Guidelines for Preparing a Paper for European Conference on Artificial Intelligence

Roger Evans[1] and **Werner Horn**[2]

Abstract. The purpose of this paper is to show a contributor the required style for a paper for ECAI-2000 and PAIS-2000. The specifications for layout are described so that non-LaTeX users can create their own style sheet to achieve the same layout. The source for the sample file is available for LaTeX users. The PostScript and the PDF file is available for all.

The layout is identical to ECAI-98 papers except for the footers. The publisher (IOS Press) will insert a footer for each page.

1 GENERAL SPECIFICATIONS

The following details should allow contributors to set up the general page description for their paper:

1. The paper is set in two columns each 20.5 picas (86 mm) wide with a column separator of 1.5 picas (6 mm).
2. The typeface is Times Modern Roman.
3. The body text size is 9 point (3.15 mm) on a body of 11 point (3.85 mm) (i.e. 61 lines of text).
4. The effective text height for each page is 56 picas (237 mm). The first page has less text height. It requires an additional footer space of 3.5 picas (14.8 mm) for the copyright inserted by the publisher and 1.5 picas (6 mm) of space before the title. The effective text height of the first page is 51 picas (216 mm).
5. There are no running feet for the final camera-ready version of the paper. The submission paper should have page numbers in the running feet.

2 TITLE, AUTHOR, AFFILIATION, COPYRIGHT AND RUNNING FEET

2.1 Title

The title is set in 20 point (7 mm) bold with leading of 22 point (7.7 mm), centred over the full text measure, with 1.5 picas (6 mm) of space before and after.

2.2 Author

The author's name is set in 11 point (3.85 mm) bold with leading of 12 point (4.2 mm), centred over full text measure, with 1.5 picas (6 mm) of space below. A footnote indicator is set in 11 point (3.85 mm) medium and positioned as a superscript character.

[1] Information Technology Research Institute, University of Brighton, Brighton, BN2 4GJ, UK, email: ecai98@cogs.susx.ac.uk

[2] Austrian Research Institute for Artificial Intelligence, Schottengasse 3, A-1010 Vienna, Austria, email: ecai2000@ai.univie.ac.at

2.3 Affiliation

The affiliation is set as a footnote to the first column. This is 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.

2.4 Copyright

The copyright details will be inserted by the publisher.

2.5 Running feet

The running feet is inserted by the publisher. For submission please insert page numbers in the middle of the running feet. Do not insert page numbers for the camera-ready version of the paper.

3 ABSTRACT

The abstract for the paper is set in 9 point (3.15 mm) medium, on a body of 10 point (3.5 mm). The word Abstract is set in bold, followed by a full point and a 0.5 pica space.

4 HEADINGS

Three heading levels have been specified:

1. A level headings
   - The first level of heading is set is 11 point (3.85 mm) bold, on a body of 12 point (4.2 mm), 1.5 lines of space above and 0.5 lines of space below.
   - The heading is numbered to one digit with a 1 pica space separating it from the text.
   - The text is keyed in capitals and is unjustified.
2. B level headings
   - The second level of heading is set is 11 point (3.85 mm) bold, on a body of 12 point (4.2 mm), 1.5 lines of space above and 0.5 lines of space below.
   - The heading is numbered to two 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.
3. C level headings
   - The third level of heading is set is 10 point (3.5 mm) italic, on a body of 11 point (3.85 mm), 1.5 lines of space above and 0.5 lines of space below.

- 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.

5 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.

6 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 4 on headings to see the results of the list macros.

7 TABLES

Tables are set in 8 point (2.8 mm) on a body of 10 point (3.5 mm). The table caption is set centred 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 centred on the table measure. If it extends to two lines each is centred.

	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

8 FIGURES

A figure caption is set centred 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.

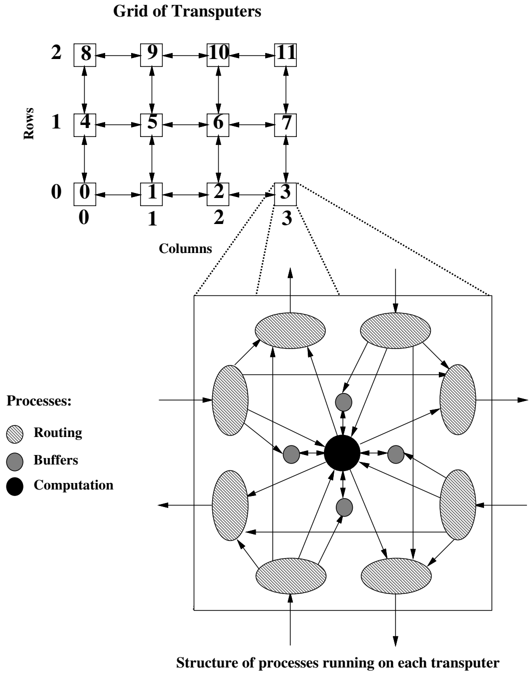

Figure 1. Network of transputers and the structure of individual processes

9 EQUATIONS

A display equation is numbered, using arabic numbers in parentheses. It is centred and set with one line of space above and below to separate it from surrounding text. The following example is a simple single line equation:

$$Ax = b \tag{1}$$

The next example is a multi-line equation:

$$(x+y)(x-y) = x^2 - xy + xy - y^2 \tag{2}$$
$$(x+y)^2 = x^2 + 2xy + y^2 \tag{3}$$

The equal signs are aligned in a multi-line equation.

10 PROGRAM LISTINGS

Program listings are set in 9 point (3.15 mm) Courier on a leading of 11 point (3.85 mm). That is to say, a non-proportional font is used to ensure the correct alignment.

A one line space separates the program listing from surrounding text.

```
void inc(x)
int* x;
{
    *x++;
}

int y = 1;
inc(&y);
printf("%d\n",y);
```

11 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.*

12 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[3] .

13 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.

14 SAMPLE CODING

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

15 PROGRAMMING MODEL

Our algorithms were implemented using the *single program, multiple data* model (SPMD). SPMD involves writing a single code that will run on all the processors co-operating on a task. The data are partitioned among the processors which know what portions of the data they will work on [7].

15.1 Structure of processes and processors

The grid has $P = P_{\rm r} \times P_{\rm c}$ processors, where $P_{\rm r}$ is the number of rows of processors and $P_{\rm c}$ is the number of columns of processors.

15.1.1 Routeing information on the grid

A message may be either *broadcast* or *specific*. A broadcast message originates on a processor and is relayed through the network until it reaches all other processors. A specific message is one that is directed to a particular target processor.

Broadcast messages originate from a processor called *central* which is situated in the 'middle' of the grid. This processor has co-ordinates $(\lfloor P_{\rm r}/2 \rfloor, \lfloor P_{\rm c}/2 \rfloor)$. Messages are broadcast using the *row–column broadcast* algorithm (RCB), which uses the following strategy. The number of steps required to complete the RCB algorithm (i.e. until all processors have received the broadcast value) is given by $\lfloor P_{\rm r}/2 \rfloor + \lfloor P_{\rm c}/2 \rfloor$.

A specific message is routed through the processors using the *find-row–find-column* algorithm (FRFC) detailed in [5]. The message is sent from the *originator* processor vertically until it reaches a processor sitting in the same row as the *target* processor. The message is then moved horizontally across the processors in that row until it reaches the target processor. An accumulation based on the recursive doubling technique [9, pp. 56–61], would require the same number of steps as the RCB requires. If either the row or column of the originator and target processors are the same then the message will travel only in a horizontal or vertical direction, respectively (see [12]).

[3] This is an example of a footnote that occurs in the text. If the text runs to two lines the second line aligns with the start of text in the first line.

16 DATA PARTITIONING

We use *data partitioning by contiguity*, defined in the following way. To partition the data (i.e. vectors and matrices) among the processors, we divide the set of variables $V = \{\, i \,\}_{i=1}^{N}$ into P subsets $\{\, W_p \,\}_{p=1}^{P}$ of $s = N/P$ elements each. We assume without loss of generality that N is an integer multiple of P. We define each subset as $W_p = \{(p-1)s + j\}_{j=1}^{s}$ (see [11], [4] and [2] for details).

Each processor p is responsible for performing the computations over the variables contained in W_p. In the case of vector operations, each processor will hold segments of s variables. The data partitioning for operations involving matrices is discussed in Section 17.3.

17 LINEAR ALGEBRA OPERATIONS

17.1 Saxpy

The saxpy $w = u + \alpha v$ operation, where u, v and w are vectors and α is a scalar value, has the characteristic that its computation is *disjoint elementwise* with respect to u, v and w. This means that we can compute a saxpy without any communication between processors; the resulting vector w does not need to be distributed among the processors. Parallelism is exploited in the saxpy by the fact that P processors will compute the same operation with a substantially smaller amount of data. The saxpy is computed as

$$w_i = u_i + \alpha v_i, \quad \forall i \in \{W_p\}_{p=1}^{P} \tag{4}$$

17.2 Inner-product and vector 2-norm

The inner-product $\alpha = u^T v = \sum_{i=1}^{N} u_i v_i$ is an operation that involves accumulation of data, implying a high level of communication between all processors. The mesh topology and the processes architecture used allowed a more efficient use of the processors than, for instance, a ring topology, reducing the time that processors are idle waiting for the computed inner-product value to arrive, but the problem still remains. The use of the SPMD paradigm also implies the global broadcast of the final computed value to all processors.

The inner-product is computed in three distinct phases. Phase 1 is the computation of partial sums of the form

$$\alpha_p = \sum_{\forall i \in \{W_p\}} u_i \times v_i, \quad p = 1, \ldots, P \tag{5}$$

The accumulation phase of the inner-product using the RCA algorithm is completed in the same number of steps as the RCB algorithm (Section 15.1.1). This is because of the need to relay partial values between processors without any accumulation taking place, owing to the connectivity of the grid topology.

The vector 2-norm $\alpha = \|\, u \,\|_2 = \sqrt{u^T u}$ is computed using the inner-product algorithm described above. Once the inner-product value is received by a processor during the final broadcast phase, it computes the square root of that value giving the required 2-norm value.

17.3 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 16) 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 17.2 (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 neighbouring 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 15.1.1). 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.

17.4 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 neighbouring 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 15.1).

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.

18 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 Conjugate 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, Chichester, 1989.

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