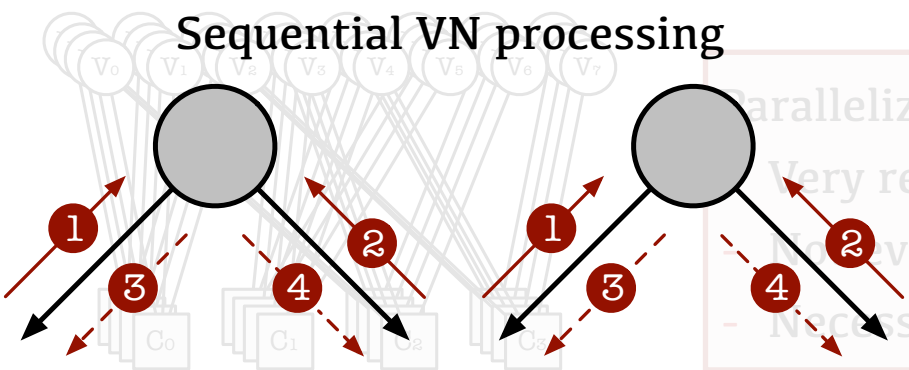


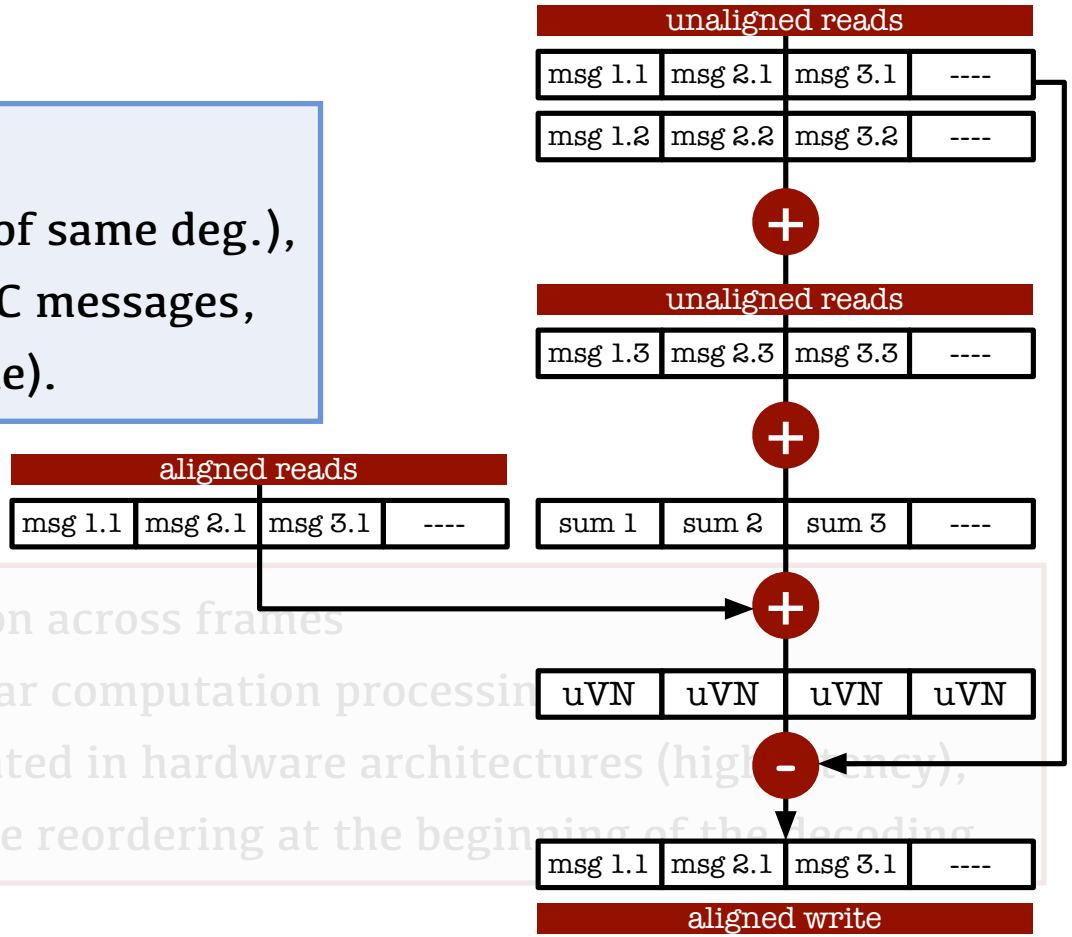


Parallelization in flooding-based decoder implementation

Parallelization of VN and CN kernels
- Parallelization is limited due to VN and CN degrees,
- Horizontal SIMD processing (bad efficiency),
- Necessitate uncollapsed memory accesses to messages.

Parallelization **across VN and CN kernels**
- Like in hardware architectures (16 VN/CN of same deg.),
- Unaligned memory accesses to C2V and V2C messages,
- Need matrix reordering (not always possible).





Parallelization across frames
- Very regular computation processin
- Not evaluated in hardware architectures (high latency),
- Necessitate reordering at the beginning of the decoding.

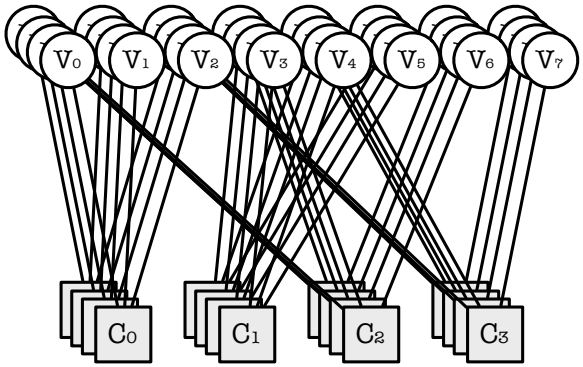
Parallelization in flooding-based decoder implementation

Parallelization of VN and CN kernels

- Parallelization is limited due to VN and CN degrees,
- Horizontal SIMD processing (bad efficiency),
- Necessitate uncollapsed memory accesses to messages.

Parallelization across VN and CN kernels

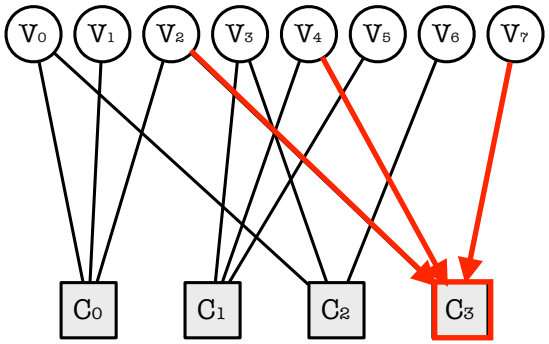
- Like in hardware architectures (16 VN/CN of same deg.),
- Irregular memory accesses to C2V and V2C messages,
- Need matrix reordering (not always possible).



Parallelization **across frames**

- Very regular computation processing (inc. memory),
- Not evaluated in hardware architectures (high latency),
- Necessitate reordering at the beginning of the decoding.

Parallelization in H. layered-based decoder implementation



Parallelization **of CN kernels**

- Parallelization is limited due to CN degrees,
- Horizontal SIMD processing (bad efficiency),
- Necessitate unaligned memory accesses to VNs.

Parallelization **across CN kernels**

- Like in hardware architectures (Q CN of same deg.),
- Unaligned memory accesses to VNs,
- Need matrix reordering (not always possible: unstructured).

Parallelization **across frames**

- Very regular computation processing (inc. memory),
- Not evaluated in hardware architectures (high latency),
- Necessitate reordering at the beginning of the decoding.

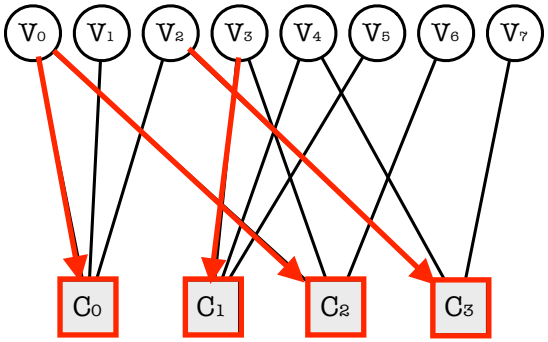
Parallelization in H. layered-based decoder implementation

Parallelization **of CN kernels**

- Parallelization is limited due to CN degrees,
- Horizontal SIMD processing (bad efficiency),
- Necessitate unaligned memory accesses to VNs.

Parallelization **across CN kernels**

- Like in hardware architectures (Q CN of same deg.),
- Unaligned memory accesses to VNs,
- Need matrix reordering (not always possible: unstructured).



Parallelization **across frames**

- Very regular computation processing (inc. memory),
- Not evaluated in hardware architectures (high latency),
- Necessitate reordering at the beginning of the decoding.

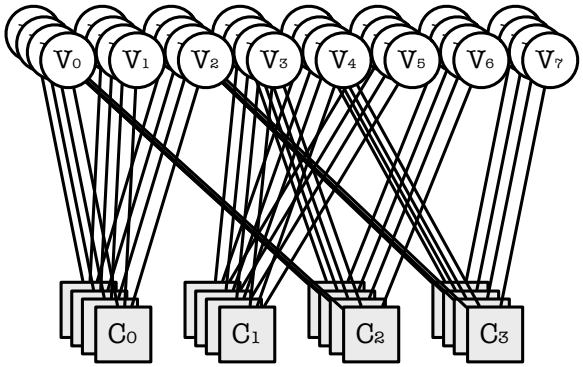
Parallelization in H. layered-based decoder implementation

Parallelization **of CN kernels**

- Parallelization is limited due to CN degrees,
- Horizontal SIMD processing (bad efficiency),
- Necessitate unaligned memory accesses to VNs.

Parallelization **across CN kernels**

- Like in hardware architectures (Q CN of same deg.),
- Unaligned memory accesses to VNs,
- Need matrix reordering (not always possible: unstructured).



Parallelization **across frames**

- Very regular computation processing (inc. memory),
- Not evaluated in hardware architectures (high latency),
- Necessitate reordering at the beginning of the decoding.

Some key parameters of a x86 decoder design

B. Le Gal and C. Jego, High-throughput multi-core LDPC decoders based on x86 processor, IEEE TPDS, 2016

Characteristics of CN computation of
« best » x86 h-layered implementation:

- 169 instructions for one CN computation,
- 47 processor clock cycles (IPC = 4).

Cn degree	# of instr.	Kernel throughput (all)	(latest)
6	169	47 cc.	66 cc.
7	194	52 cc.	73 cc.
8	218	58 cc.	79 cc.
9	242	64 cc.	84 cc.
10	269	71 cc.	89 cc.
12	320	84 cc.	103 cc.
19	517	131 cc.	150 cc.
20	548	139 cc.	156 cc.
32	858	217 cc.	242 cc.

SIMD = 32 => 47 clock cycles = 32 CNs

4x slower than hardware @ Freq. 20 faster

Characteristics of CN computation of
Marchand and al. hardware architecture:

- 6 + 6 = 12 cycles per CN computation,
- 45 CN processor in the design,

C. Marchand, L. Conde-Canencia, E. Boutillon. Architecture and finite precision optimization for layered LDPC decoders. SIPS, 2010

Some key parameters of a x86 decoder design

B. Le Gal and C. Jego, High-throughput multi-core LDPC decoders based on x86 processor, IEEE TPDS, 2016

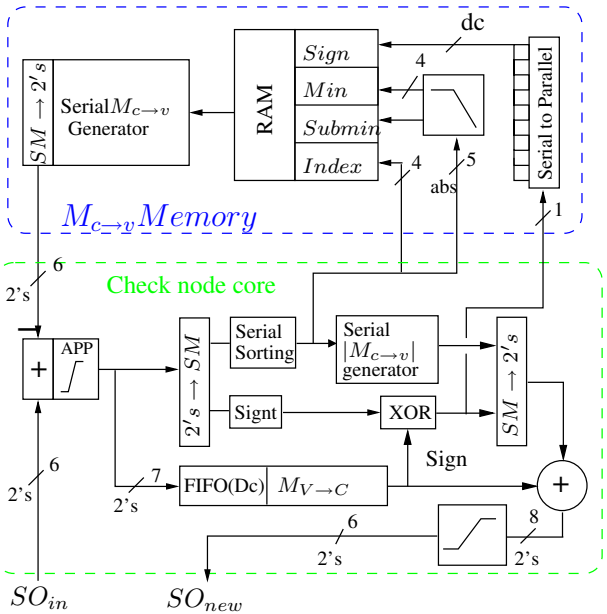
Characteristics of CN computation of « best » x86 LDPC decoder implementation:

- 169 instructions for one CN computation,
- 47 processor clock cycles (IPC = 4).

Cn degree	# of instr.	Kernel throughput (all)	(latest)
6	169	47 cc.	66 cc.
7	194	52 cc.	73 cc.
8	218	58 cc.	79 cc.
9	242	64 cc.	84 cc.
10	269	71 cc.	89 cc.
12	320	84 cc.	103 cc.
19	517	131 cc.	150 cc.
20	548	139 cc.	156 cc.
32	858	217 cc.	242 cc.

SIMD = 32 => 47 clock cycles = 32 CNs

4x slow than hardware @ Freq. 20 faster



Characteristics of CN unit of Marchand et al. hardware architecture:

- 6 + 6 = 12 cycles per CN computation,
- 45 CN processor in the design,

C. Marchand, L. Conde-Canencia, E. Boutillon. Architecture and finite precision optimization for layered LDPC decoders. SIPS, 2010

Some key parameters of a x86 decoder design

B. Le Gal and C. Jego, High-throughput multi-core LDPC decoders based on x86 processor, IEEE TPDS, 2016

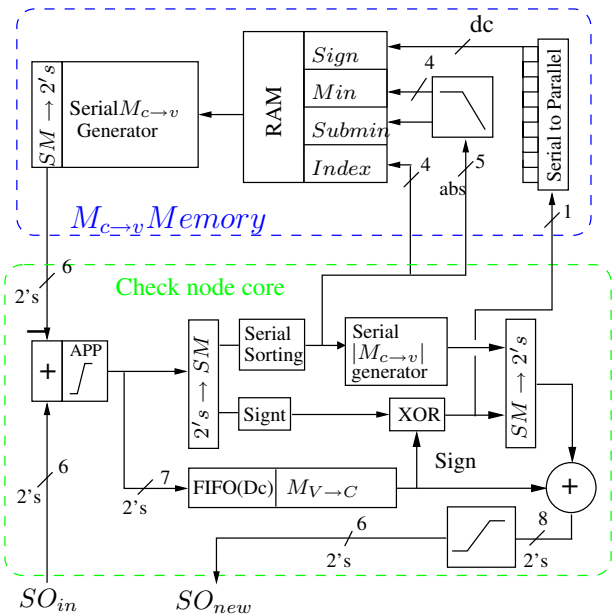
Characteristics of CN computation of the « best » x86 LDPC decoder implementation:

- 169 instructions for one CN computation,
- 47 processor clock cycles (IPC = 4).

Cn degree	# of instr.	Kernel throughput (all)	(latest)
6	169	47 cc.	66 cc.
7	194	52 cc.	73 cc.
8	218	58 cc.	79 cc.
9	242	64 cc.	84 cc.
10	269	71 cc.	89 cc.
12	320	84 cc.	103 cc.
19	517	131 cc.	150 cc.
20	548	139 cc.	156 cc.
32	858	217 cc.	242 cc.

SIMD = 32 => 47 clock cycles = 32 CNs

4x slower than hardware @ Freq. 20 faster



Characteristics of CN computation of Marchand and al. hardware architecture:

- 6 + 6 = 12 cycles per CN computation,
- 45 CN processor in the design,

C. Marchand, L. Conde-Canencia, E. Boutillon. Architecture and finite precision optimization for layered LDPC decoders. SIPS, 2010

Some key parameters of a x86 decoder design

B. Le Gal and C. Jego, High-throughput multi-core LDPC decoders based on x86 processor, IEEE TPDS, 2016

			Measured								
Code id	SIMD conf.	MF (kB)	Intr. /run	cycles /run	avg IPC	#L2 /run	#L3 /run	L2 hit rate	L3 hit rate	MCr (GBps)	MCw (GBps)
1	AVX	79	1.06E+06	3.87E+05	2.75	5168	12	100%	75%	0.009	0.000
2	AVX	170	2.30E+06	8.66E+05	2.66	11270	88	99%	90%	0.009	0.000
3	AVX	292	3.79E+06	1.58E+06	2.41	25414	2671	89%	100%	0.011	0.000
4	AVX	292	4.15E+06	1.59E+06	2.61	19422	805	96%	98%	0.011	0.000
5	AVX	472	6.68E+06	3.52E+06	1.90	85589	23113	73%	100%	0.000	0.000
6	AVX	314	4.24E+06	1.62E+06	2.62	24130	781	97%	98%	0.010	0.000
7	AVX	523	7.20E+06	2.95E+06	2.44	8196	789	90%	97%	0.011	0.000
8	AVX	1047	1.44E+07	6.84E+06	2.11	17215	10411	40%	99%	0.013	0.001
9	AVX	1206	1.55E+07	6.98E+06	2.21	70159	5518	92%	98%	0.011	0.000
10	AVX	1206	1.66E+07	6.65E+06	2.49	38425	1628	96%	96%	0.011	0.000
11	AVX	3004	4.35E+07	2.00E+07	2.17	141206	67586	52%	97%	0.036	0.013
12	AVX	2617	3.60E+07	1.88E+07	1.92	83511	59748	28%	98%	0.015	0.001
13	AVX	9555	1.31E+08	7.19E+07	1.82	1186047	1017927	14%	7%	0.017	0.105

The instruction level parallelism is still high during the overall decoding

The LLR and the message values fill in cache memory => avoid slow global memory accesses

Software LDPC decoder performances

LDPC code = (1944, 972)

LDPC decoders

	Parallelism	Target	Power (TDP)	Throughputs	Ej/bit/iter
[1]	Intra & Inter	Core i7-3960	130 W	100 Mbps @10F	130 nJ
[2]	Inter	GTX 660	140 W	927 Mbps @5L	15 nJ
[3]	Intra & Inter	GTX TITAN	250 W	237 Mbps @10F	105 nJ
[4]	Inter	Core-i7 (x4)	47 W	1522 Mbps @5L	4 nJ
[5]	Inter	ARM A15 (x4)	4 W	146 Mbps @5L	3 nJ

[1] X. Pan and al., "A high throughput LDPC decoder in CMMB based on virtual radio", in WCNC Workshop, 2013.

[2] B. Le Gal, and al., "A high throughput efficient approach for decoding LDPC codes onto GPU devices", IEEE ESL, 2014

[3] G. Wang and al., "High throughput low latency LDPC decoding on GPU for SDR systems", *GlobalSIP*, 2013.

[4] B. Le Gal and C. Jego, "High-throughput multi-core LDPC decoders based on x86 processor", IEEE TPDS, 2016.

[5] B. Le Gal and C. Jego, "High-throughput LDPC decoder on low-power embedded processors", IEEE Comm. Letter, 2015.

C. Marchand et al. (DVB-S2 LDPC code)

- Air throughput 120 Mbps @ 20L on Virtex-5 FPGA (some Watts ?)
- In [4], air throughput 232 Mbps @ 20L on INTEL Core-i7 (47 Watts).

Polar code and Turbo code decoder performances

Polar code = (2048, 1707)

Polar decoders

	Parallelism	Target	Power (TDP)	Throughputs	Ej/bit
[1]	Inter	Intel i7-2600	95 W	335 Mbps	283 nJ
[2]	Intra	Cortex-A9 ARM	< 1 W	52 Mbps	19 nJ
[3]	Intra	Corei7-4960HQ	47 W	2172 Mbps	21 nJ
[4]	Inter	Tesla K20c	225 W	964 Mbps	233 nJ
[5]	Inter	Intel i7-2600	95 W	33 Mbps (L=8)	2878 nJ

SC decoding of polar codes is not an iterative technique !

[1] G. Sarkis, P. Giard, C. Thibeault, and W. Gross, "Autogenerating software polar decoders," GlobalSIP 2014.
[2] B. Le Gal, C. Leroux and C. Jégo, "Software Polar Decoder on an Embedded Processor", SIPS, 2014.
[3] B. Le Gal, C. Leroux and C. Jego, "Multi-Gb/s software decoding of Polar Codes", IEEE TSP, 2015.
[4] P. Giard, G. Sarkis, C. Leroux, C. Thibeault, W. J. Gross, "Low-Latency Software Polar Decoders, JSPS, 2016.
[5] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast List Decoders for Polar Codes", IEEE JSAC 2016.

Polar code and Turbo code decoder performances

Turbo code = (6144, 1/3)

Turbo decoders

	Parallelism	Target	Power (TDP)	Throughputs	Ej/bit/iter
[1]	Inter	GTX 480	250 W	122 Mbps @6	339 nJ
[2]	Intra & Inter	GTX 580	244 W	192 Mbps @5	382 nJ
[3]	Intra & Inter	Xeon 5670	95 W	222 Mbps @3	142 nJ
[4]	Intra	Core-i7 4960	47 W	138 Mbps @6	57 nJ
[4]	Intra	2×E5-2680v3	240 W	716 Mbps @6	56 nJ

[1] J. Xianjun et al., "A 122mb/s turbo decoder using a mid-range GPU", IWCMC, 2013.

[2] R. Li et al., "An efficient parallel SOVA- based turbo decoder for software defined radio on GPU," IEICE, 2014.

[3] S. Zhang, R. Qian, T. Peng, R. Duan, and K. Chen, "High throughput turbo decoder design for GPP platform", ICST, 2012.

[4] Adrien Cassagne et al., Beyond Gbps Turbo Decoder on Multi-Core CPUs, Submitted in ISTC, 2016.

Polar code and Turbo code decoder performances

Turbo code = (6144, 1/3)

Turbo decoders

	Parallelism	Target	Power (TDP)	Throughputs	Ej/bit/iter
[1]	Inter	GTX 480	250 W	122 Mbps @6	339 nJ
[2]	In				382 nJ
[3]	In				142 nJ
[4]					57 nJ
[4]	Intra	2×E5-2680v3	240 W	716 Mbps @6	56 nJ

Power consumption of software decoders are mainly in range [10,500] nj/bit

Finer comparison of code families is difficult without FER, throughputs and latency constraints.

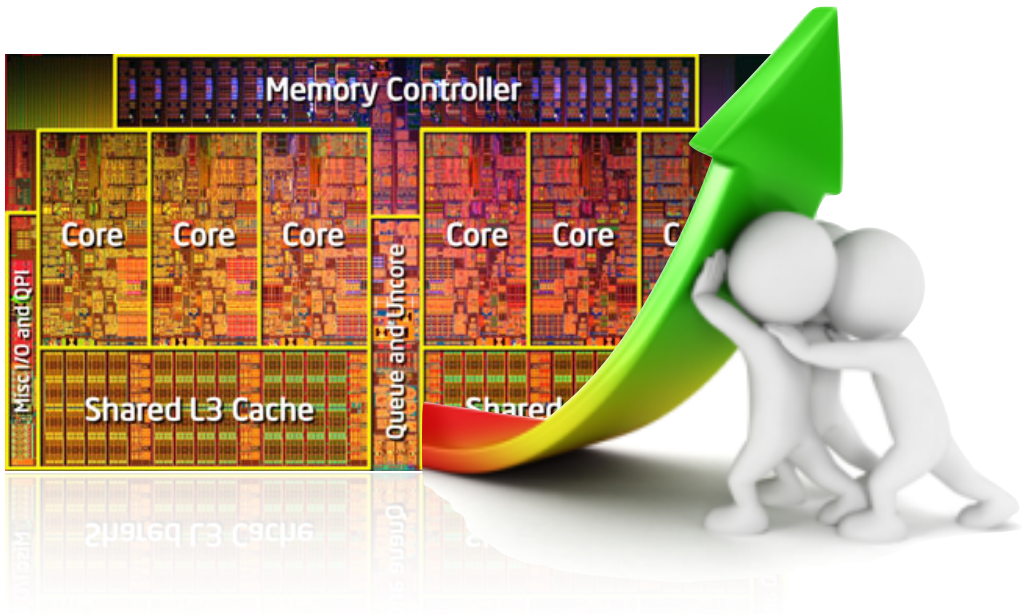
[1] J. Xianjun et al., "A 122mb/s turbo decoder using a mid-range GPU", IWCMC, 2013.
[2] R. Li et al., "An efficient parallel SOVA-based turbo decoder for software defined radio on GPU," IEICE, 2014.
[3] S. Zhang, R. Qian, T. Peng, R. Duan, and K. Chen, "High throughput turbo decoder design for GPP platform", ICST, 2012.
[4] Adrien Cassagne et al., Beyond Gbps Turbo Decoder on Multi-Core CPUs, Submitted in ISTC, 2016.

Conclusion & Future works

Conclusion on power efficiency of software ECC decoders

- Software implementation advantages:
  - « High precision » decoders (8b fixed-point),
  - Flexibility in algorithm modification (design time),
  - Easy runtime modification,
  - Reach hundreds of Mbps to Gbps throughputs,
- Sw implementations are less effective than Hw designs for handheld systems,
  - But are interesting for Software Defined Radio (SDR) or Cloud Ran (CR) systems,
- More interesting than FPGA prototypes for benchmarking:
  - ECC decoder (BER and FER perf.) => 10-6
  - Digital communication systems.

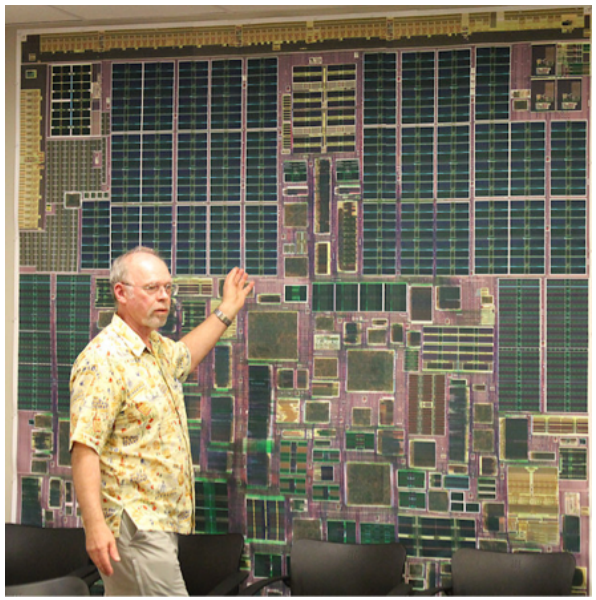
The trends in software implementation of ECC decoders



Next processor feature improvements:

- Lower power consumption (INTEL),
- Higher processing power (ARM),
- Higher computation parallelism (INTEL in 2017, SIMD = 512b, 12 cores).

The trends in software implementation of ECC decoders



Improvement of processor architectures

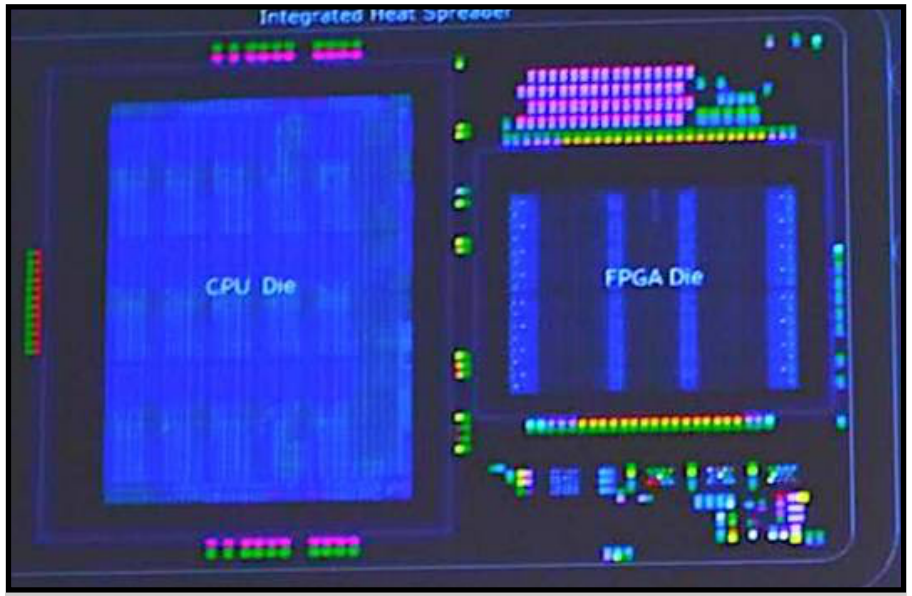
- lower power consumption for INTEL proc.
- Higher computation parallelism for INTEL processors (SIMD = 512b, 12 cores)
- Higher computation efficiency for ARM processors.

The design of custom processor cores

- ISA and memory architecture dedicated to ECC processing,
- Not complete GPP but software programmable (C/C++).

The trends in software implementation of ECC decoders

Broadwell Xeon with a built-in FPGA



The design of custom processor cores

- ISA and memory architecture dedicated to ECC processing,
- Not fully general purpose processors but programmable in software (C/C++),
- Need long hardware development cycles,
- Necessitate efficient software toolchains.

New kind of programmable architectures

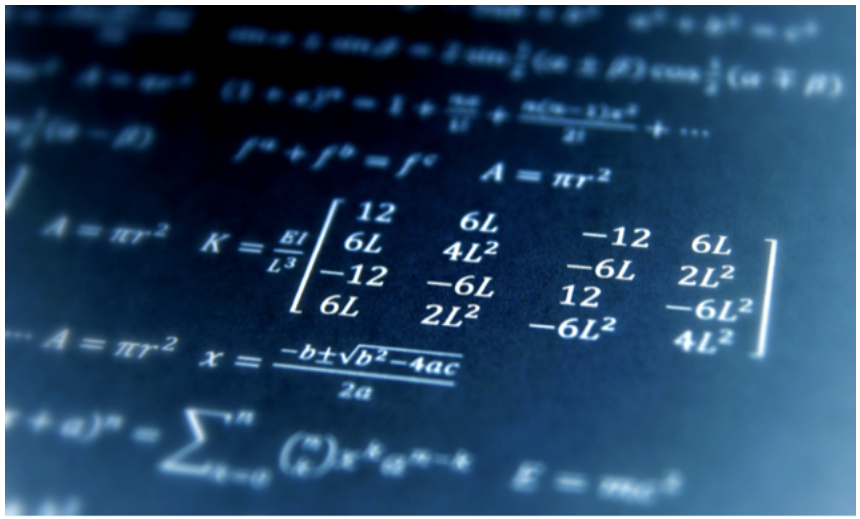
- CPUs coupled with FPGA accelerators,
- Applications can be described in C/C++ or Open-CL and automatically partitioned;

The trends in software implementation of ECC decoders

Broadwell Xeon with a built-in FPGA

New kind of programmable architectures

- CPUs coupled with FPGA accelerators,
- Applications can be described in C/C++ or Open-CL and automatically partitioned;
- The best of both world ?



Working on Error Correction Codes

- Making decoding algorithm software friendly,
- Designing new software friendly code families ?

The trends in software implementation of ECC decoders

Working on Error Correction Codes

- Making algorithm software friendly,
- Designing new software friendly code families,