Exploiting SPM-aware Scheduling on EPIC Architectures for High-Performance Real-Time Systems

Yu Liu, Wei Zhang (wzhang4@vcu.edu)
Department of ECE, Virginia Commonwealth University

***Abstract*—In contemporary computer architectures, the Explicitly Parallel Instruction Computing Architectures (EPIC) permits microprocessors to implement Instruction Level Parallelism (ILP) by using the compiler, rather than complex on-die circuitry to control parallel instruction execution like the superscalar architecture. Based on the EPIC, this paper proposes a time predictable two-level scratchpad based memory architecture, and a Scratchpad-aware Scheduling method to improve the performance by optimizing the Load-To-Use Distance.**

***Index Terms*—SPMs, Real-Time Systems**

I. Introduction

The Explicitly Parallel Instruction Computing (EPIC) style of architecture is an evolution of VLIW that has also absorbed many superscalar concepts albeit in a form adapted to EPIC [11]. The EPIC utilizes the compiler to exploit high ILP like VLIW, while dealing with dynamic factors better than the basic VLIW architecture. In order to generate a high quality schedule, the compiler must do a good job of predicting what the latency of each load will be [11], which is called the Load Sensitive Scheduling.

Scratchpad memories (SPMs), as an alternative technique to hardware-controlled caches, offer the characteristic of time predictability and reasonable performance [1], [2], [3], [7], [8]. Scratchpad memories are small physical separate memories directly mapped into the address space of a memory system, which always use high speed SRAM. To efficiently exploit SPMs, however, it is crucial to determine the memory objects assignment to scratchpad memories by either the static or dynamic algorithms. The static algorithms [2], [3] focus on proposing compilation time algorithms to statically allocate hot spots of programs into scratchpad memories, which is very friendly to the memory latency prediction. In contrast, dynamic allocation approaches [6], [7], [8] may harm time predictability, due to dynamical replacement of memory objects, although they may further improve performance.

In this paper, we propose a two-level architecture with separate L1 instruction and data SPMs and a shared L2 SPM. An ILP based static memory objects assignment algorithm is utilized for the proposed architecture in order not to harm the characteristic of time predictability of scratchpad memories. Based on our proposed memory architecture on the EPIC microprocessors, we study a novel Scratchpad-aware Scheduling method to further improve the performance of the proposed architecture by exploiting the statically known load/store latencies due to the use of SPMs, which are important for high-performance real-time systems. Our experimental results indicate that the performance of the two-level scratchpad based architecture on the EPIC processors can be improved by the Scratchpad-aware Scheduling, while our architecture has the nature of time predictability that is demanded by real-time systems.

II. Scratchpad Memories

To combine the advantages of both scratchpad memories and two-level architectures, we propose a two-level scratchpad based architecture on the EPIC microprocessors for real-time systems. The L1 scratchpad memory is the fastest SRAM with small size, while the L2 scratchpad memory is a slower SRAM with larger size. Since there is no replacement requirement in any higher level memory of our architecture, the L1 instruction, L1 data, L2 shared scratchpad and the main memory are all directly connected to the microprocessor. Also, this working mechanism shortens the memory latencies because of smaller size of data operations (the whole cache line size data needs to be updated in the cache based architecture) and no replacement requirement.

In this paper, we utilize the static algorithm to allocate memory objects to scratchpad memories, since the static method can guarantee the characteristic of time predictability in nature, which is the most pressing desire of the real-time system. An ILP based method is utilized to allocate the hot spots to the three SPMs in our architecture.

III. Scratchpad-aware Scheduling

We call an load instruction as *Load Op*, which loads data from memories. Also, we call an instruction that uses the data loaded by a Load Op as *Use Op*. The number of execution cycles scheduled between the Load Op and Use Op is refer as *Load-To-Use Distance* in this paper. Also, the *Stall-On-Use* model is commonly used to handle the data cache misses in EPIC, in which the microprocessor will not be stalled until the Use Op is executed. Obviously, the Stall-On-Use model may reduce the *Use Stall Cycles* based on the Load-To-Use Distance, and the reduced stall cycles are called the *Hidden Stall Cycles*.

According to the analysis above, we can benefit from properly scheduling the Load-To-Use Distance. For instance, if we can schedule a Load Op having large memory latency earlier and schedule its Use Op later, we can get the maximum Hidden Stall Cycles to shorten the Use Stall Cycles. Meanwhile, the microprocessor cycles within the Load-To-Use Distance can be used to schedule other instructions that do not depend on this load instruction [4], which can improve the degree of instruction level parallelism to shorten the computation time. The principle advantage of our scratchpad based architecture is time predictable since the memory objects are statically allocated to scratchpad memories in the compiler stage and no dynamic replacement is requested during the running time. Therefore, we can develop a scheduling algorithm named *Scratchpad-aware Scheduling* to be sensitive to the varying memory latencies of load instructions, as the data can be loaded from the L1 data SPM, the L2 SPM or the main memory.

Level	L1 Inst.	L1 Data	L2 Shared	Main Memory
Cache	1	1	10	100
Scratchpad	1	1	3/5	10/20

TABLE I
LATENCIES SETTINGS OF THE SPM BASED ARCHITECTURE AND ITS CORRESPONDING CACHE BASED ARCHITECTURE.

Benchmark	Scratchpad (Default)		Scratchpad (SSS)		Difference	
256-128	Computation	Use Stall	Computation	Use Stall	Diff (C)	Diff (U)
compress	4438	294	4389	294	0.989	1.000
crc	25208	984	25208	984	1.000	1.000
lms	457557	121527	452632	113346	0.989	0.933
ludcmp	3138	2698	3017	2554	0.961	0.947

TABLE II
THE COMPARISON OF PERFORMANCE IMPROVEMENT OF SCRATCHPAD SENSITIVE SCHEDULING (L1 SIZE: 128 BYTES, L2 SIZE: 256 BYTES).

IV. EVALUATION METHODOLOGY

We study the Scratchpad-aware Scheduling with our proposed two-level SPM based architecture on a EPIC processor with 4 integer functional units, 2 floating point functional units, 1 load/store unit, and 1 branch unit based on the HPL-PD architecture [10]. The EPIC processor has a global register file with 32 registers. The trimaran compiler/simulator infrastructure [9] is used to evaluate the performance of the target EPIC processor. Trimaran consists of a frontend compiler IMPACT, a backend compiler ELCOR, and a cycle-level EPIC processor simulator.

To comparatively evaluate our SPM based architecture with our Scratchpad-aware Scheduling, we select four real-time benchmarks with obvious use stall time [5]. All benchmarks are compiled by Trimaran compiler under setting gcc optimization 0 and using Critical Path Scheduling. The default value of latencies in different levels of memories for the SPM based architectures can be found in Table I, which is determined by considering the advantages of our SPM architecture to reduce the related latencies of the Cache based architecture. In Table I, the small value for the L2 share scratchpad memory and main memory is used for instruction fetch or 1 word width data operation, while the large value is for 2 words width data operation in our proposed SPM based architecture.

V. RESULTS AND ANALYSIS

Our experimental results of Scratchpad-aware Scheduling are shown in Table II, III, and IV for different cache/scratchpad size settings, respectively. As expected, our Scratchpad-aware Scheduling may improve the computation and Use Stall Cycles if there is flexibility in scheduling. Our selected four real-time benchmarks are with obvious Use Stall Cycles, which have possibility to be improved by our scheduling algorithm. From Table II, III and IV, we find that 75% of these four benchmarks get performance improvement. The maximum improvement of computation cycles is about 3.9%, and the maximum improvement of use stall cycles is about 10%.

Also, we observe larger improvement of Use Stall Cycles on the benchmarks with original large Use Stall Cycles (e.g. the benchmark *lms* and *ludcmp*), while there is almost no improvement on the benchmarks with original small Use Stall Cycles (e.g. the benchmark *compress* and *crc*). The reason is that there are more possible pairs of Load and Use Ops to be properly set the Load-To-Use Distance based on the memory latency of Load Op on these benchmarks with original more Use Stall Cycles. Also, the Hidden Stall Cycles may dominate the Use Stall Cycles calculation, which is not visible. Moreover, due to the limitation of basic block based scheduling, there may be no flexibility of adjusting the Load-To-Use Distance within a basic block, if there is no alternative candidate for Load and Use Ops in their scheduled microprocessor cycles. As a result, we cannot achieve any visible improvement for some benchmarks with small original Use Stall Cycles, such as the benchmark *crc*. Such limitation may be addressed in the future by using compiler optimizations or other scheduling approaches to enlarge the scheduling working scope, such as trace scheduling.

Benchmark	Scratchpad (Default)		Scratchpad (SSS)		Difference	
512-256	Computation	Use Stall	Computation	Use Stall	Diff (C)	Diff (U)
compress	4438	1176	4389	1176	0.989	1.000
crc	25208	984	25208	984	1.000	1.000
lms	457557	81802	456850	73621	0.998	0.900
ludcmp	3138	3223	3017	3169	0.961	0.983

TABLE III
THE COMPARISON OF PERFORMANCE IMPROVEMENT OF SCRATCHPAD-AWARE SCHEDULING (L1 SIZE: 256 BYTES, L2 SIZE: 512 BYTES).

Benchmark	Scratchpad (Default)		Scratchpad (SSS)		Difference	
1024-512	Computation	Use Stall	Computation	Use Stall	Diff (C)	Diff (U)
compress	4438	1335	4389	1335	0.989	1.000
crc	25208	329	25208	329	1.000	1.000
lms	457557	136403	456853	129461	0.998	0.949
ludcmp	3138	3336	3017	3336	0.961	1.000

TABLE IV
THE COMPARISON OF PERFORMANCE IMPROVEMENT OF SCRATCHPAD-AWARE SCHEDULING (L1 SIZE: 512 BYTES, L2 SIZE: 1024 BYTES).

VI. CONCLUSIONS

In this paper, we proposes a time-predictable two-level scratchpad based architecture, and utilize an ILP based static memory object assignment algorithm to maintain time predictability. Based on our proposed memory architecture on EPIC processors, we study the Scratchpad-aware Scheduling to further improve the performance of our proposed architecture. Our experimental results indicate that the performance of the two-level scratchpad based architecture on the EPIC processors can be improved by the Scratchpad-aware Scheduling, while keeping time predictability that is the crucial for real-time systems.

REFERENCES

[1] L. Wehmeyer, and P. Marwedel, Influence of On-Chip Scratchpad Memories on WCET prediction, In Proceedings of the 4th International Workshop on Worst-Case Execution Time (WCET) Analysis, 2004.

[2] O. Avissar and R. Barua, An Optimal Memory Allocation Scheme for Scratchpad-Based Embedded Systems, in the Journal of ACM Transactions on Embedded Computing Systems (TECS), Volume 1, Issue 1, Nov. 2002.

[3] S. Steinke, et. al, Assigning Program and Data Objects to Scratchpad for Energy Reduction, in the Proceedings of the conference on Design, automation and test in Europe, 2002.

[4] C. Hardnett, et. al, Scheduling Load Operations on VLIW Machines, Georgia Institute of Technology Technical Report GIT-CC-01-015, 2001.

[5] Real-Time Benchmarks, http://www.mrtc.mdh.se/projects/wcet/benchmarks.html

[6] S. Metzlaff, et. al, Predictable Dynamic Instruction Scratchpad for Simultaneous Multithreaded Processors, in the Proceedings of the 9th workshop on Memory performance: Dealing with Applications, systems and architecture, 2008.

[7] S. Steinke, et. al, Reducing Energy Consumption by Dynamic Copying of Instructions onto On-Chip Memory, in the Proceedings of the 15th international symposium on System Synthesis, 2002.

[8] M. Kandemir, et. al, Dynamic Management of Scratchpad Memory Space, in the Proceedings of the 38th annual Design Automation Conference, 2001.

[9] Trimaran Homework: http://www.trimaran.org

[10] V. Kathail, and M. Schlansker, and B. R. Rau, HPL-PD architecture specification version 1.1 HPL Technical Report, 2000.

[11] M. Schlansker, and B. Rau, EPIC: Explicitly Parallel Instruction Computing, In the Journal of Computer, vol. 33, no. 2, pp. 37-45, Feb. 2000.