A CASE STUDY OF ENERGY-EFFICIENT LOOP INSTRUCTION CACHE DESIGN FOR EMBEDDED MULTITASKING SYSTEMS

Ji Gu and Tohru Ishihara
Department of Communications and Computer Engineering, Kyoto University, Kyoto 606-8501, Japan

Keywords: Energy Efficiency, Optimization Techniques, Microprocessor, Multitasking, Embedded Systems.

Abstract: Microprocessors increasingly execute multiple tasks in step with the increasing complexity of modern embedded applications. Shared by multiple tasks, conventional on-chip L1 instruction cache (I-cache) usually suffers a high cache miss ratio due to inter/intra task interferences and is the most energy-consuming component of the processor chip. This paper presents a power-efficient loop instruction cache design for multitasking embedded applications, which is a two-fold technique that can significantly reduce the L1 I-cache accesses for energy saving and reduce the I-cache misses caused by task interference. Experiments on a case study show that our scheme reduces energy consumption in the I-cache hierarchy by 36.5% and I-cache misses can be reduced from 6.0% to 18.3%, depending on the frequency of context switch in the multitasking system.

1 INTRODUCTION

Microprocessors are widely used in the computing systems of our daily life, from desktop PCs, workstations, and servers, to portable consumer-electronics devices such as PDAs, mobile phones, MP3/video players and digital cameras. In step with the shrinking size of deep submicron process technology, power dissipation in processors are exponentially increasing. It has been reported that, microprocessors globally used in data centers consume 1.5% of the worldwide energy (Pan, 2009). Therefore, low power is particularly important in the design of embedded systems such as portable and handheld electronics devices. These systems are mostly battery-driven and thus reducing power consumption can prolong the lifetime of batteries that have limited energy resources.

As on-chip caches increasingly occupy a larger die size, power consumption in these components accounts for a dominant portion of the overall processor energy. Work in (Dally et al., 2008) investigates the breakdown of microprocessor power and concludes that energy consumption in caches can amount to almost 70% of the total energy dissipated in the processor chip. Apparently, the cache components are the good targets for energy optimization.

The state-of-the-art embedded applications tend to incorporate multiple tasks running on a single processor in order to fully exploit the computing resources. In multitasking environment, several tasks run simultaneously and share all resources of the processor. Simultaneous running can be achieved by allocating a time slice for each task and executing the tasks at intervals. When a task yields its time slice or is preempted by another task with higher priority, a context switch needs to be performed, which involves saving the state of the current (preempted) task and retrieving the state of the next (preempting) task. Execution state of a task includes the program counter (PC) value, stack pointer (SP), register file (RF), program code and data in the caches. Saving the values of PC, SP and RF incurs small cost so it can be done by context switch. Cache state for a task, however, might be rather large and thus infeasible to be saved or retrieved during context switch. Therefore, while shared between several tasks, cache may suffer significant interference when code and data of a task are frequently overwritten by other tasks in the cache, which leads to a large number of cache misses. Since cache misses result in accesses to the lower level memory, which incurs even more energy consumption and larger delay, multitasking interference is problematic in terms of energy consumption and performance of the system.

To improve the energy efficiency of the multitasking systems, existing schemes attempt to reducing multitasking interference by allocating tasks to different partitions of the shared cache (Reddy and Petrov, 2010) (Paul and Petrov, 2011) or scratch-pad memory (Gauthier et al., 2010). While these approaches are effective in reducing cache interference for low energy, the design complexity is very high and most of them

Gu J. and Ishihara T..
A CASE STUDY OF ENERGY-EFFICIENT LOOP INSTRUCTION CACHE DESIGN FOR EMBEDDED MULTITASKING SYSTEMS.
DOI: 10.5220/0003951001970202
In *Proceedings of the 1st International Conference on Smart Grids and Green IT Systems* (SMARTGREENS-2012), pages 197-202
ISBN: 978-989-8565-09-9

require the large design space exploration at design time. For example, to decide the optimal size of cache partition for an individual task, a large space of cache configurations needs to be searched. With more and more tasks involved in a single multitasking application, such scheme becomes rather time-consuming and thus infeasible in practice.

In this paper, we propose an energy-efficient Partitioned Loop Instruction Cache (PLIC) design to reduce the energy consumption of the shared cache in the embedded multitasking systems. The proposed PLIC is motivated by a previous work (Gu and Guo, 2010) targeting the energy efficiency of single-task based system, and improved in this paper for multitasking systems. The partitioned loop instruction cache can be shared with multiple tasks without any interferences. In comparison with I-cache partition for multitasking applications(Paul and Petrov, 2011), partitioning the loop instruction cache has lower design complexity yet can effectively reduce the task interference in the shared I-ache. With our PLIC design, access of the shared cache in the multitasking system can be significantly reduced such that cache misses due to interference can be effectively reduce as well, which, as a consequence, can efficiently reduce the energy consumption and improve the performance of the multitasking systems.

2 DESIGN OF PLIC

The proposed PLIC is designed for multitasking embedded processors, based on the loop instruction cache (Gu and Guo, 2010) for single-task based processors. In order to be shared with multiple tasks and address the features of context switch of the multitasking applications, the proposed PLIC has extended and improved the loop instruction cache design as follows:

1. PLIC introduces a partitioned structure of the loop instruction cache and a hardware-based task state table to manage the sharing of PLIC with multiple tasks. With the proposed architecture, different tasks can exploit the PLIC without interfering each other during their executions.
2. Instead of caching the decoded instructions, our PLIC design caches the encoded (original) instructions. Size of decoded instruction is highly machine dependent and can be several times larger than the encoded instruction, which may result in a large PLIC design size for the multitasking application. Hence, as smaller cache size is more energy-efficient, we opt for a PLIC architecture design for encoded instructions.

2.1 PLIC Architecture

The PLIC proposed in this paper is an extra level of small loop cache designed for multitasking processors. Fig. 1 (a) shows the PLIC design in a pipelined processor. For the sake of simplicity, only three typical pipeline stages, IF (instruction fetch), ID (instruction decode) and EX (execution), are given in the figure. The PLIC component is placed at the ID stage and controlled by special instructions at the software level.

The architecture design of PLIC can be seen in Fig. 1 (b). PLIC takes inputs from the IF, ID and EX pipeline stages at the hardware level and inputs from the OS at the software level. While inputs from the pipeline are received during task execution, inputs from OS are received only when context switch occurs. The PLIC for the multitasking processor is composed of five components: a task sate table, a PLIC index table, a PLIC branch target table, the branch address logic, and the local PC logic.

The task state table stores the information of the PLIC cache partition allocated for each task. It also has a field for the local program counter (L-PC) of each task, which is used to save the execution state of the preempted task and retrieve the state of the preempting task during context switch. The PLIC index table and PLIC branch target table are the components for caching the loop instructions, each of which is partitioned for the multitasking application. When a task is using the PLIC during its execution, only the allocated partition of each table is used. The size of each partition to be allocated for tasks is decided at compile time, based on a static loop profiling of the multitasking application. And such partition information will be loaded into the task state table at the beginning of the program execution.

The PLIC branch target table is used to cache the flow control instructions (such as jump/branch) of a loop and the PLIC index table is used for other loop instructions. These two tables, together with the branch address logic, the local PC logic, and the four special instructions (*slp*, *elp*, *brb* and *brf* as shown in Fig. 1(b)) are used to handle the complex flow control of loop executions after the loop has been cached in the PLIC after the first loop iteration. A description about such loop execution control can be seen in (Gu and Guo, 2010) and thus will not be discussed in detail here. This paper focuses on the PLIC operation for context switch, which is the feature of the multitasking system and will be elaborated in following section.

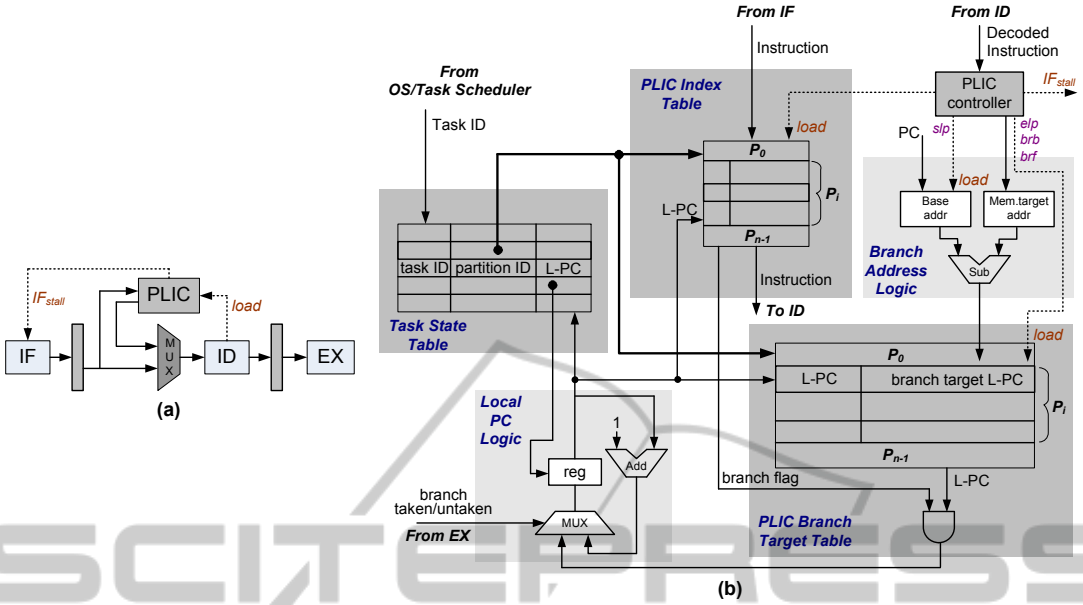

Figure 1: The proposed PLIC design based on (Gu and Guo, 2010): (a) PLIC in a processor pipeline, (b) The architecture of PLIC.

2.2 Context Switch and PLIC Operation

In multitasking systems, context switch occurs when a task yields its time slice, or a task is preempted by another task with higher priority. That is, only one task is running in the system at any point of time before and/or after context switch.

When context switch is not taking place, the PLIC is operated dynamically by the running task, with the allocated partition being used. In this case, the PLIC operation can be classified into three states: non-loop execution, first loop iteration, and following loop iteration (second to the last). In non-loop execution, the PLIC is not activated and instructions are fetched form the I-cache. In first loop iteration, the instructions are fetched from I-cache and loaded into PLIC at the same time. During following loop iteration, instructions are fetched from the PLIC only.

When a context switch occurs, state of the preempted task needs to be saved first, and then state of the preempting task needs to be retrieved for running. In the proposed PLIC based multitasking system, apart from the conventional context switch operations (i.e., saving PC, register values etc.), the PLIC operation state (non-loop execution, first loop iteration, or following loop iteration, as described above) of current task also needs to be saved for future recovery. For context switch, the PLIC operation depends on if the preempted task is using PLIC (*Case 1*) or not (*Case 2*).

- *Case 1.* Apart from performing conventional context switch and saving the PLIC operation state, local PC (L-PC) of the preempted task needs to be saved into the PLIC task state table for future recovery.
- *Case 2.* Conventional context switch at the OS level is performed, and the PLIC operation state (non-loop execution) of the preempted task is saved.

After saving all states of the preempted task, the preempting task can be put into execution based on its retrieved states.

3 EXPERIMENTAL RESULTS

3.1 Experimental Setup

We applied our partitioned loop instruction cache (PLIC) design to a multitasking application, as a case study, to evaluate the energy efficiency of our scheme in multitasking embedded systems. The multitasking application is composed of five benchmarks (i.e. tasks, *adpcm*, *jpeg*, *rawdaudio*, *sha*, *stringsearch*) from MiBench (Guthaus et al., 2001) and Powerstone (Scott et al., 1998) suites, which are widely used in the embedded domain of telecommunication, image processing, audio/vedio coding, and security.

We use a multiprocessor system with a task scheduler to emulate the multitasking system since there

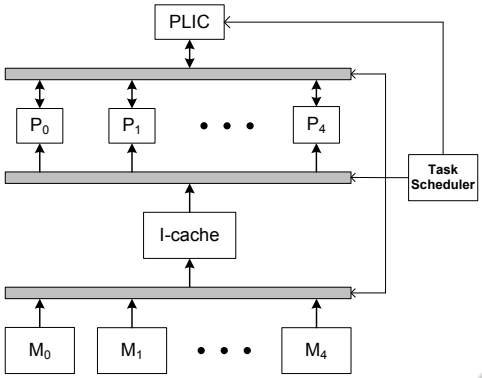

Figure 2: Experimental platform.

are no OS supporting context switch available for us. The platform can be seen in Fig. 2, which is composed of five processor cores, five main memory components, the shared L1 I-cache, the PLIC design, the task scheduler and some switching logic controlled by the task scheduler.

Each task is stored in one of the five memory components after compilation. At any time, only one processor is running by executing the task from its corresponding memory. Context switch is performed by the task scheduler by clock-gating the active processor and putting another processor into running, which is similar to saving states for the preempted task and retrieving states for the preempting task. During context switch, the task scheduler also needs to send a signal and ID of the preempting task to PLIC, for which to update the task state table and perform context switch in the PLIC cache (see Section 2.2). Since the L1 I-cache and the PLIC design is shared by multiple processors, their behavior is the same as used in a multitasking uniprocessor.

The processor cores in our platform are homogeneous and based on Simplescalar PISA (Burger and Austin, 1997). We use VHDL to specify the platform at RTL level for simulation. In this paper, we focus on the I-cache of multitasking processor and the memory hierarchy is assumed two levels with I-cache and main memory. The task scheduler utilizes a round-robbin scheme for tasks scheduling, with switching intervals of 5K, 10K, and 20K clock cycles.

Size of our PLIC design is decided by the three tables as discussed in Section 2.1. We use a PLIC index table (PIT) of 128 entries and a PLIC branch target table (PBTT) of 32 entries for partition. The task state table (TST) is set 8 entries, which is large enough for the 5 tasks in our experiment. The parameters setting of our platform is given in Table 1.

Table 1: System settings.

Processor	PISA RISC processor, 6-stage single pipeline
Instr. width	64bits
PLIC	PIT: 128x(1+64)bits
	PBTT: 32x(6+6)bits
	TST: 8x(3+3+6)bits
I-cache	8KB, 2-Way, 32B block, 1-cycle latency
Memory	64MB SDRAM, 30 cycles
Task scheduling	Round-Robbin, 5K, 10K, 20K cycles interval

3.2 Reduction of I-cache Access and Miss

We first investigate how much of the I-cache access and miss can be reduced by the proposed PLIC design for the multitasking application, since such reduction is attributed to the energy savings in the system. Fig. 3 reports the reduction rate of I-cache access and miss over the baseline system which does not have a PLIC design. The given results are based on three different switching intervals (5K, 10K, and 20K cycles) with different orders of task scheduling(system starting with T_0, T_1, T_2, T_3 or T_4).

I-cache access can be reduced by 50.9% and such reduction is independent on the scheduling policy of the multitasking system. This is due to that, sharing the partitioned PLIC, tasks do not interfere with each other so that hit rate of PLIC is identical for each running. In contrast, the frequency of context switch impacts much on the task interference in I-cache. With higher switching frequency, the I-cache miss becomes larger and more miss reduction can be achieved by our scheme. On average, our PLIC design can reduce the I-cache miss from 6.0% to 18.3%.

3.3 Energy Savings

To evaluate the energy efficiency of our scheme, we calculate energy consumption in the memory hierarchy of the baseline architecture and the PLIC design. We use the following energy model for the calculation:

$$\text{Baseline}_{energy} = \text{I-cache}_{access} \times \text{I-cache}_{energy/access} + \text{I-cache}_{miss} \times \text{Memory}_{energy/access}$$

$$\text{PLIC}_{energy} = \text{I-cache}_{access} \times \text{I-cache}_{energy/access} + \text{I-cache}_{miss} \times \text{Memory}_{energy/access} + (\text{PLIC}_{write} + \text{PLIC}_{access}) \times \text{PLIC}_{energy/access}$$

where values of I-cache_{access}, I-cache_{miss}, PLIC_{write} and PLIC_{access} are obtained by simulating the application on our VHDL model. Energy consumption per access of I-cache and main memory are calcu-

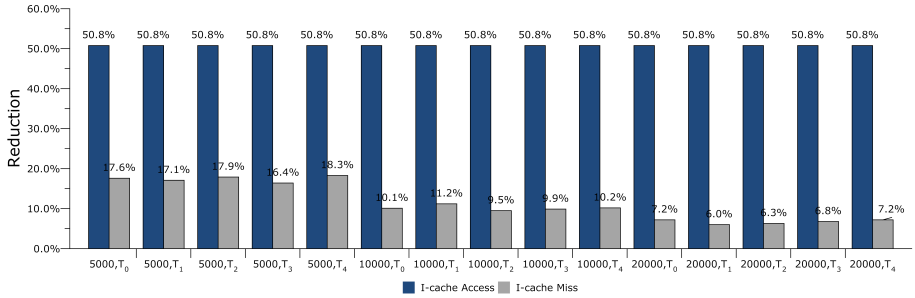

Figure 3: Reduction of I-cache accesses and misses.

lated in CACTI 5.1 (Thoziyoor et al., 2008) using 65nm process technology. For our PLIC cache, the PLIC index table can be implemented as a tagless direct-mapped cache with cache line of one instruction word, and thus we also use CACTI to calculate the energy. The other two tables (PLIC branch target table and task sate table) can be implemented as register files. Therefore, these two tables and other control logic of PLIC are synthesized in Synopsys Design Compiler for the energy values, also using the 65nm process technology. Energy consumption of caches and memory is given in Table 2.

Table 2: Energy consumption per access.

	Energy/Acess [pJ]
PLIC	56.3
I-cache	227.3
Memory	6332.5

Fig. 4 reports the energy consumption of our PLIC design normalized to the baseline system. As can be seen, PLIC can reduce energy consumption by 36.5% for the multitasking system. Note that, though reduction of cache misses varies for each task scheduling (see Fig. 3), the energy reduction is almost identical. This is because, even cache misses have increased due to task interference, the miss ratio (cache misses over cache references) is still small in the multitasking applications. As a result, energy reduction due to cache miss reduction accounts for a minor portion of the overall energy savings.

4 RELATED WORK

Several low-power techniques for embedded systems have been focused on improving the cache hierarchy for power and energy efficiency. Work in (Malik et al., 2000) makes the set-associative cache behave like a direct-mapped cache such that tag comparison can be reduced for power efficiency. The way-halting cache (Zhang et al., 2005) design uses some least significant tag bits for comparison, an un-match of which can filter the full tag comparison for power saving. Ishihara and Fallah (Ishihara and Fallah, 2005) propose a software/hardware co-design of a non-uniform cache architecture, where the cache can have different number of ways for different cache sets. The unused cache ways are disconnected from the sense-amplifiers for power saving. Techniques in (Tang et al., 2002) (Yang and Lee, 2004) (Gu and Guo, 2010) attempt to reducing cache energy by introducing an extra level of tiny and low-power cache structure in front of the conventional cache so that the power-expensive cache accesses can be mostly filtered.

For multitasking systems, work in (Reddy and Petrov, 2010) and (Paul and Petrov, 2011) propose to allocate tasks to different partitions of cache such that cache misses due to task interferences can be reduced. Gauthier et al. (Gauthier et al., 2010) attempt to finding the optimal allocation of scratch-pad memory space between tasks, which can effectively reduce accesses of the main memory and hence reducing energy consumption. In contrast, our PLIC design uses the filtering scheme to reduce the power consumption of embedded multitasking systems. By filtering the I-cache assesses, the PLIC can also reduce the cache misses caused by task interferences. In addition, as our technique is orthogonal to the technique of (Reddy and Petrov, 2010) (Paul and Petrov, 2011), they can be applied in combination for energy reduction in a multitasking system.

5 CONCLUSIONS

This paper targets the multitasking processors and attempts to reducing energy dissipation of the instruction cache hierarchy in processors. With our proposed partitioned loop instruction cache (PLIC) shared by multiple tasks, a significant amount of energy-expensive I-cache access can be filtered. Furthermore, filtering I-cache access leads to effective reduction of I-cache misses caused by task interfer-

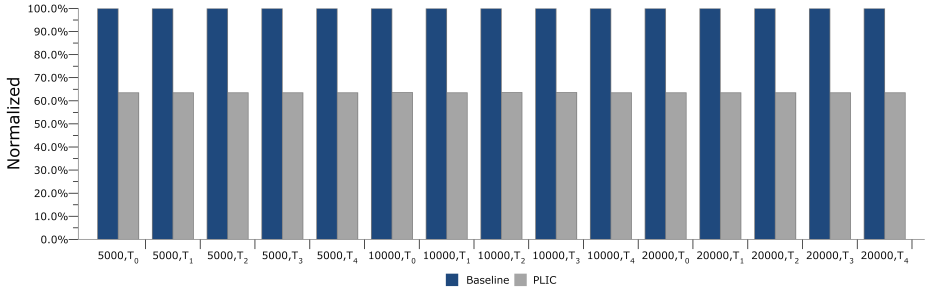

Figure 4: Energy consumption normalized to the baseline design.

ence, and hence reducing energy and delay of lower-level memory access. Experiments on a case study of multitasking application reports an energy reduction of 36.5% with the PLIC design. The reduction on I-cache misses ranges from 6.0% to 18.3%, depending on the frequency of context switch in the multitasking system.

ACKNOWLEDGEMENTS

This work is supported by JSPS NEXT program under grant number GR076.

REFERENCES

Burger, D. C. and Austin, T. M. (1997). The simplescalar tool set, version 2.0. *Technical Report CS-TR-1997-1342, Department of Computer Science, University of Wisconsin, Madison*.

Dally, W. J., Balfour, J., Black-Shaffer, D., Chen, J., Harting, R. C., Parikh, V., Park, J., and Sheffield, D. (2008). Efficient embedded computing. *IEEE Computer*, 41(7):27–32.

Gauthier, L., Ishihara, T., Takase, H., Tomiyama, H., and Takada, H. (2010). Minimizing inter-task interferences in scratch-pad memory usage for reducing the energy consumption of multi-task systems. In *Proceedings of the 2010 international conference on Compilers, architectures and synthesis for embedded systems (CASES'10)*, pages 157–166.

Gu, J. and Guo, H. (2010). Enabling large decoded instruction loop caching for energy-aware embedded processors. In *Proceedings of the 2010 international conference on Compilers, architectures and synthesis for embedded systems (CASES'10)*, pages 247–256.

Guthaus, M. R., Ringenberg, J. S., Ernst, D., Austin, T. M., Mudge, T., and Brown, R. B. (2001). Mibench: A free, commercially representative embedded benchmark suite. In *IEEE 4th Annual Workshop on Workload Characterization*, pages 83–94.

Ishihara, T. and Fallah, F. (2005). A non-uniform cache architecture for low power system design. In *Proceedings of the 2005 International Symposium on Low Power Electronics and Design (ISLPED'05)*, pages 363–368.

Malik, A., Moyer, B., and Cermak, D. (2000). A low power unified cache architecture providing power and performance flexibility. In *Proceedings of the 2000 International Symposium on Low Power Electronics and Design (ISLPED'00)*, pages 241–243.

Pan, D. Z. (2009). Low power design and challenges in nanometer multicore era. In *IEEE CAS Melbourne and Victoria University, Invited Talks, August 20, 2009*.

Paul, M. and Petrov, P. (2011). Dynamically adaptive i-cache partitioning for energy-efficient embedded multitasking. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(11):2067–2080.

Reddy, R. and Petrov, P. (2010). Cache partitioning for energy-efficient and interference-free embedded multitasking. *ACM Transactions on Embedded Computing Systems (TECS)*, 9(3):16:1–16:35.

Scott, J., Lee, L. H., Arends, J., and Moyer, B. (1998). Designing the low-power m-core architecture. In *International Sympsium on Computer Architecture Power Driven Microarchitecture Workshop*, pages 145–150.

Tang, W., Gupta, R., and Nicolau, A. (2002). Power savings in embedded processors through decode filer cache. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'02)*, pages 443–448.

Thoziyoor, S., Muralimanohar, N., Ahn, J. H., and Jouppi, N. P. (2008). CACTI: An integrated cache and memory access time, cycle time, area, leakage, and dynamic power model. *Technical Report HPL-2008-20, HP Laboratories*.

Yang, C.-L. and Lee, C.-H. (2004). Hotspot cache: Joint temporal and spatial locality exploitation for icache energy reduction. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 114–119.

Zhang, C., Vahid, F., Yang, J., and Najjar, W. (2005). A way-halting cache for low-energy high-performance systems. *ACM Transactions on Architecture and Code Optimization (TACO)*, 2(1):34–54.