ARCHI
2019 Lorient

Simulation of HW/SW Systems
A Glimpse into ELS Virtual Prototyping

Frédéric Pétrot tima.imag.fr/sls/people/petrot frederic.petrot@univ-grenoble-alpes.fr

Setting the landscape : System-on-Chip Integration Trend

July 10, 2018 06:37 ET | Source: Energias Market Research
NEW YORK, July 10, 2018 (GLOBE NEWSWIRE) -- The global system-on-chip (SoC) market was valued at USD 33.4 billion in 2017 and is expected to reach USD 128.1 billion by 2024, at a CAGR of 19.3%

Time frame	Nb of SoCs	Devices	Device Maker
2012-2018	22	Kirin	HiSilicon (Huawei)
2007-2018	29	APLx	Apple
2012-2016	33	Atom	Intel
2000-2018	46	SxC and Exynos	Samsung
2003-2019	120	MTx	Mediatek
2007-2018	136	Snapdragon	Qualcomm

(source: Wikipedia articles of the respective device families)

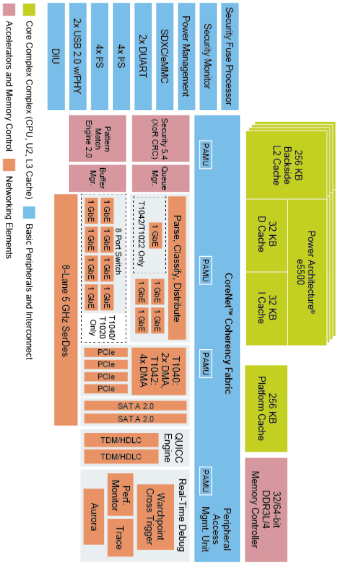
Modern SoCs

Characteristics

- Highly programmable
- Include several to many processors
- With plenty of IPs, some legacy, some ad-hoc
- Based on a few processor architectures :
  - ARM : more or less in every market
  - Power : avionics, automotive, servers
  - MIPS : consumer, networking, automotive
  - Sparc : space
  - RISC-V : hard drives :–)

A Small Example : STM32Fxx SoC

- $\simeq$ 30 IPs
- $\simeq$ 460 registers in IPs
- ? ? ? fields in registers (count hard to automate)



Modern SoCs

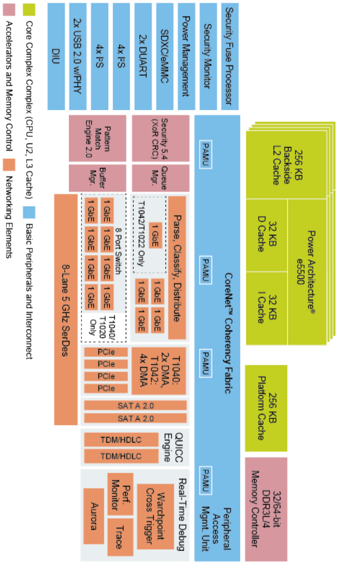
Characteristics

- Highly programmable
- Include several to many processors
- With plenty of IPs, some legacy, some ad-hoc
- Based on a few processor architectures :
  - ARM : more or less in every market
  - Power : avionics, automotive, servers
  - MIPS : consumer, networking, automotive
  - Sparc : space
  - RISC-V : hard drives :–)

How to make sure that the system works ?

Integration issue, not IP per IP validation

Need to check *interactions* within the system



SoC vs PCB

System-on-Chip

Single piece of silicon that includes all electronic components (cpus, memories, peripherals, ...) required to build a system (product)

System-on-Chip $/ =$ Printed-Card-Board

- Connections $\rightsquigarrow \infty$
- Capacitances ≈ 0 (although DRAM stays, as of now, external)
- Industrialisation $\implies$ cost $\rightsquigarrow 0$
- Modification after fabrication impossible !

Design complexity increases I

Technology push

- Number of transistors : +100% every 18 months (Moore's Law) - soon enough it will be over !
- Design productivity : +30% per year
  $\Rightarrow$ **Design Productivity Gap**
- Constant need for new design techniques and tools

Design complexity increases II

Circuit complexity push

- Hardware integration of huge circuits
  - Many complex elements : processors, interconnects, ...
  - Many CPU sub-systems in current SoC (CPU+DMA+Memory+...)
  - Massively parallel integrated computers at hand
- VHDL/Verilog hardly do the job, as by the way to System-Verilog or Chisel
  Even connecting things together becomes an issue
- Nothing like "gates to rtl" for system-level implementation yet
  HLS solves some issues, but not so many (sorry Philippe !)

Simulation goals I

Two main goals

Dimensioning the system

Helps a lot for deciding μArch/Arch parameter values
Bus width, cache size and geometry, number of issues, ...
$\Rightarrow$ Goal is to make educated guesses !

- Functionality not necessary
  $\Rightarrow$ Software doesn't actually run on it !
- Either sampling and replay samples
- Or traffic generation following probability laws

Purely performance estimation oriented
At the end of the day, a replacement to expert excel sheets

Simulation goals II

Virtually prototype the system

Check system consistency

HW/SW relationships, memory maps, device access, ...

Goal is to ensure system bring-up in days !

- Ensures functional correctness of the system
- Runs software on top of hardware models
- Would also like to get figures of merit !

Wants both correct function and accurate estimates

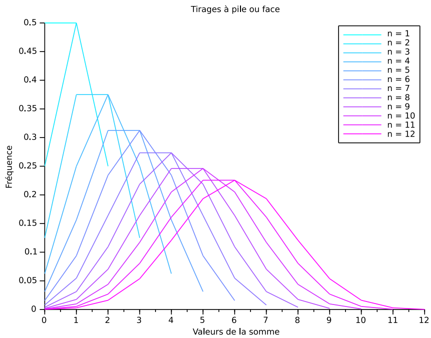
Sample based simulation I

- Mainly used in CPU μ-architectural research
- Based on the central limit theorem
- And on other statistical approaches : χ^2, clustering, etc

Sample-base simulation principle

- Record architectural snapshots
- On actual processor, FPGA, Functional simulators
- And replay snapshots on detailed μArch simulator, HW emulator, ...

(source: "SMARTS : Accelerating microarchitecture simulation via rigorous statistical sampling", Wunderlich et al., ISCA’03)



(source: Cdang, Wikipedia)

Sample based simulation II

Issues

- Quality of the samples
  Profile based characterization
  - Branch mis-prediction behavior
  - Intrinsic ILP or spatial/temporal locality, data reuse distance

  Random time sampling
  - Well, random :−)

  Periodical sampling
  - Allows for speed/accuracy trade-offs
  - Periodical behavior or phases should not match sampling period !
- Multi-thread cores and Multicores
  Very few approaches devised

Reduced input set/Truncated simulation approaches I

Reduced input set

- Limit the size of the working set : smaller arrays/matrices, files, etc
- Keep statistically similar execution profiles
  Not so easy $\Rightarrow$ define the metrics are of interest, and evaluate them all

Reduced input set/Truncated simulation approaches II

Truncated simulation

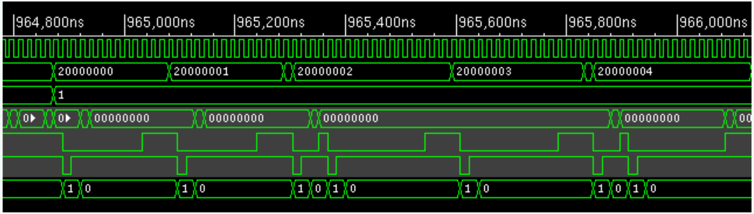
- Run Z
  Simulate accurately the first Z million contiguous instructions
- Fast-forward X + Run Z
  Simulate functionally the X first million instructions
  and accurately the following Z millions
- Fast-forward X + Warm-up Y + Run Z
  Simulate functionally the X first million instructions
  and accurately the following Y million without recording statistics,
  and then the following Z millions

Virtual prototyping

Targets full digital system simulation
Discrete event based

Approaches

Cycle-accurate, bit-accurate (CABA)



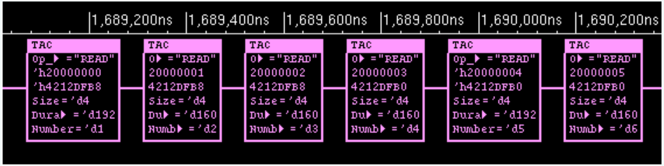
Signal based, cycle per cycle $\Rightarrow$ many events, sloooooowwww

Virtual prototyping

Targets full digital system simulation
Discrete event based

Approaches

Transaction Level Modeling (TLM)

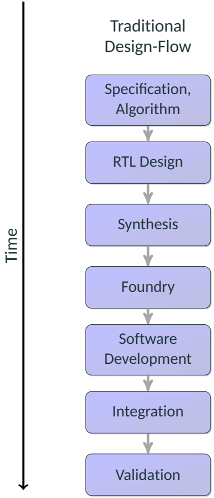


(source: *STMicroelectronics*)

Transactions based $\Rightarrow$ few events, fast

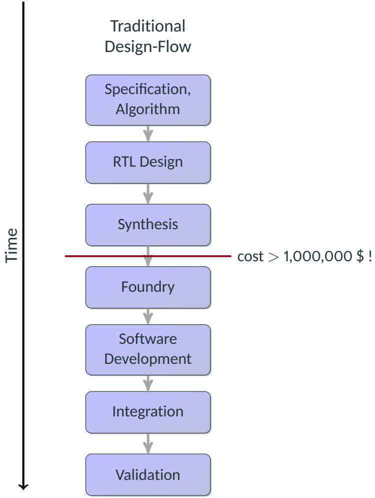
Hardware/software design flow

Courtesy of Matthieu Moy (LIP)



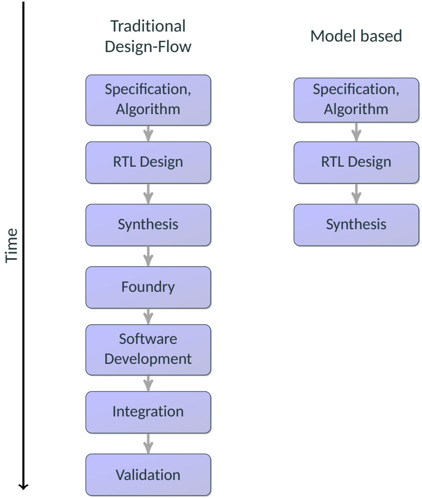
Hardware/software design flow

Courtesy of Matthieu Moy (LIP)



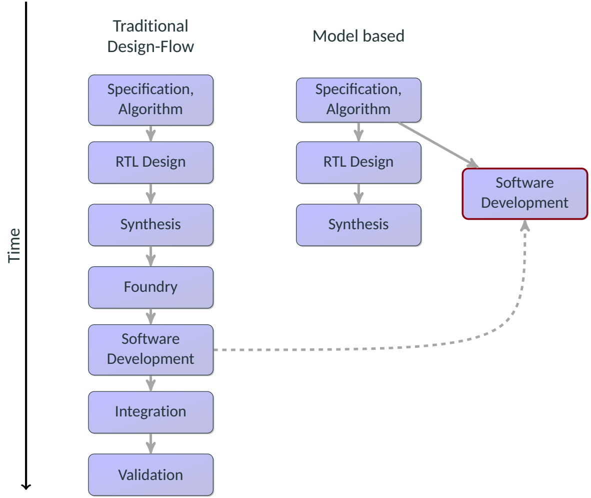
Hardware/software design flow

Courtesy of Matthieu Moy (LIP)



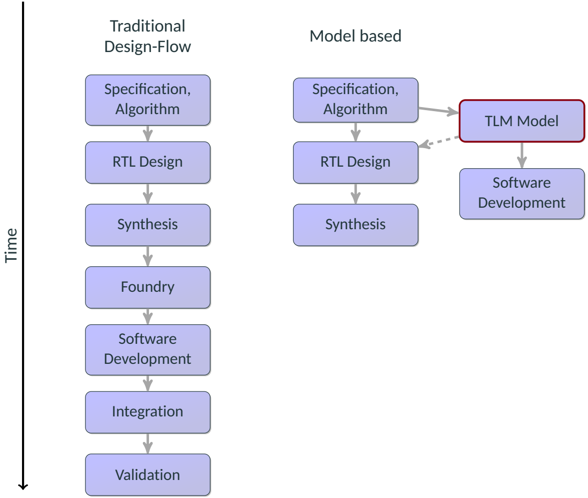
Hardware/software design flow

Courtesy of Matthieu Moy (LIP)



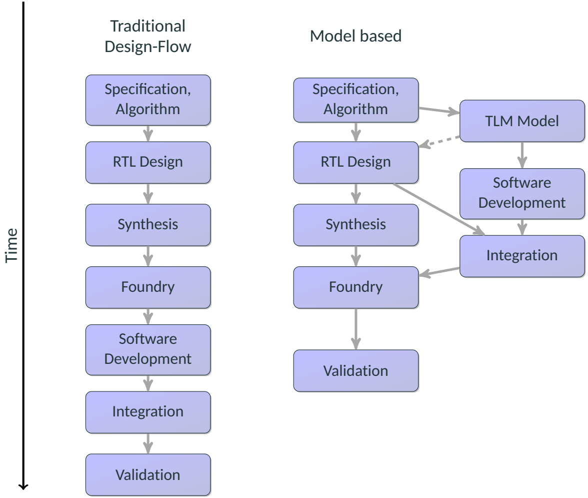
Hardware/software design flow

Courtesy of Matthieu Moy (LIP)



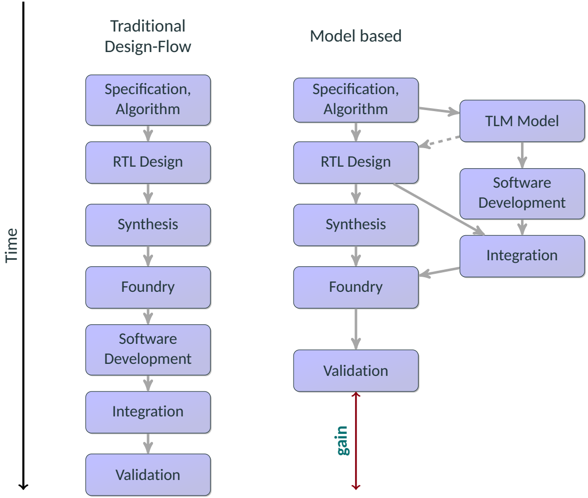
Hardware/software design flow

Courtesy of Matthieu Moy (LIP)



Hardware/software design flow

Courtesy of Matthieu Moy (LIP)



Stringent constraints on the development cycle

- Quick changes in business trends :
  Touch/fold screens, high-density pixels, AI in 'yni', ...
- Some deadlines shall not be missed :
  Christmas, Chinese New Year, Consumer Electronics Show in Las Vegas, ...

Stringent constraints on the development cycle

- Quick changes in business trends :
  Touch/fold screens, high-density pixels, AI in ’yni’, ...
- Some deadlines shall not be missed :
  Christmas, Chinese New Year, Consumer Electronics Show in Las Vegas, ...

⇒ A product that misses its deadline can bankrupt a company :
"One week late, one year late" !

Stringent constraints on the development cycle

- Quick changes in business trends :
  Touch/fold screens, high-density pixels, AI in 'yni', ...
- Some deadlines shall not be missed :
  Christmas, Chinese New Year, Consumer Electronics Show in Las Vegas, ...

$\Rightarrow$ A product that misses its deadline can bankrupt a company :
"One week late, one year late" !

$\Rightarrow$ "Time to market" demands *ad-hoc* design methods and large design teams

How much does an error cost ?

Software bug

- Firmware/Embedded software update
- Sometime easy to realize
  Your smartphone, your box, your Alexia
- Sometimes not :
  Your car, your credit-card, a plane, an orbiter

How much does an error cost ?

Software bug

- Firmware/Embedded software update
- Sometime easy to realize
  Your smartphone, your box, your Alexia
- Sometimes not :
  Your car, your credit-card, a plane, an orbiter

Hardware bug

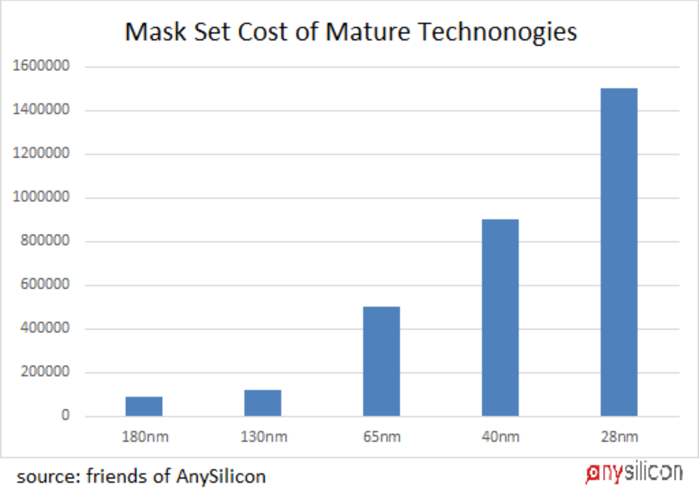
- Respin at foundry
- Cost issues :

Feature size	0.25 μm	0.13 μm	65 nm
1 layer mask cost	\$10 000	\$30 000	\$75 000
Layers	12	25	40
Total cost	\$120 000	\$750 000	\$3 M

source EETimes

How much does an error cost ?

Hardware bug



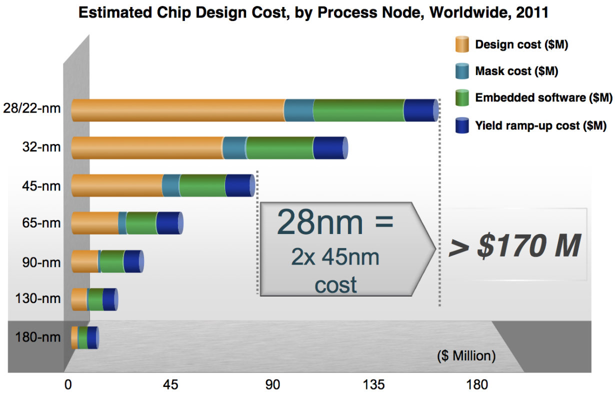
How much does an error cost ?

Hardware bug

- Already fabricated circuit : search for a workaround
  - Software trick, slower but viable
  - Engineering change order (ECO) for mask modification
    Metal patches, spare cells, ...
- SoC FPGA
  - ARM Excalibur : ARM 922 (200 MHz) + FPGA APEX 20KE
  - Xilinx Virtex 4 : PowerPC 405 (450 MHz) + FPGA + Ethernet MAC
  - But
    - FPGA cost $>>$ $10\times$ ASIC fabrication cost for high-volume
    - FPGA power consumption $>>$ $10\times$ ASIC power consumption

How much does an error cost ?

Design Cost



Page 4

XILINX ALL PROGRAMMABLE.

Challenges

When using a SoC

- Debugging software on the hardware is a pain !
  - Boot time configuration : IP reset order, IP clock settings, system setup, ...
  - IP usage, register write-order or timing, drivers, ...
  - Software races, ...
- Developers accesses to the board is “sequential”
- And often require a complex setup

When designing a SoC

- Design space exploration
  - No actual hardware, unreliable hardware, complex setup
  - Co-design issues :
    - Hardware/Software partitioning
    - Which IP kind, which actual IP
    - Evaluation of performance metrics
  - Early software development (see above)

Outline

Simulation to our rescue

A technology that spans all aspects of the design and validation of electronic systems

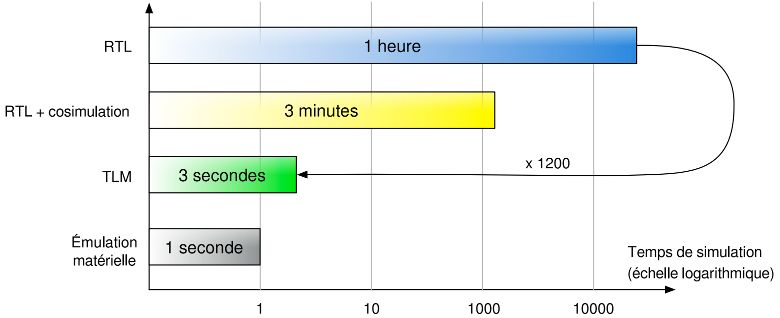
Within *this* presentation

- Simulation of digital hardware/software systems that
  - connect several IPs
  - contain several processors
  - that are actually running code
- Higher level than RTL
- With a focus on fast (and functional) simulation of software on top of hardware

Target : Design issues

- Simulation speed
  - Whole SoC simulation at RTL : several days, if not weeks, . . .
  - Encoding and decoding a single 1280x720 MPEG 4 image
    1 h using RTL simulation (courtesy of STMicroelectronics)
  - No way to test a reasonable OS or even embedded software at this pace
  - Not enough time to validate software and hardware/software integration
  - Partition design in blocks and reuse existing ones
  - Some workarounds
    - Cosimulation
    - Hardware emulation
    - Hardware in-the-loop for legacy IPs

Abstraction levels



Modeling Time Gain	
RTL	1
CABA	3
TLM	10

MPEG 4 image encoding and decoding

(source: *STMicroelectronics* (hence the legend in French))

Estimating Non-functional metrics

Accurate estimation challenging

Speed vs. Accu

- Timing (latency, throughputs, delays)
- Energy/Power
- Temperature

« Truth . . . is much too complicated to allow anything but approximations », John Von Neumann, 1947

« All models are wrong ; some models are useful », George E. P. Box, 2005

Target : Integration issues

- Functional
  - Separated IP design, reuse of existing IPs
  - Hard to ensure that integration works out of the box
  - Not only electrical problems

Target : Integration issues

- Functional
  - Separated IP design, reuse of existing IPs
  - Hard to ensure that integration works out of the box
  - Not only electrical problems
- Performances
  - Capability of a set of IPs to realize a task in a given time
  - Complex non-functional dependencies

Target : Validation issues

- Is the system compliant to its specifications ?
- Specs are more and more complex
  - Audio and video standards : MPEG x, H264, HEVC . . .
  - Weird use cases
  - Spec interpretation issues
- Data volume is increasing : HD, FHD, 4k, 8k, ...
- How do you specify the specifications ?

Outline

Hardware/Software Simulation

Clarification

Simulation :	software model of a hw/sw system
Emulation :	hardware part of a hw/sw system executed on a specific FPGA platforms
Host :	machine on which the simulation runs
Target :	machine which is simulated

Hypothesis

- Event-driven simulation
  - High abstraction level to ensure speed of simulation
- Software is a first class citizen
  - Binary executed on a model of the processor(s)

Hardware/Software Simulation

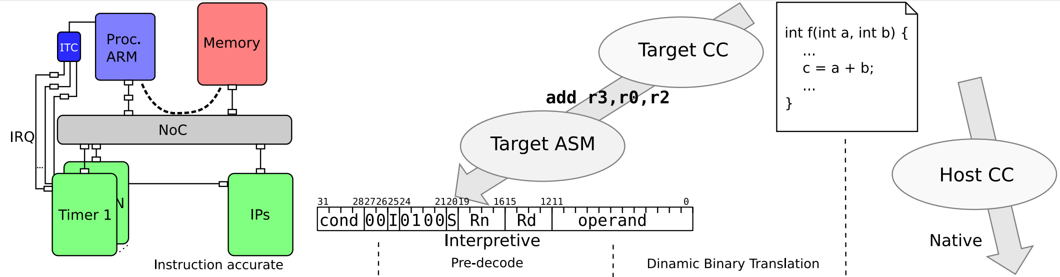
Clarification

Simulation :	software model of a hw/sw system
~~Emulation :~~	~~hardware part of a hw/sw system executed on a specific FPGA platforms~~
Host :	machine on which the simulation runs
Target :	machine which is simulated

Hypothesis

- Event-driven simulation
  - High abstraction level to ensure speed of simulation
- Software is a first class citizen
  - Binary executed on a model of the processor(s)

Software simulation technologies



Fetch and Decode / **Execute**

Instruction accurate:

```
insn = fetch(pc);
op = decode(insn);
switch (op.code) {
case BL: ...; break;
case ADD:
  proc.r[op.dst] =
    proc.r[op.src1] + proc.r[op.src2];
  break;
...
}
```

Pre-decode:

```
if (!(instr = insncache(pc))) {
  insn = fetch(pc);
  op = decode(insn);
  switch (op.code) {
  case BL: ...; break;
  case ADD:
    instr.f = execute_add;
    instr.args = fillargs(op);
    break;
  ...
  }
}
instr.f(instr.args);
```

Dinamic Binary Translation:

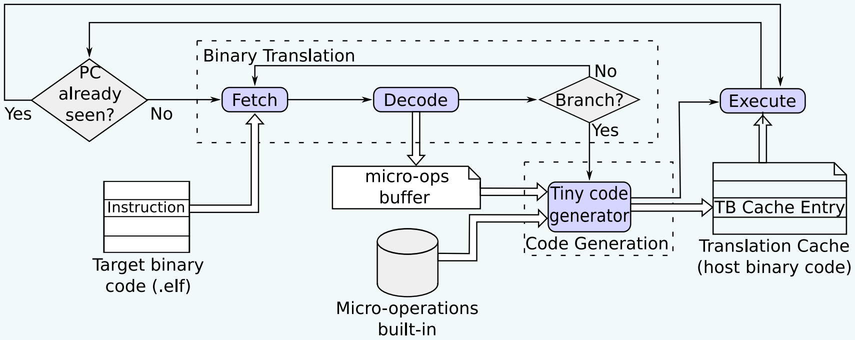
```
if (!(instr = insncache(pc))) {
  while (op != branch) {
    insn = fetch(pc);
    op = decode(insn);
    switch (op.code) {
    case BL: ...; break;
    case ADD:
      append(block,
           gen_add(op));
      break;
    ...
    }
  }
}
block();
```

Native:

```
f();
```

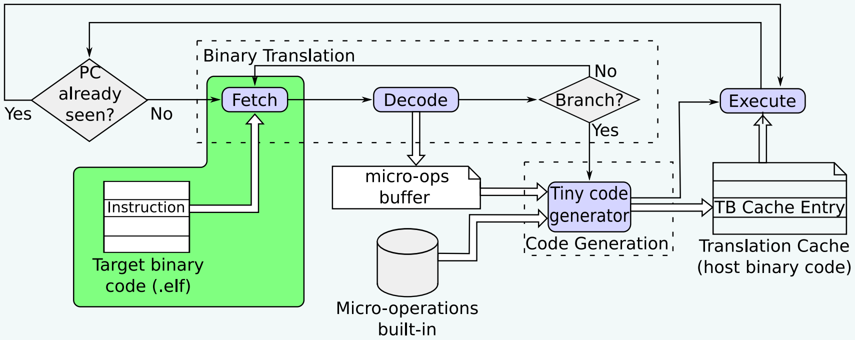
Dynamic Binary Translation Principle

Instruction Interpretation Process



Dynamic Binary Translation Principle

Instruction Interpretation Process

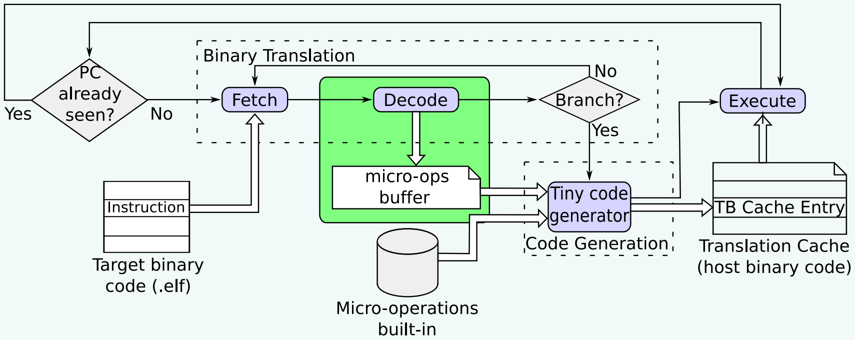


Code Generation Example

```
target_insn_x
```

Dynamic Binary Translation Principle

Instruction Interpretation Process

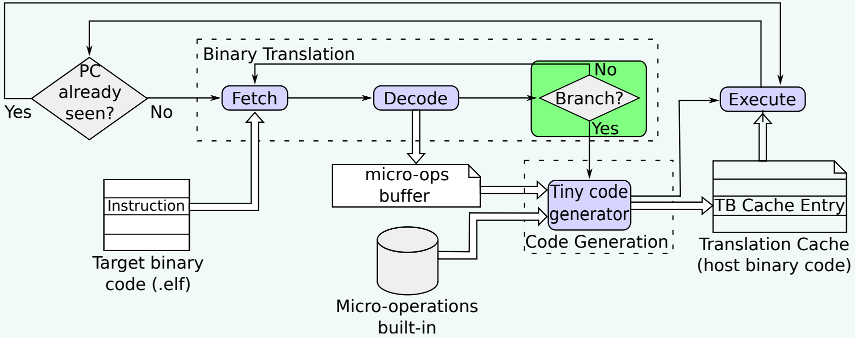


Code Generation Example

```
18   target_insn_x   uop_a
                     uop_b
                     uop_c
```

Dynamic Binary Translation Principle

Instruction Interpretation Process

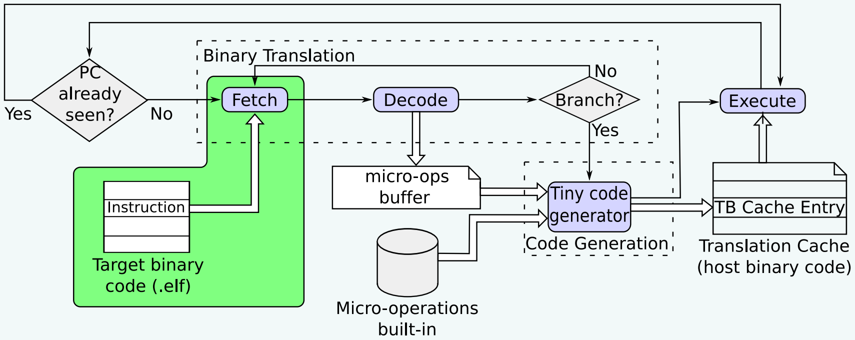


Code Generation Example

```
18    target_insn_x    uop_a
                       uop_b
                       uop_c
```

Dynamic Binary Translation Principle

Instruction Interpretation Process

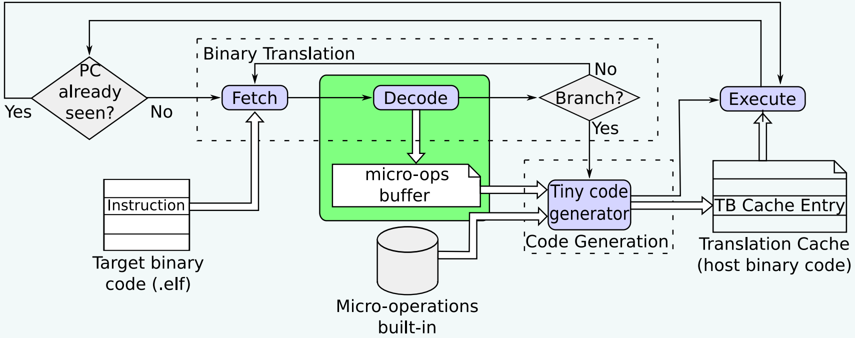


Code Generation Example

```
18   target_insn_x   uop_a   1c   target_branch
                     uop_b
                     uop_c
```

Dynamic Binary Translation Principle

Instruction Interpretation Process

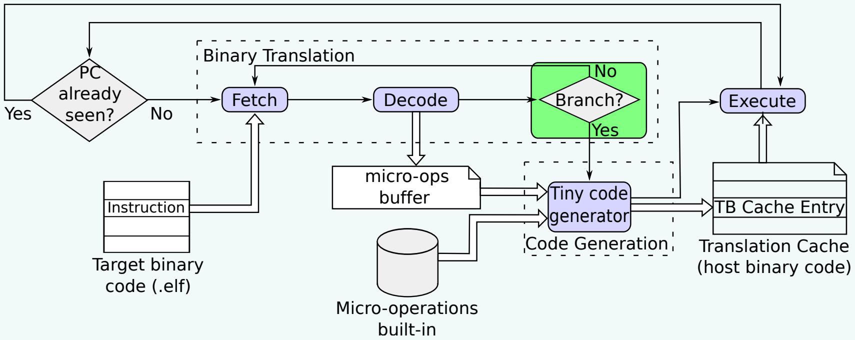


Code Generation Example

```
18   target_insn_x   uop_a   1c   target_branch   uop_d
                     uop_b                        uop_e
                     uop_c
```

Dynamic Binary Translation Principle

Instruction Interpretation Process

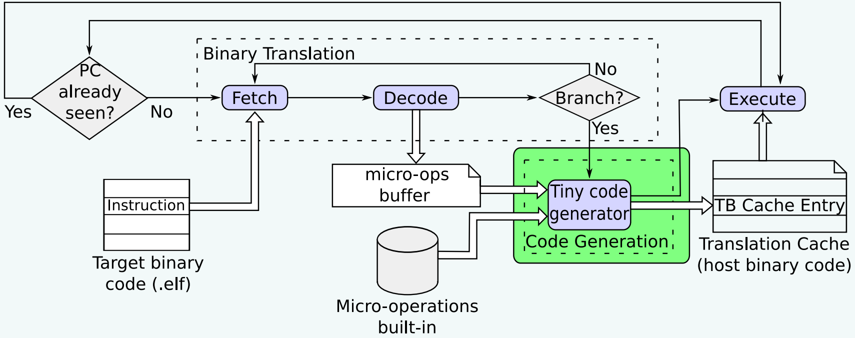


Code Generation Example

18	target_insn_x	uop_a uop_b uop_c	1c	target_branch	uop_d uop_e

Dynamic Binary Translation Principle

Instruction Interpretation Process



Code Generation Example

18	target_insn_x	uop_a uop_b uop_c	1c	target_branch	uop_d uop_e	host_insn_a.1 host_insn_a.2 host_insn_b.1 host_insn_b.2 host_insn_b.3	host_insn_c.1 host_insn_c.2 host_insn_c.3 host_insn_c.4 host_insn_d.1	host_insn_d.2 host_insn_e.1 host_insn_e.2 host_insn_e.3

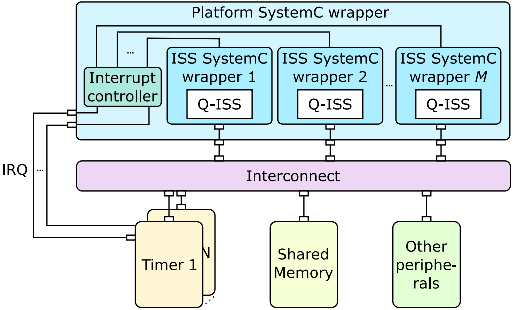
QEMU-SystemC Integration Example

SystemC wrapper : QEMU platform

- Shares QEMU "runtime" and translation cache
- Contains a SystemC wrapper for each processor (including its MMU)
- Connected to interconnect to communicate with SystemC hardware components

SystemC wrapper : processors

- Simulates independently under SystemC control
- Accesses SystemC components by mapping ranges of physical addresses as I/O (except main memory)



TLM components

- Either in SystemC or in QOM, your call !
- Benefits from QEMU existing models

DBT/Discrete Event Integration

Consequences

- Zero time translation-block interpretation
- Execution directly on the host, with TB chaining
  No way for a simulation kernel to step in

⇒ Synchronization with IPs to be defined

Two approaches

- "Closed-loop" timing-aware simulation :
  Timing computed during simulation influences future behaviors
- "Open-loop" strategy :
  Generate memory access traces and computes behavior off-line :
  No influence on future behaviors
  Often used in general purpose computer-architecture research

DBT/DE Synchronization

Synchonization points

- Cache misses (instruction and data caches)
- I/O operations (uncached registers/memories accesses)
- QEMU normal processor simulation breaks *e.g.* interrupt handling
- Predefined period of simulated time without synchronization

Interrupts

- Generated by hardware components as Interrupt pending flags
- Flags viewed by QEMU when SystemC resumes the processors
- Taken into account at the beginning of the next translation block

Code Annotation : Principles

Motivation

Estimate target execution time on the binary translated code

Insert micro-operations to :

- Increment the number of cycles according to the datasheets. Need to take into account registers, data, branch prediction, pipeline data dependencies, ...
- Emulate caches (instruction and data), TLB, branch predictors, ...

Annotation example :

Instr address	Target code	Original translation	Annotated translation	Annotated generated code
addr_instr1	target_instrX	micro-op1_instrX	micro-op1_instrX	host_instr1_micro-op1_instrX
				host_instr2_micro-op1_instrX
				host_instr3_micro-op1_instrX
		micro-op2_instrX	micro-op_annotation	host_instr1_micro-op_annotation
				host_instr2_micro-op_annotation
			micro-op2_instrX	host_instr1_micro-op2_instrX

Code Annotation : Cache Modeling

Simulation speed/accuracy trade-off

- No caches
- Caches as pure directories
  - QEMU memory used (backdoor access   SystemC access through DMI)
  - Two different possibilities varying on the time consumption scheme
    - Cache late : precomputed time consumed at the next synchronization
    - Cache wait : precomputed time consumed when a miss occurs
- Caches full
  - SystemC memory used
  - Search data and instructions over the interconnect
  - Instructions dropped as available from QEMU translation cache

Code Annotation : Cache Details

Instruction Cache

- Where ?
  - At the beginning of each translation block
  - At the beginning of each cache block
- What ?
  - Synchronize simulated cycles
  - Request over the interconnect

Data cache

- Where ?
  - Before each data access (read and write)
- What ?
  - On read miss : synchronize (write-back if wbc), fill cache block using the interconnect
  - On write hit : update the value in cache
  - On write : update the value in memory through interconnect if wtc

Code Annotation : Cache Example

Assumption : cache blocks are 8 words (32 bytes) long

	Instr address	Target code	Original generated code	Annotated generated code
start_tb:	18	instr1_reg_operation	host_instr1_for_instr1 ..... host_instrN1_for_instr1	insn_cache_verify (18); nb_cycles += cpu_datasheet [instr1]; host_instr1_for_instr1 ..... host_instrN1_for_instr1
	1C	instr2_load_from_1000	host_instr1_for_instr2 ..... host_instrN2_for_instr2	nb_cycles += cpu_datasheet [instr2]; data_cache_verify (1000); host_instr1_for_instr2 ..... host_instrN2_for_instr2
	20	instr3_store_5_to_2000	host_instr1_for_instr3 ..... host_instrN3_for_instr3	insn_cache_verify (20); nb_cycles += cpu_datasheet [instr3]; write_access (2000, 5); host_instr1_for_instr3 ..... host_instrN3_for_instr3

Cache Annotation : Accuracy

Monoprocessor results

	SOCLIB	No cache (%)	Cache late (%)	Cache wait (%)	Cache full (%)
Instructions	24114066	-0.00	0.00	0.00	0.00
Cycles instr.	31303545	-0.00	0.00	0.00	0.00
Simulated time ($*10^3$)	50635	-36.70	-0.04	-0.04	-0.04
Sim. speedup	1	553	356	55	28
Sim. slowdown	553	1	1.5	10	20

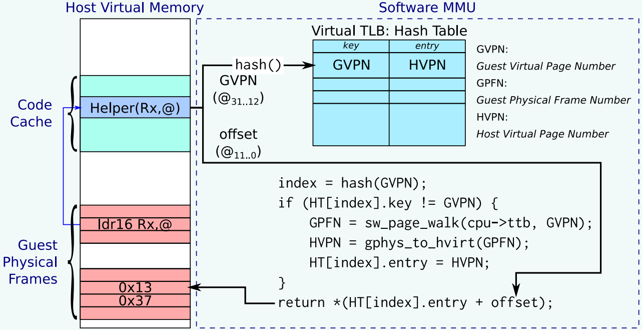
4 processors results

	SOCLIB	No cache (%)	Cache late (%)	Cache wait (%)	Cache full (%)
Instructions	25331336	35.13	22.31	5.24	6.28
Cycles instr.	32931244	34.53	22.01	5.44	6.45
Simulated time ($*10^3$)	19020	-21.07	1.34	-8.44	4.19
Sim. speedup	1	381	246	35	17
Sim. slowdown	381	1	1.5	11	22

Annotation : Caveats I

Hiding (lots of) stuff under the carpet

- Only L1 is modeled, no L2, TLB, MMU, ...
  But that just a matter of effort (and simulation speed)
- Cache model uses *host virtual addresses* ∗<%o(



  gives however no-so surprisingly pretty good results
- Very intrusive into the simulator

Annotation : Caveats II

But there is worse

Experimentation done with a limited number of cores
Simulation speed does not and cannot scale !

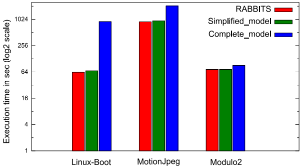
```
void qemu_invalidate_address (qemu_instance *instance, uint32_t addr, int src_idx)
{
    uint32_t dtag = addr >> dcache_line_bits;
    int32_t didx, dstart_idx = dtag & (dcache_lines - 1) & ~((1 << dcache_assoc_bits) - 1);
    uint32_t itag = addr >> icache_line_bits;
    int32_t iidx, istart_idx = itag & (icache_lines - 1) & ~((1 << icache_assoc_bits) - 1);
    int32_t i;

    for (i = 0; i < instance->m_NOCPUs; i++) {
        if (i != src_idx && (didx = dcache_line_present (i, dstart_idx, dtag)) != -1)
            instance->m_cpu_dcache_flags[i][didx].valid = 0;
        if ((iidx = icache_line_present (i, istart_idx, itag)) != -1)
            instance->m_cpu_icache_flags[i][iidx].valid = 0;
    }
}
```

Change in runtime : Branch Prediction

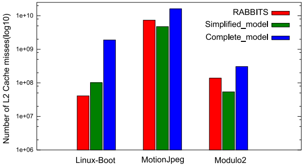
Done when exiting translation blocks

- No need to annotate at code generation time
- But not as easy as it seems :
  Large BP tables lead to host cache trashing slowing down simulation

⇒ Need proper high level branch predictor models to be usable
Seznec *L-TAGE* example from cbp3



Execution times in seconds without/ with abstract/with full *L-TAGE* predictor



Number of host L2 cache misses during simulation

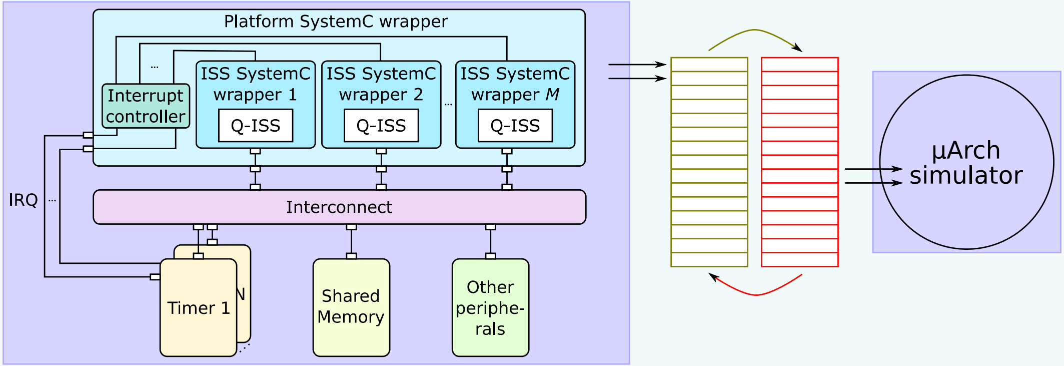
"Open-loop" approach I

Principle for cache simulation

- Log memory accesses, cache control instructions and TLB control instructions
- Replay the events on a focused memory hierarchy simulator

"Open-loop" approach II

Possible implementation



"Open-loop" approach III

Pros and Cons

Pros :

- Benefits from the parallel nature of the host
- Focused detailed simulator is hopefully faster than full system simulator
  e.g. branch prediction, which can even be fully accurate !
- Intrusiveness in full system simulator (relatively) low

Cons :

- Execution flow not altered by timing
  Caches or TLB misses
- Occurrence of external events unchanged
  Timer and other interrupts would change states
- Must evaluate the "divergences"

Outline

Sequential DBT Acceleration

QEMU Binary Translation

Guest code

```
ldr  r3, [r7, #4]
```

Generated Host Code

Get target virtual address

```
 0: movl 0x1c(%r14), %ebp
 1: addl $4, %ebp
```

Compute virtual TLB index and tag

```
 2: movl %ebp, %edi
 3: leal 3(%rbp), %esi
 4: shrl $5, %edi
 5: andl $0xfffffc00, %esi
 6: andl $0x1fe0, %edi
 7: leaq 0x2cf0(%r14, %rdi), %rdi
```

Compare tag and call slow path or continue

```
 8: cmpl (%rdi), %esi
 9: movl %ebp, %esi
10: jne  0x7f2234b234d4
```

fetch data at host virtual address

```
11: addq 0x10(%rdi), %rsi
12: movl (%rsi), %ebp
```

Slow Path Trampoline Code

Prepare architectural state and operation index

```
0: mov r14,%rdi
1: mov oi,edx
```

Prepare function return address

```
2: lea -0x7f(%rip),%rcx #line 11
```

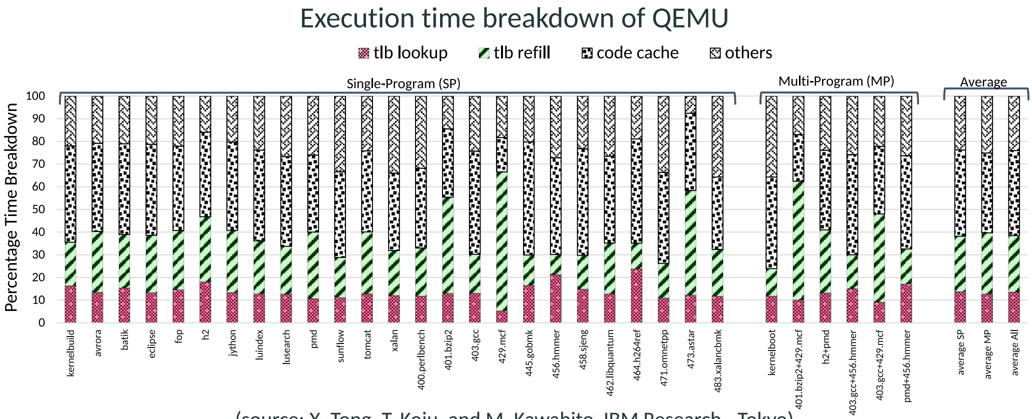
Call softmmu handler

```
3: mov $0x55fc967adec0,%r10
4: callq  *%r10
```

Retrieve handler return value and return to generated code

```
5: mov %eax,%ebp
6: jmpq 0x7fad8fcd8202  #line 11
```

Sequential DBT Acceleration



(source: X. Tong, T. Koju, and M. Kawahito, IBM Research - Tokyo)

Address translation

Floating point emulation, uses *helpers* as of today

Detect hot-paths and optimizes them (see IBM Hotspot Java VM)

Parallel DBT Acceleration

Use host multicore nature

Implement target AMO/sync instructions as host AMO/sync instructions

- Trivial, isn't it ?
- Not really !
  - AMO/sync instruction semantics are not identical
    `test-and-set/fetch-and-incr/fetch-and-add/cas/ll-sc/...`
  - Target/Host memory consistency models differ
    x86 and x64 have strong consistency model => nice hosts
    Arm has weak consistency model => need sync everywhere as host

In QEMU

MTTCG : Parallel executions of processors using host AMO/sync
Works only for Alpha (!) and ARM on x86-64 for now

Parallel DE Acceleration

PDES : Has been a research topic for long

- Needs large chunks of parallel code execution
  Synchronization is killing simulation speed
- Needs a viable parallel semantic, one that SystemC doesn't have !
  "Seven Obstacles in the Way of Parallel SystemC", Rainer Dömer, UC Irvine

Outline

A few words on benchmarks I

Benchmark : a set of programs covering all the aspects of program execution "differently"

- Program performance should not dramatically improve by trivial optimization
  Counterexample : Dhrystone
- Program characteristics should be complementary and exercise different behaviors
  Static control vs dynamic controls
  Arrays vs graphs
  Streams vs arrays, ...

A few words on benchmarks II

Popular benchmarks

SPEC
For general purpose computing architecture research
De facto standard, SPEC-INT and SPEC-FP, several generations
Neither open-source nor free

Polybench
Set of static control compute intensive kernels mainly for compilers
Also useful to evaluate processor simulators, free and open-source

Coremark
Target embedded MCU
Neither open-source nor free, very industry oriented

MiBench
Target embedded systems, free and open-source

Splash2
For parallel processing architecture research
Using the `pthread` and not much beyond that, free and open-source,
Considered by some a bit old

Parsec
For parallel processing architecture research
Rely on many libraries, hard to run without a Linux kernel
Considered more up-to-date, free and open-source

A few words on benchmarks III

Another popular benchmark

Linux boot

Free and open-source

Benchmark and usage

- Measure metrics for all programs in benchmark
  If not, explain why !
- If needed, run on top of an OS
  Papers report large variations between bare-metal and OS versions
- The more, the better
  But need clear explanations of results not a bunch of numbers !

Time for "name dropping" !

- SMARTS : sample based
- SNIPER : reduced input based
- Gem5 : full system, processors cycle approximate
  Memory hierarchy, NoC, hard to say
- SoClib :full system, processors cycle approximate
  Memory hierarchy and NoC cycle accurate on the interfaces
- QEMU : full system, no metrics other than instruction count

Quick summary

Simulation is a useful technology

- No need to be functional to perform accurate metric estimations
  At least for uniprocessor systems !
- Functional simulation however very useful for SoC design
  Fast processor simulators use DBT, open-source solution available
- Accurate estimation of power and timing still on-going research
  Although it has been on-going for decades : (