54

A Parallel Domain Decomposition Method for Spline Approximation

C. K. Pink, I. J. Anderson, and J. C. Mason

1 Introduction

The work described in this paper arises from the desire to approximate in a practical timescale a large set of discrete data with a spline function defined by B-spline basis functions. The least squares approximation of large sets of data is an important but time consuming problem, which has applications in many fields including those of graphics, image processing and computer aided design. We consider this approximation problem for what we term "uniform" sets of data, and we describe a parallel domain decomposition method for its solution. We use *uniform* in its statistical sense, namely we mean that the data is scattered in such a way that the density of points is fairly constant throughout the domain of approximation, as in a uniform distribution.

In section 2 we consider the general problem of B-spline approximation, and in section 3 we describe briefly a method produced by Anderson [And94, And97] for its efficient solution when considering uniform data sets. In section 4 we consider the problems involved with producing a parallel method based on this serial approach and we discuss the solutions of these problems. Results gained from implementing the parallel algorithm on the KSR-1 [Ken93, Pap93] machine are given and analysed in section 5.

2 The Two-dimensional Approximation Problem

Given a set of m data, (x_k, y_k, f_k), we wish to calculate the bivariate spline function that minimises the sum of the squares of the residuals between the data ordinates and the spline function evaluated at the data abscissae [Cox87]. The standard definition of a bivariate tensor-product spline is,

$$s(x,y) = \sum_{i=1}^{q_x} \sum_{j=1}^{q_y} c_{i,j} N_i(x; \boldsymbol{\lambda}) N_j(y; \boldsymbol{\mu}),$$

Ninth International Conference on Domain Decomposition Methods
Editor Petter E. Bjørstad, Magne S. Espedal and David E. Keyes

where $N_i(x;\boldsymbol{\lambda})$ and $N_j(y;\boldsymbol{\mu})$ are the B-spline basis functions in the x and y dimensions with respect to the knot sets $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ respectively. The residuals of this fit are $e_k = f_k - s(x_k, y_k)$ for $k = 1, 2, \ldots, m$. Using vectors we express the function values and residuals as $\mathbf{f}$ and $\mathbf{e}$ respectively. We wish to calculate the $q_x \times q_y$ coefficients, $\{c_{i,j}\}$, which minimise the residual sum of squares, $E = \mathbf{e}^{\mathrm{T}}\mathbf{e}$.

In order to express the B-splines evaluated at the data points in the form of a matrix $\boldsymbol{A}$, we employ a lexicographical ordering of the basis functions. The (k,ℓ)th element of $\boldsymbol{A}$ is given by $a_{k,\ell} = N_i(x_k;\boldsymbol{\lambda})N_j(y_k;\boldsymbol{\mu})$, where $\ell = (j-1)q_x + i$. Similarly, the coefficients, $\{c_{i,j}\}$, may be expressed in the form of a vector, $\mathbf{c}$, using the same lexicographical ordering. The system of equations resulting from the minimisation problem,

$$\boldsymbol{A}\mathbf{c} = \mathbf{f},$$

is usually overdetermined and thus requires the use of orthogonal transformations, e.g. Givens rotations, Householder transformations or SVD [GV89], for its solution to be determined.

The compact support property of B-spline basis functions produces a banded matrix $\boldsymbol{A}$. This matrix $\boldsymbol{A}$ has at most $n_x \times n_y$ non-zero elements in each row, where n_x and n_y are the order of the approximation in the x and y dimensions, however these elements are not in consecutive columns of the matrix. In fact, the bandwidth of the matrix [Cox82, HH74] is

$$(n_y - 1)q_x + n_x.$$

Consequently the solution of this system can be computed in $\mathcal{O}(mq_x^2n_y^2)$ [Cox82] floating point operations (flops). Therefore the solution time of the system of equations is dependent on both the order of the spline fit (in the y dimension) and the number of coefficients (in the x dimension).

The dependence of the flop count on the square of the number of coefficients is the important factor with regards to the solution time. To approximate large sets of data well, a comparatively large set of knots is needed, and as the size of the data set increases so we need to increase the size of the knot set. Consequently the linear system that needs to be solved to find the coefficients of the approximation becomes too large to be solved in a practical time.

3 Fast Serial B-spline Approximation

Anderson [And94, And97] has developed a serial method that efficiently approximates large uniform data sets. The basic premise behind the method is to convert an approximation problem on a large domain with a large data set into a set of smaller problems that may be solved more rapidly.

As an example, consider the general least squares approximation problem, described in section 2, on a rectangular domain. The matrix A produced by the system will have m rows, and bandwidth $(n_y - 1)q_x + n_x$, and takes $\mathcal{O}(mq_x^2n_y^2)$ flops to solve. Now consider this problem as a grid of 10×10 subproblems. Each subproblem produces a matrix A with approximately $m/100$ rows (because of the uniform distribution of the data) and a bandwidth of about $(n_y - 1)q_x/10 + n_x$. This system can be solved in approximately $1/10,000$ of the time taken for the original system, and therefore solving the 100 subproblems will take about 1/100 of the time taken for the original system. Clearly, there are time savings that can be made by exploiting this fact.

Simply subdividing the coefficient array and finding these coefficients from subdomains does not, however, produce the least-squares fit for the overall domain. The supports of adjacent B-spline basis functions of order n overlap by $n-1$ knot intervals. Therefore the regions of support for neighbouring subsets of coefficients also overlap by this amount. This means that many data points will be in more than one region and their values will contribute twice to the overall approximation (a problem that is called 'overfitting' [And94]). To overcome this, after each subdomain approximation has been formed, the coefficients found are added to the overall approximation,

$$c_{i,j}^{(r)} = c_{i,j}^{(r-1)} + \hat{c}_{i,j},$$

where the values $\{\hat{c}_{i,j}\}$ are the coefficients found from the subproblem and $\{c_{i,j}^{(r)}\}$ are the coefficients of the overall approximation after r subproblems. This approximation is evaluated at each data point, and a set of *modified function values* (or residuals) are formed as

$$\mathbf{f}^{(r)} = \mathbf{e}^{(r)} = \mathbf{f}^{(r-1)} - \mathbf{s}^{(r)} = \mathbf{f}^{(r-1)} - A\mathbf{c}^{(r)}$$

where $\mathbf{s}^{(r)}$ is the vector of spline values at the data abscissae. Taking these residuals to be the data ordinates for the subsequent approximation problem means that the overall approximation will approximate the correct data values.

The serial algorithm is as follows.

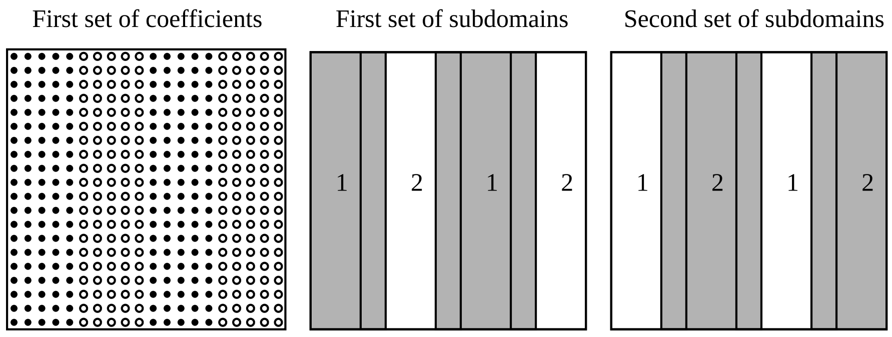
• Choose a small rectangular subregion that supports approximately $3n_x \times 3n_y$ coefficients and calculate the fit on that region.
• Place the coefficients in the correct position in the overall two-dimensional coefficient array.
• Calculate the modified function values at the data points.
• Update the data values by replacing them with the modified function values.
• Iterate this procedure until the residuals meet some convergence criterion.

The subregions are chosen by analysing the residuals in the domain and finding the region where they are largest. Anderson [And94] has proved that this method converges globally to the true least squares approximation to the data, and, for large approximation problems, he suggests that the algorithm is of $\mathcal{O}(mn_y^2)$ [And97]. This makes it faster by $\mathcal{O}(q_x^2)$ when compared to the standard global solution methods.

4 Parallel Algorithm in Two Dimensions

The serial algorithm described above has the potential to work in a parallel environment. Instead of approximating on just one subregion, we can fit on two or more of these regions simultaneously, the only restriction on these regions being that they do not overlap. If this restriction is violated, we would be finding two approximations to some of the data values. This introduces overfitting into the approximation, the problem that the serial algorithm had without modified function values. Choosing enough distinct subregions to employ all of the processing resources is relatively easy for small numbers of processors. However as the number of processors increases with respect to the total number of coefficients to be found, it becomes increasingly

Figure 1 Selecting subdomains by splitting in only one dimension (strips).



difficult to define dynamically sufficient distinct subregions. Also, in a distributed memory environment, choosing subsets of coefficients dynamically will necessitate a lot of unwanted data transfer between processors. To overcome all of these problems we require some systematic way of splitting the domain so that the subdomains are predefined and fixed rather than chosen dynamically.

Recall that the floating point operation count in the solution of the two-dimensional problem is $\mathcal{O}(mq_x^2n_y^2)$. The small subdomains used by the serial algorithm, have smaller values of q_x and, as a result, require far fewer operations to solve. However the number of coefficients in only one of the two dimensions affects the speed of the solution of the system. Therefore subdividing the coefficient set, and the domain, in the second dimension will not affect the global time taken to find the overall approximation.

For the parallel algorithm we consider subdividing the coefficient set in just one dimension, which splits the domain into long thin overlapping strips (see Figure 1). With this decomposition of the domain, we see from the second diagram in Figure 1, that each subdomain overlaps with only two neighbouring regions. Therefore, by grouping the subdomains into two sets, containing alternate subdomains, half of the domain can be fitted in parallel. The modified function values have to be calculated after each parallel section to ensure that we avoid overfitting any of the points. After both of the sets of subdomains have been fitted, we have completed one iteration of the parallel algorithm and can begin fitting on the first group of subregions again. Iterating in this way means that the mathematics of the parallel approximation method are the same as the serial method, and therefore the proof of convergence for the serial method [And97] also holds for the parallel method.

Improving Convergence Rate

A true spline fit of order n needs n knots exterior to the data at each end of the data set. For the decomposition methods described here the subdomains that are formed do not have this property. Therefore towards the edges of subdomains, where fewer than n basis functions cover the data points, the approximation is poor. In the serial algorithm the regions of poor fit are not fixed, because of the dynamic way in which the subdomains are chosen, and therefore are not a problem with regards to the convergence of the approximation. However, the parallel algorithm keeps the domains fixed, and so the approximation remains poor in the overlap regions. Hence the approximation takes a comparatively long time to converge. Changing the subdomains at each iteration of the parallel method, in a similar way to the serial algorithm, causes a large amount of data movement and is thus undesirable. An alternative is to ensure that a true spline fit is formed at every point in the domain in each iteration, thus preventing these regions of poor approximation from arising. We achieve this by increasing the amount of overlap in the subregions from $n-1$ knots to $2(n-1)$. This means that $n-1$ coefficients at the extremes of each subdomain are calculated in the neighbouring subdomain as well. This does increase the parallel workload in each iteration by a small amount [Pin97a], but it does not increase the parallel overheads, and the method converges much more rapidly.

Smaller Domains

The only problem with subdividing the domain into strips is that, for smaller data fitting problems using many processors, there may not be sufficient subdomains to employ all of the available processing resources.

For a parallel system with p processors, the length of the *full* knot set in the leading dimension is $K = q_x + n_x$. The minimum value of K, that allows the domain to be split only in that dimension and still employ all of the processors, is given by

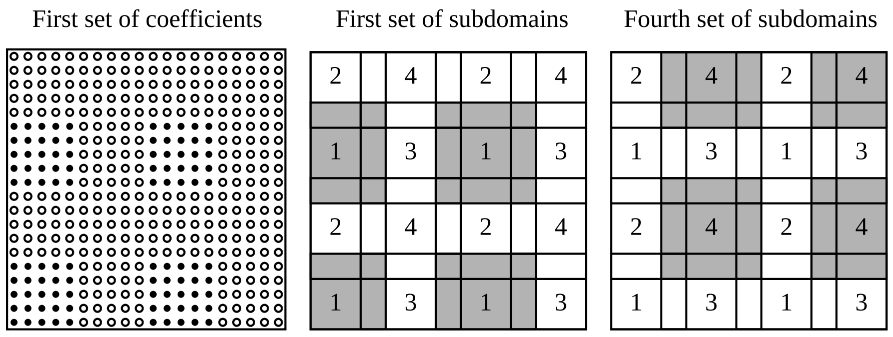
$$K = (4p + 2)(n_x - 1).$$

The reasons for this are discussed in [Pin97b]. The most commonly used two-dimensional B-spline fit is bi-cubic ($n_x = n_y = 4$), and the minimum length of the knot set for this order of approximation is thus $6(2p + 1)$. If the knot set in the second dimension is of comparable size, then the total number of coefficients needs to approach 40000 before we are able to utilise 16 processors with this decomposition of the domain. Two dimensional approximation problems far smaller in size than this are prohibitively large to solve on a serial machine.

To cope with these smaller domains in a parallel context we must consider subdividing the coefficient array into small rectangular subsets (first diagram in Figure 2). We then take the subdomains to be the regions of support of the sets of B-spline basis functions corresponding to the sets of coefficients (second diagram in Figure 2). Because each region now has as many as eight other subdomains overlapping it, the best that we can do with this decomposition of the domain is to approximate on the domain in four parallel sections. Therefore, more work is introduced into the program, because the modified function values need to be calculated more often, and also there are more synchronisation points in the program because of the increase in distinct parallel sections. However, to effectively utilise 16 processors with this decomposition of the domain we need at least 64 subregions, and on an approximately square domain this can be done by decomposing the domain into a grid of 8×8 subdomains. For a bi-cubic B-spline approximation we need only a minimum of 54×54 (about 3000) coefficients, much less than was the case with the first decomposition method.

On larger domains, however, subdividing the domain in only one dimension and therefore using strips as subdomains is the better choice. This decomposition should allow better load balancing, and less time will be spent calculating the modified

Figure 2 Selecting subdomains by splitting in both dimensions (boxes).



function values.

Decomposition Algorithm

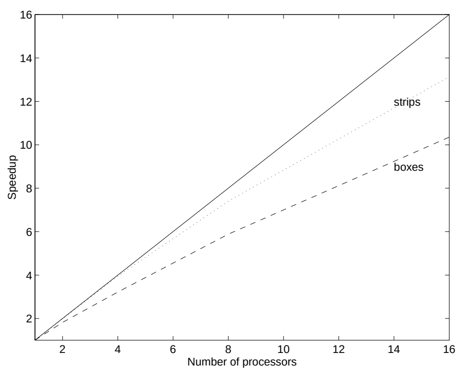
When decomposing the domain in only one dimension, we must ensure that we produce a multiple of $2p$ strips to allow an equal distribution of subdomains across the processors. The width of the subdomain, q_x, must not be too large because of the increase in time taken to solve large domains [And94, Pin97b]. If the problem is not large enough to split in only one dimension, we need to subdivide both dimensions of the domain, and this complicates the decomposition. The decomposition algorithm is summarized below, but a more in depth discussion of it is given in [Pin97b].

1. If $q_x \geq (4p+2)(n_x-1)$, then we can split the domain in one dimension.
   (a) Find s such that
   $$(4sp+2)(n_x-1) \leq q_x < (4p(s+1)+2)(n_x-1) \qquad s \in \mathbb{N},$$
   and split the domain into $2sp$, approximately equal in size, subdomains.
2. Otherwise:
   (a) Find the smallest non-identity factor of p, r say.
   (b) If $q_x \geq (4(p/r)+2)(n_x-1)$ then we may proceed. If this is not the case we use the next smallest factor of p and try the same test until it is satisfied for some r a factor of p.
   (c) Check to ensure that $q_y \geq (4r+2)(n_y-1)$. If this is not the case then we cannot split the domain into enough subregions to employ all of the processors.
   (d) Split the x dimension into $2(p/r)$ equal subdomains and the y dimension into $2r$ equal subdomains to produce a set of $4p$ subdomains.

Implementation on Parallel Architectures

The two-dimensional algorithm is readily implemented on a shared memory architecture such as the KSR-1. Its implementation on a distributed memory - message passing environment is more complicated and the details of this are given in [Pin97b].

Figure 3 Speedup gained on the KSR-1.



5 Results

We ran the program with two different sets of data and knots. The data sets used contained 50,000 and 200,000 points, and these sets were approximated using 54×54 and 204×54 coefficients respectively. Timings for the computational part of the program were made with the data sets on up to 16 processors and these timings can be found in the technical report [Pin97b]. From these timings, the speedups of the parallel system were calculated and these are shown in Figure 3.

The domain with 54×54 coefficients, labelled “boxes” in Figure 3, was decomposed in both dimensions, but the larger problem was decomposed using strips as subdomains and is labelled “strips” in Figure 3. Both methods give very good speedups on the KSR-1. On 16 processors the small problem has a speedup of 10.4 and the larger problem a speedup of 13.1. This difference is due to the better load balancing which is achieved when using strips as subdomains. With optimum load balancing there should in fact be little difference in the speedups obtained from the two decompositions when implemented on the KSR-1.

6 Conclusions

A parallel algorithm that utilises domain decomposition has been described. The results from a practical implementation of the method on the KSR-1 parallel machine show that good speedups can be achieved on two-dimensional problems with data sets which have a uniform distribution of points. The method is sufficiently versatile to be applicable to problems of higher dimension.

Acknowledgement

We should like to thank NAg Ltd. for their support of this work and we should also like to thank Manchester Computing Centre for granting the use of the KSR-1 machine and the London School of Dentistry for supplying the data used in this report.

REFERENCES

[And94] Anderson I. J. (1994) *Efficient multivariate approximation for large sets of structured data.* PhD thesis, University of Huddersfield, Huddersfield, U.K.

[And97] Anderson I. J. (1997) A piecewise approach to piecewise approximation. Preprint.

[Cox82] Cox M. G. (1982) Direct versus iterative methods of solution for multivariate spline-fitting problems. *IMA J. Numer. Anal.* 2: 73–81.

[Cox87] Cox M. G. (1987) Data approximation by splines in one and two variables. In *The State of the Art in Numerical Analysis*, pages 111–138.

[GV89] Golub G. H. and Van Loan C. F. (1989) *Matrix Computations.* John Hopkins University Press, Baltimore, Maryland.

[HH74] Halliday J. and Hayes J. G. (1974) Least squares fitting of cubic spline surfaces to general data. *J. Inst. Math. and Applics.* 14: 89–103.

[Ken93] Kendall Square Research Corporation, Waltham, Ma, USA (October 1993) *KSR Parallel Programming*, second edition.

[Pap93] Papadimitriou P. (December 1993) The KSR-1—A numerical analyst's perspective. Numerical Analysis Report 242, University of Manchester/UMIST.

[Pin97a] Pink C. K. (February 1997) Evaluating B-spline approximations in parallel in one and two dimensions. Technical Report 9705, University of Huddersfield.

[Pin97b] Pink C. K. (January 1997) Parallel B-spline approximation of large sets of uniform data. Technical Report 9706, University of Huddersfield.