Improving Software Concurrency with Hardware-assisted Memory Snapshot

JaeWoong Chung, Jiwon Seo, Woongki Baek, Chi Cao Minh, Austen McDonald
Christos Kozyrakis, Kunle Olukotun
Computer Systems Laboratory
Stanford University
{jwchung, jiwon, wkbaek, caominh, austenmc, kozyraki, kunle}@stanford.edu

ABSTRACT

We propose a hardware-assisted memory snapshot to improve software concurrency. It is built on top of the hardware resources for transactional memory and allows for easy development of system software modules such as concurrent garbage collector and dynamic profiler.

Categories and Subject Descriptors: C.1.4 [Processor Architecture]: Parallel Architecture

General Terms: Performance, Design

Keywords: Transactional Memory, Memory Snapshot

1. INTRODUCTION

Chip-multiprocessors (CMPs) bring abundant parallelism to commodity systems. However, the lack of concurrency in applications prevents us from fully exploiting the additional hardware cores. To alleviate this problem, it is important to develop architectural tools that help programmers to exploit parallelism. In search of such tools, we found that *memory snapshot* is particularly useful for improving the concurrency of software systems [2]. The basic concept of snapshots has been used widely in databases, file systems and reliable storage. A snapshot is an image of a large dataset at a particular moment in time and provides a consistent view of the constantly mutating dataset for later use. While several algorithms for memory snapshot have been proposed, their usefulness for performance optimizations is limited because they rely on software techniques.

2. MSHOT

This paper proposes *MShot*, a hardware-assisted memory snapshot system. MShot supports memory snapshots of arbitrary lifetime that consist of multiple disjoint memory regions. With a single call to the MShot interface, the system takes an atomic snapshot of all regions and isolates the snapshot from further memory updates. The snapshot image can be shared by multiple threads that operate on the snapshot data using normal load/store instructions.

The goal of MShot is to allow for algorithmic simplicity, easy code management, and performance at the same time. It provides programs with an automated way to manage semantically unnecessary contention for shared data in a concurrent system. It is often the case that some threads need access to a recent and consistent version of the data, while other threads try to update data with more recent results. For example, a memory profiler needs a consistent view on the object reference graph at recent time. Using MShot, the memory profiler can save a recent image of the object reference graph fast and atomically and safely analyze the graph in parallel with the application threads working on the up-to-date image of the graph. There is no additional coding and time wasted on synchronization between the application and profiler threads. With MShot, it is easy for unskilled programmers to understand the concept of memory snapshot and use it to improve software concurrency easily. The potential applications include, but not limited to, fast concurrent backup, checkpointing, debugging parallel programs, concurrent garbage collection, dynamic profilers, fast copy-on-write, and in-memory databases.

To support low-overhead snapshots, MShot uses hardware to build the snapshot image gradually. Our key observation to reduce the hardware cost is that the hardware components for MShot are found in other architectural proposals such as transactional memory systems (TM) [1]. TM is a promising technology that uses hardware to support atomic execution without the need for manual synchronization in user code. The hardware cost of MShot is dramatically reduced by sharing hardware resources for data versioning in TM.

We prototyped three interesting modules: garbage collection (GC), dynamic profiling, and copy-on-write. Taking advantage of the fact that "once garbage, always garbage," the snapshot-based GC takes a memory snapshot and performs collection without any interference with the application (mutator) threads running in parallel. The snapshot-based profiler takes a snapshot of the stack for call-path profiling or of the whole memory for memory profiling, while the application makes further calls or memory updates. Copy-on-write is widely used for efficient data sharing but incurs significant overhead when the shared data need to be copied. Snapshot-on-write takes a snapshot of data to be copied and executes the copy operation as a background job on another core. The main benefit from using MShot in these applications is that it allowed us to use additional cores in the CMP to hide the overhead of these modules without complex code that manages their interactions with the main application.

3. REFERENCES

[1] L. Hammond, V. Wong, et al. Transactional Memory Coherence and Consistency. In *the Proc. of the 31st Intl. Symp. on Computer Architecture (ISCA)*, Munich, Germany, June 2004.

[2] P. Jayanti. An optimal multi-writer snapshot algorithm. In *STOC '05: Proc. of the 37th ACM symp. on Theory of computing*, 2005.

SPAA'08, June 14–16, 2008, Munich, Germany.
ACM 978-1-59593-973-9/08/06.