PARALLEL COMPUTING: Problems, Methods and Applications

Selection of Papers presented at the Conference on
Parallel Computing: Achievements, Problems and Prospects
Capri, Italy, 3-7 June, 1990

Edited by

PAUL MESSINA
California Institute of Technology
Pasadena, California
U.S.A.

ALMERICO MURLI
Dipartimento di Matematica e Applicazione
Università degli Studi di Napoli
Naples, Italy

1992

ELSEVIER
AMSTERDAM • LONDON • NEW YORK • TOKYO

Parallel Computing: Problems, Methods and Applications
P. Messina and A. Murli (Editors)

Adaptive Multidimensional Quadrature on a Distributed Memory Multiprocessor

Marco Lapegna

Dipartimento di Matematica ed Applicazioni
Università degli studi di Napoli "Federico II"
Via Mezzocannone 16, 80134 Napoli Italy

1 Introduction

Let $C^s = [0,1]^s$ be the s-dimensional unitary cube, and $f(\mathrm{t}) = f(t_1, \ldots, t_s)$ a continous function on C^s; we discuss the problem of computing:

$$If = \int_{C^s} f(\mathrm{t})\, d\mathrm{t}$$

on a distributed memory multiprocessor.

In the past years, several algorithms and software have been developed in order to solve the above problem in one dimension, but high computational cost makes very hard dealing with the multidimensional case using a serial computer [4,7]. Now, parallel computing seems to be able to get over this aspect, because multidimensional quadrature has an intrinsic parallelism. In fact, we can split the integration domain into more subdomains and then concurrently integrate the function in each of them. As in the one dimensional case, adaptive quadrature is a good approach to the problem of achieving accuracy and reliability while attempting to minimize the number of function evaluations.

In this paper we present an algorithm based on a global adaptive Simpson's product rule, which dinamically balances the workload and reduces the data communication among the processors, in order to efficiently use a MIMD message passing multiprocessor with $p = 2^d$ nodes (e.g. hypercube).

This work was partially supported by C.N.R. under contract No. 88.00326.01. Special Project on Informatic System and Parallel Computation

2 Adaptive Quadrature and Parallel Computing

An adaptive algorithm for numerical quadrature is an algorithm which process a family of subdomains $\{D_k\}$ of the integration domain C^s with the aim of computing an approximation $\tilde{If}$ of If such that:

$$Ef = |\tilde{If} - If| < \varepsilon$$

where ε is an user-specified tolerance. To do this, one computes a sequence of couple:

$$\{I_i, E_i\}$$

approximating If and Ef respectively, such that:

$$\lim_{i\to\infty} I_i = If, \qquad \lim_{i\to\infty} E_i = 0$$

The approximations I_i and E_i are computed evaluating If and Ef in each subdomain of a partition of C^s and then summing the partial results.

In order to estimate If and Ef in each subdomain D_k, we use a 2-panel and a 4-panel Simpson's product rule. Called these rules S_2 and S_4 respectively, we can use the first one as estimate for If, and $|S_2 - S_4|$ as estimate for Ef. However it should be noted that these rules can be easily replaced with more efficient rules for large dimensions, like *number theoretic rules* or *lattice rules* [5,8].

The main problem in an adaptive algorithm for numerical quadrature is the criterion for processing the subdomains $\{D_k\}$, and the stopping criterion. Two basic strategies are known [3]:

locally adaptive strategy: at each stage of the algorithm a subdomain D_k is accepted if a local acceptance criterion is satisfed (generally it is required that the error estimate in this subdomain is smaller than $\varepsilon/volume(D_k)$);if this does not happen, then it is subdivided in more subdomains that are added to the set of the subdomains not yet examined (*pending set*).

globally adaptive strategy: all subdomains remain pending until an acceptance criterion for the entire pending set is satisfied (generally it is required that the sum of the error estimates in each subdomains is smaller than ε). At each stage of the algorithm the subdomain with the largest error estimate is subdivided.

Therefore a first level description of an adaptive algorithm is the following :

Initialize RESULT and ERROR;
while (the criterion acceptance is not satisfied) do

attempt to reduce ERROR and update RESULT;
endwile

where "*attempt to reduce ERROR and update RESULT*" is based on one of the previous strategies.

For our algorithm we make use of a globally adaptive strategy and in order to parallelize it, we split the subdomain with maximum error estimate among the processors.

Since all subdomains in the pending set can be successively processed, in this strategy it is necessary an ordered data structure to store them.

Thus, at each iteration we are able to find the subdomain with maximum error estimate and subdivide it. Further, this data structure can be very large, because, if we bisect a subdomain in all dimensions, 2^s new subdomains are added at each iteration. Thus a very efficient list management is required.

About this, we can note that a completely ordered list is not necessary for our aims, because we are interested only to the maximum value in the list and it is sufficient a partially ordered binary tree sometime called *heap* [3].

In a heap, the value of each node is greather or equal to that one of its subnodes. In this way it is easy to see that the maximum of the tree is in the root. Sorting such a data structure requires $O(N \log_2 N)$ comparisons in the tree.

A distributed memory multiprocessor is so called because each processor has its own private memory, which cannot be accessed directly by an other processor. Thus it is necessary an explicit message passing. Generally the time required for sending a message from a processor to another one is bigger than the time spent in a floating point calculation. Therefore communication have to be strongly reduced [1].

In order to implement the previous algorithm on a MIMD distributed memory multiprocessor, we can avoid to exchange the whole heap among all nodes at every iteration, by building a different heap in each node. Therefore the nodes have to communicate each other only the root of the heap, in order to find the subdomain with maximum error estimate and to send it to all nodes.

Once the nodes received such subdomain, they share it by bisecting the sides in every dimension. In this way the subdomains preserve always its hypercube form, and $2^s/p = 2^{s-d}$ (with $s \geq d$) new subdomains are added to the heap of each node at each iteration.

Therefore the algorithm of each node is the following:

Initialize RESULT and ERROR;
while (ERROR> ε) do

compare the own heap root with the other ones
in order to find the maximum error estimate;
if (heap root = maximum error estimate) then
send the subdomain in the heap root to all nodes;
else
receive the subdomain with maximum error estimate;
endif
share such subdomain with the other nodes, add 2^{s-d}
new subdomains to the heap and reorder it;
update RESULT and ERROR;
endwhile

In this algorithm the nodes need a controlling host just for the I/O operations. Furthermore any "problem" in the integrand function (like discontinuity or peaks) is "shared" among the nodes to guarantee a good load balancing.

3 Computational Environment and Numerical Results

The previous algorithm has been developed for a MIMD distributed memory multiprocessors; we used a Transputer board with four Inmos Transputer T800 [2], each of them with 1 Mbytes of memory, connected to an Intel 80386/80387 processor host.

For developing such an algorithm we used the FORTRAN version of Express. This is a communication environment based on the CrOS III routines, which allows the user code to be virtually identical for each machine which supports the same communication environment, without specification about the underlying architecture. In addition, Express supplies an I/O routine set which allows every node to access the host computer operating system utilities (Cubix), and a parallel graphical system (Plotix) [6].

To give an idea of the range of application of this algorithm, we used the following kinds of integrand function in two, three and four dimensions:

f_1: oscillating function

f_2: function with a singularity near the boundary

f_3: function with singular first derivative

f_4: gaussian-like function

In two dimensions we done overall graphical tests using Plotix, in order to show how the algorithm is able to locate the areas in which is hard to

integrate the function in the unit square $[0,1]^2$ and to share them among the processors. Figures $1-4$ show the results using 4 processors.

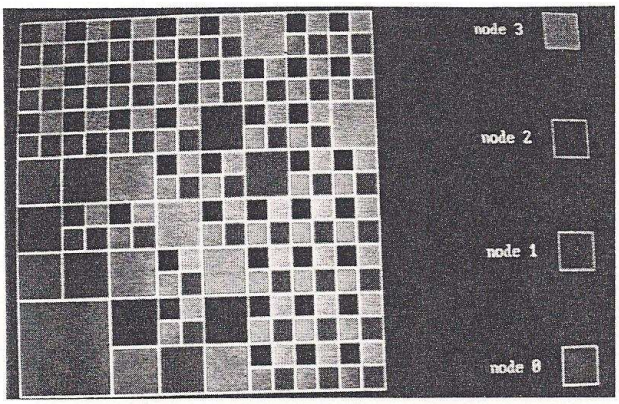

Figure 1: $f_1 = \sin\cos 15xy$
Required tolerance: 2.5×10^{-4}
Integrand evaluations number/processor: 1550

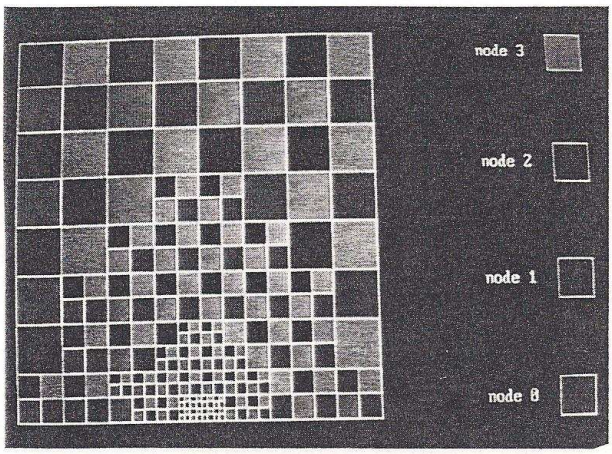

Figure 2: $f_2 = 1/((x-.5)^2 + (y-.5)^2)$
Required tolerance: 5×10^{-5}
Integrand evaluations number/processor: 1850

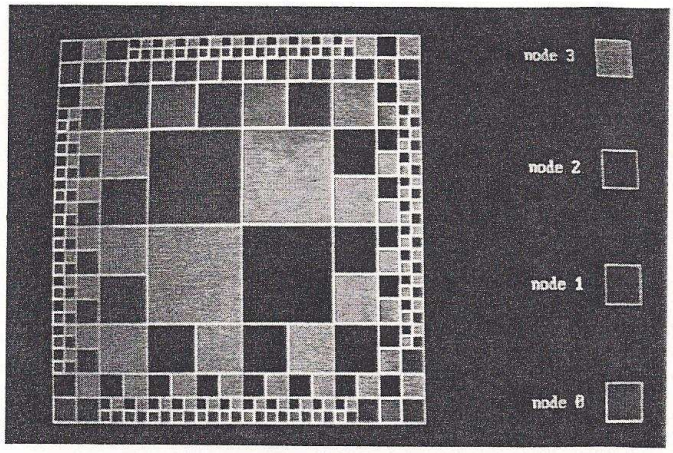

Figure 3: $f_3 = \sqrt{xy(1-x)(1-y)}$
Required tolerance: 2.5×10^{-5}
Integrand evaluations number/processor: 2175

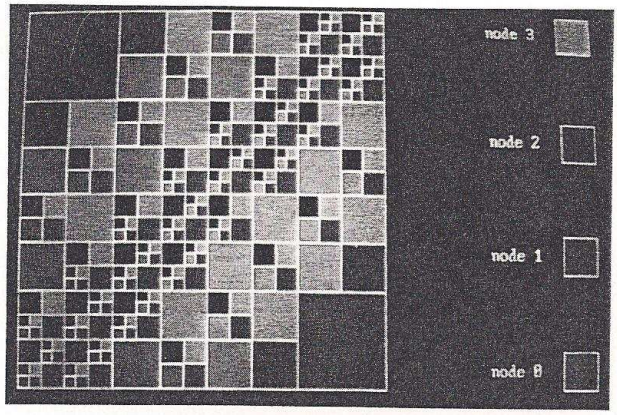

Figure 4: $f_4 = e^{-50(x-y)^2}$
Required tolerance: 2.5×10^{-5}
Integrand evaluations number/processor: 2175

The previous Figures show that the work of the nodes depends by the integrand behaviour. For example in Figure 2 we note that, in order to integrate the function f_2 in the square $[0,1]^2$, more subdomains division (represented by decreasing-side squares) are required near the singularity at (0.5 , -0.1). Furthermore it is possible to note that the workload is well distributed among the processors as showed by different colors in the picture. Analogous considerations can be done for the other functions.

For the 3-dimensional case we report timing and error results in Table 1 and Figure 5. For this case we measured about 8% of the total execution time spent in communication when one uses 4 processors.

Table 2 and Figure 6 refer to the 4-dimensional case, in which we measured only the 2% of the total execution time spent in communication using 4 processors.

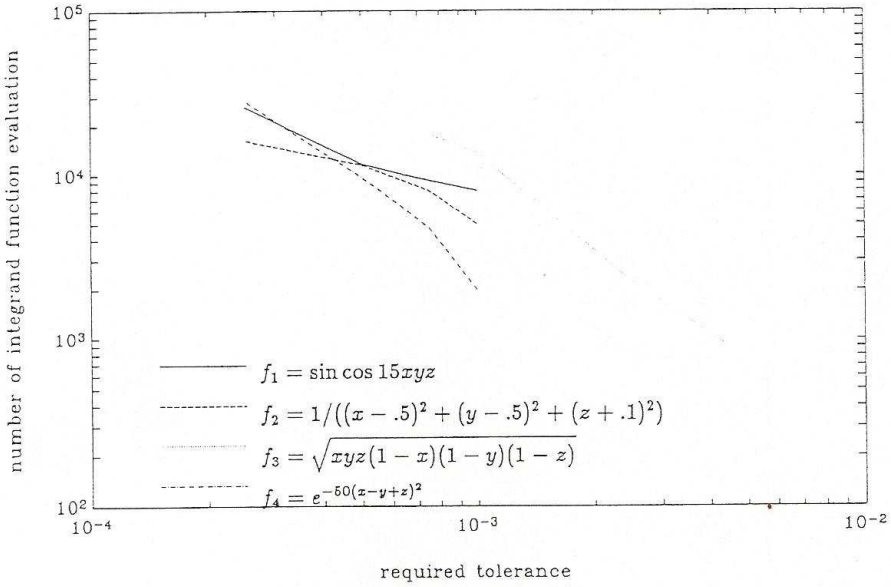

Figure 5: 3-dimensional case.
Integrand evaluations number as
function of various required tolerances.

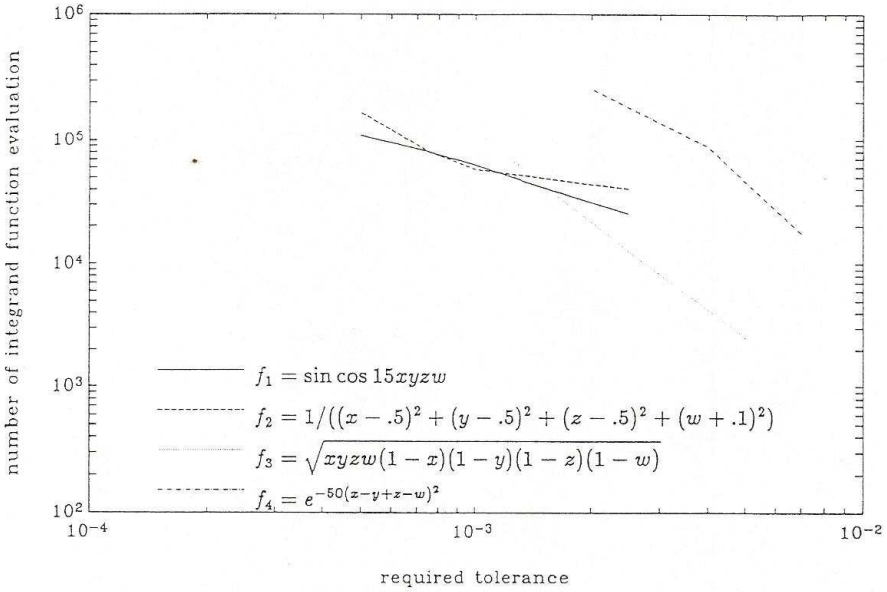

Figure 6: 4-dimensional case.
Integrand evaluations number as
function of various required tolerances.

Type	S_2	E_2	S_4	E_4
1	1.95	0.97	3.65	0.91
2	1.94	0.97	3.64	0.91
3	1.94	0.97	3.55	0.88
4	1.95	0.97	3.59	0.89

Table 1
Speed-up (S_p) and Efficiency (E_p)
using p processors with required accuracy $\varepsilon = 5 \times 10^{-4}$

Type	S_2	E_2	S_4	E_4
1	1.99	0.99	3.94	0.98
2	1.97	0.98	3.90	0.97
3	1.90	0.97	3.90	0.97
4	1.97	0.98	3.90	0.97

Table 2
Speed-up (S_p) and Efficiency (E_p)
using p processors with required accuracy $\varepsilon = 10^{-3}$

Such results show that our algorithm fully exploits the intrinsic parallelism of the multidimensional quadrature problem.

In fact, in particular for large dimensions, the Speed-up and the Efficiency values are very good.

Therefore parallel computing seems able to get over the computational cost drawback of this problem, and it can be an effective tool in order to develope efficient mathematical software for multidimensional quadrature.

4 Bibliography

[1] G.C. Fox et al - *Solving Problem on Concurrent Processors* - Prentice Hall, Englewood Cliffs, New Jersey (1988)

[2] Inmos - *The Transputer Databook* - First Edition (1989)

[3] M.A. Malcom and R.B. Simpson - *Local Versus Global Strategies for Adaptive Quadrature* - ACM Transaction on Mathematical Software, Vol. 1 No. 2 (1975)

[4] *Nag Library, Mark 13* - Numerical Algorithms Group, Oxford (1988)

[5] H. Niederreiter - *Quasi Monte Carlo Methods and Pseudo Random Numbers* - Bullettin of Amer. Math. Soc., Vol. 24 No. 6 (1978)

[6] Parasoft Corp. - *Express: A Communication Environment for Parallel Computers* - Parasoft (1988)

[7] R. Piessens et al - *QUADPACK* - Springer-Verlag, Berlin Heidelberg (1983)

[8] I.H. Sloan and P.J. Kachoyan - *Lattice Methods for Multiple Integration: Theory, Error Analysis and Examples* - SIAM Journal of Numerical Analysis, Vol. 24 No. 1 (1987)