島根大学教育学部紀要（自然科学）第30巻　1頁～9頁　平成8年12月

Implementation and Evaluation of Parallel Processing Methods for Numerical Multiple Integrals

Makoto FUKUSHIMA

Summary

A subroutine program for the parallel processing of numerical multiple integrals is implemented and its performance evaluation is carried out with a transputer-system. The subroutine program where GLP(Good Lattice Points) method is adopted for numerical integration is written by using 3L Parallel FORTRAN. Three programming methods for the parallel processing are considered and discussed in this paper for obtaining an effective subroutine program.

Keywords : Multiple Integral, Good Lattice Points, Transputer.

1. Introduction

Monte Carlo method[1] based on pseudorandom numbers has usually been used for the numerical calculation of over one-dimensional integral. However, good convergence in the numerical integration cannot be obtained by the Monte Carlo method. Good Lattice Points(GLP) method[2] based on quasirandom numbers has more effective convergence in the numerical integration than that of the Monte Carlo method. The examples of effective good lattice points for up to four-dimensional integral have been reported[3].

In this paper, we implement and evaluate a subroutine program for two-, three- and four-dimensional integrals by the GLP method using a parallel processing system with six transputers[4].

2. Parallel processing for GLP method

2.1 Parallel GLP method

We discuss the parallel programming for the GLP method in this section. A multiple integral is defined by

＊ Department of Technology, Faculty of Education, Shimane University

Table 1. Good lattice points for three- and four-dimensional integrals.[3] ($g_1=1$)

N	g_2	g_3
185	26	64
266	27	69
418	90	130
597	63	169
828	285	358
1010	140	237
1459	256	373
1958	202	696
2440	638	1002
3237	456	1107
4044	400	1054
5037	580	1997

N	g_2	g_3	g_4
1142	150	187	274
3001	174	266	1269
6007	1351	5080	3086
10007	1206	3421	2842
28117	17549	1900	24455
57091	52590	48787	38790

$$\int_{I_s} f(x)dx \tag{1}$$

where I_s is the S-dimensional unit hyper-cube. Expression (1) can be approximated by the GLP method as follows:

$$\frac{1}{N}\sum_{k=0}^{N-1} f(\{k\frac{g_1}{N}\},\cdots\cdots,\{k\frac{g_S}{N}\}). \tag{2}$$

Here, N and g_S are the specific natural numbers; $\{kg_S/N\}$ denotes the decimal of kg_S/N. The values of N and g_S are given for the two-dimensional integral ($s=2$) by using Fibonacci number F_n as follows:

$$N = F_n \tag{3}$$

and

$$\{g_1,g_2\} = \{1,F_{n-1}\}. \tag{4}$$

For three- and four-dimensional integrals, the combinations of good lattice points of N and g_S ($s=2,3,4$) have also been given in Table 1.[3] If $N\rightarrow\infty$ in expression (2), the result of the numerical integration usually becomes more accurate. However, computing time increases with N. In the case of the two-dimensional integral, the criterion of the convergence in the numerical calculation is given by

$$\left|\frac{I_n - I_{n-1}}{I_n}\right| < \varepsilon \tag{5}$$

where ε is relative accuracy and I_n is the computing result for $N=F_n$. The parallel programming of the GLP method can be implemented by a processor farm using the *flood-fill* method of Parallel

FORTRAN.[5] This method provides one *master task*[5] which divides the calculations into small and independent pieces as the packets which include the parameters for the divided calculations, such as the lower and upper bounds of integrals, and *worker tasks*[5] which are distributed to all transputers in a network for processing the divided calculations when they receive the packets. After the *worker tasks* finish the calculations, they send the results as packets back to the *master task*. If all the *worker tasks* are busy, the *master task* waits until a *worker task* becomes idle. This programming method (*Method 1*) can be described by

$$\frac{1}{N}\sum_{k=0}^{N-1} f(\boldsymbol{a}) = \frac{1}{N}\{\sum_{k=0}^{[j]} f(\boldsymbol{a}) + \sum_{k=[j]+1}^{2[j]} f(\boldsymbol{a}) + \cdots\cdots + \sum_{k=(P-1)[j]+1}^{P[j]} f(\boldsymbol{a})\} \tag{6}$$

with

$$\boldsymbol{a} = (\{k\frac{g_1}{N}\},\cdots\cdots,\{k\frac{g_S}{N}\}) \tag{7}$$

and

$$[j] = [(N-1)/P], \tag{8}$$

where [ ] denotes Gauss' notation and P is the number of the packet of the *flood-fill* method. if $MOD(N\text{-}1,P) \neq 0$ where $MOD(N\text{-}1,P)$ is the remainder of $(N\text{-}1)/P$, then we add an extra packet which has the iteration number of $MOD(N\text{-}1,P)$. From the results of the actual performance where we have investigated the effect of the numbers of packet on the approximation of equation (9) with a transputer network shown in Fig.3, we have decided the number of the packet as follows: $P=3$ for $N<1000$ or $P=16$ for $1000 \leq N$.

2.2 Parallel processing of multiple integral subroutines

When we have to repeat the integration with varying the bounds of integral, the parallel processing for the multiple integral can also be carried out with the parallel processing of the integral subroutines, where the numerical calculation is carried out by the sequential GLP method which is the basic GLP method on a uniprocessor. The parallel processing of the sequential GLP method(*Method 2*) by which each *worker task* processes a multiple integral when they receive the upper and lower bounds of the integral as a packet, is effective for the iterative calculations of the multiple integral, because the *flood-fill* method becomes more effective with increasing the CPU time of the integration. Furthermore, the parallel programming of *Method 2* is more simple than that of *Method 1*, because the program for the sequential GLP method has already been provided. However, this *Method 2* includes an inefficient process, such as while some of the transputers work with receiving the packets, the rest of them are idle and ineffective. This inefficient process arises from the variations in the CPU time of each integral and the imbalance of the numbers between the transputers and packets such as *MOD(P,* *T*) $\neq$ *0* and *P*>*T*, where *T* is the number of the transputers. It becomes important, especially when we have to repeat the integration with the several different sets of the parameters of the integrand. To avoid this, we adopt the following method (*Method 3*): 1) All the parameters, the upper and lower bounds which we need in the integral subroutine program are calculated and provided from a data file in the *master task*. Then they are passed to a packet-sending subroutine as arguments. 2) All the packets which include the parameters and bounds are send to the *worker tasks* on the transputers by the packet-sending subroutine of the *master task*. 3) The each *worker task* receives a packet and calculates one multiple integral by the sequential GLP method. Then they send the obtained result as a packet back to the packet-receiving subroutine of the *master task*. This process is repeated until all the packets are processed. The processes 2) and 3) are performed in parallel.

3. Performance results of parallel processing

We evaluate the performance of the parallel processing by the above three methods. The integral used in the calculation is given by[6]

$$i_e(t) = \frac{1}{l_d}\int_0^{l_d}\int_0^{t} I(t')f(x,t-t')dt'dx \tag{9}$$

where

$$I(t) = -I_{ij}\cos(\omega t) \tag{10}$$

and

$$f(x,t) = v(t)g(x,t) \tag{11}$$

with

$$v(t) = v_0 exp(-t/\tau)+v_s, \tag{12}$$

$$g(x,t) = \frac{1}{\sqrt{2\pi S(t)}}exp\left(\frac{-[x-M(t)]^2}{2S(t)}\right), \tag{13}$$

$$M(t) = \int_0^t v(t')dt \tag{14}$$

and

$$S(t) = 30t. \tag{15}$$

Table 2. Values of the parameters and the constants used in the calculation.

(a) Parameters

I_{ij}	l_d	ω
6.090×10^{-3}	96.0×10^{-7}	$2\pi\times300.0\times10^{9}$
1.595×10^{-2}	110.0×10^{-7}	$2\pi\times200.0\times10^{9}$
0.11455	181.0×10^{-7}	$2\pi\times 90.0\times10^{9}$
0.30160	246.0×10^{-7}	$2\pi\times 60.0\times10^{9}$

(b) Constants

v_o	v_s	τ
6.4×10^{7}	6.0×10^{6}	8.0×10^{-14}

Here, $g(x,t)$ is a Gaussian function; I_{ij}, w and l_d are parameters; v_o, τ and v_s are constants. The values of the parameters and the constants are given in Table 2. We repeat the integration of eq.(9) with increasing the upper bound t of the inner integral from $2\pi/\omega$ to $4\pi/\omega$ with an incremental value $2\pi/(21*\omega)$, where the constant value 21 is chosen from the efficiency of the further processing of the result $i_e(t)$ with a FFT program. Moreover, we repeat the integration four times with the four different sets of the parameters used in the integrand. Therefore, eighty-four results are obtained from the above integration of eq.(9). The block diagrams of the parallel processing by *Method 2* and *Method 3* are shown in Fig.1 and Fig.2, respectively.

The transputer-system used for the parallel processing is shown in Fig.3, where six T-800 transputers are used at 20 MHz clock cycle with three processor-cycle-time and 1 MB memory. The performance results in execution-time by the three methods for relative accuracy 10^{-3}, 10^{-4} and 10^{-5} are shown in Table 3, with those by a sequential GLP method[7] using one transputer. We have measured the execution-time with *f77_timer_now* routine[5] of Parallel FORTRAN. From the results, it is found that the most effective method is *Method 3* by which we obtain about 4 to 5 times smaller execution-time than that of the sequential GLP method using one transputer.

4. Conclusions

In the iterative calculations of the multiple integrals with varying the parameters of the integrand, it is found that *Methods 2* and *3*, the parallel processing of the integral subroutines by the sequential GLP method, are more effective than the parallel GLP method (*Method 1*). However, *Method 2* has an inefficient process where some of the transputers are unloaded and ineffective. Therefore, the most effective method is *Method 3*. The advantage of *Method 2* is simple programming and less memory consumption when the CPU time of every integral is almost same.

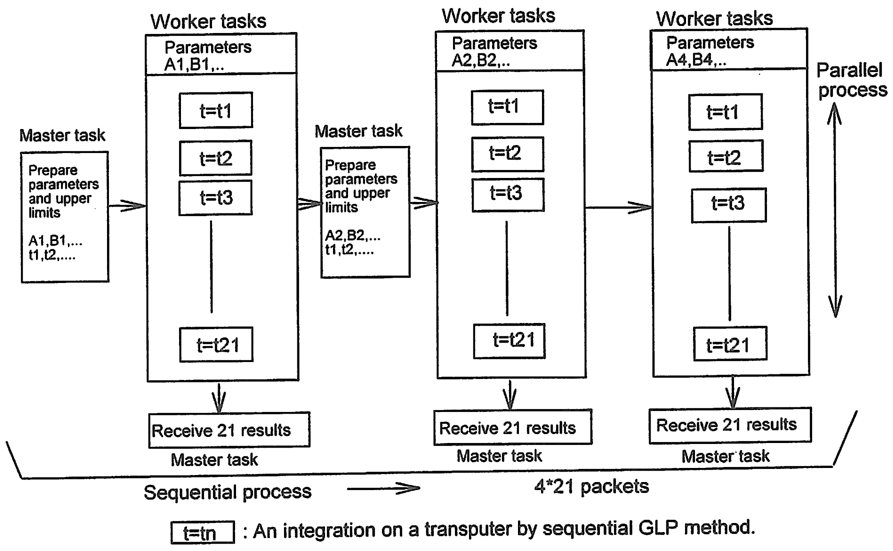

Fig.1 Parallel processing of eq. (9) by *Method 2*, where $t_1, t_2, \ldots$ and t_{21} are the upper limits of the inner integral. Master task loaded on a root transputer prepares the parameters and upper limits, and provides the packets for the *worker tasks* loaded on every transputer.

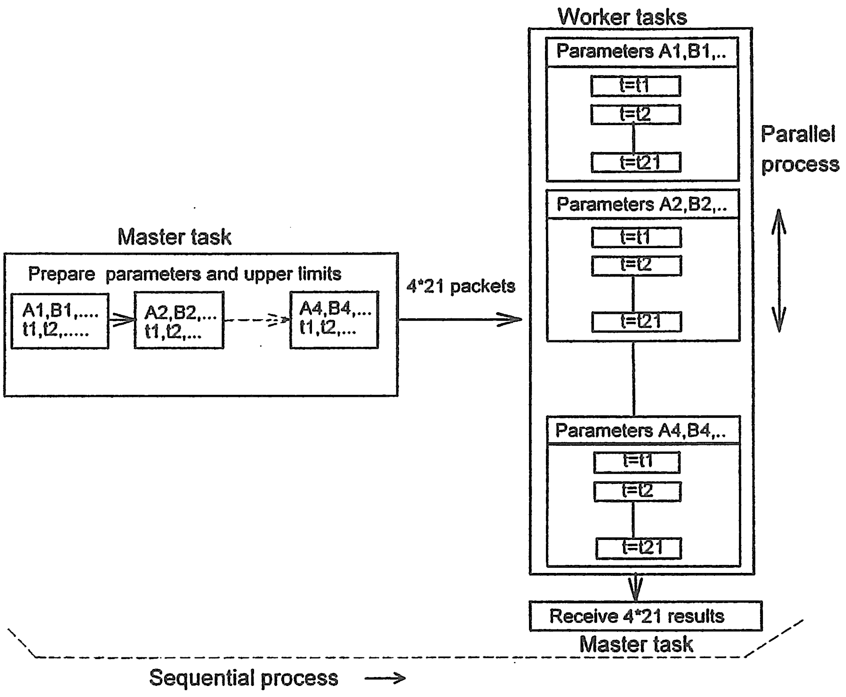

Fig.2 Parallel processing of eq.(9) by *Method 3*, where all the packets are send out by calling the packet-sending subroutine of the *master task.*

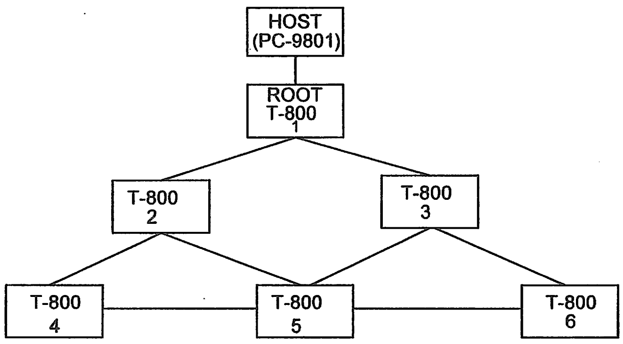

Fig.3 Parallel processing system using six T-800 transputers.

Table 3. Performance results in execution-time. Time unit is second.

Relative Accuracy	*Method 1*	*Method 2*	*Method 3*	Sequential GLP method using one transputer
10^{-3}	57.178	54.047	35.796	163.368
10^{-4}	96.571	79.220	66.508	305.993
10^{-5}	120.623	109.281	91.122	396.922

If the iterative calculations of multiple integrals are not required, the choice of *Method 1* gives about 3 times smaller execution-time than the sequential GLP method using one transputer, even though the parallel programming for *Method 1* is not so simple as *Method 2*. An example program by *Method 3* is shown in Appendix.

References

[1] H.Engels: Numerical quadrature and cubature. Academic Press, New York, 1980.

[2] E.Hlawka: Zur angenäerten berechung mehrfacher integrale. Monatsh. Math., **66**, 140-151, 1962.

[3] L.K.Hua and Y.Wang: Applications of number theory to numerical analysis. Springer, Beijing, 1981.

[4] The Transputer Databook. INMOS Ltd., 1988.

[5] Parallel Fortran. 3L Ltd., 1990.

[6] I.Song and D-S.Pang: Analysis and simulation of the quantum well injection transit time diode. IEEE Trans.,ED, **35**, 12, pp.2315-2322, 1988.

[7] M.Mori: FORTRAN77 numerical calculation programming. The Iwanami Computer Science Series, 1987.

Appendix

An example of the multiple integral by *Method 3* is presented.

1) **Purpose**: Compute a two-, three- or four-dimensional integral as follows:

$$\int_0^b\int_0^d\int_0^f\int_0^h f(x,y,w,z)\,dxdydwdz. \tag{A1}$$

2) **Usage**:

CALL MULTIN(MD,B,D,F,H,EPS,RESULT,ERREST,NT,PACKETS,PAR,NP)

3) **Arguments**:

MD -	Dimension of the integral. (Integer, Input) (*MD*=2, 3 or 4)
B, D, F,H-	Array of length *PACKETS* containing upper limits of the integral. (Real, Input)
EPS-	Relative accuracy desired. (Real, Input)
RESULT-	Array of length *PACKETS* containing computed results of the integral. (Real, Output)
ERREST-	Estimate of the absolute value of the error. (Real, Output)
NT-	Array of length *PACKETS* containing total iteration number in an integral routine. (Integer, Output)
PACKETS-	Number of the packets for *flood-fill* method. The number is equal to the iteration number of the integration. (Integer, Input)
PAR-	Two-dimensional array of length *PACKETS*NP* containing the values of the parameters of the integrand. These parameters are available in the user supplied function func such as *common par*, where *par(1)=NP,par(2)=parameter1*, *par(3)=parameter2*, ... and *par(10)=parameter9*. However, the length of *common par* must be 20 for including workspace. (Real, Input)
NP-	Number of the parameters of the integrand. *NP* must satisfy $1 \le NP \le 9$. (Integer, Input)

4) **Example**

In this example, we calculate the following integral 30 times with increasing the upper bound *d* of the inner integral from 1 to 10 for three values of the parameter *p*.

$$\int_0^1\int_0^d pxy\,dxdy \quad (p = 1,2 \text{ or } 4) \tag{A2}$$

Program List

```
        program master
c
c       sample program for GLP
c
        real*8 result(30),b(30),d(30),f(30),h(30)
        real*8 eps,errest(30),ev(3),par(30,1)
        integer packets,md,ij,ik,indx,nt(30)
c
        md=2
c
        ev(1)=0.25d0
        ev(2)=0.5d0
        ev(3)=1.0d0
c
        packets=30
        eps=1.0d-3
c
        do 100 ik=1,3
        do 200 ij=1,10
        indx=ij+10*(ik-1)
        if(ik.eq.1) then
                par(indx,1)=1.0d0
        else if(ik.eq.2) then
                par(indx,1)=2.0d0
                else
                par(indx,1)=4.0d0
        end if
        d(indx)=real(ij)
        b(indx)=1.0d0
200     continue
100     continue
        np=1
c
        call multin(md,b,d,f,h,eps,result,errest,nt,packets,par, np)
c
        do 400 ik=1,3
        do 300 ij=1,10
        indx=ij+10*(ik-1)
        abserr=abs(result(indx)-real(ij)**2*ev(ik))
        write(6,2001)ik,ij,nt(indx),result(indx),errest(indx), abserr
 300    continue
 400    continue
 2001   format(/' glp ',
       $'   p=',i2,' d=',i2,' : nt=',i7,' result=',1pd22.15/
       $' :esterr=',d7.1,
       $' :abserr=',d7.1)
        end
        double precision function func(x,y)
c
c       sample program f2 for glp
c
        real*8 x,y,p,par(20)
        common par
        p=par(2)
c
        func=p*x*y
        return
        end
```