PROCEEDINGS OF THE SIXTH SIAM CONFERENCE ON

PARALLEL PROCESSING FOR SCIENTIFIC COMPUTING

Volume II

Edited by Richard F. Sincovec
Oak Ridge National Laboratory
David E. Keyes
Yale University
Michael R. Leuze
Oak Ridge National Laboratory
Linda R. Petzold
University of Minnesota
Daniel A. Reed
University of Illinois

siam Philadelphia
Society for Industrial and Applied Mathematics

A Scalable Parallel Algorithm for the Adaptive Multidimensional Quadrature*

Marco Lapegna† Alessandra D'Alessio†

Abstract

A global adaptive algorithm for the multidimensional quadrature on a MIMD distributed memory multiprocessors is presented. In this algorithm the communications are performed only among directly connected nodes, so that without global communications, the scalability of the algorithm is preserved and a good load balancing is obtained.

1. Introduction

The numerical computation of multidimensional integrals is required in many application areas like quantum chemistry, high energy physics and statistics. However, until a few years ago, the development of efficient and reliable routines for multidimensional quadrature was not possible because of the high computational cost. Today, the availability of powerful parallel machines makes possible such a work, by exploiting the intrinsic parallelism of the problem [6]. However, adaptive strategies require some communication steps among the processors, and in order to fully exploit the computational power available, it is necessary to minimize as more as possible such a communication. In the present work we describe a global adaptive algorithm developed for the computation of a single multidimensional integral on a MIMD distributed memory multiprocessor. We pay a special attention to the problems of the scalability and the load balancing.

2. Adaptive Algorithms and Parallelism

Let us restrict our attention to the evaluation of a multidimensional integral in the form:

$$If = \int_R f(t_1, ..., t_n)\, dt_1 \cdots dt_n \tag{1}$$

where $R = [a_1, b_1] \times \cdots \times [a_n, b_n]$ is a rectangular region in the real space $\Re^n$. A global adaptive algoritm for the computation of (1) is an iterative procedure that, at each iteration, processes a partition of the integration domain R, with the aim of computing an approximation $\tilde{If}$ of If such that $Ef = |\tilde{If} - If| < \varepsilon$, where ε is an user required tolerance. To do this the algorithm computes, at each step, an approximation of If by using a fixed quadrature rule Qf in every subregion of the partition. Then this approximation is improved by subdividing the subdomains of the partition with maximum error estimates. Since all subdomains can be successively processed, in a global adaptive algorithm it is necessary an ordered data structure to store them. Since at each iteration we are interested

*This work was supported by the grant No. 91.00911.PF69 of C.N.R. (Special Project on Informatic Systems and Parallel Computing)

†Dept. Mathematics and Applications of the Univ. of Napoli "Federico II" (Italy)

only in the maximum value in the data structure, a partially ordered binary tree called *heap* can be used [7]. In a heap, the value of each node is greather or equal than the values of its subnodes, so that the root of the tree contains the maximum value of the data structure. Therefore, the basic global adaptive algorithm for the multidimensional quadrature is the following:

Algorithm 1

initialize Q *and* E
while $E > \varepsilon$ *do*
1. select the subdomain with maximum error estimate
2. subdivide it in 2 or more parts
3. compute in these parts the new approximations of Q *and of* E *4. sort the heap*
5. update the global values of Q *and* E
endwhile

For the previous algorithm Genz defines three levels of parallelization [5]:

1. Integrand level: this level of parallelization occurs when there are N integrals in the form (1) to compute, so that it is possible to assign to the P processors N/P indipendent integrals.

2. Subregion level: this level occurs in the computation of a single multidimensional integral. At this level the subdomains with maximum error estimates are splitted among all processors, and each of them computes the integral and the error approximations.

3. Basic rule level: the third level occurs when the computation of the basic rule is splitted among the processors.

It is possible to see that different kind of parallel computers achieve maximum efficiency by exploiting different levels of parallelization. For example, vector computers and SIMD computers, are suitable for a parallelization at the basic rule level because the little granularity of the computation. On the other hand, MIMD computers achieve maximum efficiency when either a parallelization at the subregion level or at integrand level is considered, because these kinds of parallelization need less communication overhead than the parallelization at the basic rule level.

3. A Scalable Parallel Algorithm

The best way to parallelize at subregion level a global adaptive algorithm is to use a copy of the Algorithm 1 in each node. At the beginning of the algorithm, the integration domain R is splitted in P subregions, of the same size, that are assigned to the P processors; then the nodes periodically self reorganize the subdivision in order to balance the workload when the integrand function shows some isolated singularity like peak or discontinuity. The self reorganization avoids to use a node as a controller, so that all nodes can be used for the computation. With this strategy it is possible to generate an heap in each processor, so that the steps 2, 3 and 4 of the main iteration cycle in the Algorithm 1 can be executed indipendently. However, for this kind of parallel algorithm, it is not easy to combine the *scalability* of the algorithm (i.e. the ability to preserve the efficiency when the number of processors increases) with the *load balancing* (i.e. the ability to equally distribute

the workload in P parts): the global communications occurring in steps 1 and 5 of the Algorithm 1 makes it poorly scalable, because the communication cost increases with the number of the processors. To improve the scalability, the proposed algorithm avoids global comunications among the processors during the search for the subdomains with largest error estimate; therefore it attempts to distribute them among the processors by only performing communications between directly connected nodes. However we note that a good load balancing is obtained if we quickly distribute the subdomains with maximum error estimate, so that none of them waste time on unimportant subdomains. We organize the nodes as a 2- dimensional periodic mesh, so that we have two directions of communication. The diameter of this topology (i.e. the maximum distance of two nodes) is $2\sqrt{P}$ when the mesh is square, then the "hard" subdomains reach all nodes after at most $2\sqrt{P}$ iteration. More precisely, our algorithm begins with a subdivision of the integration domain R among all processors. Then at the generic iteration i of the main cycle, each node attempts to send the subdomains with relatively large error estimates forward in the direction $j = mod(i, 2)$ of the mesh as showed in Figure 1.

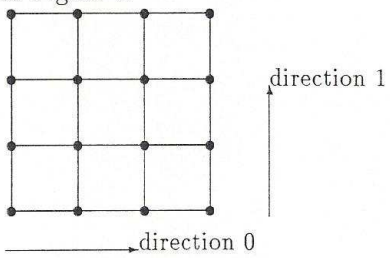

Figure 1: a 2-dimensional mesh of 16 processors

In this direction if a node has the subdomain in the heap root with an error estimate E_L such that

$$E_L > cE_N, \quad (c > 1) \tag{2}$$

where E_N is the error of the subdomain in the heap root of the next node and c is an heuristic constant, then each node sends such a subdomain to the next node, in order to balance in an efficient way the workload. In our experiments we used the test (2) with $c = 1.5$; values smaller than 1.5 improve the load balancing but deteriorate the communication overhead. The next step of the algorithm is the selection, by each node, of the subdomain with maximum error estimate in its own heap (step 1). In step 2 each of the P selected subdomains is splitted in 2 parts along that coordinate axis where the integrand function has largest fourth divided difference. This is a classical subdivision strategy [4] achieving good results in the reduction of the error estimate. Then in step 3 each node computes an approximation of the integral using a quadrature rule of degree 9 similar to the one given in [3]; further an approximation of the error is computed using a suitable strategy as in [1], in order to avoid problems like phase factor or a bad displacement of the evaluation points. Finally, after the heap sorting (step 4), all nodes check for the convergence (step 5) by using the local acceptance test for the error estimate E_{loc} in the subdomains in the generic node:

$$E_{loc} < \frac{\varepsilon V_{loc}}{V} \tag{3}$$

where V_{loc} is the volume of such subdomains and V is the volume of the whole integration domain R. The algorithm stops when all nodes pass the previous test. We remark that the

2-dimensional mesh is well suitable for several MIMD distributed memory computers based on the same topology like the Touchstone Delta System and the Symult S2010, and it is easy to map it onto a hypercupe by using the Gray code [2].

4. Numerical Tests and Conclusions

In this section we present some results in 3 and 6 dimensions by using a FORTRAN implementation of our algorithm on the test function

$$f(x_1,..,x_n) = \prod_{i=1}^{n}(\alpha^{-2} + (x_i - .3)^2)^{-1}$$

for $\alpha = .04$ and $\alpha = .36$ respectively [1]. The previous function has a peak in $(0.3,..,0.3)$, and the values for α are chosen to control the natural increase, with the dimension n, of the integrand difficulty. For the two cases we required a tolerance $\varepsilon = 3. \times 10^{-5}$ and $\varepsilon = 10^{-5}$ respectively. In Tables 1 and 2 we show the speed-up and the efficiency obtained on the Intel iPSC/860. Such results confirm our expectation about the good scalabilty and load balancing of the algorithm.

Nodes	Speed-up	Efficiency
2	1.94	0.97
4	3.86	0.96
8	7.30	0.91
16	14.10	0.88

Table 1: 3-dimensional case: $\alpha = .04$

Nodes	Speed-up	Efficiency
2	1.92	0.96
4	3.82	0.95
8	7.26	0.90
16	13.92	0.87

Table 2: 6-dimensional case: $\alpha = .36$

We note that when the number of processors P increases, the problem of a slow distribution of the subdomains among the processors might occur, therefore it might be useful to consider further generalizations in 3 and 4 dimensions for the mesh of the processors.

References

[1] J. Berntsen, *Practical error estimation in adaptive multidimensional quadrature routines*, J. of Comp. and Appl. Math., 25 (1989), pp. 327-340.

[2] G.C. Fox et al., *Solving problems on concurrent processors*, Prentice Hall (1988)

[3] A.C. Genz and A.A. Malik, *An imbedded family of symmetric numerical integration rules*, SIAM J. Numer. Anal. 20 (1983), pp. 580-588.

[4] A.C. Genz and A.A. Malik, *An adaptive algorithm for numerical integration over an N-dimensional rectangular region*, J. of Comp. and Appl. Math., 6 (1980), pp. 295-302

[5] A.C. Genz, *Parallel adaptive algorithms for multiple integrals*, in Mathematics for large scale computing (J. Diaz ed.), Dekker inc., New York (1989).

[6] M. Lapegna, *Global adaptive quadrature for the approximate computation of multidimensional integrals on distributed memory multiprocessors*, Concurrency: Practice and Experience 4 (1992), 452-466

[7] M.A. Malcom and R.B. Simpson, *Local versus global strategies for adaptive quadrature*, ACM Trans. on Math. Soft. 1 (1975), pp. 129-146