FROM RESEARCH TO INDUSTRY

cea tech

Code quality.
Why I don't get my Gflops ?
Is there other metrics?

CERN Genève

Henri-Pierre Charles

CEA LIALP Laboratory / Grenoble

12 Mars 2014

www.cea.fr

leti & list

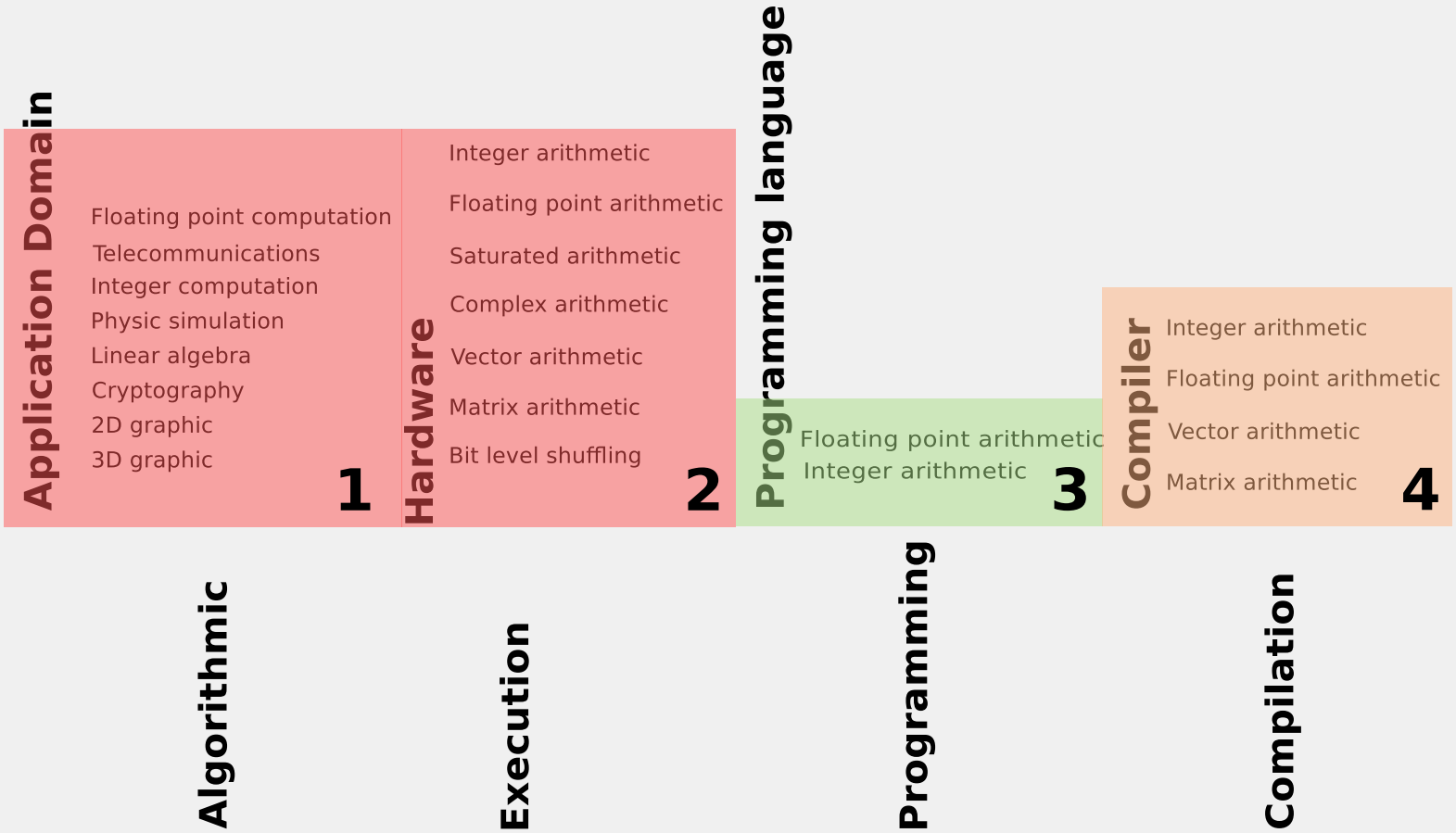
Abstract

I will talk about quality of compute intensive binary code. Why is it so difficult to reach peak performances on modern processors ?

I will present other metrics that are interesting to look at. I'll also present works done in this domain : the tool we develop and results we obtain.

From Application to binary code

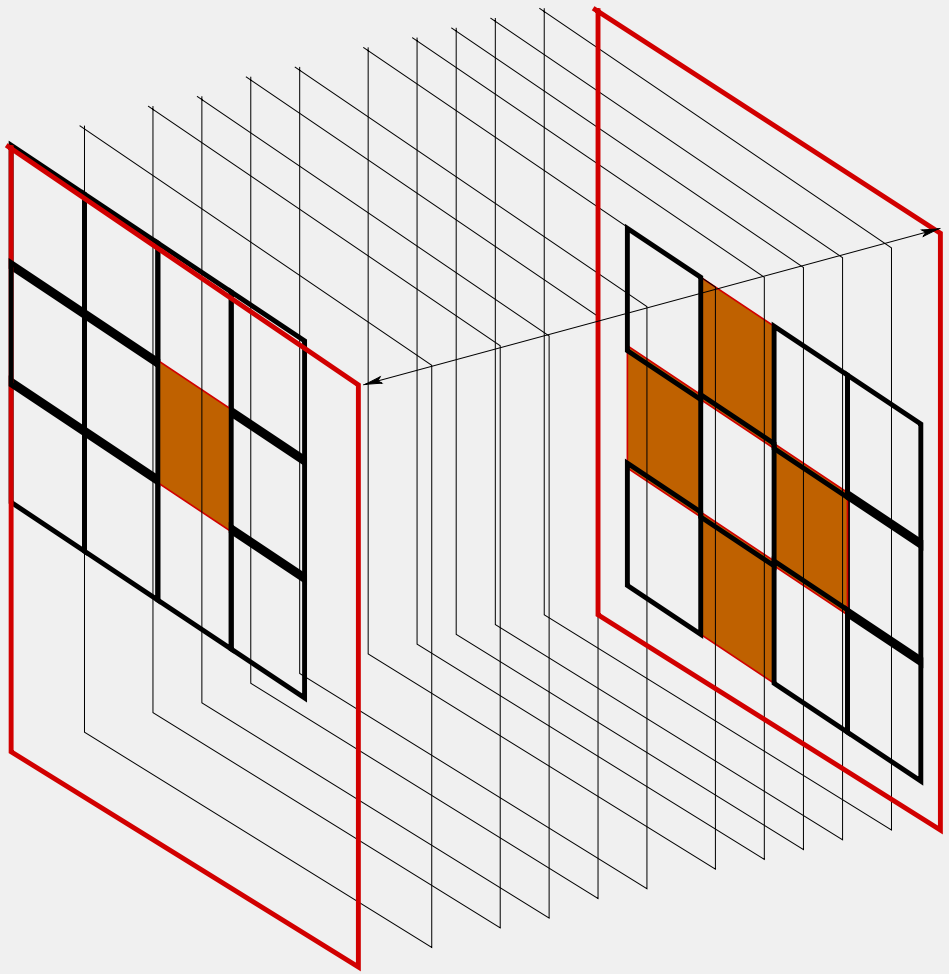
Programming Language Semantic Bottleneck



leti & list

Problem example : video compression

Illustration Video Compress



Problem example : instruction set solution

USADA8 : not a “simple instruction”

Extract from an ARM databook

A8.6.254 USADA8

Unsigned Sum of Absolute Differences and Accumulate performs four unsigned 8-bit subtractions, and adds the absolute values of the differences to a 32-bit accumulate operand.

Encoding T1 ARMv6T2, ARMv7

```
USADA8<c> <Rd>,<Rn>,<Rm>,<Ra>
```

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	1	1	0	1	1	1	Rn				Ra				Rd				0	0	0	0	Rm			

```
if Ra == '1111' then SEE USAD8;
d = UInt(Rd);  n = UInt(Rn);  m = UInt(Rm);  a == UInt(Ra);
if BadReg(d) || BadReg(n) || BadReg(m) || BadReg(a) then UNPREDICTABLE;
```

Encoding A1 ARMv6*, ARMv7

```
USADA8<c> <Rd>,<Rn>,<Rm>,<Ra>
```

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond				0	1	1	1	1	0	0	0	Rd				Ra				Rm				0	0	0	1	Rn			

```
if Ra == '1111' then SEE USAD8;
d = UInt(Rd);  n = UInt(Rn);  m = UInt(Rm);  a == UInt(Ra);
if d == 15 || n == 15 || m == 15 || a == 15 then UNPREDICTABLE;
```

Problem example : video compression

```
1  #define PIXEL_SAD_C( name, lx, ly ) \
2  int name( uint8_t *pix1, int i_stride_pix1,  \
3                  uint8_t *pix2, int i_stride_pix2 ) \
4  {                                                  \
5      int i_sum = 0;                                 \
6      int x, y;                                      \
7      for( y = 0; y < ly; y++ )                      \
8      {                                              \
9          for( x = 0; x < lx; x++ )                  \
0          {                                          \
1              i_sum += abs( pix1[x] - pix2[x] );     \
2          }                                          \
3          pix1 += i_stride_pix1;                     \
4          pix2 += i_stride_pix2;                     \
5      }                                              \
6      return i_sum;                                  \
7  }
```

(Extract from x264 video compressor from `www.videolan.org` project, $(lx, ly) \in \{(4,4),(4,8),(8,4),(8,8),(8,16),(16,8),(16,16)\}$

Problem example : video compression

Performance portability accross

- Architecture
  - Founders (INTEL, AMD, ARM)
  - Versions (ARMV7, V8, V9)
- Compilers
  - Versions
  - Vendors
  - Code generation method (static, iterative, JIT, ...)
- Deployment
  - Compilation for which target ?
  - Specialized for which data-set ?

Claim : code generation should be data dependent

Impact on speed

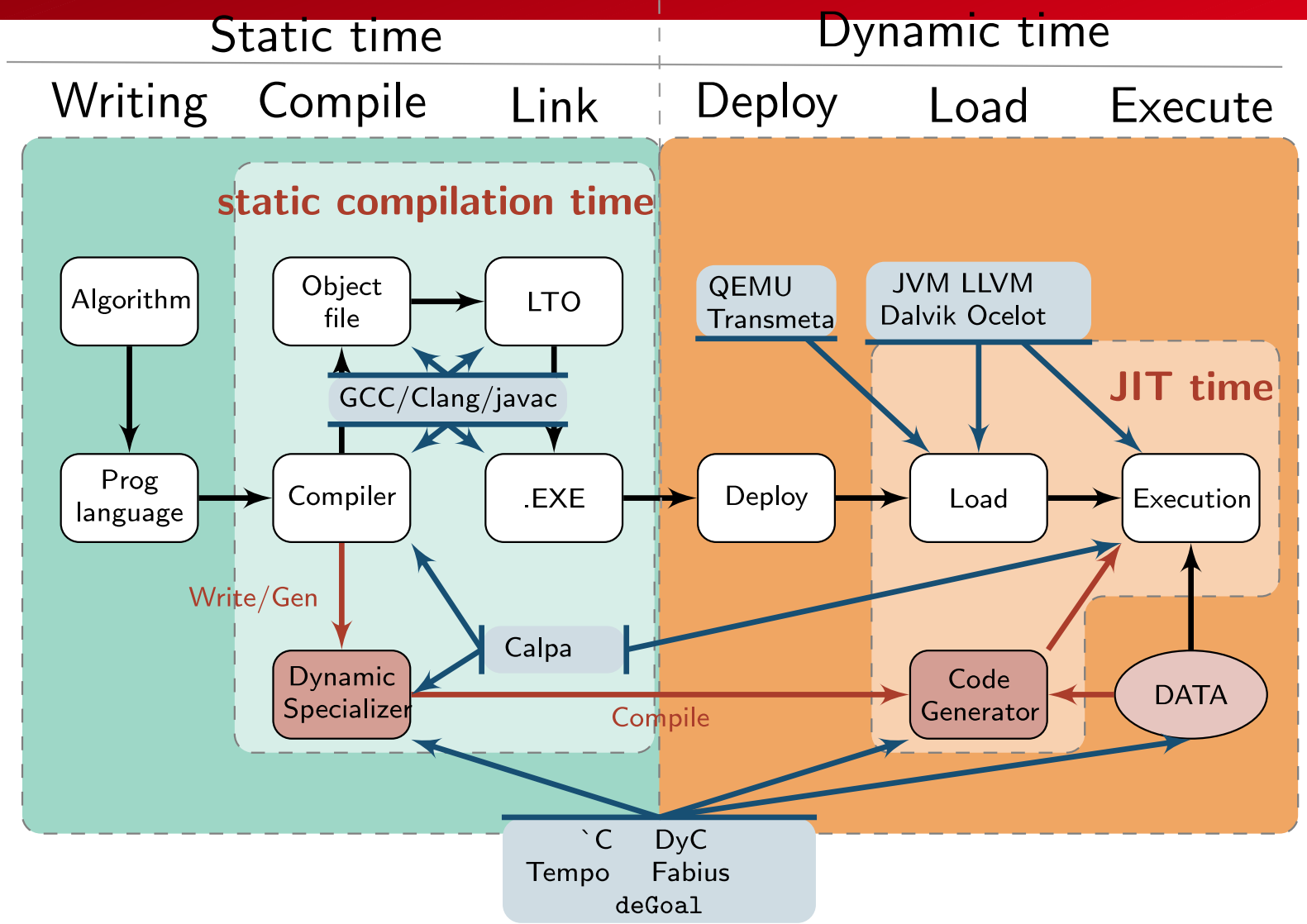
- Caches effects
- Memory aligments
- Use the correct instruction set
- “Data choregraphy”

Imortant parameters

Important parameters (for speed) on modern processors are known at differents “times” and choosen by different actors :

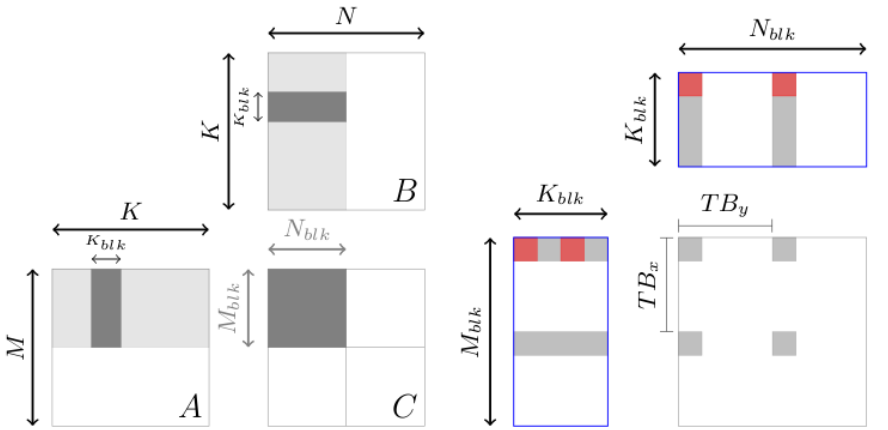
- CPU model at `Load time` by users
- Data parameters at `Execution time` by users and programmers
- “Data choregrapy” at `Writing time` by programmers

Compilation “times”



(Victor Lomüller schema)

Gemm Tiling



(a) Matrix tiling

(b) Example of data movement in a tile for the thread (1,0)

Lot of optimization parameters :

- Tiling size
- Memory allocation
  - Register
  - "Constant"
  - Textures

Each matrix size could have multiple "optimum" parameters

We experiment a "compilette" (binary code generator) which test thousands versions of Xgemm (dgemm, fgemm, etc).

H4H Projet

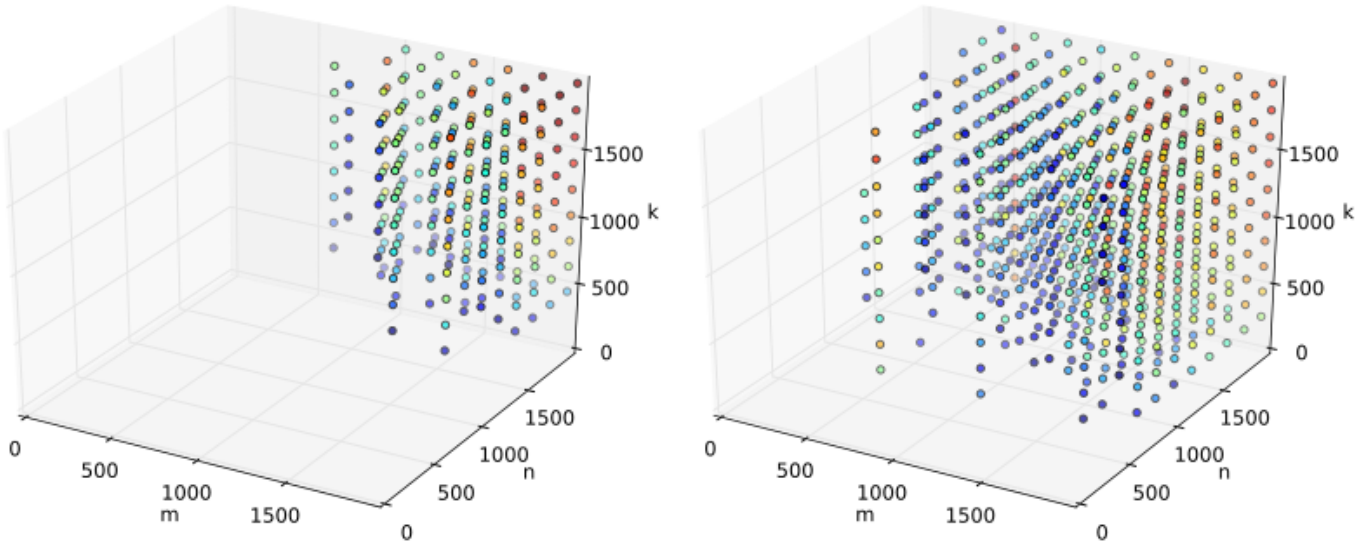
at Install time

- Write a GEMM `compilette` using tiling on GPU
- Code generation on CPU
- Exploration (machine learning)
  - Thresold detection GPU/CPU

at run-time

- If matrix size $<$ thresold
  - Use CPU
- Else use GPU :
  - optimal code generation (or reuse)
  - use the code

Experiment ResultCompare



(a) SGEMM `MAGMA` (b) SGEMM on-line search

Fig. 1: Each point represents a test case: $A \times B = C$, $A : M \times K$, $B : K \times N$ and $C : M \times N$. M, K, N vary from 64 to 1984 by step of 192. Fig. 1a and 1b represent matrix cases for which the SGEMM achieves at least 286 GFlops.

Nvidia GPU experiment (CF H4H report)

- Performance depends on data size
- (1) Install time : codelets evaluation / selection
- (2) Run time : Binary code re-generation
- 40x smaller than versionned version

Code GenerationSpeed

Facts

- Code generation based on complex IR is slow

Need

- A metric to compare code generation tool
  - At low level for dynamic code generators (insn / second, bundle / second)
  - At high level for compile farm (how many time to build 30000 binary packages ?)

CodeSize

Code size of code generator & generated code

Facts

- LLVM code generator is aproximately 20 MB
- Embedded systems could have less than 512 bytes of memory
- Power consumption will always limit memory

Needs

- Code generators that can be small and embed only necessary algorithms
- Generated code designed to be small
- Should be possible to run a dynamic code generator on a specialized core

Energy Consumption

Facts

- Memory move is costly : how much ?
- Temperature influence power consumption
- Databooks give information about timing, not energy

Needs

- Instruction power characterization
  - computation (per instruction)
  - data movement
- Energy Impact of a cache miss

Code Quality

Fact

- Speed is not the only metric
- "Bug Performance" (High level metrics)

Need

- ILP / multi core parallel level
- Energy / task
- Memory access pattern
- IP block configuration (DFT blocks configuration, Image comparator, etc)

- A lot of task to transform in compilers
- Data set characteristics are much more important than program program characteristics