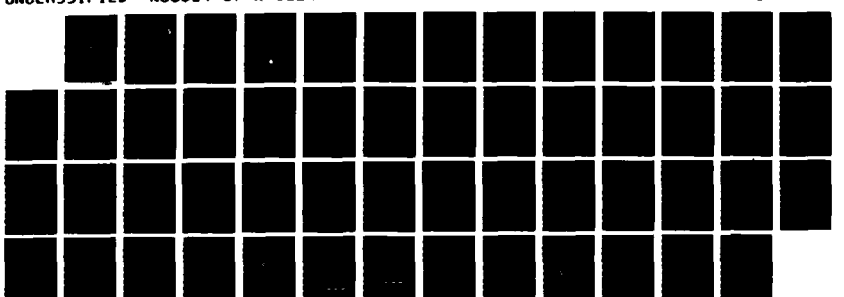
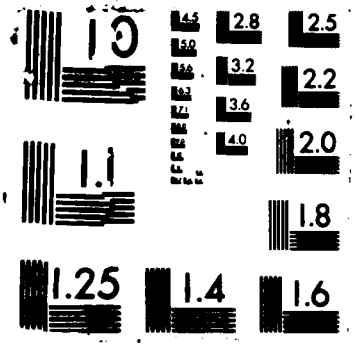
AD-A182 200 EXACT PERFORMANCE ANALYSIS OF TWO DISTRIBUTED PROCESSES 1/1
WITH MULTIPLE SYN (U) MARYLAND UNIV COLLEGE PARK DEPT
OF COMPUTER SCIENCE M ABRAMS ET AL MAY 87 CS-TR-1845
UNCLASSIFIED N00014-87-K-0124 F/G 12/5 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

UNCLASSIFIED

FILE COPY

REPORT DOCUMENTATION PAGE

AD-A182 200

	1b. RESTRICTIVE MARKINGS N/A
N/A	3. DISTRIBUTION / AVAILABILITY OF REPORT approved for public release; distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CS-TR-1845	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION University of Maryland 6b OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research
6c. ADDRESS (City, State, and ZIP Code) Dept. of Computer Science University of Maryland College Park, MD 20742	7b. ADDRESS (City, State, and ZIP Code) 800 North Quincy Street Arlington, VA 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-87-K-0124
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

DTIC ELECTE JUN 30 1987 S E

11. TITLE (Include Security Classification)
Exact Performance Analysis of Two Distributed Processes with Multiple Synchronization Points

12. PERSONAL AUTHOR(S)
Marc Abrams and Ashok K. Agrawala

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM N/A TO	May 1987	46

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

We describe a technique to exactly analyze the run-time behavior of a class of distributed computer programs composed of two processes and an arbitrary number of synchronization points. A process is viewed as a flow chart without branches, whose nodes represent local computation requiring a deterministic time to execute, and whose arcs correspond to interprocess communication. Each synchronization point is a constraint prohibiting a transition in one flow chart when the other process is executing a certain sequence of nodes in its flow chart. The global program state specifies at which flow-chart node each process is currently executing or blocked. We define a steady-state behavior is to be a finite sequence of global states and transitions repeated forever. We solve for the steady state using a Diophantine equation (whose coefficients and unknowns are integers) derived from the geometric concurrency model. The geometric model uses a two-dimensional Cartesian product of nonnegative real numbers to represent all feasible global-system states. The sequence of nodes in each flow chart is mapped to consecutive

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

intervals along each axis. Various lines through the plane correspond to various run-time behaviors. The lines are obstructed by line segments whose spatial arrangement in the plane corresponds to the synchronization constraints. The model solution yields the sequence of synchronization points where the program blocks during steady state, the blocking durations, and the duration of concurrent execution between synchronization points. An algorithm is presented to solve the chief numerical problem of evaluating the minima of two arithmetic functions arising in the analysis.

(12)

S-TR-1845 May 1987

EXACT PERFORMANCE ANALYSIS OF TWO DISTRIBUTED PROCESSES WITH MULTIPLE SYNCHRONIZATION POINTS*

Marc Abrams and Ashok K. Agrawala

COMPUTER SCIENCE TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

87 6 9 102

CS-TR-1845 May 1987

EXACT PERFORMANCE ANALYSIS OF TWO DISTRIBUTED PROCESSES WITH MULTIPLE SYNCHRONIZATION POINTS*

Marc Abrams and Ashok K. Agrawala

Department of Computer Science
University of Maryland
College Park MD 20742

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

QUALITY INSPECTED

(*) This research was supported in part by a contract (N00014-87-K-0241) from The Office of Naval Research to The Department of Computer Science, University of Maryland

Exact Performance Analysis of Two Distributed Processes with Multiple Synchronization Points

MARC ABRAMS AND ASHOK K. AGRAWALA
Department of Computer Science, University of Maryland, USA

Abstract. We describe a technique to exactly analyze the run-time behavior of a class of distributed computer programs composed of two processes and an arbitrary number of synchronization points. A process is viewed as a flow chart without branches, whose nodes represent local computation requiring a deterministic time to execute, and whose arcs correspond to interprocess communication. Each synchronization point is a constraint prohibiting a transition in one flow chart when the other process is executing a certain sequence of nodes in its flow chart. The global program state specifies at which flow-chart node each process is currently executing or blocked. We define a steady-state behavior to be a finite sequence of global states and transitions repeated forever. We solve for the steady state using a Diophantine equation (whose coefficients and unknowns are integers) derived from the *geometric concurrency model*. The geometric model uses a two-dimensional Cartesian product of nonnegative real numbers to represent all feasible global-system states. The sequence of nodes in each flow chart is mapped to consecutive intervals along each axis. Various lines through the plane correspond to various run-time behaviors. The lines are obstructed by line segments whose spatial arrangement in the plane corresponds to the synchronization constraints. The model solution yields the sequence of synchronization points where the program blocks during steady state, the blocking durations, and the duration of concurrent execution between synchronization points. An algorithm is presented to solve the chief numerical problem of evaluating the minima of two arithmetic functions arising in the analysis.

Categories and Subject Descriptors: D.2.8 [**Software**]: Metrics – *Performance measures;* D.4.8 [**Operating Systems**]: Performance – *Modeling and prediction;* C.4 [**Performance of Systems**]: Modeling techniques

General Terms: Distributed Programs, Performance, Synchronization

Additional Key Words and Phrases: Diophantine equation, geometric concurrency model

---

This research was supported by grant NSF-DCR-8405235 from the National Science Foundation.

Authors' present addresses: M. Abrams, IBM Research Division, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland; A.K. Agrawala, Department of Computer Science, University of Maryland, College Park, MD 20742, USA.

February 9, 1987

1. Introduction

The state of many continuous physical systems (e.g., an electrical circuit) may reach a fixed value as time grows to infinity. Alternately, the system may reach a limit cycle behavior, in which the stable system state varies periodically in time. In this paper, we analyze the dynamics of a class of distributed programs that displays similar behavior. We consider a distributed program to be a collection of asynchronous, sequential processes that occasionally synchronize or communicate by messages.

The class of programs we analyze meets the following assumptions. Programs consist of two processes. All events in the program take constant time. Processes execute asynchronously on separate hardware processors. Processes communicate and synchronize through messages, for example, by *send* and *receive* primitives. The synchronization constructs in each process cannot be embedded in conditionally executed pieces of code. Finally, each process loops without termination.

We view each process as a flow chart without branches. Each flow-chart node corresponds to the instructions or statements in the code which are executed between two consecutive *send/receive* primitives. Thus, flow-chart arcs correspond to a message being sent or received.

Each flow-chart node is a *local state* of the corresponding process. The *global state* of the program consists of the local state of each process. The time required to execute a state is the *state occupancy time*. We describe an execution behavior of a program by the sequence and occupancy times of global states occurring during execution, called the *program trajectory*. A program is in *steady state* when it repeats a finite sequence of global states and occupancy times forever.

Given the constant local state occupancy times and the communication delay between processes, we solve for the trajectory. The trajectory yields, at steady state, the values of the:

(1) sequence of synchronization points where the program blocks;
(2) blocking duration at each synchronization point, and
(3) duration of concurrent execution between synchronization points.

2. Related Work

Alternate techniques to model distributed program performance are queueing networks (e.g., [7, 8]), stochastic processes (e.g., [6, 12]), Petri nets (e.g., [16]), and stochastic automata ([13, 14]). A survey of these approaches is contained in [1], Chapter 1.

These techniques generally seek *average* values of performance measures at a steady state. However, it may be difficult to understand *why* the program produces these average values. The motivation for our work to obtain the trajectory is to gain insight into how to change the program to improve performance.

The *geometric concurrency model* is the basis for our analysis. The model is similar to Dijkstra's *process graph* [4] for characterization of deadlocks in multiprocessor systems. The geometric model is also similar to two-dimensional diagrams used in verification of parallel programs and communication protocols to illustrate interleaving of operations.

The geometric model has been analyzed by Papadimitriou and others [10, 11, 17] to detect deadlocks in a lock-based database transaction system with two transactions. The model was later used by Carson [2, 3] to prove liveness properties of concurrent programs composed of an arbitrary number of processes with straight-line sequences of semaphore operations.

We use the geometric model for performance analysis, in contrast to proving correctness and liveness properties. Moreover, we *analytically* solve the model, while prior work devised an algorithm (see Section 5). Advantages of an analytic solution include speed in model solution, and insight into program performance. However, we require additional assumptions of uniprogramming and constant execution times.

We next discuss mapping a program to the geometric model. We then derive a steady-state equation from the model, and solve for the trajectory. Finally, we present an algorithm which numerically calculates the steady state, and then illustrate its use.

3. Geometric Concurrency Model

We introduce the geometric model through an example: the dining philosophers' problem [5]. In this problem, two philosophers eating a meal share one set of chopsticks. Each philosopher alternately thinks and eats. If one philosopher wants to acquire the chopsticks while the second is eating, the first must wait. The problem can be modeled by a program containing one process for each philosopher.[1]

The corresponding two flow charts are shown in Figure 1(a), each containing two states. We consider the chopsticks to be held by a process (and hence unavailable to the other process) exactly when a process is in its EAT/RELEASE state. Therefore a process blocked at the transition out of its THINK/ACQUIRE state will unblock the moment the other process leaves its EAT/RELEASE state.

We denote the real and integer numbers by R and Z, respectively. A superscript of * or + (e.g., R^*, Z^+) restricts R or Z to nonnegative and positive numbers, respectively. The notation in the table below is introduced solely to formalize the geometric model, and (except for t_n^r) is not used outside of this section.

---

[1] A third process is required to receive requests for the chopsticks and to insure mutually exclusive chopstick use. We do not explicitly model this process here, in the interest of simplicity. However, the execution time of this process can be included at appropriate points in the philosopher process flow charts, as illustrated in [1].

Symbol	Meaning
$r \in \{1,2\}$	denotes either process 1 or 2
$M_r \in Z^*$	number of local states in process r
$n, n', n'' \in \{0,1,..., M_r - 1\}$	denotes one of the M_r states in process r
σ_n^r	local state in process r, such that $\forall n, \sigma_{n+1 \bmod M_r}^r$ is the successor[2] of state σ_n^r in the flowchart of process r
$t_n^r \in R^+$	state occupancy time of σ_n^r
$q \in Z^*$	each process r cycles through states $\sigma_0^r, \sigma_1^r, \ldots, \sigma_{M_r}^r$ forever, therefore we can refer to the q^{th} time r occupies state σ_n^r

This notation is illustrated in Figure 1(b). One additional definition we need is the *cycle time* of a process:[3]

$c_r = \sum_{n=0}^{M_r - 1} t_n^r$. For our example, $c_1 = 5$ and $c_2 = 3$.

Mapping states to the model. In the geometric model, each process is associated with one axis of the Cartesian product $R^* X R^*$. Intuitively, the sequence of states each process passes through is mapped to a sequence of intervals along its corresponding axis. The interval width corresponds to the occupancy time of the corresponding state. This is illustrated in Figure 2.[4]

---

2 This definition reflects the fact that flow charts do not branch and processes loop forever.

3 The cycle time is the time required to execute one loop of a process in the absence of blocking.

4 Although we show a finite portion of the plane, the reader should bear in mind that each axis extends to infinity.

Formally, the q^{th} occupancy of state σ_n^r is mapped to the interval $[\ q\,c_r + \sum_{z=0}^{n-1} t_z^r\ ,\ \ q\,c_r + \sum_{z=0}^{n} t_z^r\ )$ along the axis corresponding to process p: In Figure 2 the first, second, and third occupancies of the EAT/RELEASE state by process 2 are mapped to intervals [2,3), [5,6), and [8,9) along the y axis.

The dashed lines in Figure 2, drawn at each state boundary, partition the plane into a set of rectangles. Each rectangle potentially represents one global state. Points within a rectangle represent all possible partial executions of the global state.

Mapping program loops to the model. The heavy lines drawn every c_1 units along the x axis and every c_2 units along the y axis in Figure 2 partition the plane into *quadrants*. Each quadrant is uniquely identified by a pair of nonnegative integers, i and j. A point with x coordinate x_0 and y coordinate y_0 is in quadrant i,j, where $i = \lfloor x_0/c_1 \rfloor$ and $j = \lfloor y_0/c_2 \rfloor$ (see Figure 3). During execution, we can keep track of how many times each process has looped through its code. This pair of integers corresponds to a unique quadrant.

Mapping synchronization to the model. Synchronization between processes is represented as a finite set of *constraints* on when transitions in a flow chart can occur. Each constraint specifies that a certain transition in one flow chart must be delayed — and therefore the process must block — until the other process leaves some sequence of states in its flow chart.

Returning to Figure 1, there are two constraints, indicated by dashed lines. Process 1 (or 2) cannot enter the EAT/RELEASE state if process 2 (or 1) is in the EAT/RELEASE state.

The geometric representation of a synchronization constraint is a set of horizontal or vertical line segments in the plane placed at appropriate boundaries of states to prohibit a transition into those states from occurring. This is formalized as follows: Consider the synchronization constraint in which process 1 cannot make the transition from state σ_n^1 to its successor

while process 2 is occupying any of the sequence of states $\sigma^2_{n'}$, $\sigma^2_{n'+1}$, $\sigma^2_{n'+2}$, ... , $\sigma^2_{n''-1}$, $\sigma^2_{n''}$, for $n' \leq n''$. This constraint is mapped to the following set of line segments:

$$\{ (x,y) \mid i,j \in Z^*; \quad x = ic_1 + \sum_{z=0}^{n} t^1_z ; \quad jc_2 + \sum_{z=0}^{n'-1} t^2_z \leq y < jc_2 + \sum_{z=0}^{n''} t^2_z \} .$$

(The complementary case, of a constraint that delays transition from state σ^1_n to its successor, is mapped to an expression similar to the above, in which are interchanged variables x and y, variables i and j, and superscripts and subscripts 1 and 2.)

Figure 4 illustrates the placement of constraint line segments for the example of Figure 1. The set of line segments representing the constraint out of σ^1_0 is $\{ (x,y) \mid i,j \in Z^*; \quad x = 5i + 1; \quad 3j + 2 \leq y < 3j + 3 \}$.

Mapping program trajectories to the model. Earlier, we defined the program trajectory to be the sequence of timings of global-state transitions a program makes. This corresponds in the model to the *model trajectory*, which is a possibly infinite sequence of rays[5] in the R^*XR^* plane. Given a point Q_1 in the plane, the model trajectory starting at Q_1 is recursively defined as follows:

- If Q_1 lies on a constraint line, the model trajectory consists of a ray from Q_1 to Q_2 followed by a model trajectory starting at Q_2. Q_2 is the end point furthest from the origin of the constraint line containing Q_1.
- Otherwise, Q_1 does not lie on a constraint line. A ray with slope 1 and initial end point Q_1 either: (1) will, or (2) will not eventually intersect a constraint line. In case (1), the trajectory consists of this ray alone, of

---

[5] A *ray* is a directed line segment of possibly infinite length. A ray is directed away from its *initial end point*. In terms of a program, the ray direction corresponds to the evolution of the program forward in time.

infinite magnitude. In case (2), let Q_3 be the point at which the ray first intersects any constraint line. The trajectory then consists of a ray from Q_1 to Q_3 followed by a trajectory starting at Q_3.

Informally, in the absence of synchronization constraints, the program trajectory is represented by an infinite ray (Figure 2). The ray slope of 1 represents the fact that both processes execute at the same speed and never impede one another. Each way in which the diagonal may slice through a global-state rectangle corresponds to different state occupancy times. The global state corresponding to the end point of the vector is the initial program state. The constraint lines alter the model trajectory in a manner which represents the sequence of blockings encountered by the program during execution. Blocking is represented by a vertical or horizontal ray in the trajectory, corresponding to the case when only process 2 or process 1, respectively, can execute. Figure 4 illustrates one possible trajectory.

Mapping initial conditions to the model. The geometric model can represent any *initial condition* of a program. An initial condition consists of two real numbers, representing the time at which each process is started. Each initial condition corresponds to one point on either the x axis in the interval $[0,c_1)$, when process 1 is started first, or to a point on the y axis in the interval $[0,c_2)$, when process 2 is started first. The origin corresponds to the case when both processes start simultaneously. Henceforth, in drawing the geometric model, we shall show the result of drawing the entire set of possible trajectories for *all* initial conditions by a shaded region, as illustrated in Figure 5.

Manifestation of race conditions in the model. In our example, a *race condition* occurs when both processes simultaneously request the chopsticks. In the geometric model, a race condition manifests itself as a single point which is the end point closest to the origin of both a horizontal and a vertical constraint line (Figure 6). The behavior of any trajectory intersecting

this common end point is undefined, because we assume that any two events in a distributed system are separated by a nonzero amount of time.

Manifestation of deadlocks in the model. A *deadlock* is a global state in which both processes are blocked. In the geometric model, deadlock manifests itself as a trajectory which reaches a point in which it cannot make progress by moving right, diagonally, or up (Figure 7). Trajectories in the shaded region will deadlock. However, trajectories corresponding to initial conditions outside this region will not deadlock. We assume the program being modeled is deadlock-free, this problem having been solved by Lipski and Papadimitriou [10] and Carson [3].

Manifestation of steady state in the model. In terms of the geometric model, the steady state is represented by a trajectory consisting of either a diagonal ray of infinite length or a finite sequence of possibly diagonal, horizontal and vertical rays repeated forever.

THEOREM 1. *In the absence of deadlock and race conditions, a program fitting our model always reaches steady state.*

PROOF. Let k denote some constraint line. Consider the diagonal trajectory ray leaving the end point furthest from the origin of k. The ray either will intersect exactly one constraint line k′, because we prohibit race conditions, or it will never intersect a constraint line. If no intersection occurs, the program will repeat a sequence of global states forever, thereby reaching steady state. Otherwise constraint k′ may have the diagonal trajectory rays of several other constraint lines intersect it, but the trajectory of all of these will be mapped to the end point of constraint k′ furthest from the origin. Therefore, we can construct a finite, directed graph in which each node corresponds to one constraint line. The graph contains a directed arc from the node corresponding to constraint k to the node corresponding to constraint k′ if and only if the trajectory leaving constraint k intersects k′. All cycles in the graph are of finite length and represent steady states. □

Modeling Other Forms of Synchronization. Because we are analyzing programs with reusable resources, the number and spatial arrangement of constraint lines in each quadrant is identical. We can represent more general synchronization semantics by slightly generalizing our definition of "constraint" to distinguish the *first, second,* etc. transitions of a flow-chart node. In the geometric model, the constraint lines will cross the boundaries of quadrants. The general case is discussed in [1].

However, we restrict our attention to reusable resources because this case is more difficult to analyze than other types of synchronization. This is because we only know the global state at the moment both processes start execution. The asynchronous nature of the processes precludes *a priori* knowledge of the global state at any later time.

4. Formal Problem Statement

The fundamental analysis problem treated here follows. The geometric model consists of the following input parameters:

- The cycle time of each process: $c_1 \in R^+$ and $c_2 \in R^+$.
- The number of constraint lines in each quadrant, $N \in Z^+$.
- For each constraint line $k \in \{1,2,\ldots,N\}$:[6]
  - the abscissa of the top or right end point, $x_k \in R^+$,
  - the ordinate of the top or right end point, $y_k \in R^+$,
  - the length, $l_k \in R^+$; by convention l_k is positive if and only if line k is vertical.[7]

---

[6] Recall that each quadrant contains the same number, lengths, and spatial arrangement of constraint lines. Each constraint line in a quadrant is identified by an integer $k \in [1,N]$. Thus, when we refer to "constraint line k," we mean a replica of the k^{th} constraint line in *any* quadrant. To uniquely identify a constraint line in the plane, we must specify *both* its index k and the number i,j of the quadrant containing the line.

[7] Note that the *sign* of parameter l_k and function w, introduced below, encode which process blocks. This avoids the need for additional variables.

The output of the geometric model is two functions, $p: \{1,2,\ldots,N\} \times \{1,2,\ldots,N\} \rightarrow R^*$ (parallel) and $w: \{1,2,\ldots,N\} \times \{1,2,\ldots,N\} \rightarrow R^*$ (wait). These are defined as follows:

- *In terms of the program:* Consider the moment both processes have just completed synchronization at constraint k, and ignore all synchronization constraints *except* k'.
  - **p(k,k')** Yields the length of time both processes execute in parallel before synchronizing at constraint k'. If the processes never synchronize, $p(k, k') = \infty$.
  - **w(k,k')** Represents the time one process spends blocked or waiting at constraint k'.
- *In terms of the geometric model:* Suppose we draw a diagonal trajectory starting at the end point of k furthest from the origin until the trajectory intersects *any* replica of constraint k'.
  - **p(k,k')** Yields the trajectory length, projected on either the x or y axis. If the diagonal threads its way between replicas of constraint lines k' in the plane forever, without intersection, $w(k, k') = \infty$.
  - **w(k,k')** Yields the distance between the intersection point of the trajectory and constraint k' and the end point of k' furthest from the origin. If no intersection exists, $w(k, k') = 0$. The sign convention of $w(k, k')$ is the same as used for $l_{k'}$. Thus, negative w indicates process 2 is waiting.

To find the sequence of constraint lines the steady-state trajectory intersects, we find that value of k' which minimizes $p(k, k')$ for each k. From Theorem 1, this sequence will contain one or more cycles, each cycle being a steady-state behavior.

Example. In Figure 5, let the vertical constraint line be indexed $k = 1$, and the horizontal line $k = 2$. The input parameters to the geometric model are $c_1 = 5$, $c_2 = 3$, $N = 2$, $(x_1, y_1, l_1) = (1,3,1)$, and $(x_2, y_2, l_2) = (5,2,-4)$. The output functions from the geometric model, are:

k	k'	p(k,k')	w(k,k')
1	1	5	1
1	2	2	-2
2	1	6	1
2	2	3	-2

Because $p(1,2) < p(1,1)$, when the program leaves constraint 1 it next synchronizes at constraint 2. Because $p(2,2) < p(2,1)$, when the program leaves 2 it synchronizes next at constraint 2. Therefore, there is only one steady-state cycle, consisting of synchronization repeatedly at constraint 2. Further, the steady-state trajectory consists of a diagonal ray whose length projected on either axis is $p(2,2) = 3$ followed by a horizontal[8] ray of length $|w(2,2)| = 2$ time units. In terms of the program, in steady state both processes run in parallel for 3 time units, then process 2 blocks for 2 time units.

5. *Model Solution Alternatives*

A difficult aspect of this work was discovering a simple and straightforward framework within which to describe and solve the model. Thus, we considered three alternate approaches:

(1) *Computational geometry algorithm:* This draws the graphs of the model (e.g., Figure 5). It is the approach used in the geometric model-based

---

[8] The ray is horizontal because the sign of w(2,2) is negative.

deadlock detection algorithms in [10] in two dimensions, and in [3] in N dimensions. In our algorithm, we step a "sweep-line" [15] across the plane. We keep track of what points or intervals of points along the sweep line correspond to active trajectories or shaded regions. At each step, this information is updated depending on which constraint lines intersect the sweep line.

(2) *Integer programming.* For each pair of constraints k and k', we minimize the length of the diagonal trajectory ray with one end point on constraint k, subject to the condition that the x and y coordinates of the other end point of the trajectory lie between the x and y coordinates of the end points of some replica of constraint k'.

(3) *Analytic solution.* This is discussed in successive sections.

The first two methods share three deficiencies:

(1) *Potentially infinite search space.* Neither method will terminate in a solution if the diagonal trajectory leaving a constraint k never intersects another constraint.

(2) *Time wasted searching infeasible points.* In integer programming, each replica of constraint k' examined may or may not be a feasible solution. In the computational geometric algorithm, some points and intervals of points along the sweep line do not need updating because no intersection occurs there. These may be thought of as "infeasible solutions."

(3) *Limited insight into relationship of model inputs and outputs.* We want to qualitatively understand the relationship of model outputs upon inputs. For example, how does constraint line length affect waiting time, and what influences the number of steady-state cycles?

These deficiencies are overcome through the analytic solution presented next. The analytic solution is based on the observation that because each quadrant contains a replica of the same constraint lines, we ought to be able to analyze equivalently either a single quadrant or the entire plane. This duality manifests itself naturally in the mathematics through modular arithmetic.

A complete analytic solution to the problem requires us to obtain an analytic expression for two minimizations of arithmetic functions occurring in the analysis. These are challenging in and of themselves. So far, we have worked out the case of $N = 1$ [1]. In this paper, we solve the general case by evaluating numerically the minimum in Section 7. This numerical method, in comparison to the computation geometric and integer programming algorithms, examines only a *finite* number of *feasible* solutions. Therefore, it is guaranteed to terminate in a solution, avoiding the first two deficiencies listed above.

6. *Analytic Solution*

Consider an arbitrary constraint line k. To obtain the steady-state behavior, we must solve two problems:

(1) Find the *necessary and sufficient condition* for the diagonal trajectory ray leaving k to intersect any constraint line, say k'.
(2) Provided the intersection with k' occurs, find the *quadrant* in which the first intersection with k' occurs, from which functions p(k, k') and w(k, k') may be determined.

In the process of solving the first problem, we shall formulate an equation solving the second. A solution will exist if and only if an intersection occurs. The equation is formally derived in Chapter 5 of [1], however in the interest of space the equation is informally presented below.

Consider the case when $k = k'$, as illustrated in Figure 8(a). The diagram shows that after the program has completed synchronization at constraint k, process 1 makes two complete loops before it must block, and blocking starts within the third loop of process 2.

An alternate view is given by the time lines shown in Figure 8b. Time zero represents the moment both processes leave synchronization constraint k. Process 2 can never block, because the constraint is vertical; thus process 2 constrains process 1 in time intervals $[c_2 - l_{k'}, c_2)$, $[2c_2 - l_{k'}, 2c_2)$, ... Further, process 1 will request the resource at times c_1, $2c_1$, $3c_1$,... *until* some time that is an integral multiple of c_1 and lies within the interval when process 2 constrains process 1. Therefore, blocking occurs if and only if

$$\exists \delta_1 \in Z^+, \ \exists \delta_2 \in Z^+, \ \ni \ c_1 \cdot \delta_1 \in [c_2\delta_2 - l_{k'}, \ c_2 \cdot \delta_2) \ . \qquad (0)$$

For example, in Figure 8a $\delta_1 = 2$ and $\delta_2 = 3$ satisfy (0). In terms of the program, the *smallest* δ_1 and δ_2 satisfying (0) represents the number of loops processes 1 and 2 make, respectively, before synchronization. In terms of the geometric model, the smallest δ_1 (δ_2) yields the number of quadrants which a trajectory leaving constraint k and intersecting constraint k' moves through horizontally (vertically). Therefore, if constraint k lies in quadrant i,j and constraint k' lies in quadrant i', j', then

$$\delta_1 = i' - i$$
$$\delta_2 = j' - j \ .$$

Expression (0) is equivalent to the condition that blocking occurs if and only if

$$\exists \delta_1 \in Z^+, \ \exists \delta_2 \in Z^+, \ \exists b \in (0, l_{k'}] \ \ni \ c_1 \cdot \delta_1 - c_2 \cdot \delta_2 + b = 0 \ .$$

where b assumes real values. Because a different value of δ_1 and δ_2 will satisfy the above equation for each value of k and k′, we shall henceforth denote δ_1 and δ_2 by $\delta_1(k, k')$ and $\delta_2(k, k')$, respectively. Furthermore, from Figure 8(b) the quantity b is seen to be the blocking time w(k, k′). Therefore, the above equation becomes

$$c_1 \cdot \delta_1(k, k') - c_2 \cdot \delta_2(k, k') + w(k, k') = 0 \ , \tag{1a}$$

where

$$w(k, k') \in (0, I_{k'}] \ . \tag{1b}$$

Equation (1) must be modified for the case of $k \neq k'$, as illustrated in Figure 9(a). Here, the horizontal constraint line is k, and the vertical line k′. Because the trajectory starts at constraint k and not k′ implies that the origin of each time axis in Figure 9(b) is shifted by the difference in the x and y coordinates of the end points of constraint lines k and k′ furthest from the origin. Thus, the steady-state equation becomes:

$$c_1\delta_1(k, k') - c_2\delta_2(k, k') + w(k, k') + (y_k - x_k) - (y_{k'} - x_{k'}) = 0 \ . \tag{2}$$

Equation (2) is the central equation of this paper. We shall shortly show how to obtain p(k, k′) from $\delta_1(k, k')$ and $\delta_2(k, k')$. Three difficulties arise in solving eq. (2):

(1) $\delta_1(k, k')$ and $\delta_2(k, k')$ can only be integers, although all other variables and functions in eq. (2) are real quantities.

(2) We must find when a solution exists, and the number and form of the solutions.

(3) Two minimization problems are involved:

- Find which value of w(k, k') in the range (1b) minimizes $\delta_1(k, k')$ and $\delta_2(k, k')$ for every k and k'.[9]
- From $\delta_1(k, k')$ and $\delta_2(k, k')$, we shall find p(k, k'). To find the sequence of constraints the trajectory intersects, we must find the minimal value of p(k, k') for each k, over all values of k'.

The first two problems are solved below by transforming eq. (2) into a Diophantine equation,[10] and applying known methods. However, the third problem is difficult. To the authors' knowledge finding minimum integer solutions is not addressed in the literature. Solutions to these problems are presented next.

Earlier, we defined c_1 and c_2 to be real quantities. For the following analysis, we shall assume that c_1 and c_2 are rational quantities,[11] or equivalently *relatively prime integers*. To demonstrate the equivalence, we can multiply all input parameters (c_1; c_2; and $\forall k \in [1,N]$ x_k, y_k, l_k) by the factor L/G, where L is the least common denominator (l.c.d.) of c_1 and c_2, and G is the greatest common divisor (g.c.d.) of c_1 and c_2. Further, we must multiply output functions w(k, k') and p(k, k') by G/L. These substitutions result in the multiplication of each term in eq. (2) by the factor L/G.

---

[9] If eq. (2) has a solution, it will not be unique. In Figure 9(a), the *first* time process 1 leaves constraint k and intersects constraint k', process 1 will wait. This corresponds to the *minimum* number of loops each process makes before synchronizing, and hence the minimum positive integers $\delta_1(k,k')$ and $\delta_2(k,k')$ satisfying eq. (2). Larger values of $\delta_1(k,k')$ and $\delta_2(k,k')$ correspond to the second and successive times process 1 leaves constraint k and intersects constraint k'.

[10] A Diophantine equation has integer coefficients and unknowns, as discussed in [9].

[11] The subsequent results do not apply to processes whose cycle time is irrational. Although this case is theoretically interesting, in practice we measure programs using only rational numbers (e.g., all measurements have a precision of one microsecond).

Because the quantities $\delta_1(k, k')$, $\delta_2(k, k')$, c_1, and c_2 are integers, the expression $w(k, k') + (y_k - x_k) - (y_{k'} - x_{k'})$ must also be an integer. We represent this expression by a new function:

$$\hat{w}(k, k') = w(k, k') + (y_k - x_k) - (y_{k'} - x_{k'}) \tag{3}$$

then eq. (2) may be simplified to the following Diophantine equation:

$$c_1\delta_1(k, k') - c_2\delta_2(k, k') + \hat{w}(k, k') = 0 \ . \tag{4}$$

We can establish bounds of the value of $\hat{w}(k, k')$ based on the bounds of $w(k, k')$ in (1b):

$$\hat{w}(k, k') \in (\quad (y_k - x_k) - (y_{k'} - x_{k'}), \quad (y_k - x_k) - (y_{k'} - x_{k'}) + l_{k'} \quad] \ . \tag{5}$$

6.1 FINDING CONDITION FOR INTERSECTION. Theorem 3.3 in [9] states that:

> A necessary and sufficient condition for the Diophantine equation $a_1z_1 + a_2z_2 + \ldots + a_nz_n = d$ to have a solution is that the g.c.d. of the a_i's divides k.

By this theorem, if we consider for a moment $\hat{w}(k, k')$ to be an integer constant d, then the equation $c_1\delta_1(k, k') - c_2\delta_2(k, k') + d = 0$ has a solution if and only if the g.c.d. of c_1 and c_2 divides d. Because c_1 and c_2 are relatively prime, a solution exists if and only if d can assume an integer value in the interval of (5).

This establishes the following theorem. In the theorem, $\Omega(k)$ represents a set whose members are indices of constraint lines, in [1,N]. Index $k' \in \Omega(k)$ if and only if a diagonal trajectory ray leaving constraint k intersects constraint k', in the absence of other constraints.

THEOREM 2. *Consider a program fitting our model containing constraint lines 1,2,...,N replicated in every quadrant. Let* k, $k' \in [1,N]$. *Then* $k' \in \Omega(k)$ *if and only if the interval* $(\quad (y_k - x_k) - (y_{k'} - x_{k'}), \quad (y_k - x_k) - (y_{k'} - x_{k'}) + l_{k'} \quad]$ *contains at least one integer.*

Example. For the program in Figure 1, by Theorem 2 $\Omega(1) = \{1,2\}$ and $\Omega(2) = \{1,2\}$. For example, $2 \in \Omega(1)$ because the interval $[-3,1)$ contains integers (namely, -3 and -2).

6.2 FINDING FUNCTIONS P AND W. Next, we solve eq. (4) for unknowns $\delta_1(k, k')$, $\delta_2(k, k')$, and $\hat{w}(k, k')$. We apply the solution technique for three variable Diophantine equations in [9], pp. 67-68. There is an infinite number of solutions, and these are expressed in terms of a parameter α as follows:

$$\delta_1(k, k') = c_2\alpha + v \cdot \hat{w}(k, k') \tag{6a}$$

$$\delta_2(k, k') = c_1\alpha + u \cdot \hat{w}(k, k') \ , \tag{6b}$$

where $\alpha \in Z$, and u and v are integers satisfying[12]

$$c_2u - c_1v = 1 \ . \tag{7}$$

In terms of a program, there is a unique solution, corresponding to that value of $\hat{w}(k, k')$ minimizing $\delta_1(k, k')$ and $\delta_2(k, k')$. Rather than finding this, we obtain a relation between $p(k, k')$ and $\delta_1(k, k')$ and $\delta_2(k, k')$ in the following lemma, and then minimize $p(k, k')$.

---

[12] A solution to eq. (7) may be found by solving the following linear congruence by known methods: $c_2u \equiv 1 \pmod{c_1}$.

LEMMA

$$p(k, k') = \begin{cases} \infty & \text{if } k' \notin \Omega(k) \\ c_1\delta_1(k, k') + x_{k'} - x_k & \text{if } k' \text{ is vertical} \wedge k' \in \Omega(k) \\ c_2\delta_2(k, k') + y_{k'} - y_k & \text{if } k' \text{ is horizontal} \wedge k' \in \Omega(k) \end{cases}$$

PROOF. There are three forms for $p(k, k')$ in the statement of the lemma; we shall consider each case in turn.

Case 1: $k' \notin \Omega(k)$

By definition $p(k, k') = \infty$.

Case 2: k' is horizontal and $k' \in \Omega(k)$

Constraint k may be either horizontal or vertical; this has no bearing on the proof. The length of the projection of the trajectory on either the x or y axis is the same. We shall choose the y axis, as illustrated in Figure 10, to avoid calculating the coordinates of the intersection point. Assume constraint k is in quadrant i,j. The y coordinate of *all* points of constraint k', including the intersection point with the trajectory, is $y_{k'} + c_2 j'$. Assume constraint k' is in quadrant i', j'; its end point furthest from the origin is $y_k + c_2 j$. Their difference, $c_2(j' - j) + y_{k'} - y_k = c_2\delta_2(k, k') + y_{k'} - y_k$, is the y axis projection we seek.

Case 3: k' is horizontal and $k' \in \Omega(k)$

Transpose x and y, i and j, i' and j', 1 and 2, and "vertical" and "horizontal" in the previous case. □

Now, using eq. (6) we eliminate $\delta_1(k, k')$ and $\delta_2(k, k')$ from the lemma and obtain our final form for $p(k, k')$.

THEOREM 3. *Consider a two-process program fitting our model containing constraint lines 1,2,...,N replicated in each quadrant. Let* k, k′ ∈ [1,N] *and* k′∈Ω(k). *Then the length of the perpendicular projection on the* x *or* y *axis of the diagonal trajectory ray between constraint* k *and its first intersection with constraint* k′ *is given by the following expression:*

$$p(k, k') = \begin{cases} c_1\left(v\cdot\hat{w}(k, k') - c_2\left\lfloor\frac{c_1 v\cdot\hat{w}(k, k') + x_{k'} - x_k}{c_1c_2}\right\rfloor\right) + x_{k'} - x_k & \text{if } k' \text{ is vertical} \\ c_2\left(u\cdot\hat{w}(k, k') - c_1\left\lfloor\frac{c_2 u\cdot\hat{w}(k, k') + y_{k'} - y_k}{c_1c_2}\right\rfloor\right) + y_{k'} - y_k & \text{if } k' \text{ is horizontal} \end{cases}$$

where $\hat{w}(k, k')$ *is the value in the range of (5) yielding the minimum nonnegative value for* p(k, k′).

PROOF. Suppose constraint k′ is vertical. Then substituting eq. (6a) for $\delta_1(k, k')$ in the lemma yields

$$p(k, k') = c_1(c_2\alpha + v\cdot\hat{w}(k, k')) + x_{k'} - x_k \ . \tag{8}$$

To eliminate α, recall that the domain of p(k, k′) is R^*:

$$p(k, k') \geq 0 \ .$$

Substituting eq. (8) and solving for α yields

$$\alpha \geq -\frac{c_1 v\cdot\hat{w}(k, k') + x_{k'} - x_k}{c_1c_2} \ .$$

The minimum integer α satisfying the inequality is

$$\alpha = \left\lceil - \frac{c_1 v \cdot \hat{w}(k, k') + x_{k'} - x_k}{c_1 c_2} \right\rceil .$$

Using the fact that $\lceil -x \rceil = - \lfloor x \rfloor$ for any x, and substituting α into eq. (8) we obtain the expression given in the theorem for vertical k'.

The remaining expression in the theorem, for horizontal k', is obtained similarly. We substitute eq. (6b) for $\delta_2(k, k')$ in the lemma when k' is horizontal and eliminating α in the same manner. □

Example. Earlier, we tabulated the values of p(k, k') for all k, k'. For example, p(1,2) = 2. This value may be derived from Theorem 3 as follows. Because k' = 2 is a horizontal line, and u = 7,

$$p(1,2) = 3\left(7\hat{w}(1,2) - 5\left\lfloor \frac{21\hat{w}(1,2) - 1}{15} \right\rfloor\right) - 1 .$$

By eq. (5), $\hat{w}(1,2) \in [1,5)$. Tabulating p for all possible values of $\hat{w}$ yields

$\hat{w}(1,2)$	p(1,2)
1	5
2	11
3	2
4	8

The minimum of p occurs at $\hat{w}(1,2) = 3$, where p(1,2) = 2 and, by eq. (5), w(1,2) = − 2.

7. Algorithm for Numerical Solution

The formulas derived above are not in closed form because we require the two minima discussed earlier:

(1) To solve for trajectory length $p(k, k')$ and function $\hat{w}(k, k')$ in the expression of Theorem 3 requires finding the integral value of $\hat{w}(k, k')$ in the interval (5) *minimizing* $p(k, k')$.

(2) To find the sequence of constraints the trajectory intersects requires taking the minimum of $p(k, k')$ for each k, over all $k' \in \Omega(k)$.

Next, we present algorithm GM to numerically evaluate the minima. We first introduce three functions to formalize the minimization problems:

$s: \{1,2,\ldots,N\} \rightarrow \{1,2,\ldots,N\}$ Function s(k) yields the index of the constraint line which a trajectory leaving constraint k next intersects. Formally, if $k' \notin \Omega(k)$ then $s(k) = \infty$; otherwise $s(k) = \hat{k} \ni \hat{k} \in [1,N] \wedge \forall k' \in [1,N]$, $p(k, \hat{k}) \leq p(k, k')$.

$p: \{1,2,\ldots,N\} \rightarrow R^{\bullet}$ Function p(k) yields how long both processes execute in parallel after they have synchronized at constraint k. Formally, if $k' \notin \Omega(k)$ then $p(k) = \infty$; otherwise $p(k) = p(k,s(k))$.

$w: \{1,2,\ldots,N\} \rightarrow R^{\bullet}$ Function w(k) yields the time one process spends blocked due to constraint s(k) after leaving constraint k. Negative (positive) w(k) indicates process 2 (1) waits. Formally, if $k' \notin \Omega(k)$ then $w(k) = 0$; otherwise $w(k) = \hat{w}(k,s(k)) + (y_{k'} - x_{k'}) - (y_k - x_k)$, where $\hat{w}(k,s(k))$ is the value in the range of (5) minimizing $p(k,s(k))$.

Algorithm GM

```
integer k, k', z, u, v, s[N], m[N];
float c1, c2, LG, Δ[N,N], p[N], w[N], x[N], y[N], l[N];
set Ω[N] of [1,N];

0. /* Initialization */
      input c1, c2, N
      LG := (l.c.d. of c1, c2)/(g.c.d. of c1, c2)
      ∀k, do
          input x[k], y[k], l[k]
          x[k] := x[k]·LG; y[k] := y[k]·LG; l[k] := l[k]·LG
          od
      c1 := c1·LG; c2 := c2·LG
      solve c2u ≡ 1 (mod c1) for u;   v = (c2u − 1)/c1
      ∀k do
          ∀k', Δ[k, k'] := (yk − xk) − (yk' − xk')
          Ω[k] := ∅; s[k] := p[k] := ∞
          od

1. /* Calculate Ω[k] */
      ∀k∀k', if ContainsInt(Δ[k, k'], Δ[k, k'] + lk') then Ω[k] := Ω[k] ∪ {k'} fi

2. /* Calculate functions s and p */
      ∀k∀k',
      if k' ∈ Ω[k] then
        for z ∈ (Δ[k, k'], Δ[k, k'] + lk'] do
          if P(c1, c2,u,v,x,y,k, k',z) < p[k] then
            s[k]: = k'; p[k]: = P(c1, c2,u,v,x,y,k, k',z); m[k]: = z fi od
        fi
      ∀k, if (p[k] ≠ ∞) then p[k] := p[k]/LG fi

3. /* Calculate function w */
      ∀k, w[k] := if Ω[k] = ∅ then 0 else (m[k] − Δ[k,s[k]])/LG fi
```

Notes:
(a) Unless otherwise specified, domain of $\forall k$ and $\forall k'$ is $[1,N]$.
(b) $ContainsInt(\omega_1, \omega_2) = (\lfloor \omega_1 \rfloor - \lfloor \omega_2 \rfloor) \neq 0$

Algorithm GM collects the previous expressions to completely specify how to calculate output functions s(k), p(k), and w(k) in the geometric model given input parameters c_1, c_2, N, x_k, y_k, and l_k.

7.1 OVERVIEW OF GM. First, we briefly describe algorithm GM. We assume that the values x_k, y_k, and l_k for each constraint line $k \in [1,N]$ are

stored in arrays x[N], y[N], and l[N], respectively. For efficiency, we introduce array $\Delta[k, k']$, which stores the difference $(y_k - x_k) - (y_{k'} - x_{k'})$.

The algorithm uses one new piece of notation: P(k, k', z). This denotes the expression for p(k, k') in Theorem 3, with the value z substituted for $\hat{w}(k, k')$ in the theorem.

Step 0 initializes several variables. Next, intersection set Ω is calculated in step 1 using Theorem 2. In step 2, function P(k, k', z) is evaluated using Theorem 3 at each integer value in its domain, $(\Delta[k, k'], \Delta[k, k'] + l_{k'}]$, to calculate functions s(k) and p(k). Because c_1 and c_2 were multiplied by LG to convert them to relatively prime integers in step 1, function p(k) must be rescaled in step 2 by 1/LG. The minimum value of $\hat{w}(k, k')$ satisfying Theorem 3 for all k is stored in array m. This allows the calculation of function w(k) in step 3, which must also be rescaled by the factor 1/LG.

7.2 COMPLEXITY OF GM. Letting N be the number of constraint lines, the storage required is $8N + \lceil \log_W N \rceil + 8$, where W is the number of bits in a memory word. This consists of:

- 3N memory cells for x[N], y[N], and l[N], storing each constraint line
- one cell each for variables k, k', z, u, v, c_1, c_2, and LG
- N^2 cells for $\Delta[k, k']$
- N^2 bits, or $\lceil \log_W N \rceil$ words, for set Ω, with one bit per pair of constraints
- 3N cells to store functions s, p, and w as arrays
- N cells to store array m.

The execution time of algorithm GM is given in Table I, in terms of the number of assignment statements executed. We make the following assumptions. The time required to calculate LG and solve the congruence in step 0 by known methods is represented by τ; this quantity depends on the magnitudes of c_1 and c_2. Also, the time required to input data equals the

number of values input: 3 + 3N. The time to execute $\forall k$, $\Omega(k) := \emptyset$ is assumed to be $\lceil \log_W N \rceil$.

TABLE I Execution time of algorithm GM

Step	Time
0	$6 + \tau + \lceil \log_W N \rceil + 8 + + N^2$
1	worst: $2N^2$ best: N^2
2	worst: $3N^2D + N$ best: 0
3	worst: N best: 0

In step 1, the *ContainsInt* expression is evaluated N^2 times. The worst case occurs when the **then** clause is executed all N^2 times. In step 2, the $z \in (\Delta[k, k'], \Delta[k, k'] + l_{k'}]$ term can be executed for 0 to D times. D is defined to be the maximum number of integers in interval $(\Delta[k, k'], \Delta[k, k'] + l_{k'}]$. In step 2 the best case occurs when $\forall k$, $\Omega(k) = \emptyset$. The scaling of p[k] in step 2 occurs at worst N times.

Summing the entries of Table I yields a worst time of $3N^2(D + 1) + 10N + \lceil \log_W N \rceil + \tau + 6$, and a best time of $2N^2 + 8N + \lceil \log_W N \rceil + \tau + 6$. Thus, the worst running time is $O(N^2D)$ and the storage space O(N).

In practice, the N^2 term is not very restrictive, because two process algorithms usually have a fairly small number of synchronization constraints. The D term, however, may force us to make approximations. In step 0, we map any rational cycle time c_1 or c_2 to relatively prime integers by

multiplying both cycle times by LG. Thus, D will grow with the product $LG \cdot c_1$ or $LG \cdot c_2$, which is the ratio of the cycle time to the resolution of the measurement clock. For example, if we measure an algorithm to microsecond resolution, LG is at most 10^6. If $c_1 = c_2 =$ 100 seconds, then $D = 300 \cdot 10^8$. However, we may be willing to trade accuracy for computation time by approximating measurements by milliseconds, so that $D = 300 \cdot 10^3$.

The dependence of complexity on D is the major disadvantage of algorithm GM. However, algorithm GM has an important advantage: It will always terminate in a solution in a bounded number of steps. This is not true for the computational geometry and integer programming techniques discussed earlier, which will not terminate in a solution if $k \notin \Omega(k)$.

It is important to realize that the dependence of complexity on D and N is only a property of algorithm GM and not of the geometric model itself. If we could find a closed form expression for p(k) and s(k) (as in [1] for N = 1), then we would not require any algorithm at all.

8. Example

In the following example, we illustrate an important property of the geometric model: We can use algorithm GM to solve the model at a small number of input parameter values, and yet interpolate the model outputs over all intermediate input values and still obtain the exact model solution.

That we can do this is not obvious, because functions p(k) and w(k) are discontinuous with respect to c_1 and c_2. Figure 11 illustrates blocking times w(1) and w(2) for the program in Figure 1. The time process 2 spends in the THINK/ACQUIRE state – denoted t_0^2 – is varied between 0 and 80. The other state occupancy times are fixed to be:

Process	*State*	*Occupancy time*
1	THINK/ACQUIRE	14
1	EAT/RELEASE	14
2	EAT/RELEASE	16

The functions plotted in Figure 11 are obtained in the following manner:

(1) We solve the geometric model for an arbitrary value of t_0^2, say 22. The result is shown in Figure 12.[13] The steady-state trajectory consists of a diagonal ray and a vertical ray of length 12. Therefore, at $t_0^2 = 22$, $w(1) = 12$ and $w(2) = 0$, as shown in the graph of Figure 11.

(2) Increasing t_0^2 corresponds to "stretching" the geometric model in Figure 12 vertically, but keeping the constraint line lengths and horizontal dimensions constant. Thus, increasing t_0^2 by 4 units causes the diagonal trajectory ray to reach a race condition, where the vertical and horizontal constraint segments intersect. Thus, we can deduce that the graphs of $w(1)$ and $w(2)$ are linear between $t_0^2 = 22$ and $t_0^2 = 26$.

(3) Now, we choose a point slightly larger than the discontinuity at $t_0^2 = 26$, say $t_0^2 = 30$. The resulting geometric model is shown in Figure 13. Here, the steady state consists of a diagonal ray of projected length 32, then a horizontal ray of length 6, then a diagonal ray of project length 14, then a vertical ray of length 2. Thus, in Figure 11 we plot $w(1) = 2$ and $w(2) = 6$ at $t_0^2 = 30$.

---

[13] The dashed lines in Figures 12 and 13 are drawn every 4 time units in the x and y directions to assist the reader in identifying the coordinates of various points in the diagram.

(4) From the geometric model in Figure 13, we can deduce, by again considering the effect of stretching the diagram vertically, that $w(1)$ and $w(2)$ are linear and continuous between $t_0^2 = 26$ and $t_0^2 = 40$. At $t_0^2 = 40$, $w(2)$ becomes 0 because the steady state changes to omit the horizontal ray.

(5) This process is continued to build the graph of Figure 11.

This example illustrates two things. First, we only need to solve the geometric model for two sets of input parameters to deduce the behavior for $12 < t_0^2 < 40$. Second, graphs of w and p are discontinuous at points corresponding to a change in the number or orientation of rays of the steady-state trajectory.

9. Conclusions

We have presented an exact analysis of two process, deterministic distributed programs whose synchronization code is not embedded in conditionally executed code. Using the geometric model, we can obtain the sequence of synchronization points where the program blocks during steady state [function $s(k)$], the blocking durations [$w(k)$], and the duration of concurrent execution between synchronization points [$p(k)$].

The most pragmatic alternative to using the geometric model today is simulation. (Given an initial program state, a simulator will generate successive global states, thus enumerating the initial part of a trajectory. A simulator could be programmed to stop when it detects the first cycle.) The geometric model offers several advantages over simulation:

(1) A single solution of the geometric model yields performance at *all* initial conditions, whereas simulation only yields the behavior at one initial condition.

(2) The geometric-model solution can be used to calculate the boundaries within which the solution is linear. We then know at what interesting parameter values to resolve the model, and where the discontinuities lie in wide parameter ranges.

(3) The simulation duration is proportional to the sum of the duration of the transient and the steady-state period. However, algorithm GM is independent of these two quantities.

The chief qualitative result of the geometric model is that in the absence of race conditions and deadlocks, a program fitting our model always settles into a steady-state behavior consisting of a repetition of a finite sequence and timing of global-state transitions. There may be several steady-state cycles, depending on the initial condition. This is illustrated in Figure 14, where one steady state involves no blocking, while the second requires blocking at both constraints.

There is some similarity between distributed programs and classical dynamic systems, such as electrical circuits with feedback. Dynamic systems may reach a limit cycle behavior, analogous to the cyclic state transitions studied here.

Furthermore, we have some experimental evidence of the similarity. In [1], we measured a Dining Philosophers algorithm with between four and 64 processors. For small numbers of processes, the global-state transition sequence is deterministic. Starting at about 22 processors, small perturbations occur in the steady-state cycle for short instances of time, after which the program returns to steady state.

However, distributed programs are dissimilar from dynamic systems because of discontinuities. If we vary a constraint length or cycle time in a program, the blocking time will change linearly within a certain interval. However, if the variation is large enough, the algorithm will change

steady-state cycles, so that the blocking times and sequence of constraints intersected change discontinuously.

Discontinuities complicate design and tuning of a distributed program, because a programmer is unaware of the discontinuity locations. This causes counterintuitive behavior, such as when one process is speeded up, the overall program performance is degraded.

ACKNOWLEDGEMENTS. The authors wish to thank Satish Tripathi for several discussions during the course of this work, and Liba Svobodova for suggestions on the manuscript.

REFERENCES

1. ABRAMS, M. Performance analysis of unconditionally synchronizing distributed computer programs using the geometric concurrency model. Ph.D. Dissertation, Dept. of Computer Science, University of Maryland, 1986.
2. CARSON, S. D. Geometric models of concurrent programs. Ph.D. Dissertation, Technical Report 84-05, Dept. of Applied Math and Computer Science, University of Virginia, Charlottesville, VA, 1984.
3. CARSON, S. D., and REYNOLDS, Jr., P. F. The geometry of semaphore programs. Computer Science Technical Report 85-03, University of Virginia, Charlottesville, VA, April 1985, to appear in *ACM Trans. on Programming Languages and Systems*.
4. COFFMAN, Jr., E. G., ELPHICK, M. J., and SHOSHANI, A. System deadlocks. *ACM Comp. Surv. 3*, (June 1971), 70-71.
5. DIJKSTRA, E. W. Cooperating sequential processes. Tech. Rep. EWD-123, Technological University, Eindhoven, The Netherlands, 1965.
6. GELENBE, E., LICHNEWSKY, A., and STAPHYLOPATIS, A. Experience with the parallel solution of partial differential equations on a distributed computing system. *IEEE Trans. Comput. C-31*, 12, (Dec. 1982), 1157-1164.
7. HEIDELBERGER, P., and TRIVEDI, K. S. Queueing network models for parallel processing with asynchronous tasks. *IEEE Trans. Comp. C-31*, 11, (Nov. 1982), 1099-1109.
8. HEIDELBERGER, P., and TRIVEDI, K. S. Analytic queueing models for programs with internal concurrency. *IEEE Trans. Comp. C-32*, 1, (Jan. 83), 73-82.
9. JONES, B. W. The theory of numbers. Rinehart New York, 1955.
10. LIPSKI, W., and PAPADIMITRIOU, C. H. A fast algorithm for testing for safety and detecting deadlocks in locked transaction systems. *J. Alg. 2*, 3, (Sept. 1981), 211-226.

11. PAPADIMITRIOU, C.H. Concurrency Control by Locking. *SIAM J. Comput. 12*, 2, (May 1983), 215-226.

12. PLATEAU, B., and STAPHYLOPATIS, A. Modeling of the parallel resolution of a numerical problem on a locally distributed computing system. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Performance Evaluation Review 11*, 4, (Winter 1982), 108-117.

13. PLATEAU, B. De l'évaluation du parallélisme and de la synchronisation. Thése d'état, Université de Paris-Sud, 91405 Orsay, France, Nov. 1985.

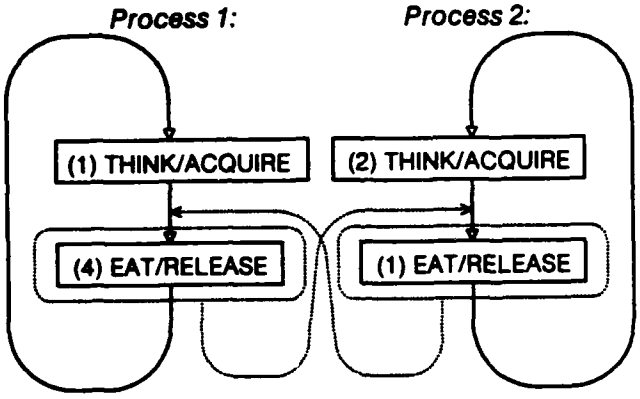
14. PLATEAU, B. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *Proceedings of the SIGMETRICS 1985*, Austin, Aug. 1985, 147-154.

15. PREPARATA, F. P. and SHAMOS, M. I. Computational geometry. Springer-Verlag, New York, 1985, Ch. 8.

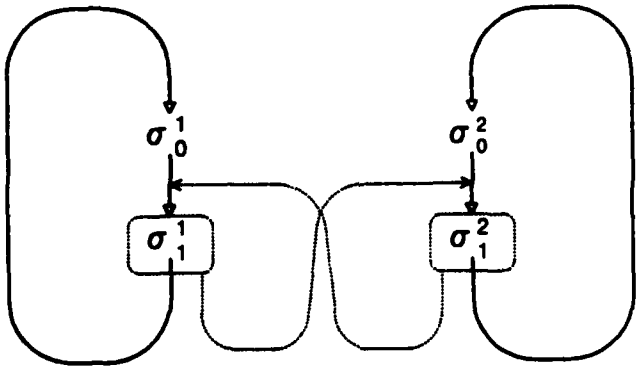
16. VERNON, M., ZAHORJAN, J., and LAZOWSKA, E. D. A comparison of performance petri nets and queueing network models. TR-669, University of Wisconsin, Sept. 1986.

17. YANNAKAKIS, M., PAPADIMITRIOU, C. H., and Kung, H. T. Locking policies: safety and freedom from deadlock. In *Proceedings of the 20th ACM Symposium on the Foundations of Computer Science*, 1979, pp. 283-287.

(a)



(b)



$t^1_0 = 1$ $\quad t^2_0 = 2$

$t^1_1 = 4$ $\quad t^2_1 = 1$

$M_1 = 2$ $\quad M_2 = 2$

$c_1 = 5$ $\quad c_2 = 3$

FIG. 1. (a) Flow charts for dining philosophers program. State occupancy times are shown in parenthesis. (b) Equivalent representation of (a).

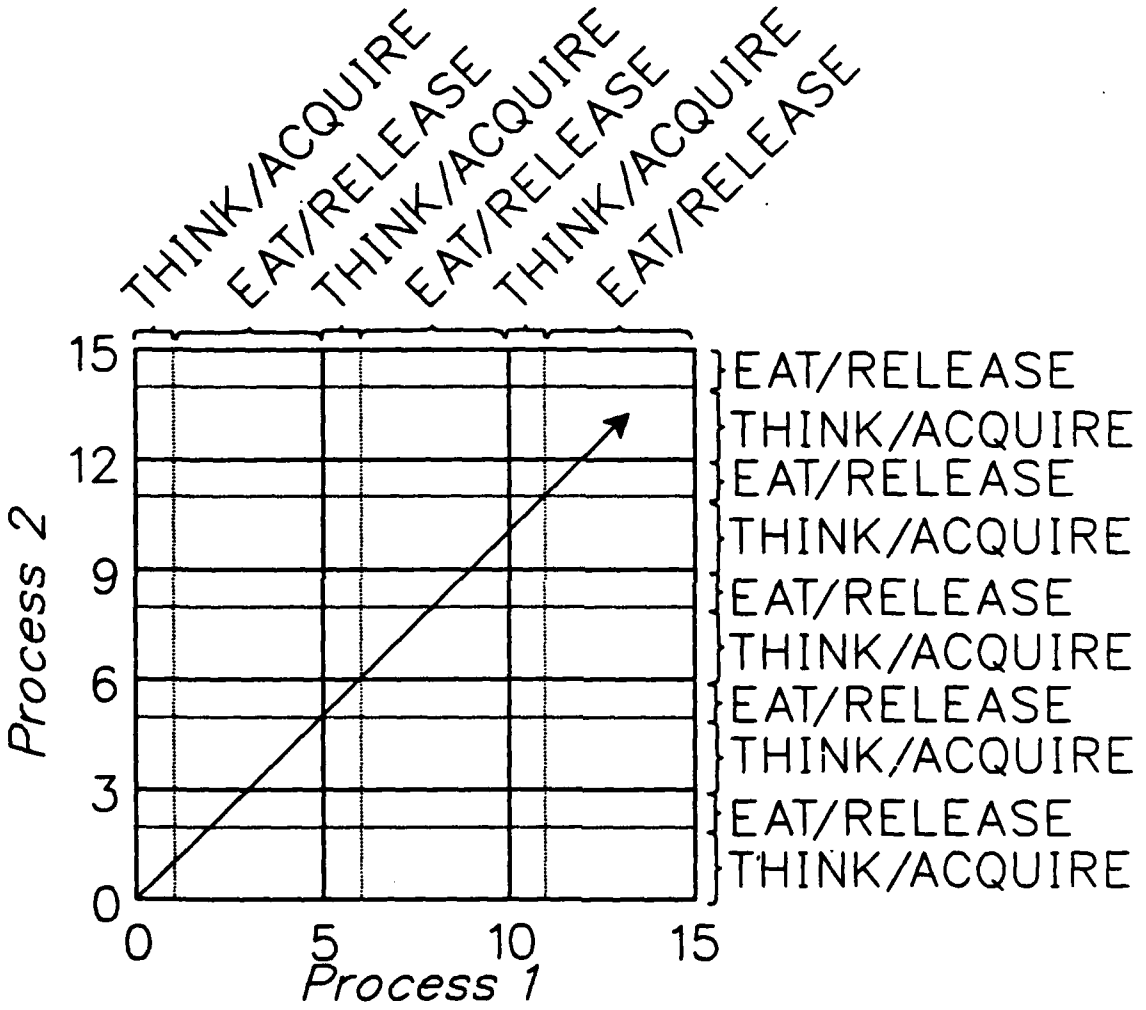


FIG. 2. Mapping states and trajectory to Geometric Model.

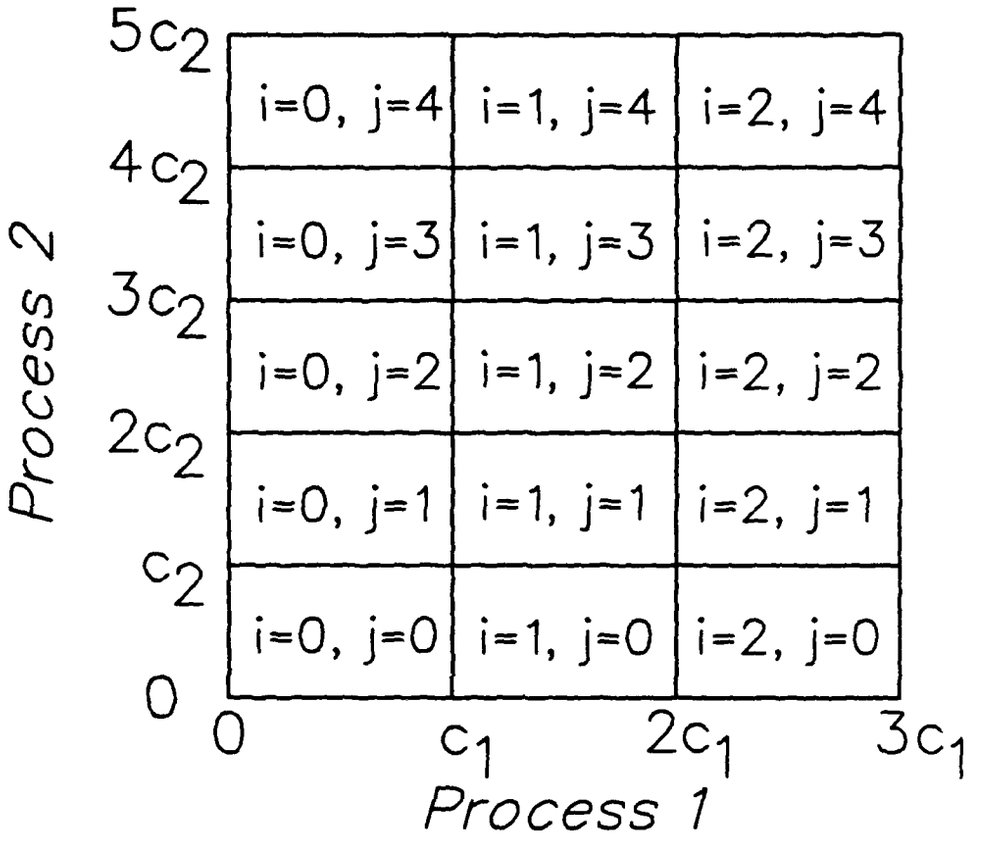


FIG. 3. Numbering each quadrant by a pair of integers. The quadrant number indicates how many times each process has looped.

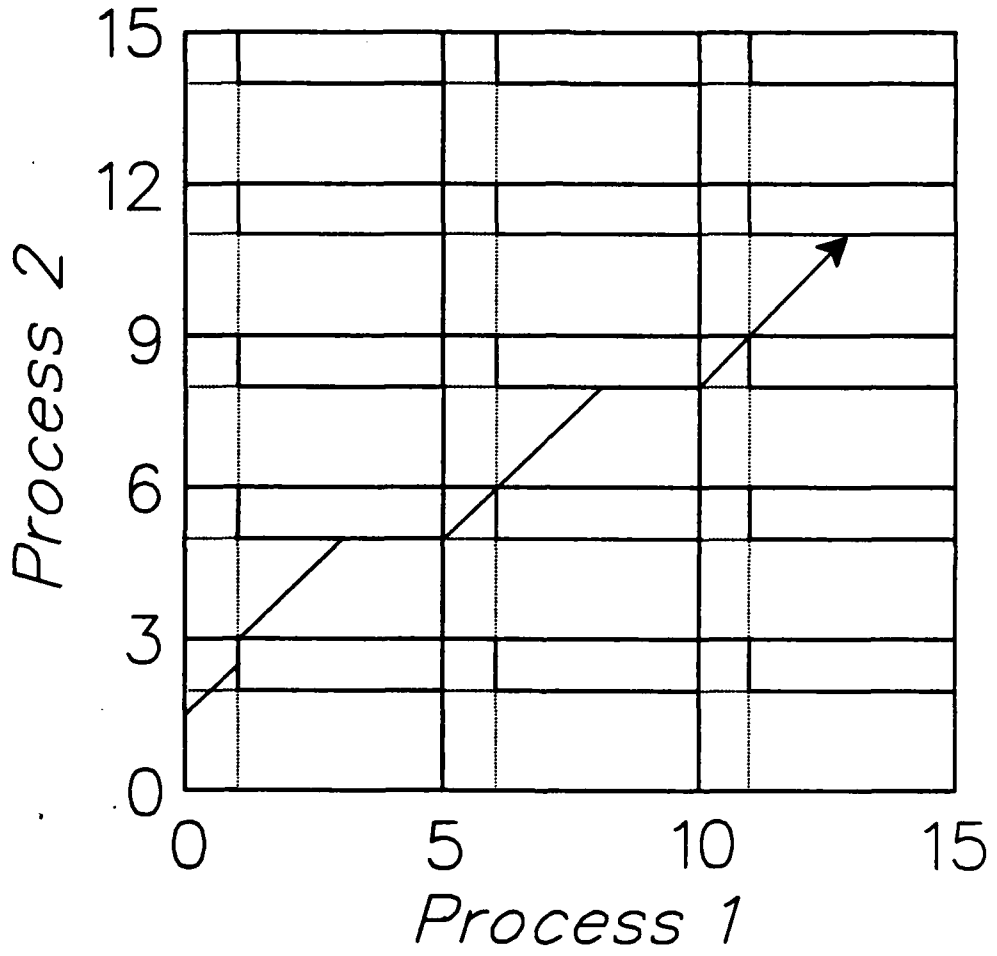


FIG. 4. Mapping synchronization constraints to Geometric Model.

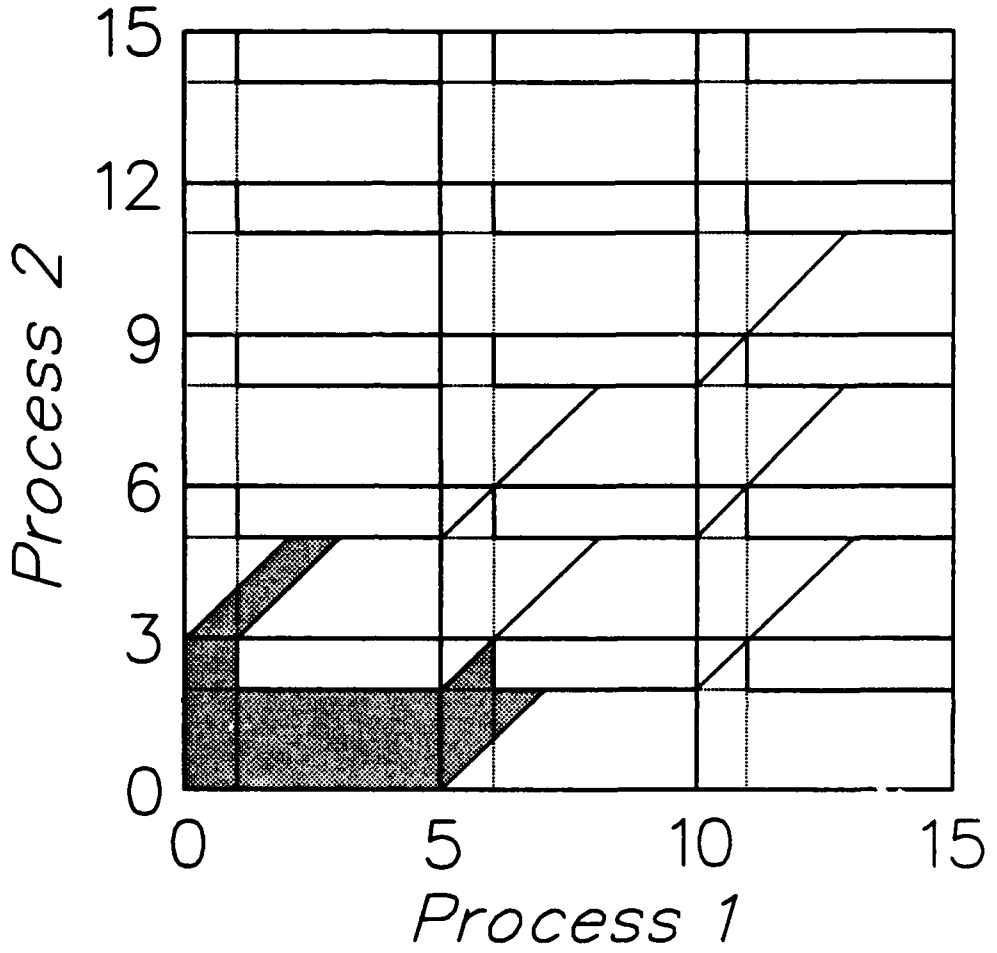


FIG. 5. Mapping initial conditions to Geometric Model.

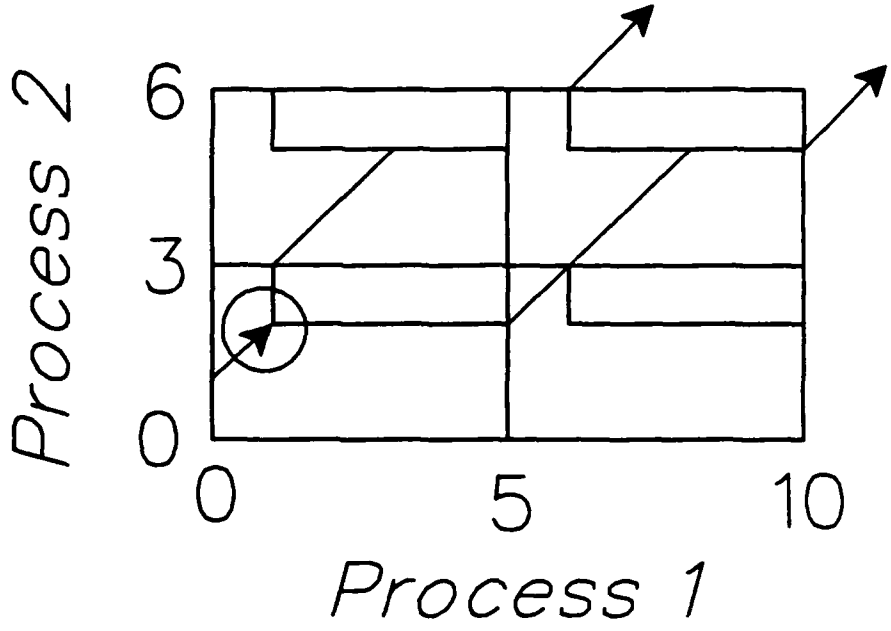


FIG. 6. Manifestation of deadlock in the Geometric Model.

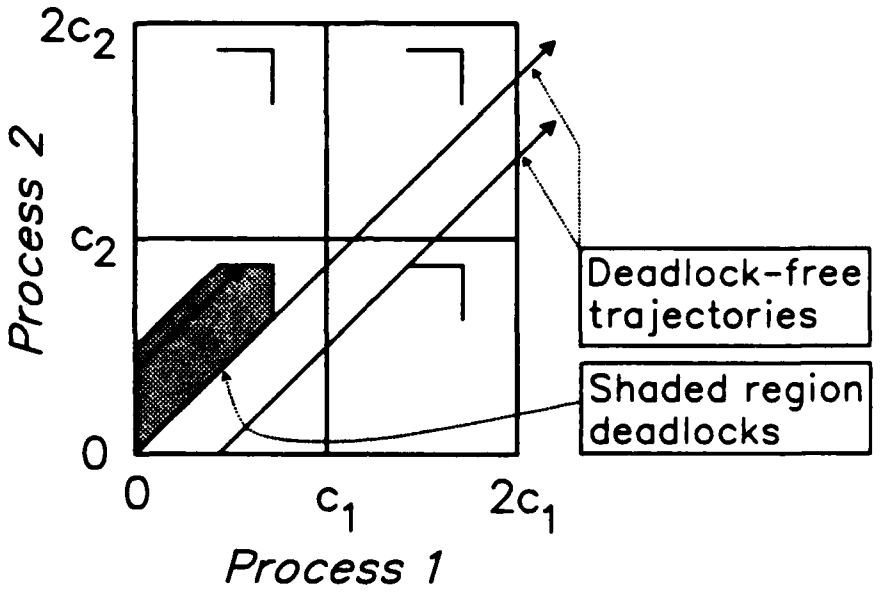
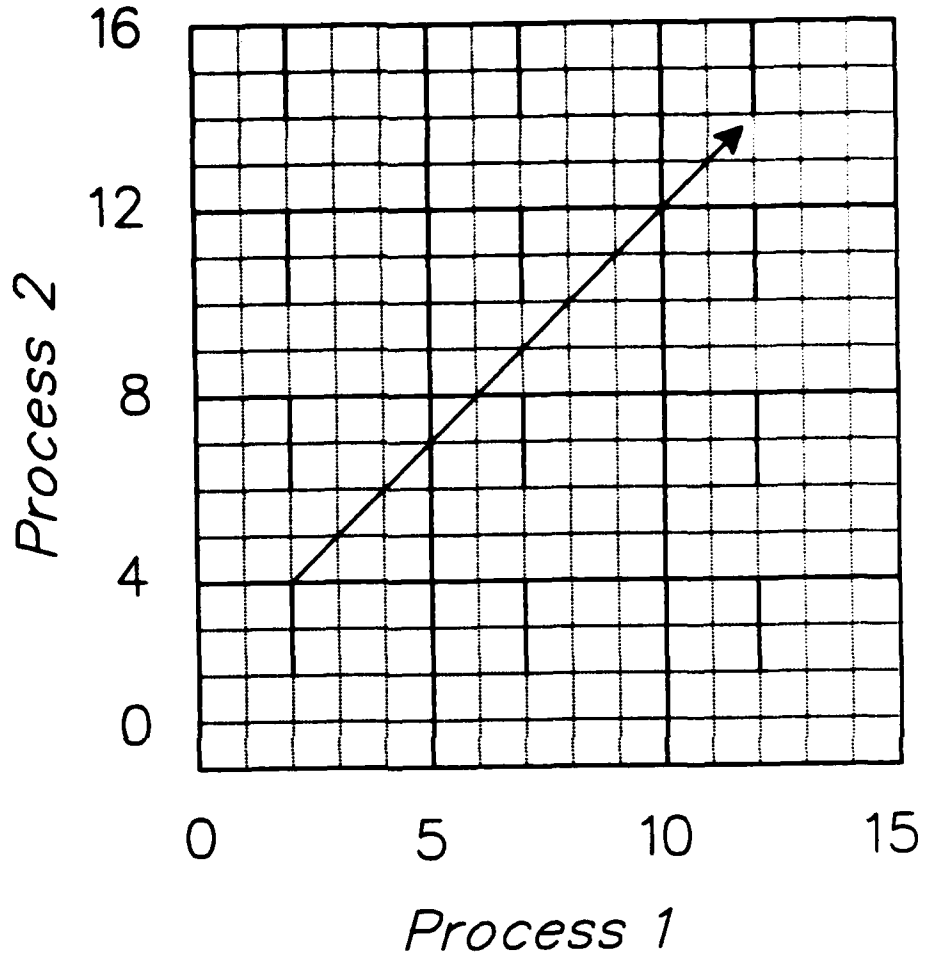
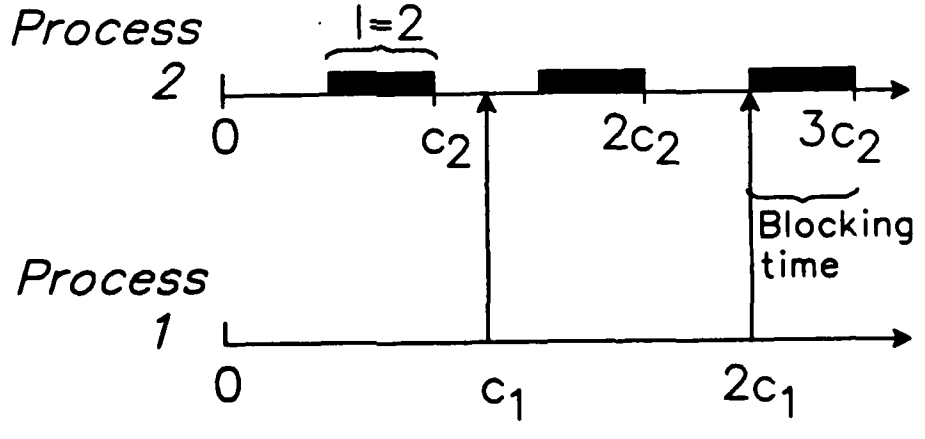


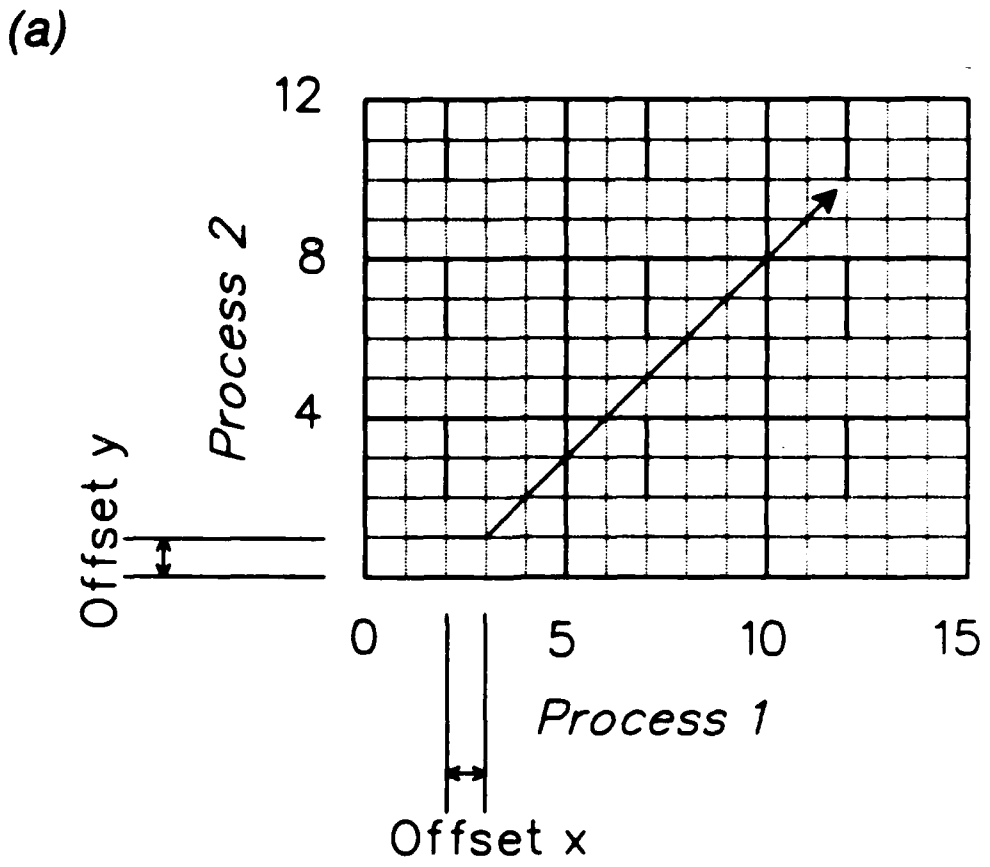
FIG. 7. Manifestation of race conditions in the Geometric Model.

(a)


16
12
8
4
0
Process 2
0
5
10
15
Process 1

(b)


Process 2
I=2
0
c2
2c2
3c2
Blocking time
Process 1
0
c1
2c1



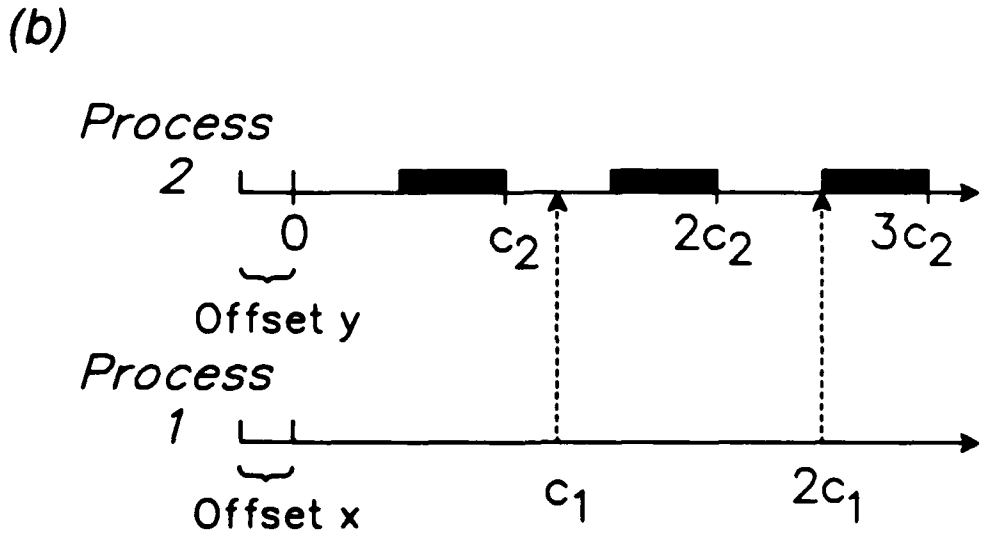


FIG. 9. (a) Steady state equation when $k \leq k'$. (b) Time lines corresponding to (a).

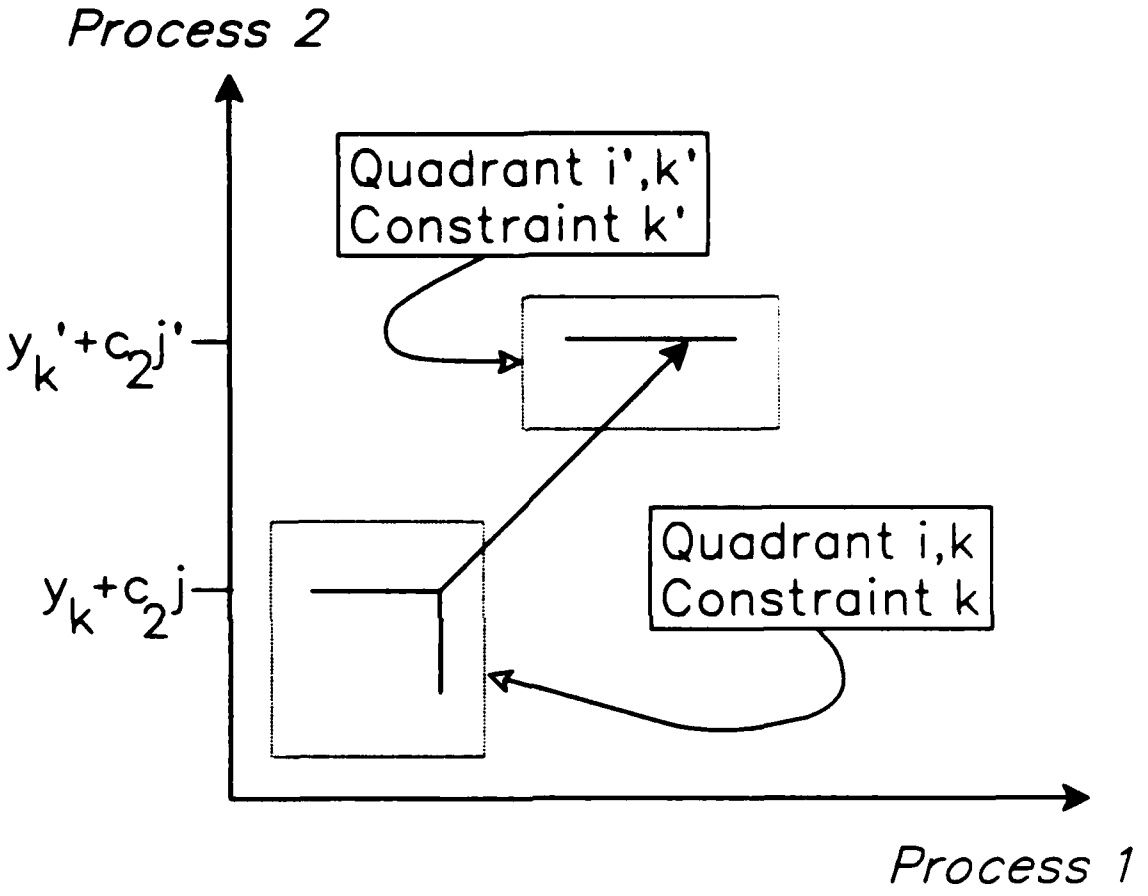


FIG. 10. Proof of the Lemma, case 2.

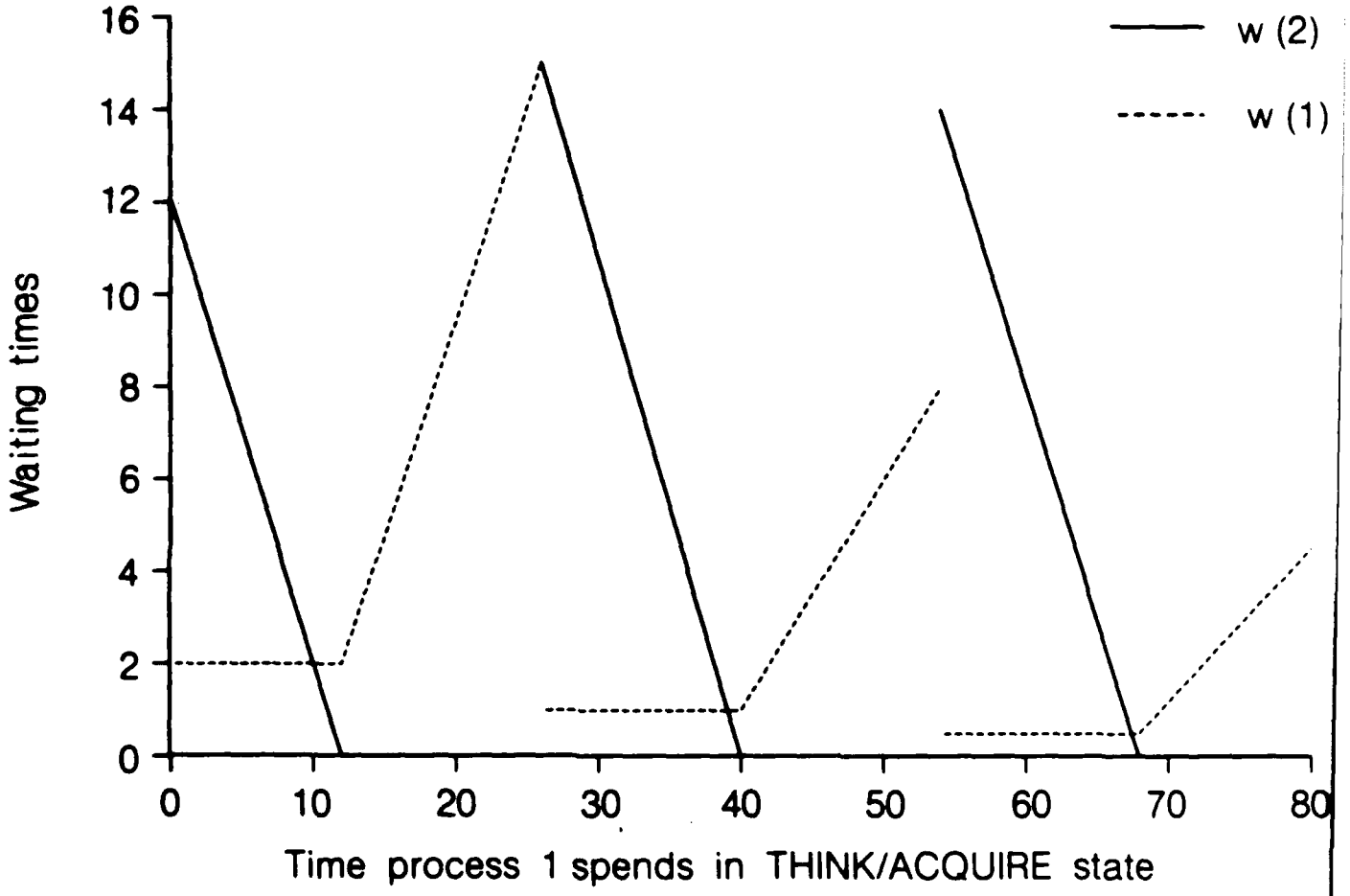


FIG. 11. Steady state waiting times (w(1) and w(2)) versus duration of time process 2 spends in THINK/ACQUIRE state (t_2) for program of Fig. 1.

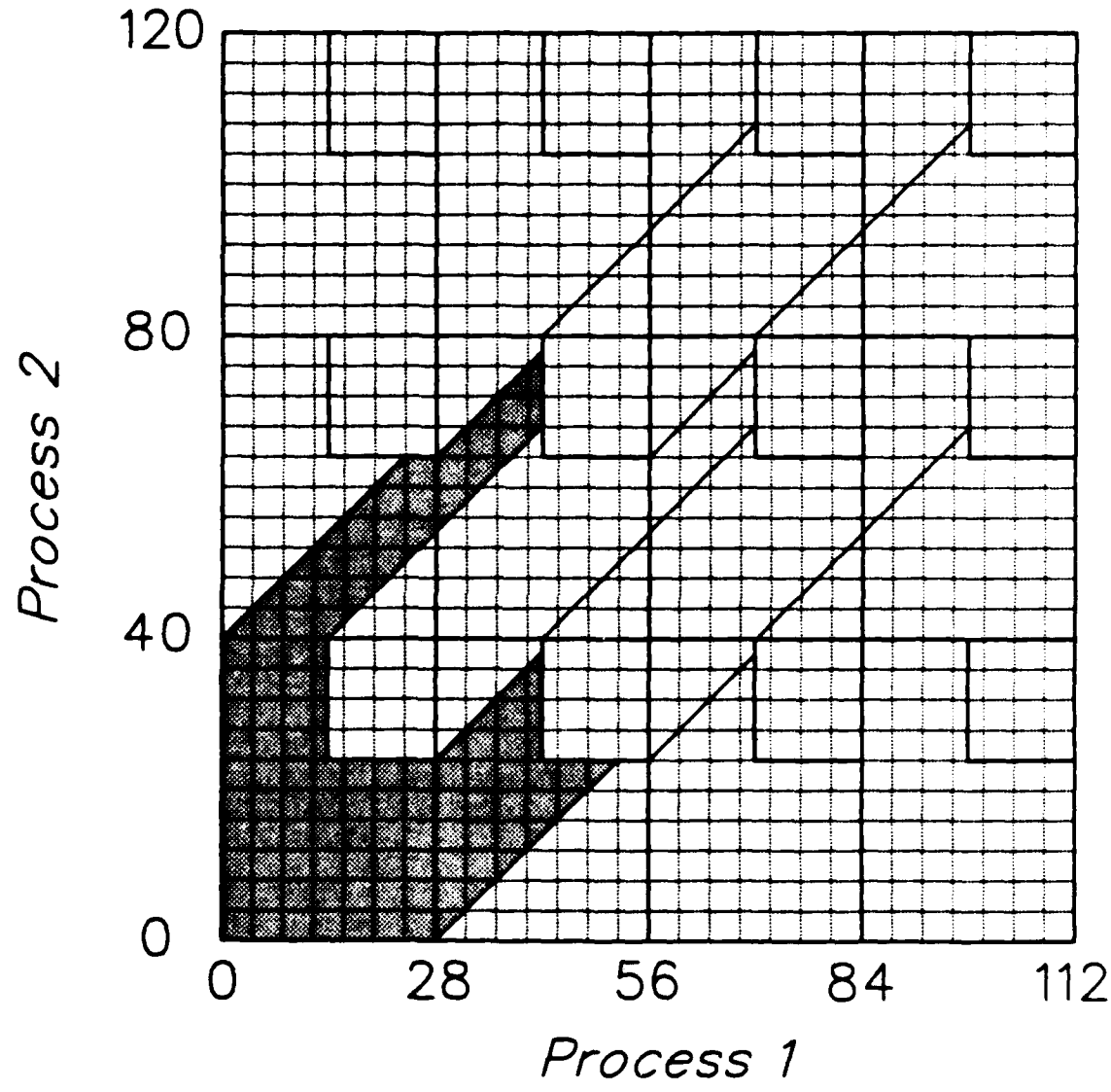


FIG. 12. Geometric Model at the point $t_\delta^2 = 22$ for Fig. 11.

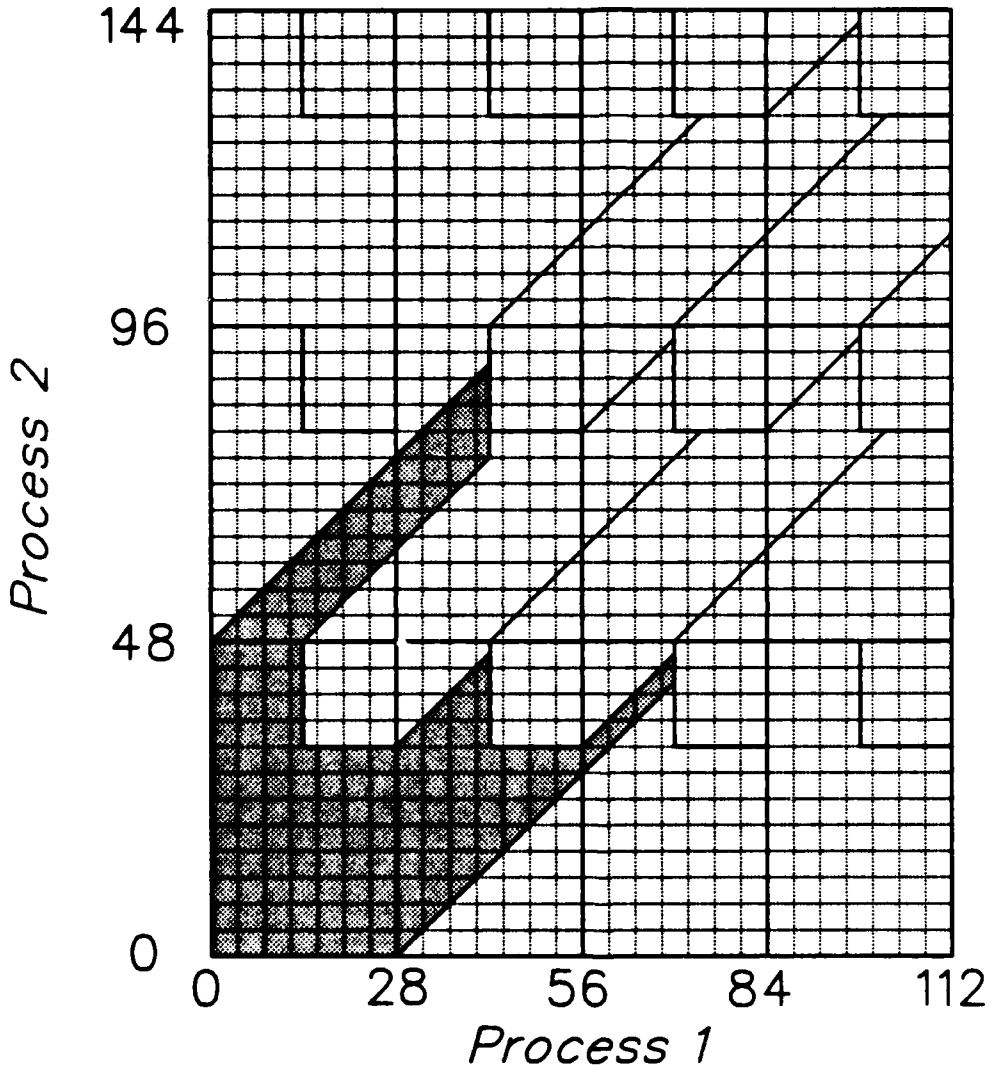


FIG. 13. Geometric Model at the point $t_0^2 = 30$ for Fig. 11.

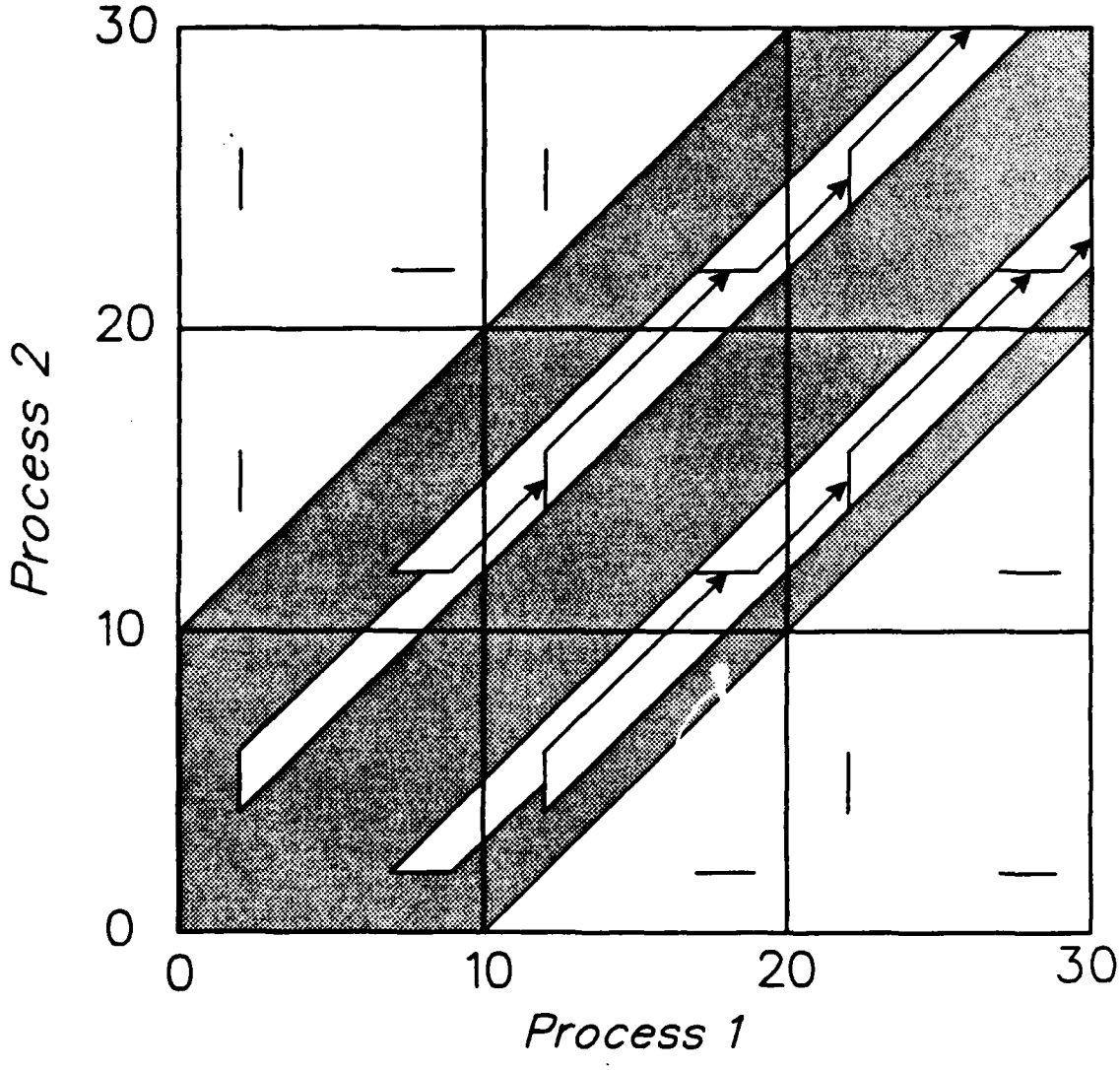


FIG. 14. Program with two different steady states.

END

7-87

DTIC