Video Game Consoles Emulation: HOWTO?

Pierre Bourdon Nicolas Hureau

{delroth,kalenz}@lse.epita.fr
http://lse.epita.fr

July 18, 2012

What is emulation?

- Running software made for one platform on another platform
- Running software compiled for a CPU on another CPU
- Simulating the hardware in a machine on another machine that does not have the same hardware

Common emulators

- Nestopia
- zsnes
- Visual Boy Advance
- Project 64
- pcsx
- pcsx2
- Dolphin

Why?

- Preservation: old consoles, collector games
- Convenience: no need to have two separate systems to play
- Improvements: resolution, gamepads, turbo mode, 3D

Why are emulators so interesting?

- Very low level
- You basically rewrite a CPU from scratch
- Host and emulated hardware rarely match well
- That means difficult technical challenges

External interfaces

- Audio/Video outputs: composite, composante, HDMI, ...
- Gamepad ports, often a proprietary interface on older consoles, now often USB/Bluetooth
- Storage devices: memory card, SD card, hard drive
- Game media reader: cartridge, CDROM, DVDROM, BDROM, GDROM, ...

Internal components

- CPU(s) of various architectures (MIPS, PPC, X86)
- GPU(s) with fixed or programmable pipeline
- DSP often present to handle sound processing
- Specialized chips for security, data decompression, decryption, hashing, ...
- Network interfaces (Wi-Fi, Ethernet, Bluetooth)
- Various controllers: optical drive, storage device, ...

Software

- Very few game consoles run games on top of an operating system: Wii, PS3, Xbox 360
- On all other consoles games run without anything under them: they have full privileges (ring0, access to hardware, etc.)
- Some consoles (Xbox, PSP, DSi, 3DS) have a "main menu" screen which is not an OS, it's actually more like a game run by default
- Games directly access hardware mapped registers, read/write to physical memory, control CPU caches, etc. None of this is done through drivers, but hardware manufacturors provide an API to game developers

Conclusion

- Not only a CPU but a large set of chips to reproduce
- Very different chips, from GPUs to hardware decryption modules
- Let's see how we can emulate all of this on a computer

High Level Emulation

- wine, cxbx (Xbox)
- No recompilation of the main game code to another ISA
- System calls and standard library calls detection and patching/hooking
- Simulate the behavior of these predefined functions with x86 code instead of simulating the hardware behind the predefined functions

Example

- `sleep(1)`
- Implemented in the standard library by looping while a special register (CPU timer) has not reached a given value
- HLE simply replaces that by a call to the native sleep function

Pros/Cons

- Fast
- Does not require a good precision to work on most stuff
- Does not require a precise knowledge of how the hardware works internally as long as you have high level docs
- Very hard to get a good precision (timing, edge cases)
- Reverse engineering the behavior or getting docs often means infringing copyright and/or patents

Low Level Emulation

- Nestopia, bsnes
- Instead of reproducing high-level behavior, reproduce the low-level behavior of the chips and run the original programs
- Some emulators go as far as emulating the exact behavior of the memory bus between the chips
- Emulates every chip in the virtual hardware independently and make them communicate through emulated bus
- Schedules execution of the chips to have the same timing as the real hardware (number of cycles per instruction for example)

Example

- `sleep(1)`
- With LLE, accesses to the hardware register are detected when they are sent to the emulated memory controller. The timer chip then returns the correct value through the bus
- A lot more work to do in the emulator

Pros/Cons

- Very accurate: you control how good the timing is
- Reverse engineering what runs on the specialized chips is not needed, you only need to reimplement their ISA
- Very slow, requires a lot of sync so multicore is not easy to implement
- ROM dumps for each chip in the console is required to run programs properly
- Some chips are not documented at all and you basically have to reverse their ISA... which is sometimes harder than reversing the ROMs!

Mixing HLE and LLE

- pcsx, pcsx2, Dolphin, Project64
- Uses HLE for some components of the system and LLE for other components
- Most programmable chips need to be emulated with LLE
- In Dolphin: CPU and DSP are LLE, GPU, DSP, DVD, IOS are HLE
- In pcsx2: EE and VU1 are LLE, SPU and GPU are HLE

Example

- `sleep(1)`
- The CPU is emulated with LLE, and access to the hardware register containing the current time is intercepted
- Instead of fully emulating the timer chip, it is emulated through HLE, so the read on the hardware register is translated to a `time()` call

Pros/Cons

- Good compromise between performance and accuracy
- Not relying on very precise timing means you can multithread the several LLE chips more easily
- Sometimes difficult to make HLE and LLE interact properly (exceptions/interrupts for example)

What is bsnes?

- Originally an SNES emulator developped by byuu
- Now supports NES, GB, GBC and GBA (every mainstream 2D Nintendo system)
- Focus on accuracy instead of speed
- Open source project (GPLv3)

Why focus on accuracy?

- Preservation of the consoles (not sold anymore)
- As close as possible to 100% games supported
- No hack to support games

SNES Hardware

- CPU: 3.58MHz R65816
- PPU: 64KB VRAM
- RAM: 128KB
- Games stored in Game Pak (16MB ROM)

GBA Hardware

- CPU: 16.78MHz ARM7TDMI
- PPU: 96KB VRAM
- RAM: 256KB + 32KB + optional 32-64KB in the Game Pak
- Games stored in Game Pak (32MB ROM)

Old hardware

- High-end desktops have 3.5GHz quad-core CPUs
- Should be fairly straightforward to emulate, right?

Synchronization

- Well, not really... Being cycle accurate is far from easy!
- The more accurate you are, the more computing power you require: more accuracy => slower emulator
- Lots of synchronizations between the chips
- Therefore a high overhead

Synchronization

- The whole goal of synchronization is not to execute things on a chip too early (or too late) which might depend on unexecuted things on other chips
- We need predictability over the whole execution
- Preemptive threading is therefore not a good option

libco

- Cooperative threading library developped for bsnes
- `cothread_t co_create(uint size, void (*entrypoint)(void))`
- `fastcall void co_swap(cothread_t new, cothread_t current)`

```
mov [edx], esp
mov esp, [ecx]
pop eax
mov [edx+0x4], ebp
mov [edx+0x8], esi
mov [edx+0xc], edi
mov [edx+0x10], ebx
mov ebp, [ecx+0x4]
mov esi, [ecx+0x8]
mov edi, [ecx+0xc]
mov ebx, [ecx+0x10]
jmp eax
```

Userspace scheduling

- co_switch used to switch between threads when synchronization is needed

```
void CPU::step(unsigned clocks) {
  /* ... */
  input.port1->clock -= clocks * input.port1->frequency;
  input.port2->clock -= clocks * input.port2->frequency;
  synchronize_controllers();
}

void CPU::synchronize_controllers() {
  if (input.port1->clock < 0) co_switch(input.port1->thread);
  if (input.port2->clock < 0) co_switch(input.port2->thread);
}
```

Userspace scheduling

- Scheduler which can re-enter last executed thread with `Scheduler::enter` and get out of it passing an event along using `Scheduler::exit`

```
enum class ExitReason : unsigned {
  UnknownEvent,
  FrameEvent,
  SynchronizeEvent
};
```

Scanline based renderer

- We'll focus on one particular problem : scanline based renderer
- Basically render one entire scanline at a time
- Fast

Demo

Air Strike Patrol running on bsnes v088 (compatibility profile)

Problem

- What if a game is relying on something happening during the rendering of the scanline?
- Air Strike Patrol
- It uses mid-scanline raster effect to show a shadow under the user's plane

Mid-scanline raster effect

- When the scanline is being rendered, developer can alter a few pixels using precise timing
- For Air Strike Patrol, the Display Register brightness is adjusted for a few scanlines in the middle of the screen
- Less brightness => shadow
- Useful as it is a gameplay element (allow better aiming when dropping bombs

Pixel based renderer

- Instead of rendering whole scanlines at a time, pixel are rendered individually
- We now have our shadow!

Demo

Air Strike Patrol running on bsnes v088 (accuracy profile)

Coprocessors

- Quick word on a funny thing about emulating old hardware: coprocessors
- Quite a few games were using coprocessors embedded in the Game Pak to speed up calculations, or do specific tasks
  - Super FX: RISC CPU used to create 3D worlds with polygons
  - DSP-1: Used for 2D, 3D coordinate tramformations
  - SA-1: R65816, same CPU as in the console, but clocked at 10MHz

What is Dolphin?

- Gamecube and Wii emulator - yes, both with the same program
- Open source project (GPL) project
- About 80 total contributors, 10 active
- A lot of users
- Accurate enough to run most games that are released on Wii on day 1 of their release!

Gamecube hardware

- CPU: 486MHz PowerPC (codenamed Gekko)
- GPU: custom ATI chipset (codenamed Flipper)
- RAM: 24MB SRAM + 16MB slower, DMA DRAM
- DSP: 81MHz custom Macronix CPU
- Games stored on DVD with a custom burning method
- 2 memory card slots (EXI), 4 controller ports (serial)

Wii hardware

- CPU: 729MHz PowerPC (codenamed Broadway)
- GPU: custom ATI chipset (codenamed Hollywood)
- RAM: 24MB SRAM + 64MB SDRAM
- DSP: 162MHz custom Macronix CPU
- IOB: 243MHz ARM CPU running an operating system
- Games stored on (larger) DVD with a custom burning method
- 2 memory card slots (EXI), 4 controller ports (serial)
- USB, Bluetooth, Wi-Fi

PowerPC primer

- Designed by IBM, Apple and Motorola in 1991
- 32 bit RISC architecture, 32 registers, 4 bytes instructions
- Separate instruction and data caches which are manually controlled (flush, invalidate, prefetch, disable)
- Several special registers (SPR) used for internal CPU features: disabling interrupts, enabling pagination, etc.
- Used in all modern gaming consoles: Wii, Xbox 360, PS3

Interpreter mode

- Each instruction of the CPU is decoded and executed when it needs to be executed
- Easy to implement but you can't reach 700MHz that way

```
while (true)
{
    u32 instr = fetch();
    handler h = decode(instr);
    h();
}
```

Static recompilation

- Before executing the program, recompile the PowerPC code to X86 code
- Fetch/Decode only done once per instruction
- Does not work: static recompilation can only be used in very specific cases
- If the PowerPC code is modified during execution, then the recompiled program is no longer valid
- We'll talk more about static recompilation later

Dynamic recompilation

- Also called Dynarec or JIT recompilation
- Main emulation method for CPUs in modern emulators
- Recompile blocks of PowerPC code when they are first executed
- When they are executed again, reuse the compiled version
- The hard part is detecting when to invalidate the compiled version because of PowerPC code changes

Block dispatcher

- Small function written in x64 assembly that serves as the entry point of the CPU emulation
- Reads the current PC, checks if there is a jit block already compiled, if not compiles it, then jumps there
- Allows easy indirect jumps emulation: when you don't know where you're going to jump, update PC and jump to the dispatcher
- In Dolphin, also checks for pending interrupts and limits the CPU speed to avoid going too fast

JIT cache invalidation

- Tracking every memory write operation is too slow
- Code and data are in the same memory space so you can't use tricks like guard pages
- Dolphin cheats and uses the fact that games "need" to invalidate the CPU icache manually after rewriting code
- Invalidate a JIT code block when a CPU icache block is invalidated

Constant values propagation

- CVP is a technique mostly used in compilers, but it can also help a lot in recompilation
- PowerPC has no instruction that takes 32 bits immediate values
- Load 32 bits == load 16 bits high then or with 16 bits
- CVP would detect at the first load that the register value is a constant and would not translate the instruction
- When the constant value is used for the first time it is loaded to a register
- Just one example of where CVP is useful... there are a lot more
- Would be even better with VM techniques like partial block specialization (yet to be tried in an emulator)

Register allocator

- PowerPC has 32 general registers, X64 has only 16 registers
- A dynamic way to allocate PPC registers to X64 registers is required
- Dolphin uses a very simple register allocator that uses all available registers and removes the first allocated ones when needed
- Possible improvement, but work on other emulators (NullDC) showed that the gain might be very small

Experiments

- Block merging / block linking: allows jumping directly from one block to another without going through the dispatcher when a jump address is constant
- JitIL: an implementation of the JIT which compiles PowerPC instructions first to an intermediate language composed of several micro-ops, then applies optimizations on the micro-ops stream, and translates the micro-ops to X64. Unfortunately not yet faster than the normal JIT

GC GPU description

- Fixed pipeline: not programmable but a lot of possible states and configuration variables
- That means we can HLE that chip by reproducing its behavior without trying to understand how it is implemented by the real hardare
- The GPU takes commands from the CPU in order to set rendering mode and get vertices, colors, textures, indices

FIFO

- The FIFO is a memory zone from which the GPU reads commands pushed by the CPU
- It's the main synchronization point between CPU and GPU
- The CPU can configure the GPU to get exceptions when the FIFO becomes too full, which makes it hard to emulate in a separate thread
- Continous zone of improvements in emulation stability

Texture cache

- The GC GPU has high speed access to the RAM, so there is no need for texture uploads like on PC
- The texture cache uploads requested textures to the PC GPU on demand and removes textures when they aren't needed anymore
- Currently uses hashing to detect data modification
- Someone is working on using the same system the JIT cache uses (data cache invalidation) to gain performance

Shader generators

- PC GPUs are programmable through shaders, the fixed GL/DX pipeline is implemented through that mechanism
- Dolphin implements the GC fixed pipeline in the same way, generating vertex/fragment shaders on pipeline state changes
- Already compiled shaders are cached in memory, and on disk when using the DX9 emulation backend
- The shaders are where the meat of the GPU emulation is

Vertex loaders

- Vertices are passed to the GPU either through the FIFO (immediate mode) or through a vertex array
- Their format and components are defined by the GPU state
- Parsing the vertices and transforming their data in a form usable by the PC GPU is done by the vertex loaders
- To make the vertex loaders faster, the translation function is compiled dynamically when the vertex format changes
- When possible vectorization is also used in order to translate several vertices at the same time

EFB

- The EFB is the name given to the framebuffer in which graphics are rendered by the GPU
- It can be either accessed directly by the CPU (mapped in memory) or copied to main memory for further processing
- Hard to implement on PC because the GPU and CPU do not share memory and there is no direct framebuffer access
- Dolphin has an option to emulate EFB copies accurately by copying and reencoding the GPU framebuffer to GC RAM
- It can also emulate it in a faster way by copying the EFB to a texture, which is enough for the needs of most games

Possible improvements

- Vertex arrays hashing with a vertex array cache to reduce transfers between CPU and GPU
- Automatic detection of EFB copy mode (to RAM or to Texture)
- Better state tracking to avoid useless changes which cause useless vertices flushes
- Offloading more work on the GPU (OpenCL texture decoding, transform feedback, etc.)

GC DSP description

- Coprocessor handling sound data and mixing them together to get one final stream as an output
- Runs at 1/6 of the CPU frequency
- Tightly coupled with the CPU so his timing has to be very precise in some cases
- Runs code uploaded by the CPU to the DSP RAM

CPU architecture primer

- Separate data/code memory spaces (Harvard architecture)
- Registers have specific uses: index registers, address registers, accumulators, multiplication result, etc.
- Most registers are 32 bits, accumulators are 48 bits
- Most instructions can be followed by an extended opcode which performs another independant operation
- Very domain specific and does not match X64 well

UCode, ROM, coeffs

- The DSP ucode is the program sent by the game to the DSP
- The DSP also has access to a ROM which contains some unrolled mixing functions in order to save space in the ucode
- This ROM cannot be distributed with Dolphin (it's Nintendo IP) so it has to be dumped by the users
- The ROM also contains a table containing coefficients, suspected to be used as FIR interpolator input data
- Very few different ucodes as most developers use the one given by Nintendo in their SDK

HLE

- Because there is only a few ucodes Dolphin can emulate the behavior of each of these ucodes without emulating their instructions directly (HLE)
- Very complex behavior and it's hard to keep the same timing as the ucode
- A lot of games have sound issues with DSP HLE. Most games using sequenced music have missing instruments because the update interval is too large

- Uses the same JIT approach as the CPU emulation
- No cache expiration because code modifications can only be done through upload which is easily detectable
- No register allocator used because all registers can fit in X64 registers
- Because the ISA does not match X64 very well it's hard to get good performance

Possible improvements

- Static recompilation: the only thing missing for that is a way to handle indirect jumps
- Using LLVM in order to optimize the recompiled code
- Improving HLE with a better documentation reverse engineered from the ucodes

What is IOS?

- Wii OS that runs on the Starlet ARM CPU
- Games still run with full privileges on the PPC CPU but must communicate with IOS for some hardware access
- Bluetooth, NAND, USB, Wi-Fi
- This is emulated using HLE, the simple syscall interface makes it very easy to do so

IPC registers

- PPC communicates with IOS through four hardware registers
- 2 readable, 2 writable
- PPC sends the address of a request structure in memory to make a syscall request
- IOS then handles the request, sends a response and triggers an interrupt

Synchronous/asynchronous requests

- Some of the syscalls must be handled synchronously, some can be done in a thread
- For example, Bluetooth needs precise timing, doing it asynchronously could cause problems
- Doing DVD IO synchronously causes stuttering due to hard drive access time
- Both styles are handled in Dolphin in order to get the best out of IOS HLE

Possible improvements

- Implement some missing IOS features (network is in progress)
- Maybe one day IOS LLE? Hard to implement and not very useful, but could be a fun project for someone wanting to do an ARM emulator

Emulation is fun

- Working on a program that emulates a whole complex system is a very interesting challenge
- There is still a lot of room for improvement, and a lot of systems not yet emulated (or not properly)
- Emulation becomes harder and harder because console CPUs are getting very near PC CPUs in terms of per core performance
- You can get started with emulation very easily, write a Chip-8 emulator, then work on Gameboy!

Questions?

- @delroth_, @kalenz
- http://code.google.com/p/dolphin-emu/
- http://byuu.org/bsnes
- #dolphin-emu @ efnet