INFORMATICA, 1999, Vol. 10, No. 2, 161–170

Hyper-Rectangle Selection and Distribution Algorithm for Parallel Adaptive Numerical Integration

Raimondas ČIEGIS

Institute of Mathematics and Informatics, Akademijos 4, 2600 Vilnius, Lithuania
Vilnius Gediminas Technical University, Saulėtekio 11, 2054 Vilnius, Lithuania
e-mail: rc@fm.vtu.lt

Ramūnas ŠABLINSKAS

Vytautas Magnus University, Vileikos 8, 3035 Kaunas, Lithuania,
Vilnius Gediminas Technical University, Saulėtekio 11, 2054 Vilnius, Lithuania
e-mail: ramas@omnitel.net

Received: February 1999

Abstract. In this paper we consider parallel numerical integration algorithms for multi-dimensional integrals. A new hyper-rectangle selection strategy is proposed for the implementation of globally adaptive parallel quadrature algorithms. The well known master-slave parallel algorithm prototype is used for the realization of the algorithm. Numerical results on the SP2 computer and on a cluster of workstations are reported. A test problem where the integrand function has a strong corner singularity is investigated. A modified parallel integration algorithm is proposed in which a list of subproblems is distributed among slave processors.

Key words: parallel adaptive integration, distributed-memory parallel computers, load-balancing, redistribution of tasks.

1. Introduction

The problem considered in this paper is the numerical approximation of the multi-dimensional integral

$$I(f,\Omega) = \int_{\Omega} f(x)\,dx \tag{1}$$

to some given accuracy ε, where $\Omega = [a_1, b_1] \times [a_2, b_2] \cdots [a_n, b_n]$ is the range of integration and $f(x)$ is the integrand function. We are interested in implementing globally adaptive integration algorithms on distributed memory parallel computers. The algorithms of this paper are targeted at clusters of workstations with standard message passing interface. We use PVM in our numerical experiments. The important features of such parallel

computers are heterogeneity of the cluster and unfavorable ratio between computation and communication rates.

Numerical integration algorithms attempt to approximate $I(f,\Omega)$ by the sum

$$S_N(f) = \sum_{i=1}^{N} w_i f(x_i). \tag{2}$$

The numbers w_i are called *weights*, and the x_i are called *knots*. Efficient numerical algorithms are adaptive and some strategy for selection of hyper-rectangles for further subdivision is included into the algorithm. We use the cubature rule pair, which was proposed by Genz and Malik (1980). Parallel adaptive algorithms were investigated in many papers (Bull and Freeman, 1995; Keister, 1996; Miller and Davis, 1992). Mostly these algorithms are targeted at shared memory computers.

In our previous paper (see Čiegis *et al.* (1997)) we have investigated various algorithms for static subdivision of the whole task. These smaller subtasks are solved in parallel by different processors. The main result of Čiegis *et al.* (1997) is that parallel static subdivision algorithms are efficient if numbers of subdivisions required within the different subregions vary only moderately. The situation is different when the integrand function f has strong singularities. The load balancing of the static subdivision – dynamic distribution algorithm becomes very poor and the quality of the parallel adaptive algorithm also degrades seriously.

In this paper we develop a new hyper-rectangle selection algorithm. The quality of the parallel adaptive cubature method is improved considerably, but additional costs of communications are also increased since a more fine-grain subdivision of the integration region Ω is implemented. This parallel algorithm attempts to calculate the multidimensional integral faster when we use p processors. The second important goal in parallel computing is to solve a larger problem. In our case the more accurate approximation of the integral requires to use a larger list of subproblems. Hence we propose a parallel integration algorithm in which the list of subproblems is distributed among processors.

We summarize the remainder of this paper. In Section 2 we review the Static Subdivision – Dynamic Distribution (SSDD) algorithm. The analysis is given for the case when the integrand function f has a strong corner singularity. Section 3 describes the new hyper-rectangle selection algorithm. Numerical results for the SP2 computer and for a cluster of workstations are presented in Section 4. The new master–slave algorithm with the distributed list of tasks is described in Section 5.

2. Static Subdivision – Dynamic Distribution Algorithm

We describe very briefly the parallel algorithm for multi-dimensional cubature that is based on a static initial subdivision of the integration region Ω. Such algorithms are investigated in (Čiegis *et al.*, 1997).

We use the well known parallel algorithm prototype: master–slave algorithm. The pool of tasks $P_j, j = 1, 2, \ldots, M$ is formed from subproblems $I(f,\Omega_j)$, where we make

partitioning of the region $\Omega = \Omega_1 \bigcup \Omega_2 \bigcup \cdots \bigcup \Omega_M$. The error tolerance ε_j for each subregion is chosen proportional to the volume of the subregion. The master process dynamically redistributes unsolved subtasks to slaves and collects the results. Slave processes calculate the approximation of the integral over the given subregion and return the result to the master process.

REMARK 1. At the end of computations the same unsolved subproblems are sent to several different slaves which finished their main tasks. This modification is important for heterogeneous clusters of workstations since it reduces the dependence of the computation time on the slow processors.

Numerical results given in (Čiegis *et al.*, 1997) lead to a conclusion that the SSDD algorithm is efficient for integrals with the integrand function f which has no strong singularities. It is important to note that the SSDD method is well adapted to the variations of workload on workstations during computations.

Now we investigate the SSDD algorithm in the case when the integrand function f has strong singularities. Let consider the following test problem (Bull and Freeman, 1995)

$$\int_0^1 \int_0^1 \int_0^1 (x_1 x_2 x_3)^{-0.9}\, dx_1\, dx_2\, dx_3, \tag{3}$$

where the integrand function has a strong corner singularity.

We compute a numerical approximation of the integral (3) to the relative accuracy $\varepsilon = 0.003$. Table 1 summarizes the performance of the SSDD method. Here M denotes the number of subregions Ω_j, N_M is the total number of required subdivisions

$$N_M = n_1 + n_2 + \cdots + n_M, \tag{4}$$

where n_j is the number of subdivisions required for the numerical approximation of the integral $I(f, \Omega_j)$, Q is the quality of the algorithm, which is defined as

$$Q = \frac{N_M}{N_1}, \tag{5}$$

Table 1
Numerical results for the SSDD algorithm

M	Number of subdivisions N_M	Quality Q	Load balancing L
2	163249	1.524	0.98
4	213123	1.990	0.94
8	255345	2.384	0.91

In order to investigate the load balancing property of the SSDD algorithm we define the parameter L:

$$L = \frac{\max_j \ n_j}{N_M}. \tag{6}$$

From the Table 1 we see that the quality of the parallel adaptive integration algorithm decreases when M becomes larger. The load balancing of this algorithm is also bad since only one process is involved in useful computations.

3. Hyper-Rectangle Selection Strategy

The numerical integration problem can be considered as a dynamic resource problem. If our parallel computer is a cluster of multi-user workstations then the available processing capacity per node may change during computations, hence we have a dynamic resource computer. The load balancing problem can be considered as the mapping of a dynamic resource problem onto a dynamic resource computer. The algorithm must include some mechanism for the migration of subproblems from over-loaded processes to under-loaded processes at run-time.

We propose the following hyper-rectangle selection algorithm.

Hyper-Rectangle Selection Algorithm (HRS)

The *master* part of the algorithm is given by the following code:

1. Form the task-list of hyper-rectangles $P_j,\ j = 1, 2, \ldots, s$,
   ranked by error estimates so that $\varepsilon_1 < \varepsilon_2 < \ldots < \varepsilon_s$.
2. Send the initial data portions to slaves.
3. `do while` (ε is greater than required accuracy ε_0)
   receive from a slave the ranked list of R_{recv} hyper-rectangles;
   put these results to the main ranked list of hyper-rectangles P;
   update the integral approximation and error estimates ε;
   send to the slave R_{send} hyper-rectangles with the largest errors.
   `end do`

The *slave* part of the algorithm is given by the following code:

1. Receive a job list from the master process.
2. Apply the adaptive numerical integration algorithm and
   make $R_{recv} - R_{send}$ subdivisions. We use $R_{recv} = kR_{send}$, $k > 2$.
3. Rank the obtained list of hyper-rectangles according
   to their error estimates and send the result to the master process.

The algorithm contains two free parameters R_{send} and R_{recv}. The optimal values of these parameters depend on the dimension of the integral and on the ratio between computation and communication rates of a parallel computer.

REMARK 2. The *quick-sort* method is used by slave processes to rank the list of subproblems.

4. Numerical Results

In this section we investigate the performance of the proposed hyper-rectangle selection algorithm. We calculate the multi-dimensional integral (3) to the relative accuracy $\varepsilon = 0.002$. The SP2 computer and a cluster of RS6000 workstations were used in computations.

We define the speed-up S_p as

$$S_p = \frac{T_s}{T_p}, \tag{7}$$

where T_p is the time used to solve the given problem on p processors, T_s represents the time for the sequential globally adaptive algorithm.

Table 2 presents the numbers of subdivisions N_p, timings T_p in seconds, speed-ups S_p and the efficiency E_p on the SP2 computer. We set the following values of parameters $R_{send} = 200$, $R_{recv} = 700$.

Table 3 gives analogous information for the cluster of workstations RS6000. We wish to test the quality of the proposed algorithm, therefore we choose to add appropriate delays in the evaluation of integrand function. A dummy loop is incorporated into the code and it is repeated 5 times.

From the the results given in Table 3 we see that the quality of the proposed algorithm degrades only slightly when the number of processors is increased. Next we investigated the sensitivity of the algorithm to the selection of parameters R_{send} and R_{recv}. Numerical experiments show that the algorithm is not very sensitive to the selection of these parameters and the time of computations varies by about ten percents for a large range of values of R_{send} and R_{recv}. Table 4 presents the dependency of computation time and computation efficiency from parameters R_{send} and R_{recv}. The computations were carried out on the cluster of five RS6000 workstations. A dummy loop is again incorporated into the code.

Table 2

Numerical results on the SP2 computer

p	Number of subdiv. N_p	Time T_p	Speed-up S_p	Efficiency E_p
1	217698	77.9	0.957	0.957
2	217897	39.8	1.874	0.937
4	219295	20.8	3.586	0.896
7	220392	12.7	5.873	0.839
15	226984	7.4	10.080	0.672

Table 3
Numerical results on the cluster of workstations

p	Number of subdiv. N_p	Time T_p	Speed-up S_p	Efficiency E_p
1	217698	313.3	0.945	0.945
2	217897	167.4	1.768	0.884
3	218596	117.7	2.515	0.838
4	219295	92.0	3.217	0.804
5	219494	74.5	3.973	0.795
6	220193	64.0	4.625	0.771
7	220392	57.8	5.121	0.732

Table 4
The influence of R_{send} and R_{recv} to the computation time T_p

R_{send}	R_{recv}	Number of subdivisions N_p	Time T_p	Speed-up S_p
10	20	217204	281.7	1.051
10	100	218924	77.2	3.835
50	300	218744	72.6	4.078
50	700	223194	70.8	4.182
200	700	219494	74.5	3.973
400	2000	224394	74.7	3.963

As introduced in (Freeman and Phillips, 1997), the time of point-to-point communication between two processes can be estimated as

$$T_{send} = \alpha + \beta n, \tag{8}$$

where α is the latency (or start-up time) and β is the incremental time per data items moved; its reciprocal is called bandwidth. The values of these parameters are $\alpha = 2ms$, $\beta = 1s/MB$ for the public domain version of PVM using Ethernet, while PVMe on SP2 achieves $\alpha = 0.1ms$, $\beta = 0.03s/MB$.

Table 5 presents the results of experiments carried out on the cluster of four RS6000 workstations when no additional delay in integral function is used. We still obtain a speed-up of the parallel algorithm.

We also investigated the efficiency of the parallel integration algorithm when the difficulty of the problem was continuously increased. Table 6 gives the results obtained on the SP2 computer with 16 processors for different values of the accuracy parameter ε. We note that only one process per node can be run on SP2, hence one master process and 15 slave processes were used in computations.

Table 5
Results of the experiments without additional delay

R_{send}	R_{recv}	Number of subdivisions N_p	Time T_p	Speed-up S_p
10	50	217555	49.6	1.45
100	300	217795	35.4	2.03
100	900	221195	29.3	2.46
200	400	217795	43.9	1.64
200	1000	219995	31.3	2.30

Table 6
The influence of problem complexity to the efficiency of the computations

ε	Number of subdivisions N_p	Time T_p	Speed-up S_p
2.0	226984	7.4	10.080
1.0	332484	11.0	10.855
0.5	503984	15.3	11.774

5. The Parallel Algorithm with Distributed List of Tasks

So far we have been interested in the solution of the integration problem as fast as it was possible. The parallel algorithms also enable us to solve larger problems as well. The HRS algorithm was organized in such a way, that the task list was maintained by the master process. Therefore the available master host memory defines the size of the task list. The task list size determines the maximum accuracy ε we can reach. The HRS algorithm does not utilize the memory of the slave processors. We propose a modified algorithm which utilizes the slave memory. Hence we can solve the larger problem than it is possible to solve with one computer. This is our primary goal. At the same time we demand that the quality and efficiency of the new algorithm must not degrade dramatically.

The Algorithm with Distributed List of Tasks (DLT)

The *master* part of the algorithm is given by the following code:

1. Get the information from slaves about the maximal sizes of the local job lists $M_k, k = 1, 2, \ldots, p$.
2. Form the initial task-list of hyper-rectangles $P_j,\ j = 1, 2, \ldots, s$, ranked by error estimates so that $\varepsilon_1 < \varepsilon_2 < \ldots < \varepsilon_s$.
3. Send the initial data portions to slaves.

4. `do while` (ε is greater than required accuracy ε_0)
   `if` (the master has a sufficient number of free memory blocks)
   `a)` receive from the j-th slave the ranked list of R_{recv} hyper-rectangles, the integral approximation value S_j and the estimated error of all hyper-rectangles kept in its local list err_j,
   `b)` put the results into the main ranked list P,
   `c)` update the integral approximation S and error estimate ε,
   `d)` send to the j-th slave R_{send} hyper-rectangles with the largest errors,
   `e)` update the S_j and err_j;
   `else`
   `f)` find a set of slaves J which have free memory blocks: $J = \{j | M_j > 0\}$,
   `g)` define the numbers of tasks $R_{distr}(j), j \in J$ that must be distributed among the slaves,
   `h)` `for` ($j \in J$)
   implement steps `a,b` and `c`,
   send to the j-th slave R_{send} hyper-rectangles with the largest errors and $R_{distr}(j)$ hyper-rectangles with the smallest errors,
   update the S_j, err_j and M_j.
   `end for`
   `end if`
`end do`

The *slave* part of the algorithm is given by the following code:

1. Receive a job list from the master process.
2. Join the received job list with the local job list.
3. Take $2R_{recv} - R_{send}$ jobs with the largest error estimates from the local job list and the apply quadrature $R_{recv} - R_{send}$ times.
4. Send the R_{recv} jobs with the largest errors to the master process.
5. Put the remaining jobs from the result list into the local job list.

We can observe, that the DLT algorithm is very similar to HRS algorithm. If we assume that the size of slave's local list of tasks is equal to zero, we will have the HRS algorithm. On the other hand, if memory allocation condition in the Step 3 of the master algorithm does not fail, we will also have the HRS algorithm. This is likely to appear when the master process has a sufficient amount of memory to achieve the desired accuracy.

References

Bull, J.M., and T.L. Freeman (1995). Parallel globally adaptive algorithms for multi-dimensional integration. *Applied Numerical Mathematics*, **19**, 3–16.

Freeman, T.L., and C. Phillips (1991). *Parallel Numerical Algorithms*. Prentice Hall, New York, London, Toronto, Sydney, Tokyo, Singapore.

Čiegis, R., R. Šablinskas and R., Waśniewski (1997). Numerical integration on distributed memory parallel systems. In M. Bubak, J. Dongarra, J. Waśniewski (Eds.), *Recent Advances in PVM and MPI. Lecture Notes in Computer Science*, Vol. 1332. Springer-Verlag, Berlin Heidelberg New York, 329–336.

Genz, A.C., and A.A. Malik (1980). Remarks on algorithm 006: an adaptive algorithm for numerical integration over an N-dimensional rectangular region. *J. Comput. Appl. Math.*, **6**, 295–302.

Keister, B.D. (1996). Multi-dimensional quadrature algorithms. *Computers in Physics*, **10**, 119–122.

Miller, V.A., and G.J. Davis (1992). Adaptive quadrature on a message passing multiprocessor. *J. Parallel Distributed Comput.*, **14**, 417–425.

R. Čiegis has graduated from the Vilnius University (Faculty of Mathematics) in 1982, received the degree of Doctor of Physical and Mathematical Sciences from the Institute of Mathematics of Byelorussian Academy of Sciences in 1985 and degree of Habil. Doctor of Mathematics from the Institute of Mathematics and Informatics, Vilnius in 1993. He is a senior researcher at the Numerical Analysis Department, Institute of Mathematics and Informatics. R.Čiegis is also a head of Mathematical Modeling Department of Vilnius Gediminas Technical University. His research interests include numerical methods for nonlinear PDE, parallel numerical methods and numerical modeling in nonlinear optics, biophysics, ecology.

R. Šablinskas was born in 1971. After receiving his master's degree in VMU he has been admitted as an network administrator in telecommunications company Omnitel. In 1995 he has been admitted as a PhD student in Kaunas Vytautas Magnus University. His research interests cover distributed and parallel computing, optimization, neural network models.

Adaptyviojo skaitinio integravimo metodo užduočių parinkimo ir paskirstymo algoritmas

Raimondas ČIEGIS, Ramūnas ŠABLINSKAS

Šiame darbe nagrinėjami skaitiniai adaptyvūs integravimo algoritmai daugiamačiams integralams skaičiuoti. Šie algoritmai skirti lygiagretiesiems kompiuteriams su paskirstytaja atmintimi arba virtualiesiems lygiagretiesiems kompiuteriams, sudarytiems iš grupės kompiuterinių stočių. Pateiktas naujas užduočių parinkimo algoritmas, leidžiantis geriau išbalansuoti darbą tarp procesorių.

Skaičiavimuose naudotos PVM ir MPI bibliotekos. Pateikti skaičiavimo eksperimento rezultatai. Sudaryta algoritmo modifikacija, kurioje užduočių sąrašas paskirstomas tarp procesorių, juo išsprendžiame didesnį uždavinį nei nuosekliu algoritmu.