Graduate School of Fundamental Science and Engineering
Waseda University

博 士 論 文 概 要
Doctoral Thesis Synopsis

論 文 題 目
Thesis Theme

Studies on Compiler Controlled Cache Coherency for Multicore Processors

申 請 者
(Applicant Name)

Boma Anantasatya	ADHI
アディ	ボマ アナンタサチャ

Department of Computer Science and Communications Engineering,
Research on Advanced Computing Systems

January, 2020

Since the last decade, processor clock speed has stopped increasing, marking the imminent end Moore's law. As a result, instead of pursuing a higher frequency single-core processor, the industry has shifted towards a multicore processor. The multicore processors do not have run as fast as the single-core processor, but they perform more works through parallelism.

Most modern multicore processors today are based on a shared memory system. Shared memory is a low-level communication paradigm that allows multiple processor cores to work and communicate through a shared memory space. In such a system, the processor cores are usually equipped with caches to improve the read and write performance to the shared memory. Cache is a small but fast memory located between the larger and slower main shared memory and the processor core. Frequently or recently used data are stored in the cache, creating an illusion of a large but fast memory. The cache can be private for each core, shared between multiple cores or a combination of both. For a performance reason, a multicore CPU commonly has private caches for each CPU core, and if necessary, another level of caches shared between multiple cores.

While beneficial, a private cache in a multicore processor introduces a cache inconsistency. If a particular processor core updates a value of a shared data in its cache, the new updated value cannot be seen by other processor cores until the new value is flushed back to the shared memory, thus causing cache inconsistency. In order to avoid such an inconsistency problem, a cache coherency control mechanism is necessary. Typically, a modern multicore processor has a hardware-based mechanism taking care of the coherency.

Nowadays, we have processors with hundreds or even thousands of cores on a single chip. Most of them use a hardware-based mechanism to maintain cache coherency. However, on a larger scale multicore, a hardware-based cache coherency circuitry gets exceedingly complex, thus generating more heat and challenging to verify. The complexity also drives the development cost and time up. A hardware-based cache coherence may not scale well to a large number of cores expected to be found on future SMP machines. On the other side of the spectrum, adding a hardware-based mechanism sometimes cannot be justified due to limited resources or specific requirements, such as in a hard-real-time embedded system or soft processor cores implemented in FPGA. Without a hardware-based cache coherency mechanism, writing a program for such a Non-Cache Coherent (NCC) system is overwhelmingly complex, as the programmer has to maintain the cache coherency manually.

Therefore, we propose a compiler-controlled coherence scheme for shared-memory multicore systems without a hardware cache coherence mechanism. The cache coherency mechanism is built into the OSCAR automatic parallelizing compiler, providing an integrated solution for automatic parallelization and non-cache coherent system support.

This thesis comprises of five chapters:

Chapter 1 "Introduction" provides the background and explains the objectives of this research. This chapter also outlines the significance of this research by comparing related works.

Chapter 2 "OSCAR Compiler and OSCAR API" introduces the OSCAR Compiler and OSCAR API. The OSCAR Compiler is a source to source automatic parallelizing compiler. The compiler takes a C program, and then it decomposes the program into coarse-grain tasks. It then analyzes the control flow and the data dependency between those tasks. Based on this information, the compiler parallelizes the task and, later on, inserts the cache manipulation code for the coherency control and finally outputs a parallelized code with API directives annotation. The OSCAR Compiler is supplemented with the OSCAR API that consists of many directives for different purposes, like platform-specific functions, power management, and cache operation. The OSCAR API converts directives in the annotated code into the correct functions or driver call.

Chapter 3 "Compiler-Controlled Cache Coherence for Multicore processor" discusses two fundamental problems in NCC systems, "stale data/true sharing" and "false sharing". Stale data is a state when a processor core updates a shared data in its cache, but other processor core cannot get the updated value until the data is flushed back to the main shared memory. In order to handle this problem, as one of the cores updates a variable in its cache, the compiler then inserts self-invalidation instruction into the code segment in which other cores access the variable. The self-invalidate is an instruction for invalidating a specific line in the cache, which forces other cores to get the latest value from shared memory, thus eliminating inconsistency. The second problem, "false sharing", happens when two or more independent data reside in a single cache line. If one of those data is changed, inconsistency may occur; this is because the granularity of the cache writeback mechanism is at the line level instead of a single byte or word. The compiler performs several elemental data restructuring operations to prevent unrelated variables from sharing a single cache line. For a scalar or small-sized one-dimensional array, usually, cache alignment is sufficient. In the case of a multi-dimensional array, the compiler performs array expansion or array padding, and for the case in which all other methods are inapplicable, the compiler uses a non-cacheable buffer. Later on, this chapter discusses the necessary changes to implement the coherency control in the OSCAR Compiler and the OSCAR API.

Chapter 4 "Development of Non-Cache Coherent Architecture Evaluation Platform" explains three different test platforms used in this research, the RP2 multicore processor and two custom SoC based on Nios II and RISC-V. The Renesas RP2 is an 8-core embedded processor which comprises of two 4-core SH-4A SMP clusters, with each cluster has its hardware coherency domain. A regular program can run with up to four cores, but a software approach is necessary for cache coherency beyond the 4-core cluster. This research was also partially motivated by this RP2 limitation. The "Nios II-based SoC" is a simple SoC generated entirely in Altera Platform Designer. Nios II is a soft processor core which is intended for single-core operation in Altera FPGA without any multicore or hardware cache coherency support. Designing multicore Nios II SoC is relatively easy, but writing a program for it is difficult without any coherency mechanism. RISC-V is a relatively new and promising open-source hardware instruction sets architecture. There are a lot of available RISC-V processor implementation in the market, but unfortunately, there is no synthesizable and functional multicore RISC-V SoC with hardware cache coherence in the market.

Chapter 5 "Performance Evaluation of Compiler-Controlled Cache Coherence" explains the performance evaluation of the Compiler Controlled Cache Coherency. Ten benchmark programs from three benchmark suites, NAS Parallel Benchmark, SPEC, and MediaBench, are compiled by the OSCAR Compiler with NCC support. The proposed method achieves similar performance as the hardware-based coherence mechanism. The proposed method also allows us to automatically parallelize and successfully run the benchmark program on NCC 8-cores Nios II SoC and 4-cores RISC-V SoC, as if they are a cache-coherent SMP machine. For example, with NAS Parallel Benchmark "cg", we can obtain 3.71 times speedup on RP2 without hardware coherence support versus 3.34 times speedup with hardware coherence turned on. Also, the same benchmark program runs with 5.66 times speedup on 8-cores RP2, 5.89 times on 8-cores Nios II SoC, and 3.68 times on 4-cores RISC-V SoC, which all are otherwise impossible to run without hardware cache coherency.

Chapter 6 "Conclusion" concludes the thesis and discuss the future of this research.

早稲田大学　博士（工学）　学位申請　研究業績書

(List of research achievements for application of doctorate (Dr. of Engineering), Waseda University)

氏 名 Boma Anantasatya ADHI　　印

（As of January, 2020）

種 類 別 (By Type)	題名、　発表・発行掲載誌名、　発表・発行年月、　連名者（申請者含む）(theme, journal name, date & year of publication, name of authors inc. yourself)
Paper ○	Boma A. Adhi, Tomoya Kashimata, Ken Takahashi, Keiji Kimura, Hironori Kasahara “Compiler Software Coherent Control for Embedded High Performance Multicore”, IEICE Transaction on Electronics Special Section on Low-Power and High-Speed Chip, Mar 2020. (To appear)
○	Boma A. Adhi, Masayoshi Mase, Yuhei Hosokawa, Yohei Kishimoto, Taisuke Onishi, Hiroki Mikami, Keiji Kimura, Hironori Kasahara “Software Cache Coherent Control”, Proceedings of the 30th International Workshop on Languages and Compilers for Parallel Computing (LCPC2017), College Station, Texas, Oct. 11-13 2017.
	Hironori Kasahara, Keiji Kimura, Boma A. Adhi, Yuhei Hosokawa, Yohei Kishimoto, Masayoshi Mase “Multicore Cache Coherence Control by a Parallelizing Compiler”, 2017 IEEE 41st Annual Computer Software and Application Conference (COMPSAC), pp. 492-497. Turin, 2017.
	Boma A. Adhi, Masayoshi Mase, Yuhei Hosokawa, Yohei Kishimoto, Taisuke Onishi, Hiroki Mikami, Keiji Kimura, Hironori Kasahara “Software Cache Coherent Control”, In: Rauchwerger L. (eds) Languages and Compilers for Parallel Computing. LCPC 2017. Lecture Notes in Computer Science, vol 11403, pp 17-25. Springer, Cham
Academic Exhibition	Hiroki Mikami, Boma A. Adhi, Tomoya Kashimata, Satoshi Karino, Kazuki Miyamoto, Takumi Kawata, Ken Takahashi, Tetsuya Makita, Tomoya Shirakawa, Yoshitake Oki, Toshiaki Kitamura, Keiji Kimura, Hironori Kasahara “OSCAR Vector Multicore System Platinum Vector Accelerator on FPGA”, IEEE/ACM SC17, Denver, Colorado, Nov. 12-17 2017.