

Heterogeneous Chip Multiprocessor Design for Virtual Machines

Dan Upton and Kim Hazelwood

University of Virginia

ABSTRACT

Multicore architectures provide an alternative to increasing clock frequencies to improve performance of modern processors. The best design for these chip multiprocessors, including structure sizes or whether to use homogeneous or heterogeneous cores, remains open for exploration. One potential design path involves using heterogeneous cores that are specialized for a given task. This paper examines the design of a core specifically for the execution of virtual machines. While virtual machines create opportunities for runtime code modification for tasks such as optimization, instrumentation, and security, they lead to overhead at run time. Using cycle-accurate simulation, we characterize a virtual machine in terms of functional unit usage, cache usage, and branch behavior. We also discuss additional structures and instructions necessary for supporting a virtual machine. By moving the execution of the virtual machine to a separate core and parallelizing its execution with that of the application, we hope to mitigate some of the overhead of execution; a smaller, specialized core can also provide savings in power consumption and die size and complexity.

1. INTRODUCTION

Modern processors are rapidly reaching the limits of performance that can be obtained by simply increasing clock frequency. Physical concerns such as wire delay and inconsistencies in the fabrication process, as well as power consumption and heat dissipation, decrease the effectiveness of simply increasing transistor count. Additionally, an upper bound on performance exists due to limits on the instruction-level parallelism (ILP) of a single thread.

Newer processor designs have taken this ILP limit into account by creating opportunities to take advantage of thread-level parallelism (TLP), including simultaneous multithreading [22] and chip multiprocessors (CMPs) [19]. CMPs may lead to greater performance improvements, as fewer structures are shared and thus the complexity for managing shared structures is reduced. Further, heterogeneous CMPs [14] provide the opportunity to mix processors of varying power, or tailor cores specifically to a class of applications [13], to improve performance or meet constraints. However, the problem remains of ensuring enough threads to keep all available cores busy.

At the same time, virtual machines (VMs) have become increasingly common. VMs are used for such applications as dynamic optimization [3], instrumentation [16], security [10], and binary translation between instruction sets [8]. VMs can also be used to virtualize the hardware to run multiple op-

erating systems in parallel [21, 5]. These systems provide opportunities at run time not available at compile time, at the cost of execution overhead.

In this paper, we explore the differences in run time behavior between VMs and many applications for which general-purpose processors are designed. This observation motivates optimizing a core of a heterogeneous CMP for execution of a VM. By doing so, we are able to provide a use for extra cores of a CMP, as well as mitigate some of the overhead of executing under control of a VM. We characterize the behavior of a VM to aid design decisions at a structural level. Further, we discuss other design issues related to additional structures necessary for synchronizing execution of a VM and application on separate cores.

The rest of the paper is organized as follows. Section 2 discusses background of heterogeneous CMPs and VMs. Section 3 characterizes a VM's performance in terms of functional unit usage, branch behavior, and cache behavior. Section 4 discusses additional structures for the VM core. Section 5 discusses related work, and section 6 concludes and presents future directions.

2. BACKGROUND

Heterogeneous CMPs. Heterogeneous CMPs consist of multiple processors with characteristics varying from processor to processor, ranging from higher-level details such as instruction set to microarchitectural details such as clock frequency, issue width, and cache size. This heterogeneity can be a part of the core's design or a result of the fabrication process. Performance asymmetry can lead to decreased performance if both the operating system and application are unaware of the heterogeneity [4]. However, work by Kumar *et al.* has suggested the potential for power savings and weighted speedup by intelligently scheduling processes on a heterogeneous CMP [11, 15]. Further work showed that heterogeneous CMP design targeting different cores to specific application classes can increase performance over that obtained by combining general-purpose cores [13].

Virtual machines. Virtual machines can be grouped into two main categories, system VMs and process VMs. System VMs provide virtualization of the hardware and execute an operating system and all of its processes. They may either reside between the hardware and operating system, as in Xen [5] or the Transmeta Code Morphing Software [8], or act as an application on another operating system, as with VMWare [21]. In order to provide certain guarantees of isolation and correctness when virtualizing hardware, many of these systems have to emulate instructions; to reduce over-

head, they may use some of the same binary optimization techniques discussed below.

Process VMs are generally designed to execute a single program per instantiation of the VM. They allow more fine-grained selection of what is executed under a VM’s control and what the VM does with a given program. For example, given a VM used to enforce a security policy, a user may opt to sacrifice performance of one application to use a heavyweight security policy, while running a more trusted application under a VM with a subset of the security policy. Beyond security [10], process VMs exist for tasks such as dynamic optimization [3] and instrumentation [16].

Although the work presented in this paper focuses primarily on process VMs, it is not unreasonable to believe that similar design choices could be made for system VMs. Intel [9] and AMD [1] have both recently introduced support at the instruction level for system VMs.

Pin. In this paper, we perform our VM characterization using Pin [16], a dynamic instrumentation system developed by Intel. While the full details of execution are beyond the scope of this paper, we offer a short overview. Although other VMs vary in some design details, we believe Pin to be a good indicator of other VMs’ performance.

Once Pin has gained control of an application, it uses a just-in-time compiler (JIT) to recompile code into traces, or superblocks, speculatively generated by following code until reaching a predefined stopping condition. These traces are stored in a code cache that allows further execution of the same code without requiring recompilation. If the target of a branch ending a trace is in the code cache, they are linked to allow execution without Pin’s intervention; otherwise control must return to Pin to compile the next trace. This change of control, called a context switch, involves overhead from saving and restoring the application state in addition to that of performing the compilation.

On many applications, the overhead of context switches and compilation can be amortized, at least in part, over the execution of the program if they execute primarily from the code cache. However, short programs or programs with little code reuse are unable to hide the overhead of a VM.

Previous literature has shown average slowdowns on SPECINT 2000 benchmarks ranging from 1.25X in DynamoRIO [6] to 43X on some tools based on Valgrind [18]. Reducing this overhead is a motivating factor the work presented here.

3. VM CHARACTERIZATION

To characterize the performance of a VM, we must execute an application under its control. However, we must be able to separate the execution of the VM from that of the guest application. To this end, we used an eight instruction assembly program so that the guest application’s execution is negligible. We have observed a correlation between the native performance of applications and the performance of those same applications under control of a VM after startup costs; thus, while our experiments do not repeatedly exercise code compilation and context switches, we feel they are still a good representation of normal performance.

To vary architectural parameters for our characterization, we used an x86 version of SimpleScalar [2], augmented with functionality required to simulate Pin. Table 1 shows our baseline architectural configuration.

Functional unit usage. We first considered the use of floating point hardware by a VM. By executing several

Parameter	Value
Processor width	8
Fetch queue size	16
Branch predictor	Combined predictor with 16K-entry meta-table, 2-lev predictor with 16K entry L1, 16K entry L2, 14-bit history XORed with address
BTB size	512 sets, 4-way
RAS size	8
RUU size	128
L1 caches	64K, 4-way, 32B blocks
Unified L2 cache	512K, 4-way, 64B blocks
Functional units	6 int ALU, 2 int mult, 4 FP ALU, 2 FP mult

Table 1: Baseline simulation parameters

benchmarks from SPECINT 2000 both natively and under Pin and counting the uses of each functional unit, we discovered Pin only executes one floating point add. This suggests that a VM core could be built without floating point hardware. For integer units, no change in IPC was observed by decreasing the number of integer multiply units from two to one; this suggests multiply instructions are sufficiently separated to avoid structural hazards. As shown in Figure 1, decreasing the number of adders from 6 to 4 led to only a slight decrease in IPC, while the drop from 4 to 2 was more substantial; a VM core could probably use only 4 adders if necessary to meet constraints.

Branch behavior. We considered the effects of decreasing the complexity of branch predictors on performance of Pin as compared to several SPECINT 2000 benchmarks. In addition to the baseline configuration, we considered one with 8K-entry tables, and one using only the bimodal predictor with a 16K-entry table. Compared to the baseline, the VM shows virtually no decrease in IPC with the half sized predictor, and only a few percent decrease with the bimodal-only predictor; some of the benchmarks were much more sensitive to the change. This suggests that a VM core, unlike a general purpose core, would suffer little performance penalty with a less complex branch predictor.

Cache behavior. For the L1 caches, we compared the change in cache miss rate as we decreased the complexity from 64K 4-way associative caches to 32K 4-way, 32K 2-way, 16K 2-way, and 16K direct-mapped caches. Figures 2 and 3 show this data. The instruction cache miss rate increased by only half a percent, and the data cache miss rate increased by less than a percent; the final miss rates were less than 1% and 7% respectively. We then changed the L2 cache from 512K 4-way to 256K 4-way and 128K 4-way. The baseline L2 configuration has nearly a 50% miss rate, and the decrease from 256K to 128K causes roughly a 15% increase in miss rate. This suggests that the VM is relatively insensitive to L1 cache size but very sensitive to the L2 cache size.

4. CUSTOMIZED VM CORE DESIGN

Beyond the microarchitectural design decisions above, designing a VM core requires additional considerations. Here we present ideas for the floor plan as well as instruction set extensions and additional structures for communication.

Floor plan. We have considered two general classes of configuration for a heterogeneous CMP including a VM core. One option includes a single VM core to be shared by all other general-purpose and specialized cores. This layout

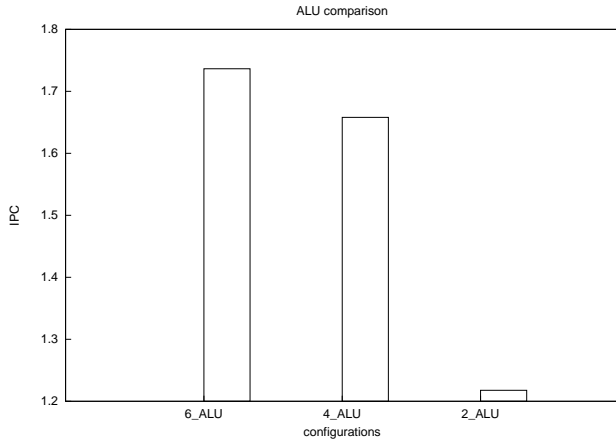


Figure 1: Performance (IPC) with respect to availability of integer ALUs.

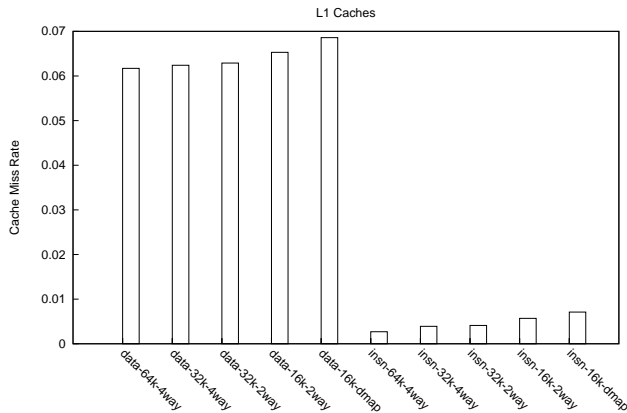


Figure 2: L1 cache miss rates with respect to varying sizes.

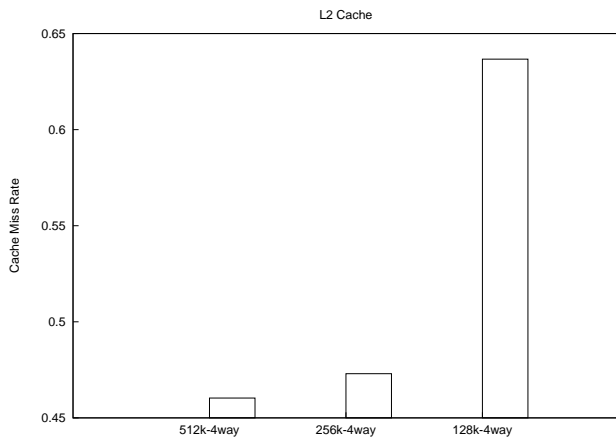


Figure 3: Cache miss rate with respect to unified L2 cache size.

lends itself easily to system VMs, since there is generally only one instance of such a VM. Alternately, each general purpose core may have a VM-specific core attached to it. Such a design may be more appropriate for process VMs, allowing each application to have its own VM and execution core. This also presents the opportunity to use conjoined core designs [12] to allow the VM cores access to floating point hardware as necessary.

Other considerations exist with respect to having one or many VM cores. Running multiple process VMs on a single core may simplify communication if they are working collaboratively. On the other hand, multiple VMs on a single core may lead to resource contention; contention in the L2 cache may be particularly detrimental to performance. A more complete study of these options and the trade-offs involved is an avenue of future work.

Additional hardware structures. In addition to communication channels between collaborating VMs, the VM must be able to synchronize and communicate with the core running its guest application. For instance, when a VM begins compiling a trace, it must receive state information from the application; conversely, when compilation is done, the VM must be able to restart the application at the appropriate location. This suggests adding registers for communication, along with modifying the VM to dynamically inject instructions into the guest application to set these registers as appropriate. Recall that the recompiled code runs from a code cache; additional hardware to map addresses from the code cache to original addresses may further decrease overhead involved in handling branches. Further study of additional structures is an avenue of future work.

5. RELATED WORK

Previous work using hardware to benefit VMs include the ADORE [7] and Trident [23] systems. ADORE used hardware performance counters already available on Itanium processors. Trident added additional structures to support event-driven optimization; however, neither system considered core design for a VM. Transmeta’s Code Morphing Software [8] was loaded into ROM on the chip, but no details are publicly available regarding whether the core was specialized for the VM. Intel VT [9] and AMD SVM [1] provide support “classical virtualization” *a la* Xen [5], but again are just extensions to existing hardware.

Several projects have designed hardware specifically for executing Java, such as JOP [20] and picoJava [17]. These projects simply provided hardware that executed the Java bytecode ISA, rather than specializing hardware for the JVM. We believe the design presented in our work is applicable to running a JVM as well as other VMs and thus is more general than those solutions.

6. CONCLUSIONS AND FUTURE WORK

The desire to keep cores of a CMP busy and to reduce the overhead of VMs can be combined by optimizing a core for execution of a VM. The work presented in this paper characterizes Pin, one such VM, in terms of functional unit usage, branch behavior, and cache behavior. We have shown that a VM core is unlikely to need floating point hardware and should need only one integer multiply unit. The VM is relatively insensitive to L1 cache sizes; in contrast, it is highly sensitive to the L2 cache size. We have also discussed

related issues including floor plan options and additional structures necessary for synchronizing a VM and application on separate cores.

We intend to follow up on this study by performing similar experiments on other VMs, which we expect to confirm our generalizations from Pin. Additionally, we will further explore the hardware layout options and extensions we have proposed. Finally, we would like to look at new opportunities for performance enhancement created by using a specialized VM core. We believe optimizations taking advantage of both a VM and extra hardware of a CMP will provide unique opportunities to continue to improve performance.

7. REFERENCES

- [1] AMD. AMD64 Virtualization Codenamed “Pacifica” Technology: Secure Virtual Machine Architecture Reference Manual, May 2005.
- [2] T. Austin, E. Larson, and D. Ernst. SimpleScalar: An infrastructure for computer system modeling. *IEEE Computer*, pages 59–67, February 2002.
- [3] V. Bala, E. Duesterwald, and S. Banerjia. Dynamo: A transparent dynamic optimization system. In *PLDI '00: ACM SIGPLAN 2000 conference on Programming language design and implementation*, pages 1–12, Vancouver, British Columbia, Canada, 2000.
- [4] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai. The impact of performance asymmetry in emerging multicore architectures. In *32nd Annual Symposium on Computer Architecture*, pages 506–517, Madison, Wisconsin, June 2005.
- [5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP '03: The 19th ACM symposium on Operating systems principles*, pages 164–177, Bolton Landing, NY, USA, 2003.
- [6] D. Bruening, T. Garnett, and S. Amarasinghe. An infrastructure for adaptive dynamic optimization. In *First Symposium on Code Generation and Optimization*, pages 265–275, San Francisco, California, March 2003.
- [7] H. Chen, W.-C. Hsu, J. Lu, P.-C. Yew, and D.-Y. Chen. Dynamic trace selection using performance monitoring hardware sampling. In *CGO '03: Symposium on Code generation and optimization*, pages 79–90, San Francisco, California, 2003.
- [8] J. C. Dehnert, B. K. Grant, J. P. Banning, R. Johnson, T. Kistler, A. Klaiber, and J. Mattson. The transmeta code morphing software: Using speculation, recovery, and adaptive retranslation to address real-life challenges. In *CGO '03: Symposium on Code generation and optimization*, pages 15–24, San Francisco, California, March 2003.
- [9] Intel Corporation. Intel®Virtualization Technology Specification for the IA-32 Intel®Architecture, April 2005.
- [10] V. Kiriansky, D. Bruening, and S. Amarasinghe. Secure execution via program shepherding. In *11th USENIX Security Symposium*, pages 191–206, San Francisco, California, August 2002.
- [11] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-isa heterogeneous multi-core architectures: The potential for processor power reduction. In *MICRO 36: 36th annual IEEE/ACM Symposium on Microarchitecture*, pages 81–93, San Diego, California, 2003.
- [12] R. Kumar, N. P. Jouppi, and D. M. Tullsen. Conjoined-core chip multiprocessing. In *37th annual IEEE/ACM Symposium on Microarchitecture*, pages 195–206, Portland, Oregon, 2004.
- [13] R. Kumar, D. M. Tullsen, and N. P. Jouppi. Core architecture optimization for heterogeneous chip multiprocessors. In *PACT '06: 15th conference on Parallel architectures and compilation techniques*, pages 23–32, Seattle, Washington, USA, 2006.
- [14] R. Kumar, D. M. Tullsen, N. P. Jouppi, and P. Ranganathan. Heterogeneous chip multiprocessors. *Computer*, 38(11):32–38, 2005.
- [15] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas. Single-isa heterogeneous multi-core architectures for multithreaded workload performance. In *ISCA '04: 31st symposium on computer architecture*, page 64, München, Germany, 2004.
- [16] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In *PLDI '05: 2005 ACM SIGPLAN conference on Programming language design and implementation*, pages 190–200, Chicago, IL, USA, 2005.
- [17] H. McGhan and M. O'Connor. Picojava: A direct execution engine for java bytecode. *Computer*, 31(10):22–30, 1998.
- [18] N. Nethercote and J. Seward. Valgrind: a program supervision framework. In *Third Workshop on Runtime Verification (RV'03)*, Boulder, Colorado, July 2003.
- [19] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang. The case for a single-chip multiprocessor. In *ASPLOS-VII: 7th conference on Architectural support for programming languages and operating systems*, pages 2–11, Cambridge, Massachusetts, United States, 1996.
- [20] M. Schoeberl. JOP: A Java optimized processor. In *Workshop on Java Technologies for Real-Time and Embedded Systems (JTRES 2003)*, pages 346–359, Catania, Italy, November 2003.
- [21] J. Sugerma, G. Venkitachalam, and B.-H. Lim. Virtualizing i/o devices on vmware workstation’s hosted virtual machine monitor. In *2002 USENIX Annual Technical Conference*, Boston, Massachusetts, June 2001.
- [22] D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simultaneous multithreading: maximizing on-chip parallelism. In *ISCA '95: 22nd annual symposium on Computer architecture*, pages 392–403, S. Margherita Ligure, Italy, 1995.
- [23] W. Zhang, B. Calder, and D. M. Tullsen. An event-driven multithreaded dynamic optimization framework. In *PACT '05: 14th Conference on Parallel Architectures and Compilation Techniques*, pages 87–98, St. Louis, Missouri, 2005.