Approximate Time Separation of Events in Practice *

Supratik Chakraborty † supratik@celery.stanford.edu
Pasupathi A. Subrahmanyam ‡ subra@research.bell-labs.com
David L. Dill † dill@cs.stanford.edu

† Computer Systems Laboratory, Stanford University, Stanford, CA 94305
‡ Bell Laboratories, Lucent Technologies, Murray Hill, NJ 07974

Abstract

Finding bounds on the time separations between events is a fundamental problem in the analysis of concurrent systems. In [4], we proposed a polynomial-time approximate algorithm for computing bounds on the separations between all pairs of events in acyclic timing constraint graphs. This paper describes applications of our algorithm to two interesting problems: (i) a buffered producer-consumer system, and (ii) a multiprocessor system operating under different scheduling and buffering schemes. We describe how the temporal behavior of each system is modeled using min and max constraints in a timing constraint graph. The resulting graph is analyzed using the algorithm of [4] to answer several important questions about the system.

1. Introduction

A central problem in the analysis of concurrent systems is computing bounds on the time separations between events. Important questions about the temporal behavior of a system can be formulated in terms of time separations between appropriate events. An analysis yielding bounds on these separations can be used to answer such questions, providing insight into the system's behavior.

In [4], we developed an efficient approximate algorithm for computing minimum and maximum time separations between *all* pairs of events in systems specified by acyclic timing constraint graphs [4]. Unlike several previous approaches [1, 10, 2, 6], our algorithm allows *both* min and max type constraints in the same system, *and* has a worst-case time complexity that is *polynomial* (quadratic in practice) in the number of events. While the computed bounds are conservative in the worst-case, experiments on several large concurrent systems have shown that our results are highly accurate and often exact in practice [4].

This paper describes applications of our algorithm to two interesting problems with inherent concurrency and disjunctive causality. The first problem involves analysis of a buffered producer-consumer system that operates in cycles in the following manner: Each producer produces a certain number of items in each cycle and inserts them in a common buffer as they are produced. Consumers retrieve items from the buffer and process them. Each producer, after producing its quota of items, waits for all items in the buffer to be consumed before starting production for the next cycle of operation. We analyze the temporal behavior of this system and answer several interesting questions about the system, e.g., "How small can the buffer be while still ensuring that no producer ever waits for an empty slot?" or "How long may an item spend in the buffer before being consumed?" or "What is the latency of one cycle of operation?"

Our second application is an analysis of the synchronization traffic and overall performance of a system of concurrently operating processors communicating via a single shared bus. This is motivated by an image processing application, where different sections of an image are independently processed by concurrent processors and the results aggregated at the end to produce the final effect. We study the effects of different scheduling and buffering schemes on the bus traffic and on the overall system performance. This helps in evaluating different buffering and scheduling schemes for this class of applications.

1.1. Previous work

Previous work in this area comes from a wide variety of contexts. Burns [3] considered cyclic graphs with fixed delays and conjunctive causality for performance analysis of asynchronous circuits. Lee [8] extended this work to deal with OR-causality. McMillan and Dill [10] proposed a polynomial-time algorithm for computing time separations between events in acyclic timing-constraint graphs with max type constraints. They also proposed a branch-and-bound technique for analyzing systems with min and max constraints and applied it to an interface timing verification problem. Myers and Meng [11] proposed a polynomial-time approximate algorithm for cyclic graphs with max-type constraints and applied it to the synthesis of timed circuits. Burks and Sakallah [2] posed the problem as a *min-max linear programming* problem and used it for computing optimal clock schedules in synchronous circuits. Lavagno

* This work was supported by a grant from the Semiconductor Research Corporation (Contract No. 96-DJ-389) and a gift from Sun Microsystems.

and Sangiovanni-Vincentelli used time separations between events to obtain optimum delay paddings for eliminating hazards in asynchronous circuits [7]. Hulgaard [6] proposed an exact algorithm for computing the maximum time separation between two specific events in cyclic systems with max type constraints. An extension of this algorithm for analyzing safe Petri nets with choices was also proposed in [6]. However, modeling min constraints using Hulgaard's techniques requires the use of arbitration choice [6]. Unfortunately, with arbitration choice, his analysis, like ours, is conservative. A large number of other timing analyzers catering to specific applications, e.g., [1, 9], have also been developed.

2. A polynomial-time approximate algorithm

In this section, we give an overview of the theory and design of a polynomial-time approximate algorithm for computing bounds on the time separations between all pairs of events in systems specified by acyclic timing constraint graphs. Details of the algorithm may be found in [4].

2.1. Mathematical theory

We represent the temporal behavior of concurrent systems using timing constraint graphs [10]. Vertices in the graph represent events and directed edges represent causal dependencies between them. An edge from event i to event j is labeled with the interval $[d_{i,j}, D_{i,j}]$ representing the delay $\delta_{i,j}$ in the propagation of event i to event j. Each event i is also labeled with a min or max operator specifying how the time of occurrence of event i, denoted t_i, depends on those of its predecessors in the timing constraint graph. For example, if there exist directed edges from events j and k to i and if event i is labeled with a min operator, then $t_i = \min(t_j + \delta_{j,i},\ t_j + \delta_{k,i})$

Given such a timing constraint graph, our goal is to compute upper and lower bounds on $(t_i - t_j)$ for every pair of events i and j. Since $\min(t_i - t_j) = -\max(t_j - t_i)$, it suffices to determine $\max(t_i - t_j)$ for every *ordered pair* of events (i, j). We restrict our analysis to acyclic timing constraint graphs, which can have multiple *source* events (events with no incoming edges). We assume bounds on the separations between the source events are given.

It has been shown by previous researchers that the general form of this problem is NP-complete [10, 2]. The feasible space of the system of constraints, which contains min and max functions, is non-convex. This renders standard convex optimization techniques inapplicable. We propose to solve the problem efficiently by first finding a small convex "envelope" of the feasible space, and then maximizing $(t_i - t_j)$, for every ordered pair (i, j) in the approximate convex space. Since the original feasible space is completely contained within the approximate convex space, maximizing $(t_i - t_j)$ in the approximate space yields an upper bound of the true maximum in the original feasible space. Unfortunately, computing the smallest enclosing convex space (convex hull) in n dimensions is an intractable problem. Our strategy, therefore, is to approximate each min and max constraint using a system of linear inequalities, such that the feasible space of this approximate system of linear inequalities contains the feasible space of the original min or max constraint from which it is derived. We present below the system of linear inequalities that results from approximating a max (or min) constraint. The detailed derivations may be found in [4].

Let $\Delta_{i,j}$ represent an upper bound (achievable or not) of the separation $(t_j - t_i)$. In addition, let $preds(i)$ represent the set of immediate predecessors of event i in the timing constraint graph. It can be then be shown [4] that any point in the solution space that satisfies $t_i = \max_{j\,\in\, preds(i)}(t_j + \delta_{j,i})$, also satisfies *all* of the following linear inequalities (denoted by the logical conjunction):

$$(\textstyle\bigwedge_{\ j \in preds(i)}\ (t_i \ \geq\ t_j + d_{j,i})) \qquad \bigwedge$$
$$(\textstyle\bigwedge_{\text{all s} \neq i}\ (t_i \leq t_s + \max_{l \in preds(i)}(\Delta_{s,l} + D_{l,i}))). \quad (1)$$

In other words, the feasible space of the system of inequalities (1), which is necessarily convex, completely contains the feasible space of the original max constraint. This leads to the observation that $-d_{j,i}$ is an upper bound of $\Delta_{i,j}$ for all $j \in preds(i)$. Similarly, $\max_{\ l\,\in\, preds(i)}\ (\Delta_{s,l} + D_{l,i})$ is an upper bound of $\Delta_{s,i}$ for all $s \neq i$. A min constraint ($t_r = \min_{j \in preds(r)}(t_j + \delta_{j,r})$) can be approximated in a similar way [4], leading to the observation that $D_{j,r}$ is an upper bound of $\Delta_{j,r}$ for all $j \in preds(r)$, and $\max_{l\,\in\, preds(r)}(\Delta_{l,s} - d_{l,r})$ is an upper bound of $\Delta_{r,s}$ for all events $s \neq r$.

2.2. A polynomial-time algorithm

The approximations outlined in Section 2.1 have been used to design an efficient algorithm for computing bounds on the time separations between all pairs of events in acyclic timing constraint graphs. We simply present the algorithm here without going into the details of the algorithm design. Further details about the algorithm may be found in [4].

Our algorithm takes as input an acyclic timing constraint graph with n events. The events are topologically indexed with event 0 as a source event. The output of our algorithm is an $n \times n$ matrix, Δ, where $\Delta_{i,j}$ represents an upper bound of $(t_j - t_i)$. Fig. 1 shows the steps of our algorithm.

The computational complexity of our algorithm is $O(n^2.p)$, where $p = \max_{all\ events\ i} |preds(i)|$. In practice, p is usually a small constant. Consequently, our algorithm has a complexity that is $O(n^2)$ for all practical purposes.

3. Applications

In this section, we apply our algorithm to analyze two interesting systems with inherent concurrency and disjunctive causality. We model the behavior of each system with a timing constraint graph, formulate important questions about the behavior of the system in terms of time separations between events, and use the algorithm of Section 2.2 to answer these questions.

Initialization:
for each i in 0 to $(n-1)$
  for each j in 0 to $(n-1)$
    if $(i == j)$ $\Delta_{i,i} = 0$;
    else if (i, j are source events)
      $\Delta_{i,j} =$ given value;
    else $\Delta_{i,j} = +\infty$;

Main Algorithm:
for each i in 1 to $(n-1)$
  1. Compute_Initial_Estimates(i);
  2. **if** (i is max type)
    for each j in 0 to $(i-1)$
      $\Delta_{i,j} = \min_{k \in preds(i)}(\Delta_{k,j} - d_{k,i})$;
      if (j is max type)
        $\Delta_{i,j} = \min(\Delta_{i,j},\ \max_{k \in preds(j)}(\Delta_{i,k} + D_{k,j})$;
   else if (i is min type)
    for each j in 0 to $(i-1)$
      $\Delta_{j,i} = \min_{k \in preds(i)}(\Delta_{j,k} + D_{k,i})$;
      if (j is min type)
        $\Delta_{j,i} = \min(\Delta_{j,i}, \max_{k \in preds(j)}(\Delta_{k,i} - d_{k,j})$;

Compute_Initial_Estimates(i: index)
if (i is max type)
  for each j in $preds(i)$ $\Delta_{i,j} = -d_{j,i}$;
  for each j in 0 to $(i-1)$
    $\Delta_{j,i} = \max_{k \in preds(i)}(\Delta_{j,k} + D_{k,i})$
else
  for each j in $preds(i)$ $\Delta_{j,i} = D_{j,i}$;
  for each j in 0 to $(i-1)$
    $\Delta_{i,j} = \max_{k \in preds(i)}(\Delta_{k,j} - d_{k,i})$

Figure 1. Timing analysis algorithm

3.1. A buffered producer-consumer system

We consider a buffered producer-consumer system (Fig. 2) operating in the following manner: During each cycle of operation, each producer, P_i, produces a certain number, n_i, of items, performing some processing between successive productions. As each item is produced, it is inserted into a buffer, which is sufficiently large so that producers never have to wait for an empty slot. Consumers retrieve items from the buffer in FIFO order and processing them. Initially, consumer C_1 retrieves the first item from the buffer, consumer C_2 retrieves the next item, and so on. Subsequently, whenever a consumer finishes processing an item, it retrieves the next available item from the buffer. After producing its quota of items, each producer waits for all items in the buffer to be consumed before starting production for the next cycle of operation. This would be the case, for example, in a manufacturing system, where the type of processing done by the consumers differs from one cycle to another. The ensemble of consumer processes must, therefore, be switched from one processing mode to another at the beginning of each cycle and the producers disallowed from inserting new items into the buffer until then.

For purposes of exposition, we describe in detail how the

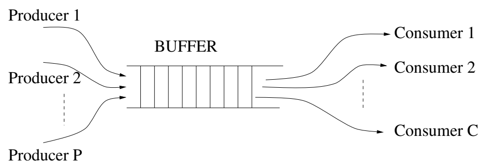

Figure 2. A buffered producer-consumer system

temporal behavior of this system is modeled using min and max constraints in a timing constraint graph. The approach used is, however, very systematic, and easily automated. In order to illustrate our approach, we consider a small example with 2 producers and 2 consumers. Each producer produces 2 items per cycle and requires a processing delay of 1 to 4 time units between successive productions. Similarly, each consumer incurs a delay of 2 to 4 time units while processing an item retrieved from the buffer.

We first observe that the buffer effectively orders the items according to their production times. To model this, we derive a system of min and max constraints that sorts the production times in chronological order. Fig. 3 shows these constraints for our illustrative example. The ordering process can be viewed as a recursive procedure that first identifies the earliest production time and separates it from the rest. The circled portion of Fig. 3 effectively implements this: the earliest production time is the time of occurrence of event $B1$, while the other production times are the times of occurrences of the events represented by the shaded vertices. The same procedure may now be recursively applied to the shaded vertices, yielding the timing-constraint graph of Fig. 3.

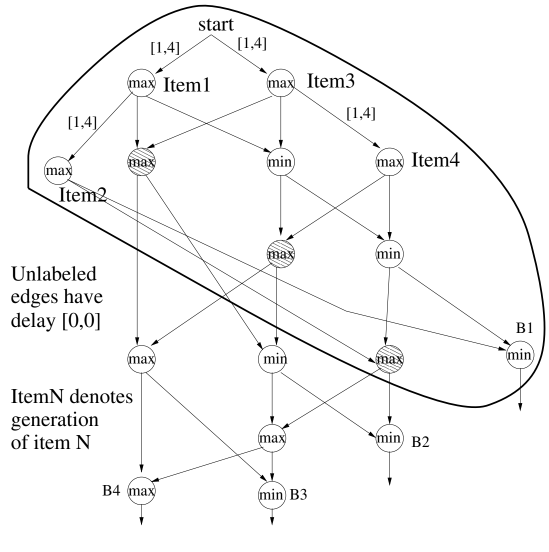

Figure 3. Constraints for chronological ordering

We now model the retrieval of items from the buffer and their subsequent processing by the consumers. Referring to our example, consumer C_1 retrieves the first item from the buffer and starts processing it as soon as it is produced (event $B1$ in Fig. 3). Similarly, C_2 starts processing the next item in the buffer as soon as event $B2$ occurs (see Fig. 3). If $E1$ and $E2$ represent the completion of processing of these items by the respective consumers, we draw edges $B1 \rightarrow E1$ and $B2 \rightarrow E2$ and label them with the processing delay of the consumers. This is shown in Fig. 4, which may be viewed as a continuation of the timing constraint graph of Fig. 3.

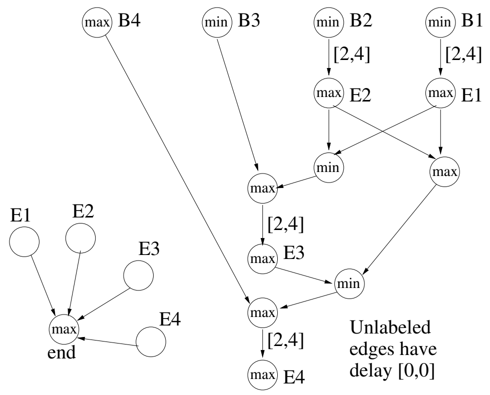

Figure 4. Retrieving items from the buffer

The consumer that finishes earliest then grabs the third item from the buffer. However, if this item has not been produced by the time C_1 or C_2 becomes free, the early finishing consumer must wait for the item to be produced, and then retrieve it from the buffer. We model this behavior by taking the maximum of t_{B3} (insertion of third item into the buffer) and $\min(t_{E1}, t_{E2})$ (earliest time a consumer becomes free) (see Fig. 4). The resulting time gives the instant when the third item is retrieved from the buffer. Processing this item now consumes an additional 2 to 4 time units. Vertex $E3$ in Fig. 4 represents the completion of processing of the third item. The retrieval of the fourth item from the buffer and its subsequent processing are modeled in a similar manner. The resulting timing constraint graph is shown in Fig. 4, where $E4$ represents the completion of processing of the fourth item.

The complete timing constraint graph for one cycle of operation (Fig. 3 concatenated to Fig. 4) is analyzed using our algorithm [4], yielding bounds on the time separations between *all* pairs of events simultaneously. These bounds are now used to answer several interesting questions about the system as described below. The entire analysis takes 0s (below system time resolution) on a Sun UltraSparc-1 (143 MHz) with 128 MB of memory.

1. The minimum buffer size required to ensure that no producer ever waits for an empty buffer slot is an important parameter of the system. To compute this, we let λ_i and Λ_i denote the times when item i enters the buffer and leaves the same. Clearly, if $\min(\lambda_{i+k} - \Lambda_i) \leq 0$, $(k+1)$ items (i.e., items $i, i+1, \ldots, i+k$) can simultaneously exist in the buffer. Thus, if a total of N items are produced in a cycle, the minimum buffer size required is: $B = 1 + \max_{i=1 \ldots N-1}(k : 0 < k \leq (N-i),\ \min(\lambda_{i+k} - \Lambda_i) \leq 0)$. For the system modeled in Figs. 3 and 4, our timing analysis indicates that: $\lambda_2 - \Lambda_1 \geq 0$, $\lambda_3 - \Lambda_1 \geq 1$, $\lambda_4 - \Lambda_1 \geq 1$, $\lambda_3 - \Lambda_2 \geq 0$, $\lambda_4 - \Lambda_2 \geq 1$, and $\lambda_4 - \Lambda_3 \geq -3$. Therefore, $k = 1$ and the minimum buffer size required is $1 + 1 = 2$. Note that since our analysis is conservative in the worst-case, this is an upper bound on the minimum buffer size. In practice, however, our analysis is exact in this and all the other examples that we have experimented with.

2. Bounds on the cycle time of the system can be obtained by computing the time separation between the start event (when all producers start producing) and the end event (when all items have been consumed). In the example above, we obtain a cycle time ranging from 5 to 12 time units.

3. Bounds on the time spent in the buffer by item i are simply the minimum and maximum values of $(\Lambda_i - \lambda_i)$. Adding the processing time of the consumers to these bounds gives bounds on the total time spent in the system (buffer + consumer) by item i. As an illustration, item 3 in our example spends between 0 and 3 time units in the buffer, and a total of 2 to 7 time units in the whole system.

4. If we sort the events $E1$, $E2$, $E3$ and $E4$ chronologically using techniques similar to those described above (see Fig. 3), we can also obtain bounds on the throughput of the system (ignoring the initial latency until the first item leaves the system). For example, in the example considered above, the time separation between the first and last items leaving the system lies between 2 and 9 time units. This translates to a throughput ranging from $\frac{4}{2} = 2.0$ items/time unit to $\frac{4}{9} = 0.44$ items/time unit.

3.2. A system of concurrent processors

Our second example is motivated by an image processing application in which disjoint sections of an image are independently processed and the results combined to produce the desired effect. The dataflow graph for such an application consists of parallel threads of computation which are eventually merged to produce the result (see Fig. 5). It

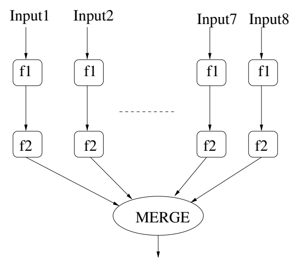

Figure 5. Example dataflow graph

is often convenient to map the computation steps in such an application to a system of concurrently operating processors that synchronize occasionally to ensure correct sequencing of operations in each thread.

In this section, we focus our attention on a multiprocessor architecture in which the processing elements use a common bus for inter-processor synchronization. Data is transferred to and from a shared main memory through separate high bandwidth buses. An example with four processing elements is shown in Fig. 6. We now wish to map the computation depicted in Fig. 5 to the architecture of Fig. 6. We consider two alternative schemes for mapping the computation steps to the processors, and analyze their effects on the synchronization traffic as well as on the overall system performance. For each mapping scheme, we also consider the effect of adding input buffers to the processors. The techniques presented here can be easily extended to analyze other mapping and buffering schemes as well.

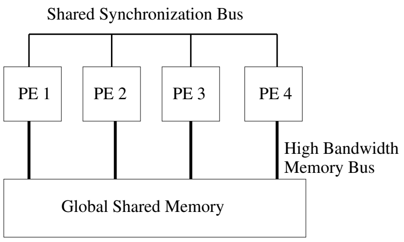

Figure 6. Multiprocessor architecture

We make the following assumptions about the operation of the system. For each thread of computation, the processor that computes the first function (f_1 in Fig. 5) also fetches the input data from main memory. Subsequently, if control transfers to a different processor, the intermediate results are forwarded to this processor by means of an inter-processor synchronization message. The processor that computes the final function in the thread writes the results to main memory and waits for the write to complete. A processor issuing a synchronization message does not wait for it to be processed. Instead, messages are queued at the receiving processor and processed in a FIFO order. For simplicity of analysis, we will assume that (i) each inter-processor synchronization message consumes B units of bandwidth, (ii) each main memory access incurs a delay of 5 to 10 time units, and (iii) function f_1 in Fig. 5 requires 3 to 5 time units to compute, while f_2 requires 1 to 2 time units to compute.

The simplest mapping of the threads in Fig. 5 to the four processors in Fig. 6 assigns a pair of threads to each processor. The timing constraint graph for this system is extremely simple and is not shown here for lack of space. It consists of four chains of events (one for each processor), each originating at the start event (the start of processing), but otherwise disconnected from each other. The chain for processor 1, for example, depicts the sequence of events: Read Input $\rightarrow$ Compute f_1 $\rightarrow$ Compute f_2 $\rightarrow$ Write Result for thread 1 followed by a similar sequence of events for thread 2. The edges in the chain are annotated with appropriate delays indicating the times required for the respective operations. The chains for the other processors are similar. There is no interprocessor synchronization traffic, and the time required for all eight threads to complete (including memory read and write times) is found to lie between 28 and 54 units of time.

Let us now add an input buffer to each processor. Equipped with the buffer, processor 1 issues two simultaneous requests to memory for the input data of threads 1 and 2. Depending on which data arrives earlier, processor 1 starts computing the corresponding thread. After it has written the results of this thread to memory and the input data for the other thread has arrived, processor 1 starts computing the other thread. This timing behavior is depicted in Fig. 7, where we only show the events for processor 1. The behaviors of the other processors are modeled similarly. We find that there is still no synchronization traffic. However, our timing analysis indicates that the time required for all the threads to complete (including memory reads and writes) now ranges from 23 to 44 time units. Thus, buffering has improved the performance of the system without increasing synchronization traffic. This is a direct consequence of the fact that buffering overlaps useful computation with data transfer from memory.

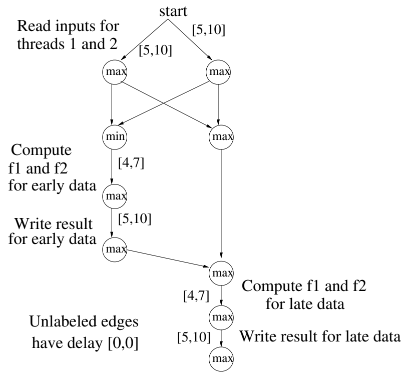

Figure 7. Simple mapping with buffering

Our next scheme is to map threads $1, 2, 3$, and 4 to processors 1 and 2 (see Figs. 5 and 6) in the following manner. For each thread, processor 1 retrieves the input data from memory and computes f_1. It then sends a synchronization message containing the result of f_1 to processor 2, which computes f_2 and writes the result to memory. Threads $5, 6, 7$ and 8 are mapped to processors 3 and 4 in a similar manner. A timing constraint graph modeling the temporal behavior of the processors under this mapping is shown in Fig. 8. Analyzing this graph, we find that the time required for all the threads to complete (including memory reads and writes) now lies between 38 and 72 time units. The intuitive reason for this performance degradation is that f_1 requires more computation time than f_2, and we have a stack of four f_1 computations on processor 1 in Fig. 8, compared to a stack of two f_1 computations and two f_2 computations in Fig. 7. In addition, unlike in the previous cases, we have significant overlap of synchronization traffic here. Our time separations analysis indicates that a maximum of two synchronization messages could overlap in time. Referring to Fig. 8, these overlapping messages could be synch5 and synch1, or synch6 and one of $\{$synch2, synch3$\}$, or synch7 and one of $\{$synch2, synch3, synch4$\}$, or synch8 and one of $\{$synch3, synch4$\}$. Therefore, the bandwidth of the synchronization bus must be at least $2B$.

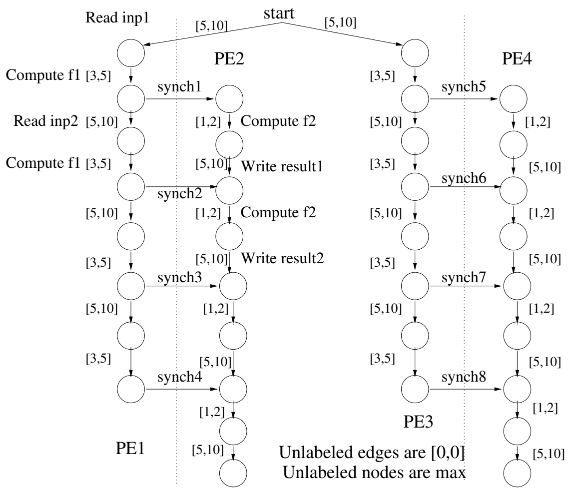

Figure 8. Alternative mapping

Finally, let us consider the effect of adding an input buffer to each processor with the same mapping of computations as above. Processor 1 now simultaneously issues requests to memory for the input data of threads 1 and 2. Depending on which data arrives earlier, it computes f_1 of the corresponding thread and then sends a synchronization message to processor 2. Processor 2 now computes f_2 of this thread and writes the result to memory. In the meanwhile, processor 1 acts on the other data which arrived later from memory, computes f_1 and issues a synchronization message to processor 2, as before. However, it now also issues requests for the input data of threads 3 and 4 and waits for one of these to arrive from memory. The processing of threads 3 and 4 then follows the same protocol as used with threads 1 and 2. The computation in threads $5, 6, 7$ and 8 is mapped to processors 3 and 4 in a similar way. The timing constraint graph of Fig. 9 shows the events occurring in processors 1 and 2. The timing behavior of processors 3 and 4 are similar. Analyzing the complete timing constraint graph, we find that the time required for all the threads to complete (including memory reads and writes) now lies between 32 and 63 time units. A maximum of two synchronization messages can still overlap in time. Thus, buffering helped to improve the performance, but failed to reduce the peak synchronization traffic.

Thus, our original mapping with input buffering appears to be the best choice among the four alternatives considered here. However, addition of input buffers requires additional hardware and the present analysis doesn't take this into account. Note that our algorithm took less than 0.01s to analyze each of these cases on a SUN UltraSparc-1 (143MHz) with 128 MB of memory.

4. Conclusion

Finding bounds on the time separations between events is a fundamental problem in the analysis of concurrent systems. We have previously proposed [4] a polynomial-time approximate algorithm for computing such bounds in

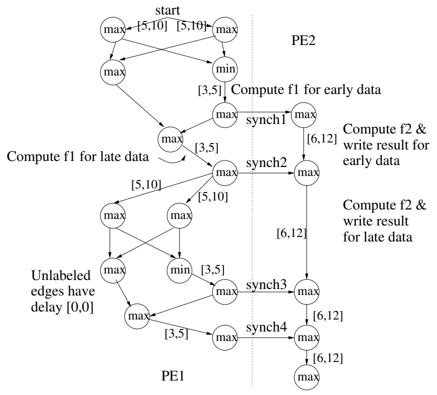

Figure 9. Alternative mapping with buffering

acyclic timing constraint graphs. While the acyclic condition may seem restrictive, it still encompasses a large and interesting class of applications. This paper described applications of our algorithm to two such acyclic systems with inherent concurrency and disjunctive causality. We have also developed a technique to apply our algorithm to certain types of cyclic systems that arise in practice. An application of this technique to an asynchronous differential equation solver chip is described in a companion paper [5].

References

[1] T. Amon and G. Borriello. An approach to symbolic timing verification. In *Proceedings of the 29th DAC*, June 1992.

[2] T. M. Burks and K. A. Sakallah. Min-max linear programming and the timing analysis of digital circuits. In *Proceedings of the ICCAD*, Nov. 1993.

[3] S. M. Burns. *Performance Analysis and Optimization of Asynchronous Circuits*. PhD thesis, California Institute of Technology, 1991.

[4] S. Chakraborty and D. L. Dill. Approximate algorithms for time separations of events. In *Proceedings of the ICCAD*, Nov. 1997.

[5] S. Chakraborty, K. Y. Yun, and D. L. Dill. Practical timing analysis of asynchronous systems using time separation of events. In *Proceedings of TAU*, Dec. 1997.

[6] H. Hulgaard. *Timing Analysis and Verification of Timed Asynchronous Circuits*. PhD thesis, University of Washington, Seattle, 1990.

[7] L. Lavagno and A. Sangiovanni-Vincentelli. Linear programming for optimum hazard elimination in asynchronous circuits. *Journal of VLSI Signal Processing*, 7(1):137–60, Feb. 1994.

[8] T. K. Lee. *A General Approach to Performance Analysis and Optimization of Asynchronous Circuits*. PhD thesis, California Institute of Technology, 1995.

[9] A. R. Martello, S. P. Levitan, and D. M. Chiarulli. Timing verification using hdtv. In *Proceedings of the 27th DAC*, pages 118–173, 1990.

[10] K. L. McMillan and D. L. Dill. Algorithms for Interface Timing Verification. In *Proceedings of the ICCD*, Oct. 1992.

[11] C. J. Myers and T. H.-Y. Meng. Synthesis of timed asynchronous circuits. *IEEE Transactions on VLSI Systems*, 1(2):106–119, June 1993.