Programming Techniques and Data Structures

Ellis Horowitz Editor

Performance Analysis of On-the-Fly Garbage Collection

TIM HICKEY and JACQUES COHEN

ABSTRACT: Parallel garbage collection systems consist of two processors, the mutator and the garbage collector, which operate on a common memory of list nodes. The garbage collector repeatedly executes a two-stage cycle that maintains a list of nodes (the free list) for use by the mutator, which is dedicated to executing a user's list processing program. All list nodes contain two extra bits for marking purposes and are either white, black, or gray. During the mark stage, the garbage collector blackens all nodes that are accessible to the mutator. During the scan stage, white nodes are placed on the free list and black nodes are whitened. The color gray is used during the mark stage to guarantee that all accessible nodes are marked. It is shown that parallel garbage collection systems exhibit three fundamental types of performance (stable, alternating, and critical), and it is determined when each of the three will occur. Moreover,it can be determined when this parallel scheme is more efficient than sequential garbage collection by obtaining analytic formulas for a variety of performance measures. These results are obtained by expressing the performance of parallel garbage collection systems in terms of conditional difference equations (CDEs) in which the state of the system after a garbage collection cycle is described by a pair of difference equations and a Boolean function. The value of the Boolean function at the beginning of a cycle selects which of the two difference equations should be applied next.

1. INTRODUCTION

As programs that involve extensive list processing become more common in industrial applications, the need for efficient, real-time garbage collection increases. A five-minute delay for garbage collection is a nuisance for researchers but could be of crucial importance in other applications. For example, a natural language interface to an emergency medicine database might be written in a list processing language, but a potential for lengthy delays due to garbage collection would render such a system untrustworthy. The efficiency of parallel garbage collectors will play a major role in the speed with which list processing becomes widespread.

The purpose of this paper is to estimate the efficiency of on-the-fly, parallel garbage collection systems in a way that can be used by an operating system to efficiently allocate memory and processor resources. The analyses of these collectors presuppose the use of *only* fast memory. When very large virtual memories are used, the appropriate measures of efficiency and the parameters effecting the performance of the systems differ from those considered here.

An abstract model of parallel garbage collection, which encompasses Dijkstra's three-color garbage collection algorithm [3] and the four-color algorithm of Kung and Song [5], is described in Section 2. Conditional difference equations (CDEs) are developed in Section 3, which predict the state of an on-the-fly garbage collection system after one garbage collection cycle based on its state before the cycle. The CDEs consist of a Boolean function and two difference equations. The Boolean function determines whether the free list will be depleted during the coming cycle and is used to select which of the two difference equations should be applied to determine the state of the memory after the cycle.

In Section 4, we show that parallel garbage collection systems exhibit three fundamental types of performance: (1) The mutator never needs to wait, (2) the mutator must wait every other garbage collection cycle, and (3) the mutator is forced to wait during every cycle.

This work was supported by the National Science Foundation under Grants MCS 79-05522 and DCR 83-17892.

These performance types are called stable, alternating, and critical performance, respectively. We had observed one and two cycle periodicity in our simulations of parallel garbage collection systems, which were made in the early stages of this research. By analyzing the CDEs, we derive formulas that determine when the three types occur. We also develop formulas in each of the three cases, for the lengths of the garbage collection cycles, the amount of wait time per cycle, the size of the free list, the amounts of (marked and unmarked) garbage, and other related parameters.

Sections 5 and 6 provide applications of the analysis in Section 4 to various design and performance problems. After defining several quantitative performance measures, we derive closed form formulas for the measures and provide examples. In particular, the productivity of on-the-fly garbage collection is shown to be at most 150 percent of the productivity of a serial garbage collection system, where the productivity is the proportion of time during which the mutator is not waiting for garbage collection.

This work is related to the results of Wadler [9] and Kung and Song [5], which are described in Section 6. Wadler developed a sufficient (but not necessary) condition for a parallel garbage collection system to be stable (in our terminology). Kung and Song described a new parallel garbage collection algorithm using output restricted deques and presented results of an (unpublished) analysis of their algorithm in the stable case. This paper is concerned with general parallel garbage collection and is aimed at obtaining results in the spirit of microanalysis described in [2], for use by operating systems and for those designing list processing systems. Directions for future research are discussed in Section 8.

2. ON-THE-FLY GARBAGE COLLECTION

2.1 The Algorithm

The garbage collection systems that are analyzed here consist of two processors, the mutator and the garbage collector, that share a common memory of N consecutive list nodes. Each node in the shared memory contains a data field, two fields containing pointers to other nodes, and a (2-bit) color field used by the garbage collector to mark nodes either black, white, or gray. There are two distinguished nodes in the list memory. The first node points to the head and tail of a linked list of nodes called the free list. The second node points to the mutator's graph, which is also called the working list. A node is *accessible* if it is in the mutator's graph and is *in-use* if it is in the free list or is accessible.

The mutator operates on the graph using primitive operations [4], which allow it to change a pointer of any accessible node to point to any other accessible node (including nil) or to point to a new node taken from the head of the free list. The action of the mutator tends to produce nodes that are not on the free list or the working list; these are the garbage nodes.

The purpose of the garbage collector is to identify garbage nodes and place them at the tail of the free list. The garbage collector accomplishes this task by repeatedly executing a two-stage cycle in which it first marks (blackens) all of the nodes that are in-use, and then, during a linear scan of memory, appends all of the unmarked nodes to the free list and unmarks (whitens) the marked nodes. The fundamental obstacle to extending serial garbage collection algorithms [1] to the two processor systems considered here is the problem of marking all of the in-use nodes in a graph *while* the graph is being modified.

A particularly elegant solution to this problem was given by Dijkstra [3]. His algorithm uses a third color, gray, during the mark stage to guarantee that all in-use nodes will be marked. No use of semaphores or other interprocessor communication is needed (see [8]), and the only restriction placed on the mutator is that when it changes a pointer field the node pointed at must be shaded. Shading is an operation that turns white nodes gray and has no effect on gray or black nodes. It must be a primitive operation.

The mark stage of Dijkstra's algorithm is begun by graying the roots of both the free list and the mutator's graph; all other nodes are either white or gray at this time. The garbage collector then chooses a gray node, shades its sons, and blackens the gray node. Figure 1a shows an initial configuration of the list memory just before the beginning of the mark stage. The reason node 1 is gray will soon be apparent. Figure 1b shows the memory midway through the mark stage. The links are the same in both figures, but the colors differ. The operations of the garbage collector and the mutator during this stage preserve the following invariance property proposed by Dijkstra: "*Every white in-use node can be reached from a gray node along a path passing through white nodes exclusively.*" It follows that an absence of gray nodes implies that all in-use nodes are marked black, and the mark stage can be ended. Note that some marked nodes may become garbage during the mark stage.

After the marking stage (see Figure 1c) there are two types of garbage nodes, (1) the unmarked garbage nodes, which will be collected during the scan stage of the current garbage collection cycle (e.g., nodes 10 and 11); and (2) the marked garbage nodes, which will not be collected until the scan stage of the next collection cycle (e.g., nodes 1 and 2). The marked garbage nodes are called slow garbage or floating garbage [9], and the unmarked garbage nodes are called quick garbage. In no case will garbage remain uncollected for more than two garbage collection cycles.

During the scan stage the memory is examined sequentially; white nodes are appended to the free list, and black nodes are whitened in preparation for the next mark stage. A snapshot of the memory configuration, midway through the scanning stage, is depicted in Figure 1d. The right pointer from node 3 to node 4 was dropped and the left pointer adjoined, while node 4 was still black, and hence the mandatory shading operation had no effect on node 4. Had this pointer manipulation occurred after node 4 was scanned and whitened, the shading operation would have colored node

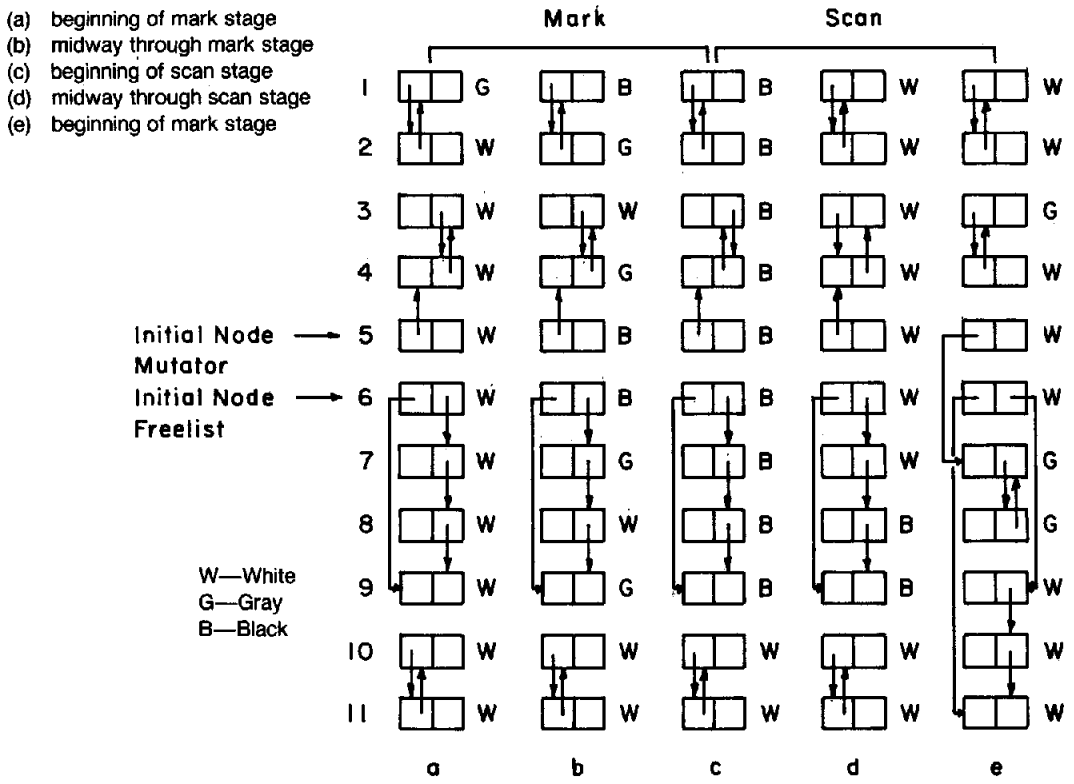

FIGURE 1. Snapshots of Memory Configurations during Parallel Garbage Collection

4 gray. Figure 1e shows the memory immediately after the scan stage. The right pointer from node 4 to 3 has been dropped and the left pointer adjoined. This explains why node 3 is gray in Figure 1e (and why node 1 in Figure 1a is gray). Moreover, two new nodes (7 and 8) have been added to the mutator's list and shaded.

Gries [4] presents a detailed program that implements this algorithm and provides a correctness proof. There are many ways to implement Dijkstra's algorithm. The operation of finding a gray node in the mark stage can be implemented (1) using a linear scan of memory, (2) using a stack in which the address of the shaded nodes are pushed, or (3) using hardware in which this is a primitive operation. Such an implementation will be called a three-color collector. Gries' implementation [4] uses the linear scan of memory. (Our analysis suggests that Gries' implementation would only be viable when the memory is small and the mutator is slow.)

A variant of the three-color collector involves using a fourth color, off-white, and requires that nodes on the free list always be off-white. This may shorten the mark stage since the nodes in the free list will not have to be marked. Such an implementation will be called a four-color collector. Kung and Song [5] present and prove correct a particular implementation of a four-color collector. In an appendix they present the results of an unpublished performance analysis of their particular collector in the stable case. In Section 4, we present a performance analysis of four-color collectors. The performance analysis of three-color collectors is similar and shows that four-color collectors react more robustly to changes in the fundamental parameters and settle into equilibrium more quickly than three-color collectors.

2.2 The Model

The model we use to investigate parallel garbage collection systems isolates the effect of five fundamental parameters N, L, r, s, and m (described in Table I and in the paragraphs below) on the efficiency of the system.

We assume that while the free list is nonempty the mutator adjoins new nodes at a constant rate of one node every r time units and that nodes are released from the mutator's graph (thereby becoming garbage) at the same constant rate. This assumption implies that the size of the mutator's graph will be constant throughout the life of the system. Let L be the number of nodes in the mutator's graph and N be the total number of nodes in the shared memory.

The average rate at which new nodes are added to the mutator's graph and the average rate at which nodes are released will be equal in any system that runs for an indefinite period of time. Although these rates need not be constant, only the limiting case where the rates are constant will be treated here. If the value of r changes relatively slowly with respect to the length of a garbage collection cycle, then our analysis can be used to describe the system's performance as a function of the garbage production rate. Nevertheless, we will

TABLE I. The Time and Space Variables for the Garbage Collection Model (upon completion of the *i*th cycle)

	Time Variables
m	time to MARK a gray node
s	time to SCAN the next list node
r	time to RELEASE a garbage node or RETRIEVE a new node
$tmark_i$	length of the ith marking stage
$tscan_i$	length of the ith scanning stage
$tcycle_i$	length of the ith cycle
$tprod_i$	amount of productive (nonwait) time in the ith cycle
$twait_i$	amount of wait time in the ith cycle

	Space Variables
N	number of list nodes in the memory
L	number of accessible nodes
π	proportion of list nodes that are accessible (L/N)
F_i	number of nodes in the free list
W_i	number of unmarked (quick) garbage nodes
B_i	number of marked (floating) garbage nodes
B^{crit}	largest possible value of B before a noncritical cycle
G^{crit}	amount of marked garbage produced during a four-color stable cycle

assume that r is constant in our analysis. The garbage collector can be described by two parameters: the rate at which nodes are scanned (1 node/s time units) and the rate at which they are marked (1 node/m time units). Again we assume that these rates are constant. These five parameters (N, L, r, s, m) will be called the static variables; they are microanalysis variables in sense of [2] and do not change from cycle to cycle.

A garbage collection system that satisfies the assumptions in the previous paragraphs is said to be in equilibrium; it is completely determined by the five parameters just described. To analyze such a system, we introduce variables that describe the state of the memory at the end of each mark stage and analyze the effect of a single garbage collection cycle on these variables. Since the state of the memory changes from cycle to cycle, these new variables will be called the dynamic variables of our model. When the mark stage ends, all gray nodes have been blackened, and so all nodes are either white or black. We therefore define a garbage collection cycle to begin with the scan stage and end with the mark stage rather than the reverse (which is the more common convention).

Three dynamic variables describe the state of the memory after the ith garbage collection cycle: (1) the size of the free list F_i, (2) the amount of quick (white) garbage W_i, and (3) the amount of slow (black) garbage B_i. The slow garbage will be whitened during the scan stage of the next cycle and will not be collected until the cycle after next. On the other hand, the quick garbage will be collected and made available to the mutator immediately. The total number of nodes that will become available for use by the mutator during the scan stage of the $(i + 1)$th cycle is $A_{i+1} = F_i + W_i$.

We make the worst-case assumption that all nodes released during a given cycle are marked and will not be collected until the cycle after next. This assumption will allow us to provide upper bounds for productivity and other performance measures for any three- or four-color collector. These upper bounds are actually realized by mutators that produce only marked garbage. For example, if a mutator only uses new cells to store temporary values, all new cells will be short-lived and hence will become garbage immediately after being shaded by the mutator. Thus all garbage will be marked. According to Lieberman and Hewitt [7], most garbage produced by actual LISP programs is short-lived.

Finally, we will let $tmark_i$ and $tscan_i$ represent the durations of the mark stage and scan stage (respectively) of the ith garbage collection cycle, and will let $twait_i$ denote the length of the time interval during which the free list is empty in the ith cycle. The length of the entire cycle is then $tcycle_i = tmark_i + tscan_i$, and the amount of productive time enjoyed by the mutator during the ith cycle is $tprod_i = tcycle_i - twait_i$.

3. CONDITIONAL DIFFERENCE EQUATIONS

In this section, the conditional difference equations (CDEs) governing the performance of the three- and four-color parallel garbage collection systems will be determined. The CDEs will allow the values of the dynamic variables after a cycle to be calculated efficiently from their values before the cycle. Hence, the system can be simulated given the initial values of the dynamic variables. Such a simulation is presented at the end of this section. The values of the dynamic variables for a given cycle will be calculated from the values of the dynamic variables in the previous cycle and the values of the static variables. The model we are analyzing consists of the five static variables (N, L, r, s, m) and six dynamic variables (F_i, W_i, B_i, $tmark_i$, $tscan_i$, $tprod_i$) described in the previous section. The dynamic variables vary from cycle to cycle, whereas the static variables do not.

A graphic representation of the effect of four garbage collection cycles on the dynamic variables for a typical four-color collector is given in Figure 2. In this example, the scan, mark, and release times are 1, 5, and 20 microseconds, respectively. The memory consists of 1000K ($= N$) list nodes, and 600K list nodes are accessible to the mutator. Initially, all nodes not accessible to the mutator are on the free list, so there are no garbage nodes. The region B in Figure 2 represents the proportion of list nodes that are marked black or gray. The other regions represent the free list (F), the mutator's graph (L), and the unmarked garbage nodes (W). For example, 8 seconds after the system begins execution, the free list is empty, 133K of the list nodes are marked garbage nodes, and 267K are unmarked garbage nodes; the remaining 600K nodes are accessible to the mutator.

During the mark stage the free list in Figure 2 is diminished at a constant rate (1 node/20 microseconds = 50K nodes/sec), which is the same rate at which the number of marked garbage nodes is growing. This is a

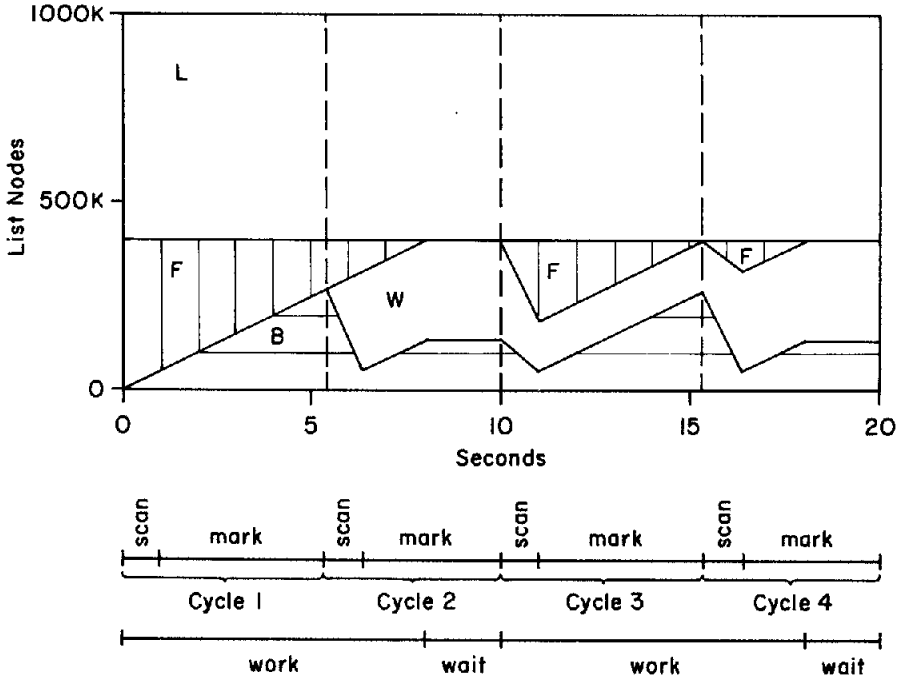

FIGURE 2. Four-Color Collector with $s = 1$, $m = 5$, $r = 20$ (microseconds), $L = 600K$, $N = 1000K$ (list nodes)

consequence of our worst-case assumption that all newly created garbage is marked. This assumption also explains why the amount of unmarked garbage remains constant during the scan stage in Figure 2. During the scan stage all of the unmarked garbage is appended onto the free list, and the marked garbage nodes are whitened. Thus, the unmarked garbage nodes after the scan stage are the nodes that were marked at the beginning of the cycle. When the mark stage ends before the free list is exhausted, the cycle is called a stable cycle. A cycle in which the free list does become exhausted is a critical cycle. In Figure 2 the first and third cycles are stable, whereas the second and fourth are critical.

3.1 Dynamic Variables Depend only on B

We first simplify the problem of determining the effect of a garbage collection cycle on the dynamic variables by showing that all values of the dynamic variables can be calculated from the sequence of values $B_0, B_1, B_2, \ldots$. There are two fundamental relations among the dynamic variables:

$$F_i + L + W_i + B_i = N \tag{1}$$

$$W_i = B_{i-1} \tag{2}$$

The first states that all list nodes at the end of the mark stage are either in the free list, in the working list, or are garbage (marked or unmarked). The second asserts that the unmarked garbage nodes at the end of a cycle (W_i) are precisely the nodes that entered the cycle as marked garbage (B_{i-1}); this assertion follows from our (worst-case) assumption that all new garbage produced during a cycle is marked and hence that all unmarked garbage comes from the previous cycle's marked garbage. Equations (1) and (2) show that the values of B_i ($i = 0, 1, 2, \ldots$) can be used to calculate the values of the other two dynamic space variables; in fact, $W_i = B_{i-1}$ and $F_i = N - L - B_i - B_{i-1}$.

The dynamic time variables can also be determined from the values of B. Since the scan stage lasts until all memory nodes are scanned and the scan rate is one node per s time units, the duration of the scan stage is the same for every cycle:

$$tscan_i = sN \tag{3}$$

The length of the mark stage for the ith cycle depends only on the number of nodes that are marked, which can be expressed in terms of the dynamic space variables. The four-color collector must mark the working list (L) and the newly released garbage (B_i), at the rate of 1 node per m time units. Thus, for the four-color collector,

$$tmark_i = m(L + B_i). \tag{4a}$$

The three-color collector, on the other hand, must mark the free list (F_i), in addition to the working list and the new garbage:

$$tmark_i = m(F_i + L + B_i)$$

$$= m(N - B_{i-1}) \tag{4b}$$

(by equations (1) and (2))

The mutator releases 1 marked garbage node every r time units except when it is forced to wait for garbage collection, so the amount of productive time in the ith cycle is $tprod_i = rB_i$. Thus, the dynamic time variables can also be calculated easily from the values of B.

3.2 The Fundamental Relation for B

The number of garbage nodes (B_i) produced during the ith cycle depends on whether the mutator must wait during the cycle. In any case the number of garbage nodes released by the mutator during the ith cycle can be no more than the number of available nodes, so $A_i = F_{i-1} + W_{i-1} = N - L - B_{i-1}$ at the beginning of the cycle, since only the unmarked garbage nodes will be added to the free list during the cycle. If the cycle is stable, then garbage nodes are produced at the constant rate of 1 node per r time units, so $B_i = tcycle_i/r$ garbage nodes will be produced.

If the cycle is critical, then the free list is depleted before the end of the cycle. Thus, the number of garbage nodes produced during the cycle is A_i, which is strictly less than the number that would have been released had the mutator continued without interruption. Therefore, in this case $B_i = A_i = N - L - B_{i-1}$ and $B_i < tcycle_i/r$. It follows that the amount of marked garbage at the end of the ith cycle must satisfy the following fundamental relation, which can be expressed solely in terms of the values of B and the static variables:

$$B_i = \mathbf{min}(tcycle_i/r,\ N - L - B_{i-1}) \qquad (5)$$

and the cycle is stable precisely when the minimum is attained by the first term.

3.3 The CDE for the Three-Color Collector

By combining equations (3) and (4b), we can express the length of the ith cycle for the three-color collector entirely in terms of B_{i-1} and the static variables:

$$\begin{aligned} tcycle_i &= tscan_i + tmark_i \\ &= sN + m(N - B_{i-1}) = (m + s)N - mB_{i-1} \end{aligned}$$

Thus, the amount of marked garbage (B_i) after the ith cycle can be calculated from its values after previous cycles using (5):

$$\begin{aligned} B_i &= \mathbf{min}(tcycle_i/r,\ N - L - B_{i-1}) \\ &= \mathbf{min}(((m + s)N - mB_{i-1})/r,\ N - L - B_{i-1}) \qquad (6) \end{aligned}$$

The conditional difference equation is obtained by determining in terms of B_{i-1} when the ith cycle is stable, which occurs when the minimum in (6) is attained by the first term. By using the expression for $tcycle_i$ above, we find that the ith cycle is stable precisely when $(r - m)B_{i-1} < r(N - L) - (m + s)N$. Let B^{crit} be the value of B for which both sides of this inequality are equal. Since a cycle can only be stable if nodes are being marked faster than they are being released, we may assume that $r > m$ and we find that the CDE for the three-color collector is

$$\begin{aligned} B_i = \ & \textbf{if } r > m \textbf{ and } B_{i-1} < B^{crit} \\ & \quad \textbf{then } ((m + s)N - mB_{i-1})/r \\ & \qquad \{\text{and the cycle is stable}\} \\ & \quad \textbf{else } N - L - B_{i-1} \\ & \qquad \{\text{and the cycle is critical}\} \qquad (7) \end{aligned}$$

where

$$B^{crit} = \frac{r(1 - \pi) - (m + s)}{r - m} N.$$

3.4 The CDE for the Four-Color Collector

The CDE for the four-color collector is slightly more difficult to obtain than that for the three-color collector. This is because the length of the ith cycle ($tcycle_i$) depends on B_i (rather than simply B_{i-1}). In fact, from equations (3) and (4a) we see that the length of the ith cycle is $tcycle_i = sN + m(L + B_i)$, so by the fundamental relation for B, the marked garbage at the end of the ith cycle must satisfy

$$B_i = \mathbf{min}((sN + m(L + B_i))/r,\ N - L - B_{i-1}) \qquad (8a)$$

Again we need to find a Boolean expression that determines when the ith cycle is stable or, equivalently, when the minimum of the two terms in (8a) is the first term.

If $r \le m$, then every cycle will be critical, because the mutator will release garbage faster than it can be marked. Thus, we may assume for the moment that $r > m$. Under this assumption, the linear equation obtained by setting B_i to the first term in (8a) is $B_i = (sN + mL + mB_i)/r$, which has a unique (and positive) solution, $G^{crit} = (sN + mL)/(r - m)$, which depends only on the static variables. Since the minimum in the formula for B_i is attained by the first term precisely when the ith cycle is stable, G^{crit} is the amount of marked garbage produced during a stable cycle. When $r > m$, formula (8a) takes the simpler form below:

$$\begin{aligned} B_i &= \mathbf{min}(G^{crit},\ N - L - B_{i-1}) \\ &= N - L - \mathbf{max}(B^{crit},\ B_{i-1}) \quad \textbf{if} \quad r > m \qquad (8b) \end{aligned}$$

where we have used the fact that $G^{crit} + B^{crit} = N - L$. This discussion shows that the ith cycle will be stable if and only if $r > m$ and $B_{i-1} < B^{crit}$; so the CDE for the four-color collector is

$$\begin{aligned} B_i = \ & \textbf{if } r > m \textbf{ and } B_{i-1} < B^{crit} \\ & \quad \textbf{then } (sN + mL)/(r - m) \\ & \qquad \{\text{and the cycle is stable}\} \\ & \quad \textbf{else } N - L - B_{i-1} \\ & \qquad \{\text{and the cycle is critical}\} \qquad (9) \end{aligned}$$

3.5 An Example

The results of the previous sections have been used to produce the graphs in Figure 3a–c. They pertain to the four-color collector. In all three figures the mutator's graph contains 300K list nodes (so L = 300K), and the times to scan (s), mark (m), and release (r) are bound to 1, 5, and 20 μs (microseconds), respectively; the systems all start with no garbage nodes. The only difference among the three systems is in the amount of space (N) allocated to each system. Thus, the graphs show how the values of the dynamic space variables (F, W, B) vary over time and illustrate, in particular, the effects of

memory allocation on wait time.

In Figure 3a, 700K list nodes are allocated to the system. After one garbage collection cycle, the system settles into an equilibrium in which the free list varies from 110K to 210K (approximately) and so the mutator does not have to wait for garbage collection. In Figure 3b, 500K list nodes are allocated to the system. The system's performance becomes periodic after 2 cycles, and the mutator is forced to wait 1 second every 2 cycles. Since 2 cycles are completed every 5 seconds, the mutator spends 20 percent of its time in garbage collection waits. In Figure 3c, the situation has deteriorated even further. The system has been given only 400K list nodes. The mutator must wait 2.3 seconds during each 4.3 second period (53 percent of the time), and the free list is nonempty only during the odd numbered cycles.

4. THREE FUNDAMENTAL TYPES OF SOLUTIONS

In this section we show that on-the-fly garbage collectors exhibit only the three types of performance illustrated in the previous example: (stable) all cycles are stable, (alternating) the cycles are alternately critical and stable, and (critical) every cycle is critical. We also derive the following formulas, which specify which of the three cases holds in terms of the static variables and the initial amount of marked garbage. Let I^{crit} denote the interval $B^{crit} < B < N - L - B^{crit}$, which by convention will be the empty set if B^{crit} is greater than $N - L - B^{crit}$, i.e., if $B^{crit} > (N - L)/2$. The performance of the system will be shown to depend as follows on the static variables r, m, s, π, and the initial amount of marked garbage (B_0):

(stable):

$$r > m \text{ and } B^{crit} \geq (N - L)/2$$

(alternating):

$$r > m \text{ and } B^{crit} < (N - L)/2$$

$$\text{and } B_0 \text{ is not in } I^{crit}$$

(critical):

$$r \leq m \text{ or } B^{crit} < (N - L)/2 \text{ and } B_0 \text{ is in } I^{crit}$$

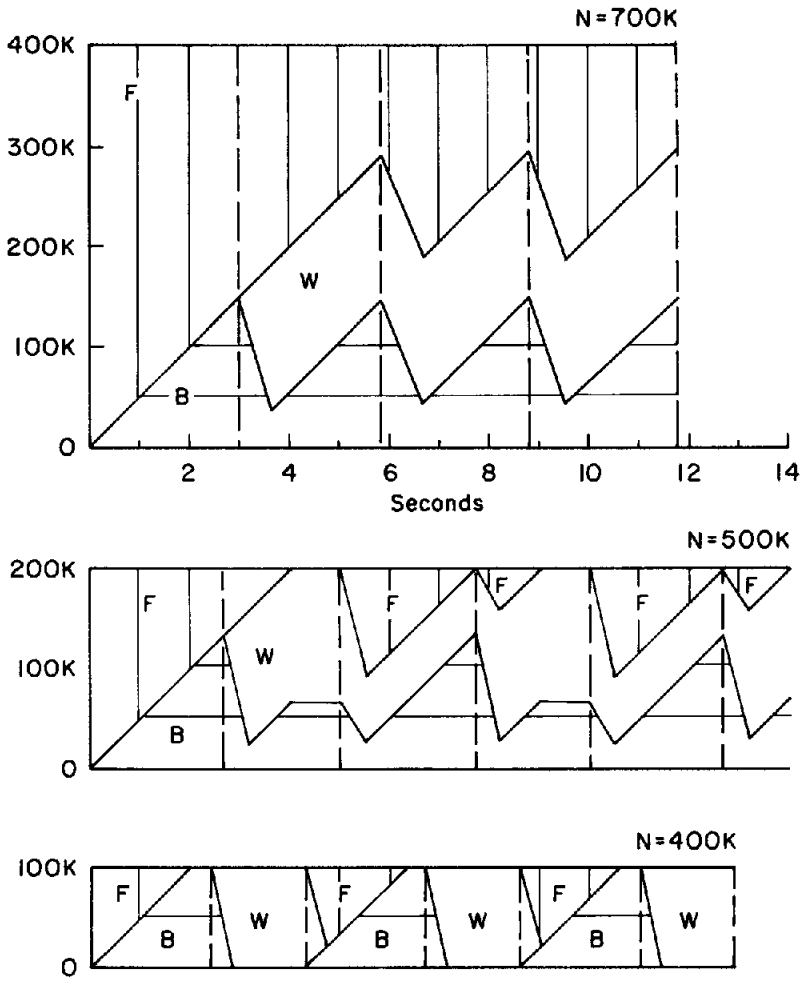

FIGURE 3. $s = 1\mu, m = 5\mu, r = 20\mu, L = 300K$

The values of all dynamic time and space variables for each of the three performance types will also be determined. A three-color collector may require several cycles of adjustment before it settles into one of these three performance types, whereas the four-color collector is more resilient and requires at most two cycles. This is essentially the only difference between the performance of the three- and four-color collectors. Only the four-color collector is analyzed here. The corresponding analysis for the three-color collector is similar.

4.1 The Stable Case

When $r > m$, the CDE for the four-color collector is $B_i = N - L - \mathbf{max}(B^{crit}, B_{i-1})$ and the ith cycle is stable precisely when B_{i-1} is less than B^{crit}. If we also have $B^{crit} \geq (N - L)/2$, then every cycle (except possibly the first) will be stable. Thus the amount of garbage produced will be $N - L - B^{crit}$ for all cycles except possibly the first. Since the other dynamic variables depend only on B, the system is periodic and repeats every cycle. (See Figure 3a for an example of a stable system.) The values of all the dynamic variables in the stable case are given in the second column of Table II, and can be verified using the results in Section 3 and in particular the relation $G^{crit} + B^{crit} = N - L$.

4.2 The Alternating and Critical Cases

Suppose now that $r > m$ and $B^{crit} < (N - L)/2$, then there are two cases depending on whether or not B_0 is in the interval I^{crit}. If B_0 is not in the interval I^{crit}, then after the first cycle it will be one of the endpoints of the interval, and B_i will oscillate between B^{crit} and $G^{crit} = N - L - B^{crit}$. When B_{i-1} is the lower endpoint (B^{crit}), the ith cycle ends just as the free list becomes empty and so the wait time is delayed until the next cycle. Figure 3b illustrates the periodicity of an alternating system. If B_0 is in the interval, or if $r \leq m$, every cycle will be critical and the amount of marked garbage will oscillate between B_0 and $N - L - B_0$, which by assumption are both greater than B^{crit}. Figure 3c shows a system in which $B^{crit} < 0$, and hence every cycle must be critical independent of the initial amount of marked garbage.

In either the alternating or the critical case, the value of B_i oscillates between a low value $B^{lo} \geq B^{crit}$ and a high value $B^{hi} \leq G^{crit}$, and these two values are related by the equation $B^{lo} + B^{hi} = N - L$. If B_0 is not in the interval I^{crit}, then these high and low values are precisely the endpoints of I^{crit}. Moreover, the amount of garbage produced in two consecutive cycles is $N - L$, which is independent of B_0. The values of the dynamic variables for each of the two cycles can be calculated from the formulas in Section 3 and appear in the third column of Table II.

TABLE II. Values of Dynamic Variables (where B^* is B^{lo} if i is odd and B^{hi} when i is even)

	Stable Case	Nonstable Cases
B_i	$G^{crit} = \frac{s + m\pi}{r - m} N$	B^*
W_i	G^{crit}	$N - L - B^*$
F_i	$N - L - 2 * G^{crit}$	0
$tscan_i$	sN	sN
$tmark_i$	$mL + mG^{crit}$	$m(B^* + L)$
$tprod_i$	$sN + mL + mG^{crit}$	rB^*
$tcycle_i$	$sN + mL + mG^{crit}$	$sN + mL + mB^*$
$twait_i$	0	$sN + mL - (r - m)B^*$

5. PERFORMANCE MEASURES

In this section, various performance measures are defined and discussed, including productivity, throughput, and wait times. Memory allocation, mutator speed, and the implementation of marking phase on these performance measures are examined. The performance of the parallel garbage collection system is also compared to a serial system. Since the performance of three-color collectors is similar to that of four-color collectors, we shall restrict our attention to four-color collectors.

5.1 Definitions

We shall consider three efficiency measures that can be applied to parallel garbage collectors: productivity, throughput, and maximum wait time. The average proportion of execution time for the system in which the mutator is active (not waiting) will be called the *productivity* of the system and will be denoted ρ. The productivity of a system in which there are never any garbage collection interruptions is 100 percent. Wadler [9] estimates that typical LISP implementations operate at 70–80 percent productivity.

A measure closely related to productivity is the *throughput* (denoted τ), which is the average rate at which nodes are taken from the free list (or, equivalently, released) by the mutator. For a system with release time r, productivity and throughput are directly proportional: $\rho = r\tau$. A third criterion for the viability of a system is the length of the garbage collection interruptions. If a system has high productivity (say 90 percent) but very long cycle lengths (say 300 seconds), it will occasionally have long garbage collection interruptions (30 seconds), which may be unacceptable for a particular application.

Parallel garbage collectors may sometimes provide a lower level of productivity or throughput than serial garbage collectors. Thus, the ratio of the productivity of the parallel collector to the productivity of a sequential version of the collector is also an important measure of system performance. This ratio will be called the productivity speedup. (Notice that throughput speedup, with the evident definition, is equal to productivity speedup.) If the productivity speedup is less than one, the serial version of the system will be more efficient.

5.2 Formulas for the Performance Measures

The formulas for the performance measures depend on which of the three cases discussed in Section 4 holds. If the system is stable, then the productivity (ρ) will be 1, the throughput (τ) will be $1/r$, and there will be no wait time. The productivity speedup (σ) will be the reciprocal of the productivity (ρ_{serial}) of a serial collector, which will be determined below.

If the system is not stable, then its performance repeats every two cycles. The productivity can then be obtained by taking the ratio of the amount of productive time over two consecutive cycles by the total length of the two cycles. Recall that if B^{lo} and B^{hi} are the repeating values of B, then $B^{lo} + B^{hi} = N - L$. Thus we find from Table II that the length of two consecutive cycles is $2sN + m(N + L)$ and the amount of productive time is $r(N - L)$. Thus, the productivity in any of the three cases is

$$\rho = \min\left(1, \frac{r(N-L)}{2sN + m(N+L)}\right)$$

$$= \min\left(1, \frac{r(1-\pi)}{2s + (1+\pi)m}\right)$$

The productivity for a serial version of the collector is just as easy to calculate. The mutator will use all of the $N - L$ nodes on the free list at a constant rate (1 node/r time units) and then must stop for garbage collection during which the L working list cells will be marked at a constant rate (1 node/m time units) and the garbage cells will be appended during the scan stage, which lasts sN time units as before. Thus the productivity for the serial garbage collector is

$$\rho_{serial} = \frac{r(N-L)}{r(N-L) + mL + sN}$$

$$= \frac{r(1-\pi)}{(r+s) - (r-m)\pi}$$

The productivity speedup is the quotient of these two productivities:

$$\sigma = \frac{\rho}{\rho_{serial}}$$

$$= \min\left(\frac{(r+s) - (r-m)\pi}{2s + (1+\pi)m}, \frac{(r+s) - (r-m)\pi}{r(1-\pi)}\right)$$

The productivity and productivity speedup depend on the proportion of memory being used by the mutator (π) and not the actual size of the mutator (N). The throughput (τ) can be easily calculated from the productivity, since $\tau = \rho/r$.

The length of a garbage collection interruption in the alternating or critical cases will depend on the initial amount of garbage (B_0). The worst case occurs when there is no garbage initially ($B_0 = 0$). In any case, the maximum wait time is the sum of the wait times over a two-cycle period: $waittime = \max((2s + (1+\pi)m - (1-\pi)r)N, 0)$.

6. APPLICATIONS OF THE PERFORMANCE ANALYSIS

6.1 Allocation of Memory for Garbage Collection Systems

The analysis in Section 5 can be applied to the problem of determining the optimum memory size to allocate to a garbage collection system, relative to a given performance measure. In this context, the size of the working list L is given and the size of the memory available to the system N is to be chosen. The dependence of the productivity, productivity speedup, and maximum waittime on memory allocation is illustrated in Figure 4 for a typical system ($s = 1\mu$, $m = 5\mu$, $r = 20\mu$). Since productivity depends only on (s, m, r, and π) and is independent of N, Figure 4 can also be interpreted as a portrayal of the system's performance when the size of the mutator changes (i.e., as L changes) and the total memory allocation (N) remains fixed.

Let π_{stable} denote the largest value of π for which the system is stable, and let π_{speed} denote the largest value for which the productivity speedup is at least one. From the previous section we see that

$$\pi_{stable} = \max\left(0, 1 - 2\frac{m+s}{m+r}\right)$$

$$\pi_{speed} = \max\left(0, 1 - \frac{m+s}{r}\right)$$

Thus the system is stable if $0 \le \pi < \pi_{stable}$. Even if the system is not stable, the productivity of the parallel system will be larger than that of the serial system when $\pi_{stable} \le \pi < \pi_{speed}$. The maximum speedup, which occurs when $\pi = \pi_{stable}$, depends only on the ratio m/r:

$$\sigma_{\max} = \frac{(r+s) - (r-m)\pi_{stable}}{(m+2s) + m\pi_{stable}}$$

$$= 1.5 - \frac{m}{2r}$$

In particular, if $r < m$ then the parallel system will perform less efficiently than the serial system, no matter how much memory is allocated. This equation implies that the productivity of the parallel collector is at most 150 percent of that of the serial collector.

6.2 Mutator Speed and Performance

Let $\lambda = m/r$ denote the relative mutator speed. Let λ_{stable} be the fastest mutator speed for which the system is stable and λ_{speed} the fastest speed for which the productivity speedup is at least one. Thus the system is stable if $\lambda < \lambda_{stable}$ and is more productive than the serial collector if $\lambda < \lambda_{speed}$. The formulas in Section 5 can be used to show that

$$\lambda_{stable} = \frac{(1-\pi)}{(1+\pi) + 2(s/m)}$$

$$\lambda_{speed} = \frac{(1-\pi)}{1 + (s/m)}$$

and that the maximum productivity speedup occurs when $\lambda = \lambda_{stable}$. Observe that when the system is not stable ($\lambda > \lambda_{stable}$), the throughput is

$$r = \frac{1-\pi}{2s + (1+\pi)m}$$

which is independent of the mutator speed. Hence, increases in mutator speed have no effect on throughput, although they do lower productivity. The effect of rela-

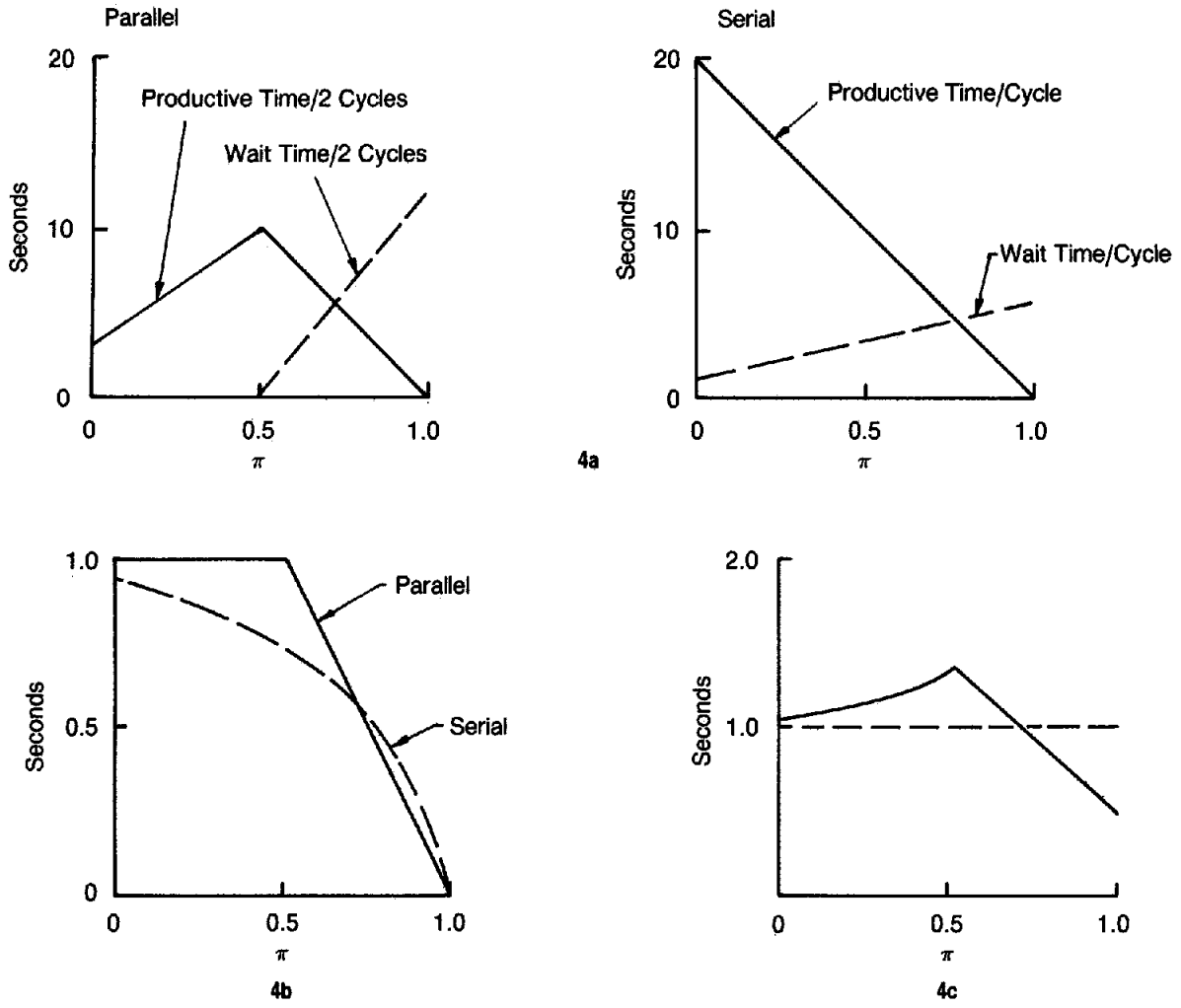

FIGURE 4a. Productive and Wait Times for Parallel and Serial Garbage Collection Systems as π Varies ($s = 1\mu$, $m = 5\mu$, $r = 20\mu$)
FIGURE 4b. Productivity of Parallel and Serial Garbage Collection as π Varies ($s = 1\mu$, $m = 5\mu$, $r = 20\mu$)
FIGURE 4c. Productivity Speedup as π Varies (s, m, r) = (1, 5, 20)

tive mutator speed on the various performance measures for a typical system ($s = 1\mu$, $m = 5\mu$, $\pi = 0.5$, $N = 1000$K) is shown in Figure 5.

6.3 Mutator Speed, Mutator Size, and Performance

We now examine the performance of a garbage collection system as a function of the relative speed of the mutator ($\lambda = m/r$) *and* the proportion of memory occupied by the user's list ($\pi = L/N$). Suppose the total memory allocation (N) is fixed, as are the garbage collector parameters (s and m). The mutator can then be described by two parameters: the proportion of memory occupied by the mutator's graph (π) and the speed of the mutator relative to the marking speed (λ).

In Figure 6 the dependence of system performance on the pair (π, λ) is illustrated. The productivity speedup (σ) is displayed in Figure 6a as a function of π and λ over the region in which it is at least one. The view is from the point (λ, π, σ) = (4, −4, 4), and the origin (0, 0, 0) is the lower left corner of the graph. The flat plane on the right side of the figure is where the productivity speedup is less than one. Figure 6b shows the regions in the (π, σ) coordinate plane where the system is stable, alternating, or critical, assuming that there is no initial marked garbage ($B_0 = 0$) and that $s = 1\mu$ and $m = 5\mu$. These three regions correspond precisely to the increasing, decreasing, and flat regions of Figure 6a. Observe that for any fixed π the productivity speedup increases as the relative mutator speed increases from zero to the boundary of the stable region. It then decreases rapidly and quickly becomes less than one.

7. RELATED WORK

One of the first analyses of parallel garbage collection appears in Wadler's paper [9] in 1976. He gives a sufficient condition for a garbage collection system to be stable [9, Cor. to Thm. 1]. The result is phrased in such a way that it can apply to any garbage collection system provided one knows differentiable functions (of time) giving upper and lower bounds on the size of the mutator's graph. Adapting the corollary to our situation and notation yields a sufficient condition for a stable sys-

tem: $L \leq N - 2r^{-1}t_{max}$ where t_{max} is the maximum length of a garbage collection cycle. We can obtain an expression for this quantity by setting $B_0 = N - L$ in the formula for $tcycle_1$ in Section 3.4. Making this substitution and simplifying yields Wadler's condition:

(CondW)

$$1 - \pi)r \geq 2(m + s) = (1 + \pi)m + 2s + (1 - \pi)m$$

This condition is consistent with (but weaker than) our precise condition

(stable) $$(1 - \pi)r \geq (1 + \pi)m + 2s$$

for a system in equilibrium to be stable. Wadler also provides theoretical support for the assumption that most of the garbage produced by a mutator will be marked [9, Section 4].

A four-color collector was introduced by Kung and Song [5] in 1977 in which the mark operation was implemented using an output restricted deque.[1] This deque resides in a memory separate from the N nodes of list memory and must be at least twice as large as the working list (the precise size requirement depends on the speed of the deque operations). In an appendix of [5], Kung and Song list the results of an (unpublished) performance analysis in the stable case. Under assumptions on the speed of the deque operations and other parameters specific to their algorithm, they present a sufficient condition (CondKS) for their system to be stable.

(CondKS)

$$(1 - \pi)r \geq (1 + \pi)m + 2s - \frac{1}{2}\left[(1 - \pi)m + 2s + \pi\frac{m^2}{r}\right]$$

This condition is consistent with our analysis of general four-color collectors.

[1] An output restricted deque is a stack in which nodes can be slipped under, in addition to being pushed on and popped off.

8. FINAL REMARKS

We conclude by mentioning a couple of related areas that are worth investigating. Perhaps the best way to improve the efficiency of a garbage collection system is to use several processors for garbage collection. Lamport [6] informally describes an algorithm that would allow any number of processors to be used in a garbage collection system, both as mutators or collectors. The method of using CDEs to determine the productivity of a system should apply equally well to an analysis of Lamport's system.

A more ambitious garbage collector is the one described in Lieberman and Hewitt [7], which is based on the lifetimes of objects and is designed to be used in a virtual memory environment. The analysis of garbage collection in a very large virtual memory space is qualitatively different from the analysis performed here. In principle, garbage collection could be bypassed altogether, although this would degrade performance by scattering the active nodes sparsely throughout mem-

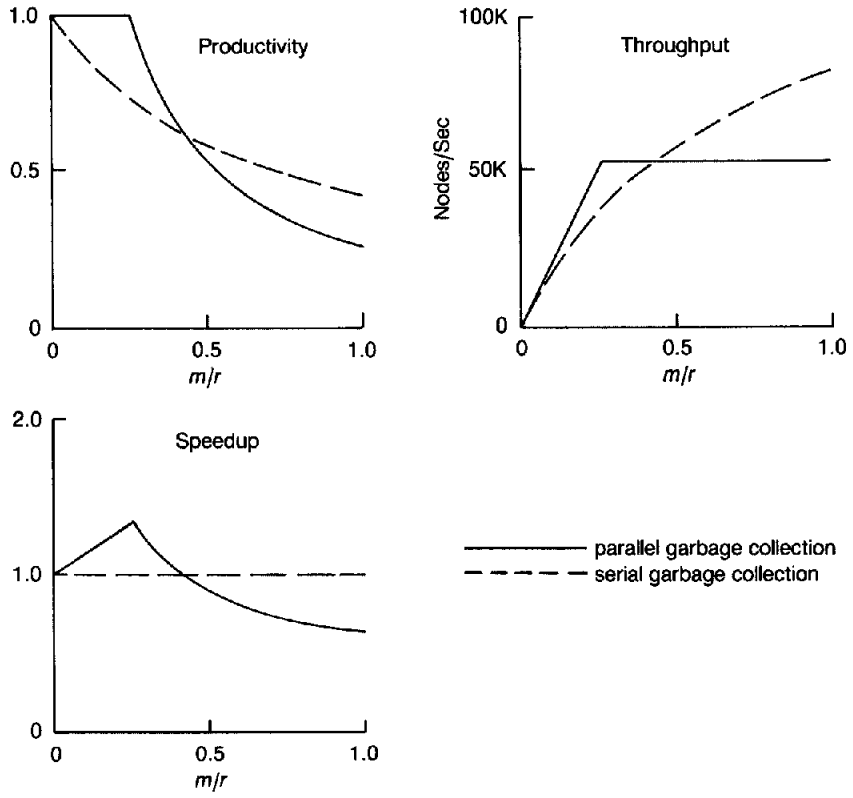

FIGURE 5. Effect of Mutator Speed on Performance

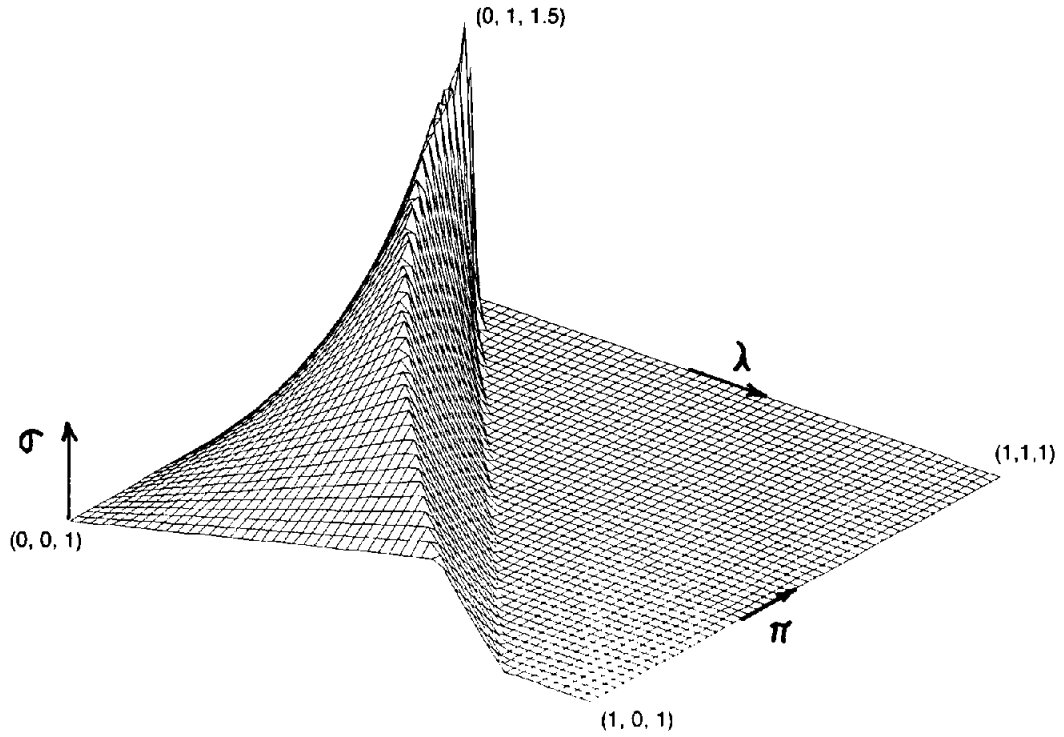

FIGURE 6a. Productivity Speedup on a Four-Color Collector ($s = 1, m = 5$)

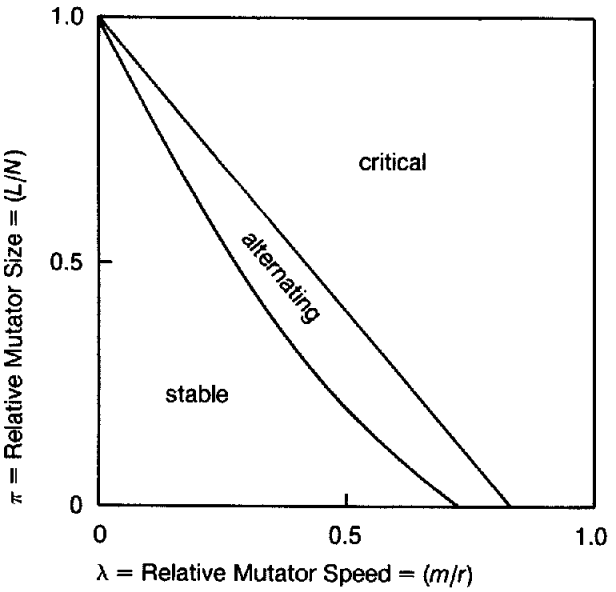

FIGURE 6b. Performance of a Four-Color Collector ($s = 1, m = 5$)

ory and thereby requiring frequent paging. Any analysis of virtual memory garbage collection schemes will need to estimate the frequency of paging, and hence the locality of pointer reference will be an important parameter.

REFERENCES

1. Cohen, J. Garbage collection of linked data structures. *Comput. Surv. 13*, 3 (Sept. 1981), 341–367.
2. Cohen, J. Computer-assisted microanalysis of programs. *Commun. ACM 25*, 10 (Oct. 1982), 724–733.
3. Dijkstra, E.W., et al. *On-The-Fly Garbage Collection: An Exercise in Cooperation.* (Notes for the 1975 NATO Summer School on Language Hierarchies and Interfaces.) Lecture Notes in Computer Science, vol. 46. Springer-Verlag, New York, 1976.
4. Gries, D. An exercise in proving parallel programs correct. *Commun. ACM 20*, 12 (Dec. 1977), 921–930.
5. Kung, H., and Song, S. An efficient parallel garbage collection system and its correctness proof. Tech. Note, Dept. of Computer Sciences, Carnegie-Mellon Univ., Pittsburgh, Pa., Sept. 1977.
6. Lamport, L. Garbage collection with multiple processes: An exercise in parallelism. In *Proceedings of the 1976 International Conference on Parallel Processing*, P.H. Enslow, Jr., Ed. IEEE Computer Society, Long Beach, Calif., 1976, pp. 50–54.
7. Lieberman, H., and Hewitt, C. A real-time garbage collector based on the lifetimes of objects. *Commun. ACM 26*, 6 (June 1983), 419–429.
8. Steele, G.L., Jr. Multiprocessing compactifying garbage collection. *Commun. ACM 18*, 9 (Sept. 1975), 495–508.
9. Wadler, P.L. Analysis of an algorithm for real-time garbage collection. *Commun. ACM 19*, 9 (Sept. 1976), 491–500.

CR Categories and Subject Descriptors: D.1.3 [**Programming Techniques**]: Concurrent Programming; D.4.2 [**Operating Systems**]: Storage Management; E.1 [**Data Structures**]; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: list processing, parallel garbage collection, marking algorithms, speedup, efficiency

Received 7/84; accepted 7/84

Authors' Present Address: Tim Hickey and Jacques Cohen, Computer Science Dept., Brandeis University, Waltham, MA 02254.