

Rapport LIFL # 2003-06

# Design of a Real-Time Scheduler for Kahn Process Networks on Multiprocessor Systems

Javed DULLOO

Javed.Dulloo@lifl.fr

Philippe MARQUET

Philippe.Marquet@lifl.fr

Laboratoire d'informatique fondamentale de Lille  
Université des sciences et technologies de Lille  
France

September, 2003

## Abstract

High-throughput real-time systems require non-standard and costly hardware and software solutions. Modern workstation can represent a credible alternative to develop real-time intensive signal processing applications. Furthermore, the programming model of Kahn Process Networks (KPN) corresponds completely to this kind of applications and fits perfectly on multiprocessor systems. We present a new activation strategy of processes in a KPN which considerably improves the existing techniques. This new activation order takes into account both deadlock detection and resolution with time constraints. With this algorithm, we do not need to wait until the entire execution has deadlocked to remove bottleneck. Moreover, we have an optimized allocation memory mechanism and we can bound the number of processes context switches.

**Keywords** Kahn Process Networks scheduling, multiprocessor architecture, real-time intensive signal processing.

## 1 Introduction

Today, high-throughput real-time systems require non-standard and costly hardware and software solutions. High-performance and high-throughput applications (intensive signal processing) require on the order of a billion (giga) floating-point operations per second (GFLOPS) and 100MB/s of data I/O [10]. Their requirements had been for years the justification for the use of specific technology at all levels: specialized DSP processors, dedicated operating systems lacking the support of standard APIs and requiring custom applications, and also specialized cluster interconnects.

These systems are not sold in high volumes and the cost to develop any application could be very expensive. With the advances in commercial workstations, problems requiring computation on the order of GFLOPS can be solved. Intel's recently introduced Itanium processor, an architecture which offers the theoretical computational power needed by intensive signal processing applications, and being backed by the IA-32 commercial success, it could

become a new industrial standard. Modern workstations offer several advantage to use it: hardware development costs are sharing with server market, software development costs are reduced because operating systems and tools offer portability, upgradability and maintainability. Although with components off the shelf (COTS) we benefit of the advantage of distributed hardware development, it is not the case with implementation: integration and testing are difficult and costly. Often, COTS are a set of distributed sub-systems. We can not do load balancing with these distributed sub-systems, and thereby, to develop an application, the solution is not always optimal.

This paper is part of a framework which aims to show the attraction of modern workstation (with an open source operating system) in real-time intensive signal processing domain. This advantage from performance and cost point of view, is found in workstation use like the development platform and also the target architecture. A first example [1, 2] can be read in which data-intensive sonar beamforming algorithms in real-time context have been implemented on multiprocessor Unix workstation.

Currently, the most general model of computation for signal processing (dataflow paradigm) is Kahn Process Network [8, 9]. Its strong point is that the scheduling algorithm does not affect its functional behavior. This reliable formal design methodology enables us to benefit advantages of multiprocessor systems: at task-level, parallelism and communication, in an application, are explicit. Other dataflow graphs [13] (computation graphs, synchronous dataflow and boolean dataflow) offer good models to describe asynchronous processing, however their definition imply the management of the production rate which could compel performance (for lazy streams). This advantage which becomes a disadvantage because no activation rule is defined. If we want to bound response time or to reduce latencies, it is necessary to create an activation rule. This can be done easily without changing semantics because the model is intrinsically deterministic, *i.e.* the execution behavior of KPN depends only on its topology.

**Contribution** We define a new activation strategy of processes in a KPN which considerably improves the existing techniques. The main contributions are that, with this algorithm, on a side, we take into account both deadlock detection and resolution with time constraints, and an another side, we do not need to wait until the entire execution has deadlocked to remove bottleneck. Finally, we propose an implementation of this algorithm in the frame of the ARTiS run-time [11].

**Organization** The next section introduces the issue of scheduling in real-time context. Section 3 defines an efficient scheduling strategy. The section which will follow, discusses the challenge related with implementing the previous scheduler on multiprocessor architecture and in section 5, we propose an implementation of this algorithm. Section 6 concludes.

## 2 KPN Scheduling

Kahn Process Networks (KPN) provide a set of processes and only first-in-first-out (unbounded) buffers, with blocking reads and non-blocking writes, as means of communication. Unfortunately, the issue of time constraints is not considered in the KPN definition. Furthermore, the programming model of KPNs totally is asynchronous. Thereby, find a correct scheduler becomes problematic in a real-time context, because we must bound the Worst

Case Execution Time (WCET). The KPN semantics does not offer any mechanism to determine this bound. Nevertheless, the KPN model guarantees that any order of execution of process leads to the same output stream, it is intrinsically deterministic.

In the KPN model, processes are connected via theoretically unbounded FIFOs. According to the corollary defined in [12, 5], *the problem of deciding whether a Kahn Process Network can be scheduled with bounded memory is undecidable*. Their implementation require bounded memory. Fixed FIFO capacity lead to artificial deadlock: process can not write on full communication channel. With Kahn semantic, there is deadlock if and only if all processes try to read on empty FIFO. Furthermore, in real-time context, it is preferred that Application Programming Interface (API) for memory allocation does not provide a mechanism for dynamic allocation but many KPN scheduler relies on the utilization of dynamically allocated FIFOs.

The scheduling issue has already been discussed by Parks [12] and Basten [3]. According to [12, 6], a dynamic scheduler should satisfy two requirements:

**Output Completeness** *The output of the realization should be equal to the output prescribed by the denotational semantics of KPNs.*

**Boundedness** *The scheduler should realize an execution where a bounded amount of memory is used for channels.*

The first algorithm [12] describes a straightforward strategy for resolving artificial deadlocks. Initially, the size of all FIFOs is predefined. If an artificial deadlock occurs, the smallest full FIFO is increased by one unit, thereby, it is guaranteed that all bottlenecks in KPN are eventually removed.

In [3], Basten shows the three major drawbacks with the precedent algorithm. He suggests an improvement: when an artificial deadlock occurs, to trace a causal chain to determine the bottleneck. Then, to enlarge the bottleneck FIFO in this causal chain. Moreover, if there are multiple bottlenecks in a single chain of causes blocking an output, Basten chooses to enlarge the smallest bottleneck FIFO. However, if the smallest bottleneck FIFO is not the cause of blocking an output, enlarge this, it is not the best strategy.

Finally, the deadlock detection issue was not discussed and it is a considerable problem in the real-time context. We will tackle this in our approach.

### 3 KPN Scheduling under time constraints

Our improvement presented in this paper defines a strategy that does not need to stop the whole network execution to resolve deadlocks and that provides an optimal memory allocation. The execution of KPNs will then benefits of the multiprocessor architecture advantages in a real-time context.

Although in our case (workstation) the memory size is not critical, we must optimize this size and as much as possible reduce the dynamic allocation memory because the class of real-time problems is exacting.

Below we have listed notations and terms used in this paper :

- $G(\mathcal{V}, \mathcal{E})$  denotes a directed acyclic graph with vertices  $\mathcal{V}$  (process) and edges  $\mathcal{E}$  (bounded FIFOs);

- $\mathcal{V} = \{v_i | i = 1, \dots, n\}$ , the set of vertices;
- $\mathcal{E} = \{e_{i,j} | \text{there is an edge between } v_i \text{ and } v_j\}$ , the set of edges;
- $W_{\mathcal{E}} = \{w_e(e_{i,j}) \in \mathbb{N} | e_{i,j} \in \mathcal{E}\}$ , the weights of the edges in  $\mathcal{E}$ ;
- $W_{\mathcal{V}} = \{w_v(v_i) \in \mathbb{N} | v_i \in \mathcal{V}\}$ , the weights of the vertices in  $\mathcal{V}$ ;
- $L(a) = \{w_v(v_i) = a | w_v(v_i) \in W_{\mathcal{V}}\}$ , the set of vertices which have the same weight  $a$ ;
- $|L(a)|$  is the number of elements in  $L(a)$ ;
- $L(a) \sqsubseteq L(b)$ , the activation of  $L(a)$  precedes the activation of  $L(b)$ ,  $\sqsubseteq$  defines an activation order on  $n$   $L(i)$  sets.

Our strategy is to define an activation order without modifying Kahn Process Network semantics and their properties. By Basten's strategy, we noticed that the position of the treated bottleneck had an importance.

$L(1) = \{w_v(v_i) = 1\}$  has all processes which have at least one entry from environment, all these processes have a weight which are worth one. The only assumption that we will make is  $L(n)$  set has only one process. Thus, a single process is selected to drive the whole network. This driving process is specified in the program and is usually the process that produces the ultimate output for the application.

Find a  $n$ -partition  $(L(1), \dots, L(n))$  of  $\mathcal{V}$ , i.e.,

$$\bigcup_{i=1}^n L(i) = \mathcal{V} \text{ and } L(i) \cap L(j) = \emptyset, \forall i \neq j$$

with

$$w_e(e_{i,j}) := w_v(v_i)$$

$$w_v(v_j) := \max\{w_e(e_{i,j})\} + 1$$

The adopted rule is to activate all processes of  $L(1)$  then  $L(2)$  and so on to  $L(n)$ :  $L(1) \sqsubseteq \dots \sqsubseteq L(n)$ . The rule to pass from  $L(i)$  activation to  $L(i+1)$  activation is done only when all processes in  $L(i)$  are blocked either in writing or reading.

The artificial deadlock is resolved when we activate  $L(n)$  then  $L(n-1)$  and so on to  $L(1)$ :  $L(n-1) \sqsubseteq \dots \sqsubseteq L(1)$ . The rule to pass from  $L(n-1)$  to  $L(n-2)$  is done in three steps: firstly, activate all processes in  $L(n-1)$  until a deadlock occurs, secondly, we increase by one unit all FIFOs which are full, thirdly we re-activate all processes in  $L(n-1)$  until a deadlock occurs, and only then  $L(n-2)$  is activated.

The following steps outline our algorithm:

1. Partition the graph in  $n$   $L(i)$  sets.
2. Activate  $L(1) \sqsubseteq \dots \sqsubseteq L(n)$  in a data-driven way.
3. If no artificial deadlock is detected, then terminate the execution (the execution is *output complete*),

4. Activate  $L(n - 1) \sqsubseteq \dots \sqsubseteq L(1)$  in a data-driven way.
5. If no artificial deadlock is detected, then terminate the execution (the execution is *output complete*).
6. Return to step 1.

If all processes are blocked in reading on empty FIFOs, the execution is *output complete* and it can be terminated.

Several remarks can be noticed. Firstly, the act of increasing the full FIFOs size only occurs after activation of the consuming processes. This fact made an optimized FIFO size because only FIFO causing a bottleneck are handled. Thus, we move the bottlenecks of fine towards the beginning. Secondly, the consequence of the activation order is that we improve the deadlock detection and resolution strategy. The bottleneck causing artificial deadlocks can be identified and removed without the need to wait the entire execution has deadlocked. It is a considerable issue when one has to meet some time constraints and there are a number of relatively independent parts in the network. Indeed, without our activation order (and obviously without any artefact), to detect an artificial deadlock requires  $1 + (m - 1)$  context switch, with  $m$  the number of processes of network. In our case, the number of context switch is always bounded by  $|L(i)|$ , with  $|L(i)| \leq m - 1, \forall m > 1$ .

In the next section, we will explain why the multiprocessor workstation is the target architecture too. Moreover, before to define the  $n$   $L(i)$  sets scheduling, we will describe the used ARTiS run-time.

## 4 Towards a real run-time

Like noticed in the first section, the multiprocessor workstation is both the development platform and the target architecture. In our case, this choice has been easy. Indeed, multiprocessor architecture provides several advantages for programming multiprocess real-time applications. It should be noted that in this paper, a multiprocessor architecture is defined as a set of processors that share a common pool of memory.

On a side, multiprocessor architecture offers us [4]:

1. Allows  $n$  highest priorities within the system
2. Combined determinism and I/O throughput
3. Efficient inter-task communication
4. Debug tools provide a single system view
5. Task migration allows load balancing

On the other side, the class of real-time applications (real-time intensive signal processing) targeted by us, require four fundamental properties from the operating system [7, 1]:

- Fast response to an external interrupt
- Deterministic program execution

- High I/O throughput
- Efficient and deterministic inter-task communication

Bridging both becomes obvious : 1) When a real-time application is composed of multiple tasks, more than one “highest priority” task in the system is authorized to have an unlimited access to a processor. 2) The largest source of non-deterministic program execution in a computer is external interrupts. The use of a symmetric multiprocessor architecture, which has the ability to route interrupts to particular processors, allows an interrupt handling implementation that achieves both determinism and high I/O throughput. 3) When a single copy of the operating system controls the entire system, the name space for inter-task communication facilities is maintained by a single entity. Moreover, the OS uses shared memory for shared data and the structures that control inter-task communication, inter-task communication can be performed with minimal overhead. 4) With an OS, tools used to debug applications provide a single system view of all tasks executing on the multiple processors.

Finally, the developer of a real-time application often wants to have static task to processor assignments, allowing him to guarantee worst case system behavior, but it is also possible to balance the load by migrating processes when the need is required. Thus, such an architecture, offers many benefits in terms of performance and practical of use.

Despite the numerous advantages of multiprocessors, it is not obvious to implement a real-time operating system on a SMP platform. The Asymmetric approach, opposed to Symmetric Multi Processing, is a way to achieve this goal. We proposed such an architecture in the frame of ARTiS [11]. ARTiS provides an Asymmetric Real-Time Scheduling for SMP systems : one can reserved real-time processor that will execute real-time tasks without latency. All the activities that are able to generated latency are hold on the non real-time processor(s). The ARTiS framework must be completed with a scheduling of the real-time activities on the real-time processors. Our proposition is then to implement a scheduler dedicated to KPN on the top of the ARTiS scheduler.

## 5 Proposal of an implementation

As we had described what ARTiS offers us as advantages, its interest become obvious.

Indeed, keeping some processors for real-time, in this case KPN processes, theirs processing will be particular. Each dedicated processor will have two queues. These queues will be of the same size  $n$ : one for  $L(1) \sqsubseteq \dots \sqsubseteq L(n)$  : Queue 1, and another one for  $L(n) \sqsubseteq \dots \sqsubseteq L(1)$  : Queue 2.

From there, the system becomes more classical. The scheduling (in space) of various processes is uniform and static. Each process during this creation receives a priority level. This priority level is obtained with the graph partitioning *i.e.*  $w_v(v_i)$ .  $L(i)$  describes the set of the various processes which have the same priority level  $p = i$ .

For practice reasons,  $L(1) \Leftrightarrow p = 0$  and so,  $L(n) \Leftrightarrow p = n - 1$ .

The scheduler can be defined by the following way :

1. The processor  $i$  activates one by one all processes of  $p$  level. When a process is completed, it is put in the other queue at  $n - p$  level.

2. As soon as  $p$  level does not contain any process, the processor  $i$  chooses a processor  $j$  uniformly at random.
3. If the victim  $j$ 's queue at  $p$  level is empty, processor  $i$  attempts to steal again.
4. If not, it steals the process from the tail of the queue at  $p$  level and begins executing it.
5. Otherwise, the processor  $i$  switches to  $p + 1$  level and return to step 1.

The performed processes from queue 1 are put in queue 2 and vice versa.

The processor can only execute or take a process of the other queue at  $p = n - 1$  level.

The artificial deadlock resolution is only allowed when processed processes come from queue 2.

The main idea of the algorithm defined above is that, the processor sequentially executes all processes in  $L(i)$  and before switches to  $L(i + 1)$ , it checks all processor queues at  $i$  level.

## 6 Conclusions

In this paper, we presented a new scheduler for KPN in real-time context on multiprocessor architecture. The main contribution of this algorithm is a very efficient scheduling of KPN: with an activation order we improve the deadlock detection and resolution strategy. The bottleneck causing artificial deadlocks can be identified and removed without the need to wait the entire execution has deadlocked. It is a considerable issue when we have to meet some time constraints and there are a number of relatively independent parts in the network. As a consequence we have an optimized allocation memory mechanism and we can limit the number of task context switches. Moreover, the two criterias for executing KPN mentioned previously are satisfied.

As we could see, we benefit from the multiprocessor architecture choice as development platform and as target architecture. The recurrence problem was not mentioned in the algorithm described in section 3. The reason is that, the problem is solved by itself with the implementation proposed in previous section. Indeed, if the developer can tag each process with the wanted priority (while following the prescribed rules in section 3), the problem does not occur. He can fit the behavior of KPN execution, according to himself.

Finally, we have described an efficient and complete run-time. The future work will be to implement, evaluate and comparison with others systems. The implementation will allow us to confirm the good properties of our proposition on real applications. Thus, a commercial workstation will become a credible alternative from performance and development cost point of view for real-time intensive signal processing applications.

## References

- [1] Gregory E. Allen and Brian L. Evans. Real-time sonar beamforming on workstations using process networks and POSIX threads. *IEEE Transactions on Signal Processing*, pages 921–926, March 2000.
- [2] Gregory Eugene Allen. Real-time sonar beamforming on a symmetric multiprocessor UNIX workstation using process networks and POSIX pthreads. M.Sc. Thesis, The University of Texas at Austin, Austin, TX, August 1998.

- [3] Twan Basten and Jan Hoogerbrugge. Efficient execution of process networks. In A. Chalmers, M. Mirmehdi, and H. Muller, editors, *Proc. Communicating Process Architectures*, pages 1–14, Bristol, UK, September 2001. IOS Press.
- [4] Stephen Brosky. Symmetric multiprocessing and real-time in PowerMAX OS. White paper, Concurrent Computer Corporation, Fort Lauderdale, FL, 2002.
- [5] Joseph T. Buck and Edward A. Lee. Scheduling dynamic dataflow graphs with bounded memory using the token flow model. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 429–432, Minneapolis, MN, April 1993.
- [6] Marc Geilen and Twan Basten. Requirements on the execution of kahn process networks. In *Programming Languages and Systems, 12th European Symposium on Programming, ESOP 2003*, pages 319–334, Warsaw, Poland, April 2003. Lecture Notes in Computer Science vol. 2618.
- [7] Steve Goddard and Kevin Jeffay. Analyzing the real-time properties of a dataflow execution paradigm using a synthetic aperture radar application. In *3rd IEEE Real-Time Technology and Applications Symposium (RTAS '97)*, Montreal, Canada, June 1997.
- [8] Gilles Kahn. The semantics of a simple language for parallel programming. In Jack L. Rosenfeld, editor, *Information Processing 74: Proceedings of the IFIP Congress 74*, pages 471–475. IFIP, North-Holland, August 1974.
- [9] Gilles Kahn and David B. MacQueen. Coroutines and networks of parallel processes. In B. Gilchrist, editor, *Information Processing 77: Proceedings of the IFIP Congress 77*, pages 993–998. North-Holland, 1977.
- [10] NSP shows promise on Pentium, PowerPC. Microprocessor Report, May 1995.
- [11] Momtchil Momtchev and Philippe Marquet. An asymmetric real-time scheduling for Linux. In *Tenth International Workshop on Parallel and Distributed Real-Time Systems*, Fort Lauderdale, FL, April 2002.
- [12] Thomas M. Parks. *Bounded Scheduling of Process Networks*. PhD Thesis, EECS Department, University of California, Berkeley, CA, December 1995.
- [13] Hideki John Reekie. *Realtime Signal Processing: Dataflow, Visual, and Functional Programming*. PhD Thesis, School of Electrical Engineering, University of Technology, Sydney, Australia, September 1995.