Multilevel Algorithms for Multidimensional Integration in High Performance Computing Environment

Giuliano Laccetti, Marco Lapegna, Diego Romano,and Valeria Mele

Department of Mathematics and Applications, University of Naples Federico II
via Cintia Monte S.Angelo – 80126 Naples (Italy)

***Abstract*— A multilevel parallel adaptive algorithm for the approximate computation of a multidimensional integral over an hyperrectangular region is described. This new algorithm is aimed at an efficient implementation on a MIMD distributed memory Multiprocessor with multi-core CPUs. For this purpose two levels have been introduced in the algorithm: an upper level is responsible of the communication among the nodes and a lower level is responsible of the coordination of the cores in a single node. First performance results on a Blade Server with 16 nodes and 2 quad-core CPUs per node have been achieved.**

I. INTRODUCTION

FROM an architectural point of view, a High Performance Computing environment can be described by means of a tree structure, where at the lowest level there are multi-core CPUs, at the middle level there are workstation clusters and blade servers, and at the highest level the wide area network connects the systems. For such a reason, the development of algorithms and scientific software for these environments needs the integration of several methodologies to face the different kinds of parallelism of each architectural level (distribution of the computation among the systems, parallelism among the CPUs as well among the cores of a single CPU). The aim of this work is to show the experiences gained during the development of adaptive algorithms for multidimensional integration on high performance computing systems (cluster of workstations, MPP systems, blade servers) with multi-core CPUs. In the proposed algorithm the tasks are dynamically scheduled on the cores of a single CPUs accordingly to the availability of the local resources, and periodically rearranged among the CPUs of the system to balance the workload. This two-levels approach produces an algorithm with a reduced number of synchronization points among the tasks and, at the same time, a high degree of scalability.

II. ADAPTIVE ALGORITHMS FOR MULTIDIMENSIONAL INTEGRATION

Since the scientific importance of the numerical computation of a multidimensional integral:

$$I(f) = \int_U f(t_1, \ldots t_n) dt_1 \ldots dt_n \qquad (1)$$

in the past several efficient routines for the solution of this problem have been developed, where $U = [a_1, b_1] \times \ldots \times [a_n, b_n]$ is a *n*-dimensional hyperrectangular region. Most of them (see for example [1, 6, 8]) are based on adaptive algorithms that allow high accuracy with a reasonable computational cost. Such algorithms are iterative procedure that,

at each iteration *j*, evaluates a composite integration rule on a family of sub-domains $s_k^{(j)}$ *(k=1,..,K)* of a partition $\wp^{(j)}$ of *U*. The aim is to compute an approximation *Q(f)* of *I(f)* and an estimate *|E(f)|* of the absolute error *|I(f)-Q(f)|* such that:

$$|I(f)-Q(f)|<|E(f)|<\varepsilon \quad (2)$$

where ε is an user-requested tolerance. To achieve this, the algorithm computes a sequence $Q^{(j)}$ of composite quadrature rules defined on the sub-domains of $\wp^{(j)}$ approaching *I(f)* and a sequence $|E^{(j)}|$ of approximations of the error approaching 0, until the user-requested tolerance is satisfied or it is realized that the error criterion (2) cannot be achieved within a allowed bound on the number of function evaluations. For dimension up to 15 we have good rules available for standard regions [2] .

Since the convergence rate of this procedure depends on the behavior of the integrand function (presence of peaks, oscillations, etc), in order to reduce as soon as possible the error, at the iteration *j* the sub-domain $\hat{s}^{(j-1)} \in \wp^{(j-1)}$ with maximum error estimate $\hat{e}^{(j-1)} = \max_{j=k,..,K} e_k^{(j-1)}$ is split in two (or more) parts s_λ and s_μ that take the place of $\hat{s}^{(j-1)}$ in the partition $\wp^{(j-1)}$. In the same way the approximations $Q^{(j)}$ and $|E^{(j)}|$ are updated.

Usually the sub-domain $\hat{s}^{(j-1)}$ is split in only two parts by halving the edges along the direction where the fourth divided difference of the integrand function is larger than in the other directions. Such a value is taken as a measure of the difficulty of the integration in that direction. This kind of procedure is called *global* adaptive algorithm. A framework for a sequential global adaptive algorithm for the computation of multidimensional integrals is therefore the following :

begin Algorithm 1
Initialize $\wp^{(0)}$, $Q^{(0)}$ and $|E^{(0)}|$
while (stopping criterion not satisfied)
1)find $\hat{s}^{(j-1)} \in \wp^{(j-1)}$ such that $\hat{e}^{(j-1)} = \max_{k=1,..,K} e_k^{(j-1)}$
2) divide $\hat{s}^{(j-1)}$ in two parts s_λ and s_μ
3) evaluate the quadrature rules and compute the error estimates in s_λ and s_μ
4) insert in $\wp^{(j)}$ the sub-domains according their error estimates
5) update $Q^{(j)}$ and $|E^{(j)}|$
endwhile
end Algorithm 1

There are several strategy to introduce parallelism in a quadrature algorithm [7] , but in order to achieve high efficiency our approach is based on the distribution of the sub-domains of $\wp^{(j)}$ among the processing units so that it is possible to process them concurrently. This approach is known as Parallelism at subdivision Level and allows to introduce a sufficient large granularity in the algorithms with a reduced cost in synchronization and communication . The main problem of this approach is that the sequence of $\wp^{(j)}$ is unpredictable, so that it is impossible to distribute uniformly the workload among the computing units beforehand. Therefore, to ensure proper balancing, a parallel algorithm needs to reorganize periodically a more suitable work subdivision among the computing units of the system (nodes as well as cores into each node). This type of load balancing (sometime called *dynamic load balancing*) reacts to the current state of the algorithm during the execution of parallel tasks

and it can improve the performance of parallel systems in case of unpredictable workload.

With the aim of defining a suitable strategy to redistribute the sub-domains among the processing units, let us observe that the two architectural levels in a MIMD distributed memory system with multi-core CPUs (node level and core level) require different algorithms development methodologies because of the very different architectural features. The upper level require a traditional approach based on message passing with the issue of reducing the communication overhead, whereas the lower level require an approach based on the concurrent access to shared memory with the other issue of reducing the synchronization overhead. For these reasons we propose a multilevel algorithm, where each level is responsible to manage data for the corresponding architectural level.
Three separate discussions on these levels and how they can be combined together in the same algorithm follow.

A. Parallelization among the cores

To implement Algorithm 1 in a multi-core based computing system, firstly we observe that the cores cannot be considered as completely independent processing unit, because they share both on-chip resources like caches or bus, as well external resources like the main memory or I/O devices [5]. These hardware features draw a picture where several computing units access to shared resources, so that a natural programming paradigm for an efficient use of these devices is based on the Thread Level Parallelism. With this approach, *P* threads can execute concurrently the Algorithm 1, with a completely asynchronous and out-of-order execution. However, it should be noted that to achieve high efficiencies in these computing environments, a key issue is the asynchronicity of the tasks accessing the on-chip shared resources. This is a critical issue for an adaptive algorithm like Algorithm 1, because, as we said, such an algorithm have to select the sub-domain with maximum error estimate to reduce as fast as possible the error estimate. Usually in an adaptive algorithm for multidimensional quadrature, the sub-domains are arranged in a data structure that has to be accessed by all threads. For a multicore version of Algorithm 1, the data structure resides in main memory which implies the steps 1) and 4) being two critical sections of the algorithm because of the risk for race condition on this shared data structure. However it should be noticed that the computational cost of such steps executed in critical sections is generally much smaller than the cost of several step 3) executions concurrent on the cores, since there are several data structures with a management computational cost of $O(\log_2 K)$, where K is the number of the stored sub-domains. Such a value is much smaller than the cost for evaluating an integration formula, mainly for large values of *n*. For such a reasons, the idle times are almost completely eliminated in Algorithm 1 when executed on a multi-core CPU.

B. Parallelization among the nodes

After the sub-domains refinement among the cores described in the previous subsection, in order to ensure proper load balancing among the nodes, it is necessary to compare periodically the data in their private memory and to reorganize a more suitable sub-domains distribution in the upper level. A common strategy in this sense is to exchange the sub-domains with a large error among the nodes. However we remember that, since a necessary condition for the scalability is the data locality, the communication time for the sub-domains rearrangements must be independent on the size of the system (i.e. the number of nodes).

So all global communications are removed to improve the scalability of our algorithm, because such communication tasks depend on the number of nodes and make the algorithm poorly scalable. Meanwhile the workload is balanced by performing only communications between pairs of directly connected nodes in a 2-dimensional periodical mesh [3, 4] . In such a topology each node has 4 neighbours: 2 for every direction. In our algorithm each node attempts to send its sub-domain with the largest error estimate to the connected nodes in the mesh, in order to share it with the other nodes. More precisely, let $\hat{e}_i$, $\hat{e}_{i+}$ and $\hat{e}_{i-}$ be respectively the largest error estimates of the sub-domains in the current node P_i, in the next node P_{i+} and in the previous node P_{i-}. If $\hat{e}_i > \hat{e}_{i+}$ then the current node P_i sends the sub-domain with largest error estimate $\hat{s}_i$ to the processor P_{i+}. In the same way if $\hat{e}_{i-} > \hat{e}_i$ the current node receives the sub-domain with error estimate $\hat{e}_{i-}$ from P_{i-}. In such a workload distribution, the communication cost is independent of the number of nodes in the system, thus enhancing the scalability of the algorithm.

C. The multilevel algorithm

At the end of this section we give a description of the whole multilevel algorithm. Algorithm 2 is the node algorithm in the Single Program Multiple Data (SPMD) model for the node P_i. The inner loop is implemented in a thread and replicated on each core of the nodes.

begin Algorithm 2 for node P_i

Initialize $\wp_i^{(0)}$, $Q_i^{(0)}$ and $|E_i^{(0)}|$

while (node stopping criterion not satisfied)

Define the neighbors nodes P_{i+} and P_{i-}

if ($\hat{e}_i > \hat{e}_{i+}$) send $\hat{s}_i$ to P_{i+}

if ($\hat{e}_{i-} > \hat{e}_i$) receive $\hat{s}_i$ from P_{i-}

while (core stopping criterion not satisfied)

{enter critical section}

1) select $\hat{s}_i^{(j-1)} \in \wp_i^{(j-1)}$ with largest error estimate in $\wp_i^{(j-1)}$

{exit critical section}

2) divide $\hat{s}_i^{(j-1)}$ into two parts s_λ and s_μ

3) evaluate the quadrature rules and compute the error estimates in s_λ and s_μ

{enter critical section}

4) insert in $\wp_i^{(j)}$ the sub-domains according to their error estimates

{exit critical section}

5) update $Q_i^{(j)}$ and $|E_i^{(j)}|$

endwhile

endwhile

compute $Q(f) = \sum_i Q_i^{(f)}$ and

$|E(f)| = \sum_i |E_i^{(f)}|$

end Algorithm 2

III. TEST RESULTS

In this section we present some results achieved on a blade server with 16 nodes each of them equipped with two Intel XEON E5410 quad-core CPU and 16 Gbytes of main memory.

The following Figure 1 refers to the Algorithm 2 implemented on a single node. Such a test is therefore aimed to measure the effectiveness of the lower level of the algorithm, i.e. the level responsible of the dynamic load balancing on the cores in the nodes.

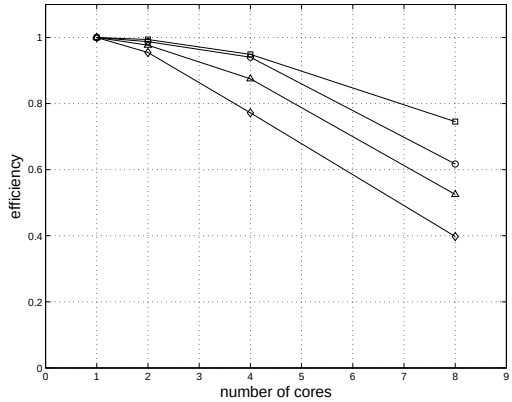

Figure 1. Algorithm 2 on a single node

The test refers to the computation of (1) with $n=10$, $U=[0,1]^n$ and the following oscillating integrand function

$$f = \cos\left(2\pi\beta_1 + \sum_{i=1}^{n} \alpha_i x_i\right)$$

where α and β are random values that determine the location and the sharpness of the oscillation. Therefore the reported results are the average on 10 executions obtained varying these parameters.

The figure shows the efficiency of the Algorithm 2 with different numbers of function evaluations, when the number of cores grows. The graph can be taken as a measure of the scalability of the Algorithm 2 on the cores of each node, i.e. as the ability to keep high efficiencies when the number of cores grows. Mainly for large problems (5217815 function evaluations), the Algorithm shows a very good scalability. However a different slope of the curves with 8 cores should be observed. This is because the CPUs have four cores each, so that the favorable ratio computation/access-to-data produces an execution with an extensive use of cached data up to four cores. On the other hand, the use of eight cores imply a more frequent access to the main memory charged to the second CPU.

Figure 2 reports the performance of Algorithm 2 measured with different numbers of cores with respect to the problem size, using the same computation from the previous test. As a measure of the performance we choose to use the number of function evaluations per second. Again we observe a significant gain in performance for sufficient large problems when the number of cores grows.

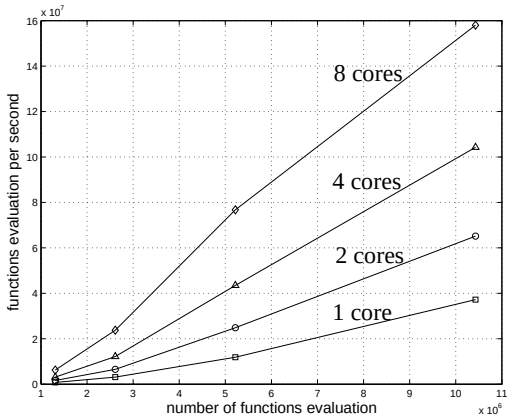

Figure 2. Performance of Algorithm 2

Finally Figure 3 reports a scalability test on the 16 nodes of the system, using one core per node. This test is therefore aimed to measure the effectiveness of the upper level of the algorithm, i.e. the level responsible of the redistribution of the sub-domains among the nodes. The distribution of the sub-domains with large error among connected nodes in a 2-dimensional periodical grid without global communication produce an high degree of scalability, that is the ability of the algorithm to

keep high efficiency when the number of nodes grows.

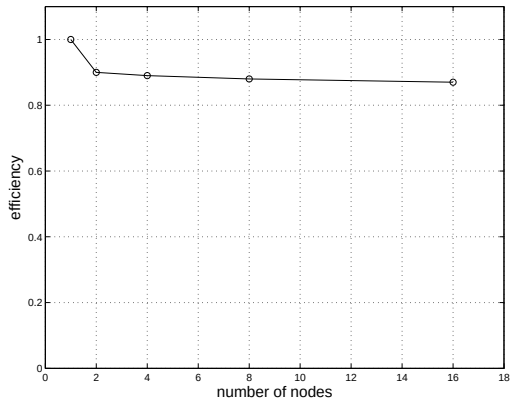

Figure 3. Scalability test

ACKNOWLEDGMENTS

This work has been supported by Italian Ministry of Education, University and Research (MIUR) within the activities of the SCOPE project (PON Ricerca" 2000-2006 - Avviso 1575)

REFERENCES

[1] J. Berntsen, T. Espelid and A. Genz - Algorithm 698: DCUHRE - An adaptive multidimensional integration routine for a vector of integrals. - ACM Transaction on mathematical software, 17 (1991), pp. 452-456.

[2] R. Cools and P. Rabinowitz. - Monomial cubature rules since ``Stroud": a compilation. - Journal of Computational and Applied Mathematics, 48 (1993), pp.309-326

[3] M. D'Apuzzo and M. Lapegna. – Scalability and Load Balancing in Adaptive Algorithms for Multidimensional Integration – Parallel Computing vol. 23 (1997), pp. 1199-1210.

[4] A. D'Alessio and M. Lapegna - A Scalable Parallel Algorithm for the Adaptive Multidimensional Quadrature – in Parallel Processing for the Scientific Computing, R. Sincovec et al eds, SIAM 1993, pp. 933 - 936

[5] J. Dongarra, D. Gannon, G. Fox, and K. Kennedy - The Impact of Multicore on Computational Science Software**.** - CTWatch Quarterly, Volume 3 Number 1, February 2007

[6] D. Khaner and O. Rechard, - TWODQD: an adaptive routine for two-dimensional integration. - Journal of Computational and Applied Mathematics, 17 (1987), pp.215-234.

[7] A. Krommer and C. Ueberhuber – Computational Integration. – SIAM 1998

[8] G. Laccetti and M. Lapegna - PAMIHR. A Parallel FORTRAN Program for Multidimensional Quadrature on Distributed Memory Architectures. – in proceedings of EUROPAR99 Conference, LNCS 1685, Springer Verlag, pp. 1144 -1148.

Giuliano Laccetti is full professor of *Computer Science* in the Faculty of Sciences, University of Naples Federico II. He has been/is involved in several (national and european) Research Projects about Parallel and Distributed Computing, Scientific Computing, Grid Computing. Presently he is scientific coordinator of the University of Naples researchers group in the european EGEE III Project.

Marco Lapegna is associate professor in the Faculty of Sciences, University of Naples Federico II, where he teach Parallel Computing, Operating Systems and Programming Laboratory. He has been involved in many research projects on the computing science topics, with special regard to parallel, grid and scientific computing.

Diego Romano is researcher at the Italian National Research Council where at present is on leave to study for his PhD on Computational and Computer Sciences at the University of Naples Federico II. His research interests are on parallel and distributed computing, focusing on topics such as Parallel Computer Graphics

Valeria Mele is PhD student in Computational Sciences at the Department of Mathematics and Applications of the University of Naples Federico II.