*Russ. J. Numer. Anal. Math. Modelling*, Vol. 17, No. 1, pp. 113–124 (2002)


# Parallel realization of statistical simulation and random number generators

M. A. MARCHENKO* and G. A. MIKHAILOV*

**Abstract** — In this paper we consider various aspects of parallel realization of the Monte Carlo method algorithms on multiprocessor computation systems. In particular, we consider the distribution of 'statistical tests' among various processors and the correlation of the corresponding results with the aim to effectively estimate functional relationships in the metric $C$.

We developed the modification of an 'astronomically' long-period 'congruential' program pseudorandom number generator. It allows us to effectively distribute generated numbers essentially among an arbitrary number of processors. We conducted multidimensional uniformity tests for the modified generator by the '$\chi$-square' criterion to the seventh dimension. All the tests gave satisfactory results. The corresponding computational routines in Fortran 90 are given in the Section 4.

It is clear that the use of $S$ independent processors reduces the computational cost of statistical simulation by a factor of $S$ due to the distribution of independent tests among them. This is because the costs of final summation and the averaging of the results are essentially insignificant. Certainly, we should admit the possibility to realize various sample volumes on various processors, using the statistically optimal method of averaging the results by the formulae of the form

$$\bar{x} = \sum_{i=1}^{S} n_i \bar{x}_i \Big/ \sum_{i=1}^{S} n_i$$

where $n_i$ is the sample volume for the $i$-th processor, $\bar{x}_i$ is the corresponding mean value. These parallel computations can be particularly advantageous for estimation of the mean values of the studied functionals in problems with random parameters because the use of the 'double randomization method' (see [2, 8]) results in a substantial increase in the effective dimension of the probabilistic space.

If $S$ is large, the necessary sample volume of base random numbers is large too. Therefore, on the one hand, it may be advisable to use combined random-pseudorandom subsequences (see Section 2). On the other hand, it is possible to use 'long-period' pseudorandom sequences if there exists a simple method of their partitioning into $S$ parts of necessary length (see Section 2, a bf-generator). These

*Institute of Computational Mathematics and Mathematical Geophysics, Siberian Branch of the Russian Academy of Sciences, Novosibirsk 630090, Russia

The work was supported by the Russian Foundation for Basic Research (99-00-90422), 'Leading Scientific Schools' (00-15-96173), and the Integration Grant of the Siberian Branch of the Russian Academy of Sciences (43).

modified algorithms for generating pseudorandom numbers require appropriately modified statistical testing (see Section 3).

Note that for the global estimate of the solution in the metric $C$ it is possible to simulate the sets of trajectories emerging from different points of the phase space on various processors, for example, to realize 'adjoint walks' when solving problems of the radiation transport theory [5]. It is advisable to use the same pseudorandom numbers for different points (possibly on various processors) and, in addition, to distribute these numbers among separate trajectories by a special determinate method (see Section 2, an lf-generator). This substantially reduces the needed sample volume of random numbers.

To conclude this section we emphasize that there is no 'ideal' parallel algorithm for simulating the stochastic ensemble of $N$ interacting particles. However, the asymptotic determinate error of the estimates of the studied functionals for this ensemble is generally expressed by the value $C_1N^{-1}$ and the corresponding probabilistic error by the value $C_2N^{-1/2}$. Therefore in order for the probabilistic error to be reduced it is advisable to independently simulate this ensemble $(C_2/C_1)^2N$ times on various processors, with the subsequent averaging of the estimates of the functionals as indicated above.

## 1. PARALLEL IMPLEMENTATION OF THE MONTE CARLO METHOD ALGORITHMS

We consider here the Monte Carlo method algorithms for estimating complex mathematical expectations, i.e. integrals (perhaps infinite-to-one ones) generally dependent on some parameters in a probabilistic measure. In this connection it should be noted that for the global estimate of the function

$$\varphi(x) = \int_\Omega g(x,\omega)P(d\omega)$$

in the bounded domain $D \subset R^k$ we can estimate its values at mesh nodes with step $h$ by the Monte Carlo method and then perform linear filling. We denote this estimate by $\tilde{\varphi}(x)$. When there is a sufficiently large number of processors, it is logical to carry out computations for different nodes on various processors, using a bf-generator (see Section 2). The estimate $\tilde{\varphi}$ converges to $\varphi$ in probability in the metric of the space $L(D)$ if

$$B(\varphi,\tilde{\varphi}) = \mathbf{E}\|\varphi(x) - \tilde{\varphi}(x)\|_{L(D)} \to_{h\to 0} 0$$

which follows (see, e.g. [11]) from the Chebyshev inequality for the arbitrary random variable $\xi$:

$$P(|\xi| \geqslant \delta) \leqslant \mathbf{E}|\xi|/\delta \quad \forall\delta > 0.$$

The problem of reducing the value $B(\varphi,\tilde{\varphi})$ arises in the connection with the computational cost of the algorithm, i.e. with the average number of operations which ensure that the inequality $B(\varphi,\tilde{\varphi}) < \delta$ holds [9]. It is easier to estimate $B(\varphi,\tilde{\varphi})$

in the space $L_2(D)$:

$$B^2(\varphi,\tilde{\varphi}) \leqslant \int_D \mathbf{D}\tilde{\varphi}(x)\,dx + \int_D (\varphi(x) - \mathbf{E}\tilde{\varphi}(x))^2\,dx.$$

Assuming that all the partial second-order derivatives of $\varphi(x)$ are uniformly bounded, we have

$$B^2(\varphi,\tilde{\varphi}) \leqslant d/n + C_0 h^4$$

where $n$ is the sample volume in the Monte Carlo method. Consequently, the problem of approximate minimization of the computational cost can be formulated as

$$S_0 = nt_0 h^{-k} \to \min_{n,h}, \quad d/n + C_0 h^4 = \delta^2. \tag{1.1}$$

Here $t_0 = t\ \mathrm{mes}(D)$, where $t$ are the average cost of a single realization of the base random variable $\xi_x$ such that $\mathbf{E}\xi_x = \varphi(x)$. The problems of the form (2.1) are studied in detail in [7]. The optimal order of the values $n$, $h$, and $S_0$ is

$$h_0^* \asymp \delta^{1/2}, \quad n_0^* \asymp \delta^{-2}, \quad S_0^* \asymp \delta^{-(2+k/2)}. \tag{1.2}$$

In this case $B(\varphi,\tilde{\varphi}) < \delta$ and hence the choice of the algorithm parameters by (2.2) ensures convergence of $\tilde{\varphi}$ to $\varphi$ in probability in the metric of the space $L_2$ as $\delta \to 0$.

We obtain the analogous results for the metric $C(D)$ by using the theorem of embedding the space $W_2^l(D)$ into the space $C(D)$ for $2l > k$. If we use the first-order derivatives, we put $k = 1$, i.e. we consider the solution to the problem on a given line. Thus,

$$\|\varphi - \tilde{\varphi}\|_C^2 \leqslant K\left(\int_D (\varphi(x) - \tilde{\varphi}(x))^2\,dx + \int_D (\varphi'(x) - \tilde{\varphi}'(x))^2\,dx\right).$$

Especially efficient here is the dependent estimate of the values of the function $\varphi$ at mesh nodes, which gives the relation

$$|\tilde{\varphi}(x) - \tilde{\varphi}(x+h)| \leqslant Ch, \quad h = o(1)$$

with probability one. In this case instead of (2.1) we have the problem

$$S_1 = nt_0 h^{-1} \to \min_{n,h}, \quad d_1/n + C_1 h^2 = \delta^2$$

whose solution yields

$$h_1^* \asymp \delta, \quad n_1^* \asymp \delta^{-2}, \quad S_0^* \asymp \delta^{-3}.$$

As shown in the introduction, we can construct the dependent estimates of the values of $\varphi$, using the same sequences of pseudorandom numbers to estimate the values of $\varphi$ at different points and, in addition, distributing these numbers among separate random tests by an lf-generator (see Section 2).

It is not difficult to see that when the number of used processors for the studied problem of estimating $\varphi(\cdot)$ in the metric $C$ is of order $\delta^{-1}$, we obtain the computational cost of order $\delta^{-2}$ as is the case with the estimate of a single value of the integral. It is interesting to note that we obtain the analogous computational cost for the metric $L_2$ if $k = 1$ with $\delta^{-1/2}$ processors.

## 2. PARALLEL REALIZATION OF RANDOM NUMBER GENERATORS

We generally model the random variable $\xi$ with given distribution by transforming one or several independent values of the random number $\alpha$ uniformly distributed in the interval (0,1), i.e. by the formula

$$\xi = \varphi(\alpha_1, \ldots, \alpha_n).$$

**2.1.** The sequence of 'sample' values of $\alpha$ is generally obtained by theoretically numerical algorithms, of which the so-called residue method (or a congruential generator) is most frequently used in the form

$$u_0 = 1, \quad u_n \equiv u_{n-1} M (\mathrm{mod}\, 2^r), \quad \alpha_n = u_n 2^{-r}. \tag{2.1}$$

Here $r$ is generally the number of binary bits used to represent a number in a computer, $M$ is a sufficiently large number mutually disjoint with $2^r$. Let us call the value $M$ the generator multiplier. The values $\alpha_n$ of this type are called pseudorandom numbers. They are checked by statistical testing, by special analytical treatments, and by the solution to standard problems (see, e.g. [1, 2]). The period length of the sequence $\{\alpha_n\}$ for the generator (2.1) is $2^{r-2}$.

In order for the results of the solution to various problems to be correlated it is advisable to use the following order of use of pseudorandom numbers. The sequence $u_n$ is first split into subsequences of length $m$, beginning with the numbers $u_{km}, k = 0, 1, 2, \ldots$ (i.e. with the initial values of the subsequences). Each of the subsequences is used to construct the corresponding 'sample trajectories' of the process simulated (i.e. separate random tests). The value of $m$ must be chosen taking into account that $m$ pseudorandom numbers are essentially sufficient to construct a single trajectory. It is clear that when using the residue method the initial values of $u_{km}$ in the above subsequences are obtained by the formula

$$u_{(k+1)m} \equiv u_{km} M^m (\mathrm{mod}\, 2^r). \tag{2.2}$$

Thus, to simulate the $k$-th trajectory we use the subsequence of the residue method, which begins with

$$\alpha_{km} = u_{km} 2^{-r}.$$

Let us call this method of distributing random numbers among tests an lf- generator (the abbreviation of a 'little-frog' generator). The multiplier $M_m$ for an auxiliary generator which ensures 'jumps' with step length $m$ is calculated by the equality

$$M_m \equiv M^m (\mathrm{mod}\, 2^r). \tag{2.3}$$

It is clear that it is advisable to choose the value $m$ for the lf-generator so that it may be divided by all the values $n$ for which an $n$-dimensional uniformity test was realized (see Section 3).

It is obvious that unlike the ordinary method of distributing random numbers 'in succession' (i.e. in the order of calling the generator), the lf-generator ensures a

small change in the results of modelling, the problem parameters changing insignificantly. Thus, the lf-generator is checked more correctly by the solution of standard problems than the ordinary generator. Moreover, the lf-generator is best suited to most important multidimensional uniformity tests for problems of estimating multidimensional integrals (see Section 3).

**2.2.** It is clear that the modified generator studied also allows us to distribute pseudorandom numbers among separate processors, however, in this case the 'jumps' must be much longer. To be precise, the value $m$ should be replaced by $\mu = mN$, where $N$ is the number of trajectories which are essentially simulated on a single processor. The 'large-scale' generator thus obtained will be referred to as a bf-generator (the abbreviation of a 'big-frog' generator).

**2.3.** We now consider the possibility to combine the pseudorandom sequences with physical random number generators (see, e.g. [3]). It is clear that we can substitute the numbers from the physical generators for the initial numbers $\alpha_{km}$ corresponding to separate trajectories. This combined method can be justified theoretically as $M \to \infty$ (see [8, 12]).

Note that we can substantially improve the distribution of 'physical' random numbers, taking a modulo 1 sum (congruent summation), i.e. using the expression

$$\alpha = \operatorname{fract}\left(\sum_{i=1}^{n} \alpha_i\right)$$

where $\alpha_i$ are the numbers from the physical generator. The distribution of the value $\alpha$ converges very fast to the uniform distribution as $n \to \infty$ (see, e.g. [8]).

Moreover, the elementwise congruent summation of the independent sequences of 'real' random numbers effectively improves the multidimensional uniformity as is evident from the assertion below.

Suppose $\{p_i^{(l)}(\cdot)\}_{i=1,\ldots,n}$ are probability distribution densities in $[0,1]^l$, which correspond to independent physical generators. The distribution convolution, as usual, implies the sum distribution of the corresponding random variables.

**Theorem 2.1.** *If for $i = 1,\ldots,n$ the distribution densities $p_i^{(l)}(\cdot)$ of independent random points in $[0,1]$ are square-integrable and $P_n^{(l)}(\cdot)$ is their congruent n-multiple convolution, then*

$$\|P_n^{(l)}(\cdot) - 1\|_{L_\infty} \leqslant \prod_{i=1}^{n} \|p_i^{(l)}(\cdot) - 1\|_{L_2} \quad \forall l.$$

This theorem is a direct consequence of Proposition A.1 in [8].

When simulating the trajectories it is advisable to use the 'expensive' real refined values of $\alpha$ as initial $\{\alpha_{km}\}$ in the residue method. In order for the results of the solution to various problems to be correlated it is necessary to store them.

It is not necessary to test this combined method, given sufficiently large values of the multiplier $M$ in the residue method and the number of summable 'physical' sequences (i.e. the value $n$ in the theorem). Its practical implementation is important, in particular, for an independent check of the results obtained by the pseudorandom numbers.

## 3. THE CONSTRUCTION AND TEST OF A CONCRETE bf-GENERATOR

**3.1.** As the original generator we consider one of the congruential generators tested in [1] with the parameters

$$M = 5^{100109} (\mathrm{mod}\, 2^{128}), \quad r = 128 \tag{3.1}$$

i.e. with period length $L = 2^{126} \approx 10^{38}$. The value of the multiplier $M$ is given in Section 4. Standard statistical tests were successfully conducted in [1] for initial $10^9$ numbers (starting from $u_0 = 1$). It is shown in [16] that $n$-dimensional distributies are concentrated on families of planes for the residue method, i.e. on manifolds of smaller dimensions. Additional investigations of these distributions were carried in [1]. However, as the results obtained in [1] show, these manifolds for the generator studied fill sufficiently densely the corresponding $n$-dimensional unit hypercubes, and this drawback can be disregarded. The congruential generator routine with parameters (3.1) is given in Section 4.

**3.2.** The chosen value of the 'jump' length for the bf-generator is

$$\mu = 10^{26} \approx 2^{86}.$$

This number of pseudorandom numbers for each processor is more than sufficient to carry out computations. The proposed bf-generator allows us, generally speaking, to distribute the initial sequence by equal parts of length $10^{26}$ among about $10^{12} \approx 2^{40}$ processors. The multiplier for an auxiliary generator which ensures 'jumps' of length $\mu$ is calculated by formula (2.3), the multiplier itself and its computational routine are given in Section 4. The corresponding initial values of the first ten subsequences for the bf-generator $u_0 = 1$, $u_\mu = M_\mu$, $u_{2\mu} = M_{2\mu}, \ldots$ are also given in Section 4.

Having modified expression (2.2), we find that the initial values of the subsequences for the lf-generators can be calculated by the formula

$$u_{j+lm} \equiv u_{j+(l-1)m} M_m (\mathrm{mod}\, 2^{128}), \quad l = 0, 1, 2, \ldots, N \tag{3.2}$$

where $\{u_j\}_{j=0,\mu,2\mu,\ldots}$ are the initial values of the subsequences for the bf-generator. The computational routine for the values $u_{j+lm}$ is presented in Section 4.

**3.3.** In order for the constructed bf-generator to be checked, $n$-dimensional uniformity tests [2] were realized for the sample volume $10^{10}$ which resulted from

combining initial $10^9$ numbers of the first ten subsequences. The aim of this check is to simultaneously use ten processor units, no more than $10^9$ pseudorandom numbers are supposed to be used on each of them, with the subsequent averaging of the derived statistical estimates as shown in the introduction. The uniformity of the multivariate distributions was checked for $n = 1,2,\ldots,7$ by the $\chi^2$-criterion, with partitioning along each axis into 100 parts for $n = 2,3$ and into 10 parts for $n = 4,\ldots,7$. We denote by $k(n)$ the number of degrees of freedom in the distribution of $\chi^2$, which corresponds to the value $n$. Thus, the number of classes, i.e. elementary 'cubes', which is equal to $k(n)+1$, was determined by $10^{2n}$ for $n = 2,3$ and by $10^n$ for $n = 4,\ldots,7$.

We used for $n = 1$ the optimal (in the sense of the maximum power of the $\chi^2$-criterion number of classes, which is specified by the formula [4]

$$k(1)+1 \sim 4\sqrt[5]{2}(R/d_\alpha)^{2/5} \tag{3.3}$$

where $R$ is the sample volume and the constant $d_\alpha = O(1)$ can be taken to be 2. In this case $R = 10^{10}$ and $k(1)+1 \approx 34800$ according to (3.3).

We denote by $\tilde{\chi}^2_{k(n)}$ the sample value of the criterion:

$$\tilde{\chi}^2_{k(n)} = \frac{1}{r_n}\sum_{i=1}^{k(n)+1}(r_i^{(n)} - r_n)^2$$

where $r_n = R(n)/(k(n)+1)$ is the theoretical rate of falling into the class, $R(n)$ is the sample volume corresponding to the value of $n$, $\{r_i^{(n)}\}$ are sample rates obtained. In order to analyse the results of the statistical check of the constructed bf-generator, we took advantage of the fact that the value $\tilde{\eta}_n = (\tilde{\chi}^2_{k(n)} - k(n))/\sqrt{2k(n)}$ for the 'real' random numbers with the used values of $k(n)$ is conventional normal to a high accuracy. The relations hold for it:

$$\mathrm{P}(|\tilde{\eta}_n| > 1) \approx 0.32, \quad \mathrm{P}(|\tilde{\eta}_n| > 2) \approx 0.05, \quad \mathrm{P}(|\tilde{\eta}_n| > 3) \approx 0.003.$$

The computations for the above statistical check of the bf-generator were carried out on PC with the single processor Athlon 700 Mhz, RAM 256 Mb under the control of OC Windows 98. The routines were written in Fortran 90 and executed in the system Compaq Visual Fortran v.6.1. The total computation time was 27h. To save time the statistical tests for all the dimensions were conducted simultaneously for a single sample of the pseudorandom numbers. The numerical results obtained are given in Table 1.

Thus, the obtained values of $\tilde{\chi}^2_{k(n)}$ are not significant and the testing may be considered as a successful one.

**3.4.** The least common multiple of the values of $n$ in Table 1 is 420. Therefore it is advisable to put $m = 420 \cdot s$ for the bf-generator, where $s$ is such a number that $m$ pseudorandom numbers are essentially sufficient to construct a single test.

**Table 1.**
Results of the statistical check of the bf-generator multidimensional uniformity.

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $R(n)$ | $10^{10}$ | $5 \times 10^9$ | $\approx 3.3 \times 10^9$ | $2.5 \times 10^9$ | $2 \times 10^9$ | $\approx 1.67 \times 10^9$ | $\approx 1.43 \times 10^9$ |
| $k(n)$ | 34 799 | 9 999 | 999 999 | 9 999 | 99 999 | 999 999 | 9 999 999 |
| $\tilde{\eta}_n$ | 0.215 | 0.622 | 1.472 | −0.76 | 0.913 | −0.448 | 1.104 |

## 4. COMPUTATIONAL ROUTINES, CONSTANTS, OPERATING INSTRUCTIONS

All the routines in this section are written in Fortran 90.

**4.1.** The function 'rnd128' implements the congruential generator (2.1) with parameters (3.1). When writing it we took into account the recommendations in [10]. Therefore it actually includes operations with numbers in the numbering system $2^{13}$, which rules out the possibility of 'overflow'.

The function 'rnd128' returns a successive value of the pseudorandom number $\alpha_n$ in accordance with formula (2.1).The function has a single parameter 'un' which is a one-dimensional array of length 10 with the 'digits' of the corresponding number in the numbering system $2^{13}$, which are written in the reverse order (see Subsection 4.2). When 'rnd128' is called, the parameter 'un' corresponds to the value $u_{n-1}$ in the congruential generator formula (2.1). After the function returns a successive pseudorandom number, the parameter already corresponds to the value $u_n$. Therefore in order for the bf- or lf-generators to be used, when 'rnd128' is first called, the corresponding initial values of the subsequences must be passed to the parameter 'un'.

```
real*8 function rnd128(un)
implicit integer*4 (i-n)
integer*4 un(10)
integer*4 m(10), c(10,10), d(10)
real*8 x(10)
data m /1941,1821,3812,1310,68,2906,2335,2609,6859,1999/
data x /4.882812499999999999999999999969d-4, &
5.960464477539062499999999999962d-8, &
7.275957614183425903320312499995d-12, &
8.881784197001252323389053344726 6d-16, &
1.084202172485504434007452800869 9d-19, &
1.323488980084844279794253907311 9d-23, &
1.615587133892632177483220101699 1d-27, &
1.972152263052529513529321413207d-31, &
2.407412430484044816319972428231 2d-35, &
2.938735877055718769921841343055 6d-39 /
data mbase /8192/, mrem /2048/
do i = 1, 10
    do j = i, 10
        c(i,j) = un(i) * m( j + 1 - i )
    enddo
```

```
enddo
do i = 1, 10
    n = 0
    do j = 1, i
        n = n + c( j, i )
    enddo
    if ( i.gt.1 ) then
        d(i) = n + d(i-1) / mbase
    else
        d(i) = n
    endif
enddo
do i = 1, 10
    if ( i.lt.10 ) then
        un(i) = jmod( d(i), mbase )
    else
        un(i) = jmod( d(i), mrem )
    endif
enddo
rnd128 = 0.0d0
do i = 1, 10
    rnd128 = rnd128 + un(i)*x(11-i)
enddo
return
end
```

**4.2.** All the numerical values below are actually the 'digits' of the corresponding numbers to base $2^{13}$, which are written in the reverse order. For example, the 'digits' of the multiplier $M$ for the generator (2.1) with parameters (3.1) are

```
1941  1821  3812  1310  68   2906   2335   2609  6859  1999.
```

This implies that

$$M = 1941 + 1821 \times 2^{13} + 3812 \times 2^{2\times 13} + 1310 \times 2^{3\times 13} + 68 \times 2^{4\times 13} + 2906 \times 2^{5\times 13} + 2335 \times 2^{6\times 13} + 2609 \times 2^{7\times 13} + 6859 \times 2^{8\times 13} + 1999 \times 2^{9\times 13}.$$

The 'digits' of the multiplier $M_\mu$ for the bf-generator with 'jumps' of length $\mu = 10^{26}$ are

```
1   0  7916  6769   8113   7234   4142   5015   3567   1526.
```

The initial values of the first ten subsequences for the bf-generator are given in Table 2. These 'digits' are written in succession in the one-dimensional array 'un', which is passed to the function 'rnd128' as a parameter. The routine for obtaining these numbers is presented in Subsection 4.3.

**4.3.** It is clear that when using the bf-generator the corresponding initial values of the subsequences must be first distributed among the processors. At the same time

**Table 2.**
Initial values of the first ten subsequences for the bf-generator with parameters (3.1).

| Digits | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $u_0$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $u_\mu$ | 1 | 0 | 7916 | 6769 | 8113 | 7234 | 4142 | 5015 | 3567 | 1526 |
| $u_{2\mu}$ | 1 | 0 | 7640 | 5347 | 2291 | 4799 | 945 | 6715 | 5714 | 914 |
| $u_{3\mu}$ | 1 | 0 | 7364 | 3925 | 7109 | 884 | 2888 | 4164 | 3748 | 1899 |
| $u_{4\mu}$ | 1 | 0 | 7088 | 2503 | 6183 | 3683 | 6066 | 4620 | 7519 | 1298 |
| $u_{5\mu}$ | 1 | 0 | 6812 | 1081 | 7705 | 5003 | 6576 | 7149 | 6654 | 1302 |
| $u_{6\mu}$ | 1 | 0 | 6536 | 7851 | 3482 | 4845 | 514 | 2625 | 3325 | 1280 |
| $u_{7\mu}$ | 1 | 0 | 6260 | 6429 | 1708 | 3208 | 360 | 6496 | 4053 | 1876 |
| $u_{8\mu}$ | 1 | 0 | 5984 | 5007 | 2382 | 92 | 2210 | 1444 | 3331 | 915 |
| $u_{9\mu}$ | 1 | 0 | 5708 | 3585 | 5504 | 3689 | 2159 | 2919 | 0 | 1593 |

it is advisable to compute the initial values of the subsequences for the lf-generator on every processor by formula (3.2) for any new independent 'sample trajectory' of the process simulated, i.e. for a separate independent random test.

The routine 'multiplier' below is designed for computations by formulae (2.3) and (3.2). It allows us to obtain both the values of the multipliers $M_m$ for the auxiliary generator ensuring 'jumps' and the initial values of the subsequences for the bf- and lf-generators. The procedure has four parameters: 'm0', 'm', and 'res' are one-dimensional arrays of size 10, the parameter 'power' is an integer. The parameters 'm0', 'm', and 'res' are also the 'digits' of the corresponding numbers to base $2^{13}$, which are written in the reverse order.

To obtain the multiplier $M_m$ we must set the parameters: 'm0' in 1, 'in' in $M$, 'power' in $m$. If the value of $m$ is too large, it is advisable to break it up into multipliers and invoke the routine 'multiplier' iteratively as many times as necessary in accordance with the expression

$$M^m = M^{m_1 \cdot m_2 \cdot \ldots \cdot m_s} = (((M^{m_1})^{m_2}) \ldots)^{m_s}.$$

In each iteration the parameter 'm0' is replaced by unity, the parameter 'm' by the obtained value of the parameter 'res', and the parameter 'power' by the required value of the exponent. Having performed all the iterations we obtain the value of the multiplier $M_m$ in the parameter 'res'.

In order to obtain the initial values of the subsequences for the bf- and lf- generators, at first we must calculate the corresponding multiplier $M_m$ and pass it to the parameter 'm', the value of $u_{j+(l-1)m}$ to the parameter 'm0'; the parameter 'power' must be set in 1. Upon invoking the procedure the parameter 'res' corresponds to a successive value of $u_{j+lm}$.

```
subroutine multiplier( m0, m, power, res )
implicit integer*4 (i-n)
integer*4 m0(10), m(10),  res(10), power
integer*4 c(10,10), d(10)
data mbase /8192/, mrem  /2048/
do i = 1, 10
```