Journal of Computational and Applied Mathematics 48 (1993) 253–277
North-Holland

CAM 1305

Monte Carlo algorithms: performance analysis for some computer architectures *

Ivan T. Dimov and Ognyan I. Tonev
Center of Informatics and Computer Technology, Bulgarian Academy of Sciences, Sofia, Bulgaria

Received 31 May 1991
Revised 28 April 1992

Abstract

Dimov, I.T. and O.I. Tonev, Monte Carlo algorithms: performance analysis for some computer architectures, Journal of Computational and Applied Mathematics 48 (1993) 253–277.

The paper deals with the performance analysis of three Monte Carlo algorithms for some models of computer architectures. To estimate the performance and the speedup of these algorithms, we introduce a special modification of the criterion for the time required to achieve a preset probable error and consider a serial (von Neumann) architecture, a pipeline architecture, and two MIMD (Multiple Instruction stream, Multiple Data stream) parallel architectures. An approach to constructing Monte Carlo vector algorithms to be efficiently run on pipeline computers has also been considered.

Keywords: Monte Carlo method; parallel algorithms; probable error estimator; performance analysis.

1. Introduction

The Monte Carlo algorithms are known to be inherently parallel. Already in the first paper [9] on the Monte Carlo method it was said that "the statistical methods can be applied by many computers working in parallel and independently".

At present, there are many different Monte Carlo algorithms for solving a wide range of problems.

To estimate the performance of the Monte Carlo algorithms on different computer architectures we have considered the following models.

(1) A serial model with time τ required to complete a suboperation (for real computers this time is usually the clock period). The important feature of this model is that each operation has to be performed sequentially and one at a time.

Correspondence to: Dr. I.T. Dimov, Center of Informatics and Computer Technology, Bulgarian Academy of Sciences, Acad. G. Bonchev str. bl. 25-A, 1113 Sofia, Bulgaria.

* Supported by the Ministry of Science, Culture and Education of Bulgaria, grant no. 831/1987.

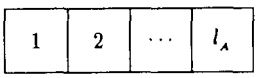

Fig. 1. The pipeline model with l_A segments.

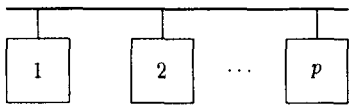

Fig. 2. The multiprocessor model with p processors.

(2) A pipeline model with startup time $s\tau$. Pipelining means an application of assembly-line techniques to improve the performance of the arithmetic operations (see Fig. 1).

(3) A multiprocessor configuration consisting of p processors. Every processor of the multiprocessor system performs its own instructions on the data in its own memory (see Fig. 2).

(4) A multiprocessor system of p processor arrays. It is a model of third type but one in which every processor is connected to a processor array of $2n-1$ processing elements. The processor array is a set of processing elements performing the same operations on the data in its own memory, i.e., SIMD (Single Instruction stream, Multiple Data stream) architecture (see Fig. 3).

Considering models instead of real computers allows for the distinction of the typical features of the architectures on the one hand, and, on the other hand, it enables general conclusions about the algorithms to be drawn.

2. General description of the Monte Carlo method

The problems one usually solves using the Monte Carlo methods can be given the following formal description. Let $f=f(x)$ and $u_i=u(x_i)$ be real-valued functions defined in the domain $G\subset\mathbb{R}^n$, and $L=L(u)$ be a linear operator defined on the space of all real-valued functions u whose arguments belong to G. It is required to calculate the sequence of functions $u_1, u_2, \ldots$, defined by the recursive formula

$$u_{i+1}=L(u_i)+f, \quad i=0, 1, 2, \ldots . \tag{1}$$

The formal solution of (1) is the truncated von Neumann series

$$u_k=f+L(f)+\cdots+L^{k-1}(f)+L^k(u_0), \quad k>0, \tag{2}$$

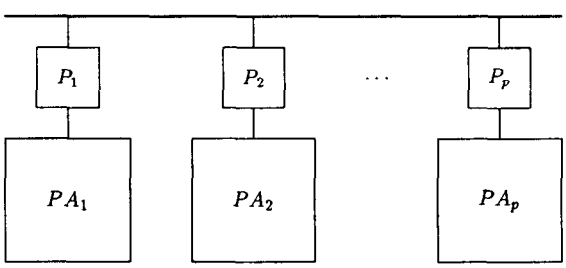

Fig. 3. The multiprocessor model consisting of p processors (P) of which everyone is connected with a processor array (PA) of $2n-1$ processing elements.

where L^k means the kth iterate of L. Special interest lies in the case in which the corresponding infinite series converges. Its sum then is a function u which satisfies the equation

$$u = L(u) + f, \tag{3}$$

and the truncation error of (2) is

$$u_k - u = L^k(u_0 - u). \tag{4}$$

The solution procedure of (1) by the Monte Carlo method will now be described in correspondingly general terms. Consider the space $\mathbb{R}^n$ over which the functions f, u_0, u_1,... are defined. Let $J(u_k)$ be a linear functional that is to be calculated. A probability distribution is now set up on each of the spaces

$$T_i = \underbrace{\mathbb{R}^n \times \mathbb{R}^n \times \cdots \times \mathbb{R}^n}_{i \text{ times}}, \quad i = 1, 2, \ldots, k, \tag{5}$$

where "$\times$" denotes the Cartesian product of spaces. Random variables θ_i, $i = 0, 1, \ldots, k$, are defined on the respective product spaces T_i and have conditional expectations

$$\mathrm{E}\theta_0 = J(u_0), \quad \mathrm{E}(\theta_1 \mid \theta_0) = J(u_1), \ldots, \mathrm{E}(\theta_k \mid \theta_0) = J(u_k). \tag{6}$$

Thus the problem is reduced to the estimation of the conditional expectation $\mathrm{E}(\theta_k \mid \theta_0)$. The computational problem then becomes one of calculating repeated realizations of θ_k and combining them into an appropriate statistical estimator of $J(u_k)$. Because of the nature of the process, every realization of θ_k is a Markov chain. So, we will next use the Markov chain as a synonym of the realization of a random variable of this type. As approximate value of the linear functional $J(u_k)$ is used

$$J(u_k) \approx \frac{1}{n} \sum_{s=1}^{N} \{\theta_k\}_s, \tag{7}$$

where $\{\theta_k\}_s$ is the sth realization of the random variable θ_k. The probable error r_N of (7) is then [5]

$$r_N = c_{0.5}\sigma(\theta)\theta_k N^{-1/2}, \tag{8}$$

where $c_{0.5} \approx 0.6745$ and $\sigma(\theta)\theta_k$ is the standard deviation.

Suppose there are different Monte Carlo algorithms to solve a problem. It follows from (8) that the computational effort for the achievement of a preset probable error is proportional to $t\sigma^2(\theta)\theta$, where t is the expectation of the time required to calculate one realization of the random variable θ. The product $t\sigma^2(\theta)\theta$ is called the *labour consumption coefficient,* whereas $[t\sigma^2(\theta)\theta]^{-1}$ is a measure for the efficiency. In these terms the optimum Monte Carlo algorithm is the one with the lowest labour consumption coefficient.

We shall consider the theory for two special cases of the operator L.

(i) L is an ordinary integral transform

$$L(u) = \int_G k(x, y)u(y)\,\mathrm{d}y. \tag{9}$$

(ii) L is a matrix and u is a vector.

Let us consider the problems for the first case. They consist in calculating

$$J(u) = (g, u) = \int_G g(x)u(x)\,\mathrm{d}x, \tag{10}$$

where $G \in \mathbb{R}^n$, and $g(x) \in L_2(G)$ is an arbitrary function. In this case, (3) becomes

$$u(x) = \int_G k(x, y)u(x)\,\mathrm{d}y + f(x), \tag{11}$$

and the random variable whose mathematical expectation coincides with $J(u)$ is

$$\theta[g] = \frac{g(\xi_0)}{p(\xi_0)} \sum_{j=0}^{\infty} Q_j f(\xi_j), \tag{12}$$

where

$$Q_0 = 1, \qquad Q_j = Q_{j-1}\frac{k(\xi_{j-1}, \xi_j)}{p(\xi_{j-1}, \xi_j)}, \quad j = 1, 2, \ldots,$$

and $\xi_0, \xi_1, \ldots$ is a Markov chain in G with initial density function $p(x)$, and transition densities $p(x, y)$, which are admissible [1] for $g(x)$ and $k(x, y)$, respectively.

This formulation includes many boundary problems as well. Let us consider a typical problem of the quasi-electrodynamics:

$$\Delta u(x) - cu(x) = -f(x), \quad x \in \boldsymbol{G}, \tag{13a}$$

$$u(x) = \psi(x), \quad x \in \partial\boldsymbol{G}. \tag{13b}$$

From the theory of fundamental solutions there follows that the solution of (13) can be represented [5] as the integral equation (11) where

$$k(x, y) = \begin{cases} \dfrac{d\sqrt{c}}{\sinh(d\sqrt{c})}\delta(y - x), & \text{when } x \notin \partial\boldsymbol{G}, \\ 0, & \text{when } x \in \partial\boldsymbol{G}, \end{cases} \tag{14}$$

$$f(x) = \begin{cases} \dfrac{1}{4\pi}\displaystyle\int \frac{\sinh\big((d - |y - x|)\sqrt{c}\big)}{|y - x| \sinh(d\sqrt{c})}\varphi(y)\,\mathrm{d}y, & \text{when } x \notin \partial\boldsymbol{G}, \\ \psi(x), & \text{when } x \in \partial\boldsymbol{G}, \end{cases} \tag{15}$$

and $d = d(x)$ is the distance from x to the boundary $\partial\boldsymbol{G}$. The Monte Carlo procedure for solving this problem is known as the "spherical process". To ensure the convergence of the process, we introduce the ϵ-strip $\partial\boldsymbol{G}_\epsilon = \{x \in \boldsymbol{G}: d(x) < \epsilon\}$ of the boundary, and assume that $u(x)$ is known for all points in $\partial\boldsymbol{G}_\epsilon$. Then,

$$\iint k(x, y)k(y, z)\,\mathrm{d}y\,\mathrm{d}z \leqslant \int \delta(y - x)\int \delta(z - y)\,\mathrm{d}z\,\mathrm{d}y = \int \delta(y - x)\,\mathrm{d}y$$

$$\leqslant 1 - \frac{\epsilon^2}{4d_{\sup}^2}, \tag{16}$$

[1] $p(x)$ is admissible for $f(x)$ if $p(x) > 0$ when $f(x) \geqslant 0$.

where $d_{\sup}$ is the supremum of all radii of the spheres lying in G. Inequality (16) ensures the convergence of the von Neumann series and therefore of (12), too.

In the second interpretation, when L is a matrix, the recurrence (1) takes the form

$$\boldsymbol{u}_{k+1} = \boldsymbol{K}\boldsymbol{u}_k + \boldsymbol{f}, \tag{17}$$

where $\boldsymbol{u}_k$, $\boldsymbol{u}_{k+1}$ and $\boldsymbol{f}$ are vectors and $\boldsymbol{K}$ is a matrix.

Knowing $\boldsymbol{K} = (k_{\alpha\beta})_{\alpha,\beta=1}^{m}$, and $u_0 = (u_1^0, \ldots, u_m^0)$, we can find the solution of (17):

$$\boldsymbol{u}_k = \boldsymbol{K}^k\boldsymbol{u}_0 + \boldsymbol{K}^{k-1}\boldsymbol{f} + \cdots + \boldsymbol{K}\boldsymbol{f} + \boldsymbol{f} = (\boldsymbol{I} - \boldsymbol{K}^k)(\boldsymbol{I} - \boldsymbol{K})^{-1}\boldsymbol{f} + \boldsymbol{K}^k\boldsymbol{u}_0, \tag{18}$$

where $\boldsymbol{I}$ is the unit matrix and the matrix $\boldsymbol{I} - \boldsymbol{K}$ is supposed to be nonsingular.

It is well known that if and only if all eigenvalues of $\boldsymbol{K}$ lie within the unit circle of the complex plane, there exists a vector $\boldsymbol{u}$ such that

$$\boldsymbol{u} = \lim_{k \to \infty} \boldsymbol{u}_k, \tag{19}$$

which satisfies the equation

$$\boldsymbol{u} = \boldsymbol{K}\boldsymbol{u} + \boldsymbol{f}. \tag{20}$$

If the Neumann series for $\boldsymbol{u}$ does not converge, there exist some other procedures (e.g., extension of the resolvent, isolating the poles of the resolvent and so on [10]) to construct a convergent Monte Carlo method. The same can be used also when the series is convergent but its rate of convergence is slow.

As an illustration let us consider the problem of evaluating the inner product $J(\boldsymbol{u}) = (\boldsymbol{g}, \boldsymbol{u})$, where $\boldsymbol{g} \in \mathbb{R}^m$ is a given vector and $\boldsymbol{u}$ is the solution of the system of linear equations

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b}, \tag{21}$$

with $\boldsymbol{b} = (b_1, \ldots, b_m)$.

We can choose a nonsingular matrix $\boldsymbol{M}$ such that

$$\boldsymbol{M}\boldsymbol{A} = \boldsymbol{I} - \boldsymbol{K} \tag{22}$$

and

$$\boldsymbol{M}\boldsymbol{b} = \boldsymbol{f}, \tag{23}$$

and then (21) becomes

$$\boldsymbol{M}\boldsymbol{A}\boldsymbol{u} = \boldsymbol{M}\boldsymbol{b}, \tag{24}$$

which is equivalent to (20). With $\boldsymbol{M}$ and $\boldsymbol{A}$ both nonsingular, and $\boldsymbol{K}$ having its eigenvalues all inside the unit circle, (17) becomes a stationary linear iterative process [2].

We will next construct a Monte Carlo algorithm to solve the above problem, i.e., to calculate

$$J(u) = \sum_{\alpha=1}^{m} g_\alpha u_\alpha. \tag{25}$$

To do that we will use a method similar to those used for solving the first kind problem (i.e., the continuous case) [13].

Let us consider the integral equation

$$u(x) = \int_G k(x, y)u(y)\,\mathrm{d}y + f(x), \tag{26}$$

for which $G = [0, m)$ is a one-dimensional segment divided into equal subsegments $G_\alpha = [\alpha - 1, \alpha)$, $\alpha = 1, 2, \ldots, m$, such that

$$\begin{cases} k(x, y) = k_{\alpha\beta}, & x \in G_\alpha,\ y \in G_\beta, \\ f(x) = f_\alpha, & x \in G_\alpha. \end{cases} \tag{27}$$

Then (26) in G_α becomes

$$u(\alpha) = \sum_\beta \int_{G_\beta} k_{\alpha\beta}u(y)\,\mathrm{d}y + f_\alpha, \tag{28}$$

and if we denote

$$u_\beta = \int_{G_\beta} u(y)\,\mathrm{d}y, \tag{29}$$

we obtain (in G_α)

$$u(x) = \sum_\beta k_{\alpha\beta}u_\beta + f_\alpha. \tag{30}$$

From (30) there follows that $u(x) = u_\alpha$, and so it becomes

$$u_\alpha = \sum_\beta k_{\alpha\beta}u_\beta + f_\alpha, \tag{31}$$

or in matrix form

$$\boldsymbol{u} = \boldsymbol{K}\boldsymbol{u} + \boldsymbol{f}. \tag{32}$$

Then similarly to (12) we construct the random variable

$$\theta[g] = \frac{g_{l_0}}{p_0} \sum_{\nu=0}^{\infty} Q_\nu f_{l_\nu}, \tag{33}$$

where

$$Q_0 = 1, \qquad Q_j = Q_{j-1}\frac{k(\xi_{j-1}, \xi_j)}{p(\xi_{j-1}, \xi_j)}, \quad j = 1, 2, \ldots,$$

and $l_0, l_1, \ldots$ is a Markov chain constructed according to the probability densities which are admissible for $\boldsymbol{g}$ and $\boldsymbol{K} = (k_{\alpha\beta})$, respectively.

If the rth component of the vector $\boldsymbol{u} = (u_1, \ldots, u_m)$ is to be calculated, we can choose the vector $\boldsymbol{g} = e^{(r)}$ with the rth component unity, the other components all zero.

The inherent parallelism of the Monte Carlo methods lies in the possibility of calculating each realization of the random variable θ on a different processor (or computer). There is no need for communication between the processors during the time of calculating the realizations — the only need for communication is at the end when the averaged value is to be calculated.

To estimate the performance of the Monte Carlo algorithms, we need modifications of the usual criterion [12]. The modification consists in replacement of $T_p(\mathscr{A})$ (which is the time

required for a set of p processing elements to solve the problem using an algorithm $\mathscr{A}$) with the mathematical expectation $ET_p(\mathscr{A})$ of this time. That is so because of the nature of the Monte Carlo method.

The criteria for speedup and paralleling efficiency [2] of an algorithm $\mathscr{A}$ will be as follows:

$$S_p(\mathscr{A}) = \frac{ET_1(\mathscr{A})}{ET_p(\mathscr{A})} \tag{34}$$

and

$$E_p(\mathscr{A}) = \frac{S_p(\mathscr{A})}{p}. \tag{35}$$

The classical problem

$$\Delta u = -f(x), \quad x \in \boldsymbol{G}, \tag{36a}$$

$$u(x) = \psi(x), \quad x \in \partial G, \tag{36b}$$

where $x = (x_1, \ldots, x_n) \in \overline{G} \subset K_n = \{0 \leqslant x \leqslant 1,\ i = 1, 2, \ldots, n\}$ is considered.

Our aim is to calculate the linear functional

$$J(u) = (g, u) = \int_G g(x)u(x)\,\mathrm{d}x. \tag{37}$$

We will next consider three Monte Carlo algorithms for the calculation of this product.

3. Algorithm $\mathscr{A}$

Using a regular discretization with step h, (36a) is approximated by the difference equation

$$\Delta_h^{(n)} u = -f_h, \tag{38a}$$

or solved for the ith point $i = (i_1, \ldots, i_n)$:

$$u_i = L_h u + \frac{h^2}{2n} f_i, \tag{38b}$$

where $\Delta_h^{(n)}$ is the Laplace difference operator, and L_h is an averaging operator. For example, the operator L_h in $\mathbb{R}^2$ is

$$L_h u = \tfrac{1}{4}\left[u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}\right] = \tfrac{1}{4}\Lambda_1(i, j),$$

and then (38b) becomes

$$u_{i,j} = \tfrac{1}{4}\Lambda_1(i, j) + \tfrac{1}{4}h^2 f_{i,j}. \tag{39}$$

The approximation error of (38) is

$$|u_i - u(x_i)| = O(h^2). \tag{40}$$

We need $h = \sqrt{\epsilon}$ to ensure consistency of the approximate error and the probability error.

[2] The paralleling efficiency is the measure of the price to be paid for the speedup achieved.

The matrix form of equations (38) is of the type of (32) but the matrix there has only $2n$ nonzero elements in every row and they all are equal to $1/(2n)$.

The Monte Carlo algorithm for solving (37) consists in simulating a Markov chain with initial density p_0, which is admissible to the vector $\boldsymbol{g}$, and probability $p_{\alpha\beta}$ for transition from the point α to the next point β equal to $1/(2n)$ if the point is inner, and equal to 0 for boundary points (the boundary is the absorbing barrier for the process). The random variable whose mathematical expectation coincides with the solution of the problem is

$$\theta = \frac{h^2}{2n} \sum_{i=1}^{i^*-1} f_i + \psi_{i^*}, \tag{41}$$

where f_i are the values of the function f in the points of the Markov chain, and i^* is the point where the Markov chain reaches the boundary $\partial \boldsymbol{G}_h$.

Every transition in the Markov chain is done following the algorithm

(i) generation of a random number (it is usually done in k arithmetic operations where $k = 2$ or 3);

(ii) determination of the next point which includes a random number of logical operations [3] with expectation equal to n, and maximum $2n$ (n is the space dimension) and one algebraic operation to calculate the coordinates of the point.

Let $m_i(h, n)$ be the mean number of steps required to reach the boundary for a process (or a chain) starting at the point i and $m(h, n)$ is the vector with coordinates m. Then $m(h, n)$ is the solution of the difference problem

$$\begin{cases} m_i = L_h m + \varphi_i, & i \in G_h, \ \varphi_i = \dfrac{2n}{h^2}, \\ m_i = 0, & i \in \partial \boldsymbol{G}_h. \end{cases} \tag{42}$$

If $\{m_i'(h, n)\}$ is the solution of problem (42) in the unit cube K_n, then the following inequality holds:

$$m_i'(h, n) \geqslant m_i(h, n), \quad i = 1, 2, \ldots . \tag{43}$$

The difference problem for $m_i'(h, n)$ could be approximated by the problem

$$\begin{cases} \Delta m'(x) = -\dfrac{2n}{h^2}, & x \in K_n, \\ m'(x) = \dfrac{1}{h^2} \sum\limits_{l=1}^{n} \left(x_l - x_l^2\right), & x \in \partial K_n, \end{cases} \tag{44}$$

where the error is of order O(1).

Let M be the maximum of $m'(x)$, i.e.,

$$M = \max_{K_n} m'(x) = \frac{1}{4h^2} = \frac{1}{4\epsilon}. \tag{45}$$

This value is independent of the problem dimension and could be reached when $x_l = \frac{1}{2}$, $l = 1, 2, \ldots, n$.

[3] The logical operation means testing the inequality "$\alpha < b$".

The result in (45) is natural because the inverse operator of $\Delta_h^{(n)}$ is uniformly bounded over h, and so it follows that

$$m_i(h, n) = \frac{c(n)}{h^2} + O\left(\frac{1}{h^2}\right). \tag{46}$$

Thus, M is an upper bound for the expectation of the number of steps in a Markov chain, which is a realization of the random variable θ. To achieve a probable error ϵ, it is necessary to average N realizations of the random variable θ where

$$N = c_{0.5}^2 \frac{\sigma^2(\theta)\theta}{\epsilon^2}. \tag{47}$$

The expectation of the number of all transitions R in the Markov chain will then be

$$R \leqslant MN = \tfrac{1}{4} c_{0.5}^2 \frac{1}{\epsilon^3} \sigma^2(\theta)\theta = t\sigma^2(\theta)\theta. \tag{48}$$

Let l_A and l_L be the number of suboperations of the arithmetic and logical operations, respectively. Then the expectation of the time required to achieve a probable error on the serial model using the algorithm $\mathscr{A}$ is

$$ET_1(\mathscr{A}) = \tau[(k + 1 + \gamma_A)l_A + (n + 1 + \gamma_L)l_L]\left(\tfrac{1}{2}(c_{0.5}\sigma(\theta)\theta)\right)^2 \frac{1}{\epsilon^3}, \tag{49}$$

where γ_A and γ_L are the number of arithmetic and logical operations to calculate the distance from a given point to the boundary ∂G. Here we assume that $f(x) = 0$ in (36a), because the number of operations to calculate a value of $f(x)$ in an arbitrary point x varies for different functions f.

In the case of the third model we suppose that it is possible to choose the number of processors to be

$$p = \left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2. \tag{50}$$

Then

$$ET_p(\mathscr{A}) = \tau[(k + 1 + \gamma_A)l_A + (n + 1 + \gamma_L)l_L]\frac{1}{4\epsilon}, \tag{51}$$

$$S_p(\mathscr{A}) = p \tag{52}$$

and

$$E_p(\mathscr{A}) = 1, \tag{53}$$

i.e., the paralleling efficiency has its maximum value in terms of the modified criterion.

The speedup can be improved using the fourth model. In this case all $2n$ logical operations at each step of the Markov chain are done simultaneously, and so

$$ET_{2np}(\mathscr{A}) = \frac{\tau}{4\epsilon}[(k + 1 + \gamma_A)l_A + 3l_L], \tag{54}$$

$$S_{2np}(\mathscr{A}) = p\left(1 + \frac{n + 1 + \gamma_L}{k + 1 + \gamma_A}\frac{l_L}{l_A}\right)\Big/\left(1 + \frac{3}{k + 1 + \gamma_A}\frac{l_L}{l_A}\right), \tag{55}$$

and $S_{2np}(\mathscr{A}) > S_p(\mathscr{A})$ when $n + 1 + \gamma_L \sim 3$.

The paralleling efficiency is

$$E_{2np}(\mathscr{A}) = \frac{1}{2n}\left(1 + \frac{n+1+\gamma_L}{k+1+\gamma_A}\frac{l_L}{l_A}\right)\Bigg/\left(1 + \frac{3}{k+1+\gamma_A}\frac{l_L}{l_A}\right). \tag{56}$$

Assuming the pipeline model [6,7] to have l_A segments, we obtain the following results:

$$ET_{pipe}(\mathscr{A}) = \tau[s+k+l_A+\gamma_A+(n+1+\gamma_L)l_L]\left(\tfrac{1}{2}(c_{0.5}\sigma(\theta)\theta)\right)^2\frac{1}{\epsilon^3}, \tag{57}$$

$$S_{pipe}(\mathscr{A}) = \left(1 + \frac{k+1+\gamma_A}{n+1+\gamma_L}\frac{l_A}{l_L}\right)\Bigg/\left(1 + \frac{s+k+l_A+\gamma_A}{n+1+\gamma_L}\frac{1}{l_L}\right). \tag{58}$$

From (55) follows that $S_{pipe}(\mathscr{A}) > 1$ when s, l_A and γ_A meet the inequality

$$s < (l_A - 1)(k + \gamma_A), \tag{59}$$

which is not a real restriction, but it becomes obvious that algorithms like $\mathscr{A}$ are not well suited for the pipeline model. This follows from the formula for $S_{pipe}(\mathscr{A})$ which does not have a factor $1/\epsilon$, whereas $S_p(\mathscr{A}) = O(\epsilon^{-2})$.

Special Monte Carlo algorithms can be constructed for the pipeline computers which are known as vector algorithms [8]. Such an algorithm will be considered later.

4. Algorithm $\mathscr{B}$

Let us consider again the problem of calculating the inner product (g, u) with a preset probable error ϵ.

The discretization formula for the Laplace equation

$$U_{i,j,k} = \tfrac{1}{6}\Lambda_{i,j,k}, \tag{60}$$

stated in Appendix A, leads to the random variable

$$\theta = \psi(\xi), \tag{61}$$

where ξ is a random point on the boundary ∂G_h. The expectation of this random variable coincides with the required value.

The Monte Carlo algorithm for estimating the expectation of θ consists in calculating repeated realizations of θ which are Markov chains with initial density p_0 admissible to g, and transition probabilities

$$p_{\alpha\beta} = \begin{cases} \frac{1}{6}, & \text{when } \alpha \in G_h, \\ \delta_{\alpha\beta}, & \text{when } \alpha \in \partial G_h, \end{cases}$$

where

$$\delta_{\alpha\beta} = \begin{cases} 1, & \alpha = \beta, \\ 0, & \alpha \neq \beta, \end{cases}$$

is the Kronecker notation. Next, we consider the problem of estimating the number of operations R required to reach the probable error ϵ.

From (48) it follows that the estimators for the number of realizations N and the number of transitions M in one Markov chain have to be obtained.

The number of realizations N is independent of the algorithm used, and is given by

$$N = \left(c_{0.5} \frac{\sigma(\theta)}{\epsilon} \right)^2 . \tag{62}$$

Let us find an upper bound for the number of steps M in a Markov chain. To do that we consider the following problem. Let D be a plane in $\mathbb{R}^3$ and Ω be the half-space bounded by D. The domain Ω is replaced by a set of points ω_h using a regular discretization. The Markov chains in the algorithm $\mathscr{B}$ are random walks in ω_h that begin at a point chosen according to the initial density p_0; the next point is selected according to the following stipulations.

(i) About the inner point α determine the maximum approximation molecule [4] $\mu(\alpha)$.

(ii) With probability $p_{\alpha\beta} = \frac{1}{6}$, select the point β uniformly at random on $\mu(\alpha)$.

(iii) If the point β is on the boundary, the Markov chain is terminated and the process begins again at a point chosen correspondingly to p_0. If the point β is an inner point of ω_h, then step (ii) is repeated using β in place of α.

Our aim is to estimate the number of inner points passed by until a boundary point is reached.

Let T be a random variable that is the number of steps in a Markov chain, and T_ν be the number of steps in the νth Markov chain [1]. Then

$$P\{T_\nu = k\} = \left(\tfrac{5}{6}\right)^{k-1} \tfrac{1}{6}, \tag{63}$$

and the expectation of the random variable T is given by

$$\mathrm{E}T = \sum_{k=1}^{\infty} k \left(\tfrac{5}{6}\right)^{k-1} \tfrac{1}{6} = 6. \tag{64}$$

This result gives an upper bound for the number of steps in a Markov chain, i.e.,

$$M \leqslant 6; \tag{65}$$

so in this case

$$R < 6 \left(c_{0.5} \frac{\sigma(\theta)}{\epsilon} \right)^2 . \tag{66}$$

For one transition in the Markov chain we have to do the following operations:

- γ_A arithmetic and 5 logical operations to find (calculate) the distance from the point to the boundary ∂G and 1 logical operation to verify if this distance is nonzero, i.e., the point is not on the boundary;
- k arithmetic operations for the generation of a random number;
- 3 logical and 1 arithmetic operations for calculating the coordinates of the next point.

So, for a serial model it is obtained that

$$\mathrm{E}T_1(\mathscr{B}) = 6\tau[(k + 1 + \gamma_A)l_A + 9l_L] \left(c_{0.5} \frac{\sigma(\theta)}{\epsilon} \right)^2 , \tag{67}$$

[4] The approximation molecule $\mu(\alpha)$ for the differential operator is the set of mesh points used for approximation of the derivatives in the operator at the point α.

and for the third model with $p = (c_{0.4}\sigma(\theta)/\epsilon)^2$ processors the expectation is

$$\mathrm{E}T_p(\mathscr{B}) = 6\tau[(k+1+\gamma_A)l_A + 9l_L]. \tag{68}$$

Then the coefficients for the speedup and the performance are

$$S_p(\mathscr{B}) = p \tag{69}$$

and

$$E_p(\mathscr{B}) = \frac{S_p(\mathscr{B})}{p} = 1, \tag{70}$$

which means that one could achieve the optimum performance.

The calculation of repeated realizations of a random variable do not need any communications between the processors, so the mathematical expectation of the idle time of the processors is zero.

The value of $S_p(\mathscr{A})$ could be improved with the fourth model. In this case all logical operations for determining the next point in the Markov chain and these for finding the distance to the boundary could be fulfilled at two clock periods, and so

$$\mathrm{E}T_{6p}(\mathscr{B}) = 6\tau((k+1+\gamma_A)l_A + 5l_L) \tag{71}$$

and

$$S_{6p}(\mathscr{B}) = \frac{(k+1+\gamma_A)l_A + 9l_L}{(k+1+\gamma_A)l_A + 5l_L}\left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2$$

$$= p\left(1 + \frac{9}{k+1+\gamma_A}\frac{l_L}{l_A}\right)\Big/\left(1 + \frac{5}{k+1+\gamma_A}\frac{l_L}{l_A}\right) > S_p(\mathscr{B}). \tag{72}$$

For the performance coefficient we obtain

$$E_{6p}(\mathscr{B}) = \frac{1}{6}\left(1 + \frac{9}{k+1+\gamma_A}\frac{l_L}{l_A}\right)\Big/\left(1 + \frac{5}{k+1+\gamma_A}\frac{l_L}{l_A}\right) < 1. \tag{73}$$

It is obvious that increasing the number of processing elements will involve increasing the idle time, i.e., efficiency will decrease.

Let us now consider the pipeline model with l_A segments for performing the arithmetic operations. In this case the coefficients of interest become

$$\mathrm{E}T_{\mathrm{pipe}}(\mathscr{B}) = 6\tau(s+k+l_A+\gamma_A+9l_L)\left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2, \tag{74}$$

$$S_{\mathrm{pipe}}(\mathscr{B}) = \left(1 + \tfrac{1}{9}(k+1+\gamma_A)\frac{l_A}{l_L}\right)\Big/\left(1 + \tfrac{1}{9}(s+k+l_A+\gamma_A)\frac{1}{l_L}\right), \tag{75}$$

and the sufficient condition for $S_{\mathrm{pipe}}(\mathscr{B}) > 1$ is

$$s < (l_A - 1)(k+\gamma_A), \tag{76}$$

which holds for all real computers.

As in the case of the algorithm $\mathscr{A}$, one can see that this algorithm is not the best one for pipeline computers.

5. Algorithm $\mathscr{E}$

There exists a different approach to the solution of problem (36) which uses the integral representation [14]

$$u(x) = \frac{1}{4\pi} \int \left[\frac{1}{|x-y|} \frac{\partial u(y)}{\partial n_y} - u(y) \frac{\partial}{\partial n_y} \frac{1}{|x-y|} \right] \mathrm{d}s_y, \tag{77}$$

for $f(x) \equiv 0$ and $n = 3$.

Representation (77) allows us to construct a Monte Carlo algorithm, called *spherical process* for computing the inner product $(g(x), u(x))$. The algorithm is as follows.

(i) We choose an ϵ-strip ∂G_ϵ of the boundary ∂G and suppose that the solution of problem (36) is known in ∂G_ϵ.

(ii) The starting point x_0 is to be chosen with respect to the initial density $p(x)$ which is admissible for the function $g(x)$.

(iii) The next point is chosen to be equidistributed on the maximum sphere in G with the center in x_0. The formula for computing x_n from x_{n-1} is

$$x_n = x_{n-1} + \omega_n d(x_{n-1}), \quad n = 1, 2, \ldots, \tag{78}$$

where ω is a unit vector uniformly distributed on the unit sphere. Then, if $x_n \in G \setminus \partial G$, the process continues by determining the next point from (78), whereas if $x_n \in \partial G$, the process is terminated with setting up $\theta_n = u(x_n)$.

The value θ is the nth realization of the random variable θ. Then, using N such realizations, we construct the approximation of the solution

$$(g, u) \approx \frac{1}{N} \sum_{n=1}^{N} \theta_n. \tag{79}$$

The number of steps M in a realization of the random variable θ is

$$M = c\,|\ln \epsilon\,|, \tag{80}$$

where c depends on the boundary ∂G [5,11].

To ensure a probable error ϵ, it is necessary to do N realizations where N is chosen from (62). Then for the expectation of the number of steps we obtain

$$R = c\left(c_{0.5}\sigma(\theta)\theta\right)^2 \frac{\ln \epsilon}{\epsilon^2}, \tag{81}$$

and each step consists of γ_A arithmetic and γ_L logical operations for finding $d(x)$, $2k$ arithmetic operations for generating two pseudo-random numbers (when $n = 3$), q_A arithmetic operations for calculation of the coordinates of the next point and 1 logical operation to determine whether $x \in \partial G$.

It then follows that

$$\mathrm{E}T_1(\mathscr{C}) = \tau\left[(2k+\gamma_\mathrm{A}+q_\mathrm{A})l_\mathrm{A} + (\gamma_\mathrm{L}+1)l_\mathrm{A}\right]c\left(c_{0.5}\sigma(\theta)\theta\right)^2\frac{\ln\epsilon}{\epsilon^2}, \tag{82}$$

whereas for the third model with $p = (c_{0.5}\sigma(\theta)/\epsilon)^2$ processors it follows that

$$\mathrm{E}T_p(\mathscr{C}) = \tau\left[(2k+\gamma_\mathrm{A}+q_\mathrm{A})l_\mathrm{A} + (\gamma_\mathrm{L}+1)l_\mathrm{L}\right]c\ \ln\epsilon, \tag{83}$$

and so for the speedup and paralleling efficiency we obtain

$$S_p(\mathscr{C}) = p = \mathrm{O}(\epsilon^{-2}) \tag{84}$$

and

$$E_p(\mathscr{C}) = 1. \tag{85}$$

If the fourth model is considered, a greater speedup can be achieved:

$$S_{6p}(\mathscr{C}) = p\left(1+\frac{2k+\gamma_\mathrm{A}+q_\mathrm{A}}{\gamma_\mathrm{L}+1}\frac{l_\mathrm{A}}{l_\mathrm{L}}\right)\Bigg/\left(1+\frac{k+\gamma_\mathrm{A}+q_\mathrm{A}}{\gamma_\mathrm{L}+1}\frac{l_\mathrm{A}}{l_\mathrm{L}}\right) > S_p(\mathscr{C}), \tag{86}$$

but the paralleling efficiency decreases:

$$E_{6p}(\mathscr{A}) = \frac{1}{6}\left(1+\frac{2k+\gamma_\mathrm{A}+q_\mathrm{A}}{\gamma_\mathrm{L}+1}\frac{l_\mathrm{A}}{l_\mathrm{L}}\right)\Bigg/\left(1+\frac{k+\gamma_\mathrm{A}+q_\mathrm{A}}{\gamma_\mathrm{L}+1}\frac{l_\mathrm{A}}{l_\mathrm{L}}\right) < 1. \tag{87}$$

In case of the pipeline model with l_A segments, the respective results are

$$\mathrm{E}T_{\mathrm{pipe}}(\mathscr{C}) = \tau\left(s+2k+\gamma_\mathrm{A}+q_\mathrm{A}+l_\mathrm{A}-1+(\gamma_\mathrm{L}+1)l_\mathrm{L}\right)c\left(c_{0.5}\sigma(\theta)\theta\right)^2\frac{\ln\epsilon}{\epsilon^2} \tag{88}$$

and

$$S_{\mathrm{pipe}}(\mathscr{C}) = \left(1+\frac{2k+\gamma_\mathrm{A}+q_\mathrm{A}}{\gamma_\mathrm{L}+1}\frac{l_\mathrm{A}}{l_\mathrm{L}}\right)\Bigg/\left(1+\frac{s+2k+\gamma_\mathrm{A}+q_\mathrm{A}+l_\mathrm{A}-1}{\gamma_\mathrm{L}+1}\frac{1}{l_\mathrm{L}}\right), \tag{89}$$

and $S_{\mathrm{pipe}}(\mathscr{C}) > 1$ when the inequality

$$s < (2k+\gamma_\mathrm{A}+q_\mathrm{A}-1)(l_\mathrm{A}-1) \tag{90}$$

holds. But it is met for the real computers where a speedup greater than one could be obtained.

In Appendix B all results concerning algorithms $\mathscr{A}$, $\mathscr{B}$ and $\mathscr{C}$ in the case $n = 3$, $f(x) \equiv 0$ and $p = (c_{0.5}\sigma(\theta)/\epsilon)^2$ are enumerated.

There one can see that the rates of increasing the time required to achieve a preset probable error ϵ using algorithms $\mathscr{A}$, $\mathscr{B}$ and $\mathscr{C}$ are $\mathrm{O}(1/\epsilon)$, $\mathrm{O}(1)$ and $\mathrm{O}(\ln\epsilon)$, respectively. Thus, algorithm $\mathscr{B}$ is faster than algorithms $\mathscr{A}$ and $\mathscr{C}$.

On the other hand, it is obvious that the speedup and the paralleling efficiency coefficients are greater for MIMD architectures (models (3) and (4)) than those for the pipeline architecture (model (2)). That is so, because the increase of the number of processing elements in models (3) and (4) involves an increase of the speedup with the same factor, while the paralleling efficiency remains constant. The formulae in Appendix B show that the third model

is the best one for such algorithms, because their paralleling efficiency is unity, i.e., the idle time of its processing elements is zero.

The Monte Carlo algorithms described include a wide range of problems with probable error of type

$$r_N = cN^{-1/2-\epsilon(n)},$$

where $\epsilon(n) > 0$ is a nonnegative function of the dimensionality of the problem. Such methods can be found in, e.g., [4,13].

The Dupach algorithm is constructed for calculation of integrals of functions from $W^{(1)}(M, G)$ with partially continuous derivatives limited by M. It consists in the following. The domain G is divided into subdomains G_j which are uniformly small by probability and by measure, i.e.,

$$p_j = \int_{G_j} p(x)\,\mathrm{d}x \leqslant \frac{c_1}{N}, \qquad d_j = \sup_{x_1,\, x_2 \in G_j} |x_1 - x_2| \leqslant \frac{c_n}{N^{-1/2}}.$$

Random points are generated in different subdomains. This allows to obtain a greater rate of convergence with $\epsilon(n) = 1/n$.

In [4] an overconvergent Monte Carlo method for solving (11), (12) has been considered. The algorithm results from the special way of separation of the domain G.

In this case the random error becomes

$$r_N = cN^{-1/2-1/(n(k-1))},$$

i.e., $\epsilon(n) = 1/(n(k-1))$, where k comes from the truncated Neumann series (2).

In this case every Markov chain is realized in one of the subdomains G_j. This means that the only difference between the algorithm described above and this algorithm is the probability density function which does not change the speedup and efficiency estimators.

In [3] an optimum (with a minimum standard deviation) algorithm for problem (9), (10) is proposed. This algorithm has a minimum probable error.

It is obtained using the idea of the importance sampling technique for construction of the transition probabilities in the Markov chain.

As the only difference between this algorithm and the classical one is the probability density function, it is obvious that the estimations for the speedup and the efficiency are the same.

However, the Monte Carlo algorithms described above are not well suited for pipeline architectures. Thus, if one wants to achieve greater speedup on pipeline computers, it is necessary to construct special Monte Carlo algorithms.

6. A Monte Carlo vector algorithm

There are Monte Carlo algorithms called vector algorithms which are more efficient for pipeline computers.

Let us now consider an approach to solve the problem

$$R_m(\Delta)u(x) = (-1)^m f(x), \quad x \in G \subset \mathbb{R}^2, \tag{91a}$$

$$\Delta^k u(x) \to f_{km}(x), \quad x \to x_0, \; x_0 \in \partial G, \tag{91b}$$

where

$$R_m(\lambda) = \lambda^m + c_1\lambda^{m-1} + \cdots + c_{m-1}\lambda + c_m \tag{92}$$

is a polynomial with real zeros and Δ is the Laplace operator [8]. This is a common problem in the investigation of the beam penetration in multilayer targets.

Suppose that all conditions ensuring the existence and the uniqueness of the solution of problem (91) are met [14] and all zeros of the polynomial $R_m(\lambda)$ are real numbers. Let $\lambda_1, \ldots, \lambda_m$ be the zeros of $R_m(\lambda)$.

Then the problem (91), (92) can be given the following representation:

$$\begin{cases} (\Delta - \lambda_1)u_1 = (-1)^m f(x), \\ (\Delta - \lambda_2)u_2 = u_1, \\ \vdots \\ (\Delta - \lambda_m)u_m = u_{m-1}, \\ u_m = u, \end{cases} \tag{93}$$

with the boundary conditions

$$u_s(x) \to f_s(x) + b_1^s f_{s-1}(x) + \cdots + b_{s-1}^s f_1 = W_s(x), \quad s = 1, \ldots, m, \tag{94}$$

where

$$\begin{cases} b_j^s = b_j^{s+1} + \lambda_s b_{j-1}^s, \quad j = 1, \ldots, s-1, \\ b_0^s = 1, \qquad b_k^{m+1} = c_k, \quad k = 1, \ldots, m-1. \end{cases} \tag{95}$$

Then the discrete problem becomes

$$u_{k,0} = \tfrac{1}{4}\sum_{i=1}^{4} u_{k,i} - \tfrac{1}{4}h^2\lambda_k u_{k,0} - \tfrac{1}{4}h^2 u_{k-1,0}, \quad k = 1, \ldots, m, \tag{96}$$

$$u_{0,0} = f_0 = (-1)^m f(x), \quad x \in \boldsymbol{G}_h. \tag{97}$$

Using the notation $k = (1 + \frac{1}{4}h^2\lambda_i)^{-1}$, $i = 1, \ldots, m$, one can obtain the following system of linear algebraic equations:

$$\begin{cases} u_{m,0} = k_m\left(\dfrac{1}{4}\displaystyle\sum_{k=0}^{4} u_{m,i} - \tfrac{1}{4}h^2 u_{m-1,0}\right), \\ u_{m-1,0} = k_{m-1}\left(\dfrac{1}{4}\displaystyle\sum_{i=0}^{4} u_{m-1,i} - \tfrac{1}{4}h^2 u_{m-2,0}\right), \\ \vdots \\ u_{1,0} = k_1\left(\dfrac{1}{4}\displaystyle\sum_{i=0}^{4} u_{1,i} - \tfrac{1}{4}h^2 f_0\right), \quad i = 1, \ldots, m, \end{cases} \tag{98}$$

or represented in a matrix form:

$$\boldsymbol{U} = A\boldsymbol{U} + \boldsymbol{F}, \tag{99}$$

where $U=(u_{m,0}, u_{m-1,0},\dots,u_{1,0})^{\mathrm{T}}$ and each $u_{k,0}$ is a vector with m_1m_2 components (here m_1 and m_2 are the number of points along x_1 and x_2, respectively),

$$A=\begin{pmatrix} k_mL_h & -\frac{1}{4}h^2k_mk_{m-1}L_h & \cdots & \left(-\frac{1}{4}h^2\right)^{m-1}\prod_{i=1}^{m}k_iL_h \\ 0 & k_{m-1}L_h & \cdots & \left(-\frac{1}{4}h^2\right)^{m-2}\prod_{i=1}^{m-1}k_iL_h \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & k_1L_h \end{pmatrix}, \tag{100}$$

$$F=\left(\prod_{i=1}^{m}k_i\left(-\tfrac{1}{4}h^2\right)^m,\dots,\left(-\tfrac{1}{4}h^2\right)^2k_2k_1,\left(-\tfrac{1}{4}h^2\right)k_1\right)^{\mathrm{T}}(-1)^m f(x). \tag{101}$$

In what follows we will denote by $W=(w_n(x),\dots,w_1(x))^{\mathrm{T}}$ the boundary conditions vector.

Obviously, (99) is of type (20) but one in which u and f are vector functions.

This interpretation allows for constructing a Monte Carlo vector algorithm. We introduce matrix weights in the following manner:

$$Q_0=\{\delta_{ij}\}_{i,j=1}^{m}, \qquad Q_j=Q_{j-1}\frac{k(x_{j-1},x_j)}{p(x_{j-1},x_j)}, \tag{102}$$

where

$$k(x_{n-1},x_n)=\prod_{l=i}^{j}k_{m+1-l}\left(-\tfrac{1}{4}h^2\right)^{l-i}L_h, \quad i,j=1,\dots,m. \tag{103}$$

Then the random variable ξ_x with $E\xi_x=J(U)$ is

$$\xi_x=\sum_{j=0}^{i^*}Q_jF(x_j), \tag{104}$$

where i^* is a random variable that is the length of the Markov chain constructed, and $J(U)$ is a linear functional of the solution U. Each Markov chain is a realization of the random variable ξ_x. From the law of large numbers follows that we need N realizations to estimate the mathematical expectation of ξ_x, i.e., the linear functional of the solution U.

Each realization is done according to the following algorithm.

(i) The process begins at the point x_0, chosen according to the initial density, and accumulate in ξ_0 the values of F_0.

(ii) From the point x_k we go to the next point x_l depending on the transition densities p_{kl} and

– if $x\in G_h$, then $\xi_0=\xi_0+Q_1F_l$ and the process continues with a transition to the next point (according to step (ii));

– if $x\in\partial G_h$, then $\xi_0=\xi_0+Q_1W_l$ and the process is terminated;

– if the boundary ∂G_h is reached at the point x_{i^*} and on the sth step, then

$$\xi_0=\xi_0+Q_sW_{i^*}. \tag{105}$$

This algorithm is used to solve the problem

$$\Delta\Delta\Delta u(x) = (-1)^3 f(x), \quad x \in G, \ x = (x_1, x_2), \tag{106}$$

$$\Delta^k u(x) \to f_{k+1}(x), \quad \text{when } x \to x_0, \ x_0 \in \partial G, \ k = 0, 1, 2, \tag{107}$$

where $G = \{(x_1, x_2): 0 \leqslant x_1 \leqslant 1, 0 \leqslant x_2 \leqslant 1\}$ is the unit square, and

$$\begin{cases} f(x) = 2 \cdot 3^6 \sin 3x_1 + 2^6 \cos 2x_2, \\ f_1(x) = 2 \sin 3x_1 + \cos 2x_2, \\ f_2(x) = -18 \sin 3x_1 - 4 \cos 2x_2, \\ f_3(x) = 2 \cdot 3^4 \sin 3x_1 + 2^4 \cos 2x_2. \end{cases} \tag{108}$$

The corresponding system of differential equations is

$$\begin{cases} \Delta u_1 = (-1)^3 f(x), \\ \Delta u_2 = u_1, \\ \Delta u_3 = u_2, \\ u_3 = u, \end{cases} \tag{109}$$

and using a grid of mesh size $h = 0.1$, we obtain the difference system

$$\boldsymbol{u} = A\boldsymbol{u} + \boldsymbol{F}, \tag{110}$$

where

$$\boldsymbol{u} = (u_{3,0}, u_{2,0}, u_{1,0})^{\mathrm{T}}, \tag{111}$$

$$A = \begin{pmatrix} L_h & -\frac{1}{4}h^2 L_h & \left(-\frac{1}{4}h^2\right)^2 L_h \\ 0 & L_h & -\frac{1}{4}h^2 L_h \\ 0 & 0 & L_h \end{pmatrix}, \tag{112}$$

$$\boldsymbol{F} = \left(\left(-\tfrac{1}{4}h^2\right)^3, \left(-\tfrac{1}{4}h^2\right)^2, \left(-\tfrac{1}{4}h^2\right)\right)^{\mathrm{T}} (-1)^3 f(x), \tag{113}$$

and the boundary conditions vector is

$$W = (f_3(x), f_2(x), f_1(x))^{\mathrm{T}}.$$

The random vector ξ_k is

$$\xi_k = \xi_k + Q_s \Phi_s, \tag{114}$$

Table 1

Points	Exact solution	Monte Carlo solution	Relative error
(0.6, 0.8)	2.6444	2.6790	−0.013
(0.2, 0.5)	1.6696	1.6895	−0.012

where

$$Q_s\Phi_l = \begin{cases} Q_s F_l, & x_l \in G_h, \\ Q_s W_l, & x_l \in \partial G_h, \end{cases} \tag{115}$$

$$Q_s F_l = \begin{pmatrix} R(s+1)\left(-\frac{1}{4}h^2\right)^3 \\ (s+1)\left(-\frac{1}{4}h^2\right)^2 \\ -\frac{1}{4}h^2 \end{pmatrix} (-1)^3 f(x), \tag{116}$$

$$Q_s W_l = \begin{pmatrix} f_3 + s\left(-\frac{1}{4}h^2\right) f_2 + R(s)\left(-\frac{1}{4}h^2\right)^2 f_1 \\ f_2 + s\left(-\frac{1}{4}h^2\right) f_1 \\ f_1 \end{pmatrix}, \tag{117}$$

where $R(1) = 1$, $R(s) = s + R(s-1)$.

In Table 1 the results from the solution of the problem in two points over 3000 realizations of the random variable are shown.

Appendix A

Consider the boundary-value problem

$$\Delta u(x) = 0, \quad x = (x_1, x_2, x_3) \in G, \tag{A.1}$$

and the boundary condition

$$u(x) = \psi(x), \quad x \in \partial G. \tag{A.2}$$

Using the finite-difference technique, one can derive a system of linear equations whose solution approximates the solution of (A.1). The system depends on the approximation molecule for the Laplace operator.

Let us consider the usual seven-point approximation molecule. The equation which approximates (A.1) in the point $x_i = (i_1 h, i_2 h, i_3 h)$ is

$$\Lambda_1(u_i) - 6u_i = 0, \tag{A.3}$$

where

$$\begin{aligned} \Lambda_1(u_i) = {} & u((i_1+1)h, i_2 h, i_3 h) + u((i_1-1)h, i_2 h, i_3 h) + u(i_1 h, (i_2+1)h, i_3 h) \\ & + u(i_1 h, (i_2-1)h, i_3 h) + u(i_1 h, i_2 h, (i_3+1)h) + u(i_1 h, i_2 h, (i_3-1)h), \end{aligned} \tag{A.4}$$

and h is the mesh size of the discrete domain ∂G_h. For brevity, in what follows we assume h to be unity.

Using (A.3) for all terms in (A.4) we obtain

$$u_i = \tfrac{1}{36}\left[\Lambda_2(u_i) + 2\Lambda_{1,m}(u_i) + 6u_i\right], \tag{A.5}$$

where

$$\begin{aligned}\Lambda_{1,m}(u_i) = {} & u(i_1+1, i_2, i_3+1) + u(i_1-1, i_2, i_3+1) + u(i_1-1, i_2, i_3-1) \\ & + u(i_1+1, i_2, i_3-1) + u(i_1+1, i_2+1, i_3) + u(i_1, i_2+1, i_3+1) \\ & + u(i_1-1, i_2+1, i_3) + u(i_1, i_2+1, i_3-1) + u(i_1+1, i_2-1, i_3) \\ & + u(i_1, i_2-1, i_3+1) + u(i_1-1, i_2-1, i_3) + u(i_1, i_2-1, i_3-1), \end{aligned} \tag{A.6}$$

and $\Lambda_2(u)$ is obtained from the formula

$$\begin{aligned}\Lambda_k(u_i) = {} & u(i_1+k, i_2, i_3) + u(i_1-k, i_2, i_3) + u(i_1, i_2+k, i_3) \\ & + u(i_1, i_2-k, i_3) + u(i_1, i_2, i_3+k) + u(i_1, i_2, i_3-k),\end{aligned}$$

when $k = 2$.

If we use the Taylor formula for terms in $\Lambda_{1,m}(u_i)$, it is easy to construct another approximation molecule for (A.1) which leads to

$$\Lambda_{1,m}(u_i) = 12u_i. \tag{A.7}$$

Then (A.5) becomes

$$u_i = \tfrac{1}{6}\Lambda_2(u_i), \tag{A.8}$$

which is of the same type as (A.3) but the step in the approximation molecule is 2. Application of the algorithm described above leads to the following theorem.

Theorem 1. *Let $x_i = (i_1, i_2, i_3)$ be an arbitrary point in G_h and k be the radius of the largest sphere in G_h with the centre in x_i. Then the following equation holds*:

$$u_i = \tfrac{1}{6}\Lambda_k(u_i). \tag{A.9}$$

To prove this theorem some preliminary statements are needed.

Lemma 2. *For each integer k the following formula holds*:

$$\Lambda_k(u_i) = \tfrac{1}{6}\left[\Lambda_{k+1}(u_i) + \Lambda_{k-1}(u_i) + \tilde{\Lambda}_k(u_i)\right], \tag{A.10}$$

where

$$\begin{aligned}\tilde{\Lambda}_k(u_i) = {} & u(i_1+k, i_2+1, i_3) + u(i_1+k, i_2, i_3+1) + u(i_1+k, i_2-1, i_3) \\ & + u(i_1+k, i_2, i_3-1) + u(i_1-k, i_2+1, i_3) + u(i_1-k, i_2, i_3+1) \\ & + u(i_1-k, i_2-1, i_3) + u(i_1-k, i_2, i_3-1) + u(i_1+1, i_2+k, i_3) \\ & + u(i_1, i_2+k, i_3+1) + u(i_1-1, i_2+k, i_3) + u(i_1, i_2+k, i_3-1) \\ & + u(i_1+1, i_2-k, i_3) + u(i_1, i_2-k, i_3+1) + u(i_1-1, i_2-k, i_3) \\ & + u(i_1, i_2-k, i_3-1) + u(i_1+1, i_2, i_3+k) + u(i_1, i_2+1, i_3+k) \\ & + u(i_1-1, i_2, i_3+k) + u(i_1, i_2-1, i_3+k) + u(i_1+1, i_2, i_3-k) \\ & + u(i_1, i_2+1, i_3-k) + u(i_1-1, i_2, i_3-k) + u(i_1, i_2-1, i_3-k).\end{aligned}$$

The proof of Lemma 2 follows from (A.3) and (A.6).

Lemma 3. *For an arbitrary integer k it follows that*

$$\tilde{\Lambda}_k(u_i) = \begin{cases} \sum_{l=0}^{(k-3)/2} (-1)^l \left(12\Lambda_{k-2l-1}(u_i) - \overline{\Lambda}_{k-2l-1}(u_i)\right) + (-1)^{[k/2]}\tilde{\Lambda}_1(u_i), \\ \qquad k \text{ odd}, \\ \sum_{l=0}^{(k-2)/2} (-1)^l \left(12\Lambda_{k-2l-1}(u_i) - \overline{\Lambda}_{k-2l-1}(u_i)\right) + (-1)^{[k/2]}\tilde{\Lambda}_0(u_i), \\ \qquad k \text{ even}, \end{cases} \tag{A.11}$$

where $[t]$ *means the integer part of* t, *and*

$$\begin{aligned} &\overline{\Lambda}_k(u_i) \\ &= u(i_1+k, i_2+1, i_3-1) + u(i_1+k, i_2+1, i_3+1) + u(i_1+k, i_2-1, i_3+1) \\ &\quad + u(i_1+k, i_2-1, i_3-1) + u(i_1-k, i_2+1, i_3-1) + u(i_1-k, i_2+1, i_3+1) \\ &\quad + u(i_1-k, i_2-1, i_3+1) + u(i_1-k, i_2-1, i_3-1) + u(i_1+1, i_2+k, i_3-1) \\ &\quad + u(i_1-1, i_2+k, i_3-1) + u(i_1-1, i_2+k, i_3+1) + u(i_1+1, i_2+k, i_3+1) \\ &\quad + u(i_1+1, i_2-k, i_3-1) + u(i_1-1, i_2-k, i_3+1) + u(i_1-1, i_2-k, i_3-1) \\ &\quad + u(i_1+1, i_2-k, i_3+1) + u(i_1+1, i_2-1, i_3+k) + u(i_1+1, i_2+1, i_3+k) \\ &\quad + u(i_1-1, i_2+1, i_3+k) + u(i_1-1, i_2-1, i_3+k) + u(i_1+1, i_2-1, i_3-k) \\ &\quad + u(i_1+1, i_2+1, i_3-k) + u(i_1-1, i_2+1, i_3-k) + u(i_1-1, i_2-1, i_3-k). \end{aligned}$$

Proof. Using formula (A.7) for each term in $\Lambda_{k-1}(u_i)$, we obtain

$$12\Lambda_{k-1}(u_i) = \tilde{\Lambda}_k(u_i) + \overline{\Lambda}_{k-1}(u_i) + \tilde{\Lambda}_{k-2}(u_i)$$

or

$$\tilde{\Lambda}_k(u_i) = \tilde{\Lambda}_{k-2}(u_i) + \overline{\Lambda}_{k-1}(u_i) - 12\Lambda_{k-1}(u_i), \tag{A.12}$$

and applying it recursively yields (A.11). □

Lemma 4. *For an arbitrary integer k the following formula holds*:

$$\overline{\Lambda}_k(u_i) = \begin{cases} 8\sum_{l=0}^{(k-3)/2} (-1)^l \Lambda_{k-2l-1}(u_i) + (-1)^{[k-2]}\overline{\Lambda}_1(u_i), & k \text{ odd}, \\ 8\sum_{l=0}^{(k-2)/2} (-1)^l \Lambda_{k-2l-1}(u_i) + (-1)^{[k-2]}\overline{\Lambda}_0(u_i), & k \text{ even}. \end{cases} \tag{A.13}$$

Proof. Using the Taylor formula one can derive the approximation formula

$$\Lambda_{1,e}(u_i) = 8u_i, \tag{A.14}$$

where $\Lambda_{1,e}(u_i) = \frac{1}{3}\overline{\Lambda}_1(u_i)$.

Then applying this formula for all terms in $\Lambda_{k-1}(u_i)$, we obtain

$$8\Lambda_{k-1}(u_i) = \tilde{\Lambda}_k(u_i) + \overline{\Lambda}_{k-2}(u_i)$$

or

$$\overline{\Lambda}_k(u_i) = -8\Lambda_{k-1}(u_i) + \overline{\Lambda}_{k-2}(u_i), \tag{A.15}$$

which leads to (A.13). □

Proof of Theorem 1. When k is unity, (A.9) becomes the usual approximation formula (A.3). We will prove (A.9) when $k = n$, provided it holds for all $k = 2, 3, \ldots, n-1$.

From Lemma 2 it follows that

$$\Lambda_n(u_i) = \tfrac{1}{6}\left[\Lambda_{n+1}(u_i) + \Lambda_{n-1}(u_i) + \overline{\Lambda}_n(u_i)\right],$$

and according to the above assumption

$$\Lambda_n(u_i) = \tfrac{1}{6}\left[\Lambda_{n+1}(u_i) + 6u_i + \overline{\Lambda}_n(u_i)\right],$$

and so for $k = n$, (A.9) becomes

$$u_i = \tfrac{1}{36}\left[\Lambda_{n+1}(u_i) + 6u_i + \tilde{\Lambda}_n(u_i)\right]. \tag{A.16}$$

Without any restrictions of generality we assume that $n = 2m-1$. So using Lemmas 3 and 4 we obtain

$$\begin{aligned}\tilde{\Lambda}_{2m-1}(u_i) &= \sum_{l=0}^{m-2}(-1)^l\left[12\Lambda_{2(m-l-1)}(u_i) - \overline{\Lambda}_{2(m-l-1)}(u_i)\right] + (-1)^m\tilde{\Lambda}_1(u_i)\\ &= \sum_{l=0}^{m-2}(-1)^l\Bigg[12\Lambda_{2(m-l-1)}(u_i) - 8\sum_{s=0}^{m-l-2}(-1)^s\Lambda_{2(m-2l)-3}(u_i)\\ &\qquad -(-1)^{m-l-1}\overline{\Lambda}_0(u_i)\Bigg] + (-1)^m\tilde{\Lambda}_1(u_i).\end{aligned}$$

From the definitions follows that

$$\overline{\Lambda}_0(u_i) = 24u_i \quad \text{and} \quad \tilde{\Lambda}_1(u_i) = 24u_i,$$

and from the assumption that

$$\Lambda_j(u_i) = 6u_i,$$

when $j < n$. Then

$$\begin{aligned}\tilde{\Lambda}_{2m-1}(u_i) &= \sum_{l=0}^{m-2}(-1)^l\left[72u_i - 8\sum_{s=0}^{m-l-2}(-1)^s 6u_i - (-1)^{m-l-1}24u_i\right] + (-1)^m 24u_i\\ &= 72u_i\sum_{l=0}^{m-2}(-1)^l - 48u_i\sum_{l=0}^{m-2}(-1)^l\sum_{s=0}^{m-l-2}(-1)^s - \sum_{l=0}^{m-2}(-1)^{m-1}24u_i\\ &\quad + (-1)^m 24u_i = 24u_i,\end{aligned}$$

and (A.16) becomes

$$u_i = \tfrac{1}{36}[\Lambda_{n+1}(u_i) + 30u_i] \quad \text{or} \quad u_i = \tfrac{1}{6}\Lambda_{n+1}(u_i). \tag{A.17}$$

The case when $k = 2m$ is similar. □

Theorem 1 is used to construct a Monte Carlo method for finding the inner product of a given vector $\boldsymbol{g}$ with the solution of the system (A.3).

The algorithm is as follows.

(i) The start point x_0 is selected according to a density admissible for $\boldsymbol{g}$.

(ii) Determine the mesh distance $d_h(x_0)$ from the selected point x_0 to the boundary; the next point is selected from among the neighbours on the seven-point approximation molecule with step $d_h(x_0)$;

- if the point is on the boundary, the process terminates;
- otherwise the process continues with (ii).

Appendix B

Here all the results for the values of interest are summarized.

B.1. Algorithm $\mathscr{A}$ *(*$f(x) \equiv 0$, $p = (c_{0.5}\sigma(\theta)/\epsilon)^2$*)*

$$\mathrm{E}T_1(\mathscr{A}) = \tau((k+1+\gamma_\mathrm{A})l_\mathrm{A} + (n+1+\gamma_\mathrm{L})l_\mathrm{L})\frac{1}{4\epsilon}\left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2,$$

$$\mathrm{E}T_{\mathrm{pipe}}(\mathscr{A}) = \tau(s+k+l_\mathrm{A}+\gamma_\mathrm{A} + (n+1+\gamma_\mathrm{L})l_\mathrm{L})\frac{1}{4\epsilon}\left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2,$$

$$\mathrm{E}T_p(\mathscr{A}) = \tau((k+1+\gamma_\mathrm{A})l_\mathrm{A} + (n+1+\gamma_\mathrm{L})l_\mathrm{L})\frac{1}{4\epsilon},$$

$$\mathrm{E}T_{2np}(\mathscr{A}) = \tau((k+1+\gamma_\mathrm{A})l_\mathrm{A} + 3l_\mathrm{L})\frac{1}{4\epsilon},$$

$$S_{\mathrm{pipe}}(\mathscr{A}) = \left(1 + \frac{k+1+\gamma_\mathrm{A}}{n+1+\gamma_\mathrm{L}}\frac{l_\mathrm{A}}{l_\mathrm{L}}\right)\Bigg/\left(1 + \frac{s+k+l_\mathrm{A}+\gamma_\mathrm{A}}{n+1+\gamma_\mathrm{L}}\frac{1}{l_\mathrm{L}}\right),$$

$$S_p(\mathscr{A}) = p,$$

$$S_{2np}(\mathscr{A}) = p\left(1 + \frac{n+1+\gamma_\mathrm{L}}{k+1+\gamma_\mathrm{A}}\frac{l_\mathrm{L}}{l_\mathrm{A}}\right)\Bigg/\left(1 + \frac{3}{k+1+\gamma_\mathrm{A}}\frac{l_\mathrm{L}}{l_\mathrm{A}}\right),$$

$$E_p(\mathscr{A}) = 1,$$

$$E_{2np}(\mathscr{A}) = \frac{1}{2n}\left(1 + \frac{n+1+\gamma_\mathrm{L}}{k+1+\gamma_\mathrm{A}}\frac{l_\mathrm{L}}{l_\mathrm{A}}\right)\Bigg/\left(1 + \frac{3}{k+1+\gamma_\mathrm{A}}\frac{l_\mathrm{L}}{l_\mathrm{A}}\right).$$

B.2. Algorithm $\mathscr{B}$ ($f(x) \equiv 0$, $n = 3$, $p = (c_{0.5}\sigma(\theta)/\epsilon)^2$)

$$\mathrm{E}T_1(\mathscr{B}) = 6\tau((k+1+\gamma_A)l_A + 9l_L)\left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2,$$

$$\mathrm{E}T_{\mathrm{pipe}}(\mathscr{B}) = 6\tau(s+k+l_A+\gamma_A+9l_L)\left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2,$$

$$\mathrm{E}T_p(\mathscr{B}) = 6\tau((k+1+\gamma_A)l_A + 9l_L),$$

$$\mathrm{E}T_{6p}(\mathscr{B}) = 6\tau((k+1+\gamma_A)l_A + 5l_L),$$

$$S_{\mathrm{pipe}}(\mathscr{B}) = \left(1+\tfrac{1}{9}(k+1+\gamma_A)\frac{l_A}{l_L}\right)\Big/\left(1+\tfrac{1}{9}(s+k+l_A+\gamma_A)\frac{1}{l_L}\right),$$

$$S_p(\mathscr{B}) = p,$$

$$S_{6p}(\mathscr{B}) = p\left(1+\frac{9}{k+1+\gamma_A}\frac{l_L}{l_A}\right)\Big/\left(1+\frac{5}{k+1+\gamma_A}\frac{l_L}{l_A}\right),$$

$$E_p(\mathscr{B}) = 1,$$

$$E_{6p}(\mathscr{B}) = \frac{1}{6}\left(1+\frac{9}{k+1+\gamma_A}\frac{l_L}{l_A}\right)\Big/\left(1+\frac{5}{k+1+\gamma_A}\frac{l_L}{l_A}\right).$$

B.3. Algorithm $\mathscr{C}$ ($f(x) \equiv 0$, $n = 3$, $p = (c_{0.5}\sigma(\theta)/\epsilon)^2$)

$$\mathrm{E}T_1(\mathscr{C}) = \tau((2k+\gamma_A+q_A)l_A + (\gamma_L+1)l_L)c\ \ln\epsilon\left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2,$$

$$\mathrm{E}T_{\mathrm{pipe}}(\mathscr{C}) = \tau(s+2k+\gamma_A+q_A+l_A-1+(\gamma_L+1)l_L)c\ \ln\epsilon\left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2,$$

$$\mathrm{E}T_p(\mathscr{C}) = \tau((k+\gamma_A+q_A)l_A + (\gamma_L+1)l_L)c\ \ln\epsilon,$$

$$\mathrm{E}T_{6p}(\mathscr{C}) = \tau((2k+\gamma_A+q_A)l_A + (\gamma_L+1)l_L)c\ \ln\epsilon,$$

$$S_{\mathrm{pipe}}(\mathscr{C}) = \left(1+\frac{2k+\gamma_A+q_A}{\gamma_L+1}\frac{l_A}{l_L}\right)\Big/\left(1+\frac{s+2k+\gamma_A+q_A+l_A-1}{\gamma_L+1}\frac{1}{l_L}\right),$$

$$S_p(\mathscr{C}) = p,$$

$$S_{6p}(\mathscr{C}) = p\left(1+\frac{2k+\gamma_A+q_A}{\gamma_L+1}\frac{l_A}{l_L}\right)\Big/\left(1+\frac{k+\gamma_A+q_A}{\gamma_L+1}\frac{l_A}{l_L}\right),$$

$$E_p(\mathscr{C}) = 1,$$

$$E_{6p}(\mathscr{A}) = \frac{1}{6}\left(1+\frac{2k+\gamma_A+q_A}{\gamma_L+1}\frac{l_A}{l_L}\right)\Big/\left(1+\frac{k+\gamma_A+q_A}{\gamma_L+1}\frac{l_A}{l_L}\right).$$

Acknowledgement

The authors wish to thank Miss Lidiya Todorova for helping with the computer programs and calculations concerning the Monte Carlo vector algorithm.

References

[1] D.R. Cox and W.L. Smith, *Renewal Theory* (Sovetskoe Radio, Moscow, 1967) (in Russian).
[2] J.H. Curtiss, "Monte Carlo" methods for the iteration of linear operators, *J. Math. Phys.* **32** (4) (1954) 209–232.
[3] I.T. Dimov, Minimization of the probable error for some Monte Carlo methods, in: *Proc. Summer School on Mathematical Modelling and Scientific Computations,* Albena, Bulgaria, 1990, CINTI, Reg. No. II 14147.
[4] I. Dimov and O. Tonev, Monte Carlo numerical methods with overconvergent probable error, in: Bl. Sendov, R. Lazarov and Iv. Dimov, Eds., *Proc. Internat. Conf. on Numerical Methods and Applications,* Sofia, 1988 (Publishing House of the Bulgarian Academy of Sciences, Sofia, 1989) 116–120.
[5] S.M. Ermakov and G.A. Mikhailov, *Statistical Simulation* (Nauka, Moscow, 1982) (in Russian).
[6] R.W. Hockney and C.R. Jesshope, *Parallel Computers: Architecture, Programming and Algorithms* (Adam Hilger, Bristol, 1981).
[7] P.M. Kogge, *The Architecture of Pipeline Computers* (Hemisphere, Washington, DC, 1985).
[8] S.E. Makharov, Discrete random walk vector algorithm for solving high order equations, in: G.A. Mikhailov, Ed., *Theory and Algorithms of the Statistical Modelling,* Novosibirsk (1984) 54–66 (in Russian).
[9] N. Metropolis and S. Ulam, The Monte Carlo method, *J. Amer. Statist. Assoc.* **44** (1949) 335–341.
[10] G.A. Mikhailov, *Optimization of the Weighted Monte Carlo Methods* (Nauka, Moscow, 1985) (in Russian).
[11] M.E. Muller, Some continuous Monte Carlo methods for the Dirichlet problem, *Ann Math. Statist.* **27** (1956) 569–589.
[12] J.M. Ortega and R.G. Voigt, Solution of partial differential equations on vector and parallel computers, *SIAM Rev.* **27** (2) (1985) 149–240.
[13] I.M. Sobol', *The Monte Carlo Numerical Methods* (Nauka, Moscow, 1973) (in Russian).
[14] S.V. Vladimirov, *Equations of Mathematical Physics* (Nauka, Moscow, 1976) (in Russian).