Non-intrusive detection of synchronization errors using execution replay.

Michiel Ronsse* and Koen De Bosschere†
Department of Electronics and Information Systems
Ghent University, Belgium

Abstract. This paper presents a practical solution for detecting synchronization errors in parallel programs. These errors are: a lack of synchronization resulting in data races, conflicting synchronization resulting in deadlock and redundant synchronization resulting in a performance penalty.

The solution consists of a combination of RecPlay, an efficient execution replay mechanism combined with automatic on-the-fly data race detection, deadlock detection and the detection of redundant synchronization during a replayed execution. The detection of data races, deadlocks and redundant synchronization normally introduces an important overhead during an execution, possibly altering the execution. However, by performing these extensive operations during a replayed and therefore unaltered execution there is almost no probe effect. Furthermore, the memory consumption during the data race detection is limited through the use of multilevel bitmaps and snooped matrix clocks. As the record phase of RecPlay is highly efficient, there is no need to switch it off, hereby eliminating the possibility of Heisenbugs because tracing can be left on all the time.

Keywords: parallel programming, cyclic debugging, data race, deadlock, redundant synchronization

1. Introduction

The never ending urge for faster and more robust computers, combined with the existence of cheap processors causes a proliferation of inexpensive multiprocessor machines. Multithreaded applications are needed to exploit the full processing power of these machines, causing a widespread use of parallel applications. Even most contemporary applications that are not CPU-intensive are multithreaded because the multithreaded paradigm makes it easier to develop servers, applications with an MDI (*multiple document interface*) such as Windows programs, etc.

However, developing multithreaded programs for these machines is not easy as it is harder to get a good view on the state of a parallel program. This is caused by the fact that there are a number of threads[1]

* ronsse@elis.rug.ac.be
† kdb@elis.rug.ac.be

[1] In this paper we will consider an execution of a parallel program as being a process consisting of N threads executing on a machine with N processors.

© 2003 *Kluwer Academic Publishers. Printed in the Netherlands.*

The remainder of this paper is not included as this paper is copyrighted material. If you wish to obtain an electronic version of this paper, please send an email to bib@elis.rug.ac.be with a request for publication P102.004.pdf.