OREGON STATE UNIVERSITY COMPUTER SCIENCE DEPARTMENT

PIPELINING PERFORMANCE OF STRUCTURED DATAFLOW NETWORKS

Fred M. Tonge

Department of Computer Science
Oregon State University
Corvallis, Oregon 97331

82-4-1

CS-TR-82-4-1

PIPELINING PERFORMANCE OF STRUCTURED DATAFLOW NETWORKS

Fred M. Tonge

Department of Computer Science
Oregon State University

PIPELINING PERFORMANCE OF STRUCTURED DATAFLOW NETWORKS

Fred M. Tonge
Department of Computer Science
Oregon State University

1. Introduction

System architectures for interconnecting large numbers of processors are being widely studied [AG82,TH75,TR82]. Of particular interest in such architectures is the exploitation of concurrency among processors. This concurrency can be parallelism, in which different parts of a single data case are processed at the same time, or pipelining, in which different processes are carried out simultaneously on successive data cases. Many problems involving file and vector processing can be viewed as pipelining problems.

Both structured and arbitrarily interconnected networks of processors have been proposed. The asserted advantage of structured networks is that they are easier to comprehend, control, and so utilize effectively (arguments analogous to those made for structured programming [DA72,KO74]). The arguments for arbitrary networks are that they allow greater flexibility of interconnection and so can be more efficient in many cases.

Also, network architectures are being studied incorporating processor activation through data flow [AC82,AG82,DA82,TR82,WA82] and through flow of control [AN75,GA82,TH75].

In all of these cases, attention must be given to developing languages for expressing procedure interconnections and rules for allocating procedures to processors so as to obtain maximum concurrency.

This paper presents a particular approach to specifying procedure interconnection and allocation. The major result is that, within stated assumptions:

> networks constructed using a small set of structured process connectives can achieve at least as good throughput (pipelining performance) as arbitrarily interconnected networks.

2. Structured Networks of Processes

In this paper we focus on the logical interconnection of processors in a network. For illustrative purposes we consider each processor to be allocated a process, and so speak interchangeably of networks of processors or networks of processes.

In structured networks of processors, we allow three patterns of interconnection: serial (S), parallel (P), and alternative (X) (figure 1). (Although we use two components in these examples, in general the connectives may join an arbitrary number of components.)

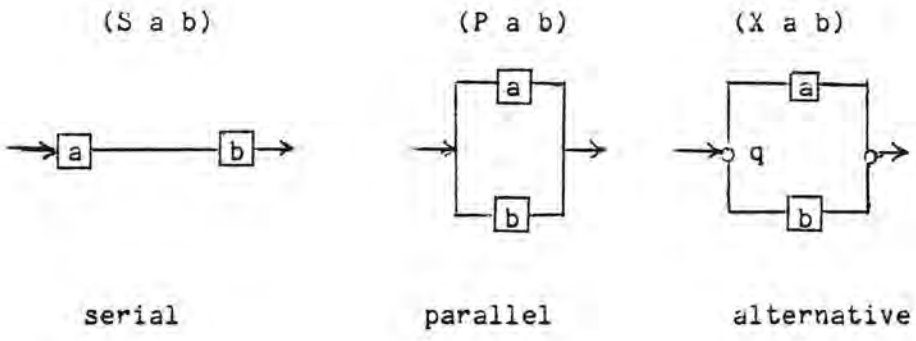


Figure 1.

In serial interconnection, process a is followed by (sends its outputs to) process b. In parallel interconnection, processes a and b each receive a copy of the input data, and an output data set is formed as a union of their individual outputs. The two processes may be carried out in either order, or in parallel if resources are available; their joint output is formed in a specified order. In alternative interconnection, only one of the component processes is performed, as selected by a predicate q on the input data set. These connectives are assumed order-preserving, in that successive output data sets are produced in the same order as the corresponding inputs.

These structured connectives can also be viewed as a way of composing a process description from its component processes. The resulting (compound) process description can then be decomposed/mapped/allocated onto interconnected processors [CO67,TO76]. As shown above, the connectives directly represent parallelism, and can be implemented to enhance pipelining. Although control-flow implementations are possible, it is natural to view such networks as data-driven, and we shall do so in the following.

A complete system utilizing this approach would include a language of expressions in which atomic and compound processes are described, the set of inter-process connectives, and a policy for allocating processes to a network of processors. All three of these would be implemented as "machine language" on each processor, so that process descriptions could be realized as programs almost directly, subject only to the level of translation done in simple assemblers. In this paper we concentrate on the set of inter-process connnectives.

Structured process description networks using these connectives are equivalent to Dijkstra's d-charts [DA72,KO74] with the addition of parallelism and the omission of cycles. The equivalent of cycles is achieved through recursive use of process names and allocation of new copies of the process description as required. Since allocation is flow-driven (components are allocated with the first invocation of a process), recursion is limited by the particular sequence of data at hand.

As an example, a process description network is given in figure 2 for

computing the square root s of a number n (assuming n>0) using t iterations of the Newton-Raphson approximation method. An informal pseudo-text is used to define atomic processes.

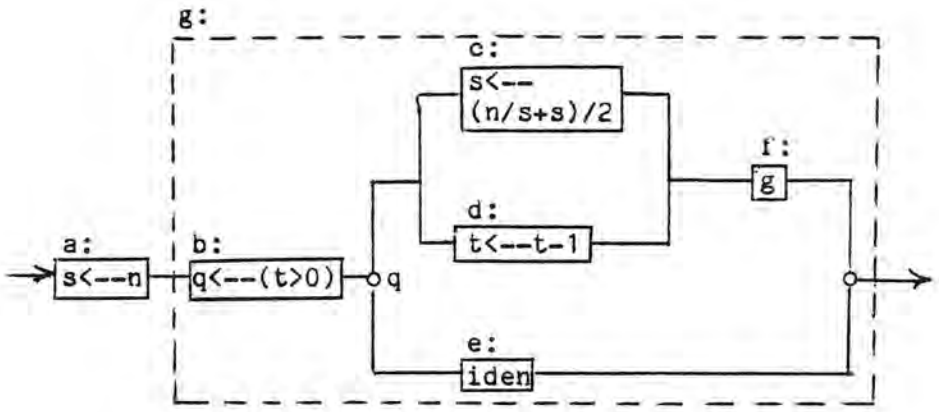


Figure 2.

This network can also be described by a process expression (here using process labels in place of actual expressions).

(S a g:(S b (X (S (P c d) f) e)))

3. Arbitrary Networks

We now consider networks constructed with greater freedom of interconnection than imposed on structured networks. We introduce five primitive interconnections from which arbitrary networks are formed. These are illsutrated in figure 3.

1. sequential, a single output to input line.

2. fork, a single line splitting to two lines. A data case arriving at the fork is copied, and copies sent along both lines.

3. join, with two output lines joining to form a single input line. Data cases from the two lines are combined into a single data case.

4. branch, a single line selected to one of two lines depending upon some value of the data case. (The value of the predicate is computed in an preceding process, so that the selection switch itself need only examine a single value.

5. merge, output from one of two lines switched to a single line. The merge acts as an arbiter to insure that data cases on the two lines are not intermixed.

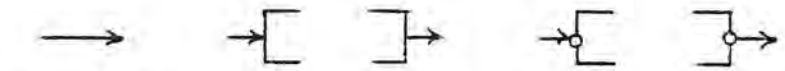


Figure 3.

Note that structured networks as previously defined are constructed from these primitive interconnections, with forks and joins (and branches and merges) occuring only in matching pairs. In arbitrary networks these interconnections can occur as desired, within the assumptions for networks given in the next section. An example of an arbitrary network is given in figure 4.

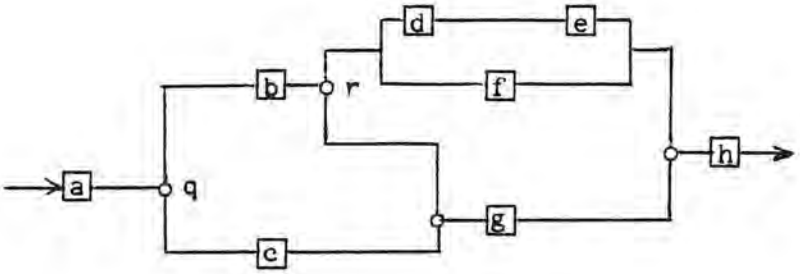


Figure 4.

4. Assumptions for Networks

We make the following assumptions for both structured and arbitrary networks:

1. As many processors as are needed are available to decompose a process description.

2. Communication and switching times are negliable. Performance is determined by processing time and by time spent waiting for computation of preceding processes.

3. The data values produced from one invocation of a process are transmitted as a single message, here called a data case.

4. Networks are acyclic.

5. Networks are "well-formed", in the sense that any data case entering a network will eventually complete processing, and no parts or copies of the data case will be left in the network [GO72,LA77].

6. Networks are order-preserving, with output data cases produced in the same order that input data cases are accepted. For structured networks this is assured since the individual connectives are order-preserving. For arbitrary networks we assume (if needed) an instantaneous ordering process appended to the network.

5. Performance Measures

The following deterministic measures of process characteristics are of interest:

e(i): elapsed time of process i from acceptance of input to production of corresponding output. For atomic processes, this is the actual processing time.

c(i): capacity of process i (maximum number of input cases that can be accepted before production of an output and acceptance of more input) -- assumed to be 1 for atomic processes.

r(i): average period of process i (time between successive output cases at maximum pipelining) at steady state assuming sufficient input.

The concept of period (the inverse of throughput) is basic to this study of pipelining performance. The period of a process depends on its input as well as on the process itself. If input arrives too slowly to keep the process active, the input period determines the process output period. Otherwise, process characteristics determine the output period. In either case, the process may alter the relative timing of data cases with a sequence of inputs.

From the above definitions, performance measures for the structured connectives can be derived.

atomic process:	e(i) = atomic processing time
	c(i) = 1 (by assumption)
	r(i) = e(i)/c(i) = e(i)
serial (S a b):	e(S) = e(a)+e(b)
	c(S) = c(a)+c(b)
	r(S) = max(r(a),r(b))
parallel (P a b):	e(P) = max(e(a),e(b))
	c(P) = min(c(a),c(b))
	r(P) = max(r(a),r(b),e(P)/c(P))

These performance measures are directly extendable to connectives with more than two components.

Performance characteristics of the alternative interconnection depend on the sequence of branches selected by successive data cases, and so can be analyzed only for specific situations. For data sequences with a regular repeating pattern, a period for the repetition as a whole may be found through analysis of the specific pattern. Those cases in which one value of the predicate occurs rarely (as in exception or error handling) may be approximated by considering the dominant value only.

For one specific case, discussed below, performance measures for the alternative can be defined across all input data sequences. We use this case in our comparative analysis of structured and arbitrary networks.

Define the path of a data case through a network to be the sequence of processes through which the data case moves. For networks without branches, all data cases have the same path. For networks with branches, a path can be characterized by an expression of the values selecting the branches.

Define the load, w(i), on process i with respect to a particular regular data sequence to be:

> w(i): the fraction of data cases in one repetition of the regular data sequence whose paths include process i.

Define the identity process iden to be an atomic process with e(iden)=0 and c(iden)=1 (a one data case buffer).

Then, for the alternative interconnection of process a and the identity process, the period may be defined as:

alternative (X a iden): r(X) = r(a)*w(a)

Since the period is an average over a sequence of data cases, the relative timing of the input data cases within the sequence must be such as to provide inputs as needed. The general definition of period holds only when the input period does not determine throughput (that is, when there is sufficient input to keep the network active).

6. Equivalent Networks

We now consider what it means to say that two networks are equivalent. The atomic processes of a network can be divided into two sets--those which compute only the values of predicates used within the network and all others. We call the latter basic processes.

A network k is equivalent to a network j if:

> for every data case, the basic processes in its path for network k are the basic processes in its path for network j;
>
> for every data case, the order of basic processes in its path for network k is consistant with (up to the partial ordering specified by parallel processes) the ordering of those processes in network j;
>
> the branching predicates of networks k and j are composed from the same set of logical variables; and

for every data case, the output for network k is exactly that for network j.

For example, the two structured networks of figure 5 are equivalent to the arbitrary network of figure 4 and to each other.

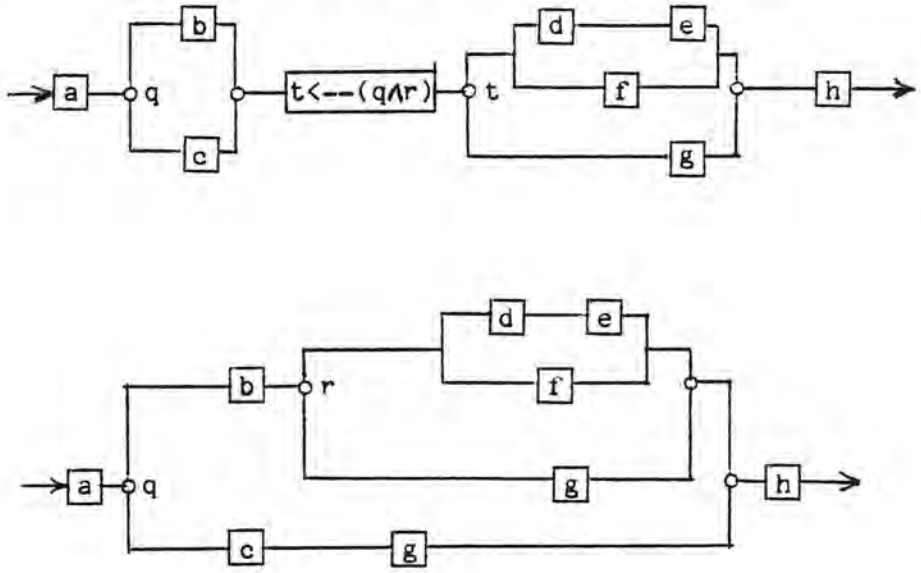
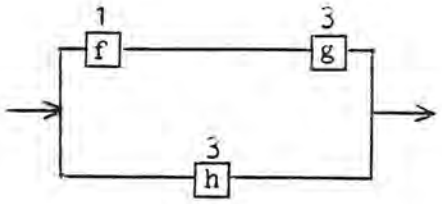


Figure 5.

7. Effect of Buffering on Throughput

The period of a structured network depends on both the elapsed times and, for the parallel case, the capacities of its components. For example, in the following network of atomic processes (figure 6), the period of the upper branch is 3, the period of the lower branch is 3, and the period of the entire network is 4. (The number given above each process is its elapsed time.)



f	g	(Sfg)
e(f)=1	e(g)=3	e(Sfg)=1+3=4
c(f)=1	c(g)=1	c(Sfg)=1+1=2
r(f)=1/1=1	r(g)=3/1=3	r(Sfg)=max(1,3)=3

h	(P(Sfg)h)
e(h)=3	e(P(Sfg)h)=max(4,3)=4
c(h)=1	c(P(Sfg)h)=min(2,1)=1
r(h)=3/1=3	r(P(Sfg)h)=max(3,3,4/1)=4

Figure 6.

Buffering can be used to reduce the period of a network by increasing the capacity of a part of the network. In figure 7 we show the network of figure 6 augmented with a buffer of capacity one (an identity process). The capacity of the lower branch is now 2, and the period of the network is 3.

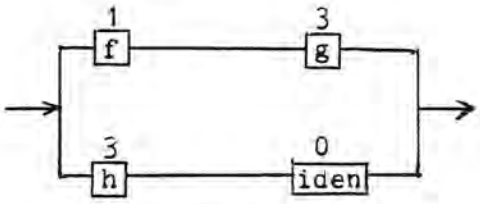


Figure 7.

The period of any structured network without predicates can be reduced to that of the maximum of the periods of its atomic components using the following observation:

> For serial connectives, the period is already that of the maximum component.
>
> For parallel connectives, the period is the maximum of component periods unless that is exceeded by (larger component elapsed time)/(smaller component capacity). In the latter case, insert sufficient buffer capacity in series with the smaller capacity component to equalize the capacities of the two components.

A straightforward procedure follows from the above for decreasing the

period of any structured network without predicates to the minimum attainable. (The introduction of buffer processes to increase pipelining throughput has been noted previously [PA76].)

More generally, in any network in which the outputs of several processes must be coordinated (such as combining the output data cases of parallel components into a single output case), a delay could be introduced into the flow of data cases, possibly resulting in forced idle (waiting) time for subsequent processes. Such forced idleness will increase the network period if the bottleneck process whose elapsed time determines overall throughput is forced to wait. In general, any process which alters the time pattern of input data cases in producing output data cases may introduce idle time for its successors. And in general, buffers may be introduced to eliminate that idleness.

While networks without predicates produce a regularly spaced output pattern, predicate networks can introduce irregular respacings into the time pattern of outputs. The following lemma is motivated by the desire to insure that respacings in a predicate network do not cause subsequent idleness of a bottleneck process.

Decoupling Lemma:

> Any process X may be replaced by an equivalent process Y (with possibly greater capacity) which preserves both the period of the original process and the timing of its input sequence.

Proof:

> (by construction) Process Y consists of the serial connection of process X and a buffer process of sufficient capacity. The buffer process collects (at most) all data cases in a repetition of the output sequence of X and then releases them with the time spacing of the input sequence.

In practice it is sufficient to provide only enough buffering to guarantee that subsequent processes can operate at maximum throughput.

One interpretation of the lemma is that, if all data cases in one repetition of the input sequence are available "at the same time", then all output can be made available "at the same time" without increasing the period.

(The introduction of buffers suggested by the decoupling lemma is only one means of altering a network to achieve maximum throughput. The addition of buffers to increase the capacity of one of two parallel processes is another. Similarly, in some cases buffers can be added to one of two processes in alternation to decrease their overall period. (In order-preserving networks, the outputs of alternative processes must be coordinated in a way similar to those of parallel processes.) Also, the period of a bottleneck process may be reduced by alternating (duplexing) two copies of the original process. The relative desirability of these approaches to increasing throughput depends in part on physical characteristics of the processors on which

the network is implemented. If processors generally have large amounts of storage, then introducing buffers may have little real cost. If processors generally have small amounts of storage, then duplexing processes may be desirable.)

8. Performance of Equivalent Networks

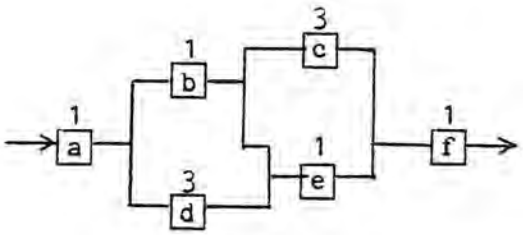
We now consider the elapsed time and period (throughput) performance of equivalent arbitrary and structured networks.

Well-known Result on Elapsed Time:

There exist arbitrary networks for which no equivalent structured network has an equal or smaller elapsed time.

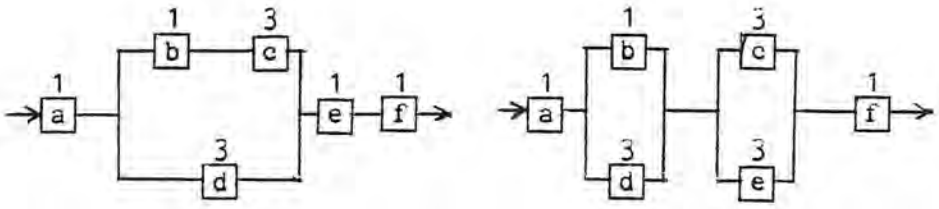
Example:

For the network of atomic processes given in Figure 8 there are twelve possible equivalent structured networks without additional processes, of which two are shown in figure 9. None of the twelve has an elapsed time less than 7, while the arbitrary network has an elapsed time of 6.



e=6, r=3

Figure 8.



e=7, r=4 e=8, r=3

Figure 9.

Theorem on Pipelining:

For any arbitrary network, there exists for any regular input sequence an equivalent structured network with a period at least as small as that of the arbitrary network.

Proof:

1. The period of a network must be at least as great as the maximum of the products of the periods of its atomic component processes multiplied by their loads in the regular sequence.

$$p(arb) \geq \max(p(i)*w(i) \mid i \text{ in } arb)$$

2. Since the network contains no cycles, component processes can be numbered with consecutive integers such that each process is assigned a higher number than any of its predecessors. (This is a common technique in critical path algorithms, for example [KE61].)

3. Since the network contains no cycles, there are only a finite number of paths from the beginning to each process, and those paths can be traced. (Techniques for path tracing are common in path analysis [AL76,FO76].) A predicate expression consisting of the intersection of those predicates in the arbitrary network which must be true for a given path can be associated with that path. A predicate expression q(i) which is the union of the expressions for all paths leading to atomic process u(i) can be associated with that process.

4. Each atomic process u(i) may be replaced by an equivalent process v(i) where

$$v(i)=(S\ z(i)\ (X\ u(i)\ iden))$$

and z(i) is a process for computing q(i).

5. The processes (numbered, say, 1 through m) can be organized into an equivalent structured network of the form:

$$(S\ v(1)\ (S\ v(2)\ \ldots\ (S\ v(m-1)\ v(m))\ \ldots\ ))$$

The numbering procedure guarantees that the ordering of the structured network is consistant with that of the arbitrary network. The predicate of the alternative connection containing process u(i) is that predicate q(i) derived from all paths to u(i) in the arbitrary network.

6. By the decoupling lemma, sufficient buffering can be added to each alternative, treated as a process, to insure that its outputs occur in a time pattern which will not force waiting in later processes. Then, the period of the structured network will be the larger of the periods of the first component or the remaining network, or, recursively:

$$p(str)=\max(p(1)*w(1),\max(p(2)*w(2),\ \ldots\ ))$$
$$=\max(p(i)*w(i) \mid i \text{ in } str)$$

Figure 10 shows a minimum period network equivalent to the network of figure 4 (with identity processes omitted for clarity).

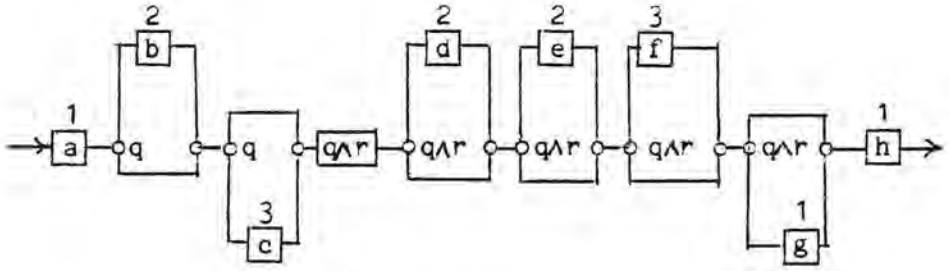


Figure 10.

In most cases, equivalent structured networks exist with processes on both branches of many alternatives, and often requiring fewer buffers for maximum throughput. For example, the equivalent structured network as defined above can generally be simplified as follows:

if q(i) is true than v(i)=u(i);

if q(i) is a predicate of the original network or its negation, then z(i) can be omitted;

any remaining z(i)'s need only be computed once.

A simplified network for that of figure 10 is given in figure 11. (The two additional buffers are required to maintain throughput while preserving order.)

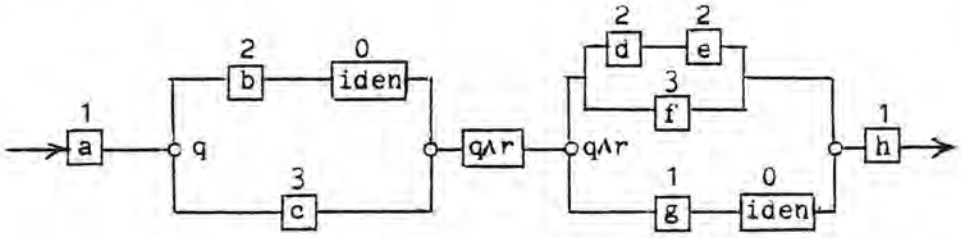


Figure 11.

From the standpoint of pipelining throughput, the only reduction in performance incurred by expressing predicate process descriptions in a structured manner need be that associated with computing additional compound predicates from already computed predicates. And only in those cases where the computation of one "and" or "or" exceeds that of the bottleneck process will throughput be affected. (These compound predicates are associated with the occurrence in the arbitrary network of merges not paired with branches, and are reminiscent of the additional control variables found necessary by Bohm and Jacopini [BO66].)

(Note that while the organization of processes in the structured

network is independent of the input pattern, the number and placement of buffers so as to achieve maximum throughput varies with the particular input. Upper limits on buffer requirements for a network may be derived by worst case analysis for each alternative or, for a particular input pattern, by static analysis of the alternatives. In almost all cases, these limits will be too severe.)

9. Concluding Remarks

The conclusion drawn from this analysis is that there need be no reduction in throughput (pipelining performance) from interconnecting networks of processes (or, constructing compound process descriptions) using only a small set of structured inter-process connectives. Indeed, in some cases improved throughput over arbitrary interconnection is achieved.

While some of the assumptions underlying the analysis may seem distant from physical reality, they do not necessarily affect the conclusion. For example, for the assumption of no communication delays to alter the conclusion, one would have to argue that communication delays will be greater in structured than in arbitrary networks. Since the magnitude of communication delays is closely related to the scheme for allocating processes to processing elements (see, for example, [TH78]), this argument would imply that a mechanized allocation process can more efficiently allocate an arbitrarily interconnected network than a structured one. This seems contrary both to intuition and to experience with the management of data structures. From a pipelining standpoint, interprocess communication channels are equivalent to buffer processes. Only if their "period" is greater than that of the maximum process do they affect throughput. Communication delays do affect performance, and various inter-processor connection schemes should be studied, but there is no reason to believe that structured networks would suffer disproportionately from those delays.

Similar arguments can be made concerning the assumption of sufficiently many processors. Again, an informal analysis suggests that whatever schemes are developed for reallocating limited numbers of processors are unlikely to be less efficient for structured networks.

Another area of interest is dynamic introduction of buffers for performance enhancement. A dynamic tuning process based on ongoing performance seems both feasible and desirable. Although appropriate introduction of buffers can also enhance the throughput of arbitrary networks, it would seem likely that the dynamic analysis of structured networks, like their static analysis, would be more straightforward.

Finally, we have not spelled out the obvious algorithms for converting arbitrary process networks into structured ones. The implication of these results is that process descriptions can be thought about and constructed in a structured manner from the beginning.

10. Acknowledgements

Work on structured process description has involved Robert Barton and Richard Cowan (then of the Burroughs Corporation) and Randell Flint of U. C. Irvine as well as the author, and all have contributed to developing the concepts presented here. This study was supported in part by the Burroughs Corporation and by National Science Foundation grant no. MCS77-02715.

11. References

[AC82] Ackerman, W.B., "Data Flow Languages", Computer (February 1982).

[AG82] Agerwala, T. and Arvind, "Data Flow Systems", Computer (February 1982).

[AL76] Allen, F.E. and J. Cocke, "A Program Data Flow Analysis Procedure", Communications of the ACM, 19 (March 1976).

[AN75] Anderson, G.A. and E.D. Jensen, "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples", Computing Surveys, 7 (December 1975).

[AR77] Arvind and K.P. Gostelow, "A Computer Capable of Exchanging Processors for Time", Proceedings IFIP Congress '77, Toronto, Canada (1977).

[BO66] Bohm, C. and G. Jacopini, "Flow Diagrams, Turing Machines and Languages with Only Two Formation Rules", Communications of the ACM, 9 (1966).

[CO67] Cooper, D.C., "Some Transformations and Standard Forms of Graphs, with Applications to Computer Programs", Machine Intelligence, 2 (1967).

[DA72] Dahl, O-J, E.W. Dijkstra and C.A.R. Hoare, Structured Programming, Academic Press, New York (1972).

[DA82] Davis, A.L. and R.M. Keller, "Data Flow Program Graphs", Computer (February 1982).

[FO76] Fosdick, L.D. and L.J. Osterweil, "Data Flow Analysis in Software Reliability", Computing Surveys, 8 (September 1976).

[GA82] Gajski, D.D., D.A. Padua, D.J. Kuck and R.H. Kuhn, "A Second Opinion on Data Flow Machines and Languages", Computer (February 1982).

[GO72] Gostelow, K.P., V.G. Cerf, G. Estrin and S.A. Volansky, "Proper Termination of Flow-of-control in Programs Involving Concurrent Processes", Proceedings of the ACM, 25th Anniversary Conference, Boston (August 1972).

[KE61] Kelley, J.E., Jr., "Critical-path Planning and Scheduling: Mathematical Basis", Operations Research, 9 (1961).

[KO74] Kosaraju, S. Rao, "Analysis of Structured Programs", Journal of Computing and Systems Sciences, 9 (1974).

[LA77] Larson, K.C., "A Token Flow Model Applied to Computer Networks", Ph. D. dissertation, Information and Computer Science, U. C. Irvine (1977).

[PA76] Patel, J.H. and E.S. Davidson, "Improving the Throughput of a Pipeline by Insertion of Delays", Third Symposium on Computer Architecture, IEEE Computer Society (January 1976).

[TH75] Thurber, K.J., "Associative and Parallel Processors", Computing Surveys, 7 (December 1975).

[TH78] Thomas, R.E., "Performance Analysis of Two Classes of Dataflow Computing Systems", M. S. thesis, Information and Computer Science, U. C. Irvine (January 1978).

[TO76] Tonge, F.M., "Expressions for Time and Space in a Recursive Realization of Parallelism", TR#79, Information and Computer Science, U. C. Irvine (May 1976).

[TR82] Treleaven, P.C., D.R. Brownbridge and R.P. Hopkins, "Data-Driven and Demand-Driven Computer Architecture", Computing Surveys, 14 (March 1982).