ELSEVIER

Simulation Practice and Theory 8 (2001) 511–527

SIMULATION PRACTICE AND THEORY

www.elsevier.com/locate/simpra

Performance analysis of automatic lookahead generation by control flow graph: some experiments

Behrouz Zarei *, Mike Pidd

Management Science Department, Lancaster University, Lancaster LA1 4YW, UK

Received 25 April 2000; received in revised form 1 December 2000

Abstract

The performance of parallel discrete event models is highly dependent on lookahead, particularly when a conservative algorithm is employed. Unfortunately lookahead is known to be problem-dependent, which restricts the implementations of conservative algorithms. This paper uses a simple queuing network to show how this lookahead affects performance and discusses various techniques for automatic generation of lookahead using control flow graphs (CFGs). These methods are tested on the queuing network simulation running on a CRAY T3E 1200E. Results indicate that the automatic lookahead techniques, though requiring some time to compute, perform as well as the best manually extracted lookahead injected into the parallel program.

Keywords: Lookahead; Automatic lookahead generation; Parallel simulation; Control flow graph; Queuing network; Performance analysis

1. Introduction

In parallel discrete event simulation PDES [1], the model to be simulated is decomposed into physical processes that are modelled as simulation objects and assigned to logical processes (LPs). The simulator is composed of a set of concurrently executing LPs running on distributed physical processors. The LPs communicate by exchanging two types of time-stamped messages: real messages, which are events in one processor and must be scheduled in another processor; and null messages, which are used to synchronise the processors. In order to

* Corresponding author.
E-mail address: b.zarei@lancaster.ac.uk (B. Zarei).

0928-4869/01/$ - see front matter
PII: S0928-4869(01)00031-3

maintain causality, the LPs must process messages in strictly non-decreasing time-stamp order. There are two basic synchronisation protocols used to ensure that this condition is not violated: the first conservative and the second optimistic. Conservative protocols, such as Chandy-Misra [2] and Bryant [3] (often known as CMB) avoid causality errors, while optimistic protocols, such as Time Warp [4], allow causality errors to occur, but implement some recovery mechanism.

Conservative protocols were the first distributed simulation mechanism. The basic problem conservative mechanisms must address is the determination of safe events. The conservative process must first determine that it is impossible to receive another event with a lower time stamp than the event it is currently trying to execute. Such events are assumed to be safe and can be executed without violation of system causality. Processes containing no safe events must be blocked; this can lead to a deadlock situation if no appropriate precautions are taken. Many optimisations of the basic conservative mechanism can be found in the literature. For instance, instead of creating null messages a mechanism can be used to detect when the simulation is deadlocked and another mechanism to break the deadlock [5] or send null messages on a demand basis, i.e. whenever a process is about to become deadlocked, it requests the next message from the sender process [6]. This approach reduces the null message traffic, though a longer delay may be required, due to transmission of two messages.

In an optimistic protocol, each LP operates as a distinct discrete event simulator, maintaining input and output event lists, a state queue, and a local simulation time (called local virtual time or LVT). Each LP processes events optimistically and moves ahead in LVT. As each LP simulates asynchronously, it is possible for an LP to receive an event from the past, termed a straggler, which violates the causality constraints of the events in the simulation. On receipt of a straggler message, the LP must rollback to undo events that have been wrongly processed. Rollback involves two steps (i) restoring the state to a time preceding the time stamp of the straggler and (ii) cancelling any output event messages that were erroneously sent. After rollback, the events are re-executed in the proper order. Optimisations of the basic mechanism have been proposed to overcome the drawback of state-saving. For instance, processes could stop the immediate sending of the anti-messages for any roll back computation and instead, could wait to see if the re-execution of the computation regenerates the same messages. If the same message is regenerated, there is no need to cancel the message [7]. Also combining optimistic and conservative protocols is employed in order to take advantage of less memory usage property of the conservative algorithm. Ideally such a protocol monitors the parallel simulation and estimates the trade-off between the conservatism cost (i.e. blocking cost) and the optimism cost (i.e. state-saving and rollback) and accordingly adjusts the protocol control parameters [8].

Fujimoto [9] outlines the main concerns of PDES and Ferscha [10] comprehensively demonstrates the synchronisation layer of the PDES. The emergence of complicated applications (such as wireless networks [11], telecommunication systems [12], manufacturing [13], and cache design [14]) and widely available parallel and distributed platforms encourages the use of PDES as a tool for decision-makers.