Achieving High Performance via Co-Designed Virtual Machines

J. E. Smith
Tim Heil
Dept. of Electrical and Computer Engr.
University of Wisconsin-Madison

Subramanya Sastry
Todd Bezenek
Computer Sciences Dept.
University of Wisconsin-Madison

1. Introduction

Today's virtual machines use a layer of software that allows programs compiled in one instruction set to be executed on a processor executing a (different) native instruction set. Virtual machines have become popular in recent years for providing platform independence; however, virtual machines also open many new opportunities for enhancing performance. The co-design of virtual machine software and the underlying hardware microarchitecture will enable enhanced instruction level parallelism and more adaptable performance mechanisms than are possible when hardware and application software are separated by instruction set architectures as is traditionally done.

In future high performance computers, a *virtual instruction set architecture* (V-ISA) will be the level for maintaining architectural compatibility. The V-ISA will be implemented with a virtual machine that blends software and hardware in a symbiotic manner via co-design. The hardware will support an implementation-dependent architecture, or *implementation instruction set architecture* (I-ISA). As such, the I-ISA has special features keyed to the specific hardware optimizations built into the microarchitecture. The co-designed virtual machine implementation will include fast compilation of the V-ISA to the I-ISA, efficient hardware performance feedback, optimizing re-compilation, and adaptive hardware performance features to enable performance improvements beyond those that can be achieved with conventional hardware and software.

In a sense, software implementing the virtual machine will become available for the hardware microarchitect to use -- in contrast to the current paradigm where the hardware designer is isolated from software via a fixed, often inflexible instruction set architecture. This application of virtual machines is also in contrast with most current VM proposals which are focussed more on maintaining compatibility and portability across platforms supporting existing ISAs, and whose performance objectives are typically no more ambitious than the minimizing performance losses that occur in the process.

2. The Search for High Performance

Processor microarchitecture has been evolving for many years. Performance increases have primarily come from exploiting larger amounts of instruction level parallelism (ILP) and from increased use of prediction and speculation. Microarchitectures have evolved from serial to pipelined to superscalar implementations. For this evolution to continue, however, it is becoming evident that a substantially different, more far-reaching approach must be used. Current obstacles to increased performance include:

(1) Limited visibility to ILP; it is well known that the "window" of instructions visible to the hardware has to be increased to provide higher ILP. Using hardware alone to increase the instruction window size typically leads to greater complexity. For substantial advances to take place, it is likely that the instruction window will have to become larger than hardware alone can support. Potentially, compiler software could make more instructions visible in the window, but software's view is static, and is often restricted. In particular, the object-oriented programming paradigm is becoming widely adopted, and by its nature, the compile-time visible instruction window can be limited due to the many method calls and dynamic linking.

(2) Instruction set compatibility has become a performance obstacle; it is difficult to change an existing instruction set to enhance performance. In fact, it is sometimes inadvisable, because changes that enhance performance in one generation may lead to added design complications (with no performance benefits) in future generations, e.g. delayed branches. Furthermore, removing old instructions is difficult because maintaining compatibility with older generation software is very important.

(3) Designers often make tradeoffs to provide the best performance when averaged over a number of application programs. However, these tradeoffs may not give the best performance for any of the individual programs, or for phases of larger programs. The ability to adapt for a specific program or program phase is often limited by decisions made at hardware design time.

(4) Researchers have become very clever in devising hardware performance enhancements. Usually these are targeted at some particular aspect of the design and give a relatively small overall performance improvement. Although each of these features makes sense by itself, any individual program may only be able to take advantage of a few of them. And, implementing them all tends to weigh down a clean pipeline structure with many small, complicated, special-purpose "appendages" -- possibly to such an extent that the overall complexity wipes out the individual benefits.

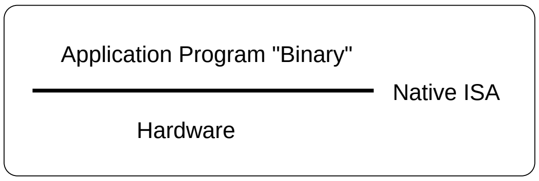
VM co-design can overcome all of the above obstacles and will open many new opportunities for computer architects. It can provide a wider scope for finding ILP, can allow the hardware designer considerable flexibility in ISA design, can allow more dynamic adaptivity in hardware performance features, and can allow a much cleaner pipeline design by removing hardware appendages (putting much of the complexity in software where it can be selectively applied). Furthermore, it can provide high performance for object-oriented programs.

3. Virtual Architectures

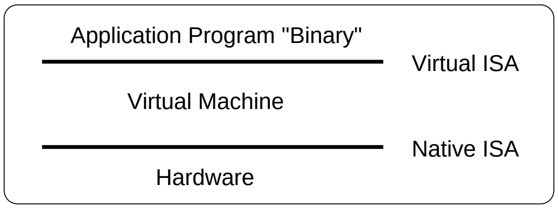
Traditional processor architecture is based on a hard division between hardware and software; see Fig. 1a. As pointed out above, this clean hardware/software division has provided a number of advantages, but, in the end, its inflexibility is also inhibiting.

The virtual architecture abstraction provides revolutionary opportunities to computer architects. This abstraction has become popular recently in the form of Java bytecodes and virtual machine (VM) implementations. Fig. 1b illustrates the VM as it is used in the Java environment. The VM layer effectively widens the traditional ISA separation between hardware and software. Now, application software is compiled to the virtual architecture specification (i.e. Java bytecodes); meanwhile the hardware is designed for some existing instruction set -- often referred to as the "native" ISA. The VM is implemented with a software layer that is placed between the traditional hardware and software layers. The VM and the underlying hardware implement the V-ISA through interpretation, just-in-time compilation, adaptive recompilation, or some combination.

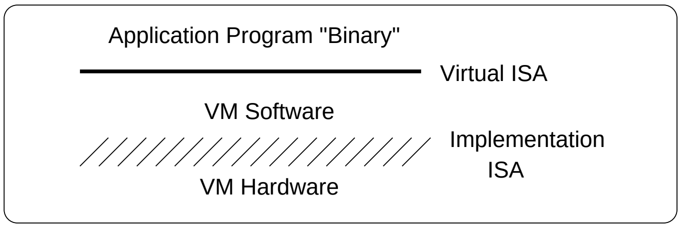
A primary goal in using VMs in the manner just described is to provide platform independence. That is, the Java bytecodes can be used on any hardware platform, provided a VM is implemented for that platform. In providing this additional layer of abstraction, however, performance is typically lost because inefficiencies in matching the V-ISA and the native ISA via interpretation and just-in-time (JIT) compilation. The V-ISA and the native ISA are defined completely independently, so there is no real opportunity for the two to mesh together well. Consequently, a typical performance goal in this environment is to provide performance "almost as fast as" native ISA execution.



a) Conventional hardware/software interface



b) Conventional virtual machine interface



c) Envisioned VM with hardware/software co-design

Figure 1. Relationships among Software, Hardware, and Virtual Machines

Through VM co-design, the V-ISA abstraction can be exploited to exceed native processor performance. This approach is illustrated in Fig. 1c. The VM software and underlying hardware are co-designed. The underlying native ISA is implementation dependent, and is *not* an existing ISA as is the case in Fig. 1b. We refer to the lower level ISA as the "implementation ISA" (I-ISA), to emphasize its implementation dependence. Because the ISA is implementation dependent, hardware and software along this boundary can be co-designed so that the division between the two becomes blurred. Flexibility enables distribution of features between hardware and software for optimal performance.

In the current environment, Java bytecodes are an obvious choice as a V-ISA, but the V-ISA could, in fact, be a conventional RISC or CISC ISA. Or, if it is desirable to support a conventional "native" ISA and Java bytecodes, both could be supported (or for that matter multiple ISAs could be supported) by the same I-ISA. In this case, the I-ISA may be better suited for some of the V-ISAs than others -- but at least the designer can choose where the tradeoffs should be. And, finally, because platform independence is a good feature, it is still provided at the V-ISA level.

4. VM Implementation

Fig. 2 illustrates the co-designed VM system we envision. A V-ISA "binary" is executed by a combination of VM hardware and software. The software may be phased in, as is becoming common with Java VMs today. That is, the software may compile to the I-ISA on demand ("Just in Time" or JIT) or it may be interpreted initially with compilation occurring only after usage is determined to be high enough. The initial JIT compilation will likely be a fast and simple one. Then, as the program runs, the hardware and software work cooperatively to optimize the program's execution. The hardware collects relevant performance information that is in a form efficiently obtained and interpreted by the software; i.e. it is designed to match software's needs.

The ways in which hardware and software can communicate to cooperatively optimize performance offer many opportunities for innovation. For example, consider hardware passing performance information to software. The hardware can be designed to collect performance information that is keyed to specific features that are in the implementation at hand. Performance data can be collected on a per-instruction basis via software selection, or for all instructions, or for all instructions within a program region. Performance information can be passed to the

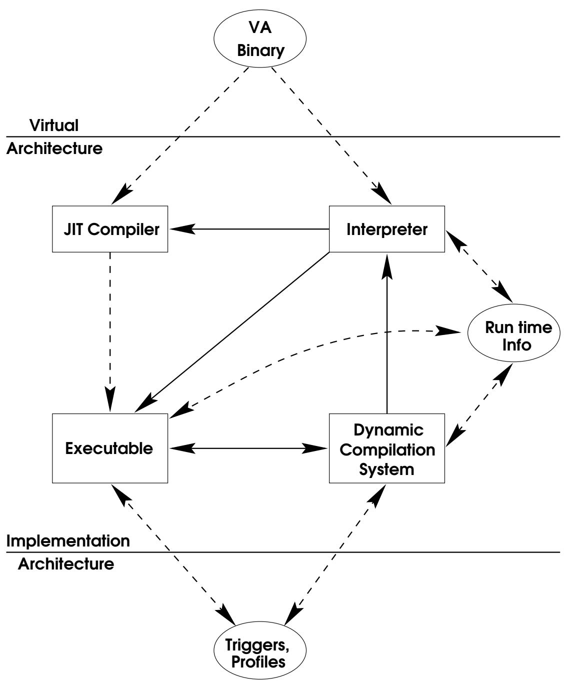


Fig. 2. Overview of co-designed VM system.

software via instructions that read performance data by polling, or there may be hardware "triggers" that cause VM software to be invoked when a certain event occurs, or when some performance characteristic is found to shift outside an operating envelope -- as when a program changes processing phases.

Going in the other direction, software can manage hardware features through implementation-dependent instructions that are tailored to specific hardware performance features and are executed as part of the dynamically re-compiled user binary. For example, one might have (a number of) prefetching load/store opcodes to control which memory operations trigger prefetches as well as the type of prefetch. VM software can also directly execute special instructions to "tune" hardware performance features to match the characteristics of user code currently being executed. For example, an instruction may reach into the hardware and adjust a branch predictor's global history length for a particular program or for a certain region of a program.

The optimization process can occur more than once; in fact it will typically occur many times as a program executes, to continue adjusting to changes in the program or data. The intermediate form or re-compiled binary persists during the programs execution, and may or may not persist between program executions -- this tradeoff is one of the many research issues.

5. Performance Optimizations

Many of the problems at the forefront of microarchitecture research can benefit from the VM co-design approach. It is these research problems that led us to begin developing this approach. Following subsections discuss some of the microarchitecture research problems in which we are most interested and the ways in which co-designed VMs can be used to help solve them.

5.1. Trace Selection and Control Independence

Maintaining a large and accurate window of dynamic instructions is important for any high performance microarchitecture. In trace processors [ROT97], high performance instruction supply relies on accurate trace prediction and good trace cache performance.

There is also the potential of maintaining a large instruction window in spite of mispredicted branches by exploiting *control independence*. A trace is control independent of a prior branch if the trace is fetched/executed regardless of the outcome of the branch; this typically occurs when the two paths following a branch re-converge before the control independent traces. Control independent traces do not have to be squashed and restarted if the branch is mispredicted.

A recent study we have done [ROT99] explores the potential and limitations of control independence in the context of superscalar processors. Control independence can potentially reduce the performance gap between real and oracle branch prediction by half. In a detailed implementation, performance improvements on the order of 30% are observed.

Trace processors offer a complexity-effective solution for implementing control independence mechanisms -- the distributed window organization naturally isolates traces from one another. However, to harvest the potential of control independence in trace processors, traces must be selected to expose the control independent points (re-convergent points) of the program. That is, the hardware trace selection algorithm (heuristics used to divide the dynamic instruction stream into traces) must identify control independent points, and then terminate/begin traces at these or related points.

Current trace selection is constrained to fairly trivial decisions because hardware has a limited view of the control flow within a program. While hardware may be able to expose some amount of control independence, a new level of sophistication that only a layer of software can provide may be necessary. Software can determine control independence relationships more precisely. But more importantly, sophisticated heuristics can consider many variables to selectively and intelligently apply the control independence information. The variables include both static information from the control flow graph and dynamic profile information from the hardware.

VM co-design provides the vehicle for this kind of trace selection. First, software can implement the otherwise overly-complex analysis. Second, there is a transfer of information between software and hardware, through the I-ISA: hardware provides dynamic profile information, and software supplies the hardware with enough information to select good traces. Co-designed trace selection can likewise be applied to improving trace prediction accuracy and trace cache performance.

5.2. Improving Control Prediction with Data Values

Conventional branch predictors use the history of past branch outcomes to predict future outcomes. Branch prediction methods have become more elaborate, and clever ways have been found to make predictors more efficient, e.g. by reducing aliasing. However, performance improvements are probably leveling off; there is only so much information that can be extracted from the stream of outcomes.

For substantial improvements in branch prediction, we need to use additional information. In particular, we have found that certain hard-to-predict branches can be made much more predictable by using data values -- i.e. the results of non-branch instructions. However, folding data values into the branch prediction process in a productive way is very difficult because there is such a large number of data values to choose from, and there are some branches where using data values hurts predictability.

We are considering VMs as a method for solving this problem, by having the software experiment on-the-fly with branch prediction algorithms that use data values. This is similar in spirit to the hardware tuning method for global branch histories in [JUA98]. Furthermore, opcodes in the implementation ISA can be used to enable/disable the use of data values on a per-instruction basis, if needed.

5.3. Memory Systems for Object Oriented Applications

The increasing relative delay of memory is a growing performance problem. Consequently, using a VM to help manage memory hierarchies appears to be a good application of our approach. Furthermore, memory management issues seem especially acute in object-oriented programs where small data structures are continuously being created, used, abandoned, and garbage collected.

Co-designed VMs can be used to improve memory hierarchy performance by reorganizing cached data and for improving cache hierarchy management (i.e. constraining the highest cache where a data item can reside.) These methods can be tightly integrated with features in the performance monitoring architecture to collect performance data on a per-instruction (or per region) basis. Optimizations include the ability to re-organize memory data -- permitted by Java semantics, and the ability to prefetch and otherwise move data in the hierarchy based on type information available in Java bytecodes. Such information can be directly communicated via the I-ISA, by using special opcodes, for example. And performance feedback from the hardware can allow dynamic runtime optimizations based on usage patterns.

5.4. Performance Tradeoffs

For VM co-design to work in the way envisioned, there are some important performance tradeoffs that must be resolved. In particular, the overhead of performing optimizations in software must be offset by performance gains eventually realized. This means that very fast and simple software methods must be used. This points to study of algorithms that are at different points in the cost (in time) and performance curve. And it also suggests hierarchical implementation of different algorithms, with more complex, better optimizing algorithms being invoked only for portions of a program that are heavily used.

6. Summary

Using a V-ISA layer with hardware/software co-design of the VM will provide a powerful new way of dynamically optimizing executing programs. Important optimizations will exploit observed run-time performance characteristics. And these optimizations can involve re-compilation on-the-fly or adjustments made to underlying hardware performance mechanisms. Optimizations can be done on a broad scale; not limited by a relatively small issue "window" as in purely hardware approaches. In effect, higher performance results from combining strengths of software compilation and dynamic hardware execution.

It is becoming very apparent that microarchitects need another avenue for expanding the search for instruction level parallelism. Co-designed VMs will likely provide such an avenue. Not coincidentally, a number of recent proposals have been put put forward for systems similar in objectives and/or approach to those of VM co-design. A number of these are listed in the bibliography.

Acknowledgements

This work was supported in part by NSF Grant MIP-9505853, by the U.S. Army Intelligence Center and Fort Huachuca under Contract DABT63-95-C-0127 and ARPA order no. D346, by an IBM Partnership Award, and by Sun Microsystems. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U. S. Army Intelligence Center and Fort Huachuca, or the U.S. Government.

7. Selected Bibliography

[ADV97] Sarita V. Adve, Doug Burger, Rudolf Eigenmann, Alasdair Rawsthorne, Michael D. Smith, Catherine H. Gebotys, Mahmut T. Kandemir, David J. Lija, Alok N. Choudhary, Jesse Z. Fang, Pen-Chung Yew, “Changing Interaction of Compiler and Architecture,” *IEEE Computer* 30(12), pp. 51-58, December 1997.

[AMM97] Glenn Ammons, Thomas Ball, James R. Larus, “Exploiting Hardware Performance Counters With Flow and Context Sensitive Profiling,” *Prog. Lang. Design and Impl.,* pp. 85-96, June 1997.

[CHE98] A. Chernoff, M. Herdeg, R. Hookway, C. Reeve, N. Rubin, T. Tye, S.B. Yadavalli, J. Yates, “FX!32 - A profile-directed binary translator,” *IEEE MICRO,* 18(2) pp. 56-64, March-April 1998.

[CON98] T. Conte, "Evolutionary Compilation to Long Instruction Word Microarchitectures for Exploiting Parallelism at All Levels, ASPLOS 98 Wild and Crazy Ideas Session, Oct. 1998.

[DEA97] Jeffrey Dean, James E. Hicks, Carl A Waldspurger, William E. Weihl, George Chrysos, “ProfileMe: Hardware Support for Instruction-Level Profiling on Out-of-Order Processors,” *Thirtieth Symp. on Microarch.,* pp. 292-302, December 1997.

[EBC96] Kemal Ebcioglu, Erik R. Altman, “DAISY: Dynamic Compilation for 100% Architecture Compatibility,” IBM Research Report RC 20538, August 1996.

[EBC97] Kemal Ebcioglu, Erik R. Altman, Erdem Hokenek, “A Java Processor Based on Fast Dynamic VLIW Compilation,” *Intl. Workshop On Security and Efficiency Aspects of Java,* January 1997.

[FIS97] Joseph A. Fisher, “Walk-Time Techniques: Catalyst for Architectural Change,” *IEEE Computer* 30(9), pp. 40-42., September 1997.

[GRI98] David Griswold, “The Java(TM) HotSpot(TM) Virtual Machine Architecture, A White Paper About Sun’s Second Generation Java(TM) Virtual Machine,” Sun Microsystems White Paper, http://java.sun.com/products/hotspot/whitepaper.html, March 1998

[HOL95] Urs Holzle, “Adaptive Optimization for Self: Reconciling High Performance with Exploratory Programming,” Sun Microsystems Laboratories Technical Report TR-95-35, 1995.

[HSI97] Cheng-Hsueh A. Hsieh, Marie T. Conte, Teresa L. Johnson, John C. Gyllenhaal, “Using NET to Capture Performance in Java-Based Software,” *IEEE Computer* 30(6), pp. 67-75, June 1997.

[JUA98] Toni Juan, Sanji Sanjeevan, Juan J. Navarro, “Dynamic History-Length Fitting: A third level of adaptivity for branch prediction,” *Fifth Intl. Symp. on Comp. Arch.,* pp. 155-166, June 1998.

[MEN97] K. N. P. Menezes, “Hardware-based profiling for program optimization,” Ph.D. thesis, Dept. of Elec. and Comp. Eng., North Carolina State Univ., 1997

[ROT97] E. Rotenberg, Q. A. Jacobson, Yiannakis Sazeides, J. E. Smith, “ Trace Processors,” *Thirtieth Intl. Symp. on Microarch.,* pp. 138-148, December 1997.

[ROT99] E. Rotenberg, Q.A. Jacobson, J.E. Smith, “A Study of Control Independence in Superscalar Processors,” to appear Fifth Intl. Symp. on High Perf. Comp. Arch., January 1999.

[SOL97] Frank G. Soltis, *Inside the AS/400,* Duke Press, 2nd Ed., 1996.