INFORMATICA, 1998, Vol. 9, No. 2, 123–140
© 1998 *Institute of Mathematics and Informatics, Vilnius*

Numerical Integration on Distributed-Memory Parallel Systems

Raimondas ČIEGIS *
Institute of Mathematics and Informatics, Vilnius Gediminas Technical University
Akademijos 4, 26000 Vilnius, Lithuania
e-mail: raimondas.ciegis@fm.vtu.lt

Ramūnas ŠABLINSKAS
Vytautas Magnus University
Vileikos 8, 3035 Kaunas, Lithuania
e-mail: ramas@omnitel.net

Jerzy WAŚNIEWSKI
The Danish Computing Centre for Research and Education
UNI-C, Bldg. 305, DK-2800 Lyngby, Denmark
e-mail: Jerzy.Wasniewski@uni-c.dk

Received: November 1997

Abstract. In this paper we describe implementation of numerical adaptive algorithms for multi-dimensional quadrature on distributed-memory parallel systems. The algorithms are targeted at clusters of workstations with standard message passing interfaces, e.g., PVM or MPI. The most important issues are communication and load balancing. Static and dynamic partitioning of the region are considered. Numerical results on various workstation clusters are reported.

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

1. Introduction

We consider the problem of determining a numerical approximate value of the multi-dimensional integral

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

within a 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. Numerical calculation of multiple integrals demands for large amounts of computing time (see Stroud, 1971). Hence parallel adaptive algorithms are particularly interesting (Bull and Freeman, 1995; Freeman and Phillips,

*Corresponding author

1991; Genz, 1982; Genz, 1990). The whole task is subdivided into smaller subtasks which can be solved in parallel on different processors. There we must try to preserve load balance and we must take into account communication costs. The solution of both problems depends strongly on the characteristics of the parallel computer used in computations. The algorithms of this paper are targeted at clusters of workstations with standard message passing interface. We use PVM in our numerical experiments (Geist *et al.*, 1993). The important features of such parallel computers are heterogenity of the cluster and unfavourable ratio between computation and communication rates. Our goal is to design parallel numerical integration algorithms which use minimal amount of data communication. We will develop static and dynamic task distribution algorithms and will investigate the importance of tradeoff between load balancing and communication costs.

In Section 2 we review the basic parallel algorithms developed for multi-dimensional integrals. In Section 3 a simple straightforward parallel implementation of the standard serial multi-dimensional algorithm is used to illustrate the importance of the load balancing and communication latency on the efficiency of parallel algorithms. Numerical results for one test problem are presented. The algorithm of initial static partitioning of the region of integration and static subtasks distribution is described in Section 4. Numerical results for the SSSD algorithm are given in Section 5. In Section 6 we describe a mosaic static subproblems distribution algorithm. Dynamic task distribution algorithms are investigated in Section 7.

2. Parallel Algorithms

Numerical integration algorithms attempt to approximate $I(f)$ by the sum (Stroud, 1971)

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

The numbers w_i are called *weights*, and the x_i are called *knots*. They determine the type of the basic rule. Efficient numerical algorithms are adaptive and a strategy for selection of hyper-rectangles for further subdivision is included into the algorithm, see, e.g., Johnson and Riess (1982). The cubature rule pair, which was used for all our numerical experiments, is due to Genz and Malik (1980). The standard routine DO1FCF in the NAG library is also based on this formula, see also parallel adaptive algorithms given in Bull and Freeman (1995); Genz (1990). This cubature formula requires $2^d + 2d^2 + 2d + 1$ evaluations of the integrand to estimate the integral $I(f)$ over a d-dimensional hyper-rectangle. We also get the dimension of the hyper-rectangle in which the integrand is most badly-behaved and this dimension is used for futher subdivision. The most simple way to get a parallel integration algorithm is to distribute $2^d + 2d^2 + 2d + 1$ integrand evaluations among p processors. This algorithm is refered to as the Fine-Grained (FG) algorithm and is well suited in the case of fast communication or when a cost of integrand evaluation is high. We mention the main drawbacks of the FG algorithm:

1. The number of processors that could be exploited is limited to the number of points in the quadrature rule for a pair of hyper-rectangles.
2. It requires a very large data communication.

The second characteristic of the FG algorithm is most important for parallel computers based on distributed workstations. In Section 3 we will give a more detailed analysis of the FG algorithm and will present the results of numerical experiments.

Many authors have considered parallel algorithms for numerical integration which exploit the coarser-grained parallelism by identifying not one but a number of hyper-rectangles to be bisected. Then each processor calculates integral approximation and error estimate for the whole hyper-rectangle (or for a number of hyper-rectangles). We obtain new algorithms which can be more efficient even for serial computers in some cases (Bull and Freeman (1995)).

Next we review some *hyper-rectangle selection* methods used in these algorithms, a more complete survey is given in Bull and Freeman (1995). In the Dynamic Asynchronous algorithm (Bull and Freeman, 1994; Bull and Freeman, 1995) each pair of processors finds hyper-rectangle with largest error estimate which is inactive, marks it active, bisects it and applies quadrature rules to both hyper-rectangles in parallel. In the Dynamic Synchronous algorithm (Genz, 1982; Genz, 1990) at each stage we identify $p/2$ hyper-rectangles with largest error and subdivide them in such a way as to keep all p processors usefully busy. Many other strategies are used in the Dynamic Synchronous algorithm (Bull and Freeman (1995)). We will mention only few of them. It is suggested to identify all the hyper-rectangles with error estimates greater than $\alpha E_{\max}$, for some α satisfying $0 < \alpha < 1$, where $E_{\max}$ is the largest error. Gladwell (1987) suggested to rank the list of hyper-rectangles by error estimates so that $\varepsilon_1 < \varepsilon_2 < \dots < \varepsilon_s$ and then to calculate r such that

$$\sum_{i=1}^{r-1} \varepsilon_i \leqslant \varepsilon, \quad \sum_{i=1}^{r} \varepsilon_i > \varepsilon, \tag{3}$$

where ε is the given accuracy. Then we identify all the hyper-rectangles with error estimates $\geqslant \varepsilon_r$.

Even these algorithms using coarser-grained parallelism can be not effective for parallel computers based on distributed workstations. They require many synchronization points and constant communication during computation time. Further for multidimensional integrals the stage of searching and updating the list of hyper-rectangles can also be cost-expensive. It is possible to parallelize the region selection stage, but up to three additional synchronization points must be included into the algorithm.

Hence in this paper we will investigate parallel numerical integration algorithms that impose some static partitioning of the region of integration and will consider various strategies of distribution of these independent subproblems among processors. Similar algorithms were considered in De Doncker and Kopenga (1992); Lapenga and D'Alessio (1993). In order to minimize the data communication we will not use any mechanism for redistribution of the work to the other processors in the case of load imbalance.

3. FG Algorithm

We will consider an implementation of the FG algorithm in this section. This example enables us to describe in detail the problems of development of parallel numerical integration algorithms for workstation clusters in the cases when we have a very unfavourable ratio between computation and communication rates.

3.1. *Parallel Algorithm*

As it was stated above, the FG algorithm coincides with the serial one, only a work of evaluation of the $2N$ integrands is distributed among p processors at each bisection stage. The most significant change that was made for our integration routine was the summation of signed regional errors instead of absolute values of the errors. We note that the routine overestimates the error. It defines the error made in region Ω_j as

$$err = S_{N7} - S_{N5}\,, \tag{4}$$

where S_{N7} and S_{N5} are approximations of the integral $I(f, \Omega_j)$ by the seventh and fifth order rules, respectively. Hence the routine uses a partition of the integration region Ω sufficient for the fifth order formula to be accurate within the specified accuracy and calculates an approximation of the integral by the seventh order basic rule.

First we consider a static task distribution algorithm for achieving the load balance on all p processors. Let assume that we have a heterogenous workstation cluster with known computational rates estimates. Let denote computational rates v_j and rank the list of workstations so that

$$v_1 \leqslant v_2 \leqslant \ldots \leqslant v_p. \tag{5}$$

Then the load balancing is achieved if i-th processor calculates N_i integrands, where the number N_i is defined by the following minimization problem

$$\min_{N_i} \max_{1\leqslant j\leqslant p} T_j = T^*, \quad T_j = \frac{N_j}{v_j}, \tag{6}$$

$$N_1 + N_2 + \cdots + N_p = 2N. \tag{7}$$

Solution of this problem is given in Čiegis *et al.* (1996) and is described in Algorithm 1.

Algorithm 1

```
1.  Calculate initial estimates of $N_i$
    $N_i = \left\lfloor 2N\frac{v_i}{V_p} \right\rfloor, i = 1, 2, \dots, p,$
    $V_p = v_1 + v_2 + \dots + v_p,$
    where $\lfloor z \rfloor$ denotes the greatest integer less than or equal to $z$.
2.  Calculate the remainder of undistributed integrands
    $q = N - (N_1 + N_2 + \cdots + N_p).$
3.  Distribute the remainder among processors
      for (i=1; i<=q; i++)
        find $T_j = \frac{N_j+1}{v_j}$,  j=1,2,...,p;
        find $T_k = \min_j T_j$ ;
        $N_k = N_k + 1;$
      end for
```

The multi-processors or clusters of workstations are not the single-user computers devoted to one application at a time. Hence computational rates of processors can change significantly during computations. In such a situation the dynamical distribution algorithm can give better load balancing results, but it enlarges the communication volumes as well. We have used the dynamical distribution algorithm in all numerical experiments reported in this section.

3.2. *Experimental Environment*

As it was mentioned above, we performed all numerical experiments on virtual parallel computers based on distributed workstations connected by a local network or INTERNET. PVM is used as a message-passing interface. The cluster was located in various Universities of Lithuania and consisted of up to 21 various heterogeneous workstations. The types of computers and relative computational rates are given in Fig. 1.

It is often convenient to view costs of data communication between nodes in terms of performing floating-point operations on the nodes. Evaluations of such rates are given in the next section for several different host architecture types.

We are interested in the performance of parallel algorithms on distributed-memory multiprocessors. Hence we will define the *speed-up* S_p as (see Čiegis *et al.* (1996))

$$S_p = \frac{T_s}{T_p}, \tag{8}$$

where T_p is the time used to solve the given problem on p processors, T_s represents a time for the sequential algorithm. In order to compare different algorithms we will also define the quality of the parallel algorithm as

$$Q = \frac{T_s}{T_1}. \tag{9}$$

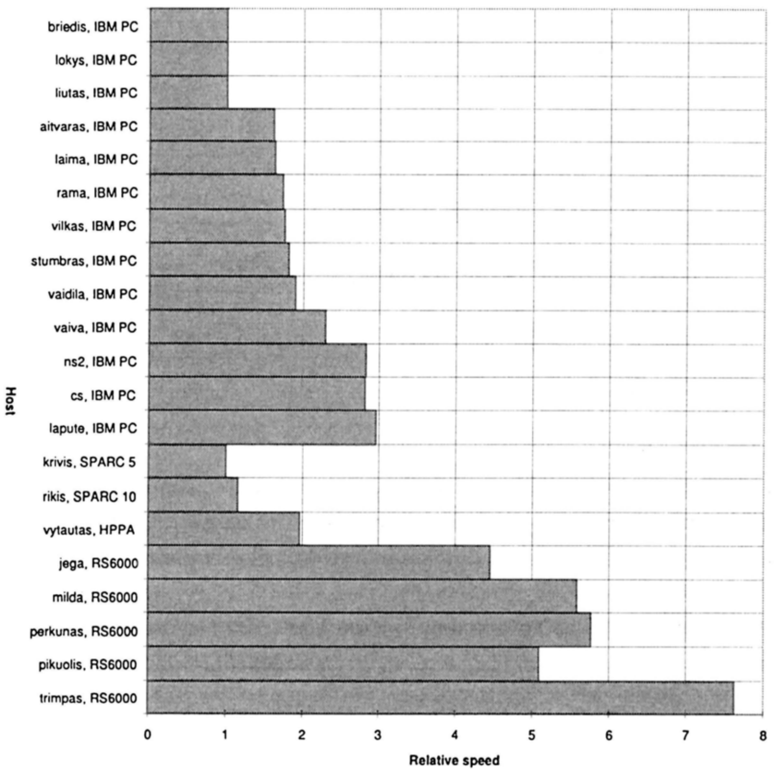

Fig. 1. Parallel Virtual Machine Cluster used in computations.

The *efficiency* of a p-processor parallel algorithm is given by Čiegis *et al.* (1996)

$$E_p = S_p \frac{v_1}{V_p}, \tag{10}$$

where v_1 is the computational rate of the workstation used as the unit of performance and V_p is a total computational power of the given cluster

$$V_p = v_1 + v_2 + \cdots + v_p\,. \tag{11}$$

We will also be interested in the *balancing* of the work distribution among the processors. The balancing defines variation of the task quantity for each of the nodes:

$$B_p = \min_{1 \leqslant j \leqslant p} \frac{pieces_{tot}}{pieces_j} \cdot \frac{v_j}{V_p}, \tag{12}$$

where $pieces_{tot}$ is the total amount of hyper-rectangles processed by the computer cluster and $pieces_j$ is the amount of hyper-rectangles assigned to one of the processors. This definition is used in the case of heterogeneous computer cluster.

3.3. *Test Problems*

We selected four problems which exhibit a variety of integrand function behaviour.

Problem 1.

$$\int_0^1 \int_0^1 \cdots \int_0^1 \sum_{i=1}^{8} \exp(2x_i)\, dx, \quad \varepsilon = 10^{-6}. \tag{13}$$

The Problem 1 has no special features in the integrand. We can expect to get a good load balancing after implementation of the distribution algorithm which is described above.

Problem 2. We calculate the same integral as in Problem 1, but we add appropriate delays so that the evaluation times for the integrand become highly varying. In our practical implementation we incorporate a dummy loop into the evaluation of integrand $f(x)$ and repeat it $\alpha(x)$ times. This effectively simulates a complicated function with varying computational profile. For the Problem 2 $\alpha(x)$ is defined by the following formula

$$\alpha(x_1, x_2, \ldots, x_8) = \prod_{j=1}^{4} \lfloor 2 \cdot x_j^2 + 1.0 \rfloor . \tag{14}$$

Problem 3 (Johnson and Riess, 1982).

$$\int_0^1 \int_0^1 \int_0^1 \int_0^1 \frac{4x_1x_3^2 \exp(x_1x_3)}{(1 + x_2 + 2x_1)^2}\, dx_1\, dx_2\, dx_3\, dx_4, \; \varepsilon = 10^{-9}. \tag{15}$$

Problem 4 (Johnson and Riess, 1982).

$$\int_0^1 \int_0^1 \int_0^1 (0.0001 + x_1x_2x_3)^{-0.9}\, dx_1\, dx_2\, dx_3, \quad \varepsilon = 10^{-7}. \tag{16}$$

This problem has a strong corner singularity, hence we again have difficulties with load balancing.

3.4. *Numerical Results for the FG Algorithm. Computation and Communication Rate Comparison for Various Complexity of the Test Problem*

In this section we present experimental results obtained for the FG algorithm. The experimental environment we described in Section 3.2 consists of heterogeneous computer cluster. It is well known that for parallel computers with distributed memory the most important characteristic is the ratio of computation to communication times. The latency times for clusters of workstations still are very large. We will try to determine the computational complexity of the integrand function which makes the FG algorithm effective. For this reason we incorporate dummy loops in function evaluation so that it evaluates the integrand function α times. The Table 1 shows the computational times in seconds for

Table 1

The computational time for the Problem 1 with $\alpha(x) = \alpha$ on various architectures with $p = 1$ processor

α	*SPARC*	*LINUX*	*HPPA*	*RS6000*
1	459	412	362	163
10	514	450	373	165
100	985	886	501	204
1000	5827	5171	1787	606

Table 2

The computational times for the Problem 1 with various p for $\alpha = 1000$

p	Total time T_{tot}	Speed-up S_p	Efficiency E_p
1	5171	1.00	1.00
2	2788	1.85	0.93
3	1896	2.73	0.91
4	1610	3.21	0.80
5	1370	3.70	0.74

various values of α on various architectures for the Problem 1 with $\alpha(x) = \alpha$. Only one slave process was used in computations.

From the Table 1 we conclude that the unit subproblem should be approximately 1000 times as difficult as the integrand function of the test Problem 1 to have a sense in parallel computations. In such cases the data exchange between processors will take up less time than computations. The Table 2 shows CPU time T_{tot}, speedup S_p and efficiency E_p for the Problem 1 on various numbers of processors for $\alpha = 1000$.

From the results in Table 2 we conclude that the efficiency of FG algorithm is degrading as we increase the number of processors since there is an increasing amount of communication that could not be parallelized.

4. Static Subdivision – Static Distribution Algorithm

In this section we will consider the most simple static subdivision – static distribution (SSSD) algorithm.

The first step is an initial static partitioning of integration area Ω into smaller hyper-rectangles

$$\Omega = \bigcup_{i=1}^{N} \Omega_i . \tag{17}$$

Thus, the integral (1) can be written as:

$$I(f,\Omega) = \sum_{j=1}^{N} I(f,\Omega_j)\,. \tag{18}$$

Numerical approximation of $I(f,\Omega_j)$ is an independent problem and it can be solved concurrently. Each subintegral $I(f,\Omega_j)$ is approximated by the numerical quadrature formula (2) with the given accuracy ε_j. The error tolerance for each subregion can be defined in non-unique way. In a *locally adaptive scheme* ε_j is chosen proportional to the volume of the region, i.e.,

$$\varepsilon_j = \varepsilon \frac{vol(\Omega_j)}{vol\Omega}\,. \tag{19}$$

In a *globally adaptive scheme* the error tolerance for each subregion is chosen to reflect an assumed distribution of the error within Ω. We will use the first scheme in all numerical experiments.

A master/slave model is used for the implementation of the SSSD algorithm. The master process distributes subtasks to slave processes and collects the results. The slave process calculates the approximation of the integral over given subregion Ω_j and returns the result to the master. The load balancing and communication properties of the SSSD algorithm are quite opposite to the same properties of the FG algorithm. Communication volumes of the SSSD algorithm are limited to the minimal amount. But the main difficulty with this algorithm stems from load balancing. There are two primary reasons for the load imbalance. Firstly, it is very likely that the number of subdivisions required within different hyper-rectangles Ω_j will vary considerably. Secondly, the computational work load of each workstation can change dramatically during the time of computation, as new users and/or applications could be added/removed in the parallel system. The numerical results will be given in the next section.

REMARK 1. We have estimated the computational rate of each processor at the beginning of computations by running a simplified benchmark with the same integrand function.

5. Numerical Results for the SSSD Algorithm

5.1. *Homogeneous Cluster Case*

First we assume that all computers in the virtual machine are of the same computational rate, i.e.,

$$v_1 = v_2 = \ldots = v_p = 1\,. \tag{20}$$

As a consequence all slave processes receive subproblems of equal size.

Table 3
The computational results of the SSSD algorithm for theProblem 1

p	T_{tot}	Q	S_p	E_p	B
1	1281	1.00	1.00	1.00	1.00
2	650	1.00	1.98	0.99	0.99
3	380	1.18	3.38	1.13	0.95
4	329	1.00	3.90	0.97	0.98
5	203	1.33	6.30	1.26	0.95
6	192	1.18	6.68	1.11	0.94
7	174	1.25	7.38	1.05	0.84
8	166	1.00	7.70	0.96	0.96

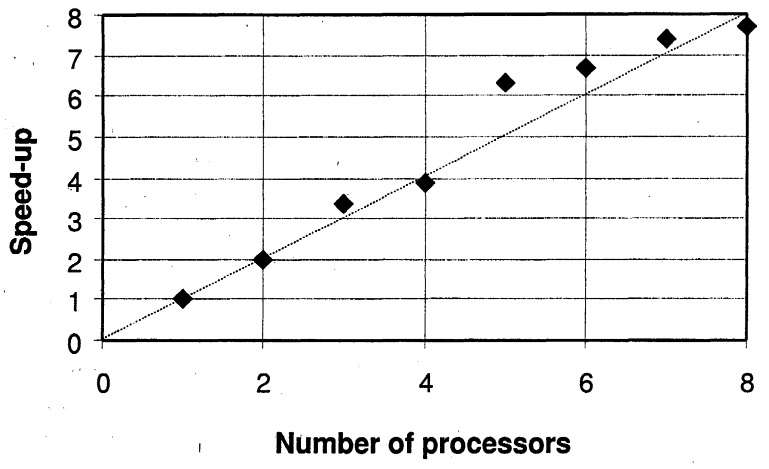

Fig. 2. The speed-up plots for the Problem 1. The cluster of workstations is homogeneous.

In Table 3 we present experimental results for the Problem 1. Experiments were performed with various numbers of workstations included into the cluster. Fig. 2 shows the speedups for the SSSD algorithm as a function of the number of processors.

We notice that we obtain a comparetively good load balancing and efficiency. The integrand function of the Problem 1 has a small variation in complexity over computation region. The imbalance is also influenced by the initial area subdivision algorithm. From the results in Table 3 we can observe this influence for various number of stations.

5.2. *Heterogeneous Cluster Case*

In this case the subproblems are distributed among workstations proportionally to the relative computational rates. The Fig. 3 shows the speedups for the Problem 1 as a function of clusters computational power. The legend indicates the number of workstations in the cluster represented by a point. The speed-up in this figure is affected by two factors – the

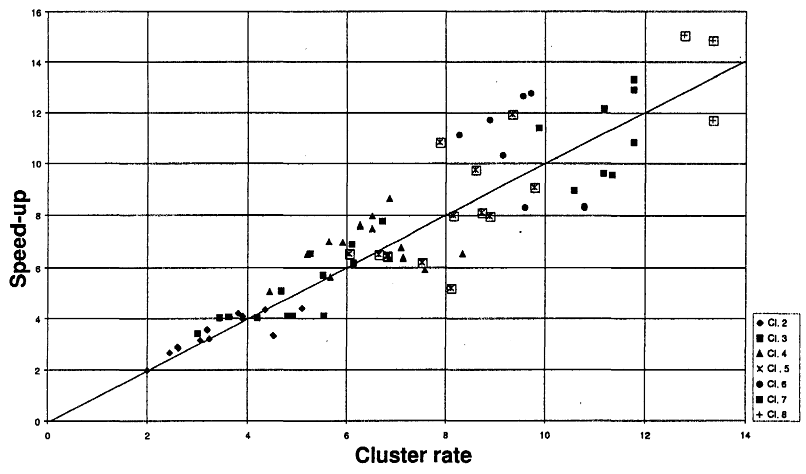

Fig. 3. The speed-up S_p for the Problem 1 on different heterogeneous computer clusters.

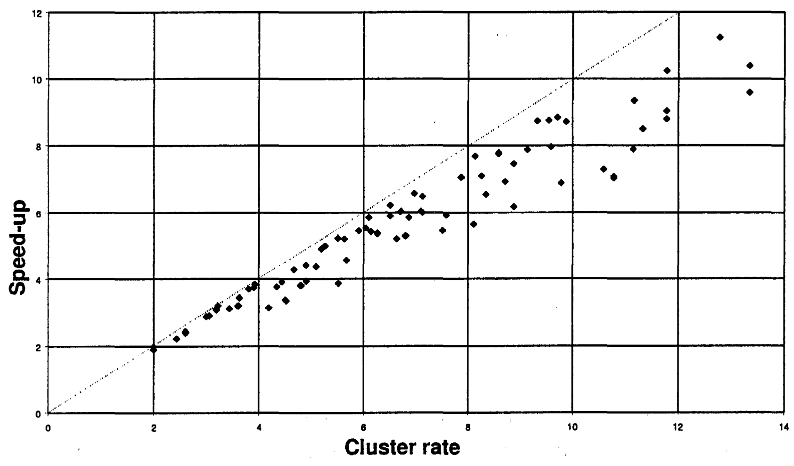

Fig. 4. The speed-up S_p^* for the Problem 1 on different heterogeneous computer clusters.

algorithm quality Q and the load balancing B. After we eliminate the quality factor, only load balancing influences the results. The Fig. 4 plots the speedups with the quality factor eliminated:

$$S_p^* = \frac{S_p}{Q}. \tag{21}$$

We notice, that speed-up trend is decreasing for large computer clusters. The fall of speed-up is also invoked by the absolute value of the computational time of the integral. The bigger computer clusters have shorter computation times which makes load balancing more sensitive for delays.

Table 4
The computational results of the SSSD algorithm for the Problem 2

p	T_{tot}	S_p	E_p	Q	B
1	915	1.00	1.00	1.00	1.00
2	718	1.27	0.64	1.00	0.80
3	451	2.03	0.68	1.04	0.83
4	520	1.75	0.44	0.97	0.70
5	433	2.11	0.42	1.03	0.74
6	496	1.84	0.31	1.00	0.65
7	515	1.78	0.25	1.00	0.60
8	547	1.67	0.21	0.94	0.55

Table 5
The computational results of the SSSD algorithm for the Problem 3

p	T_{tot}	S_p	E_p	Q	B
1	97.9	1.00	1.00	1.00	1.00
2	43.9	2.23	1.11	1.01	0.93
3	37.9	2.58	0.86	1.04	0.70
4	31.0	3.16	0.79	0.98	0.68
5	33.7	2.90	0.58	0.85	0.57
6	29.5	3.32	0.55	0.79	0.59
7	26.8	3.65	0.52	0.89	0.50
8	23.0	4.25	0.53	0.95	0.48

From the results given above we conclude that the adaptation of tasks distribution to relative computational rates of computers increases parallel efficiency of the algorithm.

The minimal communication model in the SSSD method enables us to include into a cluster of workstations subclusters with high communication latency.

5.3. *The Drawbacks of the SSSD Algorithm*

There are two primary reasons for the disappointing performance of the SSSD method on distributed memory multiprocessors. The first factor is that the subproblem distribution algorithm of the SSSD method assumes that integrand function has no special features and all subproblems are of the same computational difficulty. In the case of highly varying complexity of the integrand function the parallel performance is degraded due to load imbalance. As an illustration of this situation we consider the results obtained for the Problem 2 and the Problem 3 on homogeneous workstation cluster. The results are presented in Table 4 and Table 5, respectively.

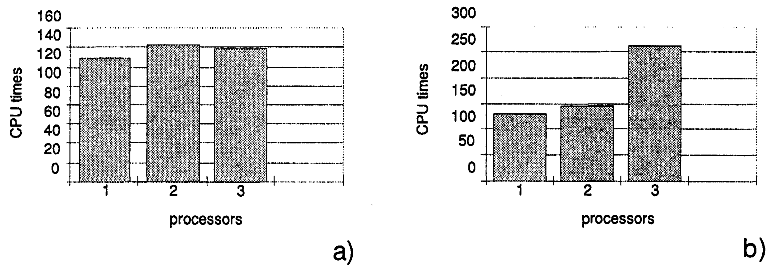

Fig. 5. CPU times of different processors in solving Problem 1 with the SSSD algorithm for the first (a) and second (b) runs.

The second reason for the disappointing parallel performance of the SSSD method on distributed workstations is that a workload of any station can change significantly during computations. The static distribution algorithm can not resolve this challenge. As an example, consider experimental results obtained for Problem 1 on 3 workstation cluster.

Two runs of the SSSD algorithm were done. Figure 5a plots the execution times for all three processors in the case of homogeneous cluster of workstations. In the second run an additional dummy problem was started on the third processor, hence the relative computational speed of this processor decreased two-fold during computation. Figure 5b plots the obtained execution times for the second run.

6. Mosaic Distribution Algorithm

The performance of the SSSD method could be improved by using the mosaic subproblems distribution algorithm (SSMD method). The main goal of the mosaic rule is to avoid large portions of sequential subregions to be sent to the same computer. We consider two simple mosaic distribution algorithms. They are described in pseudocode as follows:

```
Mosaic algorithm 1
1. Make initial partitioning of the integration region $\Omega = \bigcup_{j=1}^{M} \Omega_j$ .
2. Apply the SSSD algorithm in each of subregions $\Omega_j$ .
```

The second mosaic rule uses pseudo-random numbering of subregions. The number of subregions must be sufficiently large in order to preserve load balancing.

```
Mosaic algorithm 2
1. Make initial partitioning of the integration region $\Omega = \bigcup_{j=1}^{M} \Omega_j$ .
2. By using pseudorandom numbers define a new numbering list
   of the subregions
   $V_j = \Omega_{r(j)}, \quad j = 1, 2, \ldots, M.$
3. Distribute subproblems $I(f, V_j)$ according to distribution algo-
   rithm of the SSSD method.
```

Table 6 shows the results of SSMD method applied to the Problem 2. We have used the Mosaic algorithm 1 to generate a mosaic distribution. All results are obtained for $M = 4$.

Table 6
The computational results of the SSMD algorithm for the Problem 2

Processors p	T_{tot}	S_p	E_p
1	915	1.00	1.00
2	628	1.46	0.73
3	435	2.10	0.70
4	381	2.40	0.60
5	413	2.22	0.44
6	399	2.29	0.38
7	405	2.26	0.32
8	404	2.26	0.28

Table 7
The computational results of the SSMD algorithm for the Problem 3

Processors p	T_{tot}	S_p	E_p
1	97.9	1.00	1.00
2	46.4	2.11	1.06
3	31.2	2.27	0.76
4	26.2	3.74	0.93
5	28.8	3.40	0.68
6	25.0	3.92	0.65
7	19.1	5.13	0.73
8	17.3	5.66	0.71

Comparing the obtained results with the numerical results presented above for SSSD method we find out that the mosaic distribution algorithm gives much better parallel performance.

The results obtained with SSMD method for the Problem 3 are given in Table 7. The number of subregions is $M = 8$.

Examples given above show that the SSMD algorithm is more successful than the SSSD algorithm for many multi-dimensional integrals. Nevertheless we can construct counter-examples for which the parallel performance of the SSMD algorithm is as bad as one of the SSSD algorithm or even worse.

7. Dynamic Distribution Algorithm

The main difficulty with the SSSD and SSMD methods stems from load balancing. We have mentioned two primary reasons for the possible bad performance of these methods, i.e., the variation of work load on workstations during computation and the variation of the number of subdivisions required within the different subregions.

Table 8
Influence of the granularity on the algorithm quality for the Problem 3

M	Total pieces	Quality Q	Time T_{tot}
2	82932	1.00	43.9
4	84921	0.98	41.9
8	86989	0.95	40.6
16	96528	0.86	45.0
32	97179	0.85	45.2
64	98557	0.84	45.4
128	100181	0.83	46.7

Table 9
Influence of the granularity on the load balancing for the Problem 4

Number of packets	Time imbalance ΔT	Total time T_{tot}
2	14.6	18.2
4	9.6	16.8
8	2.4	14.1
16	0.7	8.9
32	0.1	11.0
64	0.1	16.7

A solution of this problem can be achieved only enlarging communication volumes of the method. We have noted above that for parallel computers based on distributed workstations communication bandwidth is small and latency times are very large. Hence we must try to improve the computational load balancing among the processors while preserving simultaneously communication volumes as small as possible.

In this section we describe a dynamic subproblem distribution algorithm. We will use the well known parallel algorithm prototype: master-slave algorithm. This general method is apropriate when the amount of work for solving each subproblem is difficult to predict and when slave processes do not have to communicate with one another. Such situation also arises on clusters of workstations with varying loads, even if equal amounts of work are assigned. In this case the time for each processor to complete its task might vary widely.

The master part of the algorithm is given by the following code.

```
1.  Form a pool of tasks $P_j, j = 1, 2, \dots, M$.
2.  Send one problem from the pool of tasks to each slave.
3.  do while ($M$ results are received from slaves)
        receive a local result from the $j$-th slave process;
        receive a local result from the $j$-th slave process;
        accumulate this local sum in a global sum;
        if (the pool of tasks is not empty)
            send the next task $P_k$ to the $j$-th slave.
        else
            send a termination message to the $j$-th slave.
    end do
```

All slave processes execute the following algorithm:

```
    do while (termination message is received)
        receive a message from the master process;
        if (a problem $P_k$ is received)
            calculate a local result for $P_k$;
            send this result to the master;
        end if
    end do
```

The next step in construction of the *dynamic distribution algorithm* is to define the pool of tasks P_j. The analysis in Section 3.4 shows that each problem P_j must be larger than an elementary application of bisection part of the basic integration rule on the selected hyper-rectangular. Taking into account high overheads in startup times for clusters of workstations we consider the method which is refered to as the Static Subdivision Dynamic Distribution (SSDD) method. First we make initial partitioning of the region $\Omega = \Omega_1 \bigcup \Omega_2 \bigcup \cdots \bigcup \Omega_M$. Then a pool of tasks is formed from subproblems $I(f, |\Omega_j)$, $j = 1, 2, \ldots, M$. We do not include any task redistribution mechanism into our algorithm.

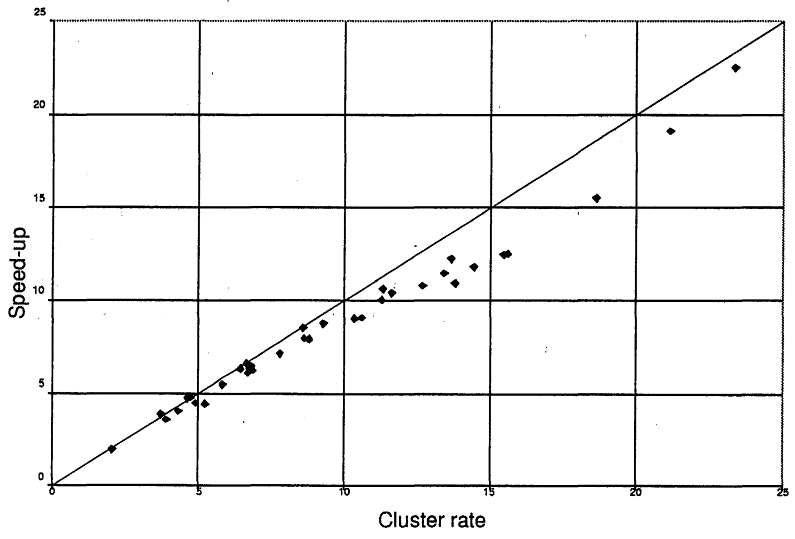

Fig. 6. The speed-up S_p for the Problem 2 on different heterogeneous computer clusters.

The experimental results given in Table 8 show how the size of dynamically distributed packets (granularity) influences computational quality Q and total computational time T for the SSDD algorithm. Two hosts were used in computations.

In the cases when the integrand function has "bad" areas, the grater granularity gives a better load balancing. In Table 9 we present results for the Problem 4, where ΔT denotes the time imbalance

$$\Delta T = T_{tot} - T_{min}. \tag{22}$$

The cluster consisted of two homogeneous workstations.

The computational results given in Fig. 6 show the speed-up S_p for various heterogeneous computer clusters for the Problem 2.

Comparing the results from Fig. 6 with the results from Fig. 4, we can observe the better load balancing of the SSDD algorithm.

References

Bull, J.M., and T.L. Freeman (1994). Parallel algorithms and interval selection strategies for globally adaptive quadrature. In C. Halatsis, D. Maritsas, G. Philokyprou and S. Theodoridis (Eds.), *PARLE'94: Parallel Architectures and Languages, Europe, Lecture Notes in Computer Science*, Vol. 817. Springer Verlag, Berlin. pp. 490–501.

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

Čiegis, R., R. Šablinskas, J. Šimkevičius and J. Waśniewski (1996). Load balancing problem for parallel computers with distributed memory. *Informatica*, 7(3), 281–294.

De Doncker, E., and J. Kapenga (1992). Parallel cubature on loosely coupled systems. In: T.O. Espelid and A. Genz (Eds.), *Numerical Integration*, Kluwer Academic Publishers, Dordrecht, Netherlands. pp. 317–327.

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

Geist, A., A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam (1993). *PVM: Parallel Virtual Machine*. The MIT Press, Cambridge, Massachusetts, London.

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.

Genz, A.C. (1982). Numerical multiple integration on parallel computers. *Comput. Phys. Comm.*, **26**, 349–352.

Genz, A.C. (1990). Subregion adaptive algorithms for multiple integrals. *Contemporary Math.*, **115**, 23–31.

Gladwell, I. (1987). Vectorisation of one dimensional quadrature codes. In G. Fairweather and P.M. Keast (Eds.), *Numerical Integration, Recent Developments, Software and Applications*, NATO ASI Series, Vol. C203. D. Reidel, Dordrecht, Netherlands. pp. 230–238.

Johnson, L.W., and R.D. Riess (1982). *Numerical Analysis*. Addison-Wesley, Reading, MA, 2nd ed.

Lapenga, M., and A. D'Alessio (1993). A scalable parallel algorithm for the adaptive multidimensional quadrature. In R.F. Sinovec, D.E Keyes, M.R. Leuze, L.R.Petzold and D.A. Reed (Eds.), *Proceedings of Sixth SIAM Conference on Parallel Processing*. SIAM, Phyladelphia, PA. pp. 933–936.

Stroud, A.H. (1971). *Approximate calculation of multiple integrals*. Prentice Hall, Englewood Cliffs, New Jersey.

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 the 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 Professor at the Kaunas Vytautas Magnus University and a Professor and a head of Mathematical Modelling Department of Vilnius Technical University. His research interests include numerical methods for nonlinear PDE, parallel numerical methods and numerical modelling in physics, biophysics, ecology.

R. Šablinskas was born in 1971. After having received his master's degree in VMU he has been admited as an engineer in telecommunications company Omnitel. In 1995 he has been admited as a PhD student in Kaunas Vytautas Magnus University. His research interest covers distributed and parallel computing, optimization, neural network models.

J. Waśniewski is a senior researcher at the Danish Computer Center for Research and Education. He has the Degree of Doctor of Mathematics. His scientific interests include parallel computing, mathematical modelling in ecology.

Skaitinio integravimo algoritmai lygiagretiesiems kompiuteriams su paskirstyta atmintimi

Raimondas ČIEGIS, Ramūnas ŠABLINSKAS, Jerzy WAŚNIEWSKI

Šiame darbe nagrinėjami skaitiniai adaptyvūs integravimo algoritmai daugiamačiams integralams skaičiuoti. Šie algoritmai yra skirti lygiagretiesiems kompiuteriams su paskirstytąja atmintimi arba virtualiesiems lygiagretiesiems kompiuteriams, sudarytiems iš grupės kompiuterinių stočių. Skaičiavimuose naudotos PVM ir MPI bibliotekos. Ištirti duomenų perdavimo laiko minimizavimo ir tolygaus skaičiavimų pasiskirstymo tarp procesorių uždaviniai. Nagrinėjami statinis ir dinaminis duomenų paskirstymo tipai. Pateikti skaičiavimo eksperimento rezultatai.