FPGA BASED HARDWARE-SOFTWARE CO-DESIGNED DYNAMIC BINARY TRANSLATION SYSTEM

Yuan Yao, Zhongyong Lu, Qingsong Shi, Wenzhi Chen

Department of Computer Science
Zhejiang University
Hangzhou, China
email: {yuanyao, lzy6032, zjsqs, chenwz}@zju.edu.cn

ABSTRACT

Binary translation is used to allow applications of one instruction set architecture (ISA) to run on another, thereby maintaining the binary level compatibility across ISAs. Conventional software binary translation systems suffer performance loss because of architectural heterogeneity amongst ISAs, control flow translation and context switches. In this paper, we propose an FPGA based hardware-software co-designed dynamic binary translation (DBT) system, which moderates these issues at a low level of hardware cost. In our DBT system, we propose a MIPS condition code flags register and a modest ISA extension to bridge the architectural gap, a hardware address mapping mechanism to accelerate the handling of control flow instructions, and a scratchpad memory to reduce performance loss during context switches.

We implement the system on Xilinx XC5VLX110T. Quantitative experiments reveal that the overall performance improvement is 56.1% over the baseline configuration, with only extra 1.4% of slices and 5.4% of BRAMs of Xilinx XC5VLX110T occupied.

1. INTRODUCTION

Binary translation is used to allow applications of one instruction set architecture (ISA) to run on another, thereby maintaining the binary level compatibility across different ISAs. Dynamic binary translation (DBT) is the most widely used approach of binary translation, which translates one non-native (source) ISA's binary executables into one native (target) ISA's binary executables at runtime, caching the translated code blocks for future reuse.

There are three major problems in DBT from the perspective of performance [1]. First, it requires effective and transparent mechanisms to eliminate the architectural heterogeneity between source and target ISAs. Second, the translation of branch instructions incurs dramatic performance loss. Since the target addresses of the branch instructions in the target machine are different from the original ones in the source machine, huge efforts will be paid on address mappings. Third, since DBT generates code on-the-fly, frequent context switches between the process of non-native code translation and the process of execution of generated native code leads to performance decline.

The time a DBT spends to run one non-native executable is of two aspects, the translation time and the execution time. Hence, to elevate the performance of a DBT system, either accelerate the translation process, or optimize the quality of the generated executables [2]. In this paper, we focus on the former aspect by adding hardware support mechanisms.

In this work, we propose an FPGA based hardware-software co-designed DBT system that translates x86 ISA executables into MIPS ISA executables. To moderate the aforementioned problems at low hardware cost, we propose a MIPS condition code (CC) flags register and a modest ISA extension to bridge the architectural gap and a hardware mechanism to accelerate the handling of control flow related instructions. In addition, we propose a scratchpad memory (SPM) to reduce performance loss during context switches. Quantitative experiments on Xilinx XC5VLX110T FPGA board reveal that our FPGA based accelerations outperform the baseline configuration by 56.1%, with only extra 1.4% of slices and 5.4% of on-chip occupied of dedicated BRAMs of Xilinx XC5VLX110T.

The rest of the paper is organized as follows. Section 2 shows the related work. Section 3 describes the baseline MIPS core and the high-level system structure of our DBT system. Section 4 presents the design goals and implementation details of hardware supports for dynamic binary translation. Section 5 exhibits and analyzes the performance of the system. Section 6 concludes the work.

2. RELATED WORK

There is a plethora of works proposed on DBT [1, 2, 3]. In this section, we focus on the works that invest hardware resources to achieve performance objectives.

Emulation of condition code is identified as a big cause of overhead. Harmonia [4] proposes to extend the ISA to reduce the most overhead of condition code emulation. It defines non-flags-modifying ALU instructions, thus avoiding the intervention of instructions that affect the CC-flags. However, the work is heavily in the context of translating ARM executables into Intel ATOM executables. In our work, to entirely eliminate the architectural

978-1-4799-0004-6/13/$31.00 ©2013 IEEE

heterogeneity, we propose an x86 CC-flags equivalent in our baseline core.

The translation of control transfer instructions is observed as another main contributor to performance loss [1, 5, 6]. The hardware address mapping mechanism in our work is first proposed by Kim et al. [5] as Jump Target-address Lookup Table (JTLT), which entirely avoids the expensive software-based target address lookup.

Endianness problem occurs when the source and target ISAs are of different endianness. David et al. [7] propose two general solutions that are address swizzling and byte swapping. However, both solutions are at the expense of performance. For example, a swap operation needs a bitwise and shift operation. Instead, we introduce hardware support for load/store instructions to access memory in a different endian order.

Previous works by Baiocchi et al. [8] have proposed to use a scratchpad memory to store the generated code, which improves the overall performance with large hardware overhead. We utilize augmented scratchpad memory to save temporary registers to speedup context switches with negligible hardware consumption (64 Bytes).

3. SYSTEM STRUCTURE

3.1. Baseline MIPS Core

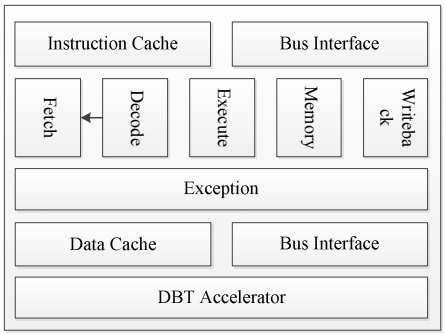

Fig. 1. Baseline core high-level structure

To support DBT in architectural level, we need to make ISA extension to the microprocessor. For purposes of simplicity and flexibility, we design and implement an FPGA based baseline 32-bit MIPS core. The core contains an in-order 5-stage pipeline and runs most of the MIPS R3000 instruction set[1]. We choose MIPS ISA for its implementation simplicity, nevertheless, the hardware supports proposed in this paper are applicable to any other source and target instruction sets.

Fig. 1 shows the high-level structure of the baseline core. We implement an 8KB directly mapping I-Cache and an 8KB 2-way set-associative D-Cache. It is optional to extend the baseline core with different hardware supports for DBT. We give the details of the hardware supports in Section 4.

[1] The baseline core runs MIPS R3000 instructions except the floating-point instructions and unaligned load and store instructions.

3.2. DBT System Execution Flow

There are two execution environments (source, target) in DBT system with distinct program counters (SPC, TPC). The whole flow of our HW/SW co-designed DBT system is illustrated in Fig. 2.

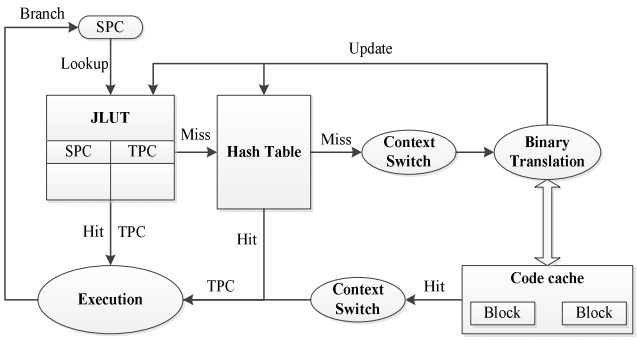

Fig. 2. Flowchart of the HW/SW co-designed DBT system

First, when the execution process is encountered with a source branch instruction, a lookup for TPC is triggered. The hardware JLUT which contains <SPC, TPC> pairs is responsible for finding TPC with index SPC. If it hits, the execution process continues from TPC with implication the target basic block has already been translated. If it fails, a software based hash table takes effect, which can be viewed as a level-two JLUT.

Second, if the hash table search also fails, a context switch occurs and the binary translation component takes over the program control flow. It translates the source executables into target executables by constructing a basic block in the code cache. Updates to JLUT and hash table are both required after the translation process is completed. Then another context switch happens and the execution procedure resumes.

Third, the execution procedure is interrupted when a branch instruction occurs and a new lookup is initiated.

4. HARDWARE SUPPORTS FOR DBT

4.1. Architectural Heterogeneity Elimination

4.1.1. MIPS CC flags Register

As there is no equivalent to the x86 CC flags in MIPS architecture, software DBTs have to emulate the functionalities of the CC flags, which involves a number of memory manipulations, which incurs dramatic performance

degradation. We propose an MIPS CC flags register to avoid the costly software emulation. The MIPS CC flags register gets affected by executing ALU related instructions.

4.1.2. Load/Store Little Endian Support

Conventional software approaches have to do byte conversions to eliminate the discrepancy in endianness, that leads to significant performance loss. We propose a modest ISA extension to the MIPS ISA to support memory accesses in little endian, that entirely avoids software byte conversions.

Table 1. Little endian load and store instructions

Instruction	Description
lw_le $rt, imm($rs)	Load one word in little endian
sw_le $rt, imm($rs)	Store one word in little endian

We define little endian load and store instructions for both word and half word accesses. Table 1 lists two word access instructions.

4.2. Control Flow Acceleration

To optimize control transfers, JLUT is implemented to accelerate the address mapping process, which is similar to the software-managed TLB in virtual memory systems [5]. Fig. 3 depicts the structure of JLUT. It is constituted of a CAM (content address memory) and a RAM (random access memory), which store SPCs and TPCs respectively.

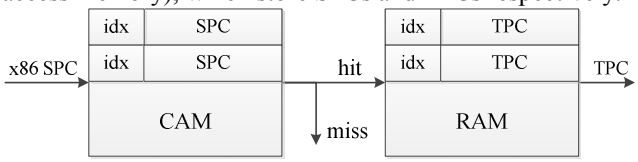

Fig. 3. Jump-address Lookup Table (JLUT)

In addition, several JLUT related instructions are also proposed to complement the original ISA [6]. One x86 SPC is used to look up in the CAM. If it hits in the CAM, then the corresponding TPC can be found in the RAM. If it misses, the control flow will be handed over to the DBT, which then look up the software hash table to make the decision whether it needs to fetch and translate a new basic block. The eviction of JLUT is managed by software.

4.3. Context Switch Optimization

Due to code cache misses, context switches enormously worsen the overall performance of a DBT system. During the initial phases of the execution, the program flow may switch between the DBT and execution process frequently, for the majority of the source executable has not been translated. To mitigate the performance loss, we propose a scratchpad memory (SPM) to speed up the context switches.

When a code cache miss occurs, we use the SPM to store the x86 registers values. After the DBT generates a new translated block, the scratchpad memory is accessed to restore the x86 registers values.

5. EVALUATION

We implement the DBT system on Xilinx XC5VLX110T. The prototype system translates a subset of x86 ISA into MIPS ISA. The FPGA prototyping speed is 50 MHz. We use lightly modified benchmarks selected from MiBench [9] to evaluate the system.

5.1. Performance Improvement

The baseline configuration is a software DBT system augmented with the proposed MIPS CC flags register. The JLUT is set to 64-entry. We measure the performance impact of different hardware supports respectively, and an overall performance improvement is also evaluated, which is referred as all. The normalized performance improvement is plotted in Fig. 4.

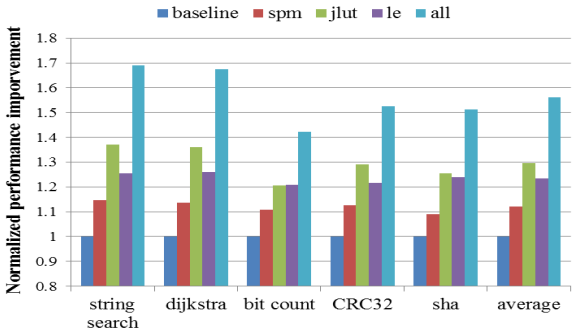

Fig. 4. Performance improvements by hardware supports

JLUT reveals the most prominent performance increase, which is at average 29.5%. Little endian support shows a performance increase by 23.5% whilst SPM shows a stable performance increase by 12.2% across benchmarks. With the exception of bit count, which shows a 42.1% speedup, the overall performance improvements are all over 50% across benchmarks, at average 56.1%. For string search, the performance increase is close to 70%, because of its considerable control flow operations and intensive memory accesses, which provide opportunities for little endian instructions and JLUT to give their best performance.

5.2. Impact on Instruction Cache Performance

To get more insights of the impact of the hardware supports on program behaviors, we also measure the performace of I-cache, as shown in Fig. 5.

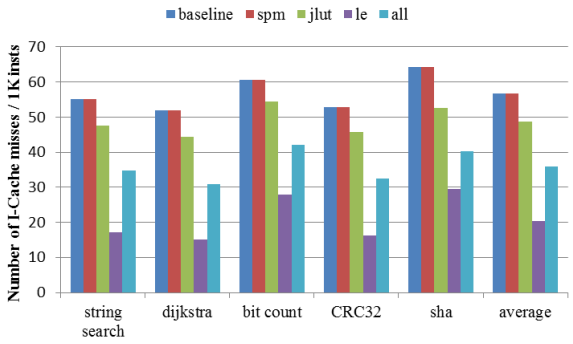

Fig. 5. Number of instruction cache misses

The impact of SPM on I-cache performance is fairly small, it's because SPM only takes effect when a context switch occurs, which does not affect the instruction locality of the generated code. JLUT improves I-cache performance by 7.9%. The speedup stems from that JLUT introduces a hardware mechanism that simplifies the process of address mapping in less than 10 instructions, which will take more than 30 instructions in a software hash table. The hardware mechanism apparently improves the program locality. The configuration with little endian support shows the most siginificant I-cache performance increase by 36.4%. We consider two aspects that are simutaneously attributable to the I-cache performance improvement. First, little endian support shortens the time spent on binary transtion. Depending on the memory intensity of the non-native program, it improves the locality of the translation process to some extent. Second, the code desnsity of the generated code cache block gets higher, which significantly relieves the pressure on I-cache.

5.3. Resource Utilization

Table 2 shows the consumed resources of the DBT system synthesized on Xilinx XC5VLX110T. The instruction cache, JLUT and SPM are mapped onto the dedicated on-chip Block RAMs. The total logic utilization of the baseline configuration is 9.2%, whilst the DBT system with all the hardware supports is 10.6%, which occupies extra 1.4% of slices and 5.4% of BRAMs of V5 on-chip resources.

Table 2. Resource utilization

	V5 Resources			
Configuration	Slices	LUTs	F/Fs	BRAMs
Baseline	9.2%	5.3%	2.8%	2.0%
SPM	9.4%	6.0%	2.8%	2.0%
JLUT	10.2%	6.4%	3.0%	7.4%
LE	9.5%	5.8%	2.8%	2.0%
All	10.6%	6.5%	3.0%	7.4%

6. CONCLUSION AND FUTURE WORK

In this paper, we propose an FPGA based hardware-software co-designed dynamic binary translation system. We present several key factors of performance and based on our baseline core, we propose hardware supports to speed up a DBT system. Finally, we validate our findings on Xilinx XC5VLX110T, which reveals an overall 56.1% performance increase at little hardware cost.

Our future work will focus on the power efficiency of a DBT system. We will also introduce multicore architecture in a DBT system.

7. REFERENCES

[1] E. Altman, D. Kaeli, and Y. Sheffer, "Welcome to the opportunities of binary translation," *IEEE Computer*, vol. 33, pp. 40–45, 2000.

[2] K. Ebcioglu, E. Altman, M. Gschwind, and S. Sathaye. "Dynamic binary translation and optimization," *Computers, IEEE Transactions on*, vol. 50, no. 6, pp. 529–548, 2001.

[3] V. Bala, E. Duesterwald, and S. Banerjia. "Dynamo: A transparent dynamic optimization system," In *PLDI*, pages 1–12, June 2000.

[4] G. Ottoni, T. Hartin, C. Weaver, J. Brandt, B. Kuttanna, and H. Wang. "Harmonia: a transparent, efficient, and harmonious dynamic binary translator targeting the Intel® architecture," In *Proceedings of the 8th ACM International Conference on Computing* Frontiers (CF '11). ACM, New York, NY, USA, , Article 26 , 10 pages, 2011

[5] H.-S. Kim and J. E. Smith, "Hardware support for control transfers in code caches," in *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*. Washington, DC, USA: IEEE Computer Society, 2003, p. 253.

[6] W. W. Hu, Q. Liu, and J. Wang et.al. "Efficient binary translation system with low hardware cost," *IEEE International Conference on Computer Design*, 2009.

[7] D. Ung and C. Cifuentes, "Dynamic binary translation using run-time feedbacks," *Sci. Comput. Program.*, vol. 60, no. 2, pp. 189–204, 2006.

[8] J. A. Baiocchi, B. R.Childers, J. W. Davidson, and J. D. Hiser. "Enabling dynamic binary translation in embedded systems with scratchpad memory," *ACM Transactions on Embedded Computing Systems*, 11(4), pp.1–33, 2012.

[9] M. R. Guthaus et al., "MiBench: A free, commercially representative embedded benchmark suite," in *IISWC*, Dec. 2001, pp. 3–14.