

The Romberg-like Parallel Numerical Integration on a Cluster System

Ali YAZICI

Atilim University, Software Engineering Department

Incek, Ankara Turkey 06586

aliyazici@atilim.edu.tr

Abstract

In this paper, an approximation of a double integral over a triangular region using a Romberg-like buffered extrapolation technique is considered. A master-worker algorithm organization was developed and implemented using the Message Passing Interface (MPI) paradigm which aims at computing the function $f(x,y)$ at the nodes of the triangle by the worker processes in parallel. The algorithm is implemented on a Sun Cluster consisting of 8xX2200 Sun Fire dual core processors using Sun's OpenMPI message passing environment. Here, the Sun Studio Performance Analyzer is utilized to display the runtime behavior of the processes and CPU's involved in the computations. A simple model is developed to predict the speed-up factor.

1. Introduction

In this paper, a master-worker implementation of the Romberg-like extrapolation for the approximation of $Q = \iint f(x,y) dS$ over a triangular region S is reported. The parallel adaptive numerical integration algorithms and their implementations on multiprocessors are given in [1], [2], [3], [4], [9], and [10]. The vectorization of the integration codes in the non-adaptive case is considered in [5] and [11]. In the case of an adaptive process, geometric parallelism is employed where the region of integration is subdivided into sub-regions, which are assigned to available processors. In a non-adaptive setting, however, data parallelism should be employed as a processor farm to exploit inherent parallelism in the evaluation of f .

This article uses data parallelism to describe a new buffered algorithm organization and its implementation on a cluster system using OpenMPI FORTRAN message passing environment [8]. In the forthcoming section the underlying numerical algorithm is described. In Section 3, the outline of the master-worker implementation is provided. Section 4, then, gives a simple model to describe the communication between the master and worker processes, and attempts to measure the performance of the algorithm. In Section 5, some test runs and performance results supported by Sun Studio Performance Analyzer and visualization tool are given. Finally, the results and discussions are given in Section 6.

2. Romberg-Like extrapolation

Consider a triangular region with the vertices (x_1, y_1) , (x_2, y_2) , and (x_3, y_3) . A first-order approximation $V_o^{(0)}$ to Q is:

$$V_o^{(0)} = \frac{A}{3} \sum_{P \in C} f(P) \quad (1)$$

(See [9]). Here, C is the set of vertices and A is the area of S . The vertices of the n_m^2 similar sub-triangles formed by the n_m -fold subdivision of the triangle (Fig.1) using the Romberg subdivision sequence $R = \{1, 2, 4, \dots, 2^k, \dots\}$ (See [12]) create the nodes of the so-called n_m^2 -copy (composite) of $V_o^{(0)}$. Here, n_m is an element of R , and m is the level of subdivision.

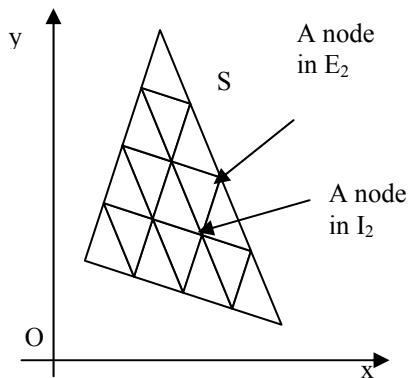


Fig. 1 4-fold subdivision of S

At each level of subdivision of the triangle, in addition to C , the set of vertices of the original triangle, several edge nodes, E_m , and interior nodes, I_m , are generated. The n_m^2 -copy of $V_o^{(0)}$ is:

$$V_m^{(0)} = \frac{A}{3n_m^2} \left(\sum_{P \in C} f(P) + 3 \sum_{P \in E_m} f(P) + 6 \sum_{P \in I_m} f(P) \right) \quad (2)$$

The implementation uses a recursive form of (2) to avoid the duplication of function evaluations. Given the first-order composite approximations, $V_m^{(0)}$, $m = 0, 1, 2, \dots$, one can obtain higher order $(2k+1)$ approximations, $V_m^{(k)}$, of Q by the linear combination of the values of $V_m^{(0)}$ using the recurrence formula as follows:

$$V_m^{(k)} = V_m^{(k-1)} + \frac{V_{m-1}^{(k-1)} - V_m^{(k-1)}}{2^k - 1} \quad (3)$$

where $k=1,2,\dots$ for the depth of extrapolation, and $m=0,1,2,\dots$ for the level of extrapolation.

In this way, higher orders of accuracy are attained by listing the first order composite estimates, and combining them into a tabular form known as Romberg table. Obviously, the total cost of the computation is dominated by the evaluation of the integrand function f at the increasing number of nodes introduced at each level of the subdivision. It is known that, the number of edge nodes in the set E_m is doubled at each step [11]. However, there is 4-fold increase in the number of interior nodes in the set I_m from one subdivision level to the next. For most integrands without any singularities, the maximum level of subdivision is observed to be seven, and the cardinality of set I_m at this level is about 33,000.

The core of the numerical integration algorithm is a triple-nested do-loop in order to generate the nodes of I_m . For d -dimensional integrals, this turns out to be a $d+1$ nested do-loop, causing a considerable number of function evaluations even at the initial levels of subdivision.

The computations (2) and (3) do not lend themselves to parallel processing directly. A parallel approach could nevertheless be to systematically construct buffers holding data coordinates of the nodes of the triangle with a buffer descriptor, and to distribute it to available workers for parallel processing. In this new approach, function evaluations are decoupled from the node generation (computation of node coordinates) so that these two tasks can be synchronized. The master-worker algorithm of the next section deals with this approach by utilizing the MPI paradigm.

3. Master-Worker implementation

The sequential implementation of the extrapolation algorithm is rather obvious. The parallelization of the algorithm is made possible by desynchronizing the function evaluations from node generations (the subdivision of S , and the determination of the coordinates of nodes). In effect, one can have a master process generating the nodes of the subdivisions and dispatching them in buffers to the workers. Then, several workers process all these buffers in parallel. The processing of the function values, the construction of the extrapolation table, and the estimate of Q are all managed by the master process. The message-passing paradigm and SPMD programming model are used below to describe the algorithm in detail.

Algorithm MPI_Romberg

```

Input MaxLevel, BufferSize, Coordinate
of vertices of S
Initialize Buffer Header
Call MPI_BROADCAST to broadcast
MaxLevel, Buffersize, and Vertex
Coordinates
If in Master Process then

```

```

Compute Area A of triangle S
Transform S to unit triangle for
efficient node generation
Compute initial trapezoidal
approximation  $V_0^{(0)}$ 
For all new edge nodes in  $E_m$  Do {
    If BUFFER full then
        Call MPI_SEND(BUFFER) to
        next worker
        CLEAR BUFFER
    else
        Generate next 3 nodes and
        store in BUFFER
    If BUFFER not empty then
        Call MPI_SEND(BUFFER) to
        next worker
    Repeat steps of node generation for
    all new nodes in  $I_m$ 
    Call MPI_BROADCAST to send
    termination FLAG to workers
    Call MPI_RECEIVE intermediate
    results  $V_m^{(k)}$  from workers
    Update Romberg Extrapolation table
    Construct Romberg table to
    approximate Q to the required
    accuracy
else /* in worker process */
    Repeat
        MPI_RECEIVE(BUFFER) from
        master
        Transform unit triangle back to
        S
        Compute f at the nodes
        embedded in the BUFFER
        Update composite values in
        the first column of the
        Romberg table
    Until all BUFFERS are
    processed
    Call MPI_SEND to send intermediate
    composite sums to MASTER
EndIf
End MPI_ROMBERG

```

The results from the workers are collected by the master process and developed to construct the Romberg-table. In the implementation used in this paper, the subdivision level has been fixed. Therefore, even if the error criteria are satisfied at a certain level of subdivision, the subdivision process continues to the maximum level, $MaxLevel$, set by the user. This difficulty can be overcome by modifying the worker process so that the extrapolation table can be formed dynamically.

4. A sample model for computation

Here, a simple model is given to clarify the factors influencing the performance of the MPI_Romberg algorithm. For this purpose, let

$$b \equiv \text{total number of buffers generated}$$

n_p ≡ number of processors

t_n ≡ time to generate a node buffer

t_r ≡ time to process a node buffer (compute f and send results)

t_o ≡ time spent on executing the initialization code in the worker process

t_e ≡ time to form the Romberg table

Different time progress patterns of the master and worker processes are shown in Figures 2 to 4 for three worker processes.

■ idle ■ busy

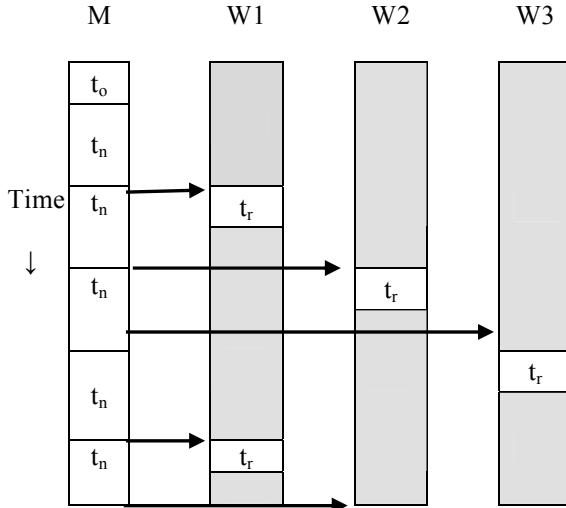


Fig. 2 Message transfer in the round-robin fashion

Case 1: $t_r \ll t_n$

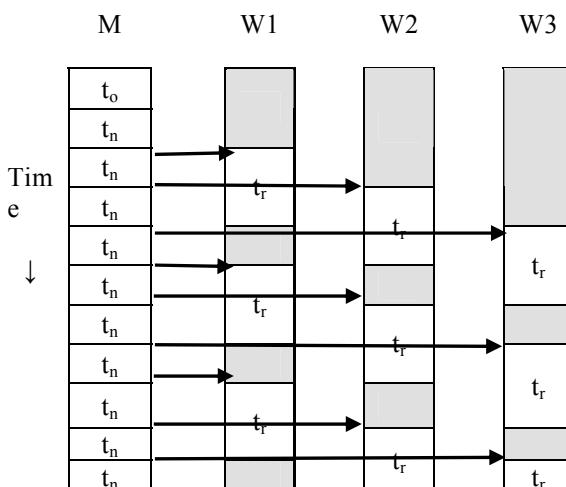


Fig. 3 Message transfer in the round-robin fashion

Case 2: $t_n < t_r < (n_p - 1) t_n$

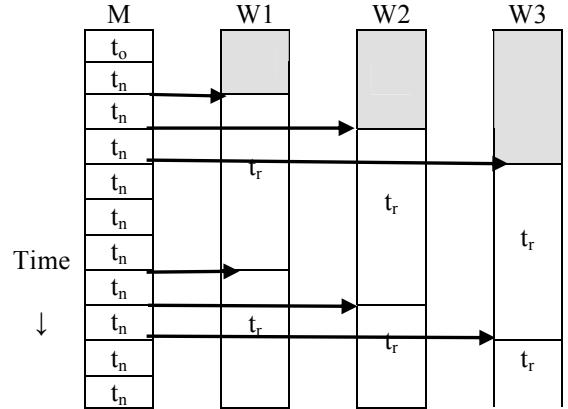


Fig. 4 Message transfer in the round-robin fashion

Case 3: $t_r \geq (n_p - 1) t_n$

In general, Case 1 (Fig. 2) is expected to happen only for very simple integrands. In this case, the worker processes wait idle since the master is not able to provide buffers fast enough. This unfavorable situation causes poor load balancing. For Case 2 (Fig. 3) $t_n < t_r < (n_p - 1) t_n$, a better load balancing can be expected. The ideal situation is shown in Case 3 (Fig. 4), where $t_r \approx (n_p - 1) t_n$. Here, we can observe a uniform distribution of the buffers among the worker processes. For this case, using Fig. 4, one can estimate the execution time of the parallel algorithm T_{par} as follows:

$$T_{par} = t_o + t_e + bt_n + t_r \quad (4)$$

The execution time T_{seq} on a single processor is estimated to be:

$$T_{seq} = t_o + t_e + b(t_n + t_r) \quad (5)$$

The overall idle time is minimized if $n_p - l = t_r / t_n$. Note that, the value of t_r , which varies from one integrand to another, depends on $f(x,y)$. Therefore, for perfect load balancing, it is not possible to determine the number of workers apriori without getting a cost estimate of f . However, based on some complexity heuristics a reasonable allocation can be made. With this choice in mind, that is $n_p = l + t_r / t_n$, the speed-up factor S_{up} is given by:

$$S_{up} = \frac{n_p}{1 + \frac{n_p - 1}{b}} < n_p \quad (6)$$

From (6), one concludes that, for large values of b , the speed-up factor approaches the total number of processors that is n_p . However, for a small value of b , the algorithm's performance becomes degraded. Similar results can be

obtained for the case where $t_r > (n_p - l) t_n$. For very complicated integrands though, the master process may stay idle until one of the slow workers terminates and receives another buffer from the system buffer.

The discussion and the simple formulae derived here are general enough to be used in a variety of message passing systems. For a detailed description of the model, see [9].

5. Mapping of the processes to processors

5.1 The Sun Cluster System [6], [7]

The master-worker algorithm of the preceding section is implemented in the Fortran language using the SPMD approach and tested on the Sun Cluster System consisting of 8xX2200 Sun Fire dual core processors with 2GB DDR2 667Mhz memory and Gigabit Ethernet for message transfer.

Implementation uses blocking MPI_SEND(), MPI_RECV(), and MPI_BCAST() routines of the OpenMPI message passing library. It is assumed that MPI_SEND() first copies the message into a system buffer, and when the control is returned to the sender, it is safe to change the data in the buffer. However, this typical behavior of MPI_SEND() may change according to the size and MPI implementation.

With this configuration, a set of experiments is performed to measure the start-up time and the bandwidth of communication, t_{com} (sec) of transmitting m_{len} bytes between processors. After a regression analysis on the time data collected, the communication time is estimated as follows:

$$t_{com} = 3.383 \times 10^{-5} + 2.6182 \times 10^{-8} m_{len} \quad (7)$$

This result about the communication times seems compatible with that of the actual processing times of the MPI_Romberg algorithm.

5.2 Tests and results

In another set of experiments, the effectiveness of the mapping of the processors to processors (1-16) is measured for integrands with varying computational complexities. The following parameters are used:

- Computational complexity:
 $n_{comp} = \{10, 20, 40, 80\}$,
- Buffer_size: $\{100, 200, 400, 800\}$ in byte s,
- Subdivision levels: MaxLevel = {8, 14} to approximate

$$\int_0^{1-x} \int_0^1 (x^5 + y^5) dy dx \quad (8)$$

It has been observed that buffer sizes used in these experiments (message length) has no significant effect on the performance of the algorithm. Therefore, in Tables 1-2 below only the results for two different lengths are shown. It has been observed that, as expected, higher speed-up values ($s_{up} = (time-n_p)/(time-2-nodes)$) are attained with larger values of n_{comp} at higher levels of subdivision. Under the same conditions, a good level of scalability is also observed. For simple integrands and low levels of subdivision (and extrapolation), the performance of the parallel algorithm is certainly degraded.

5.3 Using the Sun Studio Performance Analyzer

The Sun Studio Performance Analyzer tool is used to assess the performance of the MPI code. The tool consists of a “collector” which gathers performance data by profiling and tracing function calls and a “performance analyzer” to analyze and display the data recorded by the collector.

Table 1. CPU times for MaxLevel = 8

$\int_0^{1-x} \int_0^1 (x^5 + y^5) dy dx$		MaxLevel=8 BufferSize={100,800}Bytes			
n_{comp}	n_p	100	s_{up}	800	s_{up}
10	2	.232E-1	1.00	.237E-1	1.00
	4	.155E-1	1.50	.150E-1	1.58
	8	.628E-2	3.69	.709E-1	3.30
	16	.803E-2	2.89	.839E-2	2.82
20	2	.417E-1	1.00	.399E-1	1.00
	4	.209E-1	2.00	.261E-1	1.53
	8	.850E-2	4.91	.950E-2	4.20
	16	.871E-2	4.79	.934E-2	4.27
40	2	.733E-1	1.00	.736E-1	1.00
	4	.379E-1	1.93	.359E-1	2.05
	8	.143E-1	5.13	.166E-1	4.43
	16	.905E-2	8.10	.118E-1	6.24
80	2	.141E0	1.00	.141E0	1.00
	4	.563E-1	2.50	.552E-1	2.55
	8	.272E-1	5.18	.274E-1	5.15
	16	.259E-1	5.44	.170E-1	8.25

Table 2. CPU times MaxLevel = 16

		MaxLevel=14 BufferSize={100,800}Bytes			
n _{comp}	n _p	100	s _{up}	800	s _{up}
10	2	.810E2	1.00	.805E2	1.00
	4	.336E2	2.41	.318E2	2.53
	8	.169E2	4.79	.158E2	5.13
	16	.191E2	4.24	.172E2	4.68
20	2	.150E3	1.00	.149E3	1.00
	4	.563E2	2.66	.547E2	2.72
	8	.265E2	5.66	.244E2	6.11
	16	.184E2	8.15	.172E2	8.66
40	2	.287E3	1.00	.287E3	1.00
	4	.102E3	2.81	.100E3	2.87
	8	.443E2	6.48	.435E2	6.60
	16	.204E2	14.07	.210E2	13.67
80	2	.561E3	1.00	.561E3	1.00
	4	.192E3	2.92	.192E3	2.92
	8	.851E2	6.59	.832E2	6.74
	16	.409E2	13.72	.388E2	14.46

To visualize different performance characteristics of the implementation, several experiments have been designed and related data collected in the corresponding data sets. The *mpirun* command to run the executable named ‘romberg’ on two processors is:

mpirun -np 2 -host grid01, grid02 romberg

This test is run with n_p=8 processors, at Max_Level=14, for the Complexity=40 using BufferSize=400 Bytes.

Fig.5 shows the visual data collected by the analyzer for this experiment. Here, dashed lines represent the communication points (message transfer) between the processes and the shaded areas represent the busy times of the processors. With this tool, it is possible to investigate the events in a certain time period by magnifying an interval on the time axis. In this experiment, it is observed that master produces and sends buffers at a rate faster than the processes performing the function evaluations. This conforms to the Case 3 of Fig.4 mentioned previously.

6. Conclusions

A master-worker implementation of the extrapolation method of integration for a triangular region is shown to be very effective when mapped to a cluster system. Other regions can be handled by the same algorithm by decomposing the region into triangles and considering each one separately. The algorithm

can be modified to handle higher dimensional integrals.

The cost of sending messages (buffers) back and forth can be ignored. Moreover, the message length appears to be an insignificant factor, especially for integrals with complex integrands.

For a given buffer size, the node generation time (worker process time) is fixed. However, the speed of a worker is very much dependent on the complexity of the integrand. If an estimate of time complexity of f can be made a priori, sufficient number of workers can be allocated to minimize the overall idle time of the computation, thereby leading to perfect load balancing. Currently, this modification is being studied.

The algorithm introduced in this article uses the master-worker approach together with that of explicit buffering to provide a sophisticated and a novel technique of cluster programming. The Romberg-like algorithm introduced here requires the determination of the subdivision level at the beginning leading to a static implementation. The dynamic implementation needs some modifications and is currently under consideration.

References

- [1] K. Burrage, “An adaptive numerical integration code for a chain of transputers”, *Parallel Computing* 16, 1990, pp.305-312.
- [2] R. Ciegis, R. Sablinskas and J. Wasniewski, Applied Parallel Computing – Large Scale Scientific and Industrial Problems, Lecture Notes in Computer Science, 1998, Vol. 1541, pp.71-75
- [3] A. Genz, “The numerical evaluation of multiple integrals on parallel computers”, in: P. Keast and G. Fairweather, Eds., Numerical integration (D. Reidel, Dordrecht, 1987) 219-230.
- [4] A. Genz, “Parallel adaptive algorithms for multiple integrals”, CS-88-183, Computer Science Department Research Report, Washington State University, 1988.
- [5] I. Gladwell, “Vectorization of one dimensional quadrature code”, in: P. Keast and G. Fairweather, eds., Numerical integration (D. Reidel, Dordrecht, 1987, pp. 231-239.
- [6] D. Grigoras, A. Nicolau, B. Toursel and B. Folliot(Eds.). Advanced Environments, Tools, and Applications for Cluster Computing, Springer-Verlag, 2002.
- [7] P. Kacsuk, T. Fahringer, Z. Nemeth (Eds.). Distributed and Parallel Systems: From Cluster to Grid Computing, Springer-Verlag (2008).
- [8] P. Pacheco, *Parallel Programming with MPI*, Morgan Kaufmann Publishers, San Francisco, 1997.
- [9] J. R. Rice, “Parallel algorithms for adaptive quadrature II, metalgorithm correctness”, *Acta Informatica* 5, 1975, pp. 273-285.
- [10] J. R. Rice, “Parallel algorithms for adaptive quadrature III, program correctness”, *ACM TOMS*2, 1976, pp.1-30.

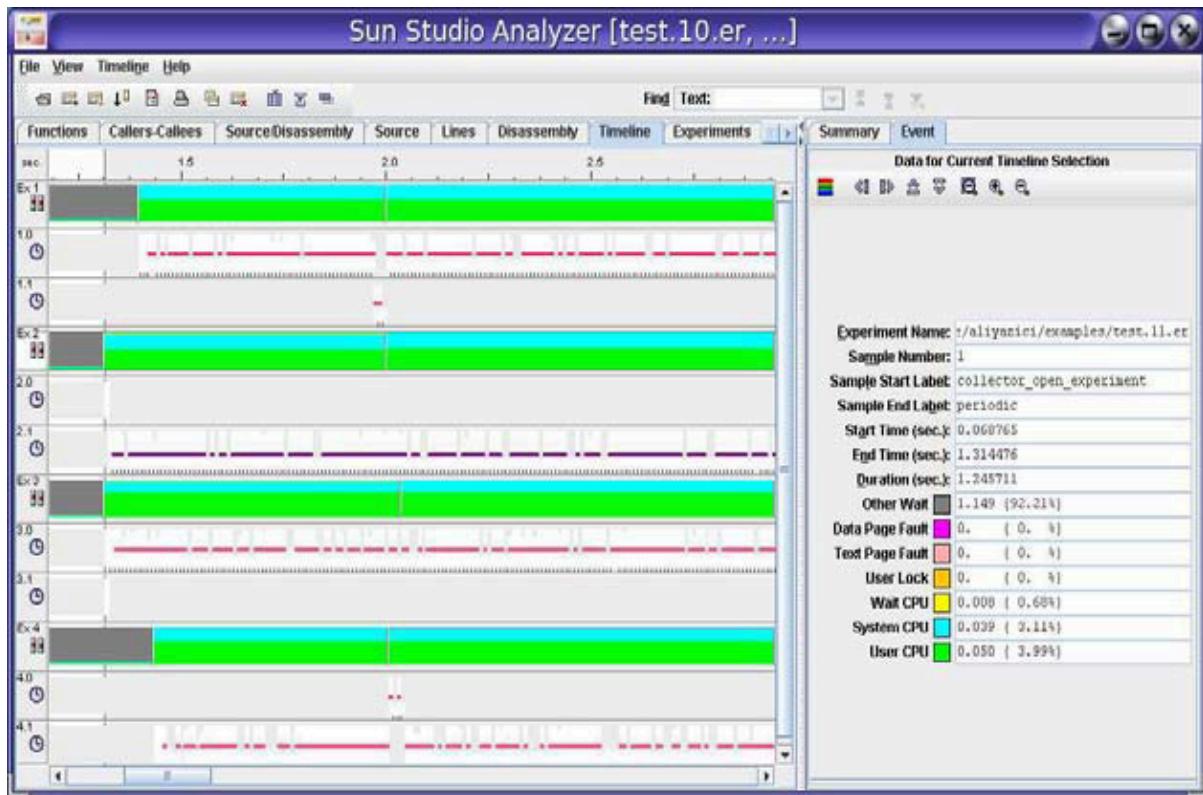


Fig. 5 Communication between processors

[11] R. B. Simpson and A. Yazici, "An organization of the extrapolation method of multidimensional quadrature for vector processing", Parallel Computing 4, 1987, pp.175-188.

[12] A. Yazici, "On the subdivision sequences of extrapolation method of quadrature", METU Journal of Pure and Applied Sciences 23, 1990, pp.35-51.