SoftHV : A HW/SW Co-designed Processor with Horizontal and Vertical Fusion

Abhishek Deb
Universitat Politècnica de Catalunya
C6-221, C. Jordi Girona, 1-3
Barcelona, Spain
abhishek@ac.upc.edu

Josep Maria Codina
Intel Research Labs Barcelona
C. Jordi Girona, 1-3
Barcelona, Spain
josep.m.codina@intel.com

Antonio González
Intel Research Labs Barcelona
C. Jordi Girona, 1-3
Barcelona, Spain
antonio.gonzalez@intel.com

ABSTRACT

In this paper we propose SoftHV, a high-performance HW/SW co-designed in-order processor that performs horizontal and vertical fusion of instructions.

SoftHV consists of a co-designed virtual machine (Cd-VM) which reorders, removes and fuses instructions from frequently executed regions of code. On the hardware front, SoftHV implements HW features for efficient execution of Cd-VM and efficient execution of the fused instructions. In particular, (1) Interlock Collapsing ALU (ICALU) are included to execute pairs of dependent simple arithmetic operations in a single cycle, and (2) Vector Load units (VLDU) are added to execute parallel loads.

The key novelty of SoftHV resides on the efficient usage of HW using a Cd-VM in order to provide high-performance by drastically cutting down processor complexity. Co-designed processor provides efficient mechanisms to exploit ILP and reduce the latency of certain code sequences.

Results presented in this paper show that SoftHV produces average performance improvements of 85% in SPECFP and 52% in SPECINT, and up-to 2.35x, over a conventional four-way in-order processor. For a two-way in-order processor configuration SoftHV obtains improvements in performance of 72% and 47% for SPECFP and SPECINT, respectively. Overall, we show that such a co-designed processor based on an in-order core provides a compelling alternative to out-of-order processors for the low-end domain where high-performance at a low-complexity is a key feature.

Categories and Subject Descriptors

C.1 [**Computer Systems Organization**]: Processor Architectures; D.3.4 [**Software**]: Programming Languages—*Processors*

General Terms

Design, Performance

Keywords

Co-designed Virtual Machine, Micro-op Fusion

1. INTRODUCTION

Out-Of-Order processors had successfully been able to exploit the instruction level parallelism (ILP). The performance benefits of out-of-order processor is due to numerous microarchitectural advancements. However, some of the fundamental ones are out-of-order execution and higher execution bandwidth in order to execute instructions in parallel. Various hardware structures such as issue queues, reorder buffers, load store queues are required for successful out-of-order execution.

Both issue queues and load store queues are CAM based structures. The multiple tag comparison required every cycle are the sources of complexity and power dissipation [15]. Scaling such structures in order to support a larger instruction window and wider issue width adds further to both complexity and power consumption. Recently, the shift in microprocessor industry has been towards simpler low power and lower-complexity processors such as the introduction of Intel Atom. However, even for a low-end domain the need for high-performance is always growing and new mechanisms must be found in order to improve performance at low-power and low-complexity.

One possible alternative to deliver power-efficient designs is the use of Co-designed Virtual Machines (Cd-VM). In such a scheme, a processor is a co-design effort between hardware and software. The software layer wraps the hardware and provides a common interface to the outside world (operating system and applications). This software layer performs dynamic translation and optimization over the source code to adapt it to better exploit the capabilities of the underlying microarchitecture.

In this paper we consider a HW/SW co-designed microprocessor based on an in-order core and using a Cd-VM. Using an in-order core drastically cuts complexity and power consumption. The Cd-VM helps in optimizing code and reordering instructions by exploiting the ILP and provides performance benefits. HW support is added in order to aggressively reorder memory instructions.

Moreover, the use of a Cd-VM further facilitates the introduction of new microarchitectural features transparent to application software and operating system. Such features are only visible to the Cd-VM, which is responsible for generating appropriate code to reap benefits of these HW features. One example of this is the use of specialized or programmable functional units that allows execution of certain code sequences faster than in conventional FUs.

The use of specialized or programmable FUs results in a power-efficient way to invest the transistor budget. However, the challenge when dealing with this specialized hardware is twofold: first, the hardware needs to be designed in a way such that it fits the most common properties of the different workloads, and second code needs to generated in order to make efficient use of these hardwares.

In this paper we propose SoftHV, a HW/SW co-designed in-order processor where the Cd-VM performs vertical and horizontal fusion of instructions leveraging two types of specialized FUs: (1) Interlock Collapsing like ALUs (ICALU) [17] that executes a pair of dependent simple arithmetic/logic micro-ops that have been previously fused into a macro-op (MOP), that is vertical fusion; and (2) Vector Load Units (VLDU) [7] executes a pair of fused parallel loads known as a vector load (VLD), that is horizontal fusion.

ICALU provides performance improvement by collapsing a dependent pair of simple ALU micro-op in a single cycle. Both ICALU and VLDU also provide additional benefit of higher effective processor bandwidth by decreasing the dynamic instruction count. Moreover, the pressure on microarchitectural resources are decreased.

The Cd-VM in SoftHV uses hardware profiling to form superblocks. It optimizes the superblock by performing some code optimizations such as copy propagation, limited dead code elimination, load-store telescoping [8]. It then selects pair of dependent simple arithmetic operations micro-ops and fuses them to generate macro-ops (MOP). Also pair of parallel load operations are fused into a vector load instruction (VLD).

The key novelty of SoftHV is the right combination of the two types of accelerators with the right set of code generation techniques allows SoftHV to provide high-performance improvements over a conventional in-order processor in two different kinds of applications suites like: (1) SPECINT, where pairs of dependent simple arithmetic/logic operations are very common; and (2) SPECFP, where the limited amount of architectural registers results in more spill code and therefore the bandwidth of memory access is a critical constraint.

Performance results presented in this paper shows that SoftHV provides average improvements in performance of 85% for SPECFP and 52% for SPECINT over a conventional four-way in-order processor. Also, SoftHV provides average improvements in performance of 72% and 47% for SPECFP and SPECINT over a two-way in-order processor, respectively.

Moreover, we show that such a co-designed in-order processor core outperforms small instruction window out-of-order processors. Hence, given the performance, complexity and power trade-offs we argue that the SoftHV core provides a compelling alternative to out-of-order processors specially for the low-end and consumer electronics domain.

We have made several contributions in this paper and he key contributions of this paper are as follows:

- We propose SoftHV, a high-performance and a low-complexity in-order processor that leverages vertical and horizontal fusion efficiently, by adaptively generating code for different types of workloads.
- We design and implement a Cd-VM that performs a large set of code generation techniques for instruction scheduling, instructions fusion, load-store telescoping, dead code elimination, copy propagation.
- Finally, we show that SoftHV outperforms a small instruction window out-of-order processor. Hence, we argue that such a co-designed processor based on an in-order core provide a more complexity-effective and power-efficient alternative to out-of-order processors.

The rest of the paper is organized as follows. In Section2 we discuss some further insights on the motivation behind using horizontal and vertical fusion. In Section 3, we provide the overview of our proposal, by briefly discussing the co-designed virtual machine architecture of SoftHV.

The superblock formation, code optimizations, micro-op fusion and scheduling is discussed in Section 4. In Section 5 the microarchitecture is detailed with special attention paid to ICALU and VLDU. Pipeline stages that are impacted in order to support such a co-designed processor is discussed. A detailed evaluation and breakdown of performance benefits is provided in 6. Finally, related work is reviewed in Section 7 and we conclude in Section 8.

2. MOTIVATION FOR HORIZONTAL AND VERTICAL FUSION

SoftHV performs horizontal and vertical fusion of instructions, that is parallel instructions and dependent instructions are fused and then executed in specialized FUs. The combination of these two types of fusion is critical for a general domain processor to achieve competitive performance results for different types of workloads. For instance, SPECINT and SPECFP applications.

In case of SPECINT, pairs of dependent simple arithmetic/logic operations are very common, and therefore a vertical fusion of dependent instructions is a key feature for a processor in order to reduce the latency of the more critical instructions.

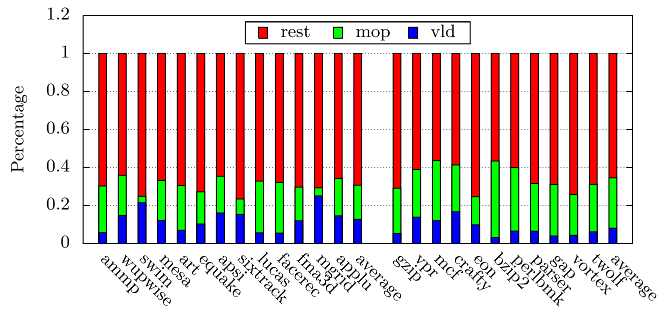


Figure 1: Micro-op Distribution

On the other hand, SPECFP presents a higher degree of parallelism. Such a degree of parallelism combined with limited amount of architectural registers results in more spill code. In turn this results in an increase of memory accesses. However, it is also frequent that spilled-code consists of many pairs of parallel loads that depend on a single source operand register and differ only in the immediate offset. Therefore a proper hardware to execute vector loads reduces the amount of accesses and effectively increases the throughput of the machine.

SoftHV combines the two ideas implemented by means of the Cd-VM, as we will see in the forthcoming sections. Figure 1 shows the code coverage provided by the instructions that are paired together (mop and vld, in the bottom of stacked histogram). Together the fused instructions provides high coverage both for SPECFP and SPECINT. However, the coverage for the type of fused instructions is very different for SPECINT and SPECFP. SPECINT consists of 28% of micro-ops fused as a pair of dependent micro-ops (mop) and 7% of parallel loads (vld). SPECFP, however, has 12% of loads fused as a pair and 20% (mop) for dependent micro-ops.

3. SOFTHV ARCHITECTURE OVERVIEW

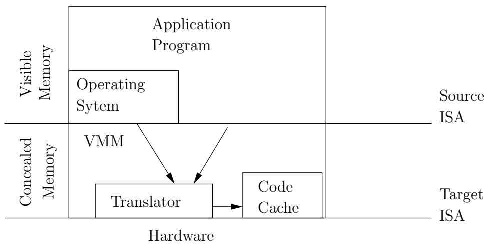


Figure 2: The Co-designed Architecture Overview

Modern out-of-order processors crack CISC like instruction into RISC like micro-ops using complex decoders. It then performs out-of-order execution by dynamically scheduling the micro-ops. A Co-designed Processor, however, takes a different approach by binary translating CISC-to-RISC using virtual machine technology. This virtual machine is referred in literature as a Co-designed Virtual Machine [20]. Figure 2 shows how a full-system source ISA to target ISA translation is provided.

Cd-VM not only translates but also optimizes the translated code by performing various code optimizations. Dynamic re-compilation enables Cd-VM to apply aggressive optimizations and rely on microarchitecture to detect and fallback on miss-speculations.

A co-designed processor also enables to introduce HW accelerators without changing the source ISA. Microarchitects can introduce new hardware accelerators and remove the redundant ones. This requires modification only to the target ISA, or micro-op ISA in other words. Furthermore, Cd-VM dynamically re-compiles the applications to generate instructions that execute on these hardware accelerators.

3.1 The Co-designed Virtual Machine Architecture

Cd-VM uses hardware Profiling (block and edge)[4] to identify frequently executed basic blocks. A superblock is formed when a hot basic block is detected. Most frequently taken edges are followed and basic blocks are appended. The superblocks are binary translated to the target ISA, followed by series of code optimizations. Finally, the superblock is stored in a part of memory concealed from both the application and the operating system. This concealed memory region comprises of Cd-VM code and data, and code cache, the location where superblocks are stored as shown in Figure 2.

Jump TLB (JTLB)[20] holds mapping from source program counter (PC) to target PC for each superblock and is accessed in parallel to the I-Cache. JTLB is managed completely and transparently by the Cd-VM. A hit in this structure indicates availability of an optimized region of code, superblock in our case, in the code cache. Code residing in the code cache is in target ISA.

3.2 Microarchitecture support for Co-designed Virtual Machine

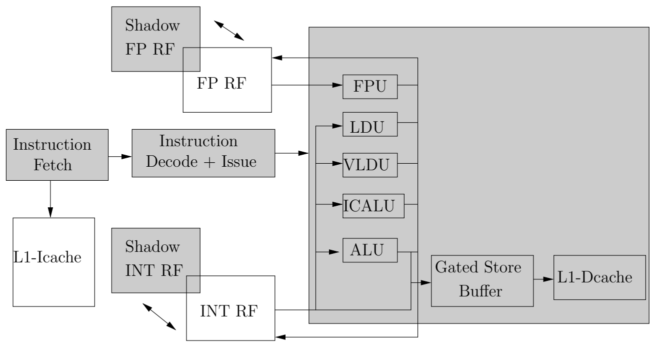


Figure 3: The Co-designed Processor Overview

Figure 3 provides an overview of the proposed co-designed processor based on decoupled in-order microarchitecture. Fetch buffers decouple fetch from decode while issue queues decouple decode from the execution back-end of the processor.

A two-level decoder as proposed in [10] is used. The ability to execute x86 code natively avoids the need for an interpreter and hence, the slowdown associated with it. X86 instructions are decoded to RISC-like micro-ops.

Cd-VM allows dynamic and aggressive code optimizations compared to a traditional compiler. However, in order to support these the underlying microarchitecture should provide mechanisms to detect miss-speculations and take the necessary corrective action.

The x86 register state is shadowed by maintaining two copies a working copy and a shadow copy as shown in Figure 3. A special commit micro-op copies all the working registers into their corresponding shadow register, when the superblock execution is successful. On the other hand, in case of an exception, a rollback operation copies the shadow copy back to the working copy. Commit and rollback is applied for stores as well; and store data is held in gated store buffer [21]. The data in gated store buffer is committed to memory when superblock successfully finishes executing all the micro-ops. This recovery mechanism is similar to the one implemented in Transmeta Crusoe [6].

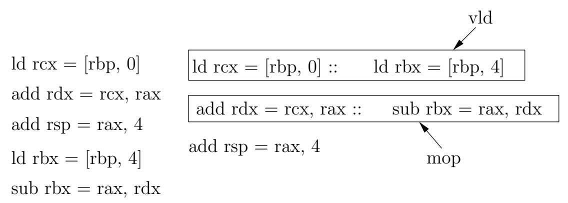
Various memory aliasing optimizations, such as Load hoisting, are speculative in nature. An alias hardware is added in the microarchitecture in order to detect a miss-speculation. Moreover, Cd-VM converts both the load and store to special micro-op. When this special store micro-op executes it will use a mask to check with previously executed loads for any miss-speculation [13]. In case of a miss-speculation the above-mentioned recovery mechanism is used to rollback and execute the unoptimized version of the superblock.

Superblocks are of various types and it depends upon whether it has multiple or single exit point(s). A single-entry-single-exit superblock, as the name suggests, has a single entry point i.e. the first basic block and a single exit point which is the last basic block. This implies that all the branches in the superblock should resolve to its next basic block in the superblock. In case, it doesn't a miss-speculation is detected and the superblock rolls back.

However, this provides an opportunity for aggressive code optimization by transgressing the basic block boundaries. For instance, the above mentioned memory optimization could be applied across the basic block boundaries.

3.3 HW Accelerators : ICALU and VLDU

Based on the characteristics of SPECINT and SPECFP applications, we introduce two functional units namely ICALU and VLDU. ICALU collapses a pair of dependent ALU micro-op and execute in a single cycle. Only a specific pair of dependent ALU micro-ops are allowed, while shifts among others are not considered. Such a fused pair of micro-ops is called as a macro-op (MOP).



```
ld rcx = [rbp, 0]
add rdx = rcx, rax
add rsp = rax, 4
ld rbx = [rbp, 4]
sub rbx = rax, rdx
```

(a) Unfused Code

```
ld rcx = [rbp, 0] ::      ld rbx = [rbp, 4]
add rdx = rcx, rax ::     sub rbx = rax, rdx
add rsp = rax, 4
```

(b) Fused Code

Figure 4: Code before and after fusion

VLDU on the other hand fuses a pair of independent load micro-op. Only those loads that have same base register and differ in the immediate offset are considered. Such a pair of fused loads are called a vector load (VLD).

Cd-VM not only enables in transparently introducing these functional units, but also providing a larger scope and more opportunity for using these functional units. Superblocks are analyzed and pairs of dependent ALU and independent load micro-ops are extracted. These pair of micro-ops are identified by setting the first bit of the micro-op to indicate whether its fused.

Figure 4 shows a code snippet before and after fusion. Note, how the fused micro-ops are placed together in the generated code. In this example, a pair of fused loads is followed by a pair of fused addition and subtraction as shown in Figure 4b. Fused micro-ops are identified by the first bit of each micro-op.

The two fused loads share same base register **rbp**, but differ in immediate offset by four bytes. Such a pair of loads is very common as they access different elements of a data structure, or different entries from an array. Hence, clubbing them together as a single entity increases processors effective bandwidth.

The **sub** micro-op in Figure 4 is dependent on **rdx** which is produced by the **add** micro-op. This fused micro-op pair not only provides benefit by executing the dependent pair in a single cycle but also provides better processor resource utilization. For instance, the sub micro-op does neither have to access Physical Register File nor the bypass network for **rdx**.

4. SOFTHV BINARY OPTIMIZATIONS

Co-designed virtual machine (Cd-VM) plays an important role in dynamically optimizing the code for an efficient use of ICALU and VLDUs. Hardware assisted block and edge profiling [4] is performed while running the application. Once a particular basic block becomes hot, the Cd-VM is invoked. The Cd-VM then creates a superblock, optimizes the code in the superblock, and finally stores the generated code in the code cache. The optimization process implemented by the proposed Cd-VM is shown in Figure 5.

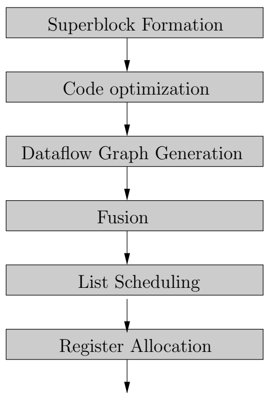


Figure 5: Code Generation Flow Chart

4.1 Superblock Formation

The superblocks formed in our approach are committed atomically and branches are converted into asserts. Therefore, it is important to consider the completion rate as part of the region formation heuristic.

The proposed scheme to form superblocks starts with the detected hot basic block and keeps including basic blocks that are targets of the edges with higher taken rate. The superblock formation continues until certain stopping conditions are met which are described as follows: (a) when the edge leads to a basic block that is already included in the current superblock; (b) when the edge leads to a basic block which is head of another superblock; (c) if the size of the superblock exceeds a threshold; and, (d) if the completion rate, according to the profile information, decreases below a threshold. In our experiments, we have judiciously set the size threshold to 500 instructions and the completion rate to 90%. The above-mentioned stopping conditions are conservative and result in smaller superblocks as shown in Table 1.

4.2 Code Analysis and Data Flow Graph Generation

After forming the superblocks, decoding the x86 instructions and cracking them into micro-ops, the micro-ops are converted into Static Single Assignment form. In this transformation virtual registers are used in order to remove WAW and WAR dependencies.

While converting the code into SSA, live-in values and live-out values are kept into the appropriate architectural

SPECFP	ammp	wup	swim	mesa	art	equake	apsi	sixtr	lucas	facer	fma3d	mgrid	applu	av.
	26.01	18.78	61.46	21.72	24.56	14.51	30.81	62.95	10.46	13.82	29.97	73.11	31.90	32.31
SPECINT	gzip	vpr	mcf	crafty	eon	bzip2	perlbmk	parser	gap	vortex	twolf	av.		
	11.53	12.90	9.78	16.64	17.08	17.91	11.86	9.44	11.28	11.09	11.90	12.86		

Table 1: Superblock Size in micro-ops

registers. Moreover, in the same traversal over the code, branches are converted into asserts.

Also, in the same traversal over the code, basic code analysis is performed in order to identify aliasing among pairs of memory instructions. In particular, this analysis is done by considering that two operations alias if the same unmodified registers and constants in their address computation match.

Data flow graph (DFG) is built considering both the memory and the register dependencies. Memory dependencies are identified by means of the code analysis discussed above. Then our scheme speculates on the rest of memory dependencies and it relies on the hardware support [6] for detecting miss-speculations. In case of a miss-speculation the whole superblock rolls back and the unoptimized version is fetched and executed.

4.3 Code Optimization

Once the DFG is built, we perform a series of code optimizations such as dead-code elimination, copy propagation, common sub-expression elimination, load-store telescoping [8] among others. Moreover, since edges are created between the memory instructions that aliases, this provides opportunities for aggressive code reordering, such as load-hoisting.

Load hoisting enables loads to scheduled above stores. Load-Store telescoping on the other hand enables bypassing of loads by forwarding the data from the producer of the store directly to the consumers of the load, for the aliasing store-load pair.

4.4 Micro-op Fusion

A single pass forward scan fusion algorithm is implemented that fuses both a pair of dependent simple ALU micro-ops and also a pair of parallel loads. The algorithm is greedy and tries to find the first suitable micro-op to fuse with the current micro-op as described in Algorithm 1 below.

Algorithm 1 Fusion Algorithm

```
for all micro-op do
  if micro-op allready fused then
    continue
  end if
  if micro-op is a load then
    scan forward for fusible parallel load
  else if micro-op is a simple arithmetic/logic op then
    scan forward for a dependent simple micro-op
  end if
  if pair found then
    mark as a new fused pair
  end if
end for
```

The pair of dependent ALUs are executed in the ICALU. ICALU considers only a specific pair of simple micro-ops such as arithmetic operations, logical operations (except shifts), register transfer operations, address generation and branch outcome determinations. A pair of independent loads that have a same base register but differ in the immediate offset. Loads with multiple source register operands are not considered for fusion.

Such a fused pair of micro-ops is called a macro-op. After considering a macro-op, the algorithm proceeds by considering micro-ops which were not included in the previous macro-ops. The algorithm proceeds in the similar fashion until all the micro-ops of the superblock have been considered.

4.5 Scheduling

Once the macro-ops are formed a scheduling step is performed. Since, the underlying processor is in-order; scheduling code is an important step. Moreover, this scheduling is performed over a superblock which provides a larger scope than a basic block. We implement a list scheduling technique that gives priority to the instructions in the critical path of the dependence graph [5]. The improvement in performance due to list scheduling is evaluated and shown in Section 6.4.

Priority is computed using dynamic programming approach using the following formula. The DFG $G = (N, E, E')$ has a node $n \epsilon N$ for each micro-op. Edges $e = (n_i, n_j) \epsilon E$ represents dependencies between micro-ops. Edges in E' represent false-dependence and is referred as an anti-edge.

$$priority(n) = \begin{cases} latency(n) & \text{if } n \text{ is a leaf} \\ max(latency(n) + \\ max_{(m,n)\epsilon E}(priority(m)), \\ max_{(m,n)\epsilon E'}(priority(m))) & \text{otherwise} \end{cases}$$

Under this list scheduling heuristic priorities are assigned to each node using latency-weighted depth. Note that, false-dependencies are also considered while computing the priorities (anti-edge). Once the priorities are computed the list-scheduling algorithm is applied. Any instruction that is ready at cycle X is a candidate to be scheduled in cycle X. An instruction with a higher priority is given preference over other instructions which are ready for the given cycle. Any tie in priority of two instructions is broken arbitrarily as described in Algorithm 2.

4.6 Register allocation

After the process of combining micro-ops to form a macro-op, a register allocation step is conducted to map virtual registers to architectural registers. The register allocation is based upon linear scan register allocator [18].

5. SOFTHV MICROARCHITECTURE

Figure 3 shows the block diagram of the proposed in-order microarchitecture. ICALU and VLDU are added in the back-end. A shadow and working copy of Register File exists in order to support precise x86 exception and tackle ordering violation. JTLB is added in the front-end in order to provide quick source PC to target PC translation for the superblocks.

Algorithm 2 List Scheduling Algorithm

```
cycle = 0
ready-list = root nodes of DFG
in-flight-list = empty list
while ready-list or in-flight-list not empty, and an issue slot is available do
  for op = all nodes in ready-list in descending priority order do
    if a FU exists for op to start at cycle then
      remove op from ready-list and add to in-flight-list
      add op to schedule at time cycle
      if op has an outgoing anti-edge then
        Add all target's of op's anti-edges that are ready to ready-list
      end if
    end if
  end for
  cycle = cycle + 1
  for op = all nodes in in-flight-list do
    if op finishes at time cycle then
      remove op from in-flight-list
      check nodes waiting for op in DFG and add to ready-list if all operands available
    end if
  end for
end while
```

The baseline microarchitecture is a two-way/four-way in-order processor. It is a decoupled microarchitecture with buffers provided for fetch and issue. Issue logic contains FIFO queues and counters are maintained to ensure back-to-back execution of micro-ops. Figure 6 shows the pipeline of the proposed processor.

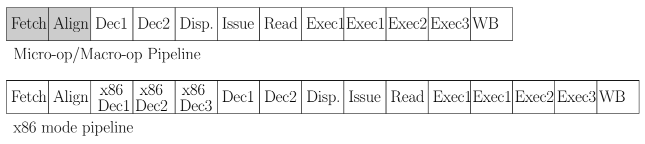


Figure 6: micro-op/macro-op pipeline

The shaded stages are when a pair of fused micro-op is treated separately. Align/Fuse stage checks the first bit of each micro-op and fuses the current micro-op with a succeeding one. All the subsequent stages treat a pair of fused micro-op as a single entity. This has an additional side-benefit of reducing pressure from microarchitecture resources such as read-ports, issue queue entries, forwarding network among others. Figure 1 shows the code coverage of a pair of fused micro-op is nearly 30%. This gives an estimate of reduction in dynamic instruction count, leading to efficient processor resource utilization.

Since, the processor uses a dual mode decoder, it can run x86 instructions natively by cracking them into micro-ops as done by a modern x86 processor [9, 3, 12]. The x86 mode pipeline of the processor is shown in the Figure 6. The primary benefit of using a dual-mode decoder is to cut down on overhead due to cold-code binary translation [11].

Modern x86 decoders, however, are complex in nature and are power-hungry. Our experiments have indicated that superblocks provide more than 90% of code-coverage. Superblocks contains optimized code in micro-op ISA, which bypass the complex x86 decoders. As a result x86-decoders are powered-off most of the times.

5.1 The pipeline Front-End

Each cycle 16-byte chunk of instruction byes is fetched from the I-cache. Instruction boundaries are determined, x86 instruction go-to first level decoder while micro-ops go directly to second-level decoder. The fuse bit of a micro-op indicates that it should be fused with the immediately following micro-op. After the micro-ops are fused they remain as a single entity for the rest of the pipeline. The decoded micro-ops or the MOPs and VLDs are then dispatched into an Issue Queue.

5.2 The pipeline Back-End

A FIFO based issue queue is used which holds the micro-ops until they are ready. The readiness of an instruction is determined using a scoreboarding mechanism that statically predicts when operands will be available. The value read from scoreboard indicates the number of cycles until a valid result will be available for forwarding. The value in each scoreboard entry counts down every cycle. Such a mechanism ensures back-to-back execution.

ICALU and VLDU co-exist with other FUs in the datapath as shown in the Figure 3. Pair of dependent micro-ops that are fused into a MOP is executed into an ICALU. The head of a MOP is sent in parallel to a normal ALU similar to [10]. Pair of fused loads (VLD), on the other hand, are executed in a VLDU.

ICALUs [17] consists of a pair of ALUs collapsed to execute a pair of dependent instructions in a single cycle. Only a subset of ALU instructions are considered that satisfy constraints such as number of input operands, simple arithmetic operations, logical operations. Shift and Add-shift are not considered.

Our experiments also showed us that most of the times the alu micro-op that is the first in the pair produces value only for the micro-op it is paired with. This implies that the first micro-op in the MOP should not be a live-out for the region, and neither should its value be consumed by any other micro-op. Introduction of ICALU not only provides speedup but also provides better microarchitecture resource utilization such as issue width, read/write ports, buffers etc.

Vector Load units enable execution of data parallel loads as a combined instruction. Only those loads that have same size, same operand, but access different banks are fused. x86 ISA consists of many loads that have same start address but different offset. Such a kind of vector loads have been proposed already in Altivec [7]. Figure 7 illustrates how a VLD generates address and reads data from D-cache for both the fused loads. A VLDU consists of a pair of Address generation unit in order to generate address for both the loads in parallel. Request is sent to the respective banks. In case of a bank conflict, processor stalls the back-end for a single cycle while the bank conflict is being resolved. Data for both the loads are then available in next cycle.

Cache-miss for scalar loads results in stalling of the back-end of the pipeline. Decoupling provided by fetch buffers and issue queues keeps the front-end unaffected. Similarly, on a cache-miss for any load fused in the vector load only the back-end pipeline is stalled. The usage of an in-order core

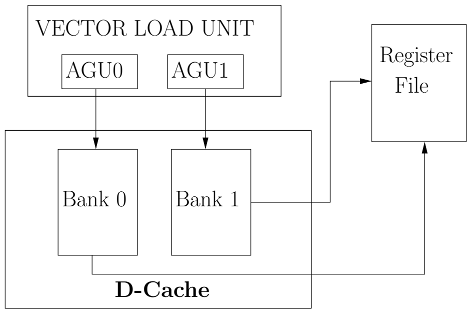


Figure 7: Illustration of a VLD execution

makes handling of vector load instruction much simpler, as memory disambiguation related complications are removed.

However, memory ordering violation is detected by the aliasing hardware. A co-designed approach for detecting such memory conflicts is used [6]. The load-and-protect instruction (ldp) performs a load operation and records the address and size of the loaded data in an architected table. The store-under-alias-mask (stam) apart from executing the usual store also contains a mask that identifies certain table entries, the ones set by preceding ldp instructions. If the address range of a ldp and stam overlaps, superblock rolls back and unoptimized code is fetched and executed.

6. PERFORMANCE EVALUATION

6.1 Experiment Methodology

Our proposed HW/SW co-designed processor is modeled using PTLSim[24]. We have implemented an in-order processor core along with a Cd-VM. We simulate a decoupled in-order processor microarchitecture. Table 2 provides detailed information of the microarchitecture of the simulated processor.

4/2-way In-order Processor Parameters	
I-Cache	16 KB, 4-way, 64 B line, 2 cycle access
Branch Predictor	Combined Predictor 64 K 16-bit history 2-level, 64 K bi-modal, 1K RAS
Fetch Width	4/2 micro-ops / x86 instructions up-to 16 bytes long
Issue Width	4/2 (2/1 LD, 2/1 FP, 2/1 INT, 2/1 ST, 1 MOP, 1 VLD)
L1 Data Cache	32 KB, 4-way, 64 B line, 2 cycles
L2 Cache	256 KB, 16-way, 64 B line, 6 cycles
L3 Cache	4 MB, 32-way, 64 B line, 14 cycles
Main Memory	154 cycles
Issue Queue	16 entry
Functional Units	2/1 LDU, 2/1 ALU, 2/1 AGU, 2/1 FPU, 1 ICALU, 1 VLDU
Register File	32-entry INT RF, 32-entry FP RF, 4 write ports each

Table 2: Baseline processor configuration

We have evaluated the proposed scheme using the SPEC2000 benchmark suite. These benchmarks have been compiled with gcc version 4.1.3 using -O3. Using the developed infrastructure, we have simulated the benchmarks for 100 million x86 instructions after the initialization and a cache warm-up period of 2 million x86 instructions.

In this evaluation we have studied the performance benefit of the proposed co-designed processor. The performance benefit in a co-designed processor is obtained both from SW and HW techniques. We discuss the benefits of code-optimization and code-reordering. The overhead of using a Cd-VM is also measured.

The benefits in performance due to vertical and horizontal fusion is measured, first in isolation. Their combined benefit is studied as well. We report results both for a 2-way and 4-way in-order processor core.

We also compare our proposed processor to variants of low-end out-of-order processor. This results are particularly important as we show that our proposed HW/SW co-designed in-order processor is a complexity-effective and power-efficient alternative to low-end out-of-order processors.

6.2 Superblock Characteristics

As mentioned in Section 4, Cd-VM forms superblocks and then optimizes them. Table 1 shows the average size of superblocks formed with our heuristic in SPEC2000. In SPECFP superblocks are more than twice as large as that compared to SPECINT. Larger superblocks provides more opportunity not only for code optimizations but also for code fusion. In the following section we show that SPECFP obtains 1.63x times the speedup obtained by SPECINT.

Figure 1 shows the distribution of micro-ops. X86 ISA results in higher load and store instructions. However the percentage of simple arithmetic instructions are higher in SPECINT. SPECFP on the other hand has nearly 20% floating point operations.

6.3 Overhead of Co-designed Virtual Machine

Code optimizations performed by Cd-VM comes with the overhead of performing the optimizations. S. Hu et al. in [11] have both categorized and quantified the overhead of Cd-VM. They pointed out that startup overhead comes due to Basic Block Translation and Super Block Translation. The majority of this overhead is, however, due to Basic Block Translation. In a processor equipped with dual-mode decoder, this overhead is eliminated, since the x86 instructions are executed natively (by breaking into micro-ops).

To measure the overhead of Super Block Translation we used estimates provided in [11, 8]. DAISY [8] report a conservative estimate of nearly 4000 source instruction to optimize a single source instruction. They however, mention that this a very conservative estimate and quote a reasonable estimate to be 1000 instructions. S. Hu [11] measured their estimate to be 1000 instructions.

Table 3 shows the overhead of Super Block Translation for the benchmarks in terms of x86 instructions. Most of the benchmarks executes billions of instructions. Hence, the startup overhead of Super Block translation is smaller than one percent. We show the number for the estimate using 1000 instructions. Its obvious from the Table above even with very a conservative estimate of 4000 instruction estimate the overhead is around 1 percent.

6.4 Code Optimizations

In Section 4 we have discussed various optimization techniques used in our proposed processor. Here we show the benefits of some of the most important techniques. We report the benefits coming from various optimizations by running four different experiments. Each experiment contains the optimizations of the previous experiment.

We choose a 4-way SoftHV configuration and report nor-

SPECFP	ammp	wup	swim	mesa	art	equake	apsi	sixtr	lucas	facer	fma3d	mgrid	applu.
	1818 K	758 K	209 K	2086 K	366 K	1144 K	2347 K	412 K	84 K	644 K	3955 K	81 K	5859 K
SPECINT	gzip	vpr	mcf	crafty	eon	bzip2	perlbmk	parser	gap	vortex	twolf		
	460 K	1292 K	281 K	4756 K	2463 K	126 K	1605 K	2797 K	2349 K	3223 K	1517 K		

Table 3: Overhead in terms of x86 instructions

malized speedup over a 4-way in-order processor. Similarly, the speedups for a 2-way SoftHV configuration is over a 2-way in-order processor.

Figure 8 shows the breakdown of benefits due to code optimization and code scheduling. A plus sign (+) preceding an optimization indicates that this experiment is run with all the optimizations of the one above it. For instance **+ dce** indicates that the experiment was run with the optimization of **cro_lst** along with its own.

Code-reordering along with load-store telescoping (**cro_lst**) provides 45% and 20% speedup in SPECFP and SPECINT respectively as shown in Figure 8. Code-reordering includes optimizations such as load-hoisting among others. A larger superblock leads to more opportunity for such optimizations; hence more the benefit. Superblocks in SPECFP on an average consist of nearly 32 micro-ops as shown in Table 1, where as, SPECINT are relatively much smaller, only 12 micro-ops long. This fact can also be verified at benchmark level. For instance swim, mgrid and sixtrack have superblocks which are nearly twice as large as the average size for SPECFP. These benchmark obtain maximum speedups of 80% for mgrid and sixtrack, and 60% for swim.

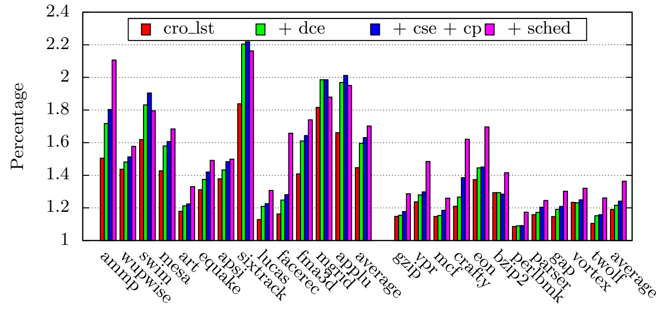


Figure 8: Code Optimizations

Traditional code optimization techniques such as dead code elimination (**dce**), along with copy propagation and common sub-expression elimination (**cse + cp**) provide a further performance improvement of 18% in SPECFP. For SPECINT, additional speedup of 5% is obtained. These configurations were run on top **cro_lst**.

List-scheduling (**sched**) is another optimization that is applied and is specially effective for SPECINT. Eon and crafty obtain 25% additional speedup due to list-scheduling. On an average, 15% additional speedup is obtained. However, since, list-scheduling is a heuristic it can provide sub-optimal solution and in some cases could even degrade. This is especially notable in some SPECFP benchmarks such as swim, sixtrack, mgrid and applu as shown in Figure 8.

Hence, code optimizations result in an overall speedups of 70% and 37% for SPECFP and SPECINT, respectively, for a 4-way SoftHV configuration over a 4-way in-order processor.

The speedups for a 2-way SoftHV configuration can be found in Table 4 against the **OPT** entry.

6.5 Performance benefit due to horizontal and vertical fusion

Our proposed processor also consists ICALU and VLDU as described in Section 5. We study the benefits in performance from each of these functional units, first in isolation. The combined benefit of both the functional units is shown as well. The experiments are run by using the optimized and scheduled superblocks and by adding ICALU and VLDU first in isolation and then together.

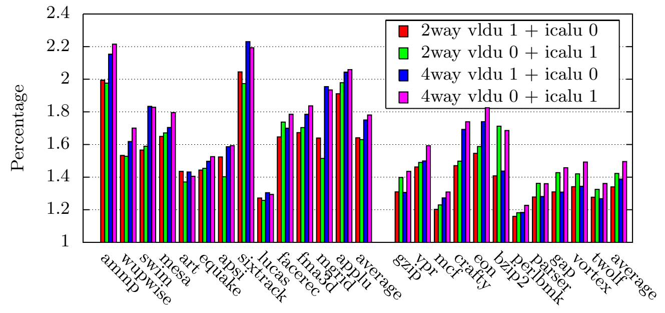


Figure 9: ICALU VLDU Isolation

Figure 1 shows the weighted distribution of the fused micro-ops. It shows that nearly 32% and 35% of micro-ops are fused as mop or vld in SPECFP and SPECINT, respectively. However, the proportion of micro-ops fused as mop or vld is different for SPECFP and SPECINT. A low ILP in SPECINT leads to many short dependent chains. As shown in Figure 1, 28% of micro-ops are fused as MOP, while SPECFP contains 20%.

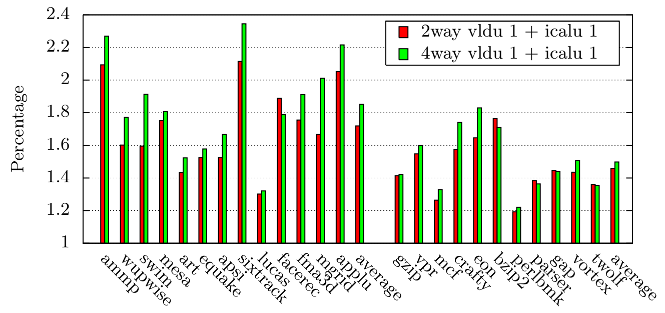


Figure 10: ICALU VLDU Combination

As a result of which, SPECINT obtains more performance benefit from ICALU 12-13% additional speedup for a 4-way

SoftHV configuration as shown in Figure 9 (**4way vldu 0 + icalu 1**). SPECFP, however, obtains 6-7% speedup from each of ICALU and VLDU on a 2-way in-order processor (**2way vldu 1 + icalu 0**).

The combined benefit of both the ICALU and VLDU over optimization is 15% (**2/4way vldu 1 + icalu 1**). The combined effect of ICALU and VLDU on speedups in some benchmarks are quite significant. This clearly shows the importance of combining horizontal and vertical fusion to reap benefits from the specialized hardware. Bzip2 in particular obtains 30% more speedup over optimization, resulting in a total speedup of 70% as shown in Figure 10. A high proportion, 35% of micro-ops are fused as MOPs that results this high speedup in bzip2.

The combined benefit of ICALU and VLDU is particularly evident in sixtrack. The speedup provided just by optimizations is 2.2x. ICALU and VLDU configuration when applied in isolation hardly provides any additional performance benefit as evident from **4way vldu 0 + icalu 1** and **4way vldu 1 + icalu 0**. Whereas, the combined effect increases the speedup to 2.35x as evident from **4way vldu 1 + icalu 1**. Hence, **4way vldu 1 + icalu 1** is our proposed SoftHV configuration.

Table 4 provides a summary of speedups obtained from optimizations and ICALU and VLDU for both 2-way and 4-way processor configuration.

	SPECINT		SPECFP	
	2-way	4-way	2-way	4-way
OPT	30%	37%	57%	70%
OPT + VLDU	33%	39%	65%	75%
OPT + ICALU	42%	49%	63%	78%
OPT + ICALU +VLDU	47%	52%	72%	85%

Table 4: Breakdown of Speedup

6.6 An alternative to Out-Of-Order processor for low-end

A modern out-of-order processor provides benefit by exploiting ILP using out-of-order execution and wider instruction issue. Our co-designed in-order processor exploits the ILP by reordering the micro-ops. It provides higher effective issue bandwidth by instruction fusion.

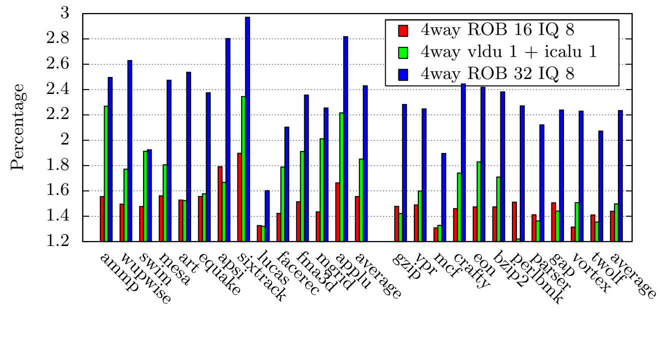


Figure 11: Comparison with low-end out-of-order

A comparison between the two design alternatives is worth a discussion. We chose variants of two-way and four-way out-of-order processor. Figure 11 shows the benefits of out-of-order processor. Note that, all the speedups reported for a n-way machine (SoftHV or out-of-order) are normalized to the corresponding n-way in-order processor.

Figure 11 suggests that the ROB size has a significant impact in the performance and is especially evident in SPECFP. Our average superblock size is 13 and 32 micro-ops for SPECINT and SPECFP respectively. For SPECINT a 32 entry ROB is larger than the superblock size. As a result, this 32-entry ROB version (4way ROB 32 IQ 8) of out-of-order processor outperforms our proposed SoftHV version **4way vldu 1 + icalu 1**.

However, compared to a out-of-order processor with 16-entry ROB (4way ROB 16 IQ 8), our proposed co-designed processor is 1.5 times faster for SPECFP. We believe that a larger superblock will provide more opportunities not only for optimization but also for fusion. For instance, Figure 1 suggests that even after fusion nearly 40% of micro-ops (loads and simple arithmetic operations) are scalar. Our analysis suggests that for loads 95% of times no other load was available. However, for MOPs 90% of time the dependent micro-op was not collapsible.

In terms of complexity it is known that an in-order processor is much more complexity-effective and power-efficient. [15] shows how an out-of-order processor adds complexity by introducing CAM based hardware structure such as issue queues and load store queues. Reorder buffer further adds to this complexity. They also discuss the impact of area and latency of Physical Register File with respect to its size and number of read and write ports.

7. RELATED WORK

7.1 Dynamic Optimization

Performing dynamic optimization in a user transparent manner is not new. Several hardware approaches have been considered [16, 19]. The main difference between our scheme and theirs is that we use a HW/SW co-design approach that offers higher flexibility of the software layer to perform different optimizations and analysis with reduced hardware complexity.

The concept of a co-designed VM is also used by Transmeta Crusoe [14, 6] processors to execute x86 code on top of a VLIW processor, and by IBM DAISY [8] and BOA [1] projects to execute PowerPC code on top of a VLIW processor. There are two main differences between these schemes and the one described in our proposal. First, we eliminate start-up overheads by eliminating the interpretation/translation step of a typical co-designed VM. Secondly, prior projects assume a VLIW target processor, while we target an x86 processor in-order design.

In addition, none of the above proposals consider any specific hardware accelerator for dealing with vertical and horizontal fusion of code at the same time. Our scheme not only features software oriented optimizations for exploiting ILP but also includes: (1) horizontal fusion of instructions to make use of the vector load unit and exploit ILP; and (2) vertical fusion of instructions to make use of the Interlock Collapsing ALU and reduce latency.

7.2 Accelerators

Many different types of fine-grained accelerators [2, 23, 22, 10] have been proposed to improve performance in a complexity-effective way. Some of them [2, 22] are designed requiring changes to the ISA and they are programmed using a static code generation scheme. This, in contrast to our approach demands an effort in forward compatibility. Applications evolve and so, accelerators may require some changes from generation to generation. Therefore, adding instructions at ISA level may pose important constraints for future processor generations.

Dynamic and transparent management of these accelerators have been studied in the past. Most of the works [2, 23] have focused on managing accelerators with hardware-based dynamic optimizers. A more limited amount of work has considered the use of co-designed VM for assigning instructions to an accelerator [17] in out-of-order design.

SoftHV provides important novelties when compared to previous approaches that provide implementations of accelerators in conventional pipeline designs. SoftHV implements a simple in-order processor with hw/sw mechanisms to perform software dynamic optimization together with two different types of specialized functional units that allow for exploiting parallelism as well as reducing latency in a complexity-effective manner.

8. CONCLUSIONS

In this paper we have presented SoftHV, a HW/SW co-designed processor that offers a complexity-effective way to achieve good performance for general purpose application. The main novelty of this scheme is the right combination of hardware and software to implement horizontal and vertical fusion of instructions to obtain significant performance improvements on different types of workloads over a conventional in-order processor.

We have shown that SoftHV is implemented by means of a Cd-VM that reorders and optimizes a superblock. Pairs of dependent simple arithmetic/logic micro-ops are fused in a vertical fashion to be later executed in ICALU. Independent loads are fused in a horizontal fashion and are executed in a VLDU.

Overall SoftHV results in an interesting co-design approach that obtains an average speedup of 85% for SPECFP and 52% for SPECINT over a conventional 4-way in-order processor. Hence, SoftHV provides an interesting design point for low-complexity high-performance processors for different types or workloads. Therefore, this can be a good alternative to small out-of-order processors for the low-end and consumer electronics domain.

9. ACKNOWLEDGMENTS

This work is partially supported by the Generalitat de Catalunya under grant 2009SGR1250, the Spanish Ministry of Education and Science under contracts TIN2007-61763 and TIN2010-18368, and Intel Corporation. Abhishek Deb is supported by a FI grant of Generalitat de Catalunya.

10. REFERENCES

[1] E. Altman et al. BOA: The Architecture of a Binary Translation Processor. Technical report, IBM, 1999.

[2] N. Clark et al. Application-Specific Processing on a General-Purpose Core via Transparent Instruction set customization. In *IEEE Intl. Symp. on Microarchitecture*, 2004.

[3] R. Colwell et al. A 0.6μm BiCMOS Processor with Dynamic Execution. In *IEEE Intl. Solid-State Circuits Conf.*, 1995.

[4] T. Conte et al. Using branch handling hardware to support profile-driven optimization. In *IEEE Intl. Symp. on Microarchitecture*, 1994.

[5] K. Cooper et al. An experimental evaluation of list scheduling. Technical report, Dept. of Computer Science, Rice University, 1998.

[6] J. Dehnert et al. The Transmeta Code Morphing Software: using speculation, recovery, and adaptive retranslation to address real-life challenges. In *IEEE Intl. Symp. on Code Generation and Optimization*, 2003.

[7] K. Diefendorff et al. Altivec extension to powerpc accelerates media processing. *IEEE Micro*, 2000.

[8] K. Ebcioglu et al. DAISY: Dynamic compilation for 100% architectural compatibility. In *IEEE Intl. Symp. on Computer Architecture*, 1997.

[9] G. Hinton et al. The Microarchitecture of the Pentium 4 Processor. *Intel Technology Journal*, 2001.

[10] S. Hu et al. An approach for implementing efficient superscalar CISC processors. In *IEEE Intl. Symp. on High-Performance Computer Architecture*, 2006.

[11] S. Hu et al. Reducing startup time in co-designed virtual machines. In *IEEE Intl. Symp. on Computer Architecture*, 2006.

[12] A. K8. Software Optimization Guide for AMD64 Processors. 2005.

[13] E. J. Kelly. Memory controller for a microprocessor for detecting a failure of speculation on the physical nature of a component being addressed, 1998.

[14] A. Klaiber. The technology behind Crusoe Processors. *Transmeta Technical Brief*, 2000.

[15] S. Palacharla et al. Complexity-effective superscalar processors. In *IEEE Intl. Symp. on Computer Architecture*, 1997.

[16] S. Patel et al. rePLay: A Hardware Framework for Dynamic Optimization. *IEEE Transactions on Computers*, 2001.

[17] J. Phillips et al. High performance 3-1 interlock collapsing alus. *IEEE Transactions on Computers*, 1994.

[18] M. Poletto et al. Linear scan register allocation. *ACM Trans. Program. Lang. Syst.*, 1999.

[19] R. Rosner et al. Power awareness through selective dynamically optimized traces. In *IEEE Intl. Symp. on Computer Architecture*, 2004.

[20] J. Smith et al. *Virtual Machines: A Versatile Platform for Systems and Processes.* Elsevier Inc., 2005.

[21] M. J. Wing et al. Gated store buffer for an advanced microprocessor, 2000.

[22] Z. Ye et al. CHIMAERA: A High-Performance architecture with a tightly-coupled reconfigurable functional unit. In *IEEE Intl. Symp. on Computer Architecture*, 2000.

[23] S. Yehia et al. From sequences of dependent instructions to functions: An approach for improving performance without ILP or speculation. In *IEEE Intl. Symp. on Computer Architecture*, 2004.

[24] M. Yourst. PTLsim: A Cycle Accurate Full System x86-64 Microarchitectural Simulator. In *IEEE Intl. Symp. on Performance Analysis of Systems and Software*, 2007.