Aust. J. Phys., 1995, **48**, 19–37

Computational Aspects of the Calculation of the Leading Divergence of a Particle–Particle Ladder

R. Sinclair

HG G32.1, D-MATH, Eidgenössische Technische Hochschule,
CH-8092 Zürich, Switzerland.

Abstract

In order to isolate the leading divergence of a particle–particle ladder graph, we wish to study the most positive eigenvalue, and its eigenfunction, of the operator describing the addition of one rung. We discuss issues of computational efficiency associated with the setting up and solving of the resulting eigenvalue problem, with an emphasis on the use of inherent symmetries to reduce the size of the problem, and the choice of machine and algorithms appropriate for the calculation. In particular, we compare various load-balancing techniques for a problem involving a large number of independent integrals, requiring greatly differing amounts of computer time, which we have implemented on an Intel Paragon massively parallel supercomputer. We find that a stack-based parallel adaptive integration algorithm performs significantly better than a more natural recursive implementation when load-balancing is a priority.

1. Introduction

In this paper we describe the development of a program used for studying the leading divergence of a particle–particle ladder in a (2+1)-dimensional nonrelativistic many-body problem. The numerical results obtained are part of a detailed mathematical analysis of a long-range many-electron system [in collaboration with E. Trubowitz (my Ph.D. supervisor, ETH Zürich), J. Feldman (University of British Columbia), J. Magnen and V. Rivasseau (Ecole Polytechnique, France)]. The motivation for this work is that it has been shown (Khveshchenko *et al.* 1993; Bares and Wen 1993; Halperin *et al.* 1993) that the presence of long-range interactions can result in non-Fermi-liquid behaviour in two dimensions. The algorithms described here have been developed with the expectation that the exact form of the interaction studied may change as the detailed analysis proceeds.

The appropriate energy dispersion is

$$e(\mathbf{k}) = \frac{|\mathbf{k}|^2}{2m} - \mu. \tag{1}$$

Here m is the particle mass, μ is the chemical potential, and $k_F = \sqrt{2m\mu}$ is the Fermi radius. In all of our calculations we have set $\hbar = 1$, $m = 1$ and $\mu = 1/2$ ($k_F = 1$). The zero-temperature noninteracting fermion Green's function is given by (Fetter and Walecka 1971, chapter 3)

0004-9506/95/010019$10.00

$$iG^0(\mathbf{x},t;\mathbf{x}',t') = (2\pi)^{-2}\int d^2\mathbf{k}\,\exp\{i\mathbf{k}\,.\,(\mathbf{x}-\mathbf{x}')\}\exp\{-ie(\mathbf{k})(t-t')\}$$
$$\times\,[\Theta(t-t')\Theta(|\mathbf{k}|-k_F)-\Theta(t'-t)\Theta(k_F-|\mathbf{k}|)]\,. \tag{2}$$

Making use of the fact that $\Theta(|\mathbf{k}|-k_F)=\Theta(e(\mathbf{k}))$, we can rewrite this as

$$iG^0(\mathbf{x},t;\mathbf{x}',t') = i(2\pi)^{-3}\int d^2\mathbf{k}\,\exp\{i\mathbf{k}\,.\,(\mathbf{x}-\mathbf{x}')\}$$
$$\times\int dk_0\,\exp\{-ik_0(t-t')\}\,\frac{1}{ik_0-e(\mathbf{k})}\,, \tag{3}$$

from which the momentum-space propagator can be read off immediately. It is

$$\frac{1}{ik_0-e(\mathbf{k})}\,. \tag{4}$$

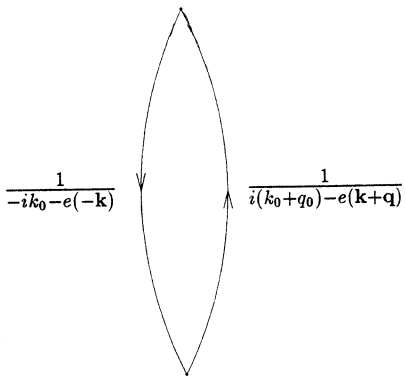

Fig. 1. The random phase bubble. Its value, $\pi(q_0,\mathbf{q})$, is obtained by integrating over k_0 and $\mathbf{k}$.

We take $-\lambda/|\mathbf{k}|^2$ as the long-range interaction. It can be renormalised using the ring approximation (Fig. 1) to the effective two-body interaction (Fetter and Walecka 1971, chapter 12). The random phase bubble (Fig. 2), which appears in the ring approximation, can be approximated by

$$\pi(k_0,\mathbf{k}) = -2\int\frac{d^2\mathbf{q}\,dq_0}{(2\pi)^3}\,\frac{1}{i(k_0+q_0)-e(\mathbf{k}+\mathbf{q})}\,\frac{1}{-ik_0-e(\mathbf{k})}$$
$$= \frac{1}{\pi}\left(1-\frac{|k_0|}{\sqrt{4|\mathbf{k}|^2+k_0^2}}\right)+O(|\mathbf{k}|^2+k_0^2)\,. \tag{5}$$

We formally sum up the ring approximation

$$\frac{-\lambda}{|\mathbf{k}|^2}\sum_{i=0}^{\infty}\left(\frac{-\lambda\pi(k_0,\mathbf{k})}{|\mathbf{k}|^2}\right)^i = \frac{-\lambda}{|\mathbf{k}|^2+\lambda\pi(k_0,\mathbf{k})} \tag{6}$$

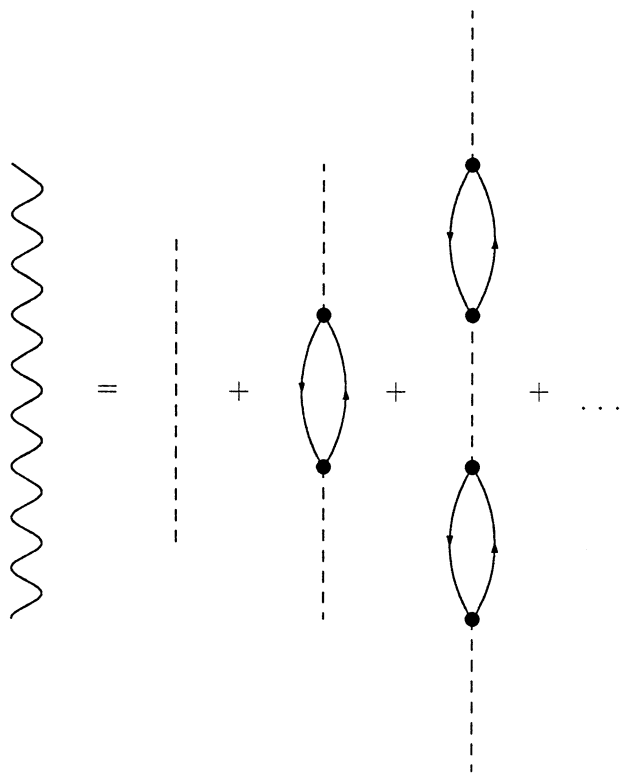

Fig. 2. Ring approximation to the effective two-body interaction. The dashed lines represent the original interaction ($-\lambda/|\mathbf{k}|^2$), the loops are random phase bubbles, and the curved line represents the effective, renormalised interaction.

and, absorbing the factor $1/\pi$ into λ, take

$$\frac{-\lambda}{|\mathbf{k}|^2 + \lambda(1 - |k_0|/\sqrt{4|\mathbf{k}|^2 + k_0^2})} \tag{7}$$

as our effective interaction.

We wish to study ladders created by the repeated addition of rungs of the type depicted in Fig. 3. The operation of adding a rung is given by integration over the newly-created momentum loop, which includes the existing graph (represented by the function V_j) and the new segment

$$\int \frac{-\lambda}{|\mathbf{s}-\mathbf{t}|^2 + \lambda(1 - |s_0 - t_0|/\sqrt{4|\mathbf{s}-\mathbf{t}|^2 + |s_0 - t_0|^2 + M^{2j}})} \times \frac{1}{t_0^2 + e(\mathbf{t})^2 + M^{2j}} V_j(t_0, \mathbf{t}) \frac{d^2\mathbf{t}\, dt_0}{(2\pi)^3}, \tag{8}$$

where M^{2j} acts as a regulariser. Here $M > 1$ and $j \in \mathbf{Z}^-$. Taking j to $-\infty$ corresponds to removal of the cutoff.

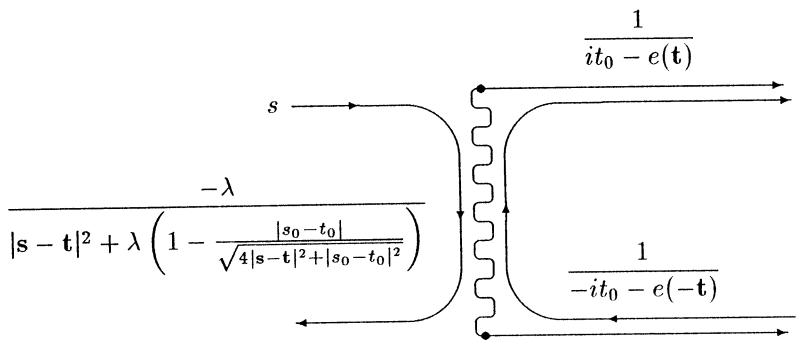

Fig. 3. One rung of the particle–particle ladder.

The dominant properties of a long ladder produced by repeated addition of such rungs can be obtained from the largest eigenvalues, and their eigenfunctions, of the operator in (1). We write the eigenvalue problem as follows:

$$\lambda_j V_j(s_0,|\mathbf{s}|,\theta_s) = \int_0^{\Lambda_k}\int_0^{\Lambda_0}\int_{-\pi}^{\pi} f_j(s_0,|\mathbf{s}|,\theta_s;t_0,|\mathbf{t}|,\theta_t)\, g_j(|t_0|,|\mathbf{t}|) \times V_j(t_0,|\mathbf{t}|,\theta_t)\, d\theta_t\, dt_0\, d|\mathbf{t}|\,, \tag{9}$$

where Λ_k is an ultraviolet cutoff and Λ_0 plays a similar role. We are in fact only interested in the most positive eigenvalue $\lambda_j^\star$ and its eigenfunction $V_j^\star$. The interaction

$$f_j(s_0,|\mathbf{s}|,\theta_s;t_0,|\mathbf{t}|,\theta_t) = -\lambda/\big[|\mathbf{s}|^2+|\mathbf{t}|^2-2|\mathbf{s}||\mathbf{t}|\cos(\theta_s-\theta_t) + \lambda\big(1-|s_0-t_0|/\sqrt{4|\mathbf{s}|^2+4|\mathbf{t}|^2-8|\mathbf{s}||\mathbf{t}|\cos(\theta_s-\theta_t)+|s_0-t_0|^2+M^{2j}}\big)\big] \tag{10}$$

and the function

$$g_j(|t_0|,|\mathbf{t}|) = \frac{|\mathbf{t}|}{(t_0^2+e(\mathbf{t})^2+M^{2j})(2\pi)^3} \tag{11}$$

have several symmetries, which we will use to reduce the size of the problem before attempting to discretise:

$$f_j(s_0,|\mathbf{s}|,\theta_s;t_0,|\mathbf{t}|,\theta_t)\, g_j(|t_0|,|\mathbf{t}|) = f_j(-s_0,|\mathbf{s}|,\theta_s;-t_0,|\mathbf{t}|,\theta_t)\, g_j(|-t_0|,|\mathbf{t}|) \tag{12}$$

$$= f_j(s_0,|\mathbf{s}|,-\theta_s;t_0,|\mathbf{t}|,-\theta_t)\, g_j(|t_0|,|\mathbf{t}|) \tag{13}$$

$$= f_j(s_0,|\mathbf{s}|,\theta_s+\phi;t_0,|\mathbf{t}|,\theta_t+\phi)\, g_j(|t_0|,|\mathbf{t}|)\,. \tag{14}$$

The symmetries (12) and (14) imply that the eigenfunctions of (2) take the following form:

$$V_j(t_0, |\mathbf{t}|, \theta_t) = \left(\frac{t_0}{|t_0|}\right)^p \mathcal{V}_j^{p,\ell}(|t_0|, |\mathbf{t}|) \exp(i\ell\theta_t)\,, \tag{15}$$

where $p \in \{1, 2\}$ and $\ell \in \mathbf{Z}$. We find that the eigenvalue problem (2) splits up into infinitely many lower-dimensional eigenvalue problems, indexed by p and ℓ. In real, symmetric form these are

$$\lambda_j^{p,\ell} \sqrt{g(|s_0|, |\mathbf{s}|)}\, \mathcal{V}_j^{p,\ell}(|s_0|, |\mathbf{s}|) = \int \mathcal{F}_j^{p,\ell}(|s_0|, |\mathbf{s}|; |t_0|, |\mathbf{t}|) \sqrt{g(|s_0|, |\mathbf{s}|)\, g_j(|t_0|, |\mathbf{t}|)}$$
$$\times \sqrt{g_j(|t_0|, |\mathbf{t}|)}\, \mathcal{V}_j^{p,\ell}(|t_0|, |\mathbf{t}|)\, d|t_0|\, d|\mathbf{t}|\,, \tag{16}$$

where

$$\mathcal{F}_j^{p,\ell}(|s_0|, |\mathbf{s}|; |t_0|, |\mathbf{t}|) = \tag{17}$$
$$\int \{f_j(|s_0|, |\mathbf{s}|, 0; |t_0|, |\mathbf{t}|, \theta_t) + (-1)^p f_j(|s_0|, |\mathbf{s}|, 0; -|t_0|, |\mathbf{t}|, \theta_t)\} \exp(i\ell\theta_t)\, d\theta_t\,.$$

2. Discretisation

We now have a set of eigenvalue problems of the form

$$\lambda_j^{p,\ell}\, x_j^{p,\ell}(|s_0|, |\mathbf{s}|) = \int_0^{\Lambda_k} \int_0^{\Lambda_0} A_j^{p,\ell}(|s_0|, |\mathbf{s}|; |t_0|, |\mathbf{t}|)\, x_j^{p,\ell}(|t_0|, |\mathbf{t}|)\, d|t_0|\, d|\mathbf{t}|\,. \tag{18}$$

We divide up the domain of integration into cells $\square_{m,n}$ $(m, n \in \{1, ..., N\})$ such that

$$\square_{m,n} \cap \square_{m',n'} = \theta \quad \text{for } (m, n) \neq (m', n') \tag{19}$$

and

$$\bigcup_{(m,n)} \square_{(m,n)} = [0, \Lambda_k] \times [0, \Lambda_0]\,. \tag{20}$$

We choose to define the $\square_{m,n}$ in such a way that $|t_0|$ and $e(\mathbf{t})$ take on values from geometric series with the factor $M^{1/\mathtt{ndiv}}$ (to allow an adjustment of the fineness of the grid), and starting values proportional to M^j (the regularisation only ensures that f_j and g_j are effectively constant for values of $e(\mathbf{t})$ and t_0 much smaller than M^j, so M^j is a natural 'unit length' to use in constructing the mesh). The point of discretising in this way is to attempt to construct cells in which f_j and g_j are approximately constant, so that integration over any given cell will be as simple as possible. We have set $M = 2$, $\Lambda_0 = 1$ and $\Lambda_k = \sqrt{2}$ for the calculations to be described here. Now

$$\square_{m,n} = I_m \times J_n \,, \tag{21}$$

where

$$I_m = \begin{cases} [\ \sqrt{1-2^{(1/\mathtt{ndiv}+j-3)}}\ ,\ 1\ ] & (m=1) \\ [\ \sqrt{1-2^{(m/\mathtt{ndiv}+j-3)}}\ ,\ \sqrt{1-2^{((m-1)/\mathtt{ndiv}+j-3)}}\,] & (1<m\le \mathtt{ndiv}.(3-j)) \\ [\ 1\ ,\ \sqrt{2^{(1/\mathtt{ndiv}+j-3)}+1}\ ] & (m=\mathtt{ndiv}.(3-j)+1) \\ [\sqrt{2^{((m-1)/\mathtt{ndiv}+2(j-3))}}+1,\ \sqrt{2^{(m/ndiv+2(j-3))}}+1\ ] & \\ \qquad\qquad (\mathtt{ndiv}.(3-j))+1<m\le 2.\mathtt{ndiv}.(3-j) \end{cases} \tag{22}$$

and

$$J_n = \begin{cases} [\ 0\ ,\ 2^{(1/\mathtt{ndiv}+j-3)}\ ] & (n=1) \\ [\ 2^{((n-1)/\mathtt{ndiv}+j-3)}\ ,\ 2^{(n/\mathtt{ndiv}+j-3)}\ ] & (1<n\le \mathtt{ndiv}.(3-j)). \end{cases} \tag{23}$$

Defining

$$N = \mathtt{ndiv}.(3-j)\,, \tag{24}$$

we have $m \in \{1,...,2N\}$ and $n \in \{1,...,N\}$. An example of such cells $\square_{m,n}$ is depicted in Fig. 4.

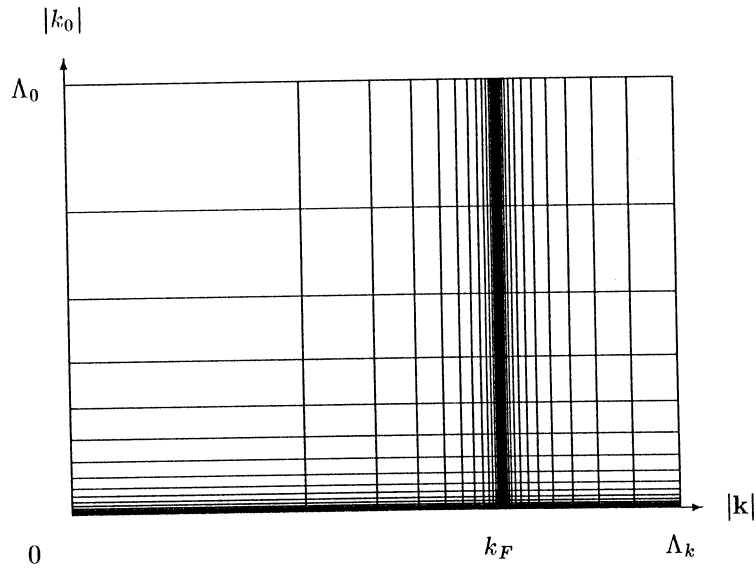

Fig. 4. An example of the division of $[0,\Lambda_k] \times [0,\Lambda_0]$ into cells $\square_{m,n}$ ($j=-6$, $\mathtt{ndiv}=2$ and $\ell=0$).

The matrix elements are given by

$$\bar{A}_j^{p,\ell}(m',n';m,n) = \int_{\square_{m',n'}} \int_{\square_{m,n}} A_j^{p,\ell}(|s_0|,|\mathbf{s}|;|t_0|,|\mathbf{t}|)\, d|s_0|\, d|\mathbf{s}|\, d|t_0|\, d|\mathbf{t}| \quad (25)$$

$$= \bar{A}_j^{p,\ell}(m,n;m',n')\,. \quad (26)$$

3. Evaluation of Individual Matrix Elements

The $4N^4$ matrix elements, in terms of the functions (10) and (11), are

$$\bar{A}_j^{p,\ell}(m',n';m,n) =$$

$$2 \int_{\square_{m',n'}} \int_{\square_{m,n}} \int_0^\pi \{f_j(|s_0|,|\mathbf{s}|,0;|t_0|,|\mathbf{t}|,\theta_t) + (-1)^p f_j(|s_0|,|\mathbf{s}|,\theta_s;-|t_0|,|\mathbf{t}|,\theta_t)\}$$

$$\times \cos(\ell\theta_t)\, d\theta_t \sqrt{g(|s_0|,|\mathbf{s}|)\, g_j(|t_0|,|\mathbf{t}|)}\; d|s_0|\, d|\mathbf{s}|\, d|t_0|\, d|\mathbf{t}|\,. \quad (27)$$

They are real due to the symmetry (6). We need only consider nonnegative ℓ. These are five-dimensional integrals.

We first consider the integral over θ_t, making the following approximation:

$$f_j(|s_0|,|\mathbf{s}|,0;\pm|t_0|,|\mathbf{t}|,\theta_t) \approx \frac{-\lambda}{a(|s_0|,|\mathbf{s}|;\pm|t_0|,|\mathbf{t}|) + b(|s_0|,|\mathbf{s}|;\pm|t_0|,|\mathbf{t}|)\,\theta_t^2}\,, \quad (28)$$

where

$$a(|s_0|,|\mathbf{s}|;\pm|t_0|,|\mathbf{t}|) = |\mathbf{s}|^2 + |\mathbf{t}|^2 - 2\,|\mathbf{s}|\,|\mathbf{t}|$$

$$+ \lambda\left(1 - \frac{\big||s_0| - (\pm|t_0|)\big|}{\sqrt{4\,|\mathbf{s}|^2 + 4\,|\mathbf{t}|^2 - 8\,|\mathbf{s}|\,|\mathbf{t}| + \big||s_0| - (\pm|t_0|)\big|^2 + 2^{2j}}}\right) \quad (29)$$

and

$$b(|s_0|,|\mathbf{s}|;\pm|t_0|,|\mathbf{t}|) = |\mathbf{s}|\,|\mathbf{t}|$$

$$+ \frac{2\lambda\,\big||s_0| - (\pm|t_0|)\big|\,|\mathbf{s}|\,|\mathbf{t}|}{\left(4\,|\mathbf{s}|^2 + 4\,|\mathbf{t}|^2 - 8\,|\mathbf{s}|\,|\mathbf{t}| + \big||s_0| - (\pm|t_0|)\big|^2 + 2^{2j}\right)^{\frac{3}{2}}}\,. \quad (30)$$

This approximation is attractive because it allows one to exploit the results

$$\int_0^\pi \frac{-\lambda}{a + b\,\theta_t^2}\, d\theta_t = \frac{-\lambda}{\sqrt{ab}} \arctan\left(\pi\sqrt{\frac{b}{a}}\right) \quad (31)$$

and

$$\int_0^\infty \frac{-\lambda}{a+b\theta_t^2}\cos(\ell\theta_t)\,d\theta_t = \frac{-\lambda\pi}{2\sqrt{ab}}\exp\left(-\sqrt{\frac{a}{b}}\ell\right). \tag{32}$$

For $\ell = 0$, we add a numerical approximation (using Simpson's rule) of

$$\int_0^\pi \left\{ f_j(|s_0|,|\mathbf{s}|,0;\pm|t_0|,|\mathbf{t}|,\theta_t) - \frac{-\lambda}{a_\pm + b_\pm\theta_t^2}\right\} d\theta_t \tag{33}$$

to (31) for the approximate value of the integral, and also use the numerical approximation of (33) as the error estimate.

For $l \neq 0$, (32) suffices as an approximation to the required integral. The error estimate is composed of two parts. One is due to the use of the original approximation (28), which can be estimated using the absolute value of the numerical approximation to (33). The other contribution is from the extension of the domain of integration from $[0,\pi]$ to $[0,\infty]$. An analytical estimate of this source of error can be obtained using the inequality

$$\left|\int_\pi^\infty \frac{\cos(\ell\theta_t)\,d\theta_t}{a+b\theta_t^2}\right| = \left|\int_0^\infty \frac{\cos(\ell\theta_t)\,d\theta_t}{a+b(\theta_t+\pi)^2}\right| \le \left|\int_0^\infty \frac{\cos(\ell\theta_t)\,d\theta_t}{(a+b\pi^2)+b\theta_t^2}\right| \tag{34}$$

and the result in (11) (where a is replaced by $a+b\pi^2$). Both sources of error are summed.

The remaining integrals are four-dimensional. We choose an adaptive grid technique based on a formula we derive by demanding that polynomials of the form

$$\begin{aligned} h(x,y,u,w) = {} & c_1 + c_2x^2 + c_3y^2 + c_4u^2 + c_5w^2 + c_6x^4 + c_7y^4 + c_8u^4 + c_9w^4 \\ & + c_{10}x^2y^2 + c_{11}x^2u^2 + c_{12}x^2w^2 + c_{13}y^2u^2 + c_{14}y^2w^2 + c_{15}u^2w^2 \end{aligned} \tag{35}$$

(odd powers are irrelevant here) be integrated exactly by

$$\begin{aligned} & \int_{-\frac12}^{\frac12}\int_{-\frac12}^{\frac12}\int_{-\frac12}^{\frac12}\int_{-\frac12}^{\frac12} h(x,y,u,w)\,dx\,dy\,du\,dw \\ & = \alpha\, h(0,0,0,0) \\ & \quad + \beta \sum_{\sigma_x\in\{-1,1\}}\sum_{\sigma_y\in\{-1,1\}}\sum_{\sigma_u\in\{-1,1\}}\sum_{\sigma_w\in\{-1,1\}} h(z_1\cdot\sigma_x, z_1\cdot\sigma_y, z_1\cdot\sigma_u, z_1\cdot\sigma_w) \\ & \quad + \delta \sum_{\sigma\in\{-1,1\}} \{f(z_2\cdot\sigma,0,0,0) + f(0,z_2\cdot\sigma,0,0) + f(0,0,z_2\cdot\sigma,0) \\ & \qquad\qquad + f(0,0,0,z_2\cdot\sigma)\}. \end{aligned} \tag{36}$$

Table 1. Parameters for integration

z_1	0·33709993123162104312	0·5
z_2	0·5	0·3162277660168379332
α	0·10666666666666666667	−1·33333333333333333
β	0·33611111111111111111	0·0069444444444444444
δ	0·04444444444444444444	0·2777777777777777778

We solve equation (36) for z_1, z_2, α, β and δ. The solution set is one-dimensional. We use two different solutions of (13) (see Table 1). The mean of the two approximations to the integral is used as the value, half of the absolute value of their difference as an error estimate. The presence of a negative weight (one of the α values), although in general undesirable (Stroud 1971, section 1.5), is necessary since we want reliable error estimates for functions with peaks which behave as

$$e(x,y,u,w) = \frac{C_1}{(x-u)^2 + \exp\gamma} + \frac{C_2}{(y-w)^2 + \exp\gamma}, \tag{37}$$

so that the behaviour of f_j for $s_0 \approx t_0$ or $|\mathbf{s}| \approx |\mathbf{t}|$ is well covered. In Fig. 5 we compare the actual error incurred by (36) (the function (37) can also be integrated over $[-1/2, 1/2]^4$ analytically) with the error estimate discussed above. The error estimate is reliable in the sense that it is consistently larger than (roughly double) the actual error for the functions (37).

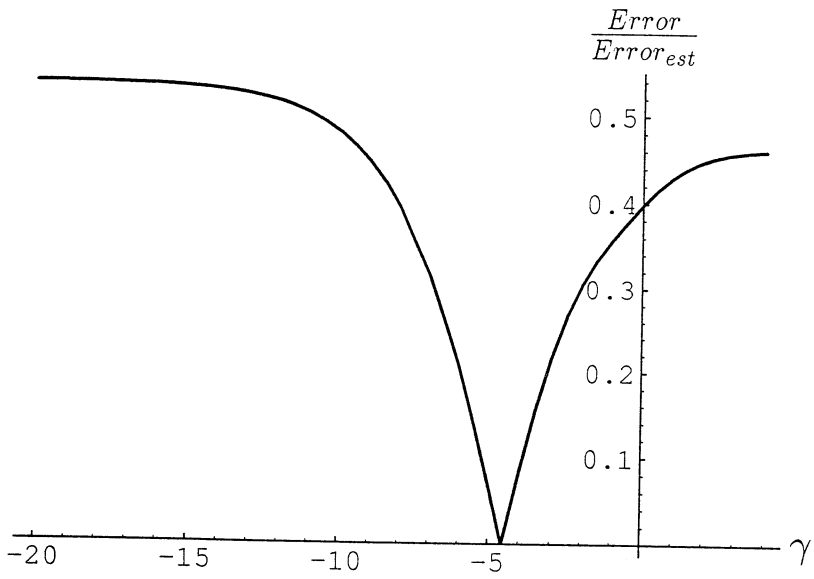

Fig. 5. Ratio of actual error to estimated error (should be less than or equal to one) of the numerical estimate of $\int 1/((x-u)^2 + \exp\gamma)\, dx\, dy\, du\, dw$ as a function of γ.

The estimates of error from the analytical approximations of the integral over θ_t are integrated over simultaneously with the function values. This integrated error is added to the estimate from the adaptive grid integration.

4. Preliminary Calculations

We implemented the approximations discussed above, with a fixed, coarse grid (i.e. no recursion and `ndiv` = 1) on a SUN10 workstation, using the QR method (Wilkinson 1978, chapter 8) to find all eigenvalues of certain matrices $\bar{A}_j^{p,\ell}$. Error estimates were not calculated to save time. We report on consistency checks below, in which we used the final version of the program to confirm assumptions made in this preliminary stage. The results of this first calculation (see Figs 6 and 7) were surprising for three reasons:

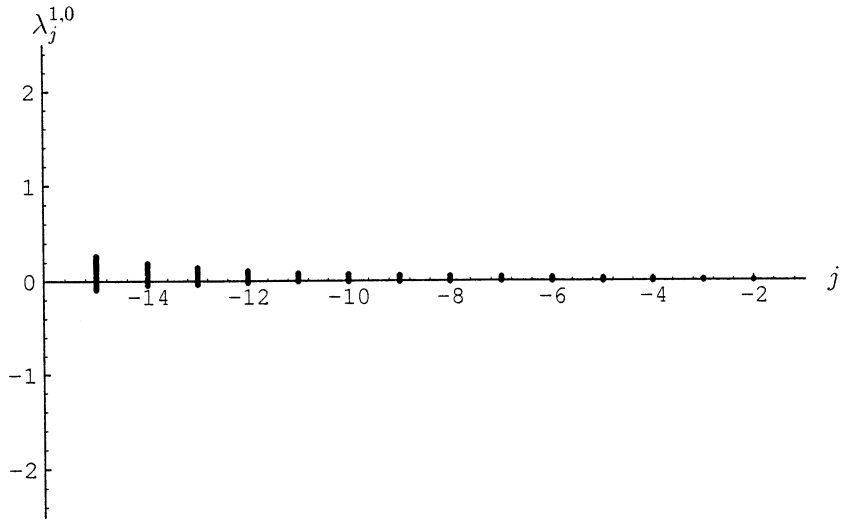

Fig. 6. All eigenvalues for $p = 1$ and $\ell = 0$ from preliminary calculations. The spectrum is discrete for each j.

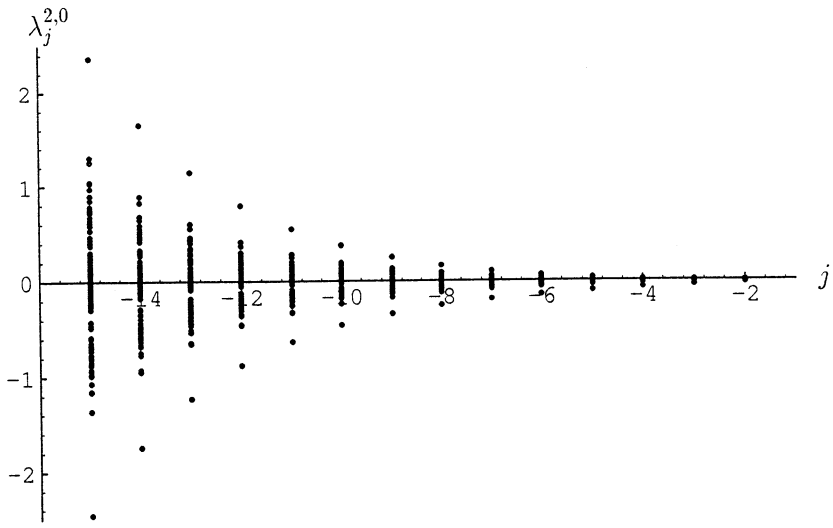

Fig. 7. All eigenvalues for $p = 2$ and $\ell = 0$ from preliminary calculations.

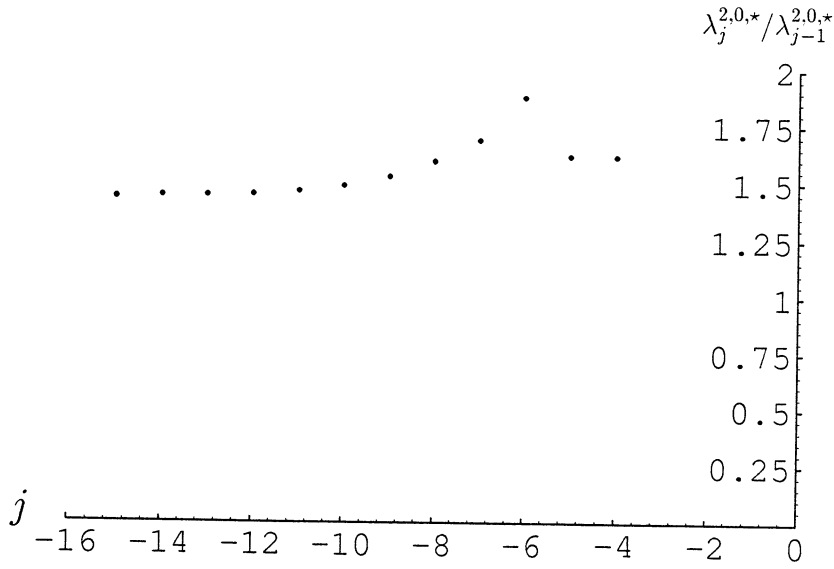

Fig. 8. Ratio of the most positive eigenvalue for a given j to the most positive eigenvalue at $j-1$ from preliminary calculations. Exponential behaviour is apparent for $j < -10$.

(i) The most positive eigenvalue for a given j already scales exponentially with j for $j < -10$ (see Fig. 8). This permits one to assume that the trends visible in these results apply to all $j \ll -10$. On the basis of this assumption, and the other preliminary data, one is justified in no longer considering the eigenvalues for $p = 1$ as candidates for the most positive eigenvalue. This immediately halves the size of the computation.

(ii) The most positive eigenvalue is well isolated from all the others. This allows one to use the shifted power method (Wilkinson 1978, chapter 9, section 4) to find the most positive eigenvalue and its associated eigenvector.

(iii) The evaluation of the matrix elements dominated the computing time, even allowing for the fact that the symmetry of the matrix means that only $2(N^4 + N^2)$ of the $4N^4$ elements need actually be calculated.

5. Parallel Implementation

The preliminary calculations also made it apparent that a workstation would not suffice for more serious computations. The time required to calculate the eigenvalues to a low accuracy for only one value of j was of the order of a few days. The problem lends itself to a massively parallel computer with nodes capable of running independently (a MIMD machine, see Akl 1989, section 1.2.4), since the most computationally intensive part is the evaluation of the very many independent integrals. It was therefore natural to make use of the Intel Paragon (Gates and Peterson 1994) at the ETH Zürich. This is a MIMD machine with 96 computing nodes (processors), based on the `i860` microprocessor. Each processor has a local memory of 32 Mbytes, and is connected to a two-dimensional communication mesh. Message-passing routines are provided (for an introduction, see Intel 1993).

Three restrictions dominated the choice of algorithms:

(i) A large calculation ($j = -20$, `ndiv` $= 2$) would involve 9×10^6 matrix elements (70 Mbyte, when working to double precision). Since no single processor has enough local memory to store the full matrix, the algorithm will need to work with a matrix distributed amongst the available processors.

(ii) Calculating all integrals to a given accuracy using an adaptive method will typically result in widely varying completion times. A second preliminary calculation using a recursive algorithm similar to the one presented in Fig. 9 confirmed this in our case. The algorithm will therefore need some load-balancing heuristics to provide an even distribution of the work amongst the processors.

(iii) Despite the presence of the communication mesh, interprocessor communication is expensive compared to floating-point operations. The algorithm must avoid communication where possible.

```
procedure r_integrate(a,b):
    approximate the integral by t ≈ ∫_a^b f(x)dx;
    if the estimated error is tolerable then return t;
    else return   integrate(a,(a+b)/2)+integrate((a+b)/2,b).
```

Fig. 9. A recursive algorithm for evaluating the one-dimensional integral $\int_a^b f(x)\,\mathrm{d}x$.

The four-dimensional extension of the procedure `r_integrate` (Fig. 9) subdivides the domain of integration into 2^4 subdomains rather than 2, but is otherwise identical. It is termed recursive because, unless the integral is trivial, it does not immediately return an answer, rather it calls (creates copies of) itself again (2^4 times), giving each copy of itself a subdomain of integration. These copies may also reproduce themselves, producing a tree structure in memory. The process continues until a copy actually has a trivial (the estimated error is tolerable, see Fig. 9) integral. It does not reproduce itself, rather it returns its result to the routine which called it, and is deleted from memory. Our implementation of `r_integrate` is based closely on the discussion of adaptive Newton–Cotes integration in Davis and Rabinowitz (1975, section 6.2.3).

Taking the above considerations into account, an outline of the general strategy for the entire calculation can be given. The central idea is a gradual shift from coarse-grained parallelism (where each processor treats only its assigned integrals) to ever finer-grained parallelism (the sharing of integrals or subdomains of integration), such that a reasonable compromise between reducing communication and load-balancing is achieved. To ensure that memory requirements be met, the matrix elements are assigned evenly over the available processors. All communication is to be through a specified 'master' processor, which keeps track of the states of individual processors. The general strategy is as follows:

(1) All integrals that meet the prescribed error tolerance without recursion are evaluated and stored.

(2) Each processor begins evaluating the remaining, nontrivial integrals assigned to it.

(3) Any processor which finishes early can ask for a further integral. When completed, the result is sent back to the processor it was originally assigned to for storage.
(4) If some processors are idle, but the others are already evaluating their last integral, these integrals are spread over a number of processors (by dividing up the domain of integration). Partial results are always returned to the processor to which the integral was originally assigned.
(5) If all processors are finished, and all partial results have been returned, calculation of the most positive eigenvalue and its eigenvector can commence, using the shifted power method.

In step 4, load-balancing is implemented by dividing up domains of integration rather than cells ($\square_{m,n}$). This is necessary, since an increase in the number of cells would effectively increase the size of the matrix $\bar{A}$, requiring still more memory. What remains to be specified is how the domains of integration should be divided up to avoid unnecessary communication. The master processor will receive only one request at a time for work from an idle processor. It will then send this request on to a processor which it knows is still active. This active processor must give up some subdomain of integration it has not yet worked on. An obvious extension to the procedure `r_integrate` (Fig. 9) which allows for this is `pr_integrate` (Fig. 10). Having been given an interval to process, a previously idle processor will apply this same recursive procedure to it until completely finished.

```
procedure pr_integrate(a,b):
    approximate the integral by t ≈ ∫_a^b f(x)dx;
    if the estimated error is tolerable then return t;
    else if another processor needs work then
        give the other processor integrate(a,(a+b)/2) to do;
        return integrate((a+b)/2,b);
    else return integrate(a,(a+b)/2)+integrate((a+b)/2,b).
```

Fig. 10. A recursive algorithm for evaluating the one-dimensional integral $\int_a^b f(x)\,\mathrm{d}x$, adapted to allow subdivision of the domain of integration amongst many processors.

Some runs were made to compare the routines `r_integrate` and `pr_integrate`, against each other, and against some more primitive algorithms, where either step 4 or both steps 3 and 4 as described above were not implemented. The time (T) required to tackle a relatively trivial problem ($j = -2$, `ndiv` $= 1$ and $\ell = 0$) was measured as a function of the number of processors (n). We define the computational 'efficiency' (see Akl 1989, section 1.3.3) to be

$$E(n) = T(1)/n\,.T(n)\,. \tag{38}$$

[The condition $E(n) \leq 1$ is almost always satisfied in a real calculation, as $T(1)/n \leq T(n)$ is usually true. Exceptions, known as 'superlinear speedup', are most often due to the fact that a single processor can work at its fastest if all the necessary data are to be found in a very small area of memory. Since the amount of data to be handled by each processor can be inversely proportional

to n, it is possible that $T(n) < T(1)/n$. The programs we are comparing have, however, a large ratio of floating-point calculations to memory accesses, so this effect does not appear in our plots.] The results are plotted in Fig. 11. The large spread of the data points is due to the fact that the need for any redistribution of work (load-balancing), involving communication, is entirely determined by the original assignment of integrals over the available processors. This is in turn determined by the number of processors.

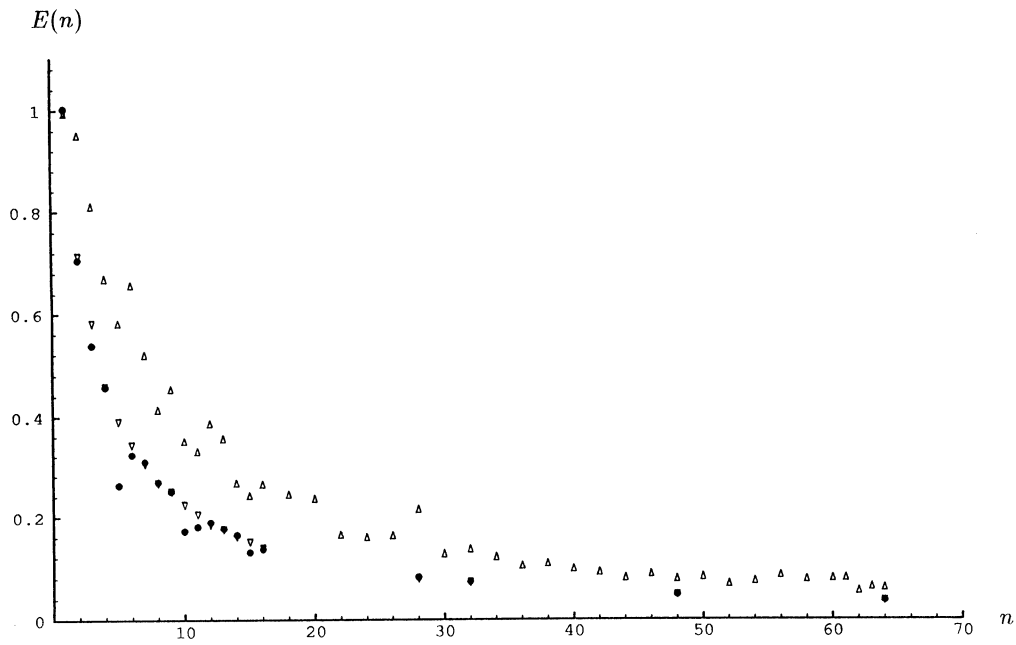

Fig. 11. Computational efficiency as a function of the number of processors n for a small calculation. Filled dots are the efficiencies of a program in which all communication between processors is forbidden. Downward-pointing triangles are for a program in which only entire integrals may be passed between processors. Upward-pointing triangles are for a program which implements all steps of the proposed algorithm, using a routine similar to `pr_integrate` to integrate.

It is apparent that, while the use of the procedure `pr_integrate` (i.e. an implementation of step 4 described above) can improve the efficiency of the computation by almost a factor of two, the actual computational efficiency reached is still very small, and drops rapidly as more processors are used together. The reason for this poor performance is that most of the integration subdomains passed on to idle processors are so small that these processors finish and return their result too quickly, causing a high volume of communication and slowing the computation down.

The cause of the problem is actually the naive recursive implementation of `pr_integrate`. A processor spends most of its time at the lower levels (where the subintervals are smaller) of the recursion tree, where there are more branches, and so, when asked to provide an interval, usually passes one from these low levels on. One would like the routine to instead pass on an untreated interval

from a higher level, but it has no access to these higher levels; it only knows about the subinterval that was passed to it. The solution to this problem is to keep a list of intervals which must still be integrated over, such that the largest one is easily found. This largest untreated interval can be passed on to an otherwise idle processor. Having constructed such a list, however, it is only natural to also use it to control the integration, making the mechanisms provided by recursion redundant. The list can be implemented as a stack, upon which subintervals are placed as subdivision occurs. One can mimic the operation of `r_integrate` by always taking intervals from the top of the stack until the stack is empty. The smallest untreated intervals are always at the top of the stack, and the largest at the bottom (they were put there at an earlier stage of the calculation, when the interval had not yet been subdivided so much). Individual processors can imitate `r_integrate` internally, but are now able to pass on the largest untreated interval to another processor on demand to achieve load-balancing. We have implemented a stack-based version of `pr_integrate`. See Fig. 12 for a sketch of the new algorithm (in the actual program, a processor does not simply wait for results from others when it has nothing more to do, rather it itself asks for more work). Note that it is not recursive.

```
procedure ps_integrate(a,b):
    I ← 0;
    put the interval [a,b] on the stack;
    while the stack is not empty:
        take [a',b'] from the top of the stack;
        approximate the integral by t ≈ ∫_{a'}^{b'} f(x)dx;
        if the estimated error is tolerable then I ← I + t;
        else put [a',(a'+b')/2] and [(a'+b')/2,b'] on the top of the stack;
        if the stack is not empty and another processor needs work then
            give it an interval from the bottom of the stack;
    wait for those processors which were given work, and add their results to I;
    return I.
```

Fig. 12. A stack-based algorithm for evaluating the one-dimensional integral $\int_a^b f(x)\,\mathrm{d}x$ allowing efficient subdivision of the domain of integration amongst many processors.

Fig. 13 is a comparison of the efficiencies of all the implementations discussed so far, applied to the same almost trivial problem as in Fig. 11. Curves have been added as a guide to the eye, where the time required by n processors was parametrised by

$$T(n) \approx T_s + \frac{1}{n}T_p + nT_c\,, \tag{39}$$

where T_s represents the non-parallelisable part of the calculation, T_p the parallelisable part, and T_c the part requiring communication. The stack-based routine `ps_integrate` does indeed bring a significant gain in performance. The actual efficiency is still low. As is typical for parallel applications, an increase in the problem size improves the situation. This is possible when T_p increases more rapidly with problem size than T_s. Efficiencies for the various implementations

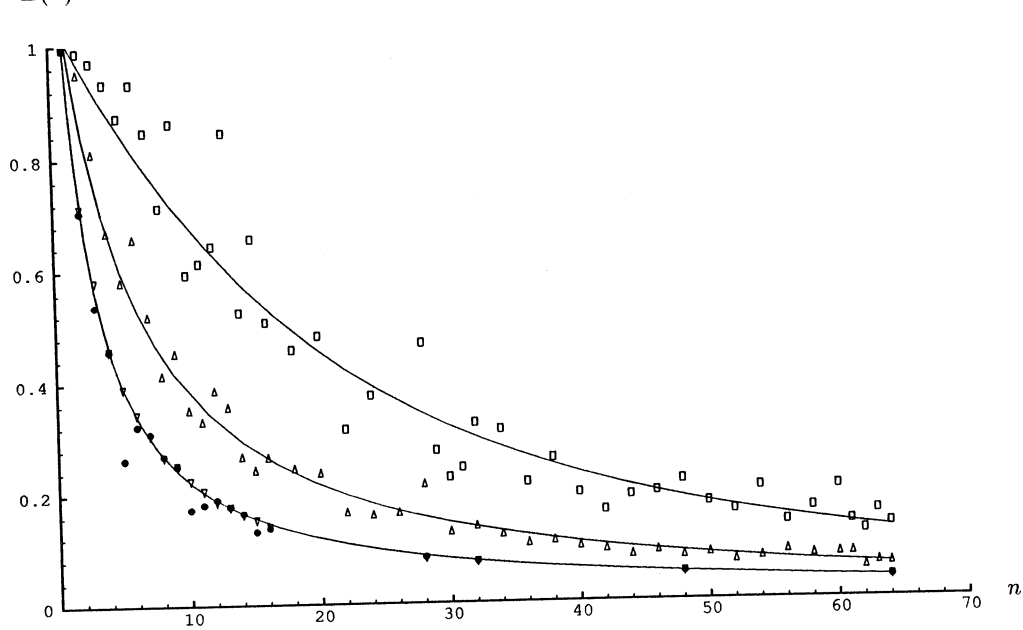

Fig. 13. Computational efficiency for a small problem. Filled dots are for a program without any interprocessor communication. Downward-pointing triangles are for a program in which only entire integrals may be passed between processors. Upward-pointing triangles are for a program which implements all steps of the proposed algorithm, using a routine similar to `pr_integrate` to integrate. Squares are for a program which instead uses `ps_integrate`.

applied to a realistic problem ($j = -12$, `ndiv` $= 1$ and $\ell = 0$) are presented in Fig. 14. One can define a crossover point in an algorithm's performance as the largest number of processors to which one more processor can be added without slowing the calculation down. We can approximate this by taking the parametrisation above, and taking the largest n for which $dT(n)/dn < 0$. In the case of `ps_integrate`, this crossover point is at 30 processors for the almost trivial problem, and at 65 processors for the realistic problem. One sees that `ps_integrate` performs significantly better than `pr_integrate`, and that its efficiency is quite high even for large numbers of processors.

6. Consistency Checks

Having written a final version of the program for the Intel Paragon, we consider it to be necessary to check that the assumptions made do actually hold. The following comments are per definition all positive—the program would otherwise have been corrected.

We concluded that the eigenvalue $\lambda_j^{2,0,\star}$ would scale exponentially with j for $j \ll -10$ on the basis of preliminary calculations without error estimates. Fig. 15 compares the earlier data with results from the parallel version of the program (where `ndiv` $= 2$).

We have to this point made no estimate of the errors due to discretisation (only those incurred during integration). Fig. 16 shows the dependence of $\lambda_{-6}^{2,0,\star}$ on `ndiv`. The error from the discretisation is indeed neglegible compared to that from integration. We also conclude that `ndiv` $= 2$ is a good working value.

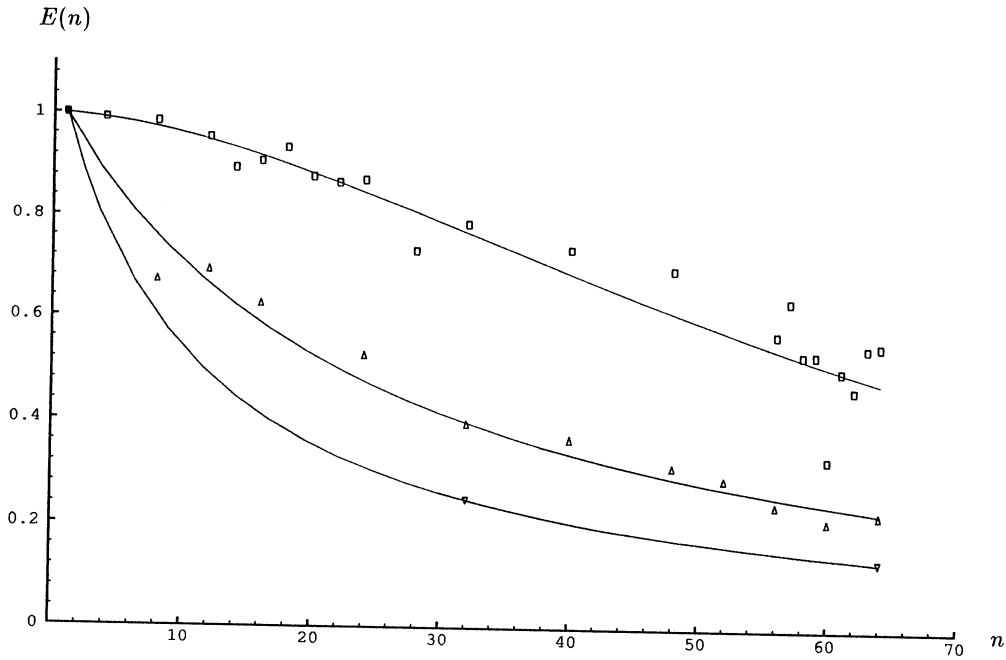

Fig. 14. Computational efficiency for a realistic problem. Downward-pointing triangles are for a program in which only entire integrals may be passed between processors. Upward-pointing triangles are for a program which implements all steps of the proposed algorithm, using a routine similar to `pr_integrate` to integrate. Squares are for a program which instead uses `ps_integrate`.

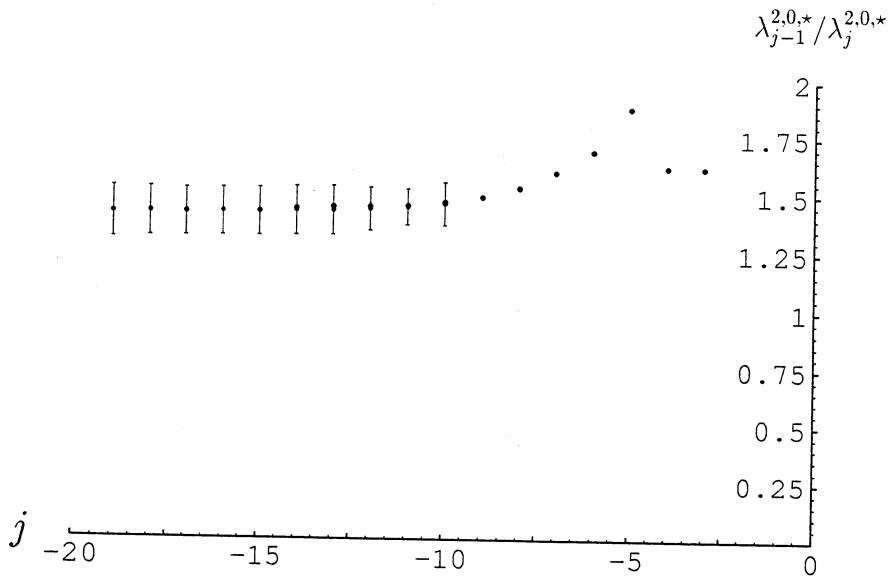

Fig. 15. Ratios of eigenvalues plotted against j. The data points without error bars are from the preliminary calculations. Those with error bars were produced by the final version of the program.

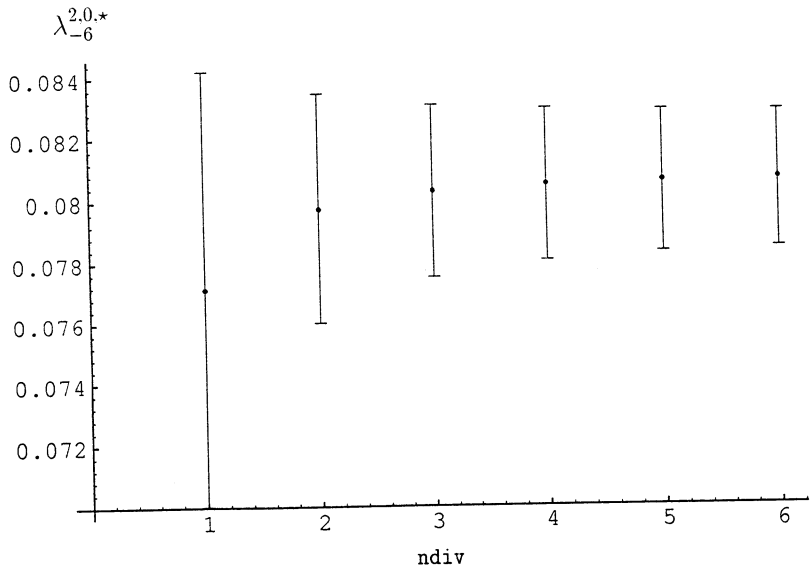

Fig. 16. Dependence of the eigenvalue $\lambda_{-6}^{2,0,\star}$ on **ndiv**. The error bars represent only the errors incurred during integration.

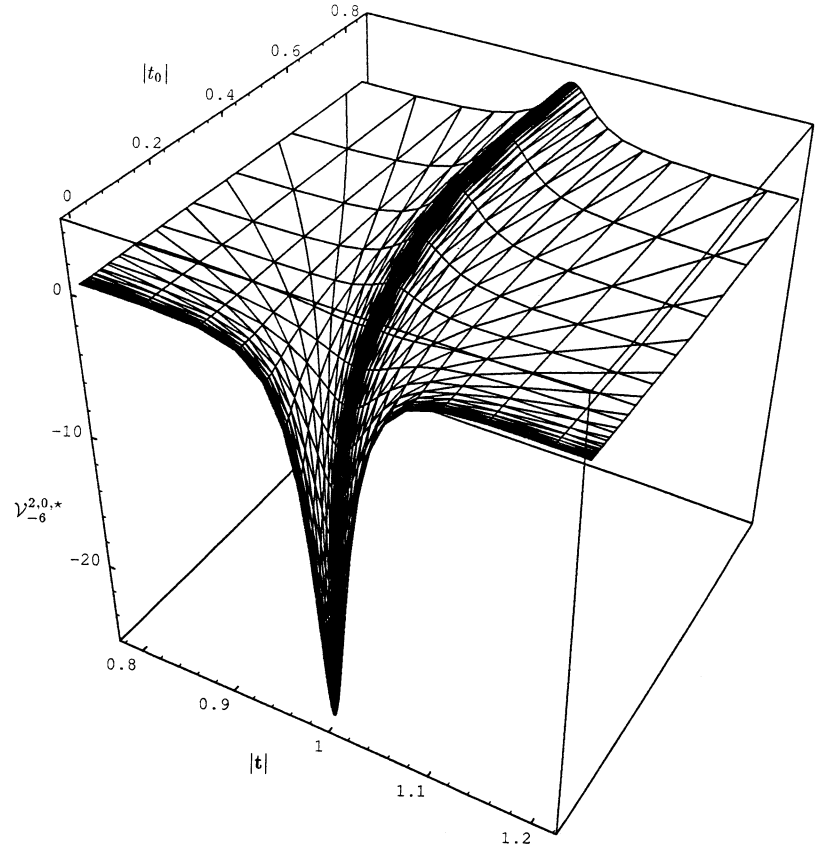

Fig. 17. The eigenvector $\mathcal{V}_{-6}^{2,0,\star}(|t_0|, |\mathbf{t}|)$ calculated with **ndiv** = 2.

Fig. 17 depicts the eigenvector $\mathcal{V}_{-6}^{2,0,\star}(|t_0|,|\mathbf{t}|)$ calculated with `ndiv` $= 2$. The cells $\square_{m,n}$ do appear to be concentrated around regions of detail.

The largest calculation ($j = -20$, `ndiv` $= 2$, i.e. 9×10^6 matrix elements) took three and a half hours of CPU time, using 66 nodes on the Paragon. Attempts to perform the same computation with the recursive integration algorithms discussed here were all aborted after four hours.

7. Conclusion

The most efficient computational approach to a given physical problem often requires much exploratory programming applied to simpler problems. We have made use of symmetries of our problem to reduce the amount of computation required, and then tailored algorithms to the computer architecture chosen, with an emphasis on efficiency. We have found, to our surprise, that the natural, recursive way to code our adaptive integration routine is not suited to the load-balancing demands of a parallel calculation. A stack-based algorithm was found to perform best. We note here that it would be equally appropriate for the parallel implementation of other adaptive integration schemes (such as those described in Lepage 1978), or in fact for any calculation involving many matrix elements expressed as integrals.

The program is available via E-mail to `sinclair@math.ethz.ch`. It is written in FORTRAN77, making use of the Intel `nx` message-passing library.

Acknowledgments

The author wishes to thank E. Trubowitz and J. Feldman for much good advice and many helpful discussions.

References

Akl, S. G. (1989). 'The Design and Analysis of Parallel Algorithms' (Prentice-Hall: New Jersey).

Bares, P.-A., and Wen, X.-G. (1993). *Phys. Rev.* B **48**, 8636.

Davis, P. J., and Rabinowitz, P. (1975). 'Methods of Numerical Integration' (Academic Press: London).

Fetter, A. L., and Walecka, J. D. (1971). 'Quantum Theory of Many-Particle Systems' (McGraw–Hill: New York).

Gates, K. E., and Peterson, W. P. (1994). *Int. J. High Speed Computing* (in press).

Halperin, B. I., Lee, P. A., and Read, N. (1993). *Phys. Rev.* B **47**, 7312.

Intel (1993). 'Paragon User's Guide', Order No. 312489–002 (Intel Corporation: Santa Clara).

Khveshchenko, D. V., Hlubina, R., and Rice, T. M. (1993). *Phys. Rev.* B **48**, 10766.

Lepage, G. P. (1978). *J. Comput. Phys.* **27**, 192.

Stroud, A. H. (1971). 'Approximate Calculation of Multiple Integrals' (Prentice–Hall: New Jersey).

Wilkinson, J. H. (1978). 'The Algebraic Eigenvalue Problem' (Clarendon: Oxford).

Manuscript received 21 June, accepted 4 October 1994