- The heading is numbered to three digits separated with a full point, with a 1 pica space separating it from the text.
- The text is keyed in upper and lower case with an initial capital for first word only, and is unjustified.

4. Acknowledgements
   This heading is the same style as an A level heading but is not numbered.

6 TEXT

The first paragraph of text following any heading is set to the complete measure (i.e. do not indent the first line).

Subsequent paragraphs are set with the first line indented by 1 pica (3.85 mm).

There isn't any inter-paragraph spacing.

7 LISTS

The list identifier may be an arabic number, a bullet, an em rule or a roman numeral.

The items in a list are set in text size and indented by 1 pica (4.2 mm) from the left margin. Half a line of space is set above and below the list to separate it from surrounding text.

See layout of Section 5 on headings to see the results of the list macros.

8 TABLES

Tables are set in 8 point (2.8 mm) on a body of 10 point (3.5 mm). The table caption is set centered at the start of the table, with the word Table and the number in bold. The caption is set in medium with a 1 pica (4.2 mm) space separating it from the table number.

A one line space separates the table from surrounding text.

Table 1. The table caption is centered on the table measure. If it extends to two lines each is centered.

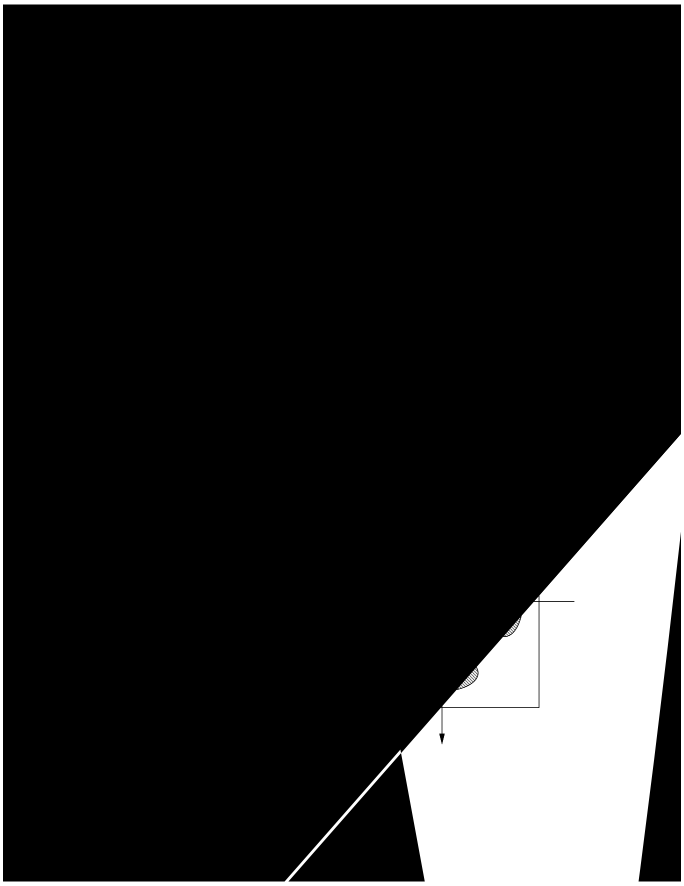
	Processors						
	1	2			4		
Window	◇	◇	□	△	◇	□	△
1	1273	110	21.79	89%	6717	22.42	61%
2	2145	116	10.99	50%	5386	10.77	19%
3	3014	117	41.77	89%	7783	42.31	58%
4	4753	151	71.55	77%	7477	61.97	49%
5	5576	148	61.60	80%	7551	91.80	45%

◇ execution time in ticks □ speed-up values △ efficiency values

9 FIGURES

A figure caption is set centered in 8 point (2.8 mm) medium on a leading of 10 point (3.5 mm). It is set under the figure, with the word Figure and the number in bold and with a 1 pica (4.2 mm) space separating the caption text from the figure number.

One line of space separates the figure from the caption. A one line space separates the figure from surrounding text.



12 THEOREMS

The text of a theorem is set in 9 point (3.15 mm) italic on a leading of 11 point (3.85 mm). The word Theorem and its number are set in 9 point (3.15 mm) bold.

A one line space separates the theorem from surrounding text.

Theorem 1 *Let us assume this is a valid theorem. In reality it is a piece of text set in the theorem environment.*

13 FOOTNOTES

Footnotes are set in 8 point (2.8 mm) medium with leading of 8.6 point (3.1 mm), with a 1 point (0.35 mm) footnote rule to column width[2] .

14 REFERENCES

The reference identifier in the text is set as a sequential number in square brackets. The reference entry itself is set in 8 point (2.8 mm) with a leading of 10 point (3.5 mm), and appears in the sequence in which it is cited in the paper.

15 SAMPLE CODING

The remainder of this paper contains examples of the specifications detailed above and can be used for reference if required.

18.3 Matrix–vector product

For the matrix–vector product $v = Au$, we use a *column partitioning* of A. Each processor holds a set W_p (see Section 17) of s columns each of N elements of A and s elements of u. The s elements of u stored locally have a one-to-one correspondence to the s columns of A (e.g. a processor holding element u_j also holds the j-th column of A). Note that whereas we have A partitioned by columns among the processors, the matrix–vector product is to be computed by *rows*.

The algorithm for computing the matrix–vector product using column partitioning is a generalization of the inner-product algorithm described in Section 18.2 (without the need for a final broadcast phase). At a given time during the execution of the algorithm, each one of $P - 1$ processors is computing a vector w of s elements containing partial sums required for the segment of the vector v in the remaining 'target' processor. After this computation is complete, each of the P processors stores a vector w. The resulting segment of the matrix–vector product vector which is to be stored in the target processor is obtained by summing together the P vectors w, as described below.

Each processor other than the target processor sends its w vector to one of its neighboring processors. A processor decides whether to send the vector in either the row or column direction to reach the target processor based on the FRFC algorithm (see Section 16.1.1). If a vector passes through further processors in its route to the target processor the w vectors are accumulated. Thus the target processor will receive at most four w vectors which, when summed to its own w vector, yield the desired set of s elements of v.

18.4 Matrix–vector product—finite-difference approximation

We now consider a preconditioned version of the conjugate-gradients method [7]. Note that we do not need to form A explicitly. This implies a very low degree of information exchange between the processors which can be effectively exploited with transputers, since the required values of u can be exchanged independently through each link.

The preconditioning used in our implementations is the polynomial preconditioning (see [10], [6], [1] and [8]), which can be implemented very efficiently in a parallel architecture since it is expressed as a sequence of saxpys and matrix–vector products.

We have l rows and columns in the discretization grid, which we want to partition among a $P_{\mathrm{r}} \times P_{\mathrm{c}}$ mesh of processors. Each processor will then carry out the computations associated with a block of $\lfloor l/P_{\mathrm{r}} \rfloor + \mathrm{sign}\,(l \bmod P_{\mathrm{r}})$ rows and $\lfloor l/P_{\mathrm{c}} \rfloor + \mathrm{sign}\,(l \bmod P_{\mathrm{c}})$ columns of the interior points of the grid.

The matrix–vector product using the column partitioning is highly parallel. Since there is no broadcast operation involved, as soon as a processor on the boundary of the grid (either rows or columns) has computed and sent a w_p vector destined to a processor 'A', it can compute and (possibly) send a w_p vector to processor 'B', at which time its neighboring processors may also have started computing and sending their own w vectors to processor 'B'.

At a given point in the matrix–vector product computation, the processors are computing w vectors destined to processor A. When these vectors have been accumulated in the row of that processor (step 1), the processors in the top and bottom rows compute and send the w vectors for processor B, while the processors on the left and right columns of the row of processor A send the accumulated r vectors to processor A (step 2). Processor A now stores its set of the resulting v vector (which is the accumulation of the w vectors). In step 3, the processors in the bottom row compute and send the w vectors for processor C while the processor at the left-hand end of the row of processor B sends the accumulated w vectors of that column towards processor B. The next steps are similar to the above.

In our implementation, we exploit the geometry associated with the regular grid of points used to approximate the PDE. A geometric partitioning is used to match the topology and connectivity present in the grid of transputers (Section 16.1).

The discretization of the PDE is obtained by specifying a grid size l defining an associated grid of $N = l^2$ interior points (note that this is the order of the linear system to be solved). With each interior point, we associate a set of values, namely the coefficients C, N, S, E and W.

19 CONCLUSION

We have shown that an iterative method such as the preconditioned conjugate-gradients method may be successfully parallelized by using highly efficient parallel implementations of the linear algebra operations involved. We have used the same approach to parallelize other iterative methods with similar degrees of efficiency (see [4] and [3]).

ACKNOWLEDGEMENTS

We would like to thank the referees for their comments which helped improve this paper.

REFERENCES

[1] L. Adams, 'm-Step preconditioned Gradient methods', *SIAM Journal of Scientific and Statistical Computing*, **6**, 452–463, (1985).

[2] P. Atkin. Performance maximisation. INMOS Technical Note 17.

[3] R.D. da Cunha and T.R. Hopkins, *The Parallel Solution of Partial Differential Equations on Transputer Networks*, 96–109, Transputing for Numerical and Neural Network Applications, IOS Press, Amsterdam, 1992. Also as Report No. 17/92, Computing Laboratory, University of Kent at Canterbury, U.K.

[4] R.D. da Cunha and T.R. Hopkins, *The Parallel Solution of Systems of Linear Equations using Iterative Methods on Transputer Networks*, 1–13, Transputing for Numerical and Neural Network Applications, IOS Press, Amsterdam, 1992. Also as Report No. 16/92, Computing Laboratory, University of Kent at Canterbury, U.K.

[5] U. de Carlini and U. Villano, *Transputers and parallel architectures – message-passing distributed systems*, Ellis Horwood, Chichester, 1991.

[6] S.C. Eisenstat, 'Efficient implementation of a class of preconditioned Conjugate Gradient methods', *SIAM Journal of Scientific and Statistical Computing*, **2**, 1–4, (1981).

[7] G.H. Golub and C.F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, 2nd edn., 1989.

[8] O.G. Johnson, C.A. Micchelli, and G. Paul, 'Polynomial preconditioners for Conjugate Gradient calculations', *SIAM Journal of Numerical Analysis*, **20**, 362–376, (1983).

[9] J.J. Modi, *Parallel Algorithms and Matrix Computation*, Oxford University Press, Oxford, 1988.

[10] Y. Saad, 'Practical use of polynomial preconditionings for the Conjugate Gradient method', *SIAM Journal of Scientific and Statistical Computing*, **6**, 865–881, (1985).

[11] C.F. Schofield, *Optimising FORTRAN programs*, Ellis Horwood Publishing, Chichester, 1989.

[12] G.D. Smith, *Numerical Solution of Partial Differential Equations: Finite Difference Methods*, Oxford University Press, Oxford, 3rd edn., 1985.