The 47th IEEE International Midwest
Symposium on Circuits and Systems

# A Runtime Mechanism for Detection of Artificial Deadlocks in Process Networks

Bharath.N and S.K.Nandy
CAD Laboratory, Supercomputer Education and Research Centre,
Indian Institute of Science,Bangalore - 560012, India.
email : [bharath@rishi.serc, nandy@serc].iisc.ernet.in

***Abstract*—Kahn Process Network (KPN) is a popular model of computation for describing streaming applications arising in media and signal processing. KPN is a collection of sequential processes that communicate through unidirectional unbounded fifos. The illusion of unbounded fifos has to be met with finite memory in real implementations. This could potentially lead to a violation of Kahn semantics, thereby resulting in "artificial deadlocks". In this paper we address this issue and present a runtime mechanism for early detection and resolution of such artificial deadlocks.**

## I. INTRODUCTION

Kahn Process Network [1] is a collection of sequential processes that communicate through unidirectional unbounded fifos. These fifos buffer data streams between processes and facilitate parallel computation. KPNs are widely used to model media processing applications [4]. KPNs explicitly specify parallelism and communication inherent in an application and are hence well suited for multiprocessor implementations [7].

Fig.1 represents an example KPN. Here, nodes denote the processes of the KPN and the edges represent the fifos between the processes. The direction of the edge denotes the direction of communication between the processes.

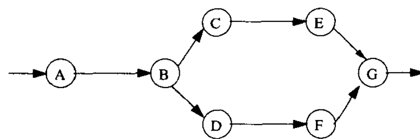


Fig. 1. Example KPN.

Execution of a KPN involves the execution of constituent sequential processes in conformance to Kahn semantics [1]. Processes communicate with each other only through reads and writes to the connecting fifo. A process blocks if it tries to read data from an empty fifo or an empty input from the environment. Kahn showed that the order of execution of the processes is irrelevant; i.e. given some input, any order of execution of the processes in accordance with blocking read semantics produces the same output. Blocking reads act as simple synchronization primitives which guarantee a consistent parallel execution.

An implementation of the KPN has to execute using finite memory. Following [7] we use the term process network (PN) to refer to a network of processes with bounds on the fifo sizes. Two issues arise; (a) how to find an efficient schedule for the processes to execute and (b) how to allocate memory to the conceptually unbounded fifos, i.e. how to obtain an illusion of unbounded fifo for the execution of the KPN in bounded memory. These two issues are related. The amount of memory allocated to the fifos can possibly influence the execution order of the processes of the KPN [7]. Also, artificial deadlocks can occur in the execution of KPNs when sufficient memory is not allocated to the fifos; all processes in the KPN block and at least one process is blocked on write to a full fifo [5]. Fig.2 shows an artificial deadlock in the PN derived from the KPN of Fig.1. The label on the arc denotes the fifo size. Processes are blocked either on a write to a full fifo, denoted by w; or a read to an empty fifo, denoted by r.

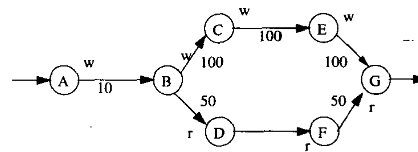


Fig. 2. A PN in artificial deadlock.

The rest of the paper is organized as follows. Section II discusses scheduling of KPNs. Section III discusses bounded scheduling of KPNs and the existing mechanisms to deal with artificial deadlocks. Section IV presents our runtime mechanism for detecting artificial deadlocks. Section V concludes.

## II. SCHEDULING OF KPNS

The problem of scheduling KPNs is to decide which process executes on which processor at each point in time [7]. The two approaches to scheduling are static scheduling and dynamic scheduling. In static scheduling the order of execution of processes is determined at compile time, whereas in dynamic scheduling this is determined at runtime. The problem of determining efficient static schedules for KPNs is known to be undecidable in general [6]. Consequently the focus in this paper is on dynamically scheduled KPNs.

In dynamic scheduling the scheduler chooses from a set of ready processes and schedules as many ready processes as there are available processors. Two approaches to dynamic scheduling are demand driven scheduling and data driven scheduling.

In demand driven scheduling the driving force is demand for data. Only data that is really needed is produced. The demand arises at the output processes of the KPN and propagates to the processes reading input from the environment. Initially the set of ready processes comprises the processes producing output. When a running processes attempts to read data from an empty fifo, it blocks. A new demand for data arises and the producer process for the data in demand now becomes ready. After the producer process produces the required data, it is no more a ready process.

In data driven scheduling, the driving force is the availability of data. Initially data is available at the inputs of the KPN

This work is partially funded by a grant from Philips Research Laboratories.

0-7803-8346-X/04/$20.00 ©2004 IEEE

and the set of ready processes comprises processes consuming data from the inputs. These processes produce data that causes the consumer process of this data to become ready. A running processes blocks when no more input data is available.

Both these dynamic scheduling approaches have disadvantages that render them unattractive. Data driven scheduling can lead to unbounded execution without ever producing output. Processes consuming input data are continuously ready as long as input data is available. The data produced by the processes consuming input can cause other processes to become ready. However a scheduling policy that repeatedly schedules the input processes can result in an unbounded execution. Demand-driven scheduling leads to many expensive context switches. Initially all fifos in the KPN are empty and a chain of context switches result when the demand propagates from the output to the input processes of the KPN.

Hybrid approaches combine data-driven and demand-driven scheduling. Kahn and MacQueen [2] extend demand-driven scheduling with anticipation coefficients. Each fifo "f" is assigned an anticipation coefficient ac(f). A producer of data demanded on fifo f can produce only an additional ac(f) amount of data after the demand on fifo f has been fulfilled. This restricts the amount of data produced for which there is no demand.

In the following section we discuss a hybrid approach by Parks [5], This is a data-driven approach with bounds on fifo sizes and is implemented in Ptolemy [10].

## III. BOUNDED SCHEDULING OF KPNs

A KPN is strictly bounded if and only if any execution of the network requires bounded memory. It is bounded if and only if there is some execution that requires bounded memory. It is unbounded if and only if any execution requires unbounded memory. The question "given a KPN, is it strictly bounded" is undecidable [5]. The bounded scheduling problem of KPNs is to decide on a scheduling strategy that executes process networks using bounded memory whenever possible [5].

Parks' solution [5] to the bounded scheduling problem allocates some initial capacity to each of the fifos in the KPN. Writes to full fifos are made blocking till space is again available on the fifo. This gives an intermediate form of data driven and demand driven scheduling where a process produces data until the fifo it is writing to is full. The producer process resumes writing when the reading process (consumer at the other end of the fifo) has read enough data to create sufficient space on the fifo. Blocking writes can cause artificial deadlocks whereby all processes in the PN block and at least one process blocks on a full fifo. These deadlocks are artificial because such deadlocks cannot occur in a KPN with unbounded fifos. Artificial deadlocks can be resolved by allocating additional capacity to the fifo so that the process waiting on a write to a full fifo can proceed [5].

Parks' strategy to deal with artificial deadlocks is as follows: Execute the PN with some initial sizes till all processes in the PN block. If an artificial deadlock occurs, i.e. there is some process blocked on a write to a full fifo, then increase the smallest full fifo by one unit; this guarantees that all blocked writes in the PN are eventually removed [5]. In implementations of PNs based on Parks' scheduling algorithm [11][12], a special thread is used to detect and initiate the deadlock resolution procedure. This thread keeps a count of all current read-blocks and write-blocks. An artificial deadlock is detected when the total number of blocks equals the number of processes and there is some process that is blocked on write to a full fifo. This strategy requires some run-time overhead to keep track of the number of current read and write blocks. Each time a block occurs, the special thread is notified, the respective block counter is incremented, and a check is made to determine whether an artificial deadlock condition exists. Each time a read or write block is removed, the special thread is again notified, and the respective block counter is decremented.

Basten and Hoogerbrugge [7] show that the smallest full fifo in a PN is not necessarily the bottleneck fifo causing the artificial deadlock. Increasing the smallest full fifo will eventually resolve the bottleneck, but may lead to unnecessarily large fifos. They introduce the concept of causal chains to identify the bottleneck causing fifo in a process network. The cause of a read or a write block in a deadlocked process network is always a unique other blocked process. Consequently it is possible to track the entire chain of causes that lead to a blocked output process. The bottleneck fifo causing the artificial deadlock is the first fifo in a chain of full fifos. When a cycle of full fifos result there is insufficient space in the cycle, the combined fifos in the cycle form the bottleneck. In case of multiple bottlenecks in a single chain of blocked processes, the smallest fifo among these bottleneck fifos is enlarged. Applying this to the PN of Fig. 2, if the bottleneck fifo is the fifo between processes E and G, Parks' approach would require the fifo between processes A and B to be resized before the size of the bottleneck fifo is increased.

Geilen and Basten [8] investigate the requirements for a runtime scheduler to achieve a correct and bounded execution of process networks. They distinguish between local and global deadlocks. A process network is in global deadlock if none of the processes can make progress; if some processes cannot progress and the rest of the network cannot initiate their progress they are in a local deadlock. Consider the PN of Fig. 3 from [9]. Processes A, D, C, E and F are blocked as shown in the figure. The output process F is blocked due to an artificial local deadlock. However processes B, G, H and I are running and output process I can continue to produce data indefinitely.

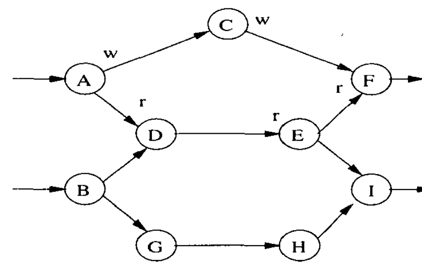


Fig. 3. A PN in local deadlock.

In Parks' approach, artificial deadlock is said to have occurred when all processes block, some of them on a write to a full fifo. Parks' method detects and responds to global artificial deadlocks only. This approach ensures that execution does not terminate due to artificial deadlocks, but does not guarantee output completeness (production of all output required by Kahn semantics).

Under the assumption of effective KPNs (every token written

to a channel is eventually read), Geilen and Basten [8] show that chain of dependencies leading to a local deadlock is cyclic. Their scheduling strategy is as follows: execute the process network in a data driven fashion until an artificial local deadlock occurs; resolve artificial local deadlocks by increasing the smallest full fifo by a finite amount. It is not clear how the cyclic chain of dependencies leading to an artificial deadlock is detected. Also the smallest full fifo on the local deadlock is enlarged.

The above solutions involve simulation and profiling of target applications at design time to decide the size of fifos in a process network. There is no guarantee that all artificial deadlocks are avoided. Inputs other than those used for profiling may lead to artificial deadlocks. Hence in the final product artificial deadlocks still need to be resolved at runtime [7]. In the next section we present a runtime mechanism to detect and resolve artificial deadlocks.

## IV. RUNTIME MECHANISM

Consider the example PN of Fig 4. Process A and Process B receive as input a number N that decides the number of tokens actually produced. If the fifo sizes were to be decided by profiling using sample inputs, it is possible that a value of "N" other than those used in the simulations needs larger fifo sizes. This insufficient memory allocation to fifos could cause an artificial deadlock [5]. This example clearly demonstrates the need for a runtime mechanism to detect and resolve artificial deadlocks.

Our approach to detection of artificial deadlocks consists of a methodology to identify all the possible deadlocking patterns for a process network and runtime monitoring of the execution of process network for the occurrence of such patterns.

The methodology to identify the deadlocking patterns builds upon the result of [8] that the chain of causes leading to an artificial deadlock is cyclic. We present two key observations regarding artificial deadlocks in process networks.

The cyclic chain of causes, henceforth called causal cycles, leading to local deadlocks correspond to the cycles in the (undirected) graph of the process network (processes form the nodes and fifos form the edges). Given a PN graph, the cycles in it that could potentially lead to causal cycles during the actual execution can be enumerated. In the example PN of Fig.4, the processes A, C, G, F, D and B, D, F, H, E form the two such cycles.

We observe that causal cycles leading to local deadlocks have one of the patterns shown in Fig.5. This observation can be intuitively explained as follows: every blocked process "A" has a unique other blocked process "B" as the cause [7]. Hence a process A can be blocked waiting for (a) another process B to read from the connecting fifo; here the process A is blocked on a write to a full channel or (b) another process B to write to the connecting fifo; here the process A is blocked on a read from an empty channel.

Consider Fig 5(a), process A is blocked on a read waiting for B to produce data, B is blocked on a read waiting for C, and so on till process X which is blocked on a write waiting for P to consume data, P is blocked on a write waiting for Q, and so on till a process R blocked on a write waiting for A. This completes the causal cycle. Fig 5(b) is explained similarly. In Fig 5(c) and 5(d) we have a set of processes, each blocked on a write and cyclically waiting for another process in the same

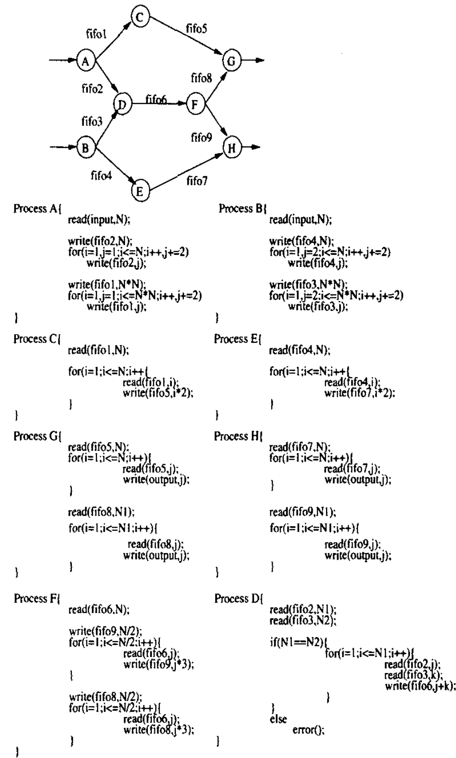


Fig. 4. Example PN.

set.

To summarize, the status of processes in a causal cycle as observed from processes X or A is one of the following (a) $w^+r^+$ (b) $r^+w^+$ (c) $ww^+$. We designate such processes as the monitor processes. Note that $rr^+$ is not an artificial deadlock (by definition).

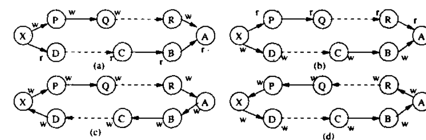


Fig. 5. Possible causal patterns in a PN.

The implication of the above observations is that the possible causal cycle patterns for a given PN can be enumerated at compile time from the PN graph. For the example PN of Fig.4, the possible causal cycle patterns are shown in Fig.6. For Fig 6(a) and 6(c) the monitor processes are B, H and for Fig 6(b) and 6(d), A, G are the monitor processes.

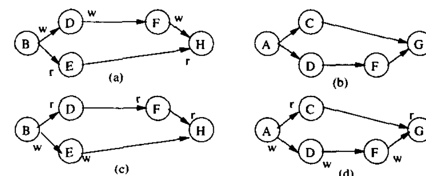


Fig. 6. Possible causal patterns in example PN (Fig 4).

It is not known which of these possible causal chain patterns will actually lead to an artificial local deadlock. This is dependent on the input and the allocated fifo capacity. Once the possible causal cycle patterns are known, deadlock detection reduces to monitoring the status of the PN for these causal patterns. To resolve the deadlock we increase the size of the smallest full fifo in the causal cycle.

Monitoring the status of the PN for the occurrence of a causal cycle can be implemented using the finite state machine (FSM) of Fig. 7. This FSM accepts strings belonging to $r^+w^+$, $w^+r^+$, $ww^+$. A in the initial state; C, D, E are the accepting states of the FSM. When a monitor process blocks on a read or a write, the status of the processes in its causal cycle are fed as inputs to the FSM; r for a read blocked process, w for a write blocked process and nb for a running process. If the FSM accepts this string, then a causal cycle has been detected and an artificial deadlock has occurred.

We have implemented our deadlock detection mechanism in YAPI [4]. YAPI is a C++ runtime library to execute applications modeled using KPNs. Our simulations verify that the proposed mechanism correctly detects and resolves artificial local deadlocks.

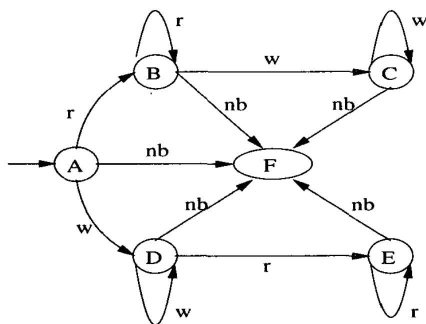


Fig. 7. fi nite state machine to detect artifi cial deadlocks - A is the initial state; C, D, E are the accepting states.

## V. CONCLUSION

In this paper, we have proposed and implemented a runtime mechanism to detect and resolve artificial deadlocks in Kahn Process Networks. Our method detects artificial local deadlocks as and when they get formed, leading to an early detection and resolution of artificial deadlocks in KPNs.


## ACKNOWLEDGMENT

We would like to thank Dr.Pradeep Desai,Dr.Narendranath Udupa and the DOPA team at Philips Research Bangalore for many helpful discussions. We would also like to acknowledge their help with the YAPI libraries and applications.


## REFERENCES


[1] G. Kahn. "The Semantics of a Simple Language for Parallel Programming".In J.L. Rosenfeld, editor, *Information Processing 74, Proceedings*, pages 471-475, Stockholm, Sweden, August 1974. North-Holland, Amsterdam, The Netherlands, 1974.

[2] G. Kahn and D.B. MacQueen. "Co routines and Networks of Parallel Processes".In B. Gilchrist, editor, *Information Processing 77, Proceedings*,pages 993-998, Toronto, Canada, August 1977. North-Holland,Amsterdam, The Netherlands, 1977.

[3] P. Stravers and J. Hoogerbrugge. "Homogeneous Multiprocessing and the Future of Silicon Design Paradigms".In *International Symposium on VLSI Technology, Systems, and Applications (VLSI-TSA), Proceedings*,Hsinchu, Taiwan, April 2001.

[4] E.A. de Kock et al. "YAPI: Application Modeling for Signal Processing Systems".In *37th. Design Automation Conference, Proceedings*,pages 402-405, Los Angeles, CA, June 2000. IEEE, 2000.

[5] T.M. Parks. "Bounded Scheduling of Process Networks".PhD thesis,University of California, EECS Dept., Berkeley, CA, December 1995. Technical Memorandum UCB/ERL M95/105.

[6] J.T. Buck. "Scheduling Dynamic Dataflow Graphs with Bounded Memory using the Token Flow Model".PhD thesis,PhD thesis, University of California, EECS Dept., Berkeley, CA, 1993. Technical Memorandum UCB/ERL M93/69.

[7] T. Basten and J. Hoogerbrugge. "Effi cient execution of process networks". In A. Chalmers, M. Mirmehdi, and H. Muller, editors, *Proc. of Communicating Process Architectures 2001, Bristol, UK, September 2001*, pages 1-14. IOS Press, 2001.

[8] M.C.W. Geilen and T. Basten. "Requirements on the Execution of Kahn Process Networks". In P. Degano, *Programming Languages and Systems, 12th European Symposium on Programming, ESOP 2003, Proceedings*, pages 319-334.

[9] M.C.W. Geilen and T. Basten. "Requirements on the Execution of Kahn Process Networks". In P. Degano, *Programming Languages and Systems, 12th European Symposium on Programming, ESOP 2003, presentation slides*, http://www.ics.ele.tue.nl/ ericurus/publications/esop03.pps.

[10] E.A. Lee. "Overview of the Ptolemy Project".Technical Memorandum UCB/ERL M01/11, University of California, EECS Dept., Berkeley, CA, March 2001.

[11] M.Goel. "Process Networks in Ptolemy II".Masters thesis, University of California, EECS Dept., Berkeley, CA, 1998. Technical Memorandum UCB/ERL M98/69.

[12] P.Laramie, R. S. Stevens, and M. Wan. "Kahn process networks in Java".ee290n class project report, University of California, EECS Dept., Berkeley, 1996.