AIIM

Association for Information and Image Management

1100 Wayne Avenue, Suite 1100
Silver Spring, Maryland 20910
301/587-8202

Centimeter

Inches

MANUFACTURED TO AIIM STANDARDS
BY APPLIED IMAGE, INC.

1 of 1

Trapezoidal Rule Quadrature Algorithms for MIMD Distributed Memory Computers

James N. Lyness, MCS, Argonne National Laboratory, Argonne, IL 60439, USA.
Stephen E Plowman, CMSR, Liverpool University, UK.

Abstract

An approach to multi-dimensional quadrature, designed to exploit parallel architectures, is described. This involves transforming the integral in such a way that an accurate result is given by the trapezoidal rule; and by evaluating the resulting sum in a manner which may be efficiently implemented on parallel architectures. This approach is to be implemented in the Liverpool NAG transputer library.

1. Introduction.

In this paper we describe how approximations for one and N dimensional integrals can be computed on a parallel computer. These algorithms are based on the (product) trapezoidal rule. In §2 we show how this rule can be implemented so as to achieve good load balancing and minimal communication between processors. These factors are critical for achieving good speedups.

It is known that the trapezoidal rule is an efficient numerical integration rule for an integrand which is smooth and satisfies periodic boundary conditions at the lower and upper limits of the integration interval. One way this efficiency can be exploited for an arbitrary integrand is to transform the integrand. In §3 we exploit a transformation (based on the Sag-Szekeres (1964) approach) for this purpose. It appears that many integrable singularities are adequately handled by this method, so long as they are located at a vertex of the integration region.

Some technical difficulties associated with this transformation are discussed in §4. It is shown there that proper care must be taken to avoid both overflow and a possibly embarrassing underflow.

In §5, some simple numerical examples are reproduced from an earlier working paper to confirm the viability of the method.

2. Cyclic Parallel Evaluation of a Product Trapezoidal Rule Sum.

One dimensional Evaluation

The end point trapezoidal rule or rectangle rule for approximating the one-dimensional integral

$$I_g = \int_a^b g(y) dy \quad (2.1)$$

may be expressed in the form

$$Q^{[m,1]}[a,b]g = h \sum_{j=0}^m \theta_{j,m} g(a+jh), \quad h = (b-a)/m, \quad \theta_{j,m} = \begin{cases} 1/2 & j=0 \\ 1/2 & j=m \\ 1 & \text{otherwise} \end{cases} \quad (2.2)$$

where m is the number of panels and $n = m+1$ is the number of mesh-points.

This work was supported in part by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy, under Contract W-31-109-Eng-28.

Correspondence: lyness@mcs.anl.gov

MASTER

The submitted manuscript has been authored by a contractor of the U. S. Government under contract No. W-31-109-ENG-38. Accordingly, the U. S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U. S. Government purposes.

When computing this sum on a serial computer it is conventional to proceed by calculating each term in the order specified by j . On a parallel computer however, we want to partition this problem into a number of tasks so that the computation is evenly shared between the processors (i.e. load balanced) and assign the tasks so that the amount of data that has to be transmitted between the processors is small. To this end we consider more general indexing schemes.

figure 2.1

Let us assume that we have p processors such that $p \leq n$. To effect good load balancing we would like each of the p processors to treat, as far as possible, the same number of function evaluations. Since there is a total of n function evaluations to be carried out, some sort of arrangement by which each processor treats either $[n/p]$ or $[n/p]+1$ function evaluations is clearly indicated. There are several ways of effecting this; two are illustrated in figure 2.1, where $n=9$ and $p=4$. The first, which we term *natural* indexing, partitions the interval and assigns to each subinterval its own processor. The second, which we term *cyclic* indexing, assigns adjacent points to different processors. It does this by assigning to the q -th processor all required function values $g(y_j)$ for which $(j-q-1)/p$ is an integer. To discuss this scheme it is convenient to re-express (2.2) in the form

$$Q^{[m,1]}[a,b]g = h \sum_{q=1}^p S_q \quad (2.3a)$$

where

$$S_q = \sum_{\substack{j=q-1(\text{mod } p) \\ j \in [0,m]}} \theta_{j,m} g(y_j), \quad y_j = a + jh \quad (2.3b)$$

and $j = i(\text{mod } p)$ has its conventional meaning, that is $(j-i)/p$ is an integer. The sums (2.3b) may be computed in parallel on p processors by assigning S_q to the q -th processor. To calculate S_q the q -th processor starts by evaluating $g(y_j)$, with $j = q-1$; then increments j by p to calculate $g(y_j)$,

for the next mesh-point, and continues in this way until j exceeds m . Note that during the calculation of S_q no communication is required between the processors at all. Communication is required only to start the process and at the end to collect together the results. This final communication is small. Note that each processor executes an identical program. Only the initial value of j is different for each processor.

Cyclic indexing has an advantage over natural indexing with regard to the effect on load balancing caused by function evaluations of uneven difficulty. If, in some part of the interval, function evaluation is significantly more or less difficult than in some other parts of the interval an imbalance in load could result if one processor were lucky and another unlucky. This can happen for example near peaks or near boundaries. In these cases cyclic indexing may provide better load balancing than natural indexing. This advantage disappears if all function evaluations are equally difficult. Moreover if their difficulty were distributed randomly, again there would be no advantage.

***N*-dimensional Evaluation**

We now illustrate this in 4 dimensions. The extension to N -dimensions is immediate. Consider a 4-dimensional Cartesian product of trapezoidal rules for evaluating the integral

$$I_g = \int_{a_1}^{b_1} \int_{a_2}^{b_2} \int_{a_3}^{b_3} \int_{a_4}^{b_4} g(\bar{y}) dy_1 dy_2 dy_3 dy_4, \quad y \in \mathbb{R}^4. \quad (2.4)$$

which can be expressed as

$$Qg = h_1 h_2 h_3 h_4 \sum_{j_1=0}^{m_1} \sum_{j_2=0}^{m_2} \sum_{j_3=0}^{m_3} \sum_{j_4=0}^{m_4} g(\bar{y}_{j_1, j_2, j_3, j_4}) \theta_{j_1, m_1} \theta_{j_2, m_2} \theta_{j_3, m_3} \theta_{j_4, m_4} \quad (2.5a)$$

where

$$h_i = (b_i - a_i)/m_i, \quad [\bar{y}_{j_1, j_2, j_3, j_4}]_i = a_i + j_i h_i, \quad \theta_{j, m} = \begin{cases} 1/2 & j = 0 \\ 1/2 & j = m \\ 1 & \text{otherwise} \end{cases}. \quad (2.5b)$$

We re-index this sum using the single index l so that

$$l = j_1 + n_1 j_2 + n_1 n_2 j_3 + n_1 n_2 n_3 j_4; \quad (2.6a)$$

it is straightforward to verify that this mapping is one-to-one and that $l \in [0, L]$, where $L = n_1 n_2 n_3 n_4 - 1$. Now (2.5) can be written as

$$Qg = h_1 h_2 h_3 h_4 \sum_{l=0}^L \theta^{(l)} g(\bar{y}^{(l)}) = h_1 h_2 h_3 h_4 \sum_{q=1}^p S_q \quad (2.6b)$$

where

$$S_q = \sum_{\substack{l=q-1 \pmod{p} \\ l \in [0, L]}} \theta^{(l)} g(\bar{y}^{(l)}), \quad \bar{y}^{(l)} = \bar{y}_{j_1, j_2, j_3, j_4}, \quad \theta^{(l)} = \theta_{j_1, m_1} \theta_{j_2, m_2} \theta_{j_3, m_3} \theta_{j_4, m_4}. \quad (2.6c)$$

We can calculate Qg in (2.6b) in parallel using the scheme proposed for the one-dimensional case. To evaluate each element in the sum, (2.6c) we require (j_1, j_2, j_3, j_4) for each value of l . This can be found by successive division since

$$l = j_1 + n_1(j_2 + n_2(j_3 + n_3 j_4)) \quad (2.6d)$$

It follows that j_1, j_2, j_3, j_4 are the remainders obtained by dividing l successively by n_1, n_2, n_3 . However, it is simpler to increment (j_1, j_2, j_3, j_4) directly. This is done as follows:

$$\begin{aligned} j_1 &\leftarrow j_1 + p \\ \text{if } j_1 > n_1 &\text{ then } j_1 \leftarrow j_1 - n_1; j_2 \leftarrow j_2 + 1; \\ \text{if } j_2 > n_2 &\text{ then } j_2 \leftarrow j_2 - n_2; j_3 \leftarrow j_3 + 1; \\ \text{if } j_3 > n_3 &\text{ then } j_3 \leftarrow j_3 - n_3; j_4 \leftarrow j_4 + 1; \end{aligned} \quad (2.6e)$$

the instruction which updates j_i being repeated if the updated value still exceeds n_i . (This can happen if $n_i < p$.) The q -th processor is provided with its initial point, usually $(q-1, 0, 0, 0)$. After treating this point, it proceeds to treat points determined using (2.6e) until it finds $j_4 > n_4$.

The Mid-Point Trapezoidal Rule.

In some automatic quadrature routines we update an approximation $Q^{[m,1]}g$ to $Q^{[2m,1]}g$. As is well known, this may be done by evaluating only m additional function values. These are those required by the Mid-Point Trapezoidal Rule.

$$Q^{[m,0]}[a,b]g = h \sum_{j=0}^{m-1} g(a + [j + 1/2]h), \quad h = (b-a)/m. \quad (2.7)$$

and using the identity

$$Q^{[2m,1]}[a,b]g = \frac{1}{2} (Q^{[m,1]}[a,b]g + Q^{[m,0]}[a,b]g) \quad (2.8)$$

The mid-point trapezoidal sum in (2.7) can be evaluated in parallel in a manner similar to that described above for the trapezoidal rule.

3. The Integrand Transformation.

The trapezoidal rule, given by (2.2), is an efficient method for approximating integrands which are smooth and naturally periodic with a period which coincides with the integration interval $[a,b]$.

Since an arbitrary integrand $g(y)$ is unlikely to satisfy these conditions we transform the integrand using an approach similar to the Sag-Szekeres method (1964). A corresponding transformation to a finite interval is possible when a or b or both are infinite. The transform is carried out in three steps. In the first step we transform the integration interval

$$\int_a^b g(y) dy = \int_0^1 f(x) dx \quad (3.1a)$$

using a transformation $y = \alpha(x)$ which takes one of the following forms; depending on the nature of the interval we use one of

$$\begin{aligned}
y &= a + (b-a)x, & f(x) &= (b-a)g(a + (b-a)x), & [a, b] \\
y &= a + \frac{1-x}{x}, & f(x) &= x^{-2}g(a + (1-x)/x), & [a, \infty) \\
y &= b - \frac{1-x}{x}, & f(x) &= x^{-2}g(b - (1-x)/x), & (-\infty, b] \\
y &= \frac{1}{1-x} - \frac{1}{x}, & f(x) &= (x^{-2} + (1-x)^{-2})g(1/(1-x) - 1/x), & (-\infty, \infty)
\end{aligned} \tag{3.1b}$$

In (3.1b), a and b stand for finite numbers. Elsewhere they may be infinite depending on context.

In the second step we apply the transform $x = \Phi(t)$ to obtain

$$\int_0^1 f(x) dx = \int_0^1 f(\Phi(t)) \Phi'(t) dt \tag{3.1c}$$

where

$$\begin{aligned}
\Phi(t) &= \frac{1}{2}(1 + \tanh(1/(1-t) - 1/t)) \\
\Phi'(t) &= \frac{1}{2}((1-t)^{-2} + t^{-2})(1 - \tanh^2(1/(1-t) - 1/t))
\end{aligned} \tag{3.1d}$$

The third step is to apply the trapezoidal rule to approximate the integral on the right of (3.1c). We evaluate

$$Q^{[m,1]}(0,1)\Psi \approx \int_0^1 \Psi(t) dt \tag{3.1e}$$

where

$$\Psi(t) = f(\Phi(t))\Phi'(t) \tag{3.1f}$$

Many functions $\Phi(t)$ are suitable; the one chosen above was used by Sag and Szekeres (1964). This was later used by Murota and Iri (1982) who called it the TANH rule and noted that it was a variant of the IMT method of Iri, Moriguti and Takasawa (1970), (see also Davies and Rabinowitz (1984)). (Takahasi and Mori (1973) use the term TANH to refer to a completely different transformation.)

figure 3.1

Further information about the behaviour of these integrand functions near the end points is given below. For the moment we note that $\Phi'(t) \rightarrow 0$ as $t \rightarrow 0$ or $t \rightarrow 1$ very rapidly. This behaviour also occurs for the higher derivatives of $\Phi(t)$. It is not necessary to evaluate $f(0)$ or $f(1)$ even when the integral has end-point singularities. Thus

$$\Psi(0) = \Psi(1) = 0 \quad (3.1g)$$

Naturally, one may apply this method to any integrand function however obtuse. For a wide class of integrands it is an efficient method, giving exponential convergence in the number of function evaluations. This class includes $f(x)$ which has a convergent Fourier series in any interval strictly contained in $[0,1]$. (This implies that $f(x)$ is integrable and allows a wide class of integrable singularities. Excluded are functions of unbounded variation in an internal interval). Note that the conditions stated are on $f(x)$ and not $g(y)$.

In the rest of the paper, remarks about the efficiency or the behaviour of the algorithm apply only to calculations whose integrand is of the class mentioned above.

Higher Dimensions

This approach can also be applied to higher dimensional integrals in the same fashion. For example, if we consider a 2-dimensional integral, then we first transform the integration region using

$$\int_{a_1 a_2}^{b_1 b_2} g(y_1, y_2) dy_1 dy_2 = \int_0^1 \int_0^1 f(x_1, x_2) dx_1 dx_2 \quad (3.2a)$$

where

$$f(x_1, x_2) = g(\alpha_1(x_1), \alpha_2(x_2)) \alpha_1'(x_1) \alpha_2'(x_2), \quad y_1 = \alpha_1(x_1), \quad y_2 = \alpha_2(x_2). \quad (3.2b)$$

and α_1, α_2 are the appropriate one dimensional transforms given by (3.1b). We then apply the transformation

$$\int_0^1 \int_0^1 f(x_1, x_2) dx_1 dx_2 = \int_0^1 \int_0^1 \Psi(t_1, t_2) dt_1 dt_2 \quad (3.2c)$$

where

$$\Psi(t_1, t_2) = f(\Phi(t_1), \Phi(t_2)) \Phi'(t_1) \Phi'(t_2) \quad (3.2d)$$

4. The Numerical Stability of the Algorithm.

The formulas given in §2 and §3 appear to be straightforward to implement. However it turns out that there are several distinct places in this calculation where sensitive programming is required to avoid unnecessary inaccuracy or breakdown due to unexpected overflow or inconvenient underflow. Some of the underlying causes for sensitivity are interrelated. All are connected with function evaluation at or near the integration interval endpoints. This stability problem occurs in each dimension separately.

Definitions.

In the following discussion, it is important to distinguish between the underflow parameter ϵ_u and the machine accuracy parameter ϵ_m . We shall illustrate the discussion by setting $\epsilon_u = 10^{-73}$ and

$\varepsilon_m = 10^{-12}$. This discussion is in the context of a machine with quiet underflow. That is, left to itself, any number too small to be represented is simply replaced by zero.

Transformation function adjustments.

It is a straightforward exercise to program the calculation of $\Phi(t)$, $\overline{\Phi}(t) = 1 - \Phi(t)$ and $\Phi'(t)$ so that each is available to near machine accuracy. Only one exponential call is required to obtain all three. However, when $t \leq 0.01$ and when $t \geq 0.99$, either $\Phi(t)$ or $\overline{\Phi}(t)$ is smaller than ε_u , the underflow parameter, and so cannot be represented in the machine. Normally such a number would be replaced by zero. For reasons which become apparent later, we recommend that these minute numbers are replaced by ε_u . However, when appropriate, we happily allow $\Phi'(t)$ to be replaced by zero. When $t \leq 0.05$ or $t \geq 0.95$, either $\Phi(t)$ or $\overline{\Phi}(t)$ is less than ε_m . Note that all these quantities, however small, are properly calculated to machine accuracy except of course when they are too small to be represented.

Situation at end points, $t = 0$ and 1 .

The final stage of the calculation involves the numerical integration, using the trapezoidal rule, of an integrand function $\Psi(t) = f(\Phi(t))\Phi'(t)$. Since $\Phi'(t) = 0$ at the endpoints, it is obvious that when $f(x)$ is bounded, the end point contribution is zero and can be omitted. But even if $f(x)$ has an integrable singularity at $x = 0$, or $x = 1$, it may be readily verified that, in the limit, the integrand function $f(\Phi(t))\Phi'(t)$ is zero at $t = 0$ and at $t = 1$. At these values of x , function evaluation $f(x)$ is unnecessary.

In theory, $x = \Phi(t)$ is 0 or 1 only when $t = 0$ or 1. $\Psi(t)$ finite elsewhere. So, in an ideal world where there is no calculational error of any sort, we can safely use the trapezoidal rule to approximate the integral, simply ignoring the two end point function values. In practice however, values of $\Phi(t)$ appear which are extraordinarily close to 0 and 1. It is necessary to ensure that, as a result of minute calculational error, these are not replaced in the machine by 0 or 1. If that were to happen, and $f(x)$ happens to have a singularity there, an overflow may occur. To ensure that this cannot occur requires action by both the user, who provides $g(y)$, and by the program. The user must ensure that $f(x)$ does not overflow for all machine attainable numbers in the x interval $(0,1)$. If we assume that $g(y)$ satisfies

$$a, b \text{ finite} \quad |g(y)| < M / (b - a) \text{ for all } y$$

$$a \text{ finite, } b \text{ infinite} \quad |g(y)| < M / 4 \text{ and } (y - a)^2 |g(y)| < M / 4 \text{ for all } y$$

$$\text{both infinite} \quad |g(y)| < M / 8 \text{ and } y^2 |g(y)| < M / 8 \text{ for all } y$$

it can be shown that $|f(x)| < M$. The user may exploit this by choosing M near the overflow parameter and "capping" the integrand function $g(y)$ appropriately.

For its part, the program helps by ensuring that $\Phi(t)$ never attains the values 0 or 1. To this end, one should replace $\Phi(t)$ by $\max\{\Phi(t), \varepsilon_u\}$ near $t = 0$ and by $\min\{\Phi(t), 1 - \varepsilon_u\}$ near $t = 1$. If this is done a program cannot ask for a function value of $f(x)$ or $g(y)$ precisely at an endpoint of its integration interval. Naturally it should not do this anyway.

Corresponding restrictions should be applied independently to $\Phi(t)$. However $\Phi'(t)$ should not be restrained in this way. When this is too small to be represented, it is replaced by zero.

Situation near endpoints.

The above remarks cover the situation at $t = 0$ and at $t = 1$. Next we turn to the situation near these end points. To clarify our ideas an example is in order. We now look at the trapezoidal rule sum

$$\frac{1}{m} \sum_{j=1}^{m-1} [f(\Phi(j/m))\Phi'(j/m)].$$

For integrands $f(x)$ which are respectable, there is a temptation to omit terms for which $\Phi'(t)$ is less than the machine accuracy parameter. Doing this, one omits about 5% of the integration interval at each end. If the program omits the corresponding function evaluation, a 10% economy may ensue. However, there are cases in which unnecessary inaccuracy could arise; if for example $f(x)$ were large very near an endpoint but minuscule elsewhere.

However, there is no justification whatever for this when $f(x)$ has any sort of singular behaviour at either end point. This happens when either a or b are infinite as singular behaviour is induced by the earlier transformation $y = \alpha(x)$. So the only case where an automatic ignoring of these marginal points is permissible is when both a and b are finite and $g(y)$ is reasonably well distributed. We do not recommend omitting any points, except of course the end points.

To illustrate these points, we look at three examples, namely $f(x) = 1$ and $f(x) = x^{-2/3}$ and $f(x) = (1-x)^{-2/3}$. The exact integrals, I_f are 1 and 3 and 3 respectively. We consider the fifty panel trapezoidal rule sum ($m = 50$); in the first two examples we examine the contribution of the three terms $j = 1, 2, 3$ to this trapezoidal rule sum. This is

$$\begin{aligned} & 0.02(f(\Phi(0.02))\Phi'(0.02) + f(\Phi(0.04))\Phi'(0.04) + f(\Phi(0.06))\Phi'(0.06)) \\ &= 0.02(0.14D-38 * f(0.28D-42) + 0.19D-17 * f(0.15D-20) - 0.15D-10 * f(0.28D-13)) \end{aligned}$$

In the first example, $f(x) = 1$ these three terms contribute

$$0.28D-40, 0.38D-19, 0.30D-12 \tag{4.1a}$$

respectively to a sum which is approximately 1. The first two, which are comfortably smaller than ϵ_m , have no effect on the result whatever. If all three and the corresponding three at the other end of the interval are ignored, the result may be compromised by an amount 0.60D-12.

The second example is different. The three function values involved are not all 1, but are large. They are

$$0.23D+29, 0.76D+14 \text{ and } 0.11D+10 \tag{4.1b}$$

respectively. Their respective contributions to the sum is obtained by multiplying them by the weights in (4.1a) above. This gives

$$0.64D-12, 0.29D-5, 0.33D-3 \tag{4.1c}$$

respectively. Terms of this size cannot be routinely omitted simply because one of the factors involved in their evaluation is small. Note that the computer has all these numbers available to machine accuracy (i.e. in this example to twelve decimal places). To make this description easier to read, we

have only written down the first two places in the above discussion.

The third example is again different. Because the singularity is at the end $t = 1$, the situation is much worse. We illustrate this using $f(x) = (1-x)^{-2/3}$. In theory, because of symmetry, one might expect this example to correspond in all significant respects to the previous example. Here the critical points are the final three. We can calculate the weights correctly. These are the same as in (4.1a) above. But $\Phi(t)$ has to be represented in the machine, and has to be less than 1. So, instead of function values (4.1b) we find all three to be about 1.00D+8. The true integrand values (4.1c) are then underestimated leaving an overall error of about 0.60D-5.

Clearly, the calculation at $t = 0$ is more sensitive than at $t = 1$. The user should, if possible arrange that his singularity occurs at the sensitive end of this integration interval. However, in view of the following remarks, the user need not worry about any singularity induced by the transformation from an infinite or semi-infinite interval. This is taken care of by the algorithm.

Special case of infinite a or b.

As mentioned above, in the semi-infinite and infinite cases, the transformation $y = \alpha(x)$ introduces a singularity. These transformations involve x and $\bar{x} = 1 - x$, in a fundamental way. $\bar{x}$ coincides with $1 - \Phi(t)$, which we have called $\bar{\Phi}(t)$. When t is close to 1, $\bar{\Phi}(t)$ is small. In order to retain correct expressions when $t > 0.95$, we cannot rely on taking the difference between 1 and $\Phi(t)$ which is very close to 1. However we may arrange that the program which calculates $\Phi(t)$ also calculates $\bar{\Phi}(t)$ and returns both. So, in formulas such as (3.1b), $\bar{x}$ may be set accurately. Thus we are able to mitigate the effect of the singularity induced by the transformation $y = \alpha(x)$ when a , b or both are infinite.

5. Results

Programs based on the techniques described here have been implemented using transputer arrays at the University of Liverpool; see Plowman (1992). We reproduce some the two dimensional results here.

$$1. \int_2^3 \int_1^\infty x^{-y} dx dy = \ln(2)$$

$$3. \int_0^1 \int_{-\infty}^\infty \frac{y}{(x^2 + 4x + 5)^2} dx dy = \frac{\pi}{4}$$

$$5. \int_0^\infty \int_0^\infty \sqrt{x+y} \exp(-x-y) dx dy = \frac{3\pi}{4}$$

$$7. \int_0^\infty \int_0^\infty \frac{\exp(-x-y) \cos(2\sqrt{xy})}{\sqrt{y}} dx dy = \frac{\sqrt{\pi}}{2}$$

$$2. \int_0^\infty \int_0^\infty \exp(-x^2 - y^2) dx dy = \frac{\pi}{4}$$

$$4. \int_0^1 \int_0^1 \frac{x}{\sqrt{x^2 - y^2}} dx dy = \frac{1}{2} (\ln(\sqrt{2} + 1) + \sqrt{2} - 1)$$

$$6. \int_0^\infty \int_0^\infty \ln(1 + y^2 x^{-2}) \exp(-x-y) dx dy = \frac{\pi}{2}$$

$$8. \int_0^\infty \int_0^\infty \left[\frac{\sqrt{x+y}}{\ln(1 + y^2 x^{-2})} \right] \frac{\cos(2\sqrt{xy})}{\sqrt{y}} \exp(-x-y) dx dy$$

The results for these integrals are shown in table 5.1. They were obtained using a N -dimensional rule-based routine running on a linear array of eight T800 transputers. The times given are in seconds and the number of panels used refer to the number used for each of the x and y components; for these examples both number are the same. For further details of the error estimate etc. the reader should refer to Plowman (1992).

QUAD.MSSNR.REAL64								
problem	1	2	3	4	5	6	7	8
error estimate	1.6E-10	6.5E-12	9.9E-11	6.8E-15	2.2E-12	2.4E-12	7.0E-15	2.4E-12
actual error	2.1E-15	2.3E-15	9.3E-13	8.2E-15	2.5E-15	5.6E-16	8.8E-16	2.5E-15
1 proc time	1.6	4.3	15.1	3.7	4.6	4.8	4.8	6.2
no. of panels	64	128	256	128	128	128	128	128
speedup 4	3.61	3.79	3.87	3.78	3.81	3.81	3.81	3.77
speedup 8	6.03	7.05	7.55	7.08	7.10	7.11	7.11	7.12

table 5.1

6. Concluding Remarks

We have found that in general, simple one dimensional integrals experience slow down rather than speed up using this technique. However, as is evident from table 5.1, two dimensional problems taking more than 4 seconds experience reasonable speed up. The possibility for speed up is clear from the description of the method. These results simply confirm that this occurs.

While our primary aim has been to obtain an algorithm having

1. good load balancing; and
2. low communication,

the sort of algorithm we have described has some additional welcome properties. These are:

3. The number of processors p occurs in the algorithm in a very simple way. These are no restrictions on p , such as being a power of 2.
4. Each processor carries out an identical program, instruction by instruction. Thus the algorithm is suitable for SIMD parallel computers.
5. The low communication cost is almost independent of the configuration of the processors.

Programs based these principles have been written and tested. A short selection of results appear in §5.

References

- P.J. Davies and P. Rabinowitz, (1984), *Methods of Numerical Integration*, 2nd Edition, Academic Press, p. 142-144.
- M. Iri, S. Moriguti and Y. Takasawa (1970), *On a Certain Quadrature Formula*, (in Japanese), *Kokyuroku* of the Res. Inst. for Math. Sci., Kyoto Univ. No. 91, p. 82-118.
- K. Murota and M. Iri (1982), *Parameter Tuning and Repeated Application of the IMT-type Transformation in Numerical Quadrature*, *Numer. Math.* vol. 3, p. 347-363.
- S. Plowman (1992), *Trapezoidal Rule Quadrature Algorithms for MIMD Distributed Memory Computers: Part 2 Implementation*, Supernode II Working Paper, Centre for Mathematical Software Research, University of Liverpool, UK.
- T.W. Sag and G. Szekeres, (1964), *Numerical Evaluation of High-Dimensional Integrals*, *Math. Comp.* vol. 18, p. 245-253.

DATE

FILMED

9 / 8 / 94

END