FROM RESEARCH TO INDUSTRY

Code generation for Embedded Systems

Henri-Pierre.Charles@cea.fr

Henri-Pierre Charles

CEA/DRT/LIST/LIALP / Grenoble

26 November 2014

www.cea.fr

leti & list

Deal ?

- Presentation (in froglish)
  - 3 hours long
  - 1 pause
- Questions (during the talk, after by mail)
- You'll have my slides
- Evaluation
  - Multiple choices questions (positive & negative counting
  - Due in X weeks (X $\approxeq$ 2)

OK ? Questions ?

Code Generation for Embedded Systems

I'll present in this talk different ways to generate binary code for embedded systems : How to use a classical static compiler such as gcc or LLLVM and many other possibilities, dynamically (during the code execution) or not using Just In Time compiler (JIT) in differents contexts. We will also see how execute code and use tools in order to mesure metrics needed on modern platforms : speed, memory footprint, energy. The talk will begin by presenting processors used in embedded systems and present notions and vocabulary used in the talk.

Questions

- How many processors you have ?
  - Now, in the class ?
  - In your room ?

Unfinished list of anwsers

In your :

- Laptop (easy)
- Phone (How many processors ?)
- DSL box
- Car / bike / bus / tram
- Wall clock
- Watch
- USB key (yes USB disk stick)
- ../..

Questions

- How many processors you have ?
  - Now, in the class ?
  - In your room ?

Unfinished list of anwsers

In your :

- Laptop (easy)
- Phone (How many processors ?)
- DSL box
- Car / bike / bus / tram
- Wall clock
- Watch
- USB key (yes USB disk stick)
- ../..

Questions

- How many processors you have ?
  - Now, in the class ?
  - In your room ?

Unfinished list of anwsers

In your :

- Laptop (easy)
- Phone (How many processors ?)
- DSL box
- Car / bike / bus / tram
- Wall clock
- Watch
- USB key (yes USB disk stick)
- ../..

Questions

- How many processors you have ?
  - Now, in the class ?
  - In your room ?

Unfinished list of anwsers

In your :

- Laptop (easy)
- Phone (How many processors ?)
- DSL box
- Car / bike / bus / tram
- Wall clock
- Watch
- USB key (yes USB disk stick)
- ../..

Questions

- How many processors you have ?
  - Now, in the class ?
  - In your room ?

Unfinished list of anwsers

In your :

- Laptop (easy)
- Phone (How many processors ?)
- DSL box
- Car / bike / bus / tram
- Wall clock
- Watch
- USB key (yes USB disk stick)
- ../..

Questions

- How many processors you have ?
  - Now, in the class ?
  - In your room ?

Unfinished list of anwsers

In your :

- Laptop (easy)
- Phone (How many processors ?)
- DSL box
- Car / bike / bus / tram
- Wall clock
- Watch
- USB key (yes USB disk stick)
- ../..

Questions

- How many processors you have ?
  - Now, in the class ?
  - In your room ?

Unfinished list of anwsers

In your :

- Laptop (easy)
- Phone (How many processors ?)
- DSL box
- Car / bike / bus / tram
- Wall clock
- Watch
- USB key (yes USB disk stick)
- ../..

Questions

- How many processors you have ?
  - Now, in the class ?
  - In your room ?

Unfinished list of anwsers

In your :

- Laptop (easy)
- Phone (How many processors ?)
- DSL box
- Car / bike / bus / tram
- Wall clock
- Watch
- USB key (yes USB disk stick)
- ../..

Compilation

- ~~Parsing~~
- ~~Program analysis~~
- ~~Program transformation~~
- Compiler or program optimization
- Code generation

W :Compiler construction

Architectures

- ~~X86~~

More interesting

- Dynamic compilation
- Run-time optimizations
- Data dependant optim.
- Hardware optimization
- Energy
- ../..

Interesting arch.

- ARM / MIPS / μcontrol
- GPU / MALI /
- ../..

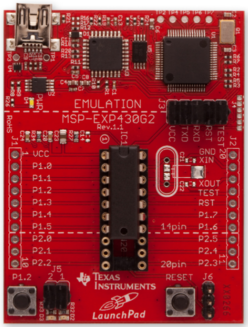
Building block for embedded system Arch Overview

Texas Microcontroller

Characteristics :

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

Illustration



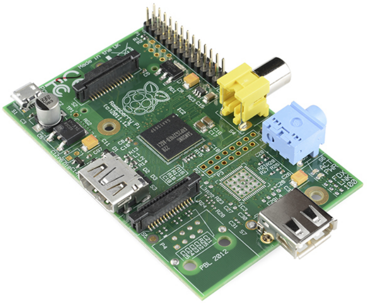
Building block for geek Web site

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



big

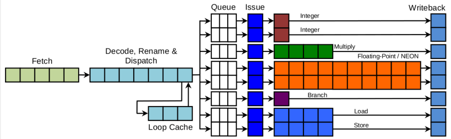


Figure 2 Cortex-A15 Pipeline

LITTLE

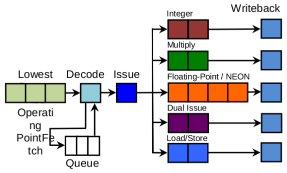
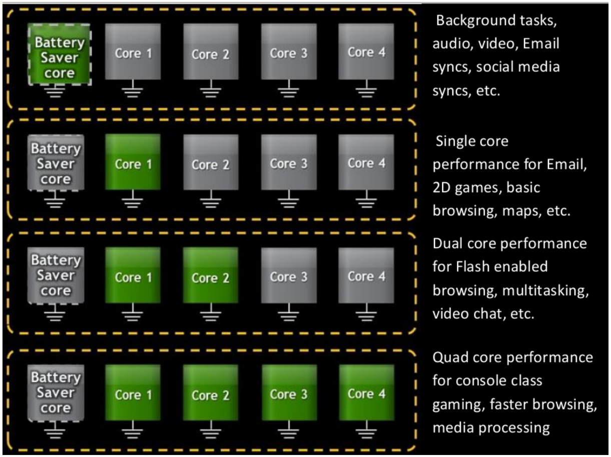


Figure 1 Cortex-A7 Pipeline

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, *To be defined*
  - (even for vector operation)
  - Low latencie switch

big.LITTLE notion


Battery Saver core
Core 1
Core 2
Core 3
Core 4
Background tasks, audio, video, Email syncs, social media syncs, etc.
Single core performance for Email, 2D games, basic browsing, maps, etc.
Dual core performance for Flash enabled browsing, multitasking, video chat, etc.
Quad core performance for console class gaming, faster browsing, media processing

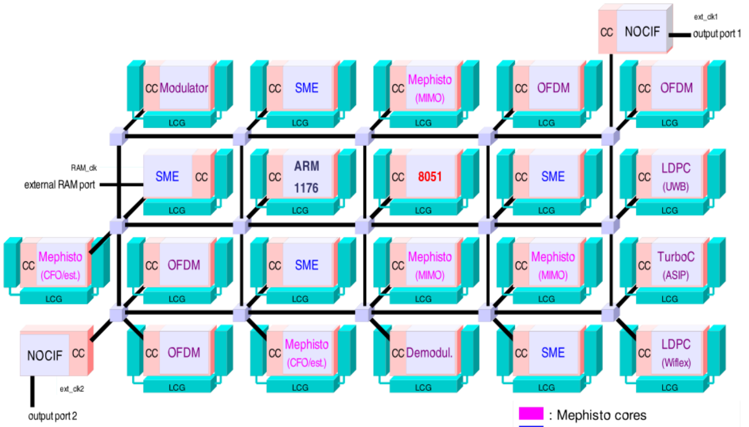
- System level selection
- Same ISA
- Usage Scenarios
- $+$ Nvidia GPU

- STM32 microcontroller / ARM Cortex-M
- RAM 255 KB max
- Flash 2048 KB max
- Sensors (temperature, etc)
- IPV6 connectiviy

Power from light (even light bulbs)

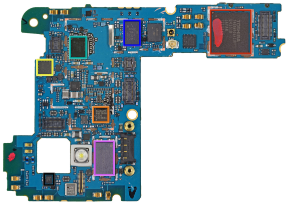
Research MPSoC dedicated to 3G LTE Advanced

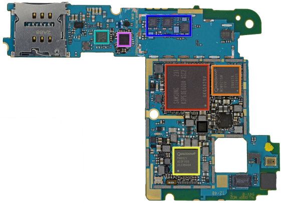
To be defined



🟪 : Mephisto cores
🟦 : Smart Memory Engine cores
🟪 : IP programmable cores
🟥 : Low power general controller
⬛ : CPU core (MAC support)

- 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



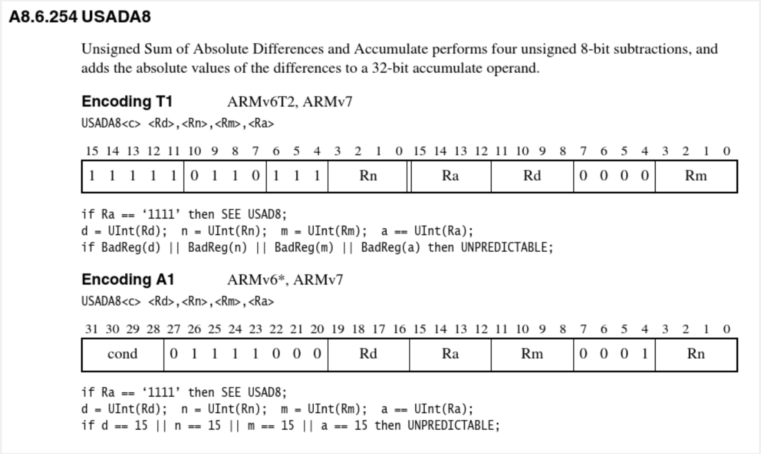


http://www.ifixit.com/Teardown/Nexus+4+Teardown/11781

- 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

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;
```

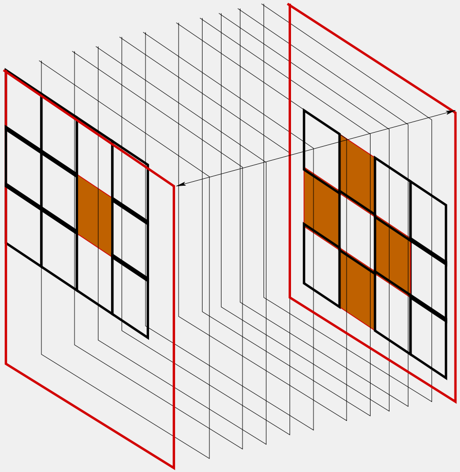
Encoding A1 ARMv6*, ARMv7

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

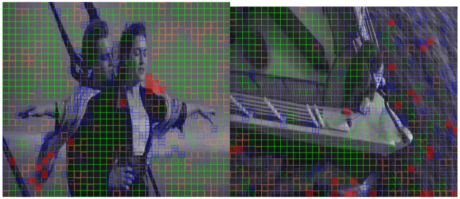
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
cond				0	1	1	1	1	0	0	0	Rd				Ra				Rm				0	0	0	1	Rn			

```
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;
```

Illustration Video Compress



Data set modify performance application during run-time :



Using a FreeBOX (ADSL Modem and TV/IP broadcasting), my home computer (Intel Celeron 1.70GHz) play correctly France3 (which use MPEG TS video format) but cannot play RTL9 (X264 video format) (2008 experiment)

Definition

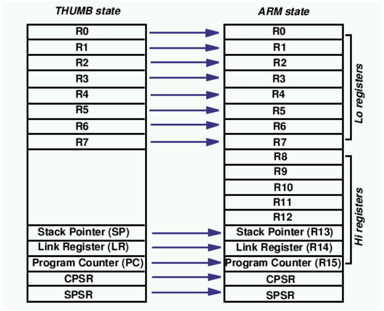
- 0 address : all operators are implicit bytecode java (Java bytecode, postscript)
- 1 address : A add (other operand implicits)
- 2 address : A += B (x86)
- 3 address : A = B + C (itanium)
- 4 address : A = B * C + D (power)

Compromise

- Code compacity, expressiveness
- “Simplicity” / efficiency

- One of the older architecture
  - 78 Intel 8086 : 16 bits
  - 82 Intel 80286 : 16 bits + MMU
  - 85 Intel 386 : 32 bits
- `http://www.intel.com`
- `http://en.wikipedia.org/wiki/X86_architecture`
- `http://www.x86-guide.com/`
- Only specialized register
- Stack machine : why ?
- 2 addresses instructions : DEST <- DEST op SRC : why ?

What is the instruction PSADBW



The "multi ISA" processor

- multimedia instruction set : NEON
- 32 bits / 3 adresses (Ratio 2.6 = 32/12)
- 16 bits / 2 adresses Thumb (Ratio 2.6 = 16/6)
- / 0 adresses Jazelle

Programming model (what is USAD8 ?)

Registers number depend on the programming model

parameter passaging depend on instruction set :

- On stack
- Register for others

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

One single chip which contains

- IP blocks (Intellectual property). What's this ?
  - 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

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 ?

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)

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`

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, ...

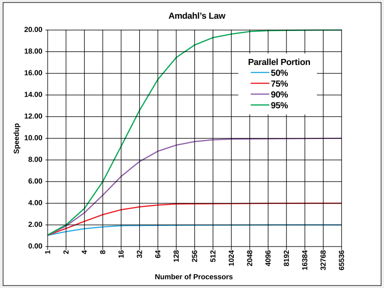
Computer science has to use “human science” tools & methodology

- 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`

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.

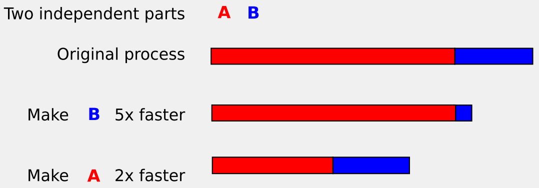
Illustration



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)

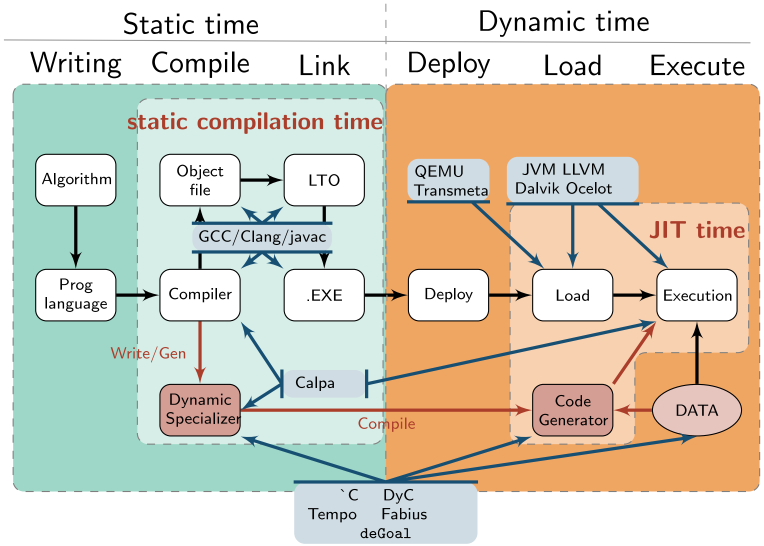
Illustration



What are speed variation for this program ?

```
int i;
for (i= 0;  i < N;  ++i)
  {
    int j;
    dest[i]= 0;
    for (j= 0;  j < N;  ++j)
      dest[i] += src[j] * m[i][j];
  }
```

Compiler, data size, target processor, available parallelism, data type, memory location, operating system, ...


Static time
Dynamic time
Writing
Compile
Link
Deploy
Load
Execute
static compilation time
JIT time
Algorithm
Prog language
Object file
LTO
GCC/Clang/javac
Compiler
.EXE
Write/Gen
Dynamic Specializer
Calpa
Compile
QEMU Transmeta
JVM LLVM Dalvik Ocelot
Deploy
Load
Execution
Code Generator
DATA
`C DyC
Tempo Fabius
deGoal

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)

(Use gcc -v to see all the steps)
Don't stop at static time (Operating system + processor) :

1. Load in memory, dynamic linking
2. Branch resolution
3. Cache warmup

Usefull tools / command line

Preprocessor	`cc -E`
Compiler	`cc -S`
Assembler	`cc -c`
Static Executable	`cc`
Desassembly	`objdump -d`
Used librairy	`ldd`
Used Function name	`nm`

Debug

- Add debuging information `cc -g`
  - Name and variable address
  - Line numbers
- Could optimize but line number less usefull
- Add backtrack information
- Profiling information
- Stack manipulation
- Use `gdb`

Simple profiling

- Compile (use `-pg`) produce `File` :
- (Add code instrumentation)
- Run `./File` produce statistics
- Use `gprof File`

Illustration

- `gcc -o Profile Profile.c -pg`
- `./Profile 20 10`
- `gprof Profile Profile.gmon`

gcc	Gnu C compiler `http://gcc.gnu.org/`
llvm/clang	`http://llvm.org/`
open64	`http://www.open64.net`
sdcc	Small device C compiler `http://sdcc.sourceforge.net`
tcc	Tiny C Compiler `http://bellard.org/tcc/`
icc	Intel C compiler x86 / Itanium
xlcc	IBM compiler power / powerpc
oracle solaris	sparc / x86
../..	

Basic optimizations. How to

- Loop unrolling
- Cache reusing
- Vectorization

- What are the effects on :
  - caches
  - compiler
  - speed

Common Sub Expression Elimination

- Remove common sub expression
- Store intermediate value
- Reuse in the next expressions
- Even invisible one (adress computation)

Illustration

- Address computation
- Math expressions
- Warning : Not for function calls !

See examples

`http://en.wikipedia.org/wiki/Loop_unwinding`

Principe

- “Déplier” le corps d’une boucle N fois
-

Benefice

- Plus de code à optimiser (CSE)
- Moins de code de branchement
- Réutilisation des caches

Boucle simple

```
for (i = 0; i < N; i++)
    sum += a[i];
```

Boucle déroulée

```
for (i = 0; i < N; i+= UNROLL) {
    sum += a[i];
    sum += a[i+1];
    /* ../.. */
    sum += a[i+UNROLL-1];
  }
for (/* No init part */; i < N % UNROLL; i++)
    sum += a[i];
```

A lot of optimization switches

- -O[0123s] Optimize more and more (s for space)
- -fxxx specific optimization exemple :

  `-funroll-loops This option makes code larger, and may or may not make it run faster.`

  `-ffast-math optimize but "it can result in incorrect output for programs which depend on an exact implementation of IEEE or ISO rules/specifications for math functions."`
- -mxxxx architecture specific optimization

  `-mthumb` Generate code for the 16-bit Thumb instruction set

How to synchronize Hw development & Sw development

- Hardware development
  - IP assembling
  - Place, route / Testing
- Software development
  - Wait for HW
  - Wait for HW
  - Start to work on
  - Update

How to synchronize HW development with SW development :

- Spice : physical level
- SystemC : `http://www.accellera.org/home/`
- GEM5 : `http://www.m5sim.org/`
- SimpleScalar : old, but still used
- Qemu : fast but not HW accurate
- Other :

QEMU is a generic and open source machine emulator and virtualizer

- System level : emulate a full system
- User level : emulate only one application
  - TCG : tiny code generator
  - KQEMU : get

Warning :

- `http://wiki.qemu.org/Download` Version : 1.3
- `http://download.savannah.gnu.org/releases/qemu/` Version : 0.15

`http://free.oszoo.org/wiki/index.php/Category:OS_images`

Qemu do a “Binary Dynamic Translation”

- Take a block of instructions
- Translate it to new architecture
- Store in a buffer
- Run it

- gcc
  - Notion of triplet : cpu-vendor-os
  - "by hand" :
  - buildroot :
- Cross tool `http://kegel.com/crosstool/`
  - `/bin/ct-ng menuconfig`
  - `/bin/ct-ng build`
- clang : the easy way, based on LLVM

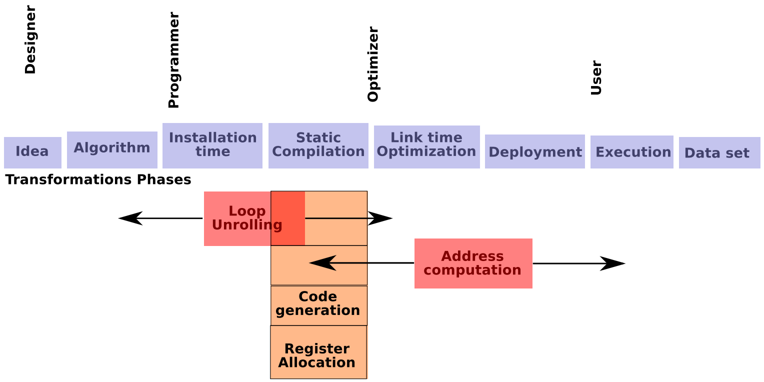
How to compile a compiler ?

A = native compiler, B = new compiler

- Build B1 with A
- Build B2 with B1
- Build B3 with B2
- Compare B2 and B3 for safety

Question : How was build the first compiler ?

- Compile with a native compiler
  - `cc Hello.c -o Hello`
  - `./Hello`
  - `Hello world`
  - `file Hello`
  - `Hello: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.18, not stripped`
- Compile with cross compiler
  - arm-elf-eabi-gcc -o Hello Hello.c
- Run with emulator
  - qemu-arm Hello


Designer
Programmer
Optimizer
User
Idea
Algorithm
Installation time
Static Compilation
Link time Optimization
Deployment
Execution
Data set
Transformations Phases
Loop Unrolling
Address computation
Code generation
Register Allocation

"Hash table" : basic tool in compiler and object management

- Performance require an "efficient" hash function
  - in speed
  - in hashing quality
- a "perfect hashing" require data knowledge.
- "Perfect hash" tools require a data set and give an hash function

`http://en.wikipedia.org/wiki/Perfect_hash_function`

`http://en.wikipedia.org/wiki/Dynamic_perfect_hashing`

Dynamic code generation

- Code generation done at run-time
  - Architecture “independance”
  - Binary size
- Oracle Java virtual machine contains “hotspot compiler”

HotSpot bytecode compiler

- Mix interpretation and compilaton
- At start do bytecode interpretation
- Count method invocation : detect “hotspots”
- Binary compile at a given thresold
- Recompile and optimize if necessary

“Compilation on demand”

When Asm is mandatory

- Assembly programming
- Macro assembly Assembly preprocessing
- Example from X264
- `http://www.videolan.org/developers/x264.html`

Lo
Ex

The Fastest Fourier Transform of the West !

- Static compilation time
  - Generate automatically many FFT codelets version
  - Evaluate performances of each
- Run Time
  - Decompose the problem in call of optimized codelets with a “plan”
  - Use the same “plan” for all identical call

Atlas : library for linear algebra compuation

- Static code
- Many different versions
- Test & optimize theses versions
- Find the best version on a given platform

Binary code generation

- Data value, data size are main parameters
- Instruction set can be choosen at run-time
- At “data value time” we could adapt the binary code

Javascript

- Scripting language
- No strong type
- Used in web browser
- In charge of
  - Local interaction
  - Refresh page without full reload
  - Local Graphic Computation
  - ../..

Jit Compilation

- Mandatory for modern application (google doc, web applications)
- JIT compilation
- Many concurrent projects (Mozilla, chrome, IE)

My favorite demo : `http://bellard.org/jslinux/`

Java programming language

- Compiled language (95')
- Normalized
- Server side applications
- JIT Compilation (no HW iteraction)
- Object oriented
- Strong data type

Jit compilation

- Multiple JVM vendors
- Mix between interpretation & JIT compil

Interested to work in this domain ?

Thesis subjects

- How to Build auto-adaptive libraries ?
- How to embed binary code generator in Java applications ?
- What is an operating system if the ram is permanent ?

`mailto:Henri-Pierre.Charles@cea.fr`