South African
Computer
Journal
Number 6
March 1992

Suid-Afrikaanse
Rekenaar-
tydskrif
Nommer 6
Maart 1992

Computer Science and Information Systems

Rekenaarwetenskap en Inligtingstelsels

The South African Computer Journal

An official publication of the South African Computer Society and the South African Institute of Computer Scientists

Die Suid-Afrikaanse Rekenaartydskrif

'n Amptelike publikasie van die Suid-Afrikaanse Rekenaarvereniging en die Suid-Afrikaanse Instituut vir Rekenaarwetenskaplikes

Editor
Professor Derrick G Kourie
Department of Computer Science
University of Pretoria
Hatfield 0083
Email: dkourie@dos-lan.cs.up.ac.za

Subeditor: Information Systems
Prof John Schochot
University of the Witwatersrand
Private Bag 3
WITS 2050
Email: 035ebrse@witsvma.wits.ac.za

Production Editor
Prof G de V Smit
Department of Computer Science
University of Cape Town
Rondebosch 7700
Email: gds@cs.uct.ac.za

Editorial Board

Professor Gerhard Barth
Director: German AI Research Institute

Professor Judy Bishop
University of Pretoria

Professor Donald Cowan
University of Waterloo

Professor Jürg Gutknecht
ETH, Zürich

Professor Pieter Kritzinger
University of Cape Town

Professor F H Lochovsky
University of Toronto

Professor Stephen R Schach
Vanderbilt University

Professor S H von Solms
Rand Afrikaanse Universiteit

Subscriptions

	Annual	Single copy
Southern Africa:	R45_00	R15_00
Elsewhere	$45_00	$15_00

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Computer Science Department, University of Pretoria, Pretoria 0002
Phone: (int)+(012)+420-2504 Fax: (int)+(012)+43-6454 email: dkourie@rkw-risc.cs.up.ac.za

Sigma Press

Guest Contribution

This guest contribution is a slightly edited report to the Foundation for Research and Development (FRD) drawn up by Ed Coffman. Ed was an FRD-sponsored guest at the 6th South African Computer Research Symposium. The report was not originally intended for general distribution. Rather, it was specifically compiled for the FRD and its staff. I am therefore grateful to both the FRD and the author for agreeing to its publication in SACJ. I believe that it contains several incisive observations that merit further thought and discussion amongst South African computer scientists. (Editor)

Impressions of Computer Science Research in South Africa

E. G. Coffman, Jr.
AT&T Bell Laboratories, USA

In commenting on the cross section of computer science research in South Africa, I will use the classification in the table of contents of the "Summary of Awards: Fiscal Year 1989," a document published recently by the US National Science Foundation. Of the 5 categories, I will treat Numeric and Symbolic Computation as inappropriate for the discussion below. In this category I noted no research in the computer science setting in South Africa. It is also common in the US and elsewhere to place this effort in other departments, e.g., departments of mathematics or applied mathematics.

Of the remaining categories I found South Africa to be strongest in software systems and engineering, to have a substantial investment in computer systems and architecture, and to be weakest in computer and computation theory.

The coverage in software systems and engineering (SSE) was broad, topical, and similar in scope to that in US universities. Technology transfer and the corresponding relations with industry seemed to be in place or developing along promising lines. I comment in passing that this was rather surprising to me. In the US the development of SSE within university departments has lagged behind almost all other disciplines of computer science. A primary problem has been the insatiable appetite of industry for all Ph.D. graduates in the SSE field.

The investment in parallel processing, computer networks, and distributed computing appears sound, although I expected to see a greater emphasis on mathematical foundations (see my remarks below), particularly in the parallel algorithms area. Given current resources, South African institutions are doing remarkably well in computer science research. But computer science is a fundamentally important course of study, beginning at an early age and extending through graduate Ph.D. research; I take this as sufficiently obvious that I need not dwell on justifications. With this in mind, and with the necessary resources in hand, South Africa should, in my opinion, expand and consolidate its computer science research effort, increase its visibility in the international arena, and correct the rather thin distribution of graduate research among universities.

I can see much of this proceeding along present lines, but I would strongly recommend a concerted development in computer and computation theory (CCT), education and research; this is mainstream computer science and forms the basis for virtually all other fields of study within computer science. It is by no means absent in South Africa curricula, but it appears to be under--represented in advanced studies and Ph.D. level research.

At the graduate level CCT is heavily mathematical. I understand that mathematical foundations are supplied by mathematics departments in certain cases. This is not ideal, but workable and it is justified by limited resources. However, it is important that mathematics departments not regard this as a mere service; faculty will have to make a major commitment to theoretical computer science, publishing in its leading journals (e.g. SIAM Journal of Computing, Journal of the ACM, Journal of Algorithms, Algorithmica, Journal of Computer and Systems Sciences, Theoretical Computer Science, etc.), and providing the supervision of theses sponsored by computer science departments and leading to degrees in computer science. I would also encourage active participation in the international computer science "theory" societies and their meetings; two highly prestigious examples of the latter are the annual Symposium on the Theory of Computing and the Foundations of Computer Science conference.

Returning to the thin distribution of computer science research, I would make the following point. If the current situation is only a stage of development - i.e., if further resources (both human and financial) can be counted on to bring at least a few of the departments to a critical mass - then little needs to be said beyond the earlier remarks. Critical mass is hard to define, but calls for adequate, expert coverage of mainstream computer science re- search. In view of the breadth of this research, 8-10 Ph.D. full- time-equivalent faculty would seem to be barely adequate; with the usual clumping of faculty in specific research areas, more would be expected. South Africa has a talent base such that there is little doubt that such departments would achieve a much wider international recognition.

On the other hand, if resources remain fixed at current or even slightly retrenched levels, then I would recommend consolidation to achieve the same goals on a smaller scale. Within a university this can often be done by establishing interdisciplinary, degree-granting laboratories or institutes of computer science, which bring together the computer science efforts located in various departments other than computer science, such as electrical engineering, industrial engineering, business/-management science, mathematics, and operations research. The idea is to enjoy the advantages (opportunity, synergy, awareness, etc.) to both students and faculty of reasonably large computer science programs. There are many examples of such intramural laboratories in North America and Europe.

This approach could also be considered among universities within a confined geographical area, admittedly with greater difficulty perhaps. The Institute of Discrete Mathematics and Computer Science connecting Princeton University, Rutgers University, AT&T Bell Laboratories, and Bell Communications Research is a possible model. Examples in South Africa might consist of universities and research institutions on the Reef or those in the Western Cape (just to mention those with which I'm a little familiar).

As a final comment, I should note that my impressions have been based on limited information which may not give a representative picture. I am sure that my reactions will be appropriately discounted where I have been off target.

Editor's Notes

Prof John Schochot has graciously accepted to be SACJ's subeditor for papers relating to Information Systems. Authors wishing to submit papers in this general area should please contact him directly. I look forward working with John, and to a significant increase in IS contributions in future.

The hand of the new production editor, Riël Smit, will be clearly evident in this issue. Those papers not prepared in camera-ready format by the authors themselves were prepared by him in TEX. He will be announcing revised guidelines for camera-ready format in a future issue. If you use TEX or one of its variations, Riël would be happy to provide you with a styles document to SACJ format.

At last some Department of National Education committee has decided that SACJ should now be on the list of approved journals. This places it amongst the ranks of some 6800 other journals. These include not merely a number of ACM and IEEE Transactions but also such journals as *Ostrich*, *Trivium*, *Crane Bag*, *Koers*, *Mosquito News*, *Police Chief*, *Connoisseur*, *Lion and the Unicorn*, *About the House* and *Ohio Agricultural Research and Development Center Department Series ESS*. You will recall that in 1990 this same committee decided that, if judged on its own merits, SACJ did not deserve to be on the illustrious list. In the absence of other evidence, we must assume that the sole reason for its revised decision is that SACJ's predecessor, Quæstiones Informaticæ, was there. (I have a secret suspicion that the committee liked that name.)

It is my understanding that for official purposes, all journals on this list are regarded as *equally* meritorious, and all of them are more meritorious than *any* conference proceedings. What does all of this mean?

The momentous implication of the committee's deliberations is that the State will not give your institution a single cent for anything that you publish in SACJ. Instead, the State and your institution will scrupulously keep a score of the annual number of publications that count - but actually don't - because someday they might! And to encourage your enthusiastic participation in this Alice in Wonderland exercise, your institution might actually give you some of the standard subsidy funding that the State should have provided according to its own formulae, but didn't.

You will not be allowed to use this money to buy yourself a car - not even a casual meal. You may only use it to finance activities that are provably directed towards producing more papers in approved journals. The great consolation, of course, is that you will not be required to pay income tax on this money. The only tax involved will be the VAT component when you spend it in an approved manner. As a good computer scientist who enjoys recursion, my vote would be that all such revenue collected by the State should be earmarked to be placed in the pay packets of committee members who decided that SACJ should be approved.

If you publish in these approved journals with sufficient regularity and enthusiasm you will almost deserve to be regarded as a researcher. What you additionally need to do, is to ensure that you befriend and impress at least three overseas referees. You then apply to the FRD for official recognition as a researcher, and if they are sufficiently impressed, they will give you more of the non-taxable kind of money that you need to spend on research to publish in approved journals.

Derrick Kourie
Editor

An Implementation of Linda Tuple Space under the Helios Operating System.

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

Abstract

We discuss the implementation of Rhoda, our Linda[i]-like Tuple Space server which runs under the Helios[ii] operating system. The approach analyses and partitions tuple space at compile time in order to reduce the run time overhead of tuple matching. The interaction between the concurrent processes and the tuple partitions is used as the basis for distributing the partitions and processes in the network. The paper presents some empirical results and discusses the suitability of the Helios nucleus for supporting the approach.
Keywords: *distributed systems, parallel processing, transputer, Linda, Helios.*
Computing review categories: *C.2.4, D.1.3, D.3.4*

1. Introduction

The Linda programming paradigm is a simple and elegant approach to parallel processing, based on the concept of *generative communication* [7]. This is a form of communication in which an active message (or tuple) may be converted through process creation and evaluation into a passive value. Linda is not a language per se, it is a small set of control and coordination operations which can be imbedded into a programming language (typically one of the well known imperative programming languages) to introduce or enhance parallel capabilities.

At the center of the Linda programming model is a shared, associative memory called *tuple space* (*TS*). Objects called *tuples* are output to - and input from - *TS* by components of the application program. At the abstract programming level, *TS* is global to all components of a parallel program, even though they might be executing on individual processors which have no physical memory in common. Parallel components of an application program (processes or tasks) never communicate directly with each other, only with *TS*. Consequently, *TS* acts as a decoupling agent. This reduces program complexity by allowing parallel programs to be decoupled both spatially and temporally.

A *tuple* is a sequence of typed (actual or formal) fields, rather similar in concept to a parameter list. In addition to passive data values, the contents of a tuple may be a reference to active executing or executable code. Tuples are selected from *TS* by associative matching.

To communicate with tuple space, Linda provides a set of six primitive operations:

out(t) output tuple *t* to the *TS*

eval(t) evaluate tuple *t* (This operation is similar to *out* in that it outputs tuple *t* to *TS*, but *t* may be an active tuple whose result is yet to be evaluated.)

in(s) input a tuple *t* from *TS* which matches the template *s* (If no matching tuple is available, the requesting process is suspended until one becomes available. If more than one matching tuple exists in *TS*, an arbitrary matching tuple is returned. The tuple is removed from *TS*.)

rd(s) read a tuple *t* from *TS* which matches the template *s* (*rd* is conceptually very similar to *in*. It returns a copy of a tuple without removing it from *TS*.)

inp(s) and *rdp(s)* similar in function to *in* and *rd*, these operations are predicates which attempt to match a tuple *t* to the template *s*, and return a failure value immediately if no match is found. If the operation succeeds, both a tuple and a success value are returned.

These operators communicate only with *TS*, and none of the high-level system services which distributed operating systems usually superimpose upon their transport layers are provided. A Linda system may make use of the existing low-level transport layer provided by a distributed operating system, or may require a specialized transport layer to be written. In the former case, application programs should be unconcerned about the particular target architecture, and about whether they will run under an operating system or as standalone programs.

A number of informative articles on the use of the

Linda approach to parallelism have already appeared in print, some of which are listed among this paper's references [2, 3, 8, 1]. We do not concern ourselves in this paper with presenting a suite of tutorial examples, or with persuading readers of the merits of this programming approach; we concentrate on implementation issues, assuming a rudimentary knowledge of the abstract programming environment presented by the Linda primitives, and a conviction of its value to parallel processing.

This paper represents a status report on an implementation effort underway at Rhodes University to build an efficient, distributed *TS*-manager for transputer-based parallel processing systems in the Helios 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.

2. An overview of the Rhoda implementation

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 transport layer which will allow *TS* to be simultaneously visible to all components of the application program. A range of strategies can be used to implement a global *TS* in a parallel processing environment in which processors do not have a shared physical memory. At one extreme, *TS* could be stored at a dedicated central node which is accessed via a transparent message routing system. Even if run-time hashing is used to improve search performance in this approach, delays caused by message routing can degrade the performance of the system, and a single centralized *TS*-manager can become a bottleneck which impedes massive parallelism. At the opposite end of the implementation spectrum, *TS* could be replicated in each processing node, and local *TS*-managers could transparently propagate changes through the network. A major encumbrance to this approach is the provision of a locking mechanism which ensures that program components wishing to remove tuples from *TS* are given exclusive delete access.

The Rhoda implementation under Helios uses a centralized *TS* model, but partitions *TS* with the view to reducing the run time matching overheads of Linda operations, and so that distributed *TS*-managers can be used to control a small (possibly localized) group of related tuples. A partitioned *TS* is in contrast to the Linda programming assumption of a single shared *TS*. This section provides a brief overview of the Rhoda compilation path, which adds additional housekeeping information to source programs to enable them to work with the partitioned model described in the remainder of the paper.

Figure 1 depicts the compilation phases present in the Rhoda compiler. C is currently used as the host language for Rhoda. Apart from the normal C pre-processor, Rhoda makes use of a second pre-processor to compile and pass a list of all tuple operations, and the program components which issue them, to a tuple analysis module. This module analyzes *TS* interaction with components of the application program, to divide tuples into groups based on their structure, and to suggest an appropriate placement strategy for tuple groups and application program components in the processor network. The grouping of tuples is an integral feature of the Rhoda implementation, and is described in more detail below. By grouping tuples at compile time, a substantial matching overhead is avoided at run time. Distinct tuple groups also facilitate the distribution of *TS* in the distributed memory environment. The initial placement strategy of the Rhoda system divides a task force (application program components and *TS*-managers) into appropriate process clusters for placement on the processor network, in positions which will incur a relatively low inter-cluster communication cost. This aspect of *TS* analysis is described in more detail by de-Heer-Menlah [5].

The tuple groupings determined by the analysis module are used by the Rhoda pre-processor to translate ideal Linda syntax into concrete C syntax which opens, closes, and addresses file-like tuple partitions. A Rhoda program usually contains a number of components (for example, a master process and a worker process), which must all be present during tuple analysis. The output of the pre-processor stage is a series of C programs, one for each unique parallel component of the original source.

3. Partitioning tuple space

The syntax for tuple fields makes provision for actual fields in the form of constant values or run-time expressions, and for formal fields denoted by program variables which are preceded by the "?" character. The Linda input primitives provide tuple templates against which tuples placed in *TS* by output primitives are compared. It is common practice for Linda programmers to use a constant valued field to ensure a correct tuple matching. For example, the initial field of a tuple is frequently a string literal. The matching process is potentially a computationally expensive operation, and is an area in which efficient implementation is a crucial issue.

The tuple templates of Linda operations are matched by associatively searching tuples within *TS* which have the same structure. Examples of syntactically correct, matching Linda primitive operations might be:

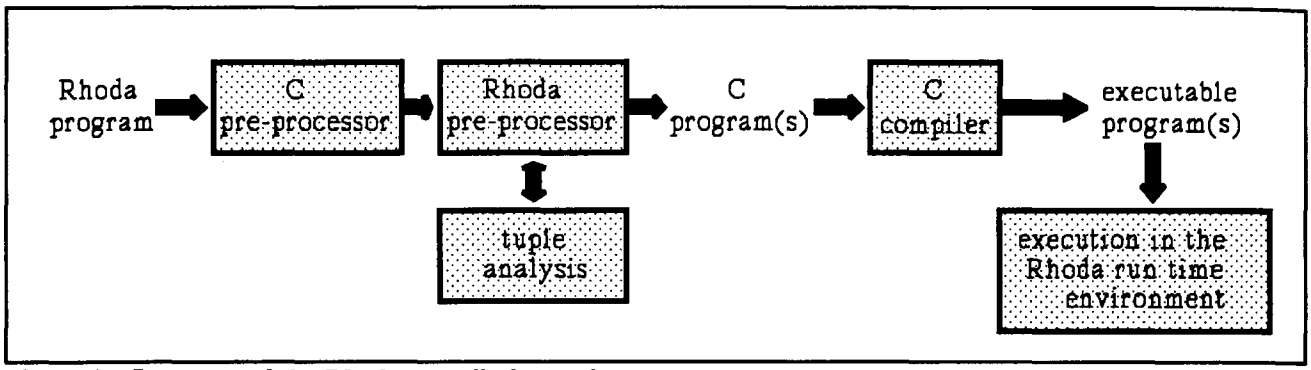

Figure 1 - Structure of the Rhoda compilation path.

```
out("element", 3, 4, value)
in("element", i+1, j, ?result)
```

If the variables *value*, *i*, *j*, and *result* were all declared to be of the same type (integers for example), then the two tuples manipulated by the above *in* and *out* operations would be regarded as having the same *structure*, *viz.* a string constant followed by three integer fields. The actual expressions (*value*, *i+1*, and *result*) would contribute their current run-time values to the *out* operation's tuple and the *in* operation's template. The formal field (*result* in the template used for the *in* operation in this example) would return the *value* of a tuple whose first three fields match those of the template. For example, if the values of *i* and *j* were 2 and 4 respectively, and the tuple *("element", 3, 4, 12)* were present in *TS*, then *result* would have the value 12 after execution of the *in* operation.

It is possible to detect at compile time that a Linda input *in("row", ?i)* could be matched to any of the following tuples

("row", 4) ("row", 10) ("row", 500)

with the consequent actual to formal assignment for the variable *i*.

It is likewise clear at compile time that the template *("result", ?i)* will not match any of the following tuples, no matter what the run time values of variables are, because the type, order, or number of fields differ.

("row", 6.42) (j, "row") ("matrix size", 50, 20)

Nor will it match a tuple such as *("col", 4)* whose type, order, and number of fields agree, because the value of the compile time string literal field of the tuple and template differ.

Since operations on one tuple group can never match tuples in another group, the partitioning of tuples into disjoint groups at compile time can be done safely. Tuple operations can first be coarsely classified into mutually exclusive groups based on their field structure (type, order, and number of fields). A subsequent finer partitioning can be done based on field information; tuples having the same field structure, but different compile time constant values in a particular field, cannot be matched.

Once compile time constants have been examined and tuple groups have been formed, the constant values are no longer of any use since all tuples (and tuple templates) within a particular group will have identical constant values in their common constant fields. Discarding such constant fields is a further compile time optimization. For example, Linda operations which refer to the tuples

("row", i, j) ("row", i+1, j+1) ("row", ?m, ?n)

will be modified to calls to the same *"row"* tuple group using the tuples

(i, j) (i+1, j+1) (?m, ?n)

Efficient searching and matching strategies can now be devised for particular tuple groups. Taken together, the dramatic reduction in the scope of a tuple search and the reduction in the number of (mostly string) fields provide a major improvement in the run time overhead of tuple matching. Analysis of the actual to formal relationships of the corresponding fields of a tuple template and its *TS* group can lead to further efficiencies in run-time matching. Zenith [13] suggests a number of instances in which a general tuple matching algorithm can be reduced to a far simpler operation.

It is possible to take the *TS* analysis further by considering which components of the application program make use of each tuple group. This provides information for the placement of *TS* groups relative to the program components which they serve in the processor network, and allows an hierarchical *TS* dependency structure to be built, thereby facilitating an hierarchical naming scheme for the distributed *TS*. For example, a Linda application program comprising three parallel component processes, P_1, P_2, and P_3, coordinates its parallel activity using three different tuple structures which can be grouped at compile time into three independent tuple groups. All

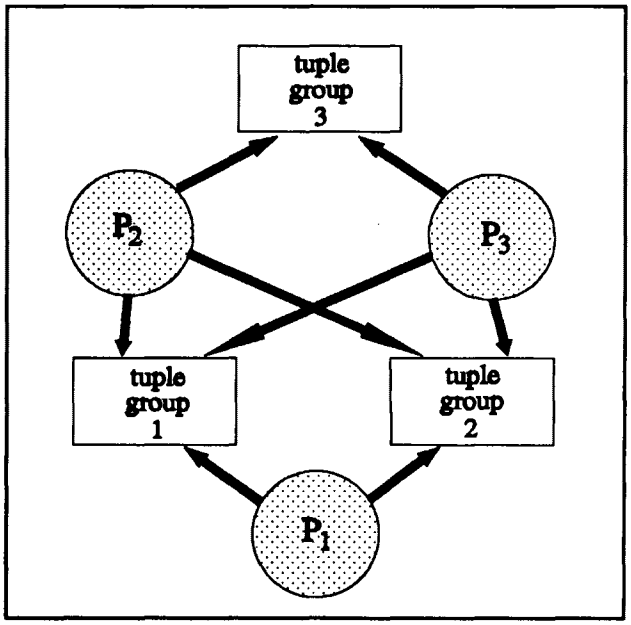

Figure 2 - Three parallel application processes referencing three tuple groups.

three processes make use of tuple groups 1 and 2, while only P_2 and P_3 make use of tuple group 3. Figure 2 demonstrates the interaction of component processes and *TS* groups for this example, and figure 3 shows the hierarchical relationship which results from the partitioning of *TS*.

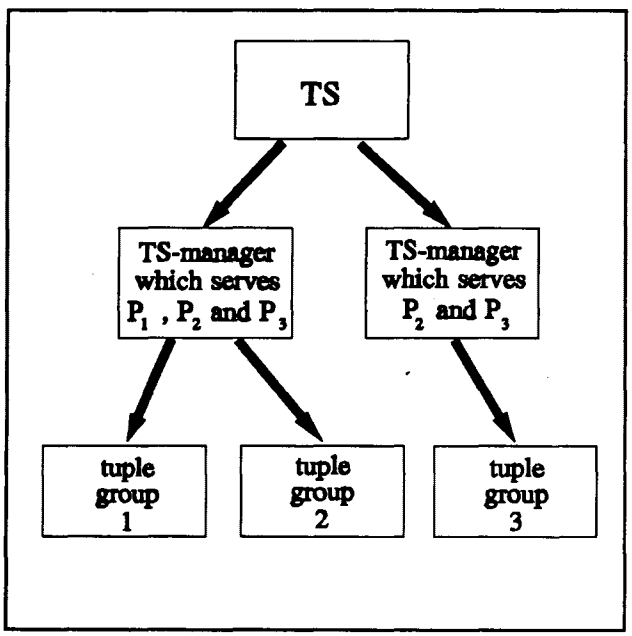

Figure 3 - The hierarchical relationship of TS.

4. The Helios environment

Helios [11] is a UNIX[iii]-like[iv], distributed, parallel operating system. The Helios nucleus, which must be present on all Helios processors, provides a small kernel (for managing message passing, hardware resources, and list handling) and a number of basic *servers* which integrate the processor into the global environment. Helios servers are based on the conventional *client/server* model, in which a server task manages a resource on behalf of its clients. The minimum set of servers required by a Helios processor includes a loader, a processor manager for managing the computing resources of the processor and for responding to requests to access executing tasks, and a number of I/O controller (IOC) processes. Additional operating system servers might be loaded on particular processors of the network to support specific facilities. These include the window server, the disk server, the RS232 server, the console server, the network server responsible for distributing and controlling the nucleus, and so on. Most importantly, Helios provides a server library facility which can be used to implement additional servers for the system using a standardized general server protocol.

To facilitate communication between distributed tasks, the process manager of the Helios nucleus spawns an IOC process for each new task, which acts as the task's intermediary with the rest of the system. The IOCs on one processor route requests to named objects on behalf of their tasks by referencing a central name table. If a name is present in the table, the IOC passes the request directly to the server whose port is represented in the entry, if not, a distributed search is initiated. Provided the name exists elsewhere, an entry is installed in the name table so that subsequent uses need not cause a search. Each physical link of the processor also has an IOC, responsible for handling distributed searches and requests from remote tasks to local servers.

Helios supports an hierarchical naming scheme for all objects in the network. Each sub-network (or cluster) is given a unique name, and the names of objects within sub-networks (processors, files, file systems, servers, tasks, and so on) must not conflict when they are identified by their position in the network hierarchy. All objects in Helios present a directory interface through which any information specific to the object may be examined and manipulated. This form of network addressing is a logical extension of the conventional hierarchical file system adopted by many operating systems. Most Helios commands which access the hierarchical directory structure are generic utilities which do not differentiate between different types of object in the hierarchy. Figure 4 is an example of the hierarchical naming scheme presented for a sub-network. In this example, the cluster comprises three processors (*00*, *01*, and *02*) and an I/O server. The Helios nucleus on each processor comprises a *tasks* directory, a number of link

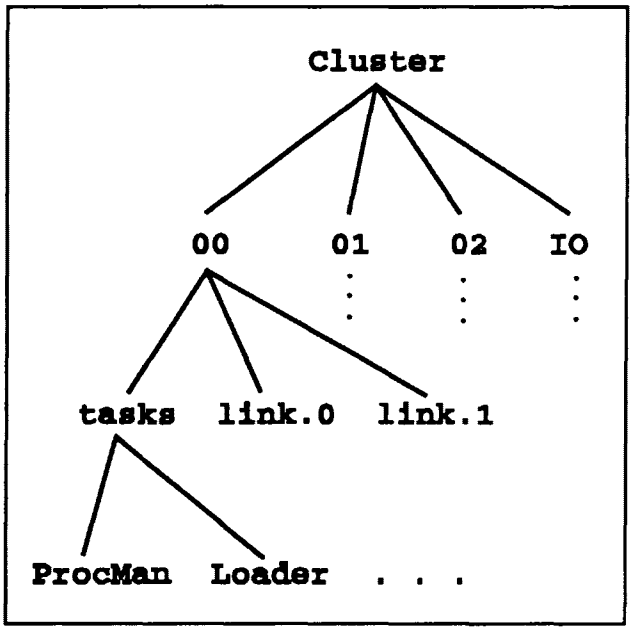

Figure 4 - Helios objects represented as a hierarchical directory.

IOCs, and so on. Objects within the *tasks* directory are the currently active tasks on that processor.

Network naming is a totally distributed service in Helios, and a distributed name server is at the heart of the naming scheme. It provides an hierarchy of names for an otherwise arbitrary topology structure.

Helios servers may be localized or distributed. All servers adhere to the same *general server protocol* (GSP). They are written as a set of calls to a distributed server library, plus a set of application specific functions [6]. The server library provides support for a message decoder and despatcher, which waits for messages on a specified port, validates them as GSP messages, and forks a worker process to execute a service procedure. The forking of a service procedure is an important aspect of Helios's support for distributed servers. The server essentially consists of the despatcher process until such time as a request arrives from the server's request port (looked up in the name table by the name server on behalf of a client process). To handle the request, the despatcher process spawns a separate process to execute the required function. This happens for each request. Normally, this process returns a reply at the end of the desired service and terminates. However, if the function performed by the spawned service process is an *open* operation (as in "open a file"), the service process remains active after a reply has been sent, and acts as a proxy server for any stream messages which are directed to it, until it is *closed.*

5. The tuple space server under Helios

The hierarchical naming structure of Helios provides an ideal support environment for a *TS* which can be grouped in such a way as to expose hierarchical relationships. *TS* in the Helios-Rhoda system has been implemented along similar lines to a directory based file server, in which each "file" corresponds to a tuple group capable of manipulating streams of tuples with the same *type signature* (as grouped by the tuple analysis module in the Rhoda pre-processor). By adopting the Helios environment, we gain directory and sub-directory structures, and their concomitant protection mechanisms, at no additional cost to the implementation; they are already part of the existing Helios server protocol and libraries. *TS* is implemented as a Helios server, using the standard GSP. The Rhoda *TS* server integrates very smoothly with rest of the Helios system because it honours this protocol, and the generic utilities which operate on other Helios objects are able to operate on *TS* structures as well. Each tuple group falls under the control of a *TS*-manager, but different tuple groups might be placed under the control of different *TS*-managers distributed across the network.

The Helios strategy of routing all GSP requests to a single port, and then spawning (by way of the despatcher) independent processes to service each of them, enables several clients to access the same server concurrently. In the *TS* server, such GSP protocols are occasional events, which open a tuple group and create a proxy process within the server to manage access to the tuple group on behalf of a particular client. Thereafter, Linda operations are reduced to direct communications between the client and the proxy process.

All client processes which produce or consume tuples with a particular type signature will open the same tuple group. To gain access to a tuple group, a client process must declare a tuple group descriptor, specifying a name for the group and a type signature for tuples which conform to the group. Thereafter, it is able to open the tuple group, use it, and close it again, simply by making appropriate *TS* server calls and supplying the appropriate name of the tuple group along with each such operation. The pre-processor prefixes each *TS* operation in the source code with a tuple group descriptor for this purpose. The first reference to a *TS* server initiates a dynamic network search and establishes a connection path between the client and its proxy service process, enabling the two to exchange messages without regard to the system topology. Thereafter the client process has a point-to-point virtual link to a dedicated proxy process, which manipulates the tuple group on its behalf, until it requests a close operation, at which time the proxy process terminates. Since several clients are able to access the same tuple group simultaneously, the *TS* proxy processes assume the responsibility for locking the tuple group and coordinating requests during operations which update the group.

Each tuple group within the *TS* server is a data structure which contains control information such as its name, size, number of clients, protection attributes, mode

information, parent directory, a locking semaphore to ensure exclusive update access, and so on. It also keeps track of current tuple values, and keeps a queue of blocked clients together with their transaction templates.

To handle the blocking semantics of the Linda primitives *rd* and *in*, proxy server processes are suspended until a suitable tuple arrives. This has the effect of suspending the client as well, while it awaits a reply from the server. A *TS* proxy server handles an unmatched request by queueing it, along with a semaphore, in the waiting queue for the tuple group it supports. The proxy then suspends itself by waiting on the semaphore. Each time a new tuple arrives for a particular tuple group as a result of an output operation, the waiting queue for that group is searched, comparing the new tuple to pending requests. To satisfy the different semantics of the Linda *in* and *rd* operations, a pass is made through the queue, locating each matching *rd* transaction which can be completed, up to the first matching *in* transaction. The *in* transaction must also be completed, and will consume the new tuple. If there is no pending *in* operation, the tuple is added to *TS* in the normal way. Completion of a pending transaction is achieved by allowing the output primitive to complete its transaction, and then waking those proxy processes whose outstanding transactions can be satisfied.

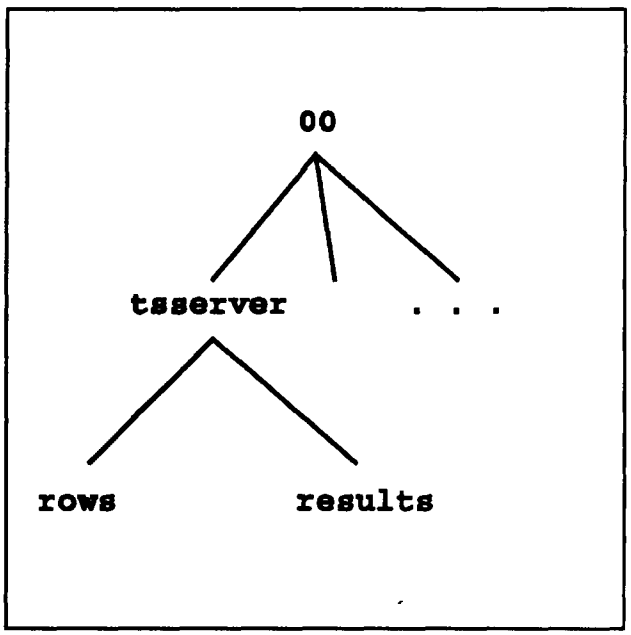

Figure 5 - A TS server in the Helios hierarchical structure.

Figure 5 illustrates the integration of the *TS* server into the hierarchical Helios naming structure. In this example, a *TS* server has been initiated on processor *00*, and one or more client processes have opened two tuple groups, named *rows* and *results*. Client processes residing anywhere within the network are able to open either of these tuple groups, and a proxy service process will be spawned on processor 00 (within this *TS* server) for each such request. So, process *A*, executing on processor *02*, which uses both of these tuple groups, will cause two independent proxy processes to be spawned within this *TS* server. Process *B*, executing on processor *00*, which uses tuple group *results*, will cause yet another proxy process to be spawned. There might well be additional *TS* servers residing on other processors in the network and managing access to other tuple groups, provided their names do not conflict with the name of this server in the naming hierarchy. Processes *A* and *B* could well be making use of these additional servers as well.

The Rhoda system makes use of the Helios processor manager to implement the *eval* primitive operation. The processor manager is a server which is present as part of the essential nucleus on all Helios processors. It sees to the creation and management of tasks on that processor, and is able to load and execute programs on behalf of clients executing on remote processors.

During the Rhoda pre-processing phase, each source function that is invoked by *eval* is transformed into a free-standing executable program by encapsulating it in a suitable code skeleton. Since an *in* or *rd* template can never match an active tuple, tuples generated by *eval* operations will be placed into their own active tuple groups. When a *TS* server is invoked, it must be supplied with the names of the processors on which it may execute active tuples. The *TS* server spawns a manager task for each such target processor, and establishes a link to that processor's processor manager. These manager tasks are responsible for monitoring the *TS* server's active tuple group, and remotely invoking processes to evaluate tuples when necessary.

Remote program invokation is a relatively expensive operation, particularly if the executable code has to be fetched from a central filing system[V]. The Rhoda implementation alleviates this overhead by modifying the skeleton that encapsulates *eval*-ed functions so that, once invoked, they repeatedly fetch and execute active tuples until a request for an active tuple matching their particular type fails. This has the same effect as reinvoking the function for every tuple of that type, but is considerably more efficient.

For monitoring purposes, each Rhoda *TS* server also provides statistical information, which appears to a client process wishing to monitor *TS* as just another set of tuples, which can always be read (i.e. they are created "on the fly" when they are requested).

6. Observations and conclusions

A number of desirable qualities are present in the Helios-Rhoda implementation:

- The system is able to execute on any transputer network with an arbitrary topology.
- With *TS* implemented as a distributed server (essentially present as part of the system nucleus on all processing nodes) no processors are dedicated to

supporting *TS*, or are excluded by the presence of a *TS*-manager from supporting part of the application task force.

- The division of *TS* into individually addressable tuple groups reduces the potentially expensive operation of associative matching to a far simpler operation, and provides a natural mechanism for partitioning *TS* space into distributable sub-spaces.
- The hierarchical structure of *TS* partitions and the inheritance of the normal filing system security mechanisms allow concepts such as private tuple spaces and tuple spaces within tuple spaces [10] to be exploited.

Our approach differs from the Yale precompiler in that we view a parallel job as a single program comprising a number of sections. Our system requires that all the components are compiled and analysed together. Once the tuples have been partitioned and common fields have been factored out, the discarded information can no longer be retrieved unless the whole job is recompiled. By contrast, the Yale effort [4] supports separate compilation, and provides a pre-linking stage which analyses and specializes the tuple space access procedures. Their goal is to optimize the accesses, but to carry enough run-time information so that the original unoptimized data can be reconstructed. This will allow new participants to join the computation dynamically.

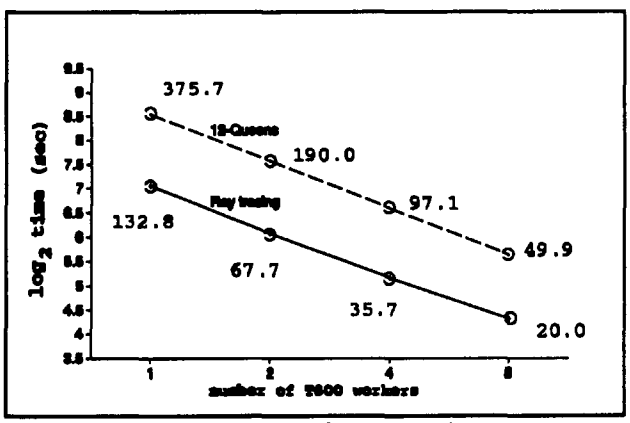

Figure 6 - Performance of ray tracing and queen placement algorithms running on the Rhoda system, as the number of worker processes is increased.

Some aspects of the system are still under development, notably the distributed *TS*-managers, and the system has only been tested with relatively small numbers of processing nodes (up to 16). The performance we have observed is encouraging. We have used the Rhoda implementation to support a number of parallel algorithms, including a state-space search and a 2-D FFT transformation. The Rhoda system is also being used as a platform for implementing a parallel version of a popular scientific and engineering matrix manipulation package[12], and as a means of parallelizing existing animation and graphics rendering applications. Figure 6 shows the almost linear improvement in speed obtained for an existing ray tracing application moved onto the Rhoda system, and for a queens placement algorithm, as the number of worker processes is increased.

We have been happy with the performance of our *TS* transport layer to date, and our experience also confirms the claims in the literature [2, 3, 8, 1] that it is easier to write parallel programs using the Linda model than it is with traditional tools. The high level abstract programming environment of the Linda operators has enabled us to think about parallelism in ways which were not always obvious when we were constrained by the concepts of semaphores and point-to-point messages.

From an implementor's point of view, Helios encourages a client/server programming model, and this has had a definite influence on our design. Without the presence of simple operating system mechanisms for creating and controlling tasks, topology independent message routing, and support for hierarchies of structures, we would not have envisaged the system as it is currently structured. This conviction is strengthened by our experience of designing a previous *TS* prototype on a network of PC's running DOS, an environment which constrained our thinking severely. Moreover, Helios's Unix-like development environment and the ANSI-C language support have isolated us from the awkwardness of the transputer's underlying RISC-like architecture, and this has improved our productivity. It is unlikely that we would have made similar progress using TDS and Occam, the customary systems programming tools used with transputers.

Notes

i. Linda is a trademark of Scientific Computing Associates, Inc., New Haven.
ii. Helios is a trademark of Perihelion Software Ltd., Somerset, England.
iii. UNIX is a trademark of AT&T.
iv. The Helios operating system includes a UNIX-compatible library, which is based on the POSIX standard [9].
v. This can be improved by program caching.

References

[1] S Ahuja, N Carriero, and D Gelernter, [1986], Domestic Parallelism - Linda and Friends, *Computer (USA)*, 19(8), 26-34.
[2] N Carriero, and D Gelernter, [1989], How to Write Parallel Programs: A Guide to the Perplexed, *ACM Comp. Surveys*, 21(3), 323-357.
[3] N Carriero, and D Gelernter, [1989], Linda in Context, *Comm. ACM*, 32(4), 444-458.
[4] N Carriero, and D Gelernter, [1990], Tuple Analyses and Partial Evaluation Strategies in the Linda Precompiler, in *Languages and Compilers for Parallel Computing*, Eds.

Gelernter, Nicolau and Padua, Pitman, 114-125.
[5] F K de-Heer-Menlah, [1990], *Analyzing Communication Flow and Process Placement in Linda Programs on Transputers*, Technical Document 90/4, Dept. Computer Science, Rhodes University.
[6] N Garnett, [1989], *Writing a Helios Server*, Technical Report No. 8, Perihelion Software Ltd.
[7] D Gelernter, [1985], Generative Communication in Linda, *ACM TOPLAS*, 7(1), 80-112.
[8] D Gelernter, [1988], Getting the Job Done, *Byte*, 13(12), 301-308.
[9] IEEE, [1988], *The Proposed POSIX Standard*, IEEE Std 1003.1.
[10] W M Leler, [1990], Linda meets Unix, *Computer*, 23(4), 43-54.
[11] Perihelion Software Ltd. [1989], *The Helios Operating System*, Prentice-Hall, London.
[12] E P Wentworth, and P G Clayton, [1991], *Matlab Meets Rhoda*, IEEE Symposium on Parallel Processing - Design and Implementation, CSIR June 1991.
[13] S E Zenith, [1990], *Linda Coordination Language; Subsystem Kernel Architecture (on Transputers)*, YALE Research Report DCS/RR-794.

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 a 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 double-space typing on one side only of A4 paper, and provide wide margins.
- 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 on separate sheets of A4 paper, and should be numbered and titled. Figures should be submitted as original line drawings, and not photocopies.
- 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.
- 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. References should thus take the following form:

  [1] E Ashcroft and Z Manna, [1972], The translation of 'GOTO' programs to 'WHILE' programs, *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255.

  [2] C Bohm and G Jacopini, [1966], Flow diagrams, Turing machines and languages with only two formation rules, *Comm. ACM*, **9**, 366-371.

  [3] S Ginsburg, [1966], *Mathematical theory of context free languages*, McGraw Hill, New York.

Manuscripts *accepted* for publication should comply with the above guidelines, and may be provided in one of the following formats:

- in a **typed form** (i.e. suitable for scanning);
- as an **ASCII file** on diskette; or
- as a **WordPerfect**, $\TeX$ or $\LaTeX$ or file; or
- in **camera-ready** format.

A page specification is available on request from the editor, for authors wishing to provide camera-ready copies. A styles file is available from the editor for Wordperfect, $\TeX$ or $\LaTeX$ documents.

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 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 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 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.

South African Computer Journal

Number 6, March 1992
ISSN 1015-7999

Suid-Afrikaanse Rekenaar-tydskrif

Nommer 6, Maart 1992
ISSN 1015-7999

Contents

GUEST CONTRIBUTION

Impressions of Computer Science Research In South Africa
E.G. Coffman, Jr. 1

RESEARCH ARTICLES

An Implementation of the Linda Tuple Space under the Helios Operating System
PC Clayton, EP Wentworth, GC Wells & FK de-Heer-Menlah 3

Modelling the Algebra of Weakest Preconditions
C Brink and I Rewitzky 11

The Design and Analysis of Distributed Virtual Memory Consistency Protocols in an Object Oriented OS
KJ McGregor and RH Cambell 21

An Object Oriented Framework for Optimistic Parallel Simulation on Shared-Memory Computers
P Machanik 27

Analysing Routing Strategies in Sporadic Networks
SW Melville 37

Using Statecharts to Design and Specify a Direct-Manipulation User Interface
L Van Zijl & D Mitton 44

Extending Local Recovery Techniques for Distributed Databases
HL Viktor & MH Rennhackkamp 59

Efficient Evaluation of Regular Path Programs
PT Wood 67

Integrating Similarity-Based and Explanation-Based Learning
GD Oosthuizen & C Avenant 72

Evaluating the Motivating Environment For IS Personnel in SA Compared to the USA. (Part I)
JD Cougar & DC Smith 79

TECHNICAL NOTE

An Implementation of the Parallel Conditional
U Jayasekera and NCK Philips 85

COMMUNICATIONS AND REPORTS

Book Review 87

The CSP Notation and its Application to Parallel Processing
PG Clayton 90