
An Integrated Approach to Architectural
Simulation, Timing and Memory Hierarchy
Evaluation

Erik R. Altman

Rene Miranda

Jaime Moreno

IBM T.J. Watson Research Center

C. Brian Hall

IBM Toronto Laboratory

Background

- Pre-hardware performance estimates are needed for new architectures and new implementations of existing architectures
- Simulators and cycle timers are slow
- Trace-driven simulators often require two passes and storage of a large amount of data
- **Result:** Not always possible to simulate meaningful programs to completion
- *Standard Approaches to Achieve Result:*
 - Sampled traces
 - Small input sets
 - Expensive special purpose acceleration hardware

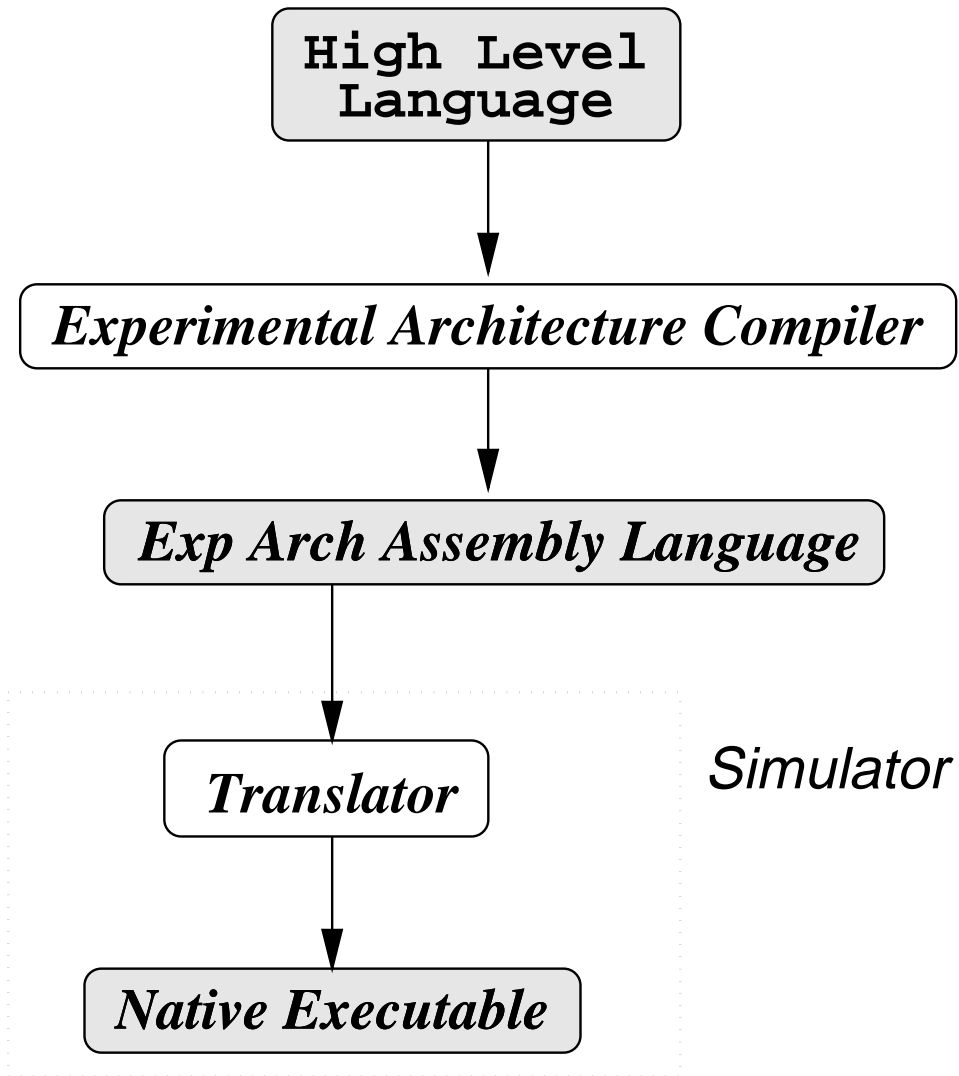
Goal

- A tool with fast turn-around time to evaluate architectural features
- A tool which can easily interchange simple and complex models
- A tool which allows a tradeoff of speed for accuracy in preliminary estimates
- A tool which runs acceptably on normal workstations

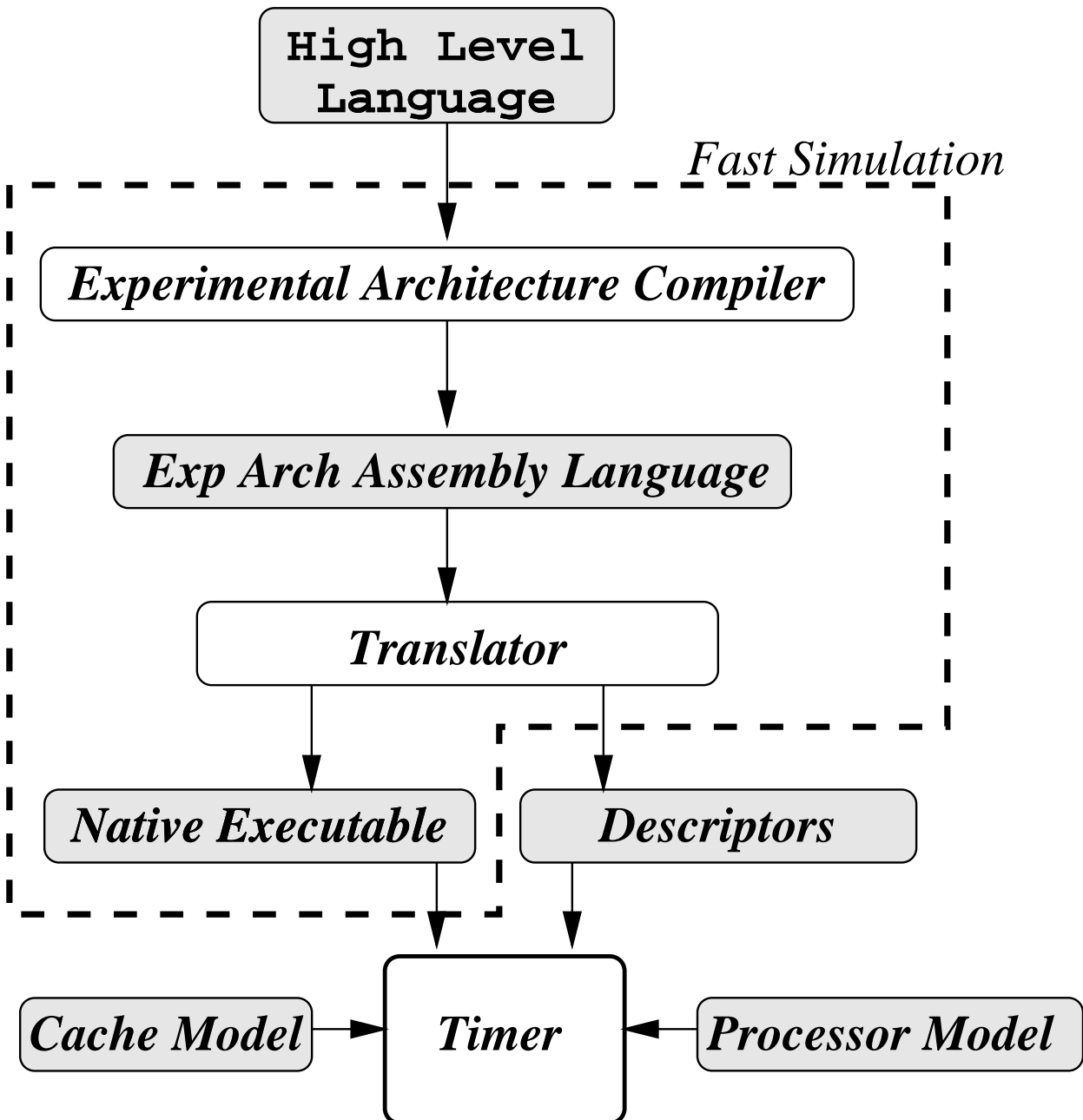
What Do We Propose?

- A *Simulator* and *Timer* for experimental architectures
- Native code directly emulates architecture
⇒ No interpretation needed
- *Timer* invoked by *Simulator* at every instruction
- *Timer* has 2 parts:
 - Processor Model
 - Memory Model

System Overview



System Overview



Descriptor Information

- Registers read by instruction. *List includes implicits such as CC.*
- Registers written by instruction. *List includes implicits such as CC.*
- Memory address components
- Is memory op?
- Is speculative op?

Mapping to Native Executable

Instruction in Experimental Architecture

```
add_shift r37, r4, r3, 4 #Exper Ins
```

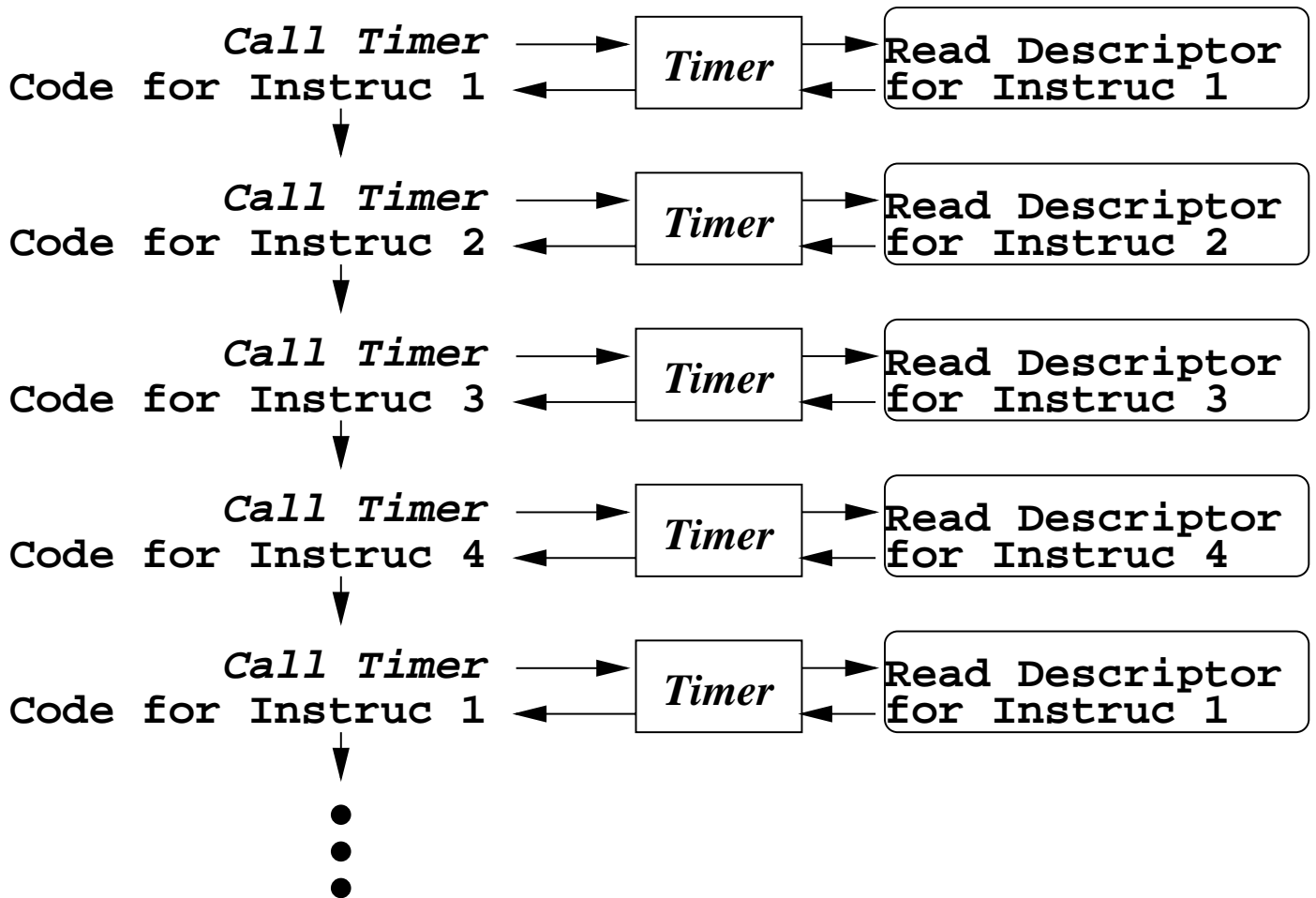
Translation into Native Code (PowerPC)

```
lwz    r31, R4_OFFSET(r13)    #Registers
lwz    r30, R3_OFFSET(r13)    #memapped
add    r31, r31, r30          #Do add
sli    r31, r31, 4            #Do shift
stw    r31, R37_OFFSET(r13)   #Store result
```

Descriptor

Registers read:	R3, R4
Registers written:	R37
Memory Op:	<i>No</i>
Memory Components:	<i><None></i>
Speculative Op:	<i>No</i>

Timer Execution



*Dynamic Sequence
of Instructions
from
Native Executable*

*Static Set of
Descriptors*

Processor Model Interface

- *Timer* called before *Simulator* (*native executable*) executes instruction \Rightarrow *Timer* can read contents of input registers
- *Simulator* calls *Timer* with
 - *Instruction address* in experimental code
 - *Descriptor address*
 - *Pointer to machine state* (*i.e. registers*)

Internal Function of Timer

- *Timer* reads instruction *Descriptor* from *read-only* storage
- *Timer* invokes **MemQuery** function of *Memory Model* to find which old memory requests are now complete
- *Timer* requests/updates resources (e.g. registers) used by instruction
- *Timer* invokes **MemRequest** function of *Memory Model* with memory reference(s) from current instruction. Each reference is tagged with unique identifier returned by **MemQuery** when request complete.
- *Timer* increments cycle counter and invokes **Mem-Cycle** function to tell *Memory Model*
- *Timer* returns to native executable

Key Points

- Modular and detailed interface between components
- Flexibility, Speed \Rightarrow No separate traces needed
- Simulation and timing are fast
- Components and Results \Rightarrow Results to any desired accuracy
- Memory requirements proportional to *static* size of program
- Single executable can be used with different Memory and Processor Models
- Detailed timing model can be used only for selected source files \Rightarrow Not waste time simulating uninteresting portions of program

Example Model: VLIW with Simple Pipeline

Characteristics:

- Function unit level model. No detailed pipeline.
- Conflicts accessing multi-bank register file
- Varied function unit latencies
- Cross-chip transmission delay
- Overflows in 11-bit address generation

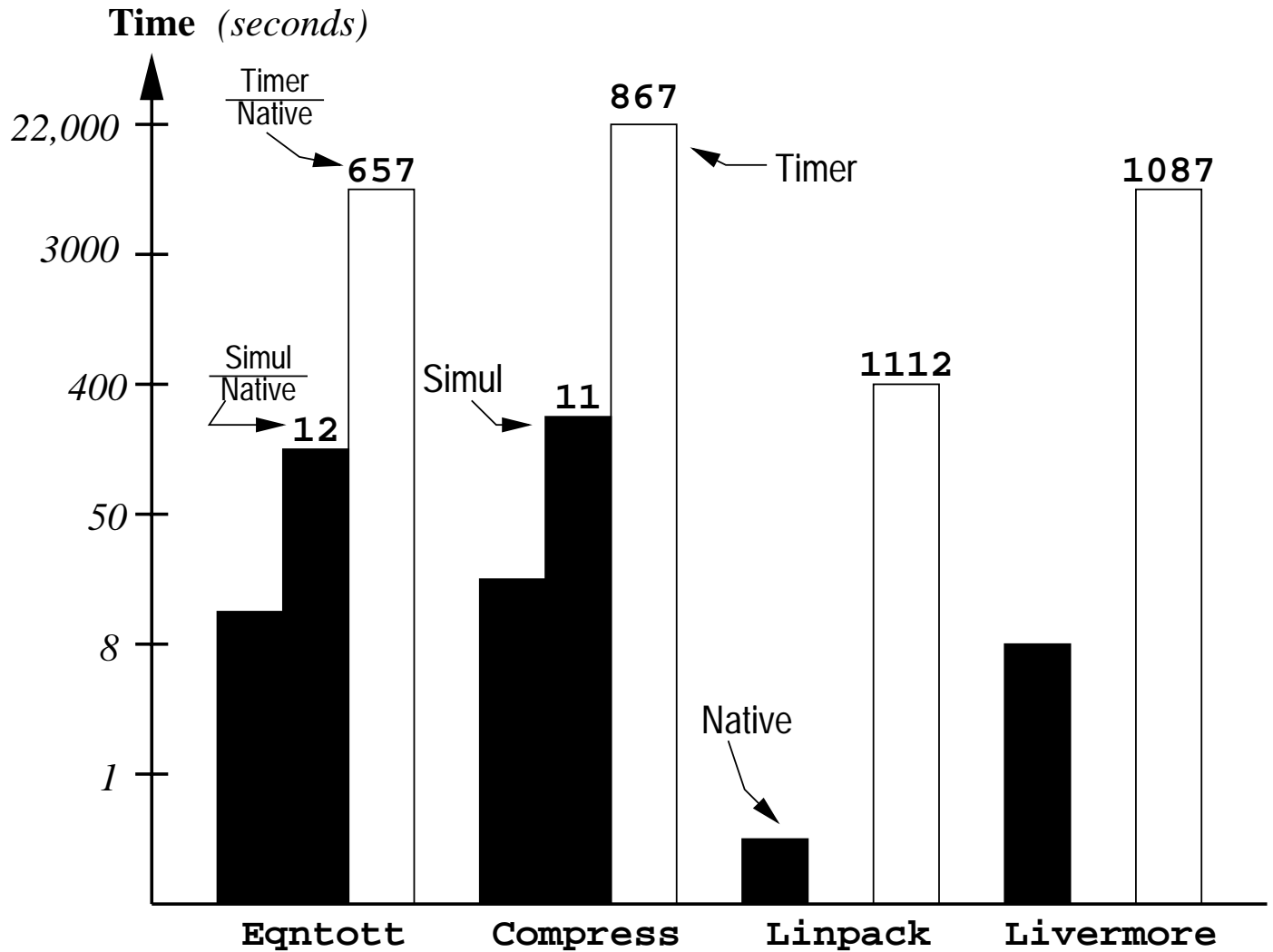
Observations on Example:

- VLIW has tree instructions consisting of many (e.g. 16) simpler PowerPC type operations
- \Rightarrow Good fit to *Simulator/Timer*. Overhead incurred once per VLIW instruction, not once per PowerPC operation
- *Note:* Input to *Simulator/Timer* is VLIW assembly language generated by VLIW compiler

Memory Model

- Lockup Free Cache
- Up to 3 Level Cache Hierarchy
- Arbitrary Associativity, Cache Size, and Line Length
- Partial fills not modelled
- Infinite Queues

Simulation Speed



General Results:

- *Simulator* 5-15 times slower than native
- *Timer* about 75 times slower than *Simulator*
- *Timer* about 1000 times slower than native

Keys to Efficiency

Architecture Independent Keys

- Direct emulation of architecture with native code. No interpretation needed
- Decoded descriptors used by *Timer* for each instruction \Rightarrow Descriptors produced at compile time, but used at runtime
- Processor and memory models are run-time configurable with environment variables \Rightarrow recompilation not needed to try architectural variants

Architecture Dependent Keys

- Opcodes in experimental architecture similar to those in native architecture
- Machine state in experimental architecture similar to native architecture (e.g. condition codes maintained in like fashion)
- The ability to efficiently process several operations at each invocation of the timer (as is possible with a VLIW architecture for example)

Surprises

- Initial *Timer* implementation decremented counters each cycle on pending resource until resource was ready. Later versions instead associated completion time with each resource.

Result: A 3-fold speedup in the *Timer*.

- Execution time of *Timer* largely independent of number of stalls in simulated program.

Conclusions

- Proposed *Simulator/Timer* approach is suitable for experimental architecture evaluation
- *Simulator/Timer* has fast turn-around time for compiler/architecture tradeoffs

Future Work

- More detailed processor and memory models
- Additional Benchmarks (e.g. **SPEC95**, **TPC**)
- Investigate other Parallel Architectures