



## "If Memory Serves" . Can we use memory for computing?

Henri-Pierre Charles

### ► To cite this version:

Henri-Pierre Charles. "If Memory Serves" . Can we use memory for computing?. COMPAS Conférence d'informatique en Parallélisme, Architecture et Système, Jun 2019, Anglet, France. cea-02169710

**HAL Id: cea-02169710**

**<https://cea.hal.science/cea-02169710>**

Submitted on 1 Jul 2019

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

FROM RESEARCH TO INDUSTRY

cea tech

## “If Memory Serves”

Can we use memory for computing ?

**Henri-Pierre Charles**

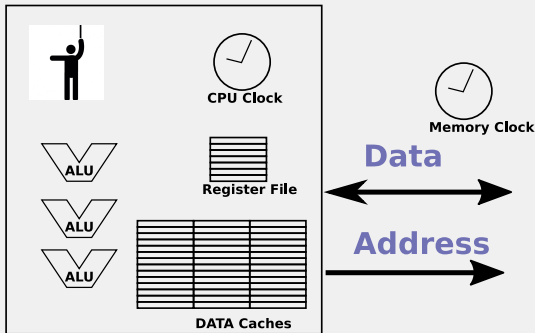
CEA/DRT/LIST/LIALP (Grenoble)

1<sup>er</sup> juillet 2019

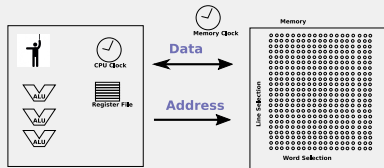
## Architecture

- Fonctionnal part : ALU (Arithmetic and logic unit)
- Dynamic part : instruction decoder
- Local storage : Registers, L1 D\$ and I\$, L2, L3 ...

## Processor schematic



## Architecture picture



## Von Neumann model

- Data & Instruction in same memory
- i.e. instructions are data
- SoC or PCB

## Instruction Memory Access

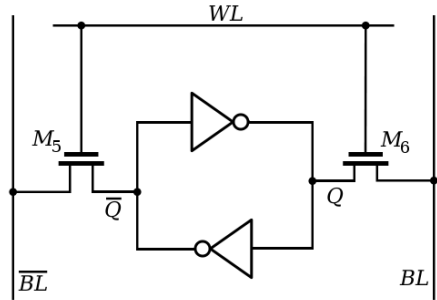
- Programs are decomposed in binary instructions by a compiler
- Memory INSN :  $ldr\ r1 = @r2$ 
  - 1 memory access (for the instruction)
  - 1 instruction cycle (Decode + RF + memory access)
- Compute INSN  $add\ r1 = r2 + r3$ 
  - Compute instruction
  - 1 memory access (for the instruction)
  - 1 computation (Decode + RF + ALU)

W: Von Neumann architecture

## SRAM memory cell depicting Inverter Loop as gates

- 6T memory cell
- Only 1 stable mode
- Read : "open" WL, read value
- Write : "open" WL, write value

## Illustration



W:Memory\_cell\_(computing)

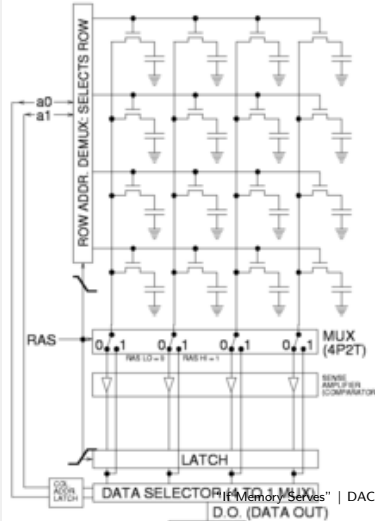
## Functions

- Select line
- Read or write
- Potentially select word in a line
- Low voltage used ; “Sense amp” to normalize

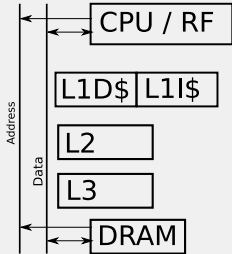
**W:**Sense\_amplifier

What every programmer should know about memory

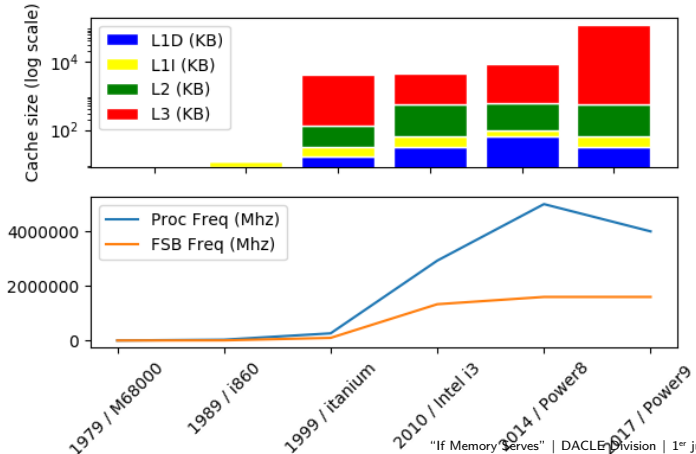
## Memory array



## Cache Hierarchy



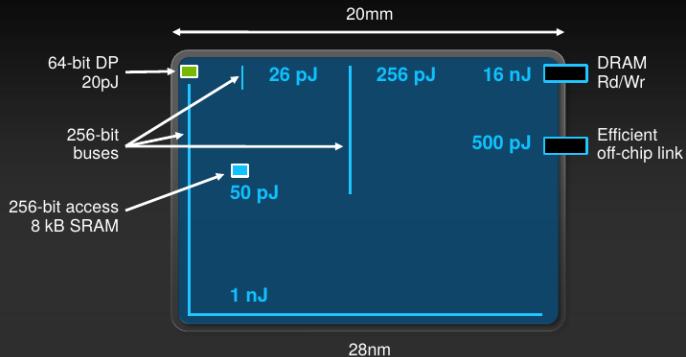
## Caches Size Evolution : "To Infinity... and Beyond !"



Bill Dally Nvidia 2010

## The High Cost of Data Movement

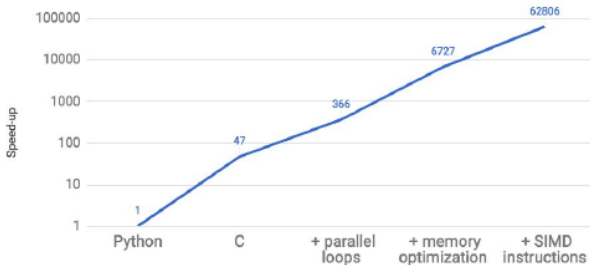
Fetching operands costs more than computing on them



## What's the Opportunity?

Matrix Multiply: relative speedup to a Python version (18 core Intel)

Matrix Multiply Speedup Over Native Python



from: "There's Plenty of Room at the Top," Leiserson, et. al., to appear.

## Programming language : C example

1972 Creation : Semantic equivalent to Intel 4004 processor but easier to read

Optimization Boolean & integer arithmetic :  
“piece of cake”

1980 FPU accelerators : no more math optimization

1990 Pipelines : complicated execution  
(delay slot, bubble)

2000 Itanium : need 5 years to obtain  
“correct” compilers

2007 CUDA, OpenCL : interleaved languages

Compilateur (Technique Ingénieur)

## Consequences

- Basic programming languages are no more at the correct semantic level (HW as higher semantic instructions than C)
- Optimization should come at algorithmic level (IA, Image processing (OpenVX, OpenCV, ... TVM : “End-to-End Optimization Stack for Deep Learning”)
- Data sets are the main optimization factor

## Where are the data ?

“It's in memory stupid !”

## Matrix multiply (sketch)

```
for (int l = 0; l < SIZE; l++)  
    for (int c = 0; c < SIZE; c++)  
        for (int k = 0; k < SIZE; k++)  
            R[l][c] += A[l][k] * B[k][c];
```

## "Real world"

```
for (c= 0; c<NCOL; c+=cacheLineSize)  
    for (l= 0; l<NLINE; l+=halfCacheLine)  
        for (c2= 0; c2<NCOL; c2+=halfCacheLine)  
            for (lk= 0; lk<halfCacheLine; lk++)  
                for (c2k= 0; c2k<halfCacheLine; c2k++)  
                    for (ck= 0; ck<cacheLineSize; ck++)  
                        res[l+lk][c2+c2k] += a[l+lk][c+ck] * b[c2+c2k][c+ck];
```

## Memory Initialization

```
# Create 'context', 'initialization'
mf = cl.mem_flags
a_g = cl.Buffer(ctx, mf.READ_ONLY | mf.COPY_HOST_PTR, hostbuf=a_np)
b_g = cl.Buffer(ctx, mf.READ_ONLY | mf.COPY_HOST_PTR, hostbuf=b_np)
```

## Interleave Python / OpenCL : prepare code

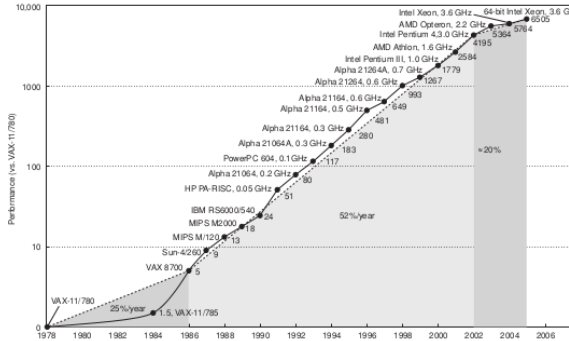
```
prg = cl.Program(ctx, """
__kernel void sum(
    __global const float *a_g, __global const float *b_g, __global float *res_g)
{
    int gid = get_global_id(0);
    res_g[gid] = a_g[gid] + b_g[gid];
}
""").build()
```

## Launch code & get result

```
res_g = cl.Buffer(ctx, mf.WRITE_ONLY, a_np.nbytes)
prg.sum(queue, a_np.shape, None, a_g, b_g, res_g)

res_np = np.empty_like(a_np)
cl.enqueue_copy(queue, res_np, res_g)
```

## La loi de Moore



Growth in processor performance since the mid-1980s. This chart plots performance relative to the VAX 11/780 as measured by the SPECint benchmarks [Hennessy]

## SW evolution : expressivity

- 1991 Python [1993] R,
- 1995 Ruby, Javascript, Java
- 1996 OCaml
- 2003 Scala [2009] GO [2010] RUST

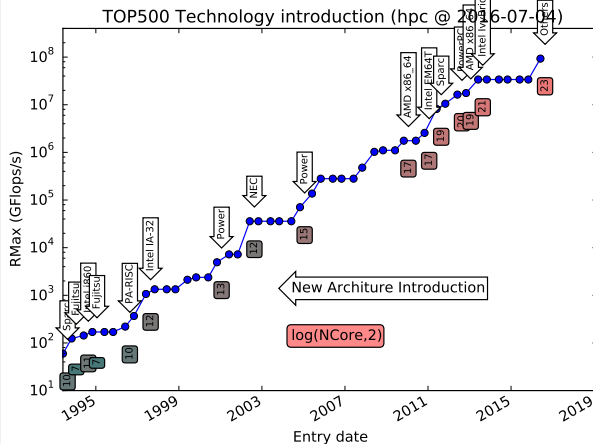
## SW evolution : scalability

- 1993 MPI,
- 1997 OpenMP

## SW evolution : accelerators

- 2007 CUDA
- 2009 OpenCL

## Top 500 & HW introduction



Link between technology introduction and the impact on performances TOP 500

Amdahl law's : "Speedup is limited by the sequential part"

## Programmer approach

- "This part is parallel" let's optimize !
- It's better to fight for a small x2 than for a big x5 !

W:Amdahl's law

## What to optimize

Two independent parts **A** **B**

Original process



Make **B** 5x faster



Make **A** 2x faster



## Many (too) metrics

- Time (Run to completion, method level?)
- IPC (Instructions per cycle)
- FLOPS (Floating points operation per second)
- FLOPSW (Floating points operation per second per watt)
- Memory bandwidth
- Cache hit ratio
- Thread / CPU ratio
- ...

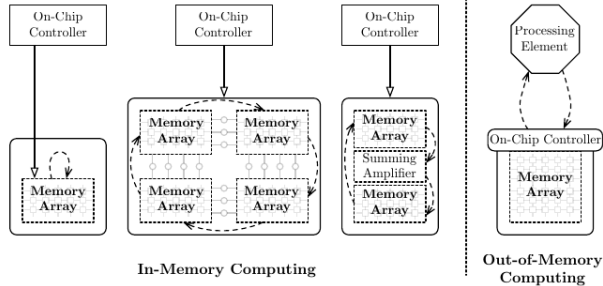
## How to measure

- Analytic (Static Code Analysis)
- Performance counter (Mouahahahaha!)
- FPS (Frame per Second)
- TPC (Transaction per second)

## High level metric : business level

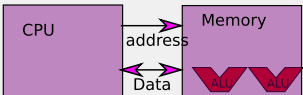
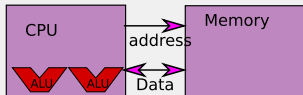
- TPCC
- Frame / s
- ../..
- A330 / Hz (cf Airbus talk ;-))

## Definition (Designer View)



Commands →    Data transfer - - - - -  
 □ Control    ○ Periphery    ▤ Memory Array    ⬡ Dedicated Processing

## Inverted model



| CPU                                                               | Memory                                     |
|-------------------------------------------------------------------|--------------------------------------------|
| control flow, address<br>compute<br>application workload, Mem I/O | answer CPU                                 |
| control flow, address<br>compute                                  | answer CPU (less),<br>application workload |

## Von neuman

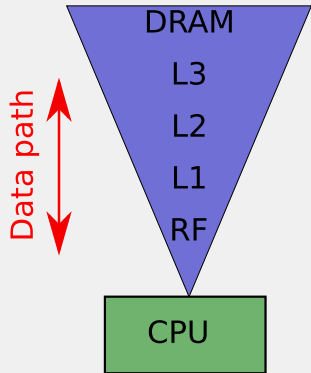
- Stored instructions
- Bottleneck = limited bandwidth (data, insn, L1, L2, L3)
- Programm with data choreography

## Non von Neumann

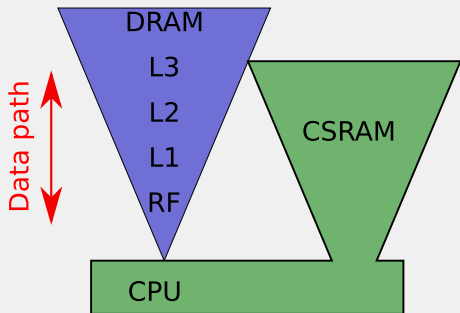
- Break model and bottleneck
- Problems
  - Send insns
  - Synchronize
  - Data layout

# CSRAM : Memory Hierarchy CSRAM

## Classical



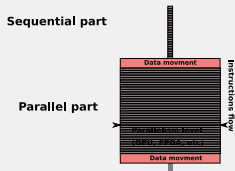
## CSRAM hierarchy



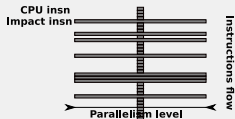
# Von Neuman broke, what about Amhdal Law's ?

Ahmdal law's : "Speedup is limited by the sequential part"

## Classical approach



## CSRAM approach



## Programmer approach

- Has to maximize parallel part
- Deal with data "choreography" between CPU and GPU.

## Programmer approach

- Ease to interlace scalar instruction and IMPACT instructions
- Do not move data

## Memory technology

**DRAM** Dense, but need refresh (external)

**SRAM** Less dense but faster (caches)

**Other** RRAM, STTRAM, ...

## Memory access

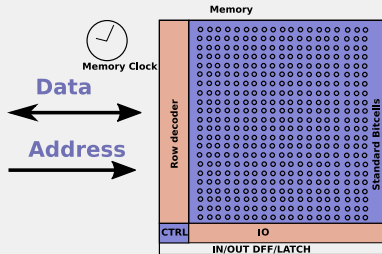
**LOAD** data or insn

- 1 Assert address
- 2 Read data

**STORE** data

- 1 Assert address & data

## Memory schematic



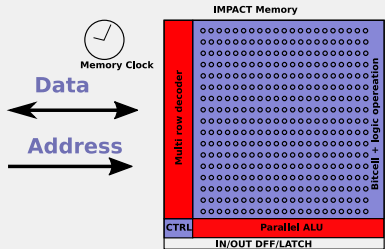
## Memory with

- Bitcell 2 ports
- Vector like alu
- Multi-line selector
- No internal instruction execution

## Design choices

- Minimize CPU modification
- Only one instruction sequence (CPU master)

## IMPACT memory



## Technology

SRAM technology (FDSOI 22nm)



-



=



## Code examples

```
void moinsImageStd()
{
    int l, c;

    for (l = 0; l < HEIGHT; ++l)
        for (c = 0; c < WIDTH; ++c)
            Out[l][c] = A[l][c] - B[l][c];
}

void moinsImageSmart()
{
    int l;

    for (l = 0; l < HEIGHT; ++l)
        Out[l] = A[l] - B[l];
} // Out A and B are array of vectors
```

## Energy gains

x35 Energy Reduction  
comparing to Scalar

## Evaluation

- Sequential Implementation : 8-bit pixels
- SIMD : 128-bit XMM registers
- C-SRAM Implementation : Use of the vectorized Arithmetic operations (SUB)

## Speedup gains

| Image size | C-SRAM | Scalar | SIMD |
|------------|--------|--------|------|
| 16x16      | 48     | x166   | x12  |
| qqVGA      | 360    | x1654  | x66  |
| QVGA       | 720    | x3307  | x130 |
| VGA        | 1440   | x6614  | X260 |

## Operation

$$\begin{bmatrix} 22 & 97 & 51 & 77 \\ 81 & 17 & 60 & 62 \\ 49 & 24 & 74 & 22 \\ 60 & 42 & 33 & 33 \end{bmatrix} * \begin{bmatrix} 28 & 68 & 17 & 22 \\ 77 & 41 & 30 & 14 \\ 82 & 99 & 14 & 70 \\ 29 & 26 & 86 & 73 \end{bmatrix} \quad (1)$$

Used in

- IA inference (Tensor flow output)
- BLAS
- Image filtering

## Naive algorithm vs C-SRAM

3 loops / 1 loop

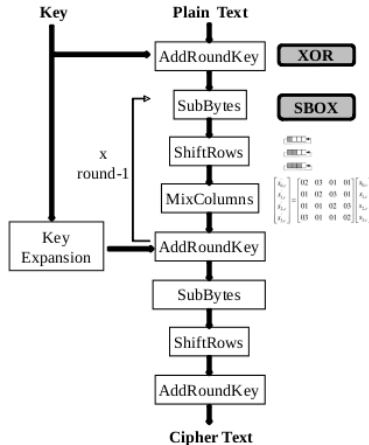
## Analytic results on matrix 8x8 uint8\_t

- Scalar :  $8^3 = 512$  ld, ld, mul, add = 2048 cycles
  - IMC :
    - 15 cycles not pipelined (8 to fill, 7 to flush)
    - 8 cycles pipelined
- Speedup = 256 in cycles

## Gains due to

Large parallelism / Data shuffling / Less data move

## Advanced Encryption Standard

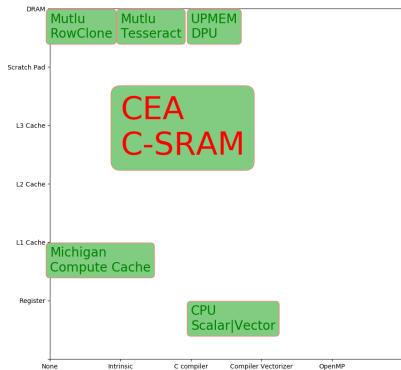


## Illustration

- Sequential Implementation : 8-bit data
- Dedicated Instruction for certain processor
  - AES-New Instruction for Intel
  - ARMv8 Cryptography extension
- C-SRAM Implementation
  - Use of new “shuffle” C-SRAM operation
  - Use of the vectorized Logic operations (XOR, AND, SHIFT)

|           | C-SRAM    | Scalar        | Recryptor [1] |
|-----------|-----------|---------------|---------------|
| Cycles    | 385       | 32544 (x84)   | 726 (x1,9)    |
| nJ        | 4,77      | 224,35 (x47)  | 7,05 (x1,5)   |
| M. Access | 10 writes | 9529 w 4440 r | n.a           |

## Illustration



## Technology / memory hierarchy

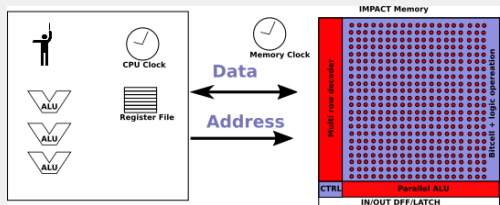
- Technology
- SW Tools level

## Scenarii

Et je n'me suis pas rendu compt' Que la  
seul' chos' qui compt'  
C'est l'endroit où s'qu'ell' tombe Y a  
quéqu'chose qui cloch' là-d'dans, J'y  
retourne immédiat'ment

La Java des bombes atomiques

## Illustration IoT



## IoT device

- “In order” simple core (RISC-V/ROCKET, CORTEX M)
- 1 IMPACT memory, STD operators
- Instruction interleaving

## Applications

- Edge IA
- Image analysis
- In memory crypto

- Data parallelism in IMPACT
- Loop, control, address computation in code

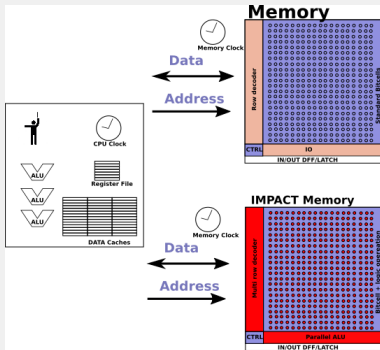
## Computing node

- Complex core (RISC-V/BOOM, CORTEX A)
- Multiple IMPACT memory, STD operators
- Dynamic RAM

## Applications

- Database
- Convolutions
- DNA matching

## Illustration : complex hierarchy

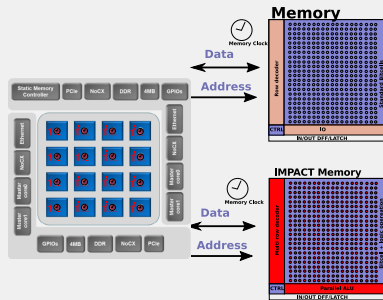


## Architecture example

- One IMPACT memory per core : local synchronous computation
- One IMPACT memory per MPSoC : dedicated core / CSRAM, other for applications

## Applications

- IMPACT for large parallel synchronous computation
- MPSoC for asynchronous computation



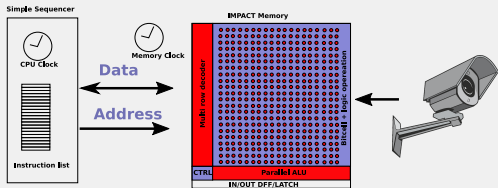
## Near Sensor Computing

- 1 sensor, e.g. a camera
- 1 automaton to send instructions
- 1 IMPACT memory,
- special operators (stochastics?)

## Applications

- Motion detection
- Pattern detection
- Other

## Illustration : edge computing



## Simulation

- Analytic speedups
- LLVM based simulation platform
- SystemC TLM + ISS level simulation
- Spice level simulation

## Software (Kernel level benchmarking)

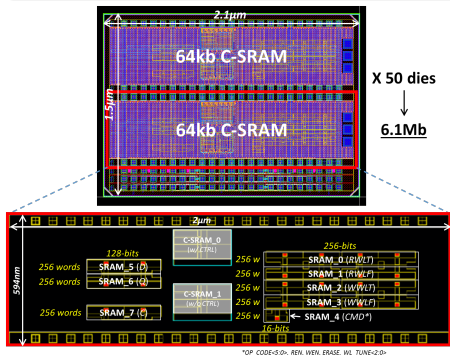
- LLVM simulation (using internal)
- Intrinsic level programming

## Publications

- 11 accepted patents
- 3 publications
- 5 patents under evaluation

## Hardware

- Circuit V1 (GF FDSOI 22nm), under testing
- Circuit V2 under development



\*OP\_CODE<5.0>, REN, WEN, ERASE, WL\_TUNE<2.0>

## Collaborative project

- ERC (Accepted)
- ANR under evaluation (2nd trial)
- European project (under evaluation)
- 

## HW roadmap

- 2 chips per year
  - Techno side
  - Functionality side

## Industrial project

../..

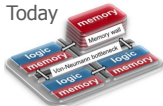
## People

- 4 permanent, 3 PhD
- 3 new PhD in October : Programming Model, System Architecture, Memory Design
- 1 CDD in October



# erc my-CUBE: Solution to Break the Memory Wall

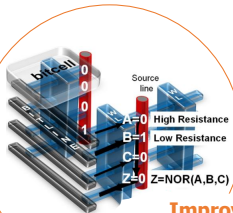
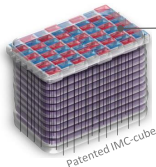
1st Topology/Technology Optimized for IMC at large scale



**Problem:**  
Energy-efficiency in  
data-abundant  
Integrated circuits

Tomorrow

**Solution:**  
Highly-parallel  
**In-Memory-Computing (IMC)**



**Improve  
Energy-Efficiency  
by 20x**

vs. Von-Neumann Systems

## MEMORY



**WP2**  
Vertical Memory

## 3D



**WP3**  
3D Process

## NANOWIRES



**WP4**  
IMC Integration

## IMC

|       |   |   |
|-------|---|---|
| NOR   | 1 | 0 |
| 1=LRS | 0 | 0 |
| 0=HRS | 0 | 1 |

**WP5**  
Circuit & Software

**WP1**  
Multi-physics



## Deliverables

Added nano-functionalities,  
IMC accelerator circuit



## Methodology

System-Technology  
Co-Optimization  
by multi-disciplinary team



## S&T Challenges

Manufacturing, heating,  
variability, software



## Applications

- Standalone NVM
- Embedded NVM | TCAM
- IMC
- Low cost
- Energy efficiency
- Security
- Reliability
- Privacy
- Performance

## François Andrieu

Directly contributed  
to leading-edge  
CMOS technology in Europe



leti  
co2tech

## Lot of work to do : develop

- Programming model
- Software tools
  - Intrinsic
  - Libraries
  - Compilation flow
- Applications for industrials
- Benchmarks for academics

## Team



Miss : Pascal Vivet, & others