DESIGN AND ANALYSIS OF PARALLEL MONTE CARLO ALGORITHMS

BY

V.C. BHAVSAR
AND
J.R. ISAAC

TR85-030, DECEMBER 1985

DESIGN AND ANALYSIS OF PARALLEL MONTE CARLO ALGORITHMS

V.C. BHAVSAR
SCHOOL OF COMPUTER SCIENCE
UNIVERSITY OF NEW BRUNSWICK
FREDERICTON, N.B., E3B 5A3, CANADA

AND

J.R. ISAAC
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY
BOMBAY, 400076, INDIA

DESIGN AND ANALYSIS OF PARALLEL MONTE CARLO ALGORITHMS

ABSTRACT

This paper demonstrates that the potential of intrinsic parallelism in Monte Carlo methods, which has remained essentially untapped so far, can be exploited to implement these methods efficiently on SIMD and MIMD computers. Two basic static and dynamic computation assignment schemes are proposed for assigning the primary estimate computations (PECs) to processors in a parallel computer. These schemes can be used to design parallel Monte Carlo algorithms for many applications. The time complexity analyses of static computation assignment (SCA) schemes are carried out using some results from order statistics, whereas those of dynamic computation assignment (DCA) schemes are carried out using results from order statistics, renewal and queuing theories. It is shown that for smaller number of processors, linear speedup can be achieved with the SCA schemes and the speedup almost equal to the number of processors can be achieved with the DCA schemes. Some computational results for Monte Carlo solutions of Laplace's equation are given to illustrate the performance of the various SCA and DCA schemes.

INDEX TERMS

Monte Carlo methods, parallel algorithms, parallel computers, analysis of algorithms, order statistics, renewal theory, pseudorandom number generators, partial differential equations.

I. INTRODUCTION

Monte Carlo methods are widely used in many scientific and engineering applications [1-6]. In these methods, the solution of a problem is represented as a parameter of a hypothetical population. Then, a random sequence of numbers is used to construct a sample of that population from which statistical estimates of the parameter are obtained [1]. The solution estimates coverge slowly (expected error is $O(\sqrt{K})$, where K is the sample size) to the exact solution and therefore a large sample size (K equal to 10^3 to 10^5 is quite common) is required to obtain a reasonable accuracy. The estimate computations often involve random walk processes, with some random walks consisting of a large number of steps. As a consequence of these two characteristics, these methods require large computing time when implemented on sequential computers and they have not been very attractive. In this paper, we present some techniques of implementing these methods on parallel computers with a motivation to reduce their execution time and we hope that this will make these techniques more attractive in future. Recently, Monte Carlo simulation has been identified as one of the most challenging areas for the application of parallel computers [7].

Parallel algorithms can be developed by exploiting the intrinsic parallelism in sequential algorithms. Intrinsic parallelism in Monte Carlo methods at the level of primary estimate computations (PECs) was first indicated, surprisingly, as early as in 1949 by Metropolis and Ulam in their paper defining the Monte Carlo method [8]. Parallelism at this level, as also within the computations of primary estimates was exploited only to some extent in [9,10] to solve partial differential equations (PDEs) on special purpose parallel computers. Thus, the potential of

intrinsic parallelism in Monte Carlo methods had remained essentially untapped in their implementation on parallel computers, until our recent work [11-19]. Recently, the application of ICL DAP in solving some problems in solid-state and elementary particle physics has been reported [2].

In [11], we suggested the use of an SIMD computer, based on bit-slice microprocessors, to implement Monte Carlo methods for solving PDEs. In [12] simulation results about the effectiveness of SIMD and MIMD computer architectures for solving PDEs by Monte Carlo methods were presented. We have given some parallel algorithms for Monte Carlo solutions of linear equations in [13]. The Monte Carlo methods for solving PDEs were reconsidered in [14] and the simulation results of a parallel algorithm based on an idea by Rosin [9] were presented. This preliminary work culminated in the development of a unified treatment for the design and analysis of parallel Monte Carlo algorithms given in [15]. In [19], we have summarized this recent work and our work reported in [15]. Recently, we have also considered the problem of implementing Monte Carlo algorithms with VLSI technology. In this paper, we detail some of the ideas presented in earlier papers and present the unified treatment given in [15].

This paper is organized as follows. In Section II, we briefly discuss the computational characteristics of Monte Carlo methods and show that the primary estimate computation (PEC) times are independent and identically distributed (i.i.d.) random variables. This crucial result is shown to have significant influence on the design and analysis of parallel Monte Carlo algorithms in Section III. We propose two basic schemes for assigning PECs to processors in a parallel computer: static computation assignment (SCA) and dynamic computation assignment (DCA) schemes. These schemes can be used to design parallel algorithms for any Monte Carlo method. In Section IV, we describe the various SCA schemes. The details of

the parallel algorithms are not included for the sake of brevity and they are given in [15]. The time complexity analyses of parallel Monte Carlo algorithms based on these schemes are carried out using some results from order statistics. Two DCA schemes are presented in Section V. The time complexity analyses of these schemes are carried out using results from order statistics, renewal and queuing theories. In Section VI, we consider the standardized exponentially distributed PEC times and derive the time complexity results for the SCA and DCA schemes. We consider the problem of solving Laplace's equation by a Monte Carlo Method in Section VII. Some computational results illustrating the performance of the various schemes are presented. Concluding remarks are given in Section VIII.

II. SEQUENTIAL MONTE CARLO ALGORITHMS

A. The Monte Carlo Method

The Monte Carlo method is generally defined as representing the solution of a problem as a parameter of a hypothetical population, and using a random sequence of numbers to construct a sample of the population, from which statistical estimates of the parameter can be obtained [1]. In practice, usually, the required solution consists of a real number, which is expressed as the expected value of some random variable X with finite variance on a probability space. A set of point $\zeta_1, \zeta_2, \ldots, \zeta_K$ are sampled independently from the probability space, commonly using pseudorandom or quasirandom number generators. In our discussion, we will assume that "ideal" random number generators are used. A set of primary estimates, $X(\zeta_1), X(\zeta_2), \ldots, X(\zeta_K)$ are obtained. The secondary estimate of the solution is obtained by the arithmetic mean

$$Y = Y(\zeta_1, \zeta_2, \ldots, \zeta_K) = \frac{1}{K} \sum_{i=1}^{K} X(\zeta_i) , \qquad (1)$$

which converges to the required solution in quadratic mean, in probability, and with probability one, as $K \to \infty$. The sampling of each point ζ_i from the population of the probability space often involves lengthy computations, such as tracing a random walk in an intricate and highly inhomogeneous space. In these cases, determination of a primary estimate usually consists of a run through either an absorbing or an ergodic Markov chain. For example, such computations are involved in Monte Carlo solutions of linear algebraic equations [1] and partial differential equations [10].

B. Time Complexity of Sequential Monte Carlo Algorithms

In general, the primary estimate computations (PECs) have the following characteristics:

C1. They are essentially independent of each other because $\zeta_1, \zeta_2, \ldots, \zeta_K$ are sampled independently from the probability space using ideal random number generators.

C2. They consist of the execution of identical steps.

C3. Their execution times are, in general, random variables because: (i) they may consist of run through an absorbing Markov chain (i.e. a random walk) and the number of state transitions determine their execution times, and (ii) the number of comparisons required to make decisions based on a sample from random number generators is variable.

Lemma 1: The PEC execution times are independent and identically distributed (i.i.d.) random variables.

Proof: The PEC execution times are random variables due to C3. The property of independency follows from C1 and identicalness follows from C2.

□

This is a very crucial result in the design and analysis of parallel Monte Carlo algorithms, as discussed in Section III.

Now, let μ and σ^2 denote the mean and variance of the i.i.d. PEC times. Also, let T(1,K) denote the time complexity of a sequential Monte Carlo algorithm in computing K primary estimates. Since in many applications of the Monte Carlo méthod the secondary estimate computation time turns out to be much smaller than the PEC times, in the ensuing analysis we will not take into account this time; if this time is not negligible in some cases, the time complexity results given in this paper can be trivially modified, as shown in [15].

Theorem 1: The time complexity of a sequential Monte Carlo algorithm carrying out K PECs, T(1,K), has the expectation $E[T(1,K)]=K.\mu$, and the variance $Var[T(1,K)] = K \cdot \sigma^2$.

Proof: Follows directly from Lemma 1.

□

Thus, it is seen that a sequential Monte Carlo algorithm has O(K) expected time complexity and also O(K) time complexity variance. In practice, it is required to choose a large value of K (10^3-10^5 is not uncommon) in order to attain a reasonable accuracy and hence by the Central Limit Theorem it follows that T(1,K) will tend to follow a normal distribution with mean $(K.\mu)$ and variance $(K\sigma^2)$. The cumulative distribution function of a sequential Monte Carlo algorithm can be easily found out by using Theorem 1 and the techniques discussed in [15,20].

III. DESIGN AND ANALYSIS OF PARALLEL MONTE CARLO ALGORITHMS - APPROACH AND ASSUMPTIONS

In this section, we discuss our approach for the development of parallel Monte Carlo algorithms. First, we detail the intrinsic parallelism in

Monte Carlo methods, identified in our earlier papers [11-14] and discuss its implications in the design of the parallel algorithms and formulate our approach. We introduce two basic schemes for the development of parallel Monte Carlo algorithms. Finally, we state the assumptions which are used in the time complexity analysis of parallel algorithms.

A. Intrinsic Parallelism

We identify the intrinsic parallelism in Monte Carlo methods at four levels as follows.

1) Parallelism between computations of variables: In Monte Carlo methods for problems involving many variables, the estimates of each of the variables can usually be obtained independently of the remaining variables and consequently their computations can be executed in parallel. For example, the estimates of each of the unknowns in a set of linear algebraic equations can be obtained independently of the other unknowns [1,13]. The complete independence of the computations of variables implies that the computational modules' granularities will be large and therefore these computations are very well suited for implementation on MIMD computer architectures.

Further, the computational algorithms of each of these variables are usually identical, but use different data. As a result, these computations are also very well suited for implementation on SIMD computer architectures.

We have for the first time exploited the intrinsic parallelism at this level and used it in solving a set of linear algebraic equations [13].

2) Parallelism in secondary estimate computation: As defined in Section II, the secondary estimate of the desired solution is usually obtained by averaging the values of K primary estimates. In a SIMD computer, this summation can obviously be carried out in $[\log_2 K]$ time steps, if required interprocessor communication network is available. In a MIMD computer, some part of the summation time may be overlapped with other computations, as shown in ensuing sections. We have exploited the parallelism at this level in solving linear algebraic equations [13] and partial differential equations [14].

3) Parallelism between different PECs: The required K PECs to be carried out for a variable are completely independent of each other (the characteristic C1). Consequently, these PECs can be executed in parallel. Since each of the PECs often involves a lengthy computation, the computational module granularities are large enough to make these computations suitable for implementation on MIMD computers. Moreover, the PECs are also very well suited for implementation on SIMD computers due to the characteristic C2.

The parallelism at this level was hinted at as early as in 1949, by Metropolis and Ulam [8], in their paper defining the Monte Carlo method. However, its use was first suggested in [9] and subsequently it was also exploited in [10] for solving PDEs with special purpose computers. In [9] the use of a drum array processor was suggested, whereas in [10] a macro-modular computer was used. Recently, we have exploited the parallelism at this level in solving linear algebraic equations and PDEs on SIMD and MIMD computers.

4) Parallelism in a PEC: In general, a PEC involves: (i) multidimensional random number generation, (ii) updating the value of a primary

estimate based on a sample from random number generators, and (iii) determination of the value of a termination condition for the computation. The intrinsic parallelism exists in all these steps.

The generation of each of the components in multidimensional random number generation can be carried out in parallel. In updating the value of a primary estimate, the intervals in which a random number sample falls has to be determined and this involves many decision operations, which can be executed in parallel. Finally, the step (iii) also involves many comparison operations, which can be carried out in parallel.

In [10] pipelining has been used to reduce the execution time, whereas in [11] we have suggested the use of hardware random number generators.

B. Design Approach

From the above discussion, it is now evident that Monte Carlo methods have a large potential for intrinsic parallelism, which has remained almost untapped so far. Further, this intrinsic parallelism can be exploited equally well for implementation on SIMD as well as MIMD computers.

In our design and analysis of parallel Monte Carlo algorithms, we consider the intrinsic parallelism only between the computations of different PECs for a variable, because these computations primarily determine the order of the time complexities of these algorithms. The results given in this paper can be trivially extended for problems involving the estimation of more than one variables, because such computations are independent of each other. If the intrinsic parallelism within each of the PECs is exploited, this usually changes the time complexity only by some constant, its order remaining unchanged.

We view the problem of designing parallel Monte Carlo algorithms exploiting the intrinsic parallelism between the PECs, as that of the assignment of PECs to processors in a parallel computer, the PEC times being i.i.d. random variables. Alternatively, each PEC can be considered as a task with random execution time. Then, we will have tasks with i.i.d. execution times and the problem can be considered as basically the scheduling of independent tasks onto multiple processors with task execution times as i.i.d. random variables.

Although the problem of scheduling a set of tasks on multiple processors has been discussed in literature, much of it assumes constant execution times (see e.g. [21-23]), and only the treatment in [20,24,25] is with the task execution times as random variables. Motivated by the problem decomposition strategies proposed in [20], we propose two basic schemes, static and dynamic, for the assignment of PECs to processors in a parallel computer. In a static computation assignment (SCA) scheme, a fixed number of PECs are assigned to each of the processors before any PECs are initiated. In contrast, in a dynamic computation assignment (DCA) scheme, the number of PECs carried out by each processor is determined during the computations, based on some global criteria.

In the SCA schemes, very small (if at all) time overheads are incurred to make the computation assignment decisions during the computations. The parallel algorithms based on these schemes cannot adapt to the variations in the PEC times (which are random variables), in the sense that the computation time is minimized based on the run-time behavior. In DCA schemes, since the assignment of PECs to processors is done during execution, the execution time of a parallel algorithm can be minimized based on its run-time behavior. Obviously, a larger time overhead is

incurred in making the computation assignment decisions. It should be noted that these schemes can be used for developing parallel algorithms for almost any Monte Carlo method and hence are fundamental in nature. The various SCA and DCA schemes for parallel Monte Carlo algorithms are discussed in Sections IV and V.

Since, usually a large number of primary estimates are used (10^3 - 10^5 is not uncommon), parallel Monte Carlo algorithms may involve simultaneous operation of a large number of pseudorandom number (PRN) generators. Obviously, PRN sequences of smaller lengths will be used on each of the processors, compared to the corresponding sequential algorithms. Thus, the PRN generators should be such that the PRN sequences generated exhibit desired statistical properties for smaller lengths. In addition, the PRN sequences generated in various processors, pooled together, should satisfy the desirable properties, e.g. small cross-correlation. These and other issues related to the pseudo-random number generation will be discussed in [36].

C. Analysis of Parallel Monte Carlo Algorithms

In many Monte Carlo applications, PEC time distributions cannot be determined a priori. In some cases, even if it may be possible to determine the distributions, the computations involved may be too time consuming. For example, the determination of the expectation of PEC times in solving an elliptic PDE itself involves the solution of another elliptic PDE, i.e. the problem of determining the expectation has the same order of time complexity as that of the problem being solved with the Monte Carlo method [15]. Therefore in the analysis of parallel Monte Carlo algorithms we obtain the time complexity results which are independent of the probability distributions of the PEC times (i.e. distribution-free) and in closed forms.

As in sequential Monte Carlo algorithms, we assume that the secondary estimate computation time is negligible. Further, we assume, as usual in literature (see e.g. [20,24,27]): (i) the use of identical processors, (ii) the register load/store and communication costs as negligible, (iii) no memory or data alignment penalties, and (iv) any one of the four arithmetic operations being performed in one time unit. Further in the analysis of MIMD algorithms, we assume, as in [20]: (i) for any runnable process a processor is always assigned, and (ii) initially all runnable processes start execution at the same time. Finally, to enable easier comparison of the performance of various algorithms we assume that, in a parallel algorithm the PEC time characteristics (e.g. their mean, variance, etc.) remain exactly same as in the corresponding sequential algorithm. This implies that, for example, the possible increase in the PEC times due to conditional branches in a SIMD algorithm (see [28]) is considered as negligible. Also, the consequences of multiprocessor system

characteristics, as discussed in [20,25], are neglected. However, such changes in the PEC times can be implicitly taken into account by suitably modifying the statistical characteristics of the PEC times. As a consequence of the stated assumptions, the computational time behavior and the time complexities of the SIMD and MIMD algorithms corresponding to a sequential algorithm, will turn out to be the same.

Since the PEC times are random variables, in the analysis of parallel Monte Carlo algorithms we have to employ the techniques of order statistics, queuing theory, and renewal theory, as discussed in Sections IV and V.

IV. STATIC COMPUTATION ASSIGNMENT (SCA) SCHEMES

In SCA schemes for parallel Monte Carlo algorithms, we assign a fixed number of PECs to the processors before any PECs are initiated, as discussed in Section III. We propose four SCA schemes for the development of SIMD and MIMD algorithms, dependent upon the number of processors (P) used:

(i) SCA Scheme 1: $P \ll K$,

(ii) SCA Scheme 2: $K/P \gtrsim 10$,

(iii) SCA Scheme 3: $P=K$,

(iv) SCA Scheme 4: $P>K$.

In the Schemes 1 and 2, we assign K PECs among P processors in such a way that their sum is equal to K and each processor is assigned the same number of PECs, as far as possible; obviously, if P divides K exactly then we will assign (K/P) PECs to each of the processors. In Scheme 3, we assign only one PEC to each of the processors. In Scheme 4 also we assign one PEC each of the processors, but terminate the computations in all active processors once K PECs are completed. The first three schemes can be considered to be

as the mere special cases of a single scheme. However, we have chosen to treat them separately because the SIMD and MIMD algorithms based on them differ (see [15] for details) and moreover, their analyses have to be carried out differently.

A. SCA Scheme 1

Assume that M=(K/P), where M is an integer. Then, in this scheme we would obtain K primary estimates when all the processors complete M PECs assigned to each of them. The expected time complexity of SIMD and MIMD algorithms based on this scheme is given in the following.

Theorem 2: The expected time complexity of parallel Monte Carlo algorithms based on SCA Scheme 1 is given as

$$\frac{K\mu}{P} \leq E[T(P,K)] \leq \frac{K\mu}{P} + \frac{\sqrt{K}.\sigma}{2} \qquad (2)$$

Proof: The execution time of the i-th processor in carrying out M PECs, say t_i, is given by the sum of the execution times of each of the PECs. By Lemma 1, the PEC times are i.i.d. random variables and therefore the computation times $t_1, t_2, \ldots, t_p$ will also be i.i.d. random variables. Since P<<K, M will be a large number and then by the Central Limit Theorem it follows that t_i's will follow a normal distribution with mean $(M.\mu)$ and variance $(M.\sigma^2)$.

The execution time of the parallel algorithms, T(P,K), is given by the random variable

$$T(P,K) = \max [t_1, t_2, \ldots, t_p].$$

Thus T(P,K) represents the extreme order statistic of the i.i.d normal random variables, t_1, t_2, ... t_p. Since a closed form expression is not available for such an order statistic, we use the upper bound on the extreme order statistic for any symmetrically distributed i.i.d. random

variables given in [29 , p.62] and obtain the upper bound given in the theorem.

The lower bound on the time complexity follows from the fact that $E[\max(t_1,t_2, \ldots, t_P)] \geq \max [E(t_1), E(t_2), \ldots, E(t_P)] = M.\mu$, which is true for any set of random variables.

□

Thus, SCA Scheme 1 parallel Monte Carlo algorithms have $O(K)$ expected time complexity and, comparing with the time complexity of sequential Monte Carlo algorithms, they achieve $O(P)$ speedup, i.e. linear speedup.

If P=2, then the expected time complexity of these parallel algorithms can be given as [24]

$$E[T(P,K)] \cong \frac{K\mu}{P} + \sqrt{\frac{K}{\pi P}} \cdot \sigma \quad , \tag{3}$$

as $K \to \infty$.

B. SCA Scheme 2

In this scheme, we have $(K/P) = M \sim 10$ and we assign K PECs among P processors, as in Scheme 1. In Scheme 1, since M is large, the i.i.d. random variables $t_1, t_2, \ldots, t_P$ were considered as normal, whereas in this scheme, since M is smaller, such an approximation may not be acceptable.

Theorem 3: The SIMD and MIMD algorithms based on Scheme 2 have the expected time complexity

$$\frac{K\mu}{P} \leq E[T(P,K)] \leq \frac{K\mu}{P} + \frac{(P-1)\sqrt{K}}{\sqrt{(2P-1)P}} \cdot \sigma \tag{4}$$

Proof: The upper bound follows from the distribution-free upper bound on the extreme order statistic of i.i.d. random variables given in [30, pp. 46-47] and the lower bound remains same as in Theorem 2.

□

In practice, K is usually a large number (10^3 to 10^5 is not uncommon) and since K/P ~ 10, we will usually have P>>1. In this case, the upper bound in (4) can be simplified to the following.

$$E[T(P,K)] \leq \frac{K\mu}{P} + \sqrt{\frac{K}{2}} .\sigma \quad .$$

It can be easily seen that the lower bound on the speed up for SCA Scheme 2 parallel algorithms will be $O(\sqrt{P})$.

C. SCA Scheme 3

In this scheme, we assign one PEC to each processor. Thus the execution time of the parallel algorithms based on this scheme will be given by the extreme order statistic of K i.i.d. random variables represented by PEC times and we have the following.

Theorem 4: The SCA Scheme 3 parallel Monte Carlo algorithms have the expected time complexity.

$$\mu \leq E\,[T(P,K)] \leq \mu + \frac{(K-1).\sigma}{\sqrt{(2K-1)}} . \qquad (5)$$

Proof: Follows directly by using the distribution-free upper bound on the extreme order statistic given in [30].

□

The asymptotic time complexity of these parallel algorithms will be given as

$$\mu \leq E[T(P,K)] \leq \mu + O(\sqrt{K}) \qquad (6)$$

with asymptotic speedup over the corresponding sequential algorithms as

$$O(\sqrt{K}) \leq S(K,K) \leq K.$$

In a Scheme 3 SIMD algorithm, we would use a global counter to keep track of the completed PECs. The global counter will be updated by only

one processor at a time and the remaining processors in the SIMD computer will be idle. Consequently, the time complexity result would get modified to the following.

$$\mu + K\tau \leq E[T(K,K)] \leq \mu + \frac{(K-1)\sigma}{\sqrt{(2K-1)}} + K\tau \quad , \tag{7}$$

where τ is the time required to update the global counter by a processor. Thus, the expected time complexity will turn out to be $O(K)$, whereas by neglecting such an updating time, the lower bound is a constant and the upper bound is $O(\sqrt{K})$. This demonstrates that the sequential program segments in an SIMD algorithm may increase the order of time complexity. In Scheme 3 MIMD algorithms, the global counter updating time may not contribute much time overhead, because this time will be overlapped with the PEC times of active processors.

It is interesting to compare the cumulative distribution function (cdf) of the time complexity of sequential Monte Carlo algorithms with the Scheme 3 parallel algorithms. In the sequential algorithms, the cdf will be given by the K-fold convolution, where as in the parallel algorithms it is given by the K-th power [15]. Consequently, the variance of the time complexity of parallel Monte Carlo algorithms will turn out to be much smaller than that of the corresponding sequential algorithms. This is an attractive feature of these parallel algorithms.

D. SCA Scheme 4

In this scheme, we have $P > K$. We assign one PEC to each processor with a motivation to minimize the execution time of the parallel Monte Carlo algorithms by terminating the execution soon after we have K PECs

completed out of the initiated P PECs. Unfortunately, this may result in a biased estimate of the desired solution as discussed below.

In Monte Carlo methods, the execution time of PECs is usually related to either the number of terms considered from a finite or an infinite series (e.g. Neumann series used in solving linear equations [1]), or the lengths of random walks (e.g. studies in nuclear physics [3], molecular chemistry [4]). As a result, the execution time is related to the accuracy of the estimates obtained. In the present scheme, since we always choose the earliest completing K PECs, we are effectively limiting the number of terms from a series or truncating lengths of random walks. Consequently, in this scheme we may obtain biased estimates.

In [15], we have given an SIMD algorithm based on this scheme. We have also given two MIMD algorithms, one with and the other without interruption of processes, as suggested in [24] for designing MIMD algorithms.

The time complexity of Scheme 4 parallel Monte Carlo algorithms is represented by the Kth order statistic of P i.i.d. PEC times, with $P > K$. Unfortunately, distribution-free closed form expressions are not possible for such an order statistic. However, tables are available for normal distribution (see [30, p. 28]), gamma distribution, logistic distribution, etc. (see [30, pp. 226-227]). In section VI, we give the expected time complexity results for Scheme 4 parallel algorithms, when the PEC times are exponentially distributed i.i.d. random variables.

V. DYNAMIC COMPUTATION ASSIGNMENT (DCA) SCHEMES

In DCA parallel Monte Carlo algorithms, whenever a PEC is completed on a processor, the decision as to whether or not to initiate another PEC on that processor is made during the execution, based on some global criteria.

The main motivation is to minimize their execution time. Here, we propose two basic schemes for such parallel algorithms. Scheme 1 can be used in the development of parallel Monte Carlo algorithms employing arbitrary number of processors. Scheme 2 is designed specifically for implementations with P<K. The closed form distribution-free time complexity results for these parallel algorithms can be obtained only when P<<K and hence we consider here only such cases in the time complexity analysis.

A. DCA Scheme 1

In this scheme, given P processors, first we initiate P PECs in parallel. Whenever a PEC is completed on a processor, we initiate another PEC on that processor, if K PECs have not already been completed. Obviously, in this scheme we will initiate more than the required K PECs. The details of the parallel Monte Carlo algorithms are given in [15].

Fig. 1 illustrates a possible execution time profile for DCA Scheme 1 parallel algorithms (under various assumptions stated in Section III) in obtaining nine primary estimates with three processors, P1, P2, and P3. The time $t_j^{(i)}$ represents the time instant at which the j-th PEC is completed by the i-th processor. The behavior of the parallel algorithms, taking into account computations on all the three processors, is represented by the virtual processor P*. For P*, time t_i represents the time instant at which the ith PEC is completed. It is seen that, when the required number of PECs are completed, the incomplete PECs on the remaining (P-1) processors get aborted. The aborted computation time is significant only in MIMD algorithms; in SIMD algorithms this time is not significant, because if these (P-1) processors did not carry out the computations which later got aborted, any way they would have been idle. In order to eliminate the

wasteful computations in the DCA Scheme 1 MIMD algorithms, we give DCA Scheme 2 in the next subsection.

Theorem 5: The DCA Scheme 1 parallel Monte Carlo algorithms, carrying out K complete PECs with P processors, have the expected time complexity

$$E[T(P,K)] \simeq \frac{K\mu}{P} + \frac{(P-1)(\mu^2 - \sigma^2)}{2P\mu} \tag{8}$$

Proof: The time $t_j^{(i)}$, as shown in Fig. 1, represents the sum of j i.i.d. PEC times. Hence, the computational behavior of the ith processor is an ordinary renewal process [31, p. 25].

Since all the PEC times are i.i.d. random variables, a DCA Scheme 1 parallel algorithm consists of P independent ordinary renewal processes in simultaneous operation, all with the same probability distribution. Therefore, the behavior of the parallel algorithm, represented by P*, can be considered as the behavior of the superposition of the constituent P ordinary renewal processes. Then, the theorem follows by using the result in [31, pp. 73-75].

□

Thus, DCA Scheme 1 parallel Monte Carlo algorithms have O(K) expected time complexity and comparing with the time complexity of sequential algorithms, they achieve speedup approximately equal to P, i.e. ideal speedup. If K is large, as usually it will be, then the computing time on each processor can be considered as normally distributed and then by using result in [31, p.73] we have

$$Var[T(P,K)] = Var[T(1,K)]/P^2. \tag{9}$$

Again, this shows that the variance of the time complexity of parallel algorithms may be much smaller than that of corresponding sequential algorithms.

In DCA Scheme 1 MIMD algorithms, we will have a critical region to update the global counter for keeping track of completed PECs and to update the secondary estimate. If the critical region execution time for the constituent processes in such parallel algorithms in not to be neglected and the contention among them to execute the critical region is considered, then behavior of these algorithms can be modelled as a queuing system, similar to the models of time shared computer systems as shown in Fig. 2. Such a queuing system has been extensively studied as a model of single processor timeshared computer systems (see [32, pp. 27-31],[33, pp. 144-154]), when the probability distributions of all the servers are exponential. Assuming that the mean critical region execution time is equal to τ_c, the steady-state probability that no process will be executing the critical region is given by

$$\pi_o = \frac{1}{\sum_{j=0}^{P} \frac{P! r^j}{(P-j)!}}$$

where $r = (\tau_c/\mu)$.

Since P<<K, each process will execute a large number of PECs and therefore the steady-state solution is justifiable. Then, the expected time complexity of DCA Scheme 1 MIMD algorithms will be given as

$$E[T(P,K)] = \frac{\tau_c \cdot K}{(1 - \pi_0)} \quad . \qquad (11)$$

B. DCA Scheme 2

This scheme is designed specifically for MIMD algorithms. In this scheme, given P processors, first we initiate P PECs in parallel. Whenever a PEC is complete, we initiate another PEC on that processor only if the

required K PECs have not been already initiated. Thus, in this scheme we initiate only K PECs and therefore no PECs get aborted as in DCA Scheme 1.

Fig. 3 illustrates a possible execution time profile the DCA Scheme 2 parallel Monte Carlo algorithms, assuming that nine primary estimates are to be computed using three processors. It is seen that, when the ninth PEC is initiated at time t_9^*, six PECs are already complete and two are in progress. When the longest of these three PECs completes at time t_9, we obtain the required nine primary estimates. Thus, in general, when K-th PEC is initiated, (K-P) PECs are already complete and (P-1) PECs are in progress. When the largest of these P PECs terminates, we obtain the required K primary estimates.

Theorem 6: The DCA Scheme 2 MIMD algorithms, carrying out K PECs with P processors (P<<K) have the expected time complexity

$$\tau \lesssim E[T(P,K)] \lesssim \tau + \frac{(P-1)\,.\,\sigma}{\sqrt{(2p-1)}} \quad , \tag{12}$$

where

$$\tau = \frac{K\mu}{P} + \frac{(P-1)(\mu^2 - \sigma^2)}{2P\mu} \quad .$$

Proof: When the K-th PEC is initiated, (K-P) PECs are complete, say at time $t_{(P-K)}$. The time $t_{(P-K)}$ is a random variable representing the time instant at which (K-P) renewals are completed in the superposed renewal process of the constituent P ordinary renewal processes, each ordinary renewal process comprising the i.i.d. PEC times. Therefore, by using the result in [31, p. 75], the expectation of $t_{(P-K)}$ will be given as

$$E[t_{(P-K)}] \simeq \frac{(K-P)\mu}{P} + \frac{(P-1)(\mu^2 - \sigma^2)}{2P\mu} \tag{13}$$

Immediately after $t_{(P-K)}$, we have one new PEC and (P-1) PECs already in progress. Then, the additional time required after $t_{(P-K)}$ to complete these P PECs, say t_p, will be given as

$$t_p = \max \text{ [residual times of (P-1) PECs, lifetime of new PEC]}$$

$$= \max [t_r^{(1)}, t_r^{(2)}, \ldots, t_r^{(P-1)}, t_m] \qquad (14)$$

where $t_r^{(i)}$ is a random variable representing the residual lifetime of i-th PEC, and t_m is the random variable representing the lifetime of the new PEC. The time t_p is the extreme order statistic of independent, but <u>not</u> identically distributed, P random variables. Since distribution-free closed form expressions are not possible for the expectation of the right-hand-side of (14), we obtain the upper bound on the expectation of t_p as

$$E[t_p] \leq \max \text{ [life times of P PECs]}$$

$$= \mu + \frac{(P-1).\sigma}{\sqrt{(2P-1)}} \qquad (15)$$

The lower bound on $E[t_p]$ is given as follows.

$$E[t_p] = E[\max\{t_r^{(1)}, t_r^{(2)}, \ldots, t_r^{(P-1)}, t_m\}]$$

$$\geq \max \{E[t_r^{(1)}], E[t_r^{(2)}], \ldots, E[t_r^{(P-1)}], E[t_m]\}$$

$$\geq \mu . \qquad (16)$$

Now, the theorem follows from (14), (15) and (16).

□

Thus, the DCA Scheme 2 MIMD algorithms have O(K) time complexity and, comparing with sequential algorithms, they achieve speedup approximately equal to P.

VI. AN EXAMPLE

In this section, we illustrate the performance of the sequential and parallel Monte Carlo algorithms, when the PEC times are standardized i.i.d. exponential random variables; i.e. their mean $\mu=1$, standard deviation $\sigma=1$, and their probability distribution function (pdf) given by

$$f(t) = e^{-t}, \ (t \geq 0). \tag{17}$$

Then the pdf of the time complexity of sequential Monte Carlo algorithms will be given by

$F_{(1,K)}$ = K-fold convolution of $f(t)$

$$= \frac{t^{(K-1)} . e^{-t}}{(K-1)!} \ . \tag{18}$$

Since in practice usually K is large, the random variable $T(1,K)$ will be asymptotically normally distributed. Further, we have

$$E[T(1,K)] = K, \tag{19}$$

and $$Var[T(1,K)] = K. \tag{20}$$

A. Performance of SCA Schemes

The performance of Schemes 1 and 2 can be found out by using (2), (4), and (5) is given in Table 2.

In Scheme 3, the cdf of the execution time of the parallel algorithms will be given by

$$F_{(K,K)} = [1 - e^{-t}]^{(K-1)} \tag{21}$$

and their pdf as

$$f_{(K,K)} = K \, e^{-t} \, [1-e^{-t}]^{(K-1)} \ . \tag{22}$$

Further, it can be easily derived that

$$E[T(K,K)] = \sum_{i=1}^{K} i^{-1}$$

$$= H_K \qquad (23a)$$

$$= \log_e K, \text{ (if K is very large)}, \qquad (23b)$$

where H_K is the K-th harmonic number. Also, we would obtain

$$\text{Var } [T(K,K)] = \sum_{i=1}^{K} i^{-2}, \qquad (24)$$

and if K is very large, we can use the result in [34, p. 75] and obtain

$$\text{Var } [T(K,K)] = \frac{\pi^2}{6} . \qquad (25)$$

Thus for SCA Scheme 3, the expected time complexity increases logarithmically as against linear increase in sequential Monte Carlo algorithms. Moreover, the variance of the parallel Monte Carlo algorithms is bounded for large values of K, whereas it increases linearly for the sequential Monte Carlo algorithms.

For SCA Scheme 4, we had noted that distribution-free time complexity results are not possible. However, we can determine the expected time complexity for the present example. It can be easily shown that the Kth order statistic of P i.i.d. PEC times considered here is given as

$$E[T(P,K)] = \sum_{i=(P-K+1)}^{P} i^{-1}$$

$$= H_P - H_{(P-K)} \qquad (26)$$

$$\simeq \log_e P - \log_e (P-K), \text{ if } (P-K) >> 1 . \qquad (27)$$

The variance of the time complexity will be given by

$$\text{Var}[T(P,K)] = \sum_{i=(P-K+1)}^{P} i^{-2} . \qquad (28)$$

Thus, it is seen that the expectation as well as the variance of the time complexity of SCA Scheme 4 parallel Monte Carlo algorithms can be reduced by increasing the number of processors.

B. Performance of DCA Schemes

The performance of DCA Scheme 1 using (8) will be given as

$$E[T(P,K)] = \frac{K}{P} . \tag{29}$$

This implies that we can achieve speedup exactly equal to P, the one which is always desired and rarely achieved in practice.

It can be easily proved that the expected time complexity of DCA Scheme 2 will be given by

$$E[T(P,K)] = \frac{K}{P} + H_P . \tag{30}$$

The expected time complexity results given in this section are summarized in Table 2.

VII. COMPUTATIONAL RESULTS

In this section, we present some computational results illustrating the performance of the schemes for parallel Monte Carlo algorithms given in Section IV and V. We consider the Laplace's equation defined on a rectangle,

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = 0 \quad , \quad 0 < x < 6, \; 0 < y < 10 , \tag{31}$$

with boundary conditions

$$U(x,0) = 10, \quad U(x,10) = U(0,y) = U(6,y) = -10 .$$

This problem is selected for two reasons: (i) the analytic solution to this problem can be easily obtained and the Monte Carlo solutions could be compared, and (ii) the same equation has been considered in [10].

In solving PDEs by Monte Carlo methods, the solution a point in the given region can be obtained independently of the solution at other points [10,11]. We have selected three representative points in the region given in (31): the first point as (1,1) near the boundary, the second point (3,5) at the centre of the region and the third point (2,3) in between these two points. In the method described in [10], the given region is replaced by a square mesh, as is in a finite-difference method. Fig. 4 shows the given region divided into a mesh, with mesh size $h = \frac{1}{4}$. Now a set of random walks are defined on the set of states comprising [10]: (i) one transient state corresponding to each of the internal nodes (i.e. the unknowns), and (ii) one absorbing state for each of the boundary nodes. In solving (31), the state transition probabilities for a step from any one internal node to its four neighboring nodes turns out to be equal to 0.25. Equation (31) was solved on a sequential computer for obtaining solutions at the selected three points and the Monte Carlo solutions were in good agreement with the analytic and numerical solutions when 2048 primary estimates were used (see [15]). The random walk duration statistics was collected for solutions at each of the selected points. Since parallel computers were not accessible, we have used the statistics collected for estimating the performance of various schemes for parallel algorithms. We consider the computational times of 2048 primary estimates on parallel computers, with the number of processors varying from 2,4,8,...,2048 in logarithmic steps, the computational time is being in terms of the number of random walk steps (i.e. the duration of random walks) in a primary estimate computation.

A. SCA Schemes

The first three SCA schemes discussed in Section IV are considered to be as follows:

1. Scheme (P<<K): for P = 2,4,8,..., 128.
2. Scheme (K/P~10): for P = 128, 256, 512, and 1024.
3. Scheme (P=K): for P = 2048.

Now, we use the sample mean and the standard deviation of the durations of random walks to obtain the bounds on the speedup of the above three schemes, using the results in Theorems 2,3 and 4. Figures 5,6, and 7 depict the upper and lower bounds on the speedup expected in solving (31) at points (1,1), (2,3), and (3,5), respectively, on a log-log scale. It is seen that, until about P=64 there is no significant difference between the upper and lower bounds. The difference increases rapidly as the number of processors are increased beyond this point. At P=128, the lower bounds obtained by Scheme 1 and Scheme 2 are indicated by points A and B, respectively.

We estimate the performance of SCA parallel algorithms based on the following. It is known that given N i.i.d. random processes operating in parallel, the characteristics of the ensemble can be derived from successive observations on a single source, due to the ergodicity principle [36]. In the present case, we use the data generated during Monte Carlo solutions on a sequential computer and estimate the performance of SCA schemes as follows. We divide the data about random walk durations into P disjoint sets with only M successive samples in each set. The execution time on a sequential computer is represented by the sum of the durations of all 2048 random walks in a run, whereas the execution time on a parallel computer is given by the maximum of the sum of the durations of random walks for each of the P sets. The speedup estimated for Monte Carlo solutions at the

chosen three points using by the various SCA Schemes is given in Figures 5,6, and 7,for two runs.

We make the following observations from Figures 5,6, and 7. The estimated speedup follows closely the lower bound, until about P=32, but being always less than this bound. Here, it may be expected that the estimated performance should lie between the upper and lower bounds. However, it may be recalled that the speedup is defined as the ratio of the expectations of the execution times on sequential and parallel computers [20], and here we have presented the results only for two runs. From P=64 to P=2048, the estimated speedup is almost always within the upper and lower bounds. In many cases, the slope of the estimated speedup decreases as the number of processors increases, as expected.

B. DCA Schemes

The speedup using the two DCA schemes from Section V is computed using Theorems 5 and 6 by substituting the sample mean and the standard deviation of the durations of random walks, and the results are shown in Figures 5,6, and 7. It is seen that for Scheme 1 the speedup closely follows a linear speedup. Further, for the solution at point (1,1) the speedup is greater that P, due to the fact that in this case $\mu<\sigma$ (refer to (8)). For Scheme 2, the upper bound on the speedup is the same as that of Scheme 1 by Theorem 6. The lower bound on the speedup starts drooping after about $P>256$, due to the denominator expression in (8). It is not possible to estimate the performance of DCA schemes from the data generated from the execution of sequential algorithms, as discussed in [15, p. 189].

VIII. CONCLUSION

In this paper, we have demonstrated that the potential of intrinsic parallelism in Monte Carlo methods, which had remained essentially untapped so far, can be exploited for implementing these methods efficiently on parallel computers.

We have shown that the primary estimate computation (PEC) times in these methods turn out to be independent identically distributed random variables. This crucial result is shown to greatly influence the design and analysis of parallel Monte Carlo algorithms.

We have identified the intrinsic parallelism in these methods at four levels. The potential of this intrinsic parallelism is shown to be equally well suited for exploitation on SIMD as well as MIMD computer architectures. The computations involved have large module granularity, little concurrency control requirement and they do not demand specialized interprocessor interconnection networks.

Since the PEC times primarily determine the time complexity of a Monte Carlo method, we have proposed two basic schemes for developing parallel Monte Carlo algorithms utilizing intrinsic parallelism between PECs. In static computation assignment (SCA) schemes we assign a fixed number of PECs to the processors before any PECs are initiated, whereas in dynamic computation assignment (DCA) schemes the decision as to whether or not to initiate another PEC on a processor, whenever it completes a PEC, is made during the execution based on some global criteria. The DCA parallel Monte Carlo algorithms are more adaptable to the variations in the PEC times. The SCA and DCA schemes for parallel Monte Carlo algorithms can be applied for almost any Monte Carlo methods. In [15], we have illustrated their application in developing parallel algorithms for Monte Carlo methods used

in solving linear algebraic equations and partial differential equations, and the details of the parallel algorithms are also given. In this paper, we have derived the time complexity results of these schemes. The analyses of SCA schemes were carried out using some results from order statistics, whereas those of DCA schemes were carried out using these results as well as some results from queuing theory and renewal theory. The time complexity results obtained for these schemes are independent of PEC time distributions and therefore are universal. The importance of such distribution- free results stems from the fact that for many Monte Carlo methods it is almost impossible to obtain the PEC time distributions a priori (see [15, pp. 92-165]). These schemes are shown to have achieve either O(P) or almost equal to P speedup, when the number of processors (P) is much smaller than the required number of primary estimates. We have shown that when the PEC times are standardized exponential random variables, it is possible to obtain exact time complexity results, rather than bounds.

Finally, we have illusrated the performance of the various schemes for parallel Monte Carlo algorithms with the solution of Laplace's equation.

The treatment in the paper assumes that "ideal" random number generators are used to drive Monte Carlo computations. However, since pseudorandom number generators are usually employed in practice, we have discussed some problems in pseudorandom number generation in parallel computations in the Appendix. Further research is needed in this area.

We hope that the results in this paper, in [15], and our recent work [17-19] generates interest in the design and analysis of parallel Monte Carlo algorithms.

REFERENCES

[1] J.H. Halton, "A retrospective and prospective survey of the Monte Carlo method," SIAM Review, vol. 12, pp. 1-63, 1970.

[2] K.C. Bowler and G.S. Pawley, "Molecular dynamics and Monte Carlo simulations in solid-state and elementary particle physics," Proc. IEEE, vol. 72, no. 1, pp. 42-55, Jan. 1984.

[3] J. Spanier and E.M. Gelbard, Monte Carlo Principles and Neutron Transport Problems. Reading, MA: Addison-Wesley, 1969.

[4] J.M. Hammersley and D.C. Handscomb, Monte Carlo Methods. London: Methuen, 1964.

[5] Yu. A. Schreider (Ed.), The Monte Carlo Method. Oxford: Pergamon Press, 1966.

[6] K. Binder (Ed.), Monte Carlo Methods in Statistical Physics. Berlin: Springer-Verlag, 1979.

[7] L.S. Haynes, R.L. Lau, D.P. Siewiorek, and D.W. Mizell, "A survey of highly parallel computing," Computer, vol. 15, no. 1, pp. 9-24, Jan. 1982.

[8] M. Metropolis and S.M. Ulam, "The Monte Carlo method," J. Amer. Statist. Assoc., vol. 44, pp. 335-341, 1949.

[9] R.F. Rosin, "An algorithm for concurrent random walks on highly parallel machines," Cooley Electronics Lab., University of Michigan, Ann Arbor, TR 151, 1964.

[10] E. Sadeh and M.A. Franklin, "Monte Carlo solution of partial differential equations by special purpose digital computer," IEEE Trans. Comput., vol. C-23, pp. 389-397, 1974.

[11] V.C. Bhavsar and V.V. Kanetkar, "A multiple microprocessor system (MMPS) for the Monte Carlo solution of partial differential equations," in Advances in Computer Methods for Partial Differential Equations, vol. 2, R. Vichnevetsky (Ed), New Brunswick: IMACS, pp.205-213, 1977.

[12] V.C. Bhavsar and A.J. Padgaonkar, "Effectiveness of some parallel computer architectures for Monte Carlo solution of partial differential equations," in Advances in Computer Methods for Partial Differential Equations, vol. 3, R. Vichnevetsky and R.S. Stepleman (Ed.), New Brunswick: IMACS, pp. 259-264, 1979.

[13] V.C. Bhavsar and J.R. Isaac, "Some parallel algorithms for Monte Carlo solution of partial differential equations," in Proc. 15th Annual Convention of Computer Society of India, Bombay, Feb. 8-11, 1980, vol. III, pp. 3.76-3.81, 1980.

[14] V.C. Bhavsar, "Some parallel algorithms for Monte Carlo solutions of partial differential equations," in Advances in Computer Methods for Partial Differential Equations, vol. 4, R. Vichnevetsky and R.S. Stepleman (Ed.), New Brunswick: IMACS, pp. 135-141, 1981.

[15] V.C. Bhavsar, "Parallel algorithms for Monte Carlo solutions of some linear operator problems," Ph.D. Thesis, Department of Electrical Engineering, Indian Institute of Technology, Bombay, Nov. 1981.

[16] V.C. Bhavsar and J.R. Isaac, "Design and analysis of parallel algorithms for Monte Carlo methods," in Proc. 10th IMACS World Congress on 'System Simulation and Scientific Computation', Montreal, Canada, Aug. 8-13, 1982, vol. 2, pp. 323-325.

[17] V.C. Bhavsar, "VLSI algorithms for Monte Carlo solution of linear equations," presented at the APICS Annual Computer Science Seminar, St. Francis Xavier Univ., Antigonish, N.S., Canada, 4-5, Nov. 1983.

[18] V.C. Bhavsar, "VLSI algorithms for Monte Carlo solutions of partial differential equations," to appear in Proc. Fifth IMACS International Symp. on 'Computer Methods for Partial Differential Equations', Bethlehem, Pa., June 1984.

[19] V.C. Bhavsar, "Design and analysis of parallel Monte Carlo algorithms: Some results and new directions," submitted to 1984 Int. Conf. on Parallel Processing, Aug. 1984.

[20] J.T. Robinson, "Some analysis techniques for asnychronous multiprocessor algorithms," IEEE Trans Software Eng., vol. SE-5, pp. 24-31, Jan. 1979.

[21] D.J. Kuck, "A survey of parallel machine organization and programming," Computing Surveys, vol. 9, no. 1, 1977.

[22] M.J. Gonzalez Jr., "Deterministic processor scheduling," Computing Surveys, vol. 9, pp. 173-204, 1977.

[23] E.G. Coffman Jr. (Ed.), Computer and Job-Shop Scheduling Theory. New York: John Wiley, 1976.

[24] H.T. Kung, "Synchronized and asynchronous parallel algorithms for multiprocessors," in Algorithms and Complexity: New Directions and Recent Results, J.F. Traub (Ed.). New York: Academic Press, pp. 153-200, 1976.

[25] G.M. Baudet, "The design and analysis of algorithms for asynchronous multiprocessors," Ph.D. Dissertation, Carnegie-Mellon University, 1978.

[26] H. Niederreiter, "Quasi-Monte Carlo methods and pseudorandom numbers," Bull. Ame. Math. Soc., vol. 84, pp. 957-1041, 1978.

[27] D. Heller, "A survey of parallel algorithms in numerical linear algebra," SIAM Review, vol. 20, pp. 740-777, 1978.

[28] H.T. Kung, "The structure of parallel algorithms," Carnegie Mellon Univ., Dept. of Computer Science, CMU-CS-79-149, 1979. Also in Advances in Computers, M.C. Yovits (Ed.), New York: Academic Press, vol. 19, 1980.

[29] E.J. Gumbel, "Statistical theory of extreme values (main results)," as Chapter 6 in Contributions to Order Statistics, Sarahan, A.E. and B.G. Greenberg (Ed.). New York: John Wiley, 1962.

[30] H.A. David, Order Statistics. New York: John Wiley, 1970.

[31] D.R. Cox, Renewal Theory. London: Methuen, 1962.

[32] A.L. Scherr, An Analysis of Time-Shared Computer Systems. Cambridge: M.I.T. Press, 1966.

[33] H. Kobayashi, Modeling and Analysis: An Introduction to System Performance Evaluation Methodology. Reading, Mass: Addison-Wesley, 1978.

[34] D.E. Knuth, The Art of Computer Programming, vol. 1, Fundamental Algorithms. Reading, Mass: Addison-Wesley, 1968.

[35] E.V. Krishnamurthy and S.K. Sen, Computer-Based Numerical Algorithms. New Delhi: Affiliated East West Press, 1976.

[36] V.C. Bhavsar, "On Pseudo-random Number Generators for Parallel Computers," In preparation.

TABLE 1

Pec (Primary estimate computation) time Distribution-Free Time Complexity Results for the Various Schemes for Parallel Monte Carlo Algorithms

SCHEME	EXPECTED TIME COMPLEXITY
SCA Scheme 1 P << K	$\frac{K\mu}{P} \leq E[T(P,K)] \leq \frac{K\mu}{P} + \frac{\sqrt{K}.\sigma}{2}$ If P=2, $E[T(P,K)] \cong \frac{K\mu}{P} + \sqrt{\frac{K}{\pi P}}.\ \sigma$, as $K \to \infty$.
SCA Scheme 2 K/P ~ 10	$\frac{K\mu}{P} \leq E[T(P,K)] \leq \frac{K\mu}{P} + \frac{(P-1).\sqrt{K}}{\sqrt{(2P-1)P}} \cdot \sigma$
SCA Scheme 3 P = K	$\mu \leq E[T(P,K)] \leq \mu + \frac{(K-1)}{\sqrt{2K-1}} \cdot \sigma$
SCA Scheme 4 P > K	Pec-time Distribution-Free Time Complexity results are not possible.
DCA Scheme 1	$E[T(P,K)] \cong \frac{K\mu}{P} + \frac{(P-1)(\mu^2-\sigma^2)}{2P.\mu}$
DCA Scheme 2	$\tau \leq E[T(P,K)] \leq \tau \frac{(P-1)}{\sqrt{2P-1}} \cdot \sigma$ where $\tau = \frac{K\mu}{P} + \frac{(P-1)(\mu^2-\sigma^2)}{2P\mu}$

K = The desired total number of primary estimates of a problem variable

M = mean of i.i.d pec times, σ = standard deviation of i.i.d. pec times

P = number of processors

Note: The time complexity of a sequential Monte Carlo algorithm is given as

$E[T(1,K)]=K.\mu$

TABLE 2

Expected Time Complexity of Parallel Algorithms based on various SCA and DCA Schemes, when Pec-times are i.i.d. exponential random variables with $\mu = 1$ and $\sigma = 1$. Time Complexity of Sequential Algorithm = K.

SCHEME	TIME COMPLEXITY
SCA Scheme 1 P << K	$\frac{K}{P} \leq E[T(P,K)] \leq \frac{K}{P} + \frac{\sqrt{K}}{2}$ If P = 2, $E[T(P,K)] \cong \frac{K}{2} + \frac{\sqrt{K}}{\sqrt{2\pi}}$
SCA Scheme 2 K/P ~ 10	$\frac{K}{P} \leq E[T(P, K)] \leq \frac{K}{P} + \frac{(P-1)\sqrt{K}}{\sqrt{(2P-1)P}}$ If P >> 1, then $\frac{K}{P} \leq E[T(P, K)] \leq \frac{K}{P} + \sqrt{\frac{K}{2}}$.
SCA Scheme 3 P = K	$E[T((P,K)] = H_K \cong \log_e K$
SCS Scheme 4 P > K	$E[T(P,K)] = H_P - H_{(P-K)}$ $\cong \log_e P - \log_e (P-K)$
DCA Scheme 1 P << K	$E[T(P,K)] = \frac{K}{P}$
DCA Scheme 2 P << K	$E[T(P,K)] = \frac{K}{P} + H_P$

(Note: H_n represents the n-th harmonic number)

FIGURE CAPTIONS

Fig. 1 A possible execution time profile for DCA Scheme 1 parallel Monte Carlo Algorithms.

Fig. 2 A queuing system model for DCA Scheme 1 MIMD algorithms.

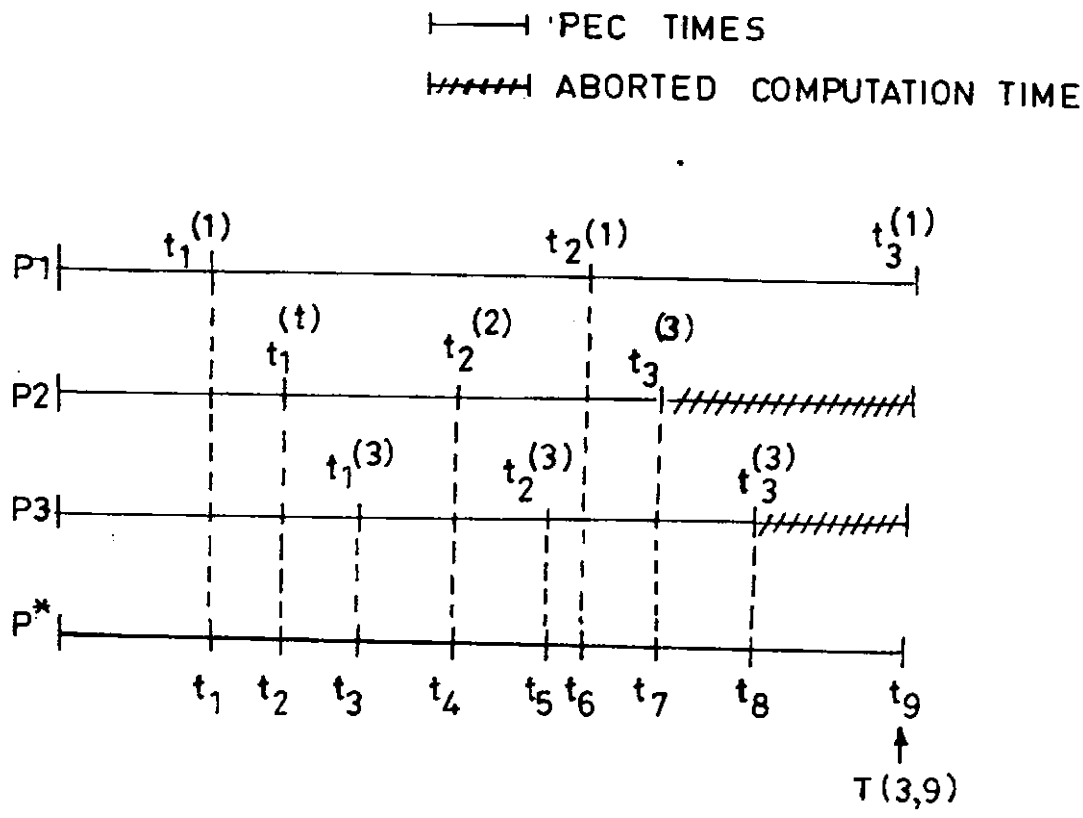
Fig. 3 A possible executino time profile for DCA Scheme 2 MIMD algorithms.

Fig. 4 The region for Laplace's equations with mesh, mesh size h=¼.

Fig. 5 Performance of various schemes for parallel Monte Carlo algorithsm for solving Laplace's equation at point (1,1) in the given region.

Fig. 6 Performance of various schemes for parallel Monte Carlo algorithms for solving Laplace's equation at point (2,3) in the given region.

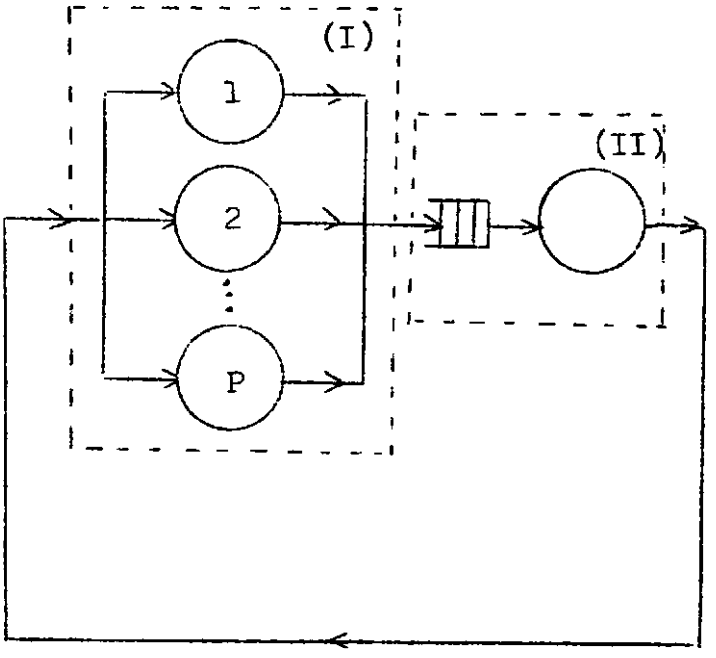
Fig. 7 Performance of various schemes for parallel Monte Carlo algorithms for solving Laplace's equation at point (3,5) in the given region.



P1, P2, P3 : PROCESSORS

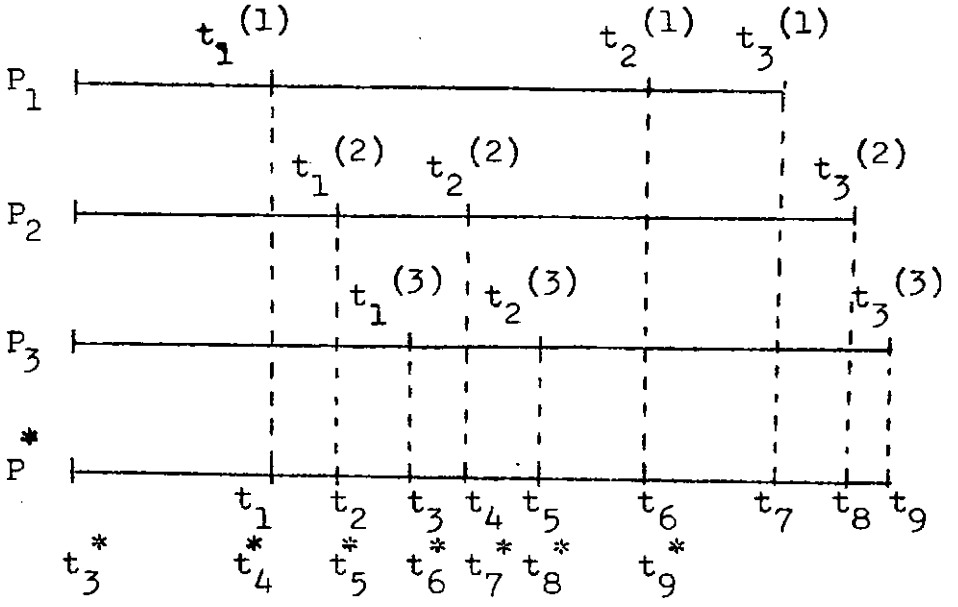
P* : VIRTUAL PROCESSOR REPRESENTING THE EXECUTION TIME BEHAVIOUR OF THE PARALLEL ALGORITHMS

Fig. 1 A possible execution time profile for DCA Scheme 1 parallel Monte Carlo Algorithms.



1. P Customers in the whole system : P processes
2. P Servers in System - I : Pecs on P processors
3. 1 Server in System-II : the critical region execution
4. At most P servers active at the same time in the entire system.

Fig. 2 A queuing system model for DCA Scheme 1 MIMD algorithms.



$$T(3,9) = t_9$$

$t_j^{(i)}$: time at which j-th pec is completed by the i-th processor

t_i : time at which i-th pec is completed in the parallel algorithm

t_i^* : time at which i-th pec is initiated in the parallel algorithm

Fig. 3 A possible execution time profile for DCA Scheme 2 MIMD algorithms.

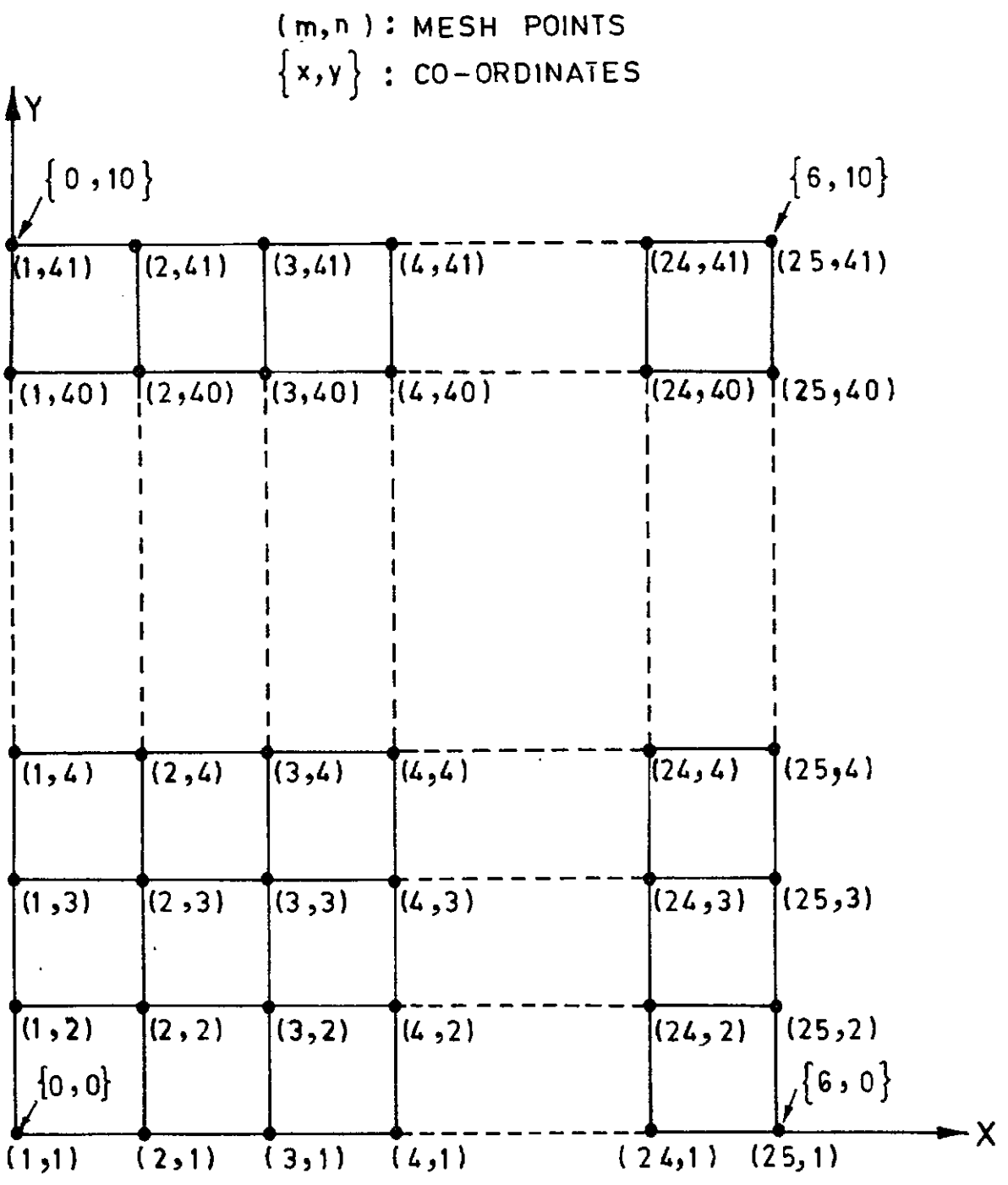


Fig. 4 The region for Laplace's equations with mesh, mesh size $h=\frac{1}{4}$.

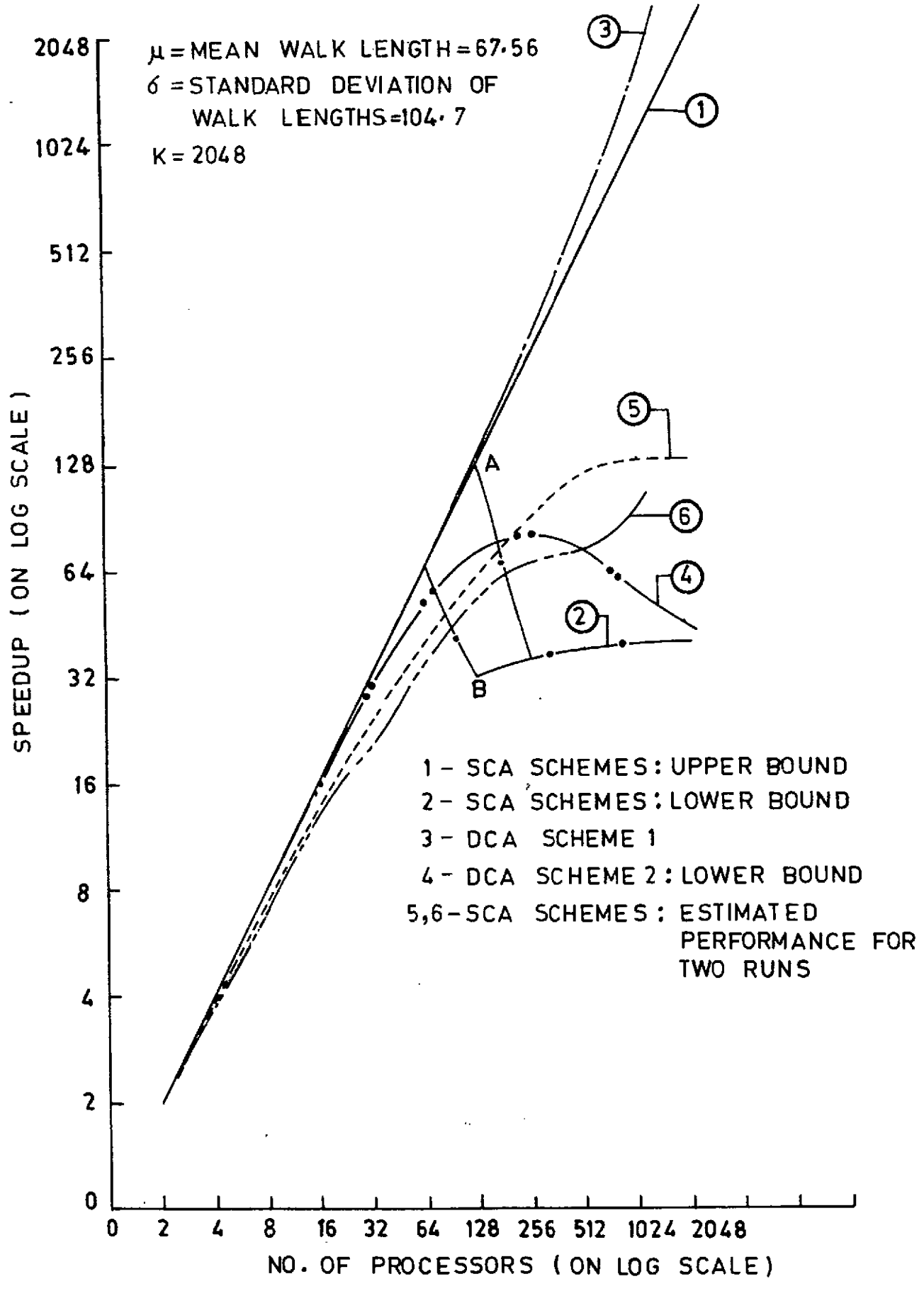


Fig. 5 Performance of various schemes for parallel Monte Carlo algorithms for solving Laplace's equation at point (1,1) in the given region.

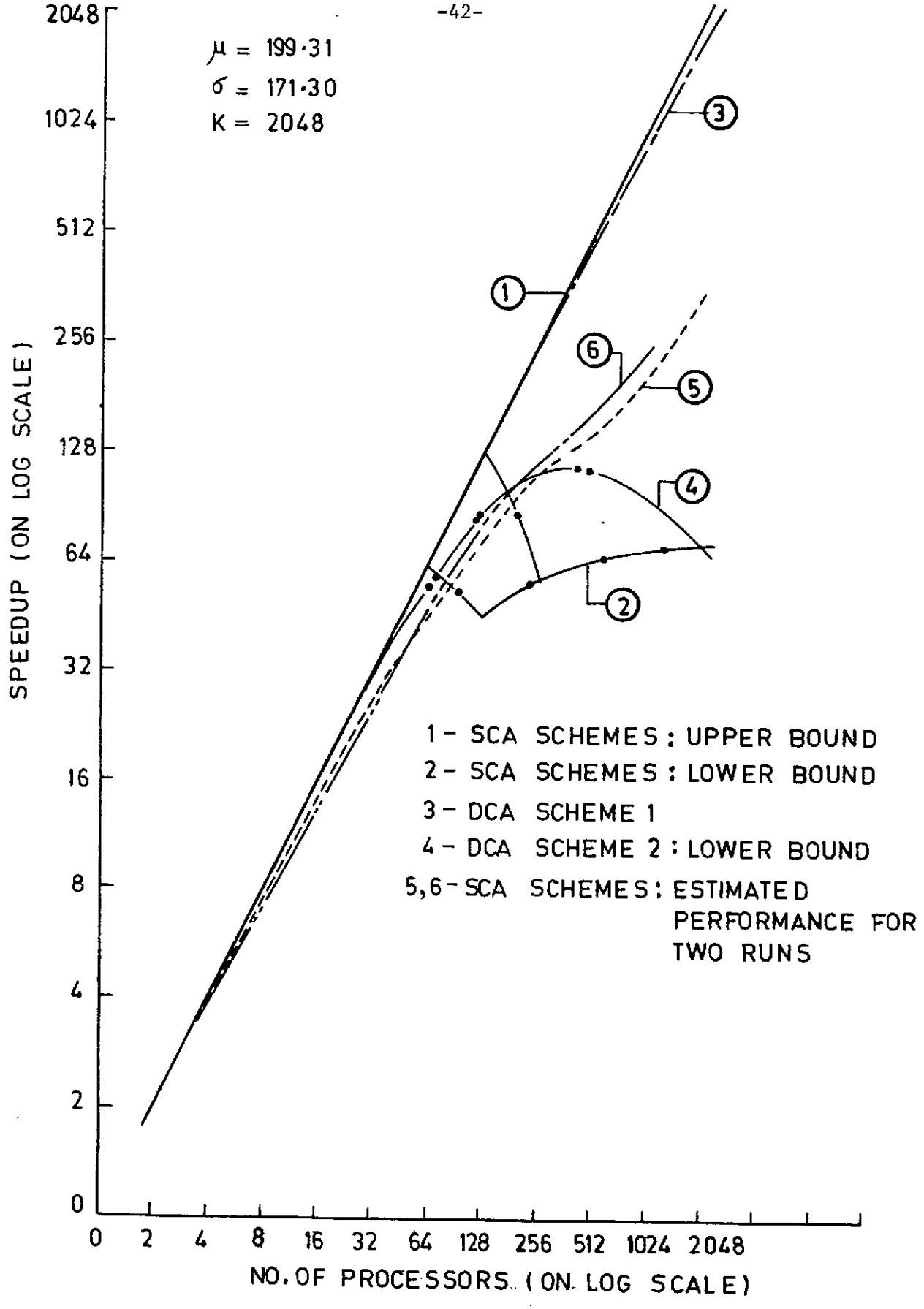


Fig. 6 Performance of various schemes for parallel Monte Carlo algorithms for solving Laplace's equation at point (2,3) in the given region.

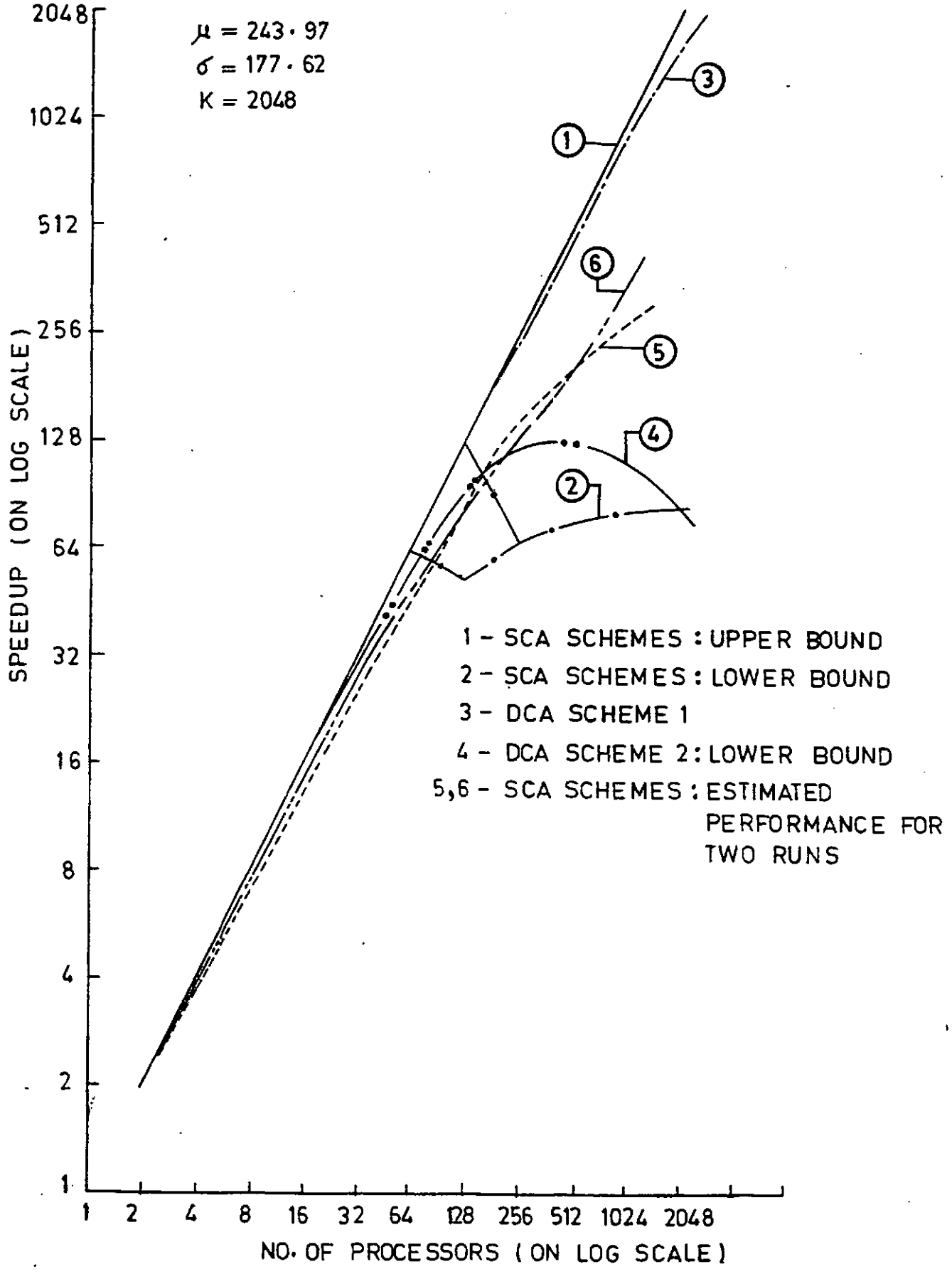


Fig. 7 Performance of various schemes for parallel Monte Carlo algorithms for solving Laplace's equation at point (3,5) in the given region.