The Effect of Contention on the Scalability of Page-Based Software Shared Memory Systems

Eyal de Lara, Y. Charlie Hu, Honghui Lu, Alan L. Cox, and Willy Zwaenepoel

1 Department of Electrical and Computer Engineering
2 Department of Computer Science
Rice University, Houston TX 77005, USA

Abstract. In this paper, we examine the causes and effects of contention for shared data access in parallel programs running on a software distributed shared memory (DSM) system. Specifically, we experiment on two widely-used, page-based protocols, Princeton’s home-based lazy release consistency (HLRC) and TreadMarks. For most of our programs, these protocols were equally affected by latency increases caused by contention and achieved similar performance. Where they differ significantly, HLRC’s ability to manually eliminate load imbalance was the largest factor accounting for the difference. To quantify the effects of contention we either modified the application to eliminate the cause of the contention or modified the underlying protocol to efficiently handle it. Overall, we find that contention has profound effects on performance: eliminating contention reduced execution time by 64% in the most extreme case, even at the relatively modest scale of 32 nodes that we consider in this paper.

1 Introduction

In this paper, we examine the causes and effects of contention for shared data access in parallel programs running on a software distributed shared memory (DSM) system. Specifically, we analyze the execution of a representative set of programs, each exhibiting a particular access pattern that causes contention. In each of these cases, to quantify the effects of contention on performance, we have either modified the application to eliminate the cause of the contention or modified the underlying protocol to efficiently handle that particular access pattern. Overall, we find that contention has profound effects on performance: eliminating contention reduced execution time by 64% in the most extreme case, even at the relatively modest scale of 32 nodes that we consider in this paper.

Our experiments are performed on a network of thirty-two single-processor nodes using both Princeton’s home-based (HLRC) protocol and Rice’s TreadMarks (Tmk) protocol. Both are widely-used, page-based, multiple-writer protocols implementing Lazy Release Consistency (LRC). From our experiments, we derive three specific conclusions.
First, in comparing the results on 8 nodes to 32 nodes, we find that the effects of increasing contention for shared data are evident in the increasing latency to retrieve data. In the worst case, latency increased by 245%.

Second, in one case, the Barnes-Hut program from the SPLASH benchmark suite, the HLRC protocol handles contention more effectively than the Tmk protocol. It more evenly distributes the number of messages handled by each node.

Third, the distribution of the number of messages handled by each node is no less important than the total number of messages. For example, in Barnes-Hut, eliminating the message load imbalance under Tmk (through protocol modifications), brought Tmk’s performance to the same level as HLRC’s, even though Tmk sends 12 times more messages than HLRC.

The rest of this paper is organized as follows. Section 2 provides an overview of TreadMarks and HLRC’s multiple-writer protocols. Section 3 discusses the sources of contention in greater detail and defines the notion of protocol load imbalance. Section 4 details the experimental platform that we used and the programs that we ran on it. Section 5 presents the results of our evaluation. Section 6 compares our results to related work in the area. Finally, Section 7 summarizes our conclusions.

2 Background

2.1 Lazy Release Consistency

The TreadMarks (Tmk) protocol and the Princeton home-based (HLRC) protocol are multiple-writer implementations of lazy release consistency (LRC).

Lazy release consistency (LRC) is an algorithm that implements the release consistency (RC) memory model. RC is a relaxed memory model in which ordinary accesses are distinguished from synchronization accesses. There are two types of synchronization access: acquire and release. Roughly speaking, acquire and release accesses are used to implement the corresponding operations on a lock. In general, a synchronization mechanism must perform an acquire access before entering a critical section and a release access before exiting. Essentially, the benefit of RC is that it allows the effects of ordinary memory accesses to be delayed until a subsequent release access by the same processor.

The LRC algorithm further delays the propagation of modifications to a processor until that processor executes an acquire. Specifically, LRC insures that the memory seen by the processor after an acquire is consistent with the happened-before-1 partial order, which is the union of the total processor order of the memory accesses on each individual processor and the partial order of release-acquire access pairs.

2.2 TreadMarks and Home-Based LRC

The main difference between Tmk and HLRC is in the location where updates are kept and in the way that a processor validates its copy of a page. In Tmk,