FROM RESEARCH TO INDUSTRY

cea tech

Compilation & architecture de calcul, où passe l'énergie ?

Henri-Pierre CHARLES

CEA DACLE department / Grenoble

19 novembre 2020 en Visio conférence

Introduction : WhoAmI

Academic CV

- 1993 : PhD on compilers : instruction and data cache optimizations in GCC 2.95 on intel platforms
- 2008 : HdR on compilers : optimization on multimedia applications, dynamic application
- 2012 : Research director CEA Grenoble

Main research topic :

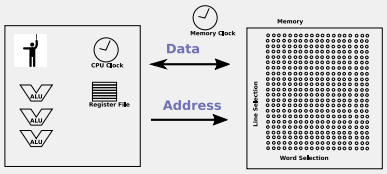
Make application auto-adaptive depending on datasets and computing architecture

Prof. Positions

- 1993 : Assistant professor Université de Versailles Saint-Quentin en Yvelines
- 2010 : Research engineer CEA Grenoble

Introduction : Von Neumann Architecture

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 : $ldr1 = @r2$
  - 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)

Von Neumann architecture

Introduction : VonNeumannRealMobileGalaxyFold

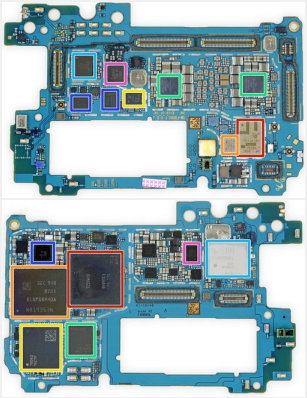
IP Blocks

- 12 GB RAM K3UHAHA Samsung; Snapdragon 855 Qualcomm : 4 big cores, 4 little; 512 GB FLASH eUFS KLUFG8RHDA-B2D1 Samsung; SDR8150 Qualcomm : LTE X24 RF Transceiver Architecture; etc
- Skyworks 78160 front-end module; Skyworks 77365 power amplifier module; Skyworks 13716 low band front-end module; etc

Energy reduction

A lot of heterogeneous IP blocks for energy reduction ! (Many compilation chains)

Mother board



Vue éclatée Galaxy Fold

Introduction : VonNeumann Cygnus Super Computer

Hw Description

- Intel Xeon (+memory)
- Nvidia GPU (+memory)
- Infiniband network

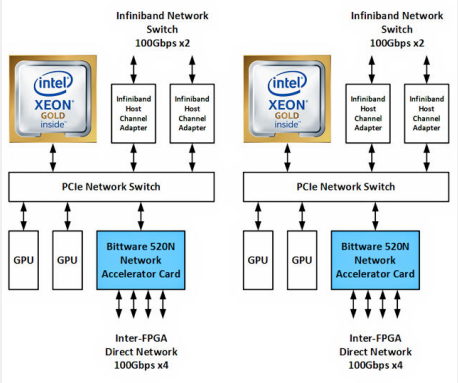
Hw Description

- C / Fortran / ...
- CUDA (GPU)
- OpenCL for FPGA circuit synthesis

Complexity management

A complex computing architecture imply a complex SW infrastructure

Schematic



Cygnus Description

Introduction : Compiler Small history

Programming language : C example

1972 Creation : Semantic equivalent to Intel 4004 processor but easyer 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

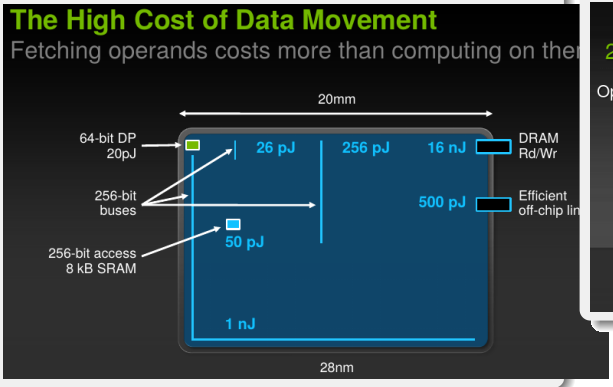
- Basic programming languages are no more at the correct semantic level (HW as higher semantic instructions than C)
- Optimization should come at algorithmic level (IA, Image processing (OpenVX, OpenCV, ... TVM: "End-to-End Optimization Stack for Deep Learning")
- Data sets are the main optimization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

Where are the data?

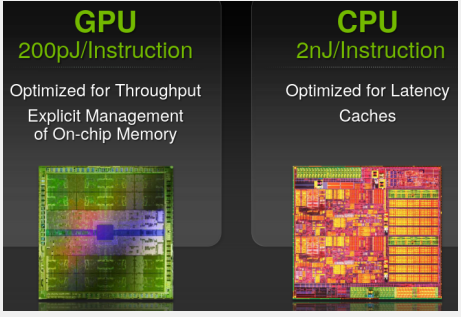
"It's in memory stupid !"

Introduction : Dally Vision from Nvidia

Bill Dally Nvidia 2010



Instruction cost



Introduction : Instruction Break Down

What is "real computing"

- Copy data from disk to DRAM (or from DRAM to DRAM)
- Copy data from DRAM to L3
- Copy data from L3 to L2
- Copy data from L2 to L1
- Copy data from L1 to Register
- Compute into ALU

Compiler job

Keep data in register as long as possible

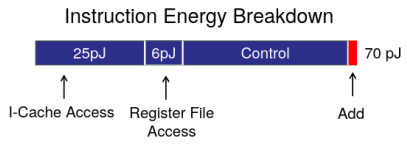
Instruction breakdown

Rough Energy Numbers (45nm)

Integer	
Add	
8 bit	0.03pJ
32 bit	0.1pJ
Mult	
8 bit	0.2pJ
32 bit	3 pJ

FP	
FAdd	
16 bit	0.4pJ
32 bit	0.9pJ
FMult	
16 bit	1pJ
32 bit	4pJ

Memory	
Cache	(64bit)
8KB	10pJ
32KB	20pJ
1MB	100pJ
DRAM	1.3-2.6nJ



1.1: Computing's Energy Problem: (and what we can do about it)

Introduction : End of Dennard Scaling

Illustration

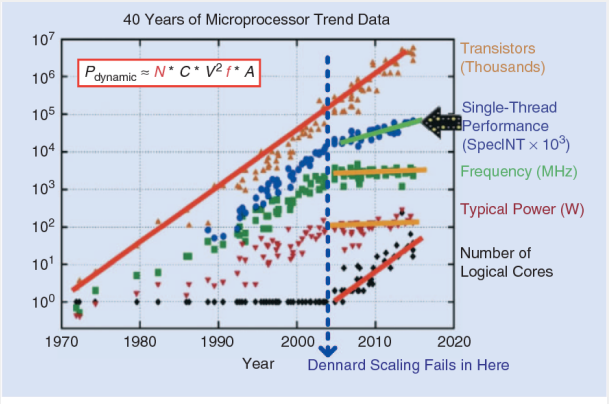
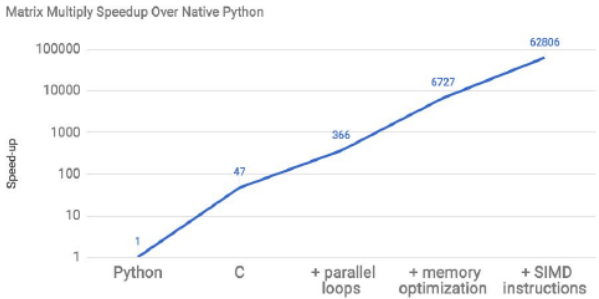


FIGURE 1: The Dennard scaling failed around the middle of the 2000s [24].

"Time Moore: Exploiting Moore's Law From The Perspective of Time"

What's the Opportunity?

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



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

Compilers : SW Parallelism Level

Code Optimisation

- -O3 basic optimization
- More than 700 individual optimization starting with `-f` (release 8.3)
- Example `-ffast-math`, `-floop-strip-mine`

Discussion on "fast-math"

What are the objects transformed by compilers ? Is it real "integer" or "real number" arithmetic ?

HW specific Optimization

- More than 210 optimization HW specific, starting with `-m`
- example `-mevexlig=[128|256|512]`

Parallelism level

- IPC
- Vectorization

Compiler : HW Parallelism Level uArch

Notions in micro architecture

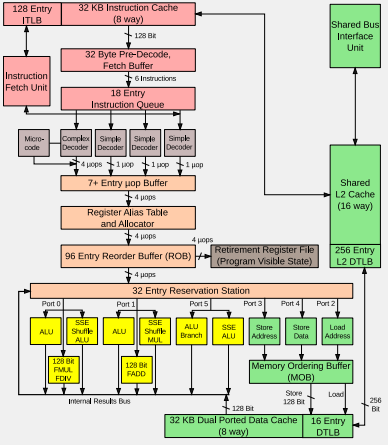
- Hidden micro architecture
- Notion of “issues”
- Notion of ROB
- Pipeline length

Intel Example

- 700 high level instructions
- RISC internal uInstructions
- Analyze with “performance counters”

Intel Micro architecture

Intel Core2 micro arch



Intel Core 2 Architecture

Models : C Language for Architecture

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];
```

Compiler : SW Parallelism Level CUDA

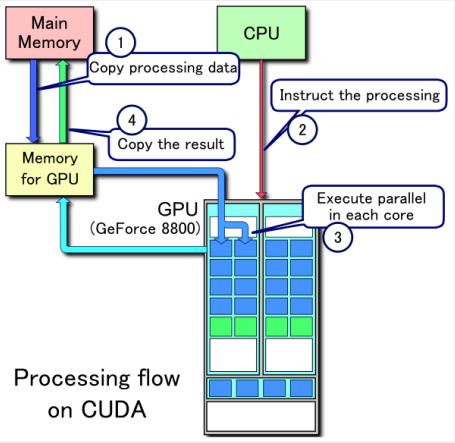
CUDA for NVIDIA GPUs

- Programming language for data parallelism
- CUDA
- Should organize “Data Choregraphy”

Program for data parallel accelerators

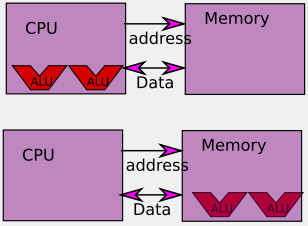
- Normalized: No
- Portable : No (NVIDIA only)
- Scalable : No

Example of CUDA processing flow



C-SRAM : Inverted Von Neumann

Inverted model



CPU	Memory
control flow, address compute application workload,Mem I/O	answer CPU
control flow, address compute	answer CPU (less), application workload

Von neuman

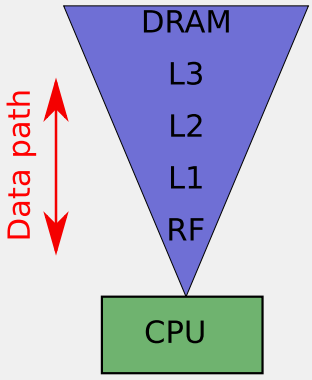
- Stored instructions
- Bottleneck = limited bandwidth (data, insn, L1, L2, L3)
- Programm with data choregraphy

Non von Neumann

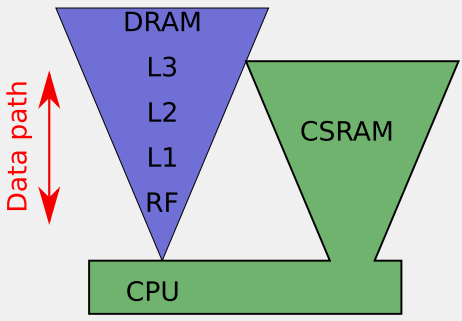
- Break model and bottleneck
- Problems
  - Send insns
  - Syncrhonize
  - Data layout

C-SRAM : Memory Hierarchy CSRAM

Classical



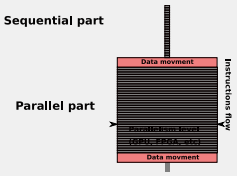
CSRAM hierarchy



Von Neuman broked, what about Amhdal Law's ?

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

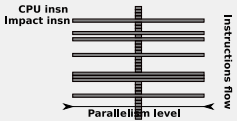
Classical approach



Programmer approach

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

CSRAM approach



Programmer approach

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

C-SRAM : CSRAM Memory

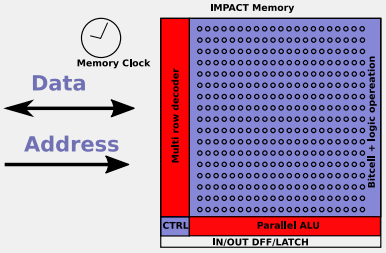
Memory with

- 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)

CSRAM : Resultats Motion Detection

-

=

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
```

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

Energy gains : x35 Energy Reduction comparing to Scalar

CSRAM : Future Results Integer Matrix Multiply

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 cyles
- 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

Conclusion : Conclusion

TOP500.org

- TOP500.org is a ranking of the 500 most powerful supercomputers
- #1 in top 500, Fugaku, based on ARM reach 82% of peak performance on 7 million cores (29 MW) on LINPACK
- #1 in top 500, Fugaku, based on ARM reach 2.9% of peak performance on 7 million cores on HPCG

Notion of "Performance Bug"

Lack of performance = Loss of energy

Recommendations

- Use small data or at least a correct data type
- Know your computing architecture
- Look at your data locality (spatial and temporal)
- Know your software tools capabilities, identify bootlenecks : arithmetic intensity, parallel efficiency, IPC
- Interact with hardware designers for new applications