阅读报告 1: Shared Memory Consistency Models: A Tutorial

黄科乐 201828013229084

9-27

1 Introduction

1 The goal of this tutorial article is to provide a description of sequential consistency and other more relaxed memory consistency models in a way that would be understandable to most computer professionals. Beacuse the authors think it is important to make sure the correctly and widely used by programmers. To achieve this goal, the authors describe both system-centric emphasis and programmer-centric view of relaxed consistency models.

1.The rest of this article is begining with a short note of who should be concerned with the memory consistency model of a system.

2.Then the authors describe the implications of both view, hardware and compiler implementations, on sequential consistency.

3.Then describe several relaxed memory consistency models.

4.The last part is concerned with programmer-centric view of relaxed memory consistency models.

2 Memory Consistency Models - Who Should Care?

The effect of the memory consistency model is pervasive in a shared memeory system. The model affects every level such as programmability, performance, portability. Not even the machine code interface, but although the high level language interface. Therefore, both the programmers who use the high level and machine language should concern concern this model.

3 Memory Semantics in Uniprocessor Systems

Most high-level uniprocessor language present simple sequential semantics for memory operations, which allow the programmer to assume that all memory operations will occur one at a time in the sequential order specified by the program. Fortunately, the compiler and hardware optimizations can support it well.

4 Understanding Sequential Consistency

Each processor issues memory operations in program order and the switch provides the global serialization among all memory operations.

5 Implementing Sequential Consistency

5.1 Architectures Without Caches

5.1.1 Write Buffers with Bypassing Capability

The first optimization the authors consider illustrates the importance of maintaining program order between a write and a following read operation. On a write, a processor simply inserts the write operation into the write buffer and proceeds without waiting for the write to complete. This bypassing is allowed as long as the read address does not match the address of any of the buffered writes.

5.1.2 Overlapping Write Operations

The second optimization illustrates the importance of maintainning program order between two write operations. The key difference compared to the previous example is that multiple write operations issued by the same processor may be simultaneously serviced by different memory modules.

5.1.3 Non-Blocking Read Operations

The third optimization illustrates the importance of maintaining program order between a read and a following read or write operation.

5.2 Architectures With Caches