New Directions in Binary Translation

Kemal Ebcioğlu
Global Supercomputing Corporation, NY, USA
06/19/2010
Binary Translation Workshop – ISCA 2010
St. Malo, France
Overview of a historical accomplishment of binary translation

– BT allows one to use new architectural features without having to change the base architecture or its software.
  – Translate a code fragment of the base architecture machine that does not have the feature
  – to a program of the BT machine that has the feature
  – in a manner transparent to the user.
– Retains the customer investment.
– Possibly including complete OS emulation
– Transmeta Crusoe, IBM DAISY and BOA, IBM DIF,...
Possible advantages of a new BT machine

• Higher performance
• Lower power
• Less design complexity
• More reliability
• Better security
• More efficient virtualization
• Better debugging capability
• Your favorite architectural features...
BT machine architecture style does not matter

• Binary translation originally emerged as a way to facilitate transition from an existing architecture to a particular new architecture
  – Silberman and Ebcioglu, IEEE Computer June 1993, for IBM S/390 to VLIW
  – Transmeta Crusoe, for Intel x86 to VLIW
• However, the BT technology is orthogonal to the style of the BT machine architecture
• BT can be used for taking advantage of any architectural enhancement.
What makes BT work?

1. A new BT machine and a method to program it
2. Compilation mechanism to compile a base architecture binary code fragment to the BT machine code representation.
3. 1 and 2 should satisfy: Code fragment execution on BT machine should have results indistinguishable from base architecture execution.
4. Ability to switch from base machine to BT machine when translated code fragment entry point is reached
5. Ability to switch from the BT machine to the base machine at the end of a translated code fragment
Additional requirements for dynamic translation

• *Static* binary translation uses off-line profiling, more expensive compiler optimizations. Translation is done once. Simple run-time.

• If *dynamic* binary translation is used, we also need:
  1. Statistics gathering mechanism during base architecture execution, for detecting hot regions.
  2. Faster, just-in-time compilation and optimization mechanism
Meaning of BT execution being indistinguishable from base execution

• Given an entry register state, program counter and memory contents before executing the code fragment

• If the base machine completes the code fragment starting in that state:
  – The BT machine final register state, program counter and memory contents upon exit from the code fragment should be identical to results of base execution.
  – Except for memory locations and registers that are dead on the exit
    • E.g. procedure stack frame contents after returning from a procedure are dead

• If the base architecture execution of the code fragment does not terminate, neither should the BT execution
Some goals in compilation of the code fragment

- Pure Performance: Execution time should approach critical path length of execution trace of base code fragment.
  - Critical path must include base engine-BT engine communication penalties as well.

- Performance/watt
- Performance/area
- Redundancy for reliability
- ...
BT execution being indistinguishable from base architecture is difficult in the presence of parallelization

• Difficulties have been solved:
  – Indirect branches
  – Precise exceptions
  – Memory consistency
  – Control speculation and data speculation
  – Memory mapped IO
  – Emulation of source architecture memory mapping
Example: load-store telescoping with precise exceptions
(Ebcioglu, Altman, Gschwind, Sathaye
Optimizations and Oracle Parallelism with Dynamic Translation
Proc. MICRO-32, 1999)

- Execute operations and loads with aggressive speculation as soon as their operands are ready, but commit results to architected registers and memory in original order
  - When exception occurs, one can pinpoint the original base instruction \( n \)
  - All base instructions up to \( n \) have been completely executed
  - No base instructions after \( n \) has been executed
Example: load-store telescoping with precise exceptions (2)

**Original code**
- Addi r3=r4,1
- Store r3,0(r5)
- ...
- Store r6,0(r7)  //r5=? r7
- If (cr0.eq) goto L1
- Load r8=0(r5)
- Addi r9=r8,2

**Binary translation**
- Addi r3=r4,1
  - Addi r9’=r4,3;;
- Store r3,0(r5);;
- ...
- Store r6,0(r7)
  - If (cr0.eq) goto L1;;
- LoadVerify r3,0(r5)
  //trap if r3 != *r5
  - Copy r8=r3
  - Copy r9=r9’;;
Requirements of a BT machine

• Hardware Software co-design of a new engine is a must.
  • Your favorite DSP is not a good binary translation engine for x86, PPC or Zseries, or SPARC!

• BT machine must have basic operations for supporting efficient base architecture emulation
  • Should not simulate a x86 parity flag in software!
Requirements of a BT machine (2)

- Extra non-architected registers. Required for emulating CISC with RISC.
- Correct endianness
- Correct support primitives for emulating memory management and IO model and privilege levels of the target machine
- Correct memory coherence/consistency support
Additional requirements for compile time parallelization

• If the target machine has compile time parallelization, we need, e.g.:
  • Many extra registers
  • Memory coherency between base engine and BT engine
  • Control speculation support (e.g. 33rd bit)
  • Data speculation support
  • Thread level parallelization support
  • Features for reliability, security
Degrees of interpretation

• BT engine execution
  – Direct compilation of hot code fragment to parallel ASIC circuits
• New general purpose parallel execution ISA implemented in ASIC circuits

• Base engine execution
  • Direct implementation of base ISA in ASIC, using possibly an existing design
  • Software interpretation of base engine on BT engine (suitable for low cost solutions)
Degrees of interpretation (2)

• Dynamic BT just in time compiler
  • Implementation of JIT compiler in dedicated ASIC
  • Implementation of JIT compiler in software
• Dynamic BT, instrumenting base execution for detecting hot regions
  • Implementing instrumentation in base execution hardware
  • Software interpreter with statistics collection
• Large design space, one powerful idea
Some BT visions
(Altman, Ebcioglu, Gschwind, Sathaye
Advances and Future Challenges in Binary Translation and Optimization
Proc. IEEE, Nov. 2001)

• Convergence BT architecture for multiple popular base architectures
  • BT engine has union of features to supports all base engines
    – Advantages: Optimize a simple BT architecture to obtain benefits on all base architectures.
• Convergence software virtual machine:
  – Single VM and OS Implemented with optimizing just in time compilation on many different architectures
  – Write once, run everywhere capability
  – Similar to .NET, java VM vision
  – More effective hardware resource usage in data centers with multiple virtualized server architectures
A future vision for BT: Extreme Scale Computing

• Uniprocessor performance has reached a plateau
  – Frequency
  – Design complexity
  – Power
  – Memory wall

• The most popular remedy is multi core, but:
  – Parallel programming is difficult.
  – There is a programming productivity challenge

• OS services cause serialization and have unacceptable overheads at extreme scale
A future vision for BT: extreme scale computing (2)

- Cloud computing and virtualization are excellent, disruptive technologies
- Cloud computing achieves unprecedented economies of volume and efficiency
- Cloud computing will be a must for extreme-scale computing
- But cloud computing is neither scalable, nor reliable nor secure enough for use in critical extreme-scale applications.
BT’s place in Extreme Scale Computing

• Making extreme-scale computing work will require many new architectural features
• Multi-cores plus traditional VLSI scaling are unlikely to succeed by themselves
• Binary translation could be at the heart of such a solution for utilizing the new architecture features, while not breaking the base architecture execution model.
Closing

• Beware of complexity
  – VLIW was supposed to be a simple architecture, but IA-64 turned out to be the opposite
  – Success of an engineer is about complexity management
  – Parkinson’s law
    • Work expands so as to fill all available time
  – Do not let this happen!

• Be bold, take research risks, but keep it simple