This paper presents a new mechanism to reduce memory overhead by identifying the unnecessary memory ordering operations.

**Strengths:**
1. It points out that previous algorithms inserting memory fences indiscriminately during compilation are based conservative program analysis, and can have a considerable performance impact.
2. Compared to the previous processor-centric architectures, the mechanisms provided in this paper allow one processor to control the ordering of operations performed by another processor, which is not processor-centric.
3. It summaries the characters of synchronization and memory ordering operations in lock intensive Java workloads and demonstrate that a lot of memory ordering operations occur superfluously.
4. The CMO model eliminates most inefficiencies that caused by redundant memory ordering operations, and hence improve the performance.

**Weaknesses:**
1. Implementation of CMO need to go through the program and find those unnecessary memory orderings, it may cause performance degradation to the system.
2. The performance improvements in the software prototype are hindered by the high cost of remote memory ordering.
3. The mechanism provided in this paper address only specific classes of synchronization stalls. It may not work well for those programs with few lock intensive operations.
4. The mechanisms depends on hardware and software at the same time, so when the hardware platform changes, the implementation may also need some modifications.