SPECIAL ISSUE

7th SA COMPUTER RESEARCH SYMPOSIUM
PROCEEDINGS

Guest Editor: Judy M Bishop

Organised by the SA Institute of Computer Scientists in association with the Computer Society of SA

Sponsored by Persetel and the FRD
When the first SA Computer Symposium was held at the CSIR in the early eighties, it was unique. There was no other forum at the time for the presentation of research in computer science. In the intervening decade, conferences, symposia and workshops have sprung up in response to demand, and now there are several successful ventures, some into their third or fourth iteration. Each of these addresses a specific topic - for example, hypermedia, expert systems, parallel processing or formal aspects of computing - and attracts a specialised audience, well versed in the subject and eager to learn more. For the main part, the proceedings are informal, and certainly not archival.

SACRS, though, is still unique, in that it deliberately covers a broad spectrum of research in computing, and in addition, seeks to provide a lasting record of the proceedings. To achieve the second aim, we negotiated with the SA Institute of Computer Scientists for the proceedings to form a special issue of the SA Computer Journal, and the copy you have in front of you is the result. The collaboration between the symposium committee and the journal’s editorial board placed high standards on the refereeing and final presentation of the papers, to the symposium’s benefit, while we were still able to maintain a fresh, audience-oriented approach to the selection of papers.

This is SACJ’s first such special issue, and the largest issue (at 145 pages) to date. We hope that it is only the beginning of future such collaborations.

In all 29 papers were received, all were refereed twice, and 19 were chosen for presentation by the programme committee. All the papers were thoroughly revised by the authors on the basis of the referee’s comments, and the committee’s suggestions aimed at making the material more accessible to a broadly-based audience. Papers had to be new, and not to have been presented elsewhere, a requirement that is still unusual within the SA conference round.

A third goal of SACRS has been to invite keynote speakers, usually from overseas. This year, we are fortunate to present Dr Vinton Cerf, the father of the Internet and a world-renown expert on computer networks. Although his paper is not available for this special issue, it will appear later in SACJ. Through the good offices of Professor Chris Brink of UCT, we also have three other speakers from Germany, Canada and the US adding interest to the event, and two of their papers appear in this issue.

The programme committee originally devised a theme for the symposium - "Computing in the New South Africa". We received several queries as to the meaning of this theme, but unfortunately few papers that addressed it directly. One prospective author went as far as to enquire whether computer research would survive in the new South Africa. Another felt that his work was definitely not in the theme, as it was genuine, old world, basic, theoretical science! Nevertheless, there are two papers that consider one of South Africa’s key issues, that of language. Others look at the success we have achieved in applying technology to mining, and the future of low-cost operating systems. In all, the mix of papers represents a balance between the theoretical and the practical, the past and the future, all firmly based in the computing of the present.

Organising the symposium has involved the hard work of several people, and I would like to thank in particular

- Derrick Kourie, my co-organiser, and the editor of SACJ for his invaluable advice and hard work throughout the planning and implementation stages;
- Riël Smit, the production editor, for attaining such a high standard in such a short time for so many papers;
- Gerrit Prinsloo and the staff at the CSSA for their efficient and quite delightfully unfussy organisation;
- Persetel for their very generous sponsorship of R25000, and Tim Schumann for taking a genuine interest in our events;
- the Foundation for Research Development for sponsoring Vint Cerf’s visit;
- and finally the Department of Computer Science of the University of Pretoria for providing the ideal working conditions for undertaking ventures of this kind, and especially Roelf van den Heever for his unfailing encouragement and support.

Judy M Bishop
Organising Chairman, SACRS 1992
Guest Editor, SACJ Special Issue
Referees

The journal draws on a wide range of referees. The following were involved in the refereeing of the papers selected for this special issue. Their role in certifying the papers and their contribution to enhancing the quality of papers is sincerely appreciated.

John Barrow
Ronnie Becker
Danie Behr
Sonia Berman
Liesbeth Botha
Theo Bothma
Chris Brink
Peter Clayton
Ian Cloete
Antony Cooper
Elise Ehlers
Quintin Gee
Andy Gravell
Wendy Hall
David Harel
Scott Hazelhurst
Derrick Kourie
Willelm Labuschagne
Doug Laing
Dave Levy
Graham McLeod
Hans Messerschmidt
Deon Oosthuizen
Ben Oosthuysen
Neil Pendock
D Petkov
Martin Rennhackkamp
Cees Roon
Jan Roos
John Shochot
Morris Sloman
Riel Smit
Pat Terry
Walter van den Heever
Lynn van der Vegte
Herman Venter
Herna Viktor

UNISA
University of Cape Town
University of Pretoria
University of Cape Town
University of Pretoria
University of Pretoria
University of Cape Town
Rhodes University
Stellenbosch University
CSIR
Rand Afrikaanse University
DALSIG
University of Southampton
University of Southampton
The Weizmann Institute of Science
University of the Witwatersrand
University of Pretoria
UNISA
ISM
University of Natal
University of Cape Town
University of the Orange Free State
University of Pretoria
University of Pretoria
University of the Witwatersrand
University of Natal
Stellenbosch University
University of Pretoria
University of Pretoria
University of the Witwatersrand
Imperial College, London
University of Cape Town
Rhodes University
University of Pretoria
University of Pretoria
University of Fort Hare
Stellenbosch University

ii
Placing Processes in a Transputer-based Linda Programming Environment

P.G. Clayton*, F.K. de-Heer-Menlah, and E.P. Wentworth
Department of Computer Science, Rhodes University, Grahamstown, 6140 RSA
Internet: cspc@alpha.ru.ac.za

Abstract

This paper presents a process placement algorithm, designed for use in a transputer-based implementation of the Linda programming environment. A static, heuristic approach is adopted, which takes cognisance of the special characteristics of Linda programs, in particular those of the master-worker class. The algorithm has been used in conjunction with a distributed transport layer. Initial results have been promising.

Keywords: distributed systems, parallel processing, transputer, Linda, Helios.

Computing Review Categories: C.2.4, D.1.3, D.3.4

Received: February 1992, Accepted: April 1992

1 Introduction

The Linda programming paradigm is a simple and elegant approach to parallel processing, adopted by programmers primarily for its ease of use [1]. A Linda programming environment is characterised by a shared, associative memory, called tuple space (TS) [3], which is visible simultaneously to all components of the application program, even though they might reside on separate physical processors. Parallel components of a program deposit, withdraw, and copy items of data and code, known as tuples, from the shared TS. The simplicity of the Linda approach to developing parallel programs hinges on components of an application program communicating only via TS, thereby being decoupled in time and in space from each other.

A second desirable feature of the Linda approach is its ability to dynamically balance the processing loads of parallel processors, as long as the program is set up to exploit this property. This effect is particularly apparent in parallel programs which fall in the master-worker model. This model is the most flexible embodiment of the style of Linda programming known as Agenda Parallelism [2]. Here, a master process throws work into TS for several worker processes to do. As long as there are work requests in TS for workers to fulfil, they are able to work in their own time and keep themselves busy, which achieves a good overall load balance. As long as TS is prepared to communicate, and since workers never communicate directly with each other, worker cycles are not wasted waiting for synchronization points to be reached. This pleasing effect for programmers of parallel systems assumes that all component processes of an application have easy access to TS. In distributed, parallel processing environments (i.e. multicomputers, such as transputer arrays), this assumption is often difficult to realize. Processors have no physical memory in common and they communicate using message passing. Although the replicated worker processes of a master-worker model dynamically balance the workload amongst themselves, network traffic and processor overloading due to tuple group servers are potential bottlenecks. Along with the transport layer, required to communicate TS information between components of the application program, a process placement strategy is required which will place components of the application process onto the processor network in such a way as to provide efficient access to TS, and which will place components of TS where they can most efficiently serve the components of the application program.

A side effect of the high level of decoupling between parallel components of a Linda program is that efficiency becomes more of a concern of the implementation and less of a concern of the application programmer. This places pressure on the developers of a Linda implementation to provide an efficient implementation of TS, which should include an efficient process placement strategy. This paper describes a fast (< 1% of the total program compilation time), heuristic algorithm for static process placement, developed specifically for a Linda implementation on transputers. As such, the work forms part of an implementation effort under way at Rhodes University to build an efficient, distributed TS-manager for transputer-based parallel processing systems in the Helios [17] operating environment. To distinguish the experimental effort at Rhodes University from existing commercially available implementations of Linda, our system is known as Rhoda. For the purpose of this paper, the terms Linda and Rhoda are used interchangeably, although Rhoda is generally used to refer specifically to
the Rhodes implementation. The complete system is described in [6], and the distributed run time transport layer in [7].

This paper concentrates on the process placement algorithm developed for use with the distributed TS transport layer, and assumes a rudimentary knowledge of the abstract programming environment presented by the Linda primitives and a conviction of its value to parallel processing. Many informative articles on the fundamental concepts of the Linda approach to parallel programming have appeared in print, some of which are among this paper’s references [2,3,11].

2 Background and fundamental concepts

Distributing tuple space
The pattern matching that is necessary to locate and withdraw tuples (TS is an associative space), and the locking that is required for exclusive access for the purpose of deleting tuples, are usually expensive operations in Linda. A pre-processor in the Rhoda implementation examines the type signatures of tuples at compile time, and groups tuples based on field structure (type, order, and number of fields) and on field content (constant values). Information about tuple group/client interaction is also collected at compile time. The grouping of tuples dramatically reduces run time search times. More importantly, there is no possibility of interference between processes which are searching different tuple groups, so locking requirements can be refined. The compile time analysis performed by the pre-processor is described in more detail in [8].

Compile time partitioning is essential to the efficient and correct functioning of the distributed TS at run time. Components of the application program and tuple groups form an interacting task force of processes in the Helios environment, all of which need to be placed sensibly upon an arbitrary transputer topology. The heuristic algorithm of this paper groups components of both the application program and components of the TS transport layer into logical clusters at compile time, for mapping onto physical processors at run time. The algorithm minimises inter-processor communication (IPC) costs, while guarding against the violation of processor constraints and while attempting to balance relative processor loads. The term cluster is used to refer to a group of components which will be loaded onto the same physical processor.

Process placement
In distributed memory systems, process placement (allocation) is an important design criterion which influences the system’s response time and throughput [4,9,13]. It might be reasonable to expect the speed of a distributed memory system with \( n \) processors, each having a processor speed of \( k \), to be \( n^*k \). Unfortunately this is not the case; the speed attained is almost certainly less. The difference is due to factors such as control overheads, inter-processor communication, unbalanced processor loading, queuing delays, and precedence relationships. The goal of a placement algorithm is to minimise one or more of these factors so as to improve the performance of the system.

Process placement can be either dynamic or static. In dynamic allocation, components are allocated to processors at run time, so the approach has the advantage of being responsive to the run time state of the processors. Disadvantages include increased run time complexity and increased execution time to decide on and implement the altered allocation. A survey of a number of dynamic allocation algorithms are given in [10], [14] and [20].

Static allocation maps components onto processors before the execution of a program. This approach has the advantage of paying allocation costs only once for a program’s execution, but this requires compile time predictions of how the program will behave at run time. Hence the allocation is based on less reliable cost functions than dynamic allocation methods.

The algorithm of this paper falls into the static camp, and follows a heuristic model which is well suited to the logical clustering nature of the Rhoda allocation problem. Heuristic models have been shown [9,13] to be faster, more extensible, and simpler than integer programming and graph theory methods, and in some cases the only technique that can be used to solve difficult placement problems.

By drawing a process graph of a Linda program (as has been done for the figures in this paper), one can expose two important characteristics. Firstly, it reveals that processes communicate only with tuple groups, but not with one another. This decoupling must be considered in a clustering algorithm. Tuple groups may be treated just like other processes, which simplifies the placement problem to one in which there are no first class processes. Secondly, adding additional worker processes to the graph (i.e. replicating existing worker processes) can be achieved without altering the meaning of the program. This is an attribute which this paper endeavours to exploit, by replicating workers in an arbitrary processor topology as much as can be cost effectively accomplished. This takes the responsibility for topology specifications away from the programmer, and gives it to a process placement algorithm. Apart from the obvious advantages to the time and the level of expertise needed to develop a parallel programming solution, this approach aids program portability across (currently only homogeneous) multiprocessor architectures with differing processor topologies and sizes.

It was considered reasonable in the design of a placement algorithm for Rhoda to make the assumption that the number of unique processing components in the application program would not exceed the number of physical processors available.
3 Selecting a process placement algorithm

Before a new algorithm was developed, a number of existing heuristic placement algorithms were considered for the Rhoda placement problem. Here are listed some that were found to fulfill partially the requirements, limitations, and special circumstances of Rhoda.

The simple classification algorithm by Gylys and Edwards [12] searches for pairs of modules such that, when these are assigned to the same processor (fused), the greatest inter-process communication cost is eliminated. This algorithm is simple but does not provide a mechanism to guarantee that the number of clusters found will not be more than the number of available processors. This algorithm does not cater for module replication.

A second algorithm by the same authors [12] defines initial centroids (a mean communication volume) for each candidate cluster. Using these centroids, the algorithm forms a fixed number of clusters to ensure that the number of clusters is not greater than the number of processors. However, finding eligible pairs of processes and/or clusters is very expensive [4]. This algorithm is unable to cater for Rhoda’s module replication, and, in the worse case, it may not converge.

A Module Clustering Algorithm described in [9] forms clusters to obtain the minimum inter-processor cost without considering any constraints. If the clusters meet the load balancing constraints, the algorithm terminates, otherwise a Module Reassignment Algorithm shifts modules from the overloaded processors to under-loaded processors. In the worst case, this algorithm does not converge [4], neither does it cater for module replication.

The Task Response Time Model and the Module Assignment Algorithm with Module Replications [4], were designed primarily to cater for programs with strong precedence dependencies between their modules. Although the algorithm pair caters for module replication, it was designed for a real time system and is heavily oriented towards satisfying minimal time constraints. This makes it very complex, and not well suited for general IPC cost optimization.

These algorithms and others in the literature [15,16,18,19,21,22,23, and 25], all assume that there are more components to be placed than processors. This is not typically the case for the class of Linda programs (master-worker) at which the algorithm of this paper is aimed. There are characteristically more processors than processes. It is an important aspect of the algorithm of this paper that worker processes are replicated in order to maximize processor utilization.

4 The allocation environment

Two sets of parameters are used to describe the allocation environment for the placement algorithm. The first set describes the hardware configuration to be used. In the Rhoda environment, these parameters are obtained from the Helios Resource Map.

\[ n \] the number of transputers in the network.
\[ Td(n,n) \] the connectivity matrix of the transputers in the network. The matrix element \( Td[i][j] \) has a value of 1 if the nodes \( i \) and \( j \) are connected directly. The other element values denote the shortest distance between nodes with no direct links. This distance is calculated in terms of the number of intermediary hardware links.
\[ N(n,p) \] peripheral connectivity matrix for the transputer nodes \( (p = \text{number of peripheral controllers in the network}) \). This matrix describes the nodes in the network that control peripheral devices.

The second set of parameters used to describe the allocation environment characterizes the application program. In the Rhoda environment, these parameters are obtained from a TS analysis module and a cost pre-processor (both part of the compilation pre-processor stage [6]). The following information is gathered in this way:

Physical parameters:
\[ M(n) \] maximum load vector for the transputers in the network.
\[ I(n) \] Initial load vector for the transputers in the network.

Module information:
\[ m \] the number of unique components of which the application program is comprised.
\[ C(m,n) \] a matrix of data transmission costs between the components to be placed. Each cell \( C[i][j] \) denotes the communication cost between components \( i \) and \( j \).
\[ E(m) \] the component load vector. This vector gives the processing cost of each process of the application program and each tuple group.
\[ R(m) \] the component replication vector. A vector element \( R[i] \) denotes the number of instances of component \( i \).
\[ H(m) \] components peripheral hardware requirements vector. This vector specifies the particular hardware requirements (if any) of each component.

5 The allocation problem

The Rhoda allocation problem has been formulated as an optimization problem over the space defined by the transputer network and the components to be placed.

Let \( v \) be number of clusters formed, where \( v \leq n \), \( x_{ij} \) denote inter-cluster communication between \( i^{th} \) and \( j^{th} \) cluster, \( t_i \) denote the possible maximum load permitted
for each cluster, and
\[ O_i \] denote the load of cluster \( i \).
The total inter-cluster communication cost, \( Q \), will be represented by
\[
Q = \sum_{i=1}^{v} \sum_{j=i+1}^{v} x_{ij}
\]
An ideal solution to the placement problem requires that conflicting requirements be satisfied. On the one hand we would like to have an optimally balanced processor load, but this might be achievable only if we are prepared to move large traffic volumes in the network. On the other hand, clustering components to reduce the traffic volumes might produce unbalanced processor loads. Some compromise is necessary, and this is achieved by allowing approximate rather than demanding exactly balanced processor loads.

Let \( \alpha \) be the tolerance accepted for a balanced cluster load. Typical values for the algorithm of this paper will be between 10% and 20%.

The objective of the algorithm is to find the allocation vector \( P(v) \) such that \( Q \) is minimised subject to the following constraints:
\[
\forall i : \alpha_i \leq O_i \leq \alpha_i + \alpha_i
\]
and
- no component is replicated within the same cluster (which would result in pointless context switching),
and
- no tuple group is replicated within the system (which would produce an obvious inconsistency error).

6 Broad outline of the allocation algorithm

The steps involved in the heuristic placement algorithm are as follows:
1. Acquire the relevant application description parameters.
2. Extract the hardware description parameters from the resource map.
3. Verify that any replications required by the vector \( R \) can be accommodated in the physical network while satisfying the replication constraints (i.e. no two identical components may be assigned to the same cluster and no tuple group may be replicated). If the required replication cannot be accommodated by the network, report an error and stop.
4. Modify the communication cost matrix \( C \), component load vector \( E \), and the peripheral hardware requirements vector \( H \) to contain entries for each replicated instance of a program component.
5. Group the components into clusters.
6. Balance the cluster loads to satisfy the tolerance criteria.
7. Calculate the inter-cluster communication costs for the clusters.
8. Allocate clusters that need certain fixed processors using the peripheral connectivity matrix \( N(n,p) \), and the components peripheral requirements vector \( H \).
9. Assign the remaining clusters using the transputer connectivity matrix \( Td \) such that the total IPC cost is minimised.

It is necessary for the algorithm to test whether the network can accommodate the required replications, since the vector \( R \) is supplied to the algorithm. To do this the algorithm first forms clusters without replications in step 3. An examination of the placement components in each cluster, their processing costs, the cluster loads, and the processor loads, makes it possible to decide whether or not the required replication is feasible.

7 An example of process replication

Figure 1 illustrates the process graph of a simple Linda matrix multiplication program. \( M \) denotes the master process, \( W \) denotes the worker process, and \( g0,...,g4 \) denote five unique tuple partitions. For illustration purposes, the exact contents of each tuple group is unimportant. In a real matrix multiplication implementation, \( g0 \) and \( g1 \) might be used to distribute row and column information, \( g2 \) to distribute matrix size information, \( g3 \) to distribute requests for work to be done (eval tuples), and \( g4 \) to return results. The arcs denote information flow and coupling between the components.

![Figure 1 - A process graph of a Linda program for matrix multiplication.](image)

Figure 2 depicts the process graph after replication of the single worker shown in figure 1. Traffic loads have been re-distributed. The processing costs of the master process and tuple groups remain the same as they are not replicated. Similarly, the communication costs of process \( M \) with the tuple groups remain the same. Because the replicated worker processes continuously look for tasks at run time, just like the single worker process, each worker process is assigned the processing cost of the single worker.

Re-distributing the traffic between tuple groups and
the replicated processes entails determining whether a
tuple group will share its traffic between the replicated
workers or will increase the network traffic as the number
of workers increases. The latter occurs when each
replicated process needs its own copy of global data.
Under such circumstances, even if the network has a high
degree of connectivity and many processors, this increased
traffic cost may outweigh the advantages of further
replication. Determining such circumstances falls outside
the scope of this work and is therefore not considered. In
figure 2, \( g_0, g_1, \) and \( g_4 \) share their traffic from figure 1,
while the global traffic from \( g_2 \) and \( g_3 \) must be
transmitted to each new worker.

![Figure 2 - The matrix multiplication process graph
after replication](image)

Inspection of the process graph reveals that a tuple
group communicates with every process. This suggests
that a tuple group should be clustered with the process
having the heaviest communication.

8 Cluster allocation in more detail

Typically, the processing costs of Rhoda application
processes are heavy in relation to those of tuple groups.
A process can therefore be clustered with a number of
tuple groups. Communication costs are expected to be
reasonably small in relation to processing costs (the
approach of this paper is aimed at coarse grained
applications), but are the dominant costs that must be
minimised for an efficient Linda transport layer. The
clustering algorithm therefore aims to minimise the inter­
cluster communication costs. The algorithm starts with a
cluster and assigns components until the cluster is full or
no suitable component can be found. The algorithm then
proceeds to create new clusters until no more clusters can
be created. The main steps in the algorithm are as
follows.

1. Initialise the cluster loads.
2. Select an attached component (a component that
   requires a specific processor) using vector \( H \) as a
centroid (component to be used to select tuple groups
   for this cluster). If there is no attached component,
   then select a component.
3. Update the current cluster load.
4. Select the unassigned tuple group with the highest
   communicating cost that has not already been
   considered for this centroid. The communicating cost
   becomes a candidate inter-module communicating
   (IMC) cost (the communication cost between modules
   resident on the same processor). If all tuple groups
   have already been considered, start a new cluster by
   repeating step 2.
5. Check if the combined load of the selected tuple
   group and the current cluster satisfy the tolerance
   constraint. If not, select the next tuple group by
   repeating step 4.
6. Obtain the highest communicating cost (cut-cost) of
   this tuple group with the other components which are
   not part of this cluster. Compare the cut-cost with the
   IMC. If the IMC is less than the cut-cost, discard this
   candidate and select the next tuple group by repeating
   step 4.
7. Cluster the tuple group and update the current cluster
   load. Repeat from step 4 until no more tuple groups
   can be located for the current cluster.
8. Repeat steps 2 through 7 until all the components
   have been clustered.

The process graph and table in figure 3 illustrate how
the components in figure 2 would be clustered by this
algorithm. In this example, process \( M \) needs an I/O
device. The node controlling the I/O device has an initial
processing load of 20 and a maximum load of 110. The
other nodes in the network all have an initial load 0, and
a maximum load of 100. A minimum inter-cluster
communication cost and a reasonable processor load
balance is achieved when all tuple groups are grouped
with process \( M \).

Tuple groups which are not clustered constitute
clusters on their own. These tuple groups are assigned to
the main clusters when cluster loads are balanced. The
algorithm is therefore guaranteed to converge.

9 Cost functions required

The success of the Rhoda process placement algorithm
relies on the quality of the initial cost functions. Some of
the costs which have to be estimated at compile time are:
- the communication costs between the components to
  be placed,
- the processing costs for the application processes and
  the tuple groups, and
- the initial and maximum loads for the transputer
  nodes in the network.
These estimates are currently supplied by the programmer in the working Rhoda system. The literature does not give much guidance on estimating processing costs and communication costs at compile time. Vranes [24] suggests lines of machine language instructions as an estimate of processing cost, and Chu et al. [5] suggest using branching probability and loop frequencies. Sarkar [20] suggests a more sophisticated execution profile system for SISAL based on frequency and size counts for functions. An investigation into the behaviour of Rhoda programs is currently under way at Rhodes University with the aim of automating this aspect of the preprocessor stage as well.

10 Observations and conclusions

We have presented a new placement algorithm for the Rhoda system. This heuristic algorithm allocates components to nodes in a transputer network with an arbitrary topology. The allocation is carried out in two phases: first logically clustering the components, and then placing the clusters on the transputer nodes. The algorithm operates under the following constraints:

- it is aimed at programs which solve coarse grained problems,
- it does not suggest a topology (Connections between processors are fixed beforehand and all have identical capacities. The algorithm works for arbitrary topologies, but does not suggest an ideal topology.).
- it assumes each node in the network is capable of running a TS server, and
- it assumes all cost functions are supplied beforehand.

Linda has no direct support for precedence relationships. Consequently, the algorithm of this paper does not cater for precedence relationships. Explicit code in the application program is needed to cater for precedence relationships or to implement the scheduling of such processes at run time.

This work has been tailored to the specific characteristics of parallel Linda programs, but remains language independent (Linda is not a language but a language add-on). The algorithm can be included in future language pre-processors for the Rhoda system.

Empirical evaluation of the algorithm has been limited since the distributed TS-managers which support the run-time process distribution of the placement algorithm are still under development, and do not yet support the full functionality of the placement output. In addition, the system has only been tested with relatively small numbers of processing nodes (up to 16), and its performance in massively parallel environments is not guaranteed to be as good. However, since the goal of the placement algorithm is to replace the existing manual approach with an automatic technique at compile time, the new algorithm was evaluated by comparing its efforts with the manual placements of Linda programs selected from the three classes of Linda programs which have been identified [2]. In all tests [8] the placements achieved were found to be as good as the best hand tailored placements known (the measured speedup was at least as good). Figure 4 shows the speedup attained as the number of worker processes is increased for two benchmark applications.

References


Acknowledgements: The authors are grateful for the constructive comments of the anonymous referees.
Notes for Contributors

The prime purpose of the journal is to publish original research papers in the fields of Computer Science and Information Systems, as well as shorter technical research papers. However, non-refereed review and exploratory articles of interest to the journal's readers will be considered for publication under sections marked as Communications or Viewpoints. While English is the preferred language of the journal, papers in Afrikaans will also be accepted. Typed manuscripts for review should be submitted in triplicate to the editor.

Form of Manuscript
Manuscripts for review should be prepared according to the following guidelines:

- Use wide margins and 1 ½ or double spacing.
- The first page should include:
  - title (as brief as possible);
  - author's initials and surname;
  - author's affiliation and address;
  - an abstract of less than 200 words;
  - an appropriate keyword list;
  - a list of relevant Computing Review Categories.
- Tables and figures should be numbered and titled. Figures should be submitted as original line drawings/printouts, and not photocopies.
- References should be listed at the end of the text in alphabetic order of the (first) author's surname, and should be cited in the text in square brackets [1, 2, 3]. References should take the form shown at the end of these notes.

Manuscripts accepted for publication should comply with the above guidelines (except for the spacing requirements), and may be provided in one of the following formats (listed in order of preference):

1. As (a) \LaTeX\ file(s), either on a diskette, or via e-mail/ftp — a \LaTeX\ style file is available from the production editor;
2. In camera-ready format — a detailed page specification is available from the production editor;
3. As an ASCII file accompanied by a hard-copy showing formatting intentions:
   - Tables and figures should be on separate sheets of paper, clearly numbered on the back and ready for cutting and pasting. Figure titles should appear in the text where the figures are to be placed.
   - Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin, if they are not clear in the text.

Further instructions on how to reduce page charges can be obtained from the production editor.

4. In a typed form, suitable for scanning.

Charges
Charges per final page will be levied on papers accepted for publication. They will be scaled to reflect scanning, typesetting, reproduction and other costs. Currently, the minimum rate is R20-00 per final page for \LaTeX\ or camera-ready contributions and the maximum is R100-00 per page for contributions in typed format.

These charges may be waived upon request of the author and at the discretion of the editor.

Proofs
Proofs of accepted papers in categories 3 and 4 above will be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Note that, in the case of camera-ready submissions, it is the author's responsibility to ensure that such submissions are error-free. However, the editor may recommend minor typesetting changes to be made before publication.

Letters and Communications
Letters to the editor are welcomed. They should be signed, and should be limited to less than about 500 words.

Announcements and communications of interest to the readership will be considered for publication in a separate section of the journal. Communications may also reflect minor research contributions. However, such communications will not be refereed and will not be deemed as fully-fledged publications for state subsidy purposes.

Book reviews
Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertisement
Placement of advertisements at R1000-00 per full page per issue and R500-00 per half page per issue will be considered. These charges exclude specialized production costs which will be borne by the advertiser. Enquiries should be directed to the editor.

References
Contents

Forward ................................................................. i
Referees ........................................................................... ii

Machine Translation from African Languages to English
DG Kourie, WJ vd Heever and GD Oosthuizen ..................... 1
Automatically Linking Words and Concepts in an Afrikaans Dictionary
PZ Theron and I Cloete .................................................. 9
A Lattice-Theoretic Model for Relational Database Security
A Melton and S Shenoi .................................................. 15
Network Partitions in Distributed Databases
HL Viktor and MH Rennhackkamp .................................. 22
A Model for Object-Oriented Databases
MM Brand and PT Wood ............................................... 27
Quantifier Elimination in Second Order Predicate Logic
D Gabbay and HJ Ohlbach ............................................. 35
Animating Neural Network Training
E van der Poel and I Cloete ........................................... 44
HiLOG - a Higher Order Logic Programming Language
RA Paterson-Jones and PT Wood .................................... 53
ESML - A Validation Language for Concurrent Systems
PJA de Villiers and WC Visser ....................................... 59
Semantic Constructs for a Persistent Programming Language
SB Sparg and S Berman ............................................... 65
The Multiserver Station with Dynamic Concurrency Constraints
CF Kriel and AE Krzesinski .......................................... 75
Mechanizing Execution Sequence Semantics in HOL
G Tredoux ....................................................................... 81
Statenets - an Alternative Modelling Mechanism for Performance Analysis
L Lewis ............................................................................. 87
N Pendock ...................................................................... 95
Galileo: Experimenting with Graphical User Interfaces
R Apteker and JM Bishop ................................................. 99
Placing Processes in a Transputer-based Linda Programming Environment
Accessing Subroutine Libraries on a Network
PH Greenwood and PH Nash .......................................... 117
A Multi-Tasking Operating System Above MS-DOS
R Foss, GM Rehmet and RC Watkins ............................... 122
Using Information Systems Methodology to Design an Instructional System
BC O'Donovan .................................................................. 126
Managing Methods Creatively
G McLeod ......................................................................... 131
A General Building Block for Distributed System Management
P Putter and JD Roos .................................................... 141