Contents lists available at ScienceDirect

J. Vis. Commun. Image R.

journal homepage: www.elsevier.com/locate/jvci

Fast and scalable lock methods for video coding on many-core architecture

Weizhi Xu [b,f], Hui Yu [c], Dianjie Lu [d], Fenglong Song [b], Da Wang [b], Xiaochun Ye [b], Songwei Pei [e], Dongrui Fan [b], Hongtao Xie [a,*]

[a] Institute of Information Engineering, Chinese Academy of Sciences, National Engineering Laboratory for Information Security Technologies, Beijing, China
[b] Institute of Microelectronics, Tsinghua University, Beijing, China
[c] Key Lab. of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[d] School of Information Science and Engineering, Shandong Normal University, Jinan, China
[e] Department of Computer Science and Technology, Beijing University of Chemical Technology, Beijing, China
[f] State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

ARTICLE INFO

Article history:
Received 10 April 2014
Accepted 5 June 2014
Available online 25 June 2014

Keywords:
Many-core
Hardware lock
Centralized lock
Distributed lock
Micro-benchmarks
Godson-T
Software lock
Single-core processor

ABSTRACT

Many-core processors are good candidates for speeding up video coding because the parallelism of these applications can be exploited more efficiently by the many-core architecture. Lock methods are important for many-core architecture to ensure correct execution of the program and communication between threads on chip. The efficiency of lock method is critical to overall performance of chipped many-core processor. In this paper, we propose two types of hardware locks for on-chip many-core architecture, a centralized lock and a distributed lock. First, we design the architectures of centralized lock and distributed lock to implement the two hardware lock methods. Then, we evaluate the performance of the two hardware locks and a software lock by quantitative evaluation micro-benchmarks on a many-core processor simulator Godson-T. The experimental results show that the locks with dedicated hardware support have higher performance than the software lock, and the distributed hardware lock is more scalable than the centralized hardware lock.

1. Introduction

1.1. Background and motivation

High efficiency video coding (HEVC) [1,2] is the newest video coding standard. Compared with H.264/AVC, HEVC aims to provide a doubling in coding efficiency [3]. The price to be paid for higher video compression efficiency is higher computational complexity. HEVC encoders are expected to be several times more complex than H.264/AVC encoders [4]. Video coding may be restricted in mobile computation because of its high complexity [5–7]. As a result, it is important to accelerate video coding.

Traditional single-core processor depends on instruction level techniques to improve the performance, such as super-scalar and pipeline. As the frequency of the processor becomes higher, some problems of the single-core architecture rise up, such as power and heat dissipation. As a result, multi-core/many-core processors are proposed to solve the above problems through exploiting the parallelism in the applications [8]. Multiple/many simple cores are integrated on chip instead of one complex core to reduce power and heat dissipation without undermining overall performance. Many-core processors are good candidates for speeding up video coding because the parallelism of these applications can be exploited more efficiently by the many-core architecture [5–7].

Locks can ensure that the access of the shared memory is exclusive among different threads of many-core architecture, which makes the parallel program execute correctly. The design of locks on many-core architecture is very important because it can affect the performance of the parallel programs greatly.

For traditional multi-core architecture, there are many researches on locks, which can be classified into two categories, software locks and hardware locks. The disadvantages of software locks are high overhead of synchronization, poor scalability, large storage requirement and so on. For example, Test&Set lock [9] is a kind of software lock which demands that all the threads using the same lock execute the Test&Set instruction repeatedly. Test&Set lock can introduce a lot of memory and network operations and its scalability is poor when the number of threads increases. When there are hundreds or thousands of cores on chip for a

* Corresponding author.
E-mail address: xiehongtao@iie.ac.cn (H. Xie).

http://dx.doi.org/10.1016/j.jvcir.2014.06.009
1047-3203/© 2014 Elsevier Inc. All rights reserved.

many-core architecture, software locks cannot meet the performance demands of many-core processor, and a synchronization wall forms.

Although software locks are more flexible than hardware locks, an on-chip hardware lock can make good use of the fast communication on chip and improve the efficiency of locks, which can affect the speed of the whole program. A fine-grain hardware lock method is proposed on many-core processor Cyclops-64 with a synchronization state buffer (SSB) on chip [10]. However, coarse-grain locks for many-core processor are not studied in previous work.

Therefore, we propose two hardware coarse-grain locks for many-core architecture, a centralized lock and a distributed lock, with dedicated on-chip hardware supports. We evaluated the two hardware locks and a software lock on many-core architecture. Experimental results show that hardware support can achieve much higher performance than software lock, and the distributed lock is more scalable than the centralized lock.

1.2. Related works

Locks provide a way to visit the shared data exclusively among different threads. Different lock methods have different processing overhead, storage overhead and impact on network. Software locks are often implemented by the primitive read-modify-write, such as TAS, FAA, SWAP, CMPXCHG and LL/SC.

Test&Set lock [9] needs to repeatedly visit the flag, which leads to plenty of memory and network operations. When the number of threads increases, the scalability of Test&Set lock is poor. Several methods are proposed to improve the performance of Test&Set lock, such as Test&Set Lock with Backoff and TEST&TEST&SET. There are also many other lock methods, such as Ticket Lock [9], Array-Based Lock [13,14,15], QOLB (Queue on Lock Bit) [16,17], Lock Box [18], Lock Cache [19], Distributed lock based on bus [20]. Fine-grain locks or synchronization methods are also proposed, such as Full/Empty bit [21] and SSB [22]. Load-linked (LL) and store-conditional (SC) [11,12] is a pair of instructions which can be used to visit the shared data correctly. Transaction memory [22] uses a lock-free method to visit the shared data.

This paper is organized as follows. After introduction, we introduce a centralized lock method for many-core architecture in Section 2. We propose a distributed lock method for many-core architecture in Section 3. In Section 4, we analyze the experimental results for the three lock methods. Conclusions are given in Section 5.

2. A centralized lock method for many-core architecture

In this section, we introduce a centralized lock method for the on-chip many-core processor, which is implemented by a dedicated hardware called centralized lock manager (CLM). As shown in Fig. 1, Core1, 2, 3 and 4 are the on-chip processor cores. Cores and CLM are connected by routers and on-chip network.

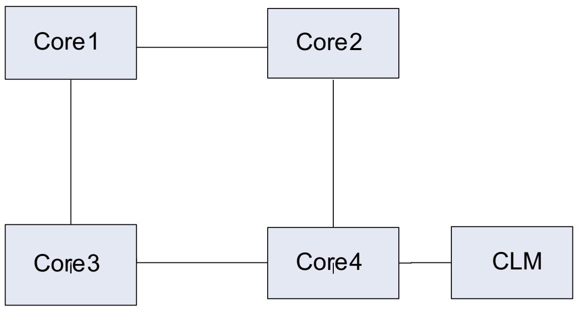

Fig. 1. A centralized hardware lock method for many-core architecture.

CLM organizes the lock requests from the cores as a FIFO queue. When a core sends an acquire-lock request, the request is sent to CLM through on-chip network. When CLM receives the request, it allocates an item in the queue for this request (Fig. 2). If the requested lock is in use by other cores, the request is put into the queue and maintained by CLM. When a core releases a lock, the release-lock request is sent to CLM which will send an acknowledgment message to the first waiting core in the queue. One advantage of CLM is that the core waiting for a lock is queued in CLM instead of repeatedly accessing the shared memory, which avoids the overhead of visiting the off-chip memory and congestion of network.

3. A distributed lock method for many-core architecture

We design a simple lock manager (LM) for each core to form the distributed lock manager (DLM, Fig. 3). Each LM is the home of a lock, and the number of LMs is the same as the number of cores. So the number of locks which DLM can support is also the same as the number of cores. A home LM is the LM which maintains the according lock. A local LM is the LM which is attached to the according core. For example, LM1 is the local LM for Core1 and is also the home LM for Lock1. A local core is the core which the LM is attached to. There are mainly two items in an LM as follows.

(1) *TAIL:* The ID of the last core which requests the home lock. The home lock is the lock maintained by the local LM. For example, LM1 maintains Lock1, LM2 maintains Lock2, and so on.
(2) *NEXT:* The ID of the core which is waiting for the lock used by the local core.

DLM organizes the locks as a distributed FIFO queue. When a core sends the acquire-lock request which contains the home LM ID of the lock, the request is sent to the home LM through on-chip network. After receiving the request, the home LM checks its TAIL. If TAIL is NULL, which means that the lock is not used by other cores, an acknowledgment is sent to the core. At the same time, the core ID in the request message is recorded in TAIL, which means the core is at the tail of the distributed queue. If TAIL is not NULL, which means that there is another core using the lock, the request message is forwarded to the LM whose ID is recorded in TAIL. At the same time, the core ID in the request message is recorded in TAIL (Fig. 4). When the LM receives the forwarded message from home LM, it updates its NEXT to the core ID in the message.

When a core releases a lock, it first sends a release message to its local LM. If there is a waiter in NEXT, an acknowledgment is sent to the waiting core. If there is no waiter in NEXT, a release message is sent to the home LM of the lock (Fig. 5).

When the home LM receives the release message, it will check whether the core ID in TAIL is the same as that in the message. A same core ID means that there are no other waiters for this lock, and TAIL is set to NULL. If different, an acknowledgment is sent to the core whose ID is in TAIL (Fig. 6).

Fig. 7 describes a working scenario of DLM, the six steps in the figure are explained as follows.

(1) Core2 sends an acquire-lock message to LM1 to acquire Lock1.
(2) Because Lock1 is free, LM1 sends an acknowledgment to Core2 and sets its TAIL to Core2.
(3) Core3 sends an acquire-lock message to LM1 to acquire Lock1.