J1: Compilers Challenges for Computing In Memory

Micro architecture & Compilers Internals ACACES july 2022 summer school, HiPEAC, Fiuggi, Italy

Henri-Pierre CHARLES

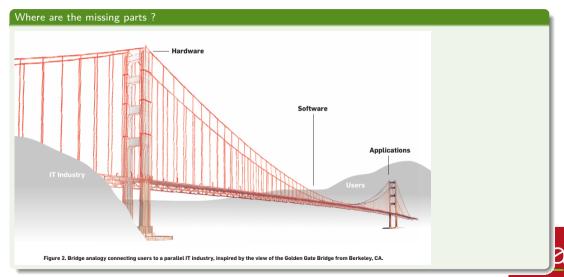
CEA DSCIN department / Grenoble

july 11, 2022



Intro: Presentation Bridge

Motivation



[Ref]K. Asanovic et al., "A View of the Parallel Computing Landscape"

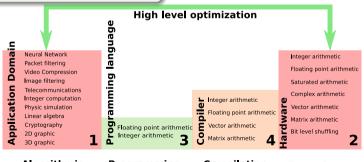
Introduction: What's the problem?

Programming language (When are you born ? :-)

- Are invented and standardized between 70 and 90'
- Based on logic, integer and floating point arithmetic.
- Compiler use arithmetic rules for optimization

Computer architecture

- Use real world datasets: bitmap images, vector images, datagram, ...
- Hardware development and Low level programming research domains are deconnected





Algorithmic

Programming

Compilation

Execution

Introduction : C Data Types

Basic Data representation

Motivation

Integers w Binary-coded_decimal, w Two's_complement (C)

Characters wBaudot_code (1901), wASCII (1970) (C), wLatin 1 1998, wUTF-8 1992.

Floating point Intel, Ibm, IEEE (C), Unum, Posit, VRP, Stochastic

Others: Pixels (RGV, YUV, CMJN, ...), IP addresses, IA Objects (Cats. dogs. ...)

Constructed Data representation

Integer array index

Characters Strings, Text
Chaînes de caractères Chaînes. Texte

Floating point matrix, vector

Pixels Images (array of struct, struct of array), Sprites,

TCP packet ../.. (look at BPF : compilation for packet filtering)

Type Based Compiler Optimisations

Integer Algebra rules : commutativité, distributivité

Character None

Floating point None, unless using -fast-math which broke algorithms based on numerical stability

Pixels None

IP addresses None (BPF DSL specialized compiler)

IA objects Human slaves

Constructed Data Representation

- Compiler has not clue of the semantic of the constructed datatype.
- "Leave the axe to the programmer"!
- Please give complex data type semantic to a compiler

Introduction: Problem Example Image Compression

ARM specialized instruction

Motivation 0000000

- usad8 rd, rs1, rs2 (Page 4482 / 7476)
- Used in all video compression tools (VLC, FFMPEG)
- [Ref]A Global Approach For MPEG-4 AVC **Encoder Optimization**

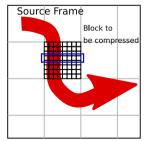
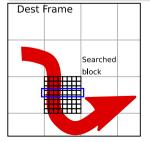


Image compression basic: block comparison

- Res = $\sum_{i=0}^{8} |in0[i] in1[i]|$
- No compiler support, should use assembly level programming
- (See Videolan VLC project)
- Software support :
 - Assembly level function : no compiler support
 - Programmer tiling model : no compiler support





master/common/pixel.c

Introduction: VLC Implementation

https://github.com/vlc-mirror/

```
https://github.com/vlc-mirror/x264/blob/
master/common/arm/pixel-a.S
       * pixel sad WxH
 52
      #define PIXEL SAD C( name, lx, ly ) \
      static int name( pixel *pix1, intptr t i stride pix1, \
 55
                     pixel *pix2, intptr t i stride pix2 ) \
 56
         int i sum = 0:
         for ( int v = 0: v < 1v: v++ )
             for( int x = 0: x < 1x: x++ )
 60
 61
                 i_sum += abs( pix1[x] - pix2[x] );
             pix1 += i stride pix1;
             pix2 += i stride pix2:
 66
 67
         return i sum:
 68
 69
 70
     PIXEL SAD C( x264 pixel sad 16x16, 16, 16 )
     PIXEL SAD C( x264 pixel sad 16x8, 16, 8 )
```

```
.macro SAD4 ARMV6 h
    function x264 pixel sad 4x\h\() armv6
                    {r4-r6,1r}
48
        push
49
        1dr
                    r4, [r2], r3
50
        1dr
                    r5, [r0], r1
51
        1dr
                   r6, [r2], r3
52
        1dr
                    lr. [r0], r1
53
        usad8
                    ip, r4, r5
    .rept (\h - 2)/2
55
        1dr
                    r4, [r2], r3
                    r5, [r0], r1
56
        1dr
57
        usada8
                    ip, r6, lr, ip
        1dr
                    r6, [r2], r3
58
59
        1dr
                    1r. [r0], r1
60
        usada8
                    ip, r4, r5, ip
61
    .endr
62
        usada8
                    r0, r6, lr, ip
63
                    {r4-r6.pc}
        pop
```

https://github.com/vlc-mirror/x264/blob/

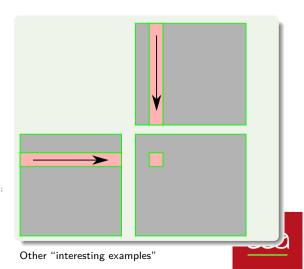
Models: C Language for Architecture

Motivation 0000000

```
Matrix multiply (sketch)
for (int I = 0: I < SIZE: I++)
  for (int c = 0; c < SIZE; c++)
    for (int k = 0; k < SIZE; k++)
      R[i][c] += A[i][k] * B[k][c];
```

```
"Real world"
for (c= 0: c<NCOL: c+=cacheLineSize)</pre>
for (I= 0; I<NLINE; I+=halfCacheLine)</pre>
 for (c2= 0: c2<NCOL: c2+=halfCacheLine)
   for (Ik= 0; Ik<halfCacheLine; Ik++)
    for (c2k= 0; c2k<halfCacheLine; c2k++)
     for (ck= 0: ck<cacheLineSize: ck++)
        res[I+Ik][c2+c2k]+= a[I+Ik][c+ck]* b[c2+c2k][c+ck];
```

Learn to program = learn to serialize / schedule on defined hardware !

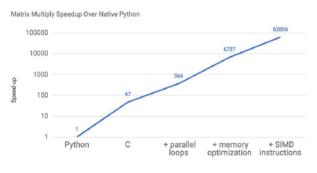


Code optimization

Motivation 000000

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.

Intro: Compilation

```
Source
#include <stdio.h>
int main(int arc, char * arrqv[])
  printf("Hello world !\n");
  return 0:
```

Un compilateur est un logiciel permettant de transformer un programme source (écrit dans un langage de programmation) dans un autre langage de programmation cible, le plus souvent dans le langage d'un processeur permettant d'exécuter le dit programme.

Destination

```
0000000000000063a <main>:
63a:
        55
63b:
        48 89 e5
63e:
        48 83 ec 10
642:
        89 7d fc
645:
        48 89 75 f0
649:
        48 8d 3d 94 00 00 00
        e8 bb fe ff ff
650:
655:
        b8 00 00 00 00
65a:
        c9
65b:
        c3
        0f 1f 40 00
65c:
```

Technology which link

- Link between artist and machine
- Expressiveness
- Automatism
- Numeric to analog
- Huge economic effect
- Huge social effect

Joseph Marie Charles dit "Jacquard"



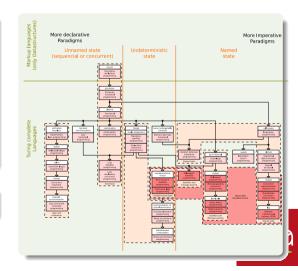
Introduction: Programming Model Paradigms

What's a programming model?

- A Knowledge in the programmer mind (aka w Programming_paradigm):
 - imperative procedural
 - object-oriented
 - declarative
 - functional
 - logic
 - mathematical ../..
- A real language which as multiple paradigms :
 - w Comparison of multiparadigm_programming_languages

Wich paradigm link to hardware features?

None



Scientific Evolution: Compilation Research Domains

Compilation Topics Map

Find code structure Extract parallelism : polyhedral approach Assertion on legacy code correctness proof, hard realtime, model checking

Security HW attach counter mesure, obfuscation

Tools for scalability Systems and library for big parallel machines

Reproductibility Statistics tools and reproductible research

Ad hoc code optimization Application driven code optimization

Legacy compiler optimization follow the HW evolution

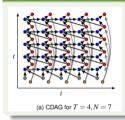
New code generation paradigm JIT. Dynamic code generation

https://top500.org

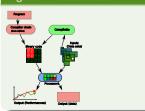
LAPACK June 2002 #1 : HPE Cray 8,730,112 cores, Linpack perf 1,102.00 PFlop/s, 65% of peak performance (usualy better)

HPCG Fugaku 7630848 cores, 16,004 TFlops 2.98 % of peak performance!

Polyhedral model



Algorithmic accelerator

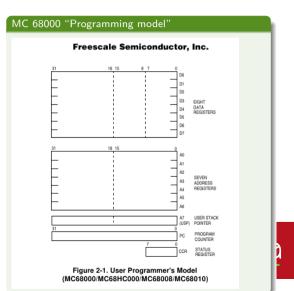




CPU Programming Model (MC68000)

What's inside the programmer head?

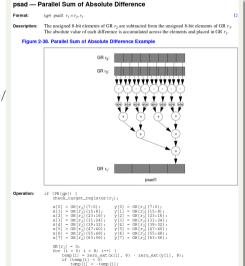
- Not only the ISA + addressing modes
- "Some" processors notions :
 - Registers
 - UALs / vectors width / # parallel UAL
 - Pipeline
 - Branch penalty
 - Caches L1 \$D \$I / Caches L2 / Caches L3
 - Interprocessors communication bandwith versus Latency
 - ...



Introduction: Low Level ProgrammingModel

ISA is the frontier between HW & SW

- Give parsable semantic (even if not "arithmetic compatible")
- Give programmer comprehension
- Give hardware semantic
- Example: https://www.intel.com/content/dam/www/public/us/en/ architecture-vol-3-manual.pdf



SOA Arch: Language Classification / Addresses

Definition

- 0 address: stack machines; all operators are implicit (bytecode java) (Java bytecode, postscript)
- Register machines :
 - 1 address : A add (other operand implicits, ARM/Jazelle)
 - 2 address : A += B (x86, ARM thumb)
 - 3 address : A = B + C (itanium, ARM, RISCV, power, kalray)
 - 4 address : A = B * C + D (power)
 - Comparison of instruction set architectures

Compromise

- Code compacity, expressiveness
- "Simplicity" / efficiency
- Code generation speed



Micro Architecture Evolution •00000000000000

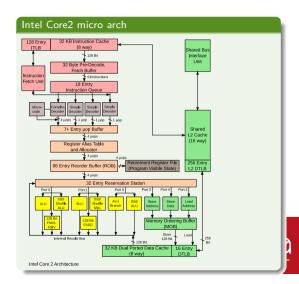
What every programmer should know about performance

- Hidden micro architecture
- Memory hierarchy

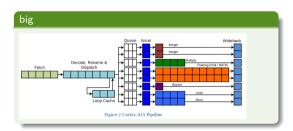
Intel Example

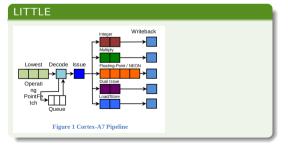
- "700" simple high level instructions
- 17000+ instructions variants
- RISC internal ulnstructions

Intel Micro architecture



SOA ARCH: ARM big.LITTLE





ARM: "more than 30 billion processors sold with more than 16M sold every day ARM" (Nov 2013) http:

//www.arm.com/products/processors/index.php

- 4 big processors + 4 little
- Same ISA. . . .
- (even for vector operation)
- Low latencie switch

big.LITTLE notion



HWParallelismLevel GPU

Characteristics

- Heterogeneous processing. In terms of
 - processing CPU + GPU
 - programming language parallelism level
- Need multiple programming language (X + Y) $X \in C$, Python, ... $Y \in CUDA$, OpenCL, ...

How to use

- Organize "data choregraphy"
- Thread synchronization



MPSoC for smart phone

- Aggregate muliple IP on a common die
 - Computing node
 - GPU (+TPU ?)
 - DSP
 - Modem, bluetooth, ...

How to use

Java for application processor

Micro Architecture Evolution

- Android for OS (with Google store)
- C for DSP, modem



HWParallelismLevel MultiCPU

Multiple CPU, same processor

- Share L3, global memory
- Same periphericals
- Multiple clocks

Best application

- Same peripherical access (disk, network if)
- Big data parallel applications



Which level is our science?

w System C

₩ QEMU

- Atomistic level / Physicists
- Electronic level / HW engineer
- Instruction level / Compiler & application
- System level /

Name	In	Out	Time
	Atoms posi-	Electonic be-	ns
	tion	havior	- 1
™SPICE	Elect. behav-	ms	- 1
	ior + layout		- 1
∞VHDI	Bloc behavior		- 1

Event

Functionnal

Functionnal

behavior

behavior

System behav-

Binary code

Binary code

ior



few

sec.

real

time

HWParallelismLevel VLIW

Argumentation

- Explicit use of multiple ALU
- Example for MM
 - 2 add for address computation
 - 2 load from memory
 - 1 MAD
 - 1 store

Example

- Intel Itanium (2 bundle of 3 parallel instructions)
- Kalray MPPA (See Kalray presentation)

Metric

- Insn / cycle
- Bundle filling rate



HWParallelismLevel Cluster

Multiple systems

- Share network
- Multiple clocks
- Multiple disks
- Multiple OS
- Multiple languages

Illustration

- Multiple tier applications
- Web applications (LAMP or variants)
- Complex paralle data analysis



HWParallelismLevel ILP

ILP - Instruction Level Parallelism

- How to use multiple ALU
- How to use special instructions. Examples
 - Multiply and Add (matrix multiply)
 - SAD (Video compression)

Metrics

- Count clock cycle / instruction count
- Compute INSN / cycle
- Verify against CPU description



HWParallelismLevel MultiCore

Many computing core

- Multiple instruction flow
- Asynchronous programming
- Share the same die

How to use

- Same die : should optimize data / code sharing
- Augment spacial data locality
- Augment code sharing

Best usage : Same code / data

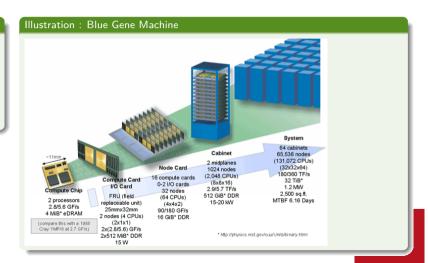
- Numeric simulation



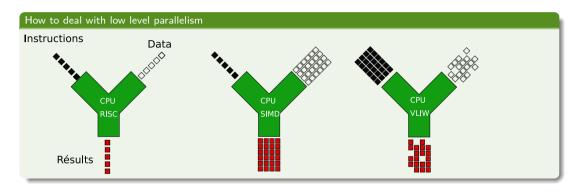
HWParallelismLevel MultipleLevel

Argumentation

- Which level fit ?
 - ALU, Operator
 - VLIW, OOO, Instruction, uArch
 - Multicore
 - MultiCPU
 - Multi computer
 - Heterogeneity



Processor Words and Instructions Size



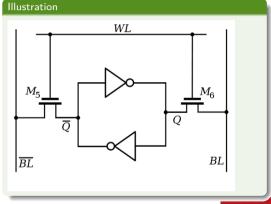
- Scalar RISC CPU: "simple" for compiler
- SIMD CPU aka Vector instructions : good for dense computation
- VLIW CPU: good for complex workload



Introduction: Remember Memory Cell 101

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



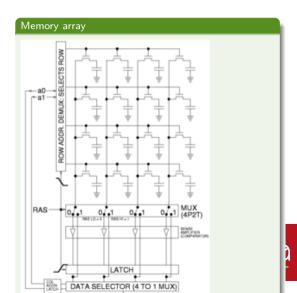
Memory_cell_(computing)



Functions

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

What every programmer should know about memory

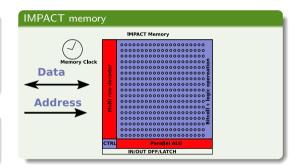


Memory with

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

Design choices

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



Technology

SRAM technology (FDSOI 22nm)



Arch: Risc Versus Cisc

Difference between CISC & RISC

- CISC : complex instruction set computer
- RISC: reduced instrution set computer
- Is it no more "simple" versus "complex"!
- w Load-store architecture

Instruction level Data access

	ALU	Reg.	bank	Caches	Memory
RISC RISC	<aritl< th=""><th>nmetic</th><th>instructions></th><th></th><th></th></aritl<>	nmetic	instructions>		
RISC			<data access<="" th=""><th>instruction</th><th>ns></th></data>	instruction	ns>
CISC		<	All instructions	access>	



Arch: ■RISC-V

RISCV

- Open source Instruction set https:
 - //riscv.org/technical/ specifications/
- Open source implementations
- Foundation based model (Swiss)
- Enterprises based on RISCV : SiFIVE. SiPEARL. GreenWayesTech
- Univesity based on RISCV : ETHZ https: //pulp-platform.org/
- RISC instruction set

Instruction extensions

Bases: RVWMO, RV32I Base Integer, RV32E Embedded, RV64I, RV128I Many extensions wRISC-V\#ISA_base_and_extensions:

- M: Multiplication
- A: Atomics LR/SC & fetch-and-op
- F: Floating point (32-bit)
- D: FP Double (64-bit)
- Q: FP Quad (128-bit)
- Zicsr: Control and status register support
- C: Compressed instructions(16-bit)
- J: Interpreted or JIT compiled languages support
- G: Shorthand for the IMAFDZicsr Zifencei base and extensions, intended to represent a standard general-purpose ISA
- V : Standard Extension for Vector Operations
- P : Standard Extension for Packed-SIMD Instructions
- & many others

Arch: Kalray

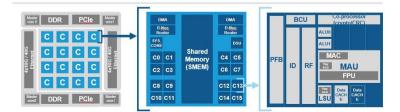
Argumentation

- Multiple grid of cores (MPSoC)
- Scratchpad Memory
- VLIW instruction set (explicit parallelism)
- Not yet open instruction set description
- 256 cores
- 1234 page ISA description

Parameters

- 5 issues architecture
- 64 registers
- 64 bits

MPPA BOSTAN ARCHITECTURE





Arch: ARM-ISA

wARM architecture family

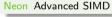
- ARM-A / Architecture WAArch64 ISA A64 / AArch32 A32 / T32 "Application"
- ARM-R Cortex-R WARM Cortex-R "Real time"
- ARM-M Cortex-M WARM Cortex-M "eMbedded" (16 bits ISA) .. less registers
- 7000+ page is a documentation

Instruction set familly

"classical" 32 bits instruction set (load/store)

Thumb Compressed instruction set

Jazelle Java support





ISA: Instruction Encoding

Instruction Encoding Trade-Off

- How to encode a data range access ?
- How to encode a large parallelism ?
- How to provide "easy" SW acessibility ?

Examples

- How many registers ? 64 registers need 6 bits, instruction with 3 addresses = 18 bits
- Data word width? Compromise between numérical / vector len and bus width (energy!)
- How many parallel operation ?
 - Vector
 - VLIW



Arch: Instruction Format

- How many instructions? opcode width
- How many registers? register encoding width
- Data size ? immediated encoding width
- Example on 32 bits RISCV instructions :

Format		Bit																														
Format	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)
Register/register	funct7							rs2					rs1					funct3			rd				opcode							
Immediate	imm[11:0					11:0	1						rs1 funct3						3	rd					opcode							
Upper immediate	immediate							i	mm[31:12]									rd					opcode							
Store	imm[11:5]		rs2					rs1						unct:	3	imm[4:0]						opcode										
Branch	[12] imm[10:5]		rs2					rs1					f	unct:	3	imm[4:1] [11]			opcode													
Jump	[20] imm[10:1]					[11]						imm[19:12]							rd						opcode							

- opcode (7 bits): Partially specifies which of the 6 types of instruction formats.
- funct7, and funct3 (10 bits): These two fields, further than the opcode field, specify the operation to be performed.
- rs1, rs2, or rd (5 bits); Specifies, by index, the register, resp., containing the first operand (i.e., source register), second operand, and destination register to which the computation result will be directed.

