FROM RESEARCH TO INDUSTRY

cea tech

Dynamic Compilation Everywhere

Henri-Pierre Charles

CEA DACLE department / Grenoble

Introduction : Course Organization

Dynamic Code Generation

- State of the art (SOA)
  - Architecture
  - Compiler
- Applications impact
- Metrics
- IR for code generation
- Fast code generation
- How to debug

Evaluation

- QCM at the end (whithout documents)
- “Short term memory”
- Questions topics are given : search for 

Assumption ; you

- understand my broken froglish
- know C language
- know compiler structure
- know a script language

Intro Why Dynamic Compilation

Pro

- Dynamic application
- Memory hierarchy versus data set
- Unknown running architecture
- Specialized instructions
- Fast development cycle

Cons

- Static Compilation
- Can verify, assert
- As many time as needed

Dynamic Code Generation ! = Dynamic Optimization (or programming)

Instruction Set Architecture

- Lowest programming Level model : assembly language
- Allow to use processor full capacity
  - Special instructions (multimedia, vectors)
  - Strange memory access
- Binary representation
- Register access

What is the “instruction execution algorithm” ?
Is the assembly language still used ?

https://en.wikipedia.org/wiki/Instruction_set

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>
```

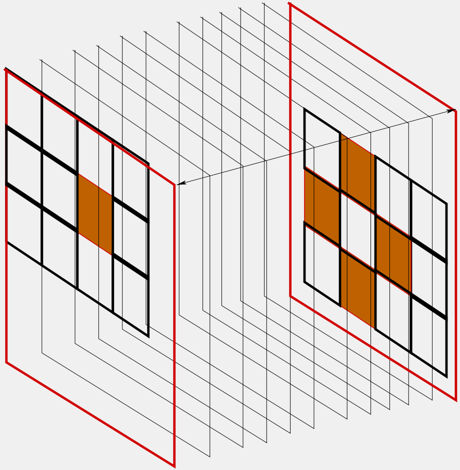
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond				0	1	1	1	1	0	0	0	Rd				Ra				Rm				0	0	0	1	Rn			

```
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;
```

file:///home/hpc/Desktop/UseFullDocs/ProcessorISA/ARM/ARM/A9docs/DDI0406B_arm_architecture_reference_manual.pdf

Problem example : video compression

Illustration Video Compress



Problem example : video compression

```
#define PIXEL_SAD_C( name, lx, ly ) \
int name( uint8_t *pix1, int i_stride_pix1,  \
                 uint8_t *pix2, int i_stride_pix2 ) \
{                                                   \
    int i_sum = 0;                                  \
    int x, y;                                       \
    for( y = 0; y < ly; y++ )                       \
    {                                               \
        for( x = 0; x < lx; x++ )                   \
        {                                           \
            i_sum += abs( pix1[x] - pix2[x] );      \
        }                                           \
        pix1 += i_stride_pix1;                      \
        pix2 += i_stride_pix2;                      \
    }                                               \
    return i_sum;                                   \
}
```

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

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)
  - 2 address : A += B (x86)
  - 3 address : A = B + C (itanium)
  - 4 address : A = B * C + D (power)

Compromise

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

SOA ARCH : ARM big.LITTLE

big

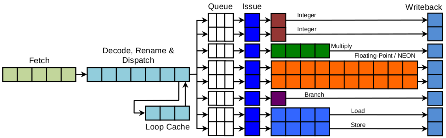


Figure 2 Cortex-A15 Pipeline

LITTLE

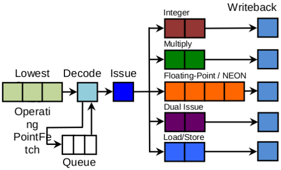


Figure 1 Cortex-A7 Pipeline

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

SOA Architecture : Raspberry

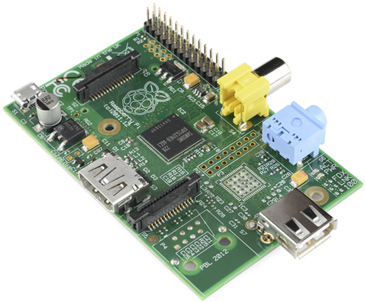
Building block for geek Web site

Raspberry

Characteristics :

- ARM1176JZF-S (ARMv6) 700MHz
- RAM : 256 Mo
- 2 video out ; audio in/out
- SDCARD mem ; 1 Port USB 2.0 ;
- 300 mA
- Full OS : linux, BSD, ...
- 20 $

Illustration



SOA Architecture : Texas Msp430

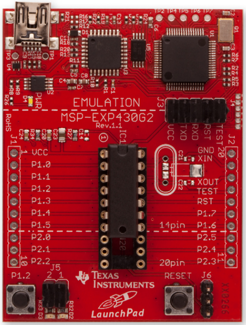
Building block for embedded system Arch Overview

Texas Microcontroller

Characteristics :

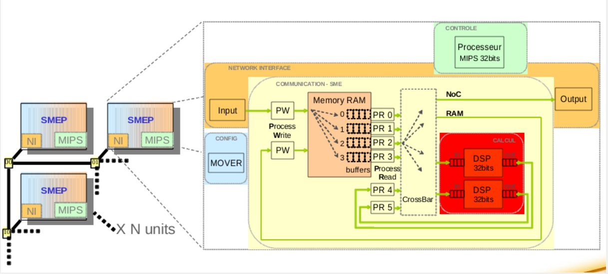
- 8 to 24 MHz
- 512 to 1024 bytes of ram
- Bare metal
- 1.75$ to 2.45$
- Cross compiler

Illustration



SOA Architecture : Genepy

Genepy Bloc diagram



Genepy characteristics

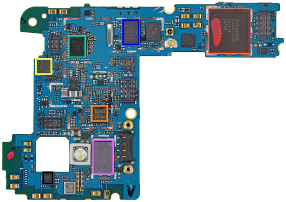
LTE modem processor

- Very low power (G Insn / mw)
- Hard to program (Kb of RAM), specialized operators

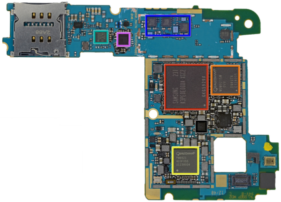
Locally heterogeneous (MIPS + DSP Mephisto) / Globally homogeneous

Architecture Nexus4

- Face
  - (red) Toshiba THGBM5G6A2JBA1R 8GB Flash
  - (orange) SlimPort ANX7808 SlimPort Transmitter (HDMI output converter)
  - (yellow) Invensense MPU-6050 Six-Axis (Gyro + Accelerometer)
  - (green) Qualcomm WTR1605L Seven-Band 4G LTE chip
  - (blue) Avago ACPM-7251 Quad-Band GSM/EDGE and Dual-Band UMTS Powe Amplifier
  - (violet) SS2908001
  - (black) Avago 3012 Ultra Low-Noise GNSS Front-End Module



- Back
  - Samsung K3PE0E00A 2GB RAM. We suspect the Snapdragon S4 Pro 1.5 GHz CPU lies underneath.
  - Qualcomm MDM9215M 4G GSM/CDMA modem
  - Qualcomm PM8921 Power Management
  - Broadcom 20793S NFC Controller
  - Avago A5702, A5704, A5505
  - Qualcomm WCD9310 audio codec
  - Qualcomm PM8821 Power Management



http://www.ifixit.com/Teardown/Nexus+4+Teardown/11781

Rappels Nexus4Compil

- Application SDK :
  - Java programming language
  - Dalvik virtual machine
  - Android Market / Google controled
- Native compilation
  - Modem stack
    - Native code
    - Cross compiler
    - Closed source
  - System stack mostly opensource
    Cyanogenmod build
    - Virtual machine Cyanogenmod
    - Building from source

Description

- C language (K&R), C90 up to C11
- Normalized
- gcc, clang, icc, tcc, ...

Compilation chain

- Preprocessing (# stuff)
- Compilation
  - Lexical / Syntaxic
  - IR form
  - Phases / passes
  - Instructions selection
- Assembly
- Load

https://en.wikipedia.org/wiki/C_(programming_language)

Rappels Static Compilation chain

Static compilation (on C language) :

1. Preprocessor (all # stuff : rewriting) `cc -E`
2. Compilation (from C to textual assembly) `cc -S`
3. Assembly (from textual asm to binary asm) `cc -c`
4. Executable (binary + dynamic library)

Optional

- Profiling : Compile (use -pg) produce `File` ; Run `File` ; Use `gprof`
- `cc -da` dump all intermediate representation

(Use gcc -v to see all the steps)

Don't stop at static time (Operating system + processor) : Load in memory, dynamic linking ; Branch resolution ; Cache warmup

SOA Java

Description

- Java Language (Sun microsystems / Oracle), 95
- Normalized
- Stack machine
- Enormous API (+10000 classes) "Compile once, run anywhere"
- Oracle, IBM, Google (+/-)

Compilation chain

- javac (.java to .class bytecode)
- interpretation
- "Hotspot" compiler 
  - trigger most used method
  - server or client profile
- class to executable

https://en.wikipedia.org/wiki/Java_(programming_language)

SW-SOA Dalvik

Description

- Java language
- Android API
- Normalized ...
- Google toolbox
- Security
- Store
- Dalvi is a register based machine

Compilation chain

- java to dalvik
- Dalvik VM optimized for small memory machines
- dalvik jitted

https://source.android.com/devices/tech/dalvik/

Description

- Low Level Virtual Machine `http://llvm.org`
- Many compilation project (compiler toolbox)
- C ; C++ ; Objective-C
- Other frontend via dragonegg

Compilation chain

- Native compilation (clang)
- LLVM bytecode interpretation (lli)
- Ahead of time compilation ()
- JIT

SOA JavaScript

Description

- Script language
- Trace based compilation
- Tons of tools (server side and client side)
- Used as back-end language
- Normalized, but with variants

Compilation chain

- Source interpretation
- JIT part of scripts

https://en.wikipedia.org/wiki/JavaScript
https://en.wikipedia.org/wiki/Comparison_of_server-side_JavaScript_solutions
https://en.wikipedia.org/wiki/Node.js
http://bellard.org/jslinux/

SW-SOA JavaScriptVM

VM Category

- Web Browser
  - V8 Chrome browser
  - Spider monkey Firefox
- Server side
- Embedded
  - “Smallest javascript”
  - Tiny-JS / STM32 C++

SW-SOA Other

Other Languages

- Python, PHP, LUA, FORTH, . . .
- Fast development
- Huge library

Compilation chain

- None
- Script compression
- JIT
- Native Call Interaction 

Arch Memory Cache Hierarchy

Memory type	Perm. ?	Access time	Who care ?
Processor Register	no	ns	Compiler
Cache L1	no	ns/ms	Compiler
Cache L2	no	ns/us	Compiler
Cache L3	no	us/ms	Compiler /Application
RAM	yes/no	10 ms	Application
Flash / SSD	yes	.1 s	System
Hard drive	yes	1-10 s	System
Tape Backup	yes	mn/hr	User

Price / Access Time / Who care about

Arch Instruction Execution

Argumentation

- While true
  - Fetch instruction
  - Increment PC
  - Decode instruction
  - Fetch / Get Data
  - Execute operation
  - Store result

Vocabulary

- Register
- UAL
- PC
- RISC/CISC

https://en.wikipedia.org/wiki/Instruction_cycle

SW-SOA OOO

Out of Order principle

- "Intelligent Reordering"
- Add a dispatch queue
- CF Slide ARM big.LITTLE

Illustration

- Easy to compile
- Same ISA, evolution at micro architectural level (TICK/TOCK Intel) 
- Difficult to compile very efficient code

https://en.wikipedia.org/wiki/Out-of-order_execution

SW-SOA MicroController

Description

Single chip with

- Compute node
- RAM / ROM
- I/O
- DAC Converter
- Chip radio (IoT)

Programming

- Bare metal 
- Cross compilation
- Interactive (Script)

https://en.wikipedia.org/wiki/Microcontroller

MPSoC : Multiprocessor System on Chip

One single chip which contains

- IP blocks (Intellectual property).
  - FFT / Turbo code / ../..
- Processors
- Memory

Why ?

- IP : Save energy
- Same chip : Save energy, ease integration
- Memory on chip : Save energy
- Multiple processors : parallelism to ... save energy

http://en.wikipedia.org/wiki/System_on_a_chip

Vocabulaire Lois

- Moore `http://en.wikipedia.org/wiki/Moore's_law`
- Amdahl `http://en.wikipedia.org/wiki/Amdahl's_law`
- Memory bound `http://en.wikipedia.org/wiki/IO_bound`
- CPU bound `http://en.wikipedia.org/wiki/CPU_bound`

Vocabulaire Speedup

Notion

Speedup $S = 100 * \frac{T_{seq}}{T_{opt}}$
Can be between

- 1 and N processor
- 1 and vectorized
- non optimized versus optimized version

Mesure Quality what are the execution conditions : data set, computer workload, reproducibly, ...

Computer science has to use "human science" tools & methodology

Metrics How to Measure Performance ?

What scale ?

- Application level (TPS, Frame/s, .../)
- Run to completion
- Method level
- Instruction level

Tools

- Wall clock
- `gettimeofday`
- Performance counters

Full application or function call ?

- Cold start / warmup 
- Statistic / multiple calls
- How many calls

Vocabulaire Peak / Sustained

Notions

Peak performance : maximal theoretical performance, assuming no bubble

Sustain performance : real acheived performance, on a real benchmark

Cost

- What is the percentage you're ready to lose ? 90% 95% ?
- How many are you ready to pay (time, money) to minimise this loss ?

`http://www.top500.org`

Vocabulaire : Cycle par seconds / Flops

Units

- **Mips** Million operation per second
- **Flops** Floating point operation per second
  `http://www.top500.org`
- **Flops/Watt** Floating point operation per watt per second
  `http://www.green500.org`
- **IPC** : Instructions per Cycle

How to mesure

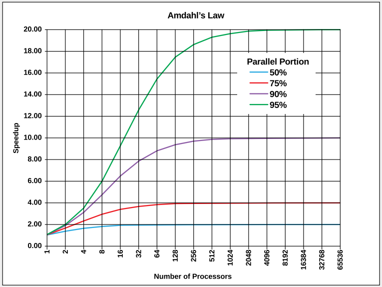
- Analytique (wall clock)
- Instrumentation
  - "Portable" (`gettimeofday()`)
  - Hardware (hardware performance counter)

Vocabulaire Amdahl

Argumentation

The speedup of a program using multiple processors in parallel computing is limited by the sequential fraction of the program. For example, if 95% of the program can be parallelized, the theoretical maximum speedup using parallel computing would be 20x as shown in the diagram, no matter how many processors are used.

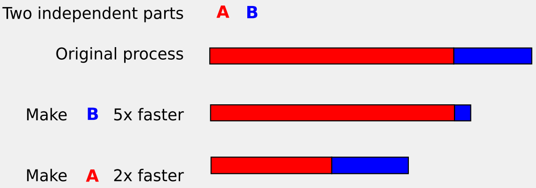
Illustration



Argumentation

Assume that a task has two independent parts, A and B. B takes roughly 25% of the time of the whole computation. By working very hard, one may be able to make this part 5 times faster, but this only reduces the time for the whole computation by a little. In contrast, one may need to perform less work to make part A be twice as fast. This will make the computation much faster than by optimizing part B, even though B's speed-up is greater by ratio, (5x versus 2x)

Illustration



Intermediate representation (IR)

Represent a program

- Each compiler has a/many IRs
- Instruction List
- Programming model (0/1/2/3 address)
- Easy to manipulate (SSA)
- List of instructions

What for ?

- (re) Generate code
- Binary code at static compil time
- Binary code at run-time

IR LLVM

LLVM Intermediate representation

- SSA form
- “Easy” to read
- Used for OpenCL
- Many Possible Scenarii

Examples

- Compilte to native `clang file.c -o file`
- Compile to bytecode `clang -emit-llvm -S file.c -o file.bc`
- Interpret : `lli file.bc`
- Compile ahead of time : `llc test.bc`
- Optimize bytecode :`opt test.bc`

IR Java ByteCode

Java Bytecode

- Stack based machine
- Byte Instructions
- Fast interpretor
- Bytecode Verifier/prover

Examples

- `javac File.java`
  "Bytecompile" java program
- `jar tf a.jar File.class`
  Link
- `java File.class or a.jar`
  Launch VM
  - `-server|-zero|-javm|avian`
    choose VM
  - `-XX:CompileThreshold=N`
    1500=client, 10000=server
- `javah .h` for JNI call

https://en.wikipedia.org/wiki/Java_bytecode

https://en.wikipedia.org/wiki/Java_bytecode_instruction_listings

IR Java ByteCode

Java VM interpretor

- Read an instruction (a byte)
- Choose a case (`switch`)
- Use statistics (method count)
- Method call :

```
if call count > thresold
then Jit compile

if method native
then call native
else call interpretor
```

HotSpot JIT

- Choose thresold (client versus server)

- JIT compile a method
- Iterative compilation
- Context switch between native and virtual environment !

Android Runtime

- Android Virtual Machine (Not JVM)
- Register based
- Dalvik was JITed, ART is ahead of time (install time)

Compiler options

- everything - compiles almost everything
- speed - maximizes runtime performance, (default option)
- balanced - performance/compilation investment.
- space - prioritizing storage space.
- interpret-only
- verify-none - should be used only for trusted system code.

https://en.wikipedia.org/wiki/Android_Runtime