·O·U·G·

occam® user group · newsletter

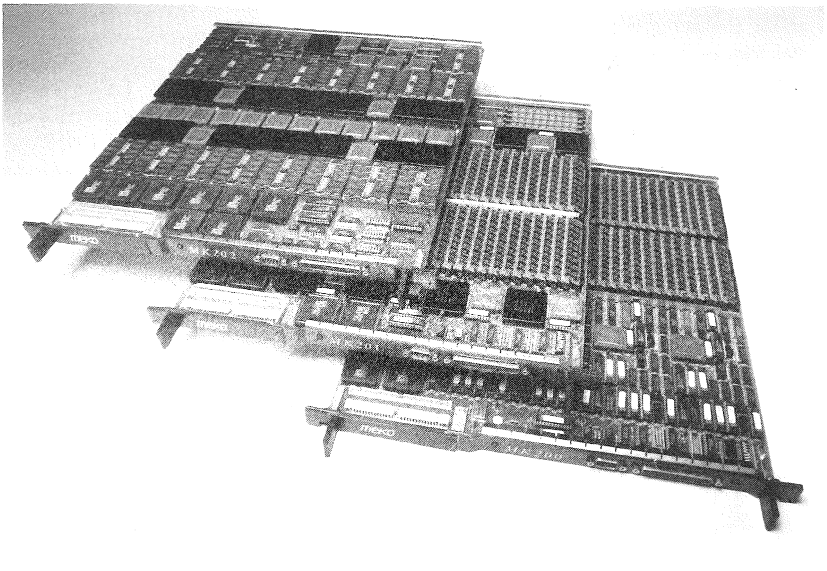
for all users of occam and the transputer

N? 12 January 1990

Contents

EDITORIAL 2
Contributions to the newsletter 3
Occam user group publications 3
North American transputer users group publications 4
FORTHCOMING 4
Twelfth occam user group technical meeting 4
North American transputer users group spring meeting 5
Third Japanese transputer/occam international conference 5
Thirteenth occam user group technical meeting 6
North American transputer users group fall meeting 6
Transputing 1991 7
REPORTS 9
Inaugural meeting of the occam user group: Latin America 9
Eleventh occam user group technical meeting 10
Why did the transputer cross The Pond? 14
Second seminar of the Swedish transputer user group 15

continued on back cover



Meiko In-Sun Computing Surface Hardware (see page 98)

occam is a trade mark of the INMOS Group of Companies

EDITORIAL

New groups seem to be springing up all around the world; this issue of the newsletter carries contact addresses for the first time for groups in Sweden and Latin America. We have contributions from as far afield as Brazil, the Basque country, and Bristol. It is all here: from exotic hardware (see for example pages 19, 82, 98 and many of the product announcements) to expressions of concern that software should be as unexciting as possible (see pages 22, 27). I would particularly like to draw your attention to the initiative to promote the occam language (see page 84); and to the CODE system from C-DAC (see pages 86 and 18), an impromptu presentation of which stole the show in the exhibition at the OUG technical meeting in Edinburgh.

For me, perhaps the most noticeable impact that SGS-Thomson have had on Inmos is in the way they are taking their marketing more seriously. Regulars were as astonished as I was, I am sure, to see an Inmos stand at the exhibition in Edinburgh *with the company's name on it*! I believe that Inmos were also one of the sponsors of the distressingly well-lubricated banquet at that meeting. (A bibulous colleague seated near your most temperate correspondent reports that the sight of full or almost-full bottles of wine being carried away from the tables is always distressing.)

Speaking at the panel session in the Edinburgh meeting, David May was able to confirm exactly one thing about the much-rumoured next-generation transputer. He was happy to dispel any doubt that its code name really is 'H1'. (Later in the bar sources close to Inmos told me a most unlikely tale to explain the name, something about the actor Nigel Hawthorne, and his character Sir Humphrey Appleby.) He was less forthcoming on specifications, although Inmos have made some public hints about the sort of specification at which they are aiming (see page 83). I note that the device is expected in early 1991; could it just be a coincidence that the international joint meeting of user groups (see page 7) is planned for the end of April 1991?

When I first drafted these notes, one of the things I was going to say was that we seem to be getting an increasing number of commercial contributions at occam user group technical meetings. This, I was going to say, is sure sign that there is a growing base of applications out there to report. The Edinburgh meeting, however, turned out to be rather short of papers from the industry – although there were a creditable number of commercial exhibitors. I remain convinced about the strength of the application base, and so am disappointed that we are not seeing enough of the work reported. Let me therefore take this opportunity to encourage readers in the industry to send papers to the OUG technical meetings, and those of its sister organisations of course. Needless to say, the same applies to contributions to this newsletter.

Please forgive me a personal note: I keep being mistaken for the Geraint Jones who works for the National Transputer Centre in Sheffield. I have even been congratulated on moving to a new job there. I trust, Geraint, that if you are ever mistaken for me you are at least as flattered by the mistake. Ladies and gentlemen! it is a common name, but we are two, as far as I know unrelated, and in no way responsible for each other's commissions or omissions. *gj, 11 December 1989*

CONTRIBUTIONS TO THE NEWSLETTER

Please contribute announcements, articles, letters about anything that looks as though it belongs in your newsletter. In particular we welcome letters, short articles or news about work being done with occam or transputers; calls for, discussion of, and reports on meetings of the group or related societies; ideas for new ways the group could help its members, or better ways of organizing what we do; details of material published elsewhere in books and journals; information about new products and courses.

Life would be easiest for the editor if you were able to submit material of longer contributions by electronic mail to `oug-news@uk.ac.oxford.prg`; or to send either unformatted ASCII files (on an IBM PC compatible floppy disk, or some other medium by negotiation) or clean camera-ready copy to the editor at the address below. Camera-ready copy should be arranged not to look out of place when its linear dimensions are reduced to about 70%, i.e. from A4 to A5.

Pictures are welcome as black-and-white prints, and will be subjected to the same reduction in size. We particularly need shorter and shorter pictures to fill the rapidly closing gap at the bottom of the front page!

Copy for the next edition must arrive by *25th May 1990.*

Geraint Jones
Programming Research Group
11 Keble Road
Oxford OX1 3QD
United Kingdom

Tel: +44 865 273851
Fax: +44 865 273839
oug-news@uk.ac.oxford.prg

OCCAM USER GROUP PUBLICATIONS

Occam user group technical meetings are held twice a year. The proceedings of all the recent meetings are published by International Organisations Services BV and are available directly from the publisher.

- ▷ OUG–7, 14–16 September 1987, Grenoble, *Parallel programming of transputer based machines*, ed. Traian Muntean (pp. 480, $110·00).
- ▷ OUG–8, 27–29 March 1988, Sheffield, *Developments using occam*, ed. Jon Kerridge (pp. 214, $50·00).
- ▷ OUG–9, 19–21 September 1988, Southampton, *Occam and the transputer – research and applications*, ed. Charlie Askew (pp. 176, $50·00).
- ▷ OUG–10, 3–5 April 1989, Enschede, *Applying transputer based parallel machines*, ed. André Bakkers (pp. 318, $65·00).
- ▷ OUG–11, 25–26 September 1989, Edinburgh, *Developing transputer applications*, ed. John Wexler (pp. 206, $55·00).

IOS
Van Diemenstraat 94
1013 CN, Amsterdam
The Netherlands

Fax: +31 20 22 60 55

or in the USA and Canada
IOS
PO Box 2848
Springfield VA, 22152-2848
United States of America
Fax: +1 703 250 47 05

or in Japan
IOS Japan Department
Highway Development Co. Ltd
1st Golden Building, 8-2-9 Ginza
104 Tokyo – Chuoku
Japan
Fax: +81 35 72 86 72

NORTH AMERICAN TRANSPUTER USERS GROUP PUBLICATIONS

The proceedings of the first two conferences of the North American Transputer Users Group can be obtained by sending a cheque, money order or purchase order to NATUG at the address below.

- ▷ The proceedings of NATUG 2 consist of thirty-one papers, 451 pages in all, for $40·00.
- ▷ There are still a few copies of the NATUG 1 proceedings available; the price is $30·00, for 209 pages including full colour illustrations.

When copies of these two proceedings are depleted, IOS will take over the reprinting and sales.

NATUG
c/o G. S. Stiles
Department of Electrical Engineering
Utah State University
Logan, Utah 84322-4120
United States of America

FORTHCOMING

TWELFTH OCCAM USER GROUP TECHNICAL MEETING

Monday to Wednesday, 2–4 April 1990, University of Exeter, England

The Occam User Group invites all those interested in the programming and application of transputer based architectures to attend its twelfth Technical Meeting at the University of Exeter. The conference includes lectures, an exhibition, a panel session with key speakers, and meetings of the Special Interest Groups. The emphasis of the conference is on the techniques and tools that allow transputers to be used effectively.

The meeting and exhibition will take place in Streatham Court and accommodation will be provided on campus in the Duryard Halls of Residence. The University is situated one mile from the centre of Exeter and its attractive site enjoys fine views over the Exe Estuary and Dartmoor Hills. A conference dinner will take place on Tuesday 3rd April.

All enquiries should be made to the Conference Organiser:

Dr Stephen J. Turner	Fax: +44 392-264067
Department of Computer Science	Tel: +44 392-264048
University of Exeter	Telex: 42894 EXUNIV G
Prince of Wales Road	steve@uk.ac.exeter.cs
Exeter EX4 4PT	
England	

NORTH AMERICAN TRANSPUTER USERS GROUP SPRING MEETING

26–27 April 1990, Santa Clara, California

The 1990 Spring meeting of the North American transputer users group will be held in Santa Clara, California, on 26th and 27th April. Contributions are being solicited in the areas of hardware, software, and applications – particularly embedded and real-time systems. Presentations will be allotted twenty minutes each, with ample additional time for questions; this format will allow for approximately twenty-four talks. Additional papers may be presented at poster sessions if there is sufficient interest. All accepted papers will appear in a published proceedings.

Abstracts of 1000–1500 words must be received by 8th January 1990. Notices of acceptance will be sent out by 22th January, and final drafts of accepted papers will be due by 19th February. Authors should indicate on the abstract whether they would be willing to present the paper in a poster session.

Abstracts – electronic mail submissions are preferred – should be submitted to:

Professor Alan Wagner, NATUG 3 Program Chair	wagner@cs.ubc.cdn
Department of Computer Science	Tel: +1 604 228-6450
University of British Columbia	
Vancouver, BC	
Canada V6T 1W5	

THIRD TRANSPUTER/OCCAM INTERNATIONAL CONFERENCE

organised by the occam user group Japan
17–18 May 1990, Tokyo, Japan

OUG Japan is pleased to announce the third transputer/occam international conference to be held on 17–18 May 1990 in Tokyo. This is a place to find wide varieties of transputer based parallel processing systems through original refereed papers, invited talks, tutorials and commercial exhibits. The official language of the conference is English. Simultaneous translation from English to Japanese is provided whenever necessary. The registration fee is ¥20 000 and this includes:

▷ admission to all technical sessions, tutorials and exhibits,
▷ admission to all lunches,
▷ admission to the final party.

The conference organizing committee consists of:

Prof. Dr Tosiyasu L. Kunii (Chairman), University of Tokyo;
Prof. Dr Tadao Nakamura, Tohoku University;
Prof. Dr Eiichi Miyamoto, Hokkaido University;
Prof. Dr Shinji Tomita, Kyushu University;
Associate Prof. Dr Mitsuru Ishizuka, University of Tokyo;
Research Associate Dr Shigeki Sugano, Waseda University.

Call for papers

Full papers in English are solicited and should contain previously unpublished original high-quality results. The submitted paper should be typed, double spaced, about 10–20 pages in length with an abstract of 100–200 words and a maximum of ten keywords. All papers will be peer reviewed and will be published for distribution at the conference and also for later dissemination. Papers must be submitted by 1st January 1990. Acceptance or rejection of the papers will be notified by 1st February 1990. Final camera-ready copy must be provided by 1st April 1990.

For more information, contact:

Mr Kazuto Matsui (Secretary, OUG Japan) Tel: +81 3 280-4125
Technical Marketing, Inmos Division Fax: +81 3 280-4131
SGS-Thomson Microelectronics K.K.
4F Nisseki-Takanawa Building 2-18-10
Takanawa Minato-ku Tokyo 108
Japan

THIRTEENTH OCCAM USER GROUP TECHNICAL MEETING

18–20 September 1990, York, England

Advance notice is given of the thirteenth technical meeting of the occam user group, which will take place at the University of York from 18th to 20th September 1990. The provisional deadline for extended abstracts of papers is 4th May 1990. For further information about this meeting contact:

Dr Hussein Zedan Tel: +44 904 432744
Department of Computer Science Fax: +44 904 432767
University of York zedan@uk.ac.york.minster
Heslington
YorkYO1 5DD
England

NORTH AMERICAN TRANSPUTER USERS GROUP FALL MEETING

12–13 October 1990, Ithaca, New York

Advance notice is given of the fourth meeting of the North American transputer users group which is to be held at Cornell University, New York.

TRANSPUTING 1991

First world conference of national occam and transputer user groups
22-26 April 1991, Santa Clara, California

Advance announcement

The first world conference of all occam and/or transputer user groups is planned:

TRANSPUTING 1991	
Place:	Santa Clara, California
Dates:	week commencing 22nd April 1991

The goals of this conference are:
- ▷ to present 'state-of-the-art' research on all aspects of parallel computing based upon communicating process architectures;
- ▷ to demonstrate 'state-of-the-art' products and applications from as wide a range of fields as possible;
- ▷ to progress the establishment of international software and hardware standards for parallel computing systems;
- ▷ to provide a forum for the free exchange of ideas, criticism and information from a world audience gathered from Industry, Commerce and Academia;
- ▷ to establish and encourage an understanding of the new software and hardware technologies enabled by the transputer;
- ▷ to promote an awareness of how these technologies may be applied and what their advantages are.

The conference themes will *include*: education and training issues, formal methods and security, performance and scalability, porting existing systems, parallelisation paradigms, tools, programming languages, support environments, standards and applications.

Applications *include*: embedded real-time control systems, workstations, supercomputing, consumer products, artificial intelligence, databases, modelling, design, data gathering and the testing of scientific or mathematical theories.

The conference programme will contain invited papers from established international authorities in these fields together with submitted papers. All papers will be fully refereed by an international programme committee. Only papers of high excellence will be accepted. The proceedings of this conference will be published internationally and copies will be given to delegates when they register at the start of the meeting.

Programme committee

The members of the programme committee will be invited experts from Industry and Academia together with existing committee members from the joint organising user-groups based in Australia, France, Germany, India, Japan, Latin America, New Zealand, North America, Sweden, and the United Kingdom.

The aim is to spread the organising load around the world, to ensure that all points of view and expertise are properly represented and to obtain the highest standards of excellence. We shall be seeking endorsements from professional institutions on all five continents, as well as sponsorship from commercial and governmental organisations.

Provisional conference structure

```
PAR
  ... exhibition / demonstrations (Monday .. Friday)
  SEQ
    ... seminars / workshop (Monday)
    ... conference papers (Tuesday .. Thursday)
    ... seminars / workshop (Friday)
```

Provisional timetable

February 1990	Official publicity and first call for papers, exhibitors and delegates.
1st August 1990	Deadline for submitted papers.
1st October 1990	Notification of acceptance to authors; launch final call for exhibitors and delegates.
6th January 1991	Receive revised camera-ready copy of papers.
22nd April 1991	**TRANSPUTING 1991.**

Impact on existing OUG/TUG meetings

The OUG and NATUG will not organise meetings at Easter 1991. *Transputing 1991* will subsume the OUG and NATUG meetings normally scheduled for this time.

Competition

Find a suitable name for the parent organisation for all our existing user groups. You have to do better than

- ▷ WOTUG (the World Occam and Transputer Groups);
- ▷ ATM (the Association for Transputing Machinery).

Also, find a better name for *Transputing 1991.* Prizes are unspecified for now, but will be up to the usual OUG standards. Entries should be sent to the editor of this newsletter.

Further information

Companies and other organisation wishing to be associated with *Transputing 1991* should contact the chairman or secretary of their local user group. We also welcome offers of help from individuals with special skills.

Further details will be announced in line with the provisional timetable above. In case of difficulties, please contact Peter Welch (OUG Chairman) or Dyke Stiles (NATUG Chairman).

Peter Welch

REPORTS

INAUGURAL MEETING OF THE OCCAM USER GROUP: LATIN AMERICA

14–16 September 1989, Florianópolis, Santa Catarina, Brazil

Rafael D. Lins, Universidade Federal de Pernambuco

Invited speakers

Michel Gien, Chorus Systemes, France. Michel presented an inspiring overview of distributed operating systems, including a description of the Chorus system applied to various hardware configurations.

S. Vedat Demiralp, University of Kent, Canterbury, UK. Vedat conducted a two-day tutorial on occam and transputer programming. Demonstrations of occam and C running on a transputer system were included.

Denis Nicole, University of Southampton, UK. Denis described recent and future MIMD parallel computing architectures including the Esprit P1085 Supernode, the Esprit Genesis project and the just-initiated Puma project to build a 'universal message-passing architecture' in collaboration with Inmos and others,

Local speakers

Claudio Amorim, COPPE/UFRj, Rio de Janeiro, RJ. Work at Rio on using occam for numerical analysis was described, including an implementation of a translator from Actus to occam.

Clésio Tozzi, Unicamp, Campinas – SP. The use of occam and transputers for image recognition and in industrial applications was outlined.

Antonio Carols Ruggiero, Universidade São Carlos, São Carlos – SP. The proposed use of transputers to implement a dataflow machine and an NMR tomographer was described. The solution of the molecular dynamics problem using transputers was also addressed.

N. Furuya, Unicamp, Campinas – SP . Experience in using parallel C on transputers for the analysis of numerical algorithms was described.

Rafael Lins, Universidade Federal de Pernambuco, Recife. Work at Recife on processors for interval arithmetic, on NMR tomography, and on functional language implementations was described.

Discussions

All present expressed enthusiasm for the formation of an occam user group in Latin America. Attendees at the meeting included academics from Argentina, Brazil, and Chile. Hope was expressed that the OUG:LA would help facilitate access to transputer hardware and software in South America.

The next meeting is planned for May 1990 in Petropolis, Rio de Janeiro and it is hoped that members of the group will be able to attend the proposed 'Transputing 1991' in the United States.

Acknowledgements

The meeting was sponsored by Institutio de Engenharia de Software da Brasil, CNPq, UFPE, Unicamp, UFCS, Projeto Ethos and the British Council, for whose support the participants are deeply grateful. Particular thanks are owed to Prof. E. Tadao Takahashi of Projeto Ethos and Antonio Carols Lariani from IBM Brasil.

Contact

For any further information please contact Rafael Lins, who has agreed to act as chairman of OUG:LA, at:

Rafael D. Lins
Av. Dr José Rufino 656
Estância
50.781 – Recife – PE
Brazil

Tel: +55 81 251 0713
Fax: +55 81 326 4880

ELEVENTH OCCAM USER GROUP TECHNICAL MEETING

Monday and Tuesday 25–26 September 1989
University of Edinburgh, Scotland
Nigel J. Anderson, Univeristy of Edinburgh

This meeting, which was held in Edinburgh on the 25th and 26th of September this year, went under the title of *Developing transputer applications.* The conference was held in the Appleton Tower, part of the campus of the University of Edinburgh, and immediately preceded the Edinburgh Concurrent Supercomputer Project's annual seminar. The site allowed for exhibitions by a number of commercial organisations with interests in parallel processing and the transputer.

The papers

There was a very full program of papers, with eighteen scheduled to be presented in the two days, as well as the now traditional panel session, and presentations by Inmos. The contributors were almost entirely from academic institutions, with the largest number coming from those within the United Kingdom. However, there were also contributions from European institutions, and the United States. One paper could not be presented, due to travel difficulties affecting its author, so the expected contribution from Asia did not occur. The paper – on an Intelligent character reader – is in the proceedings. These were published by IOS, as before, under the title *Developing Transputer Applications* and edited by John Wexler (see page 3).

The spread of interests covered in the papers was almost as wide as the geographical distribution of the authors. Within the general frame of reference of the meeting, there were three main strands. These can be categorised as operating systems, development tools and new paradigms for particular types of problems.

A brief overview of the papers presented is given below, along with the names of their presenters.

TROS: A REAL TIME KERNEL FOR A FAULT TOLERANT MULTI-PROCESSOR COMPUTER BASED ON ARGUMENT FLOW – ERIC VERHULST, R. LAUWEREINS, R. CUYVERS, J. PEPERSTRAETE, INTELLIGENT SYSTEMS INTERNATIONAL, BELGIUM. This paper describes the design and implementation of a fault tolerant system to provide a load balancing kernel written in occam and t-code, for a multi-user multi-transputer system. The system is designed to produce an environment in which code from more than one user's program may be resident on a single compute element. It describes the topology independence achieved through the use of argument flow techniques, and the fault tolerance achieved against both hardware and software failures.

DYNAMICITY THROUGH OCCAM AND TDS – D. MILLOT, J. VAUTHERIN, UNIVERSITÉ PARIS-SUD, FRANCE. This paper describes an attempt at dynamic process creation in occam, for the production of pipelines of indefinite length, by the use of self modifying software. The situation on a single transputer, and that for known topologies is considered. An attempt at extension to unknown topologies is also considered.

A CASE TOOL FOR DESIGNING DEADLOCK FREE OCCAM PROGRAMS – W. D. CROWE, R. HASSON, P. E. D. STRAIN-CLARK, THE OPEN UNIVERSITY. This paper describes a graphically based tool for designing programs (initially in occam, but potentially for any language which obeys the CSP model). It describes the interface selected, which is a hierarchical one, based on the concept of reusable components, and the tools available within the system for investigating the performance of the program being prototyped.

A DEADLOCK DETECTION TOOL FOR OCCAM – IR WOUTER JOOSEN, PROF. DR IR PIERRE VERBAETEN, UNIVERSITY OF LEUVEN, BELGIUM. This paper discusses the design and implementation of a *static* tool to perform deadlock detection in occam programs. It describes the system, implemented under Unix, produced using the yacc and lex compiler and parser generation tools. It processes the program source, and produces action sequences, which are program execution paths. The important events in such traces are PARbegin, PARend and channel communications. Deadlocks and infinite loops are detected by exhaustive searching of the action sequences, with programmer interaction to provide such things as values for loop counters. The tool is described by the authors as an adjunct to run-time debugging tools, and is seen as having its greatest utility in the design and early implementation stages.

SOLVING PARTIAL DIFFERENTIAL EQUATIONS VIA CELLULAR AUTOMATA: A BINARY AND STATISTICAL APPROACH – F. DESBOIS, A. COSNUAU, Y. MORCHOISNE, ONERA CENTRE DE CALCUL, FRANCE. Most systems for solving

PDEs use physical models, for example, the lattice gas model. The approach described here uses a numerical method, which uses Booleans, and probabilistic methods.

TOWARDS A SOFTWARE ARCHITECTURE FOR SOLID MODELLING SYSTEMS ON PROCESSOR NETWORKS – D. P. MALLON, N. S. HOLLIMAN, P. M. DEW, J. R. DAVY AND A. DE PENNINGTON, UNIVERSITY OF LEEDS. This paper discusses the various steps taken in the production of a machine independent system for solid modeling. It describes a three level model for this, based on the following: a machine architecture level, to allow compatibility between different real machines; a task specification level, which is both application and hardware independent; and a geometric application level, which is an application specific way to express geometric algorithms. The use of 'message to destination' schemes on a number of architectures is discussed, as a method of providing the machine architecture level. As an example of the higher levels, a constructive solid geometry modelling system is described.

AN IRREGULAR DISTRIBUTED SIMULATION PROBLEM WITH A DYNAMIC LOGICAL PROCESS STRUCTURE – MING Q. XU, STEPHEN J. TURNER, NIE PIN, UNIVERSITY OF EXETER. This paper discusses the implementation of a host-parasite interaction simulation, with hosts and parasites represented as logical processes. The system uses a modification of the 'time warp' approach, with the addition of logical process creation and deletion, and the system being action driven. That is, it is based on the interactions between logical processes.

A GENERALLY CONFIGURABLE MULTI-GRID IMPLEMENTATION FOR TRANSPUTER NETWORKS – OSAMA EL-GIAR AND TIM HOPKINS, UNIVERSITY OF KENT. Multi-grid is an iterative technique for solving equations, which gains over such approaches as Gaus-Seidel and successive over-relaxation, by using a sequence of grids in their solution. The approach taken computationally is one of dividing the grids into strips for processing. Results are presented for the application with a number of different configurations of the grid step and number of processors, and are compared with results for a similar system on the Intel hypercube.

SELF-ADJUSTING MAPPING: A HEURISTIC MAPPING ALGORITHM FOR MAPPING PARALLEL PROGRAMS ONTO TRANSPUTER NETWORKS – HONG SHEN, ÅBO AKADEMI UNIVERSITY, FINLAND. This paper describes the development of an algorithm for mapping arbitrary process graphs onto arbitrary processor graphs, based on a division of the task into the following units. A grouping module, which attempts to produce a sub-optimal clustering of processes into tasks that can be placed on the processor graph by the placement module. The final module is routing, which produces the connection between the real links and placed tasks and it is expected that all three can be used to work together to find a nearly optimal solution, in a cyclic manner.

INVESTIGATION OF COMMUNICATIONS PATTERNS IN OCCAM PROGRAMS – ROSEMARY CANDLIN, QUIANGYI LUO, NEIL SKILLING, UNIVERSITY OF EDINBURGH. This paper describes two approaches to examining communications in occam programs. The first, direct measurement, was applied to a real application program on a Meiko computing surface, and the second utilizes discrete event simulation of the same program. Both systems are described, with some emphasis on the differences between the two approaches, and a comparison of the results obtained by each, giving a good indication that the simulation was producing results close to the performance of the real system.

SYSTEM CONFIGURATION FOR VERY LARGE DATABASE PROBLEMS – ALAN G. CHALMERS, DEREK J. PADDON, UNIVERSITY OF BRISTOL. This paper discussed the implementation of an extension of the process farm approach to load balancing, with a system controller and 2^n worker processors, each of which was composed of an application process and a number of task and data control processes, to deal with the distributed nature of the data involved. The application of this approach to radiosity calculations was discussed, as was the choice of a topology on which to mount the system.

A COMPARISON OF PARALLEL IMPLEMENTATIONS OF FLUX CORRECTED TRANSPORT CODES – JING-MING JONG, UNIVERSITY OF WASHINGTON, USA AND G. S. STILES, UTAH STATE UNIVERSITY, USA. This paper describes the authors' experience of FCT systems for computational fluid dynamics problems on a number of different architectures, and compares the relative speeds of the algorithm on them. Whilst for pure speed, networks of transputers are nowhere near as fast as true vector supercomputers, when the cost/performance ratios are taken into account networks of transputers score very highly.

SIMULATING NEURAL NETWORKS IN DISTRIBUTED ENVIRONMENTS – JUKKA VANHALLA AND KIMMO KASKI, TAMPERE UNIVERSITY OF TECHNOLOGY, FINLAND. Two types of neural network simulation are discussed, these being the sparse distributed memory model and the Hopfield model. There is interest in implementing neural networks on parallel systems, due to their inherent parallelism. The authors describe their results for the two models, and compare the two as to their suitability for this type of parallelisation.

ATTRIBUTE EVALUATION ON A NETWORK OF TRANSPUTERS – MATTHIJS F. KUIPER, ATZE DIJKSTRA, UNIVERSITY OF UTRECHT, NETHERLANDS. This paper discusses the application of attribute grammars to the problems involved in producing parallel compilers. It describes a static approach which allows program decomposition, and therefore the utilisation of as many of the available processors as possible. This is compared to the approach taken in pipelined parallel compilers, where the number of processors to be used is fixed.

AN OBJECT ORIENTATED STYLE FOR THE MEIKO – MATTHEW CHALMERS, UNIVERSITY OF EAST ANGLIA. This paper describes a novel approach to programming on a Meiko Computing Surface, based on SmallTalk. Objects are composed of their normal activity, and a link to their superclass, allowing inheritance of properties. All of this takes place within the context of a message passing scheme, which passes messages to objects. However, the objects have control over the order in which they accept messages. The original application for which the system, called *Chancer*, was designed was a ray tracer.

C-NET: A C++ BASED LANGUAGE FOR DISTRIBUTED AND REAL-TIME PROGRAMMING – JEAN-MARC ADAMO, ÉCOLE NORMALE SUPÉRIEURE DE LYON, FRANCE. This paper discusses the philosophy and constructs of a C++ based language which contains parallel constructs and exception handling as well as object orientation. It gives the syntax, and examples of the use of the language, which bases its parallelism on that available in occam.

REAL-TIME TRANSPUTER MODELS OF LOW LEVEL PRIMATE VISION – ANDREW B. SMITH AND PETER H. WELCH, THE UNIVERSITY OF KENT. Human vision is thought to consist of an initial parallel stage in which feature data is gathered, followed by a sequential recognition stage. This paper discusses the implementation of a simulation of the parallel stage on a network of transputers. The authors note that their initial, highly parallel scheme is less efficient than one developed later with a much higher degree of sequential activity.

Social aspects of the meeting

Despite the shortness of the meeting, it was apparent how quickly those attending became a cohesive unit, and how barriers of language and background were quickly overcome. Within the commercials slot, a book entitled *The Transputer Community* was advertised, published by the Edinburgh University Press, being a study of the transputer phenomenon, and covering the development of this community, showing the degree of interest raised by such developments, even among social scientists.

Much praise is due to the meeting organiser, John Wexler, who coped wonderfully with the difficulties inherent in setting up such an event, and edited the proceedings, as well as carrying out his normal duties within the Supercomputer Project. He was also charged with the organisation of the Supercomputer seminar. Despite all this he managed to keep the whole thing running smoothly, and retained his good humour. His crowning moment in this field must have been the conference dinner however, which was very well attended, and universally regarded as being of the very highest quality. It must be said that the organisers of the next meeting will have their work cut out to match it.

WHY DID THE TRANSPUTER CROSS THE POND?

As you might expect, the founder's second consecutive absence did not prevent the holding of the third biannual Roger Shepherd memorial joke contest at the occam user group meeting in Edinburgh. Among the repeatable contributions were

Q: What do you call a ring of T400 transputers?
A: Unsociable.

Two transputers pass five others who are playing brass instruments in a confined space; says one to the other: 'What's up?'; comes the reply: 'Nothing really – just a bandwidth problem.'

Q: What would you get if you privatized INMOS?
A: Lots of H_1Owners.

(I should explain for readers lucky enough to live outside the target area for the advertisements that the meeting coincided with a pre-privatization publicity campaign for the English and Welsh water boards, which urged everyone to become a H_2Owner.)

> There were a surgeon, and architect and a parallel programmer, who were discussing whose was the oldest profession.
> 'Mine must be', said the surgeon, 'as God removed a rib from Adam to create Eve – quite a surgical feat.'
> 'No contest,' said the architect, 'out of chaos God created the ordered garden of Eden – a major architectural project.'
> 'That's nothing,' said the parallel programmer, 'who do you think created the chaos.'

Q: Why shouldn't a Transputer run C_NET?
A: Because its pins get caught in the mesh.

Q: What is a transputer user's favourite Christmas carol?
A: 'occam all ye faithful, joyful and triumphant.'

Q: What do occam processes and condoms have in common?
A: Both are safe 99% of the time, and neither can be reused.

Someone claimed that it was Jean-Paul McSartre who said after attending the Edinburgh occam user group meeting, 'Och Aye! therefore occam'. [I think it was someone more like René Cathcart.]

Perhaps the shortest contribution was a piece of paper bearing the single word *occam* written in a hand remarkably like Iann Barron's. I suspect that if a joke could only have been found for it, *tightly coupled embedded systems* – much discussed in the panel session – would have made a successful punch-line.

On the basis of audience response, the group's regular clap-o-meter and chairman chose Colin Willcock's – the occam processes that are safe 99% of the time – to receive McTavish's (ostensibly) nuclear-powered self-heating haggis that had appeared in a talk earlier in the day. Alan Chalmers walked away with a larger haggis which was apparently the second prize – for the architect of chaos. *gj*

SECOND SEMINAR OF THE SWEDISH TRANSPUTER USER GROUP

21 November 1989, The Royal Institute of Technology, Stockholm

Martin Törngren, The Royal Institute of Technology

The Swedish market has been a tough one for the Inmos transputer. When the transputer was introduced it represented new ideas and new technologies. It usually takes some time for a new microprocessor to get accepted and for companies to change to the new microprocessor. Another reason for scepticism and delay is the fact that Inmos was not able to provide sufficient development tools in 1985. Inmos has always emphasized the importance of occam. However, even though many people enjoy occam, it is a new language. There is even now no occam compiler for other computers!

Today, four years after the introduction, things are looking much better. There are now a lot of different development tools available. New transputer chips and modules have appeared. SGS-Thomson gives stability to the transputer product and has introduced dramatic price reductions.

In Sweden many companies and universities have shown interest in the transputer and evaluations have been made. Recently the first transputer based product, an image processing computer, was released in Sweden. In addition to Inmos representatives, sales representatives from other manufacturers like Microway have appeared.

The Swedish Transputer User Group (STUG) was formed in May 1989. STUG now has about sixty members in Sweden. STUG arranges seminars and publishes a newsletter. STUG operates with a very limited budget and has the support of Inmos representatives in Sweden, which handles some of STUG's administrative details. The first STUG seminar was held at The Royal Institute of Technology in Stockholm in May. At this seminar an executive committee was elected to look after the group and it was agreed to publish a newsletter.

The seminar

The seminar was hosted by the Department of Machine Elements at the Royal Institute of Technology in Stockholm. Around fifty people from Universities, Technical Universities and companies were represented at the seminar.

The keynote speaker was Stephen Maudsley, application engineer at Inmos, who talked about new transputer products like the new low cost T400. He reported on plans for the future development of the transputer. He gave information on the H1 project, which aims at a transputer with the following characteristics:

- ▹ 10 times faster than todays transputers
- ▹ code compatible with current transputers
- ▹ caches and support for virtual channels

Stephen also mentioned new modules for ethernet with TCP/IP support, SCSI and the VME bus. Other information included the cooperation between Inmos and the ASIC division of SGS-Thomson, the new breakpoint debuggers (initially C and occam) and future optimizing compilers.

After Stephen's talk there was a debate which centered around the following questions:

- ▹ Will there be any transputers with I/O on the chip (like the M212)?
  Answer: Probably no, instead ASICs with link units will be available.
- ▹ Is the transputer used for embedded systems applications?
  Answer: It is, but you would not necessarily hear about it.
- ▹ How can Inmos, a manufacturer of advanced parallel processing components, produce such low quality man-machine interfaces for their development tools.
  Answer: ??
- ▹ In what way should occam develop? Should it be kept as a secure language or should for instance records and/or recursion be included?
  One answer: occam is a very nice language for 'smaller embedded applications' with a static software structure.
- ▹ Do you want an operating system or a library? Should the operating system be easy to modify by the user?

An obvious drawback with the current transputer scheduler is the few priority levels. This limits the use of the transputer in hard real-time applications.

Other speakers at the seminar

Lars Estreen from the Department of Electronic Motor Drives at KTH had visited the *International conference on applications of transputers* in Liverpool. His overall impression was that the transputer was mainly used for image processing and super computer applications. He also emphasized the lack of an overall transputer standard for operating systems and communications.

Hans Johansson also from the Department of Electronic Motor Drives had attended an occam course arranged by Inmos in Bristol. He briefly reviewed the course and recommended it for people not experienced with the TDS.

Christer Juren from the Swedish Institute of Space Physics had visited the OUG meeting in Edinburgh and the subsequent supercomputer conference. He made a very interesting presentation of the Edinburgh concurrent supercomputer and also detailed the OUG proceedings.

Mariadata representatives of Alsys in Sweden presented the newly validated Ada compiler which however was not yet ready for demonstration.

Peter Ygberg from Bofors Aerotronics talked about his experiences of the Helios operating system which he enjoyed using.

My own contribution presented aspects on using the transputer in real-time control systems for applications in Mechatronics, particularly real-time properties, features which can cause non-deterministic behaviour and how to interface the transputer to sensors and actuators.

Mats Hanson from the Department of Machine Elements at KTH talked about new microprocessors for embedded systems. He particularly described an ongoing Master of Science project for fifteen students with guidance from researchers which aims at building a transputer based robot control system. Mr Hanson also proposed a joint project between SGS-Thomson in Sweden and the department.

The seminar was rounded up by refreshments and a few informal demonstrations. The CVR group showed their image processing computer and Datacraft, sales representatives of among others Microway, displayed quadputer transputer boards. Unfortunately neither Inmos nor Atari with their workstation could make a demonstration.

Future

The seminar was a positive one. There's definitely a growing interest of the transputer in Sweden.

STUG intends to publish newsletters shortly after seminars. The deadline for this newsletter is 10th December. The newsletter is written in Swedish but does accept papers in English.

The next seminar has not been scheduled but will probably be in the spring of 1990.

SPECIAL INTEREST GROUPS

EDUCATION AND TRAINING

Roger M. A. Peel, University of Surrey

The Education and Training SIG meeting during the twelfth OUG technical meeting in Edinburgh was attended mainly by members of various educational institutions, as well as by representatives of Inmos and several major industrial users of transputers.

Much of the meeting was spent reviewing hardware and software products suitable for use in education. One new product which generated particular interest was the CODE system from C-DAC (Pune, India), which closely emulates the Inmos Transputer Development System, but runs on a personal computer with no subsidiary transputer card (see page 86). Comprising an editor, occam 2 translator and an interpreted run-time environment, one of its major features is an interactive debugger which permits the state of variables and processes to be inspected during program execution. At a one-off price of $400, and with site licences available for $4000, this software is likely to be of interest to many members of the educational community.

Other products mentioned were the interfaces, TRAM motherboard and server software for the Acorn Archimedes from Gnome Computers, as well as reminders that the Inmos TDS1 product is freely available for personal computers from Jon Kerridge, and the VAX/Unix version from Peter Welch. Bob Stallard's occam 2 compiler (for Suns, etc.) is available from Loughborough University. Michael Poole (Inmos) announced that their new occam 2 compiler (written in C) could be made available in source form to organisations prepared to commit to porting it to novel non-transputer environments. Support for this activity would however be limited.

The remainder of the meeting was spent discussing how educational support might increase the uptake of transputers by industry. There was support for the suggestion that pre-packaged courses, complete with documentation, discs containing all the example programs and tutorial notes, should be made available to remote educational institutions and to industrial concerns not prepared to send their staff for direct training.

There was also considerable sympathy with the view that many technologists in the field were aware of the benefits of using transputers and occam, but that they were unable to convince their management that parallel program design was not too difficult. There also seemed to be reasons why employers did not want to see design methodologies in external courses. In addition, some engineers often prefer to avoid the concurrent solution because more complicated conventional languages and processors are a greater technical challenge!

Finally, there was muted interest in an *occam educator's workshop*, although there did not appear to be any convenient dates until 1990.

Roger M. A. Peel
Department of Electronic and Electrical Engineering
University of Surrey
Surrey GU2 5XH
United Kingdom

Tel: +44 483 509284
Fax: +44 483 34139
Telex : 859331
roger@uk.ac.surrey.ee

HARDWARE

Denis Nicole, University of Southampton

This is a report of the meeting of the Hardware SIG held during the eleventh technical meeting of the OUG at Edinburgh, September 1989. As usual, the group engaged in a wide-ranging discussion about most things transputer.

Miniaturized transputer arrays

Hugh Webber of the Royal Signals and Radar Establishment described some work, mainly by Kevin Palmer, on the construction of miniature 'Supernode' switched transputer arrays. He has been able to build transputer modules of two types. One is a ceramic module sized two inches by one and a quarter inches which carries a transputer and 256 kbytes of DRAM. The other is a silicon hybrid with the same footprint as a transputer which carries both the transputer and, underneath it, 64 kbytes of DRAM all in a twenty-eight pin package.

They have been able to construct a whole sixteen transputer Supernode including camera, framestore, B007 graphics and LCD display in a $6'' \times 4'' \times 2''$ cuboid. The system consumes 20 watts and can be powered for four hours by a standard military battery of the same size.

Dynamic switching

There was some discussion of the use of dynamic reconfiguration of transputer arrays for message passing. The Linda machine from Cogent in the United States implements part of its communications in this way. Considerable care was required to ensure that no random traffic on the links was produced by the reconfiguration of the C004 switches. Some work on a Supernode at RSRE suggested that dynamic switching could also be effective on that architecture.

Static column addressing

A new UK company, Division, has developed a graphics TRAM which uses static column addressing of dynamic memory. This is implemented in a small set of conventional PALs. Some discussion of the merits of this approach ensued, with J. Kidd suggesting that, while it may be worthwhile for display systems, it is not useful for most other transputer systems.

Standardisation

D. Nicole had been approached by M. Jane of the UK transputer initiative with a suggestion that interested users and vendors should attempt to establish hardware standards for transputer systems. An example was presented of a language vendor having to test his compiler on B004, B008 and B001 boards separately. These standards can take several forms: they can enshrine existing practice in such areas as control port addressing, link buffering and TRAM I/O mechanical organisation; they can develop new standards, for example, for miniature TRAMs; they can attempt to guide new implementations on current transputers towards facilities that will be supported directly in new generation transputers, in such matters as through-routed message-passing protocols; finally, they can attempt to provide a common standard software interface for disparate hardware, such as Supernode/B008/Meiko/Parsytec link switching. Comments and offers of help and guidance to D. Nicole, please.

Communications regarding the hardware SIG should be addressed to:

Denis A. Nicole
Department of Electronics
and Computer Science
The University
Highfield
Southampton SO9 5NH
United Kingdom

Tel: +44 703 787167
Fax: +44 703 592865
dan@uk.ac.soton.ecs

FORMAL METHODS

R. P. Stallard, Racal-Milgo Ltd

A meeting was held at the Edinburgh Technical Meeting of the OUG. There was not much news to report.

I spent much of the time outlining ideas not on occam itself but on what might be the form of the language on top of occam. It is my view that there must be a convergence between use of formal techniques and software engineering. What is needed to promote the occam model is a high level programming environment that maps neatly onto occam. Such a system could be of 'object-orientated' type, but even better would be 'process-orientated'. Such systems have the benefit of clearly defining an interface to a module and also managing the inevitable changes to a software system. All this is possible directly in occam but it is not made compulsory. Why should we still use systems that make straightforward changes so difficult? I hope to produce an article on this topic for the next newsletter.

In the brief time that was left, discussion on adding more features to occam started, I propounded the view that all further developments should be banned (or else we will need to apply Occam's razor to occam). If users want more features they should use a higher level language. The panacea of the perfect programming language has been sought after for almost as long as the Elixir of Eternal Life, you can not make a language suitable for every application. My personal plea remains: please leave occam as an efficient medium level language with fast execution.

Deadlock got its usually mention, it was felt that tools to combat it run the danger of oversimplifying the problem, but some degree of modularization (hiding internal channel intercommunication) obviously helps. It is encouraging that two papers at Edinburgh described tools to detect deadlock. I hope they will soon be available for all of us to use.

R. P. Stallard
Racal-Milgo Ltd
Station Road
Hook, Hants
United Kingdom

GRAPHICAL PROGRAM DEVELOPMENT TOOLS

a new special interest group

Mike Roberts, City University, London

At the last occam user group meeting I proposed the formation of a 'Graphical program development tools' special interest group. As the suggestion did not meet with quite the hilarity I anticipated (six people were genuinely interested) I decided to test the water with the following short piece. Anyone interested in the formation of the SIG may contact me at the address below.

What is a graphical program development tool?

Graphical program development tools do, as their name implies, use graphics in the program development process. Many experimental tools have been produced for use in all stages of the sequential software life cycle ranging from high level project management systems to low level tools using graphics in the programming process.

As yet however, few have been produced for parallel systems though many feel that graphical tools may help in that 'Holy grail' of parallel processing – the export of parallel systems and languages into the so called real world.

They fall naturally into two main areas – program visualisation tools and visual programming tools. Program visualisation is the use of computer graphics to enhance program presentation and facilitate the visualisation, understanding and effective use of programs by humans. Visual programming on the other hand is a collection of related techniques through which algorithms are expressed using various graphical representations. In short programming visualisation shows aspects of the program graphically, where as visual programming makes use of graphics as the program input medium. For initial informed introductions to both areas see references [1, 2].

But can such methods aid concurrent programming? I think that they can. Most of the reasons behind the adoption of graphics based programming tools centre on increasing the use of the left side of our brains, little used in the programming process at present. With the increased software complexity often shown in concurrent programs, it makes sense to bring as much as is possible of our underutilised brains to bear upon the task. Several recent reports from within the occam community [3, 4, 5, 6, 7] demonstrate the viability of such tools and can be seen as supporting this opinion.

If sufficient interest is expressed by members of the OUG, I will organize an initial SIG meeting at the Exeter technical meeting.

References

[1] B. A. Myers, *The state of the art in visual programming and program visualisation*, Report Nº CMU-CS-88-144, Computer Science Department, Carnegie Mellon University, Pittsburg; presented at the British Computer Society Displays Group's Symposium on *Visual programming and program visualisation*, London, 16 March 1988.

[2] Nan C. Shu, *Visual programming*, Van Nostrand Reinhold, New York, ISBN 0-442-28014-9, 1988.

[3] W.-D. Crowe, R. Hasson, P. E. D. Strain-Clark, *A CASE tool for designing deadlock free occam programs*, in the Proceedings of the 11th occam user group technical meeting, ed. John Wexler, *Developing transputer applications*, OUG-11, Edinburgh, IOS, September 1989.

[4] F. Mourlin, *Graphical environment for occam programming*, occam user group newsletter Nº 11, July 1989.

[5] M. Roberts, P. M. Samwell, *A visual programming system for the development of parallel software*, in the Proceedings of the Second International Conference on Software Engineering for Real Time Systems, Cirencester, IEE, September 1989.

[6] M. Stephenson, O. Boudillet, *GECKO: a graphical tool for the modelling and manipulation of occam software and transputer hardware toplogies* in the Proceedings of the 9th occam user group technical meeting, ed. Charlie Askew, *occam and the transputer – research and applications*, OUG-9, Southampton, IOS, September 1988.

[7] S. Stepney, *GRAIL: graphical representation of activity, interconnection and loading*, in the Proceedings of the 7th occam user group technical meeting, ed. Traian Muntean, *Parallel programming of transputer based machines*, OUG-7, Southampton, IOS, September 1987.

Mike Roberts m.roberts@uk.ac.city
The Centre for Information Engineering
City University
Northampton Square
London EC1V 0HB
United Kingdom

TECHNICAL CONTRIBUTIONS

SAFETY FIRST

Peter Welch, chairman of the occam user group

Performance – a secondary consideration

I was getting a little alarmed. Recently, the performance of our *transputer* system shot up overnight – our *T800*s doubled their MIPS rating from 10 to 20 and their MFLOPS went from 1·5 to 2·3! This upgrade cost us nothing and caused absolutely no disruption to our user service during the conversion. Indeed, our users never noticed a thing – but we now have much more exciting numbers to put on our posters to impress visiting industrialists.

A few compulsory sessions of reading X-window source code soon forced the truth out of our system administrator. These dramatic improvements were brought to us thanks to a new aggressive marketing policy from Inmos! After a few moments of quiet thought, this did not seem to be such a bad thing. In this world, technical excellence is no guarantee of success – it is not even a pre-requisite! Inmos is well justified in moving to the same level as its competitors and fighting to their rules. After all, there are some exciting questions to be answered. Can four *T800*s outperform an *i860*? Will the *H1* knock everything else out of sight? Never mind the quality – can you begin to feel the terraflops?

Hang on a moment. Chasing performance figures is not the only basis for impressing world opinion. There is another angle that I do not recall ever seeing promoted in any *transputer* marketing literature:

- ▹ an angle without which all the performance in the world is just candy-floss;
- ▹ an angle on which public attention is beginning to be focussed;
- ▹ an angle upon which the world will soon insist;
- ▹ an angle around which the *transputer* was conceived;
- ▹ an angle on which other technologies are a little bit wobbly.

With such a basis, surely the marketing people can come up with something fairly macho about...

Safety – the primary consideration

A 'safety critical' system is one whose failure may lead to loss of, or injury to, life or wallet. The percentage of computer systems that can be classified as safety critical is steadily growing. During the next decade, computers will not only be controlling the big things like commercial airliners, nuclear power stations and railway signals, we will be surrounded by them and dependent on them for most aspects of everyday living – personal communicators, gas cookers, car braking systems, etc. In the near future, almost all computer systems will contain some safety critical components.

The *transputer* processor, along with the *occam* programming language, is based upon formal mathematical theories of (parallel) computing [8, 9]. These theories have a long and mature pedigree. *Transputer* networks and *occam* processes obey a rich collection of simple algebraic laws [10, 11] that give them a chance of becoming amenable to formal specification, derivation and verification. These properties are an essential bedrock for the routine mass production of safety critical applications. By comparison, most other technologies in practice today are built upon sand.

High performance is a by-product of this discipline. Simple manipulations at all levels of design and implementation preserve the semantics of the system, allowing it to be tuned for various criteria of efficiency. But, performance must always be a secondary consideration – there is no merit in producing the wrong answers faster than all your competitors!

A little example – some school algebra

Forget about parallel computing. Consider a basic computational notion, common to all types of computing architecture and language: expression evaluation.

Occam expressions have a pure mathematical semantics – i.e. 'what you see is what you get' (WYSIWYG). For instance, the meaning of a sub-expression depends only on what is in that sub-expression and not upon its surrounding context. [In the higher realms of functional programming, this is known as 'Referential Transparency'.]

The practical importance of this is that expressions behave in the way we all learned at school – for example:

$$(a * b) + (a * c)$$

may be replaced by:

$$a * (b + c)$$

and the semantics of the whole program is unchanged. For 'classical' languages (like *FORTRAN*, *Pascal*, *C*, *Ada*, ...) this is not true: suppose a were a sub-expression whose evaluation side-effected the current state of the system!

Evaluating *occam* expressions cannot change system state:

- ▹ assignment of values to variables (:=) is not an operator and may not be embedded within expressions (unlike in *C*);
- ▹ the input/output processes (?, !) are not operators and may not be embedded within expressions (unlike many standard i/o functions in other languages e.g. *getchar* from *C*);
- ▹ no operators cause state change (unlike ++ in *C*);
- ▹ function calls cannot cause state change (unlike those in *FORTRAN*, *Pascal*, *C*, *Ada*, ...).

Consequences

A minor consequence of this freedom from side-effects is that the order in which operands are evaluated has no significance. Therefore, in *occam* there is no rule such as: 'operands are evaluated from left to right'. The designer has one less decision to worry needlessly about and the compiler has the freedom to choose the optimum ordering. Simple transformations may be applied – perhaps, by a tool other than the compiler – to assist in such optimisations. Of course, if more than one processing element is available, operand evaluation may also proceed in parallel.

[Actually, C *has no rule for ordering the evaluation of operands either. In this case, however, the conditions are ripe for a serious disaster – it's so easy to write code that has several different legal interpretations. If evaluating expressions can have procedural side-effects, then the order in which this happens does matter – even if the language definition pretends it does not!]*

The major consequence is the aid to our clear understanding of the meaning of code at this level. The properties are transparent – they are what we expect and there can be no nasty surprises.

These are not considerations of marginal importance. They are a necessary part of the foundations of a sound engineering discipline that will enable the production of safe software for safety critical systems.

If you accept the above arguments, there is an immediate corollary to contemplate: 'classical' programming languages have no safe rôle to play in safety critical applications. Since it is almost inevitable that those languages will nevertheless play major rôles in such systems, we have cause for grave concern.

Interim Defence Standard 00–55 (Draft)

The Interim Defence Standard 00–55, published as a draft by the UK Ministry of Defence [12], defines 'procedures and technical requirements' for the development of safety critical software.

A crucial new feature is that 'formal' (or, at least, 'rigorous') methods are to be used at all levels of specification, design, implementation and verification. Encouraging news for those of us with a nervous disposition – formal proofs (or 'rigorous arguments') may not completely guarantee our safety, but they certainly increase confidence levels and our present levels could do with a bit of a boost!

The software industry, however, seems to be in a small panic over this standard – largely for the wrong reasons (software people are extremely conservative!). Some proper reasons follow.

The 00–55 standard lists a number of approved formal specification and design techniques. These include both *CCS* [8] and *CSP* [9]. However, the document elsewhere prohibits the use of parallel processing (either across multiple 'processors' or within a single 'processor') as an 'unacceptable practice'! We are invited to use *CSP* for design but we are not allowed to exploit the results from that design!!

There is also a mandate that the programming language used must be 'high-level' and must have a compiler with an 'approved national or international validation certificate'. This seems to be aimed at the *Ada* world, but is probably also intended to allow other languages with ISO or ANSI standardisations (i.e. *C*, *FORTRAN* and *Pascal*). Unfortunately, none of these languages has a semantics that is simple enough or well-enough defined to be amenable to the use of formal methods (or even rigorous ones). The same is true for any credible subset (or superset) of these languages. The example issue raised earlier in this article is only the start of a long list of problem areas that are deeply embedded in their fundamental design – for further reading, see references [13, 14, 15].

Floating-point arithmetic is also banned! Presumably we are expected to use some fixed-point software instead. It is true that a realisation of floating-point arithmetic cannot be validated by testing. It is also true that one or two manufacturers have recently been embarrassed by some nasty bugs emerging from 'industry standard' hardware implementations. However, fixed-point arithmetic is no less easy to check-out and I know of no formally verified products. On the other hand, the *T800* floating-point micro-code was derived through formal transformation of *occam* source code that carries a formal proof of correctness with respect to a formal specification in *Z* of IEEE–754 arithmetic [16]. I know in which of these two mechanisms I would rather put my trust!

For additional observations on 00–55, see reference [17].

Concluding remarks

The fact that the M.o.D. has published 00–55 (with its central compulsory theme on the use of formal methods) is clear evidence that safety considerations are uppermost in the minds of a large group of customers for computer systems. Commercial buyers will certainly follow the military lead and demand at least as high a level of security.

There is a danger that the internal contradictions within 00–55 may give formal methods a bad name and that industrial systems suppliers may be able to push it away for some while. Marrying formal methods with classical programming languages will always lead to tears – but the problems lie mainly with the latter partners, not the former.

The pressure for safety will get stronger every year, eventually dominating the pressure for performance. Inmos is in an excellent position to capitalise on this market. The *T800* demonstrates that performance need not be sacrificed to meet the demands of safety – indeed that high performance is a consequence of high security.

Why does Inmos publicity not:

- ▷ highlight the formal methods being used in the design of the *H1*?
- ▷ boast about the security improvements delivered by *transputer*-based parallel computing?
- ▷ shout more loudly about *occam*?

After this, it would be interesting to find out about all those MIPS, MFLOPS and Mbaud – but these will merely be the icing on the cake!

References

[8] Robin Milner, *A Calculus for Communicating Systems*, ECS–LFCS–86–7, Laboratory for the Foundations of Computer Science, Edinburgh University, 1986.

[9] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice–Hall International, 1986.

[10] C. A. R. Hoare *et al.*, *Laws of Programming*, Commun. ACM Vol. 30, Nº 8, pp. 672–686, August 1987.

[11] A. W. Roscoe and C. A. R. Hoare, *Laws of Occam Programming*, Technical Monograph PRG–53, Oxford University Computing Laboratory, 1986.

[12] Ministry of Defence, *Requirements for the Procurement of Safety Critical Software in Defence Equipment (Interim Defence Standard 00–55)*, Directorate of Standardisation, Ministry of Defence, Room 5150A, Kentigern House, 65 Brown Street, Glasgow, G2 8EX, Scotland, May 1989.

[13] Peter Welch, *Going to Ceed?*, Internal Memorandum, Computing Laboratory, The University, Canterbury, Kent, CT2 7NF, England, March 1989.

[14] Roger Shepherd, *Compiler Support for Floating-point Computation*, Letter to the Editor, Software – Practice and Experience, 18, p. 1193, December 1988.

[15] Roger Shepherd, *Security Aspects of Occam 2*, Technical Note 32 (72–TCH–032–00), Inmos Ltd, Bristol, 1987.

[16] G. Barrett, *Formal Methods Applied to a Floating Point Number System*, Technical Monograph PRG–58, Oxford University Computing Laboratory, 1987.

[17] David May, *Draft Military Standard 00–55*, Internal Memorandum, Inmos Ltd, Bristol, July 1989 (appears in this newsletter).

Peter Welch
Computing Laboratory
The University
Canterbury
Kent CT2 7NF
United Kingdom

Tel: +44 227 764000 x3629
phw@uk.ac.ukc

DRAFT MILITARY STANDARD 00–55

David May, Inmos Ltd, Bristol

The proposed standard for safety critical software contains many requirements for *mathematically* proven software. This note comments on these proposals.

The state of the art

Over the last twenty years, considerable progress has been made in modelling and reasoning about computer programs. A variety of notations and languages have been developed; some of these are supported by computer-assisted proof systems. Examples of program verification systems are the Boyer-Moore theorem prover, Gypsy, AFFIRM, SDVS, LCF, m-EVES and NuPRL. Although some impressive proofs have been done, the state of the art in mechanised program verification is still limited to small programs (i.e. a few tens of pages of code).

New theories and formalisms to support software engineering have also emerged. These include CCS and CSP for concurrent programs, and VDM and Z for system specification. So far no theorem proving tools for these formalisms have been implemented, but some useful computer-based tools related to them have been developed (e.g. the 'B tool' for Z, and the occam transformation system). However, these do not support the construction of machine checked proofs in an explicit logical calculus.

In summary, it is now clear that formal methods are applicable to several areas of hardware and software design. In some cases, mechanised proof systems have been demonstrated. There have been one or two commercially important applications of formal methods (such as the use of Z and the occam transformation system in the design of the Inmos IMS T800 transputer floating point unit). However, none of the languages and tools have yet reached sufficient maturity for widespread use.

Programming languages

There are few programming languages suitable for the kind of mathematical verification process outlined in the proposed standard. In order to perform mathematical verification of programs, the programming language must have a mathematical semantics. The semantics must be small enough to allow it to be used in the construction of program proofs – ideally, it must be small enough to be embedded in a proof checker. All of the systems mentioned above employ simple languages or language subsets – these languages have been designed to be simple enough to allow program proofs. *No existing internationally standardised language has a simple enough semantics for this purpose.* Even if sufficient resources were made available to construct a mathematical semantics for these languages, it would not be suitable for program proofs.

The best solution (from a technical viewpoint) would be to create a new (international standard?) language for safety-critical software. As it is only necessary to write small programs with simple control structures and simple data structures, a very small language designed for mathematical verification can be used. It would require a simple – and verifiable – compiler. An alternative might be to standardise the occam language (or a language closely related to it) – see the appendix to this note.

Compilers

There are no *verified* compilers for existing standard languages. The languages are all too big and complex. Further, there is an increasing trend towards the use of optimising compilers. The standard prohibits the use of optimising compilers – but all compilers optimise to some extent. Which optimising techniques are considered unsafe?

Validation is not a substitute for verification. It can be used to show the presence of bugs in the compiler, but not to show their absence. The best approach would be to employ a simple language for which a simple verified compiler can be constructed. An alternative process would be to use proof techniques to prove that the compiler output (instruction level program) implements the high-level language program.

If there is no method of verifying that the assembly language program correctly represents the high level language program, it would be better to write the programs in assembly language and verify them mathematically in terms of a mathematical specification of the machine instructions.

Programming restrictions

The proposed list of restrictions in the standard appears ad-hoc and inconsistent with the proposed list of design methods. If the use of multiprocessing and parallel processing are both prohibited, where is the need for CSP and CCS? Why has use of `goto` statements, exceptions and pointers not been prohibited?

Specifically, with reference to section 21 of the draft standard (see figure 1):

1. Floating point arithmetic may be dangerous – but it seems better to use IEEE arithmetic (for which there is a mathematical specification in Z) than un-specified fixed point arithmetic.
2. Although there is no obvious need for the use of recursion in safety-critical software, there is no reason to treat it as a dangerous technique. It is often much easier to prove the correctness of recursive programs than the equivalent non-recursive ones; for safety critical applications there is the additional obligation to prove that the depth of recursion is bounded.
3. The problem with interrupts is the interaction between the interrupt service routine and the interrupted program. Is the idea to allow programs which consist of *only* a timer service routine? If not, this restriction does not seem to achieve anything.

```
21 Unacceptable Practices
21.1 This Standard prohibits the use of practices which are unsafe,
or difficult to analyse such as:
1. floating point arithmetic;
2. recursion, whether simple or mutual;
3. interrupts, except for a timer interrupt at fixed intervals;
4. assembly level programming languages;
5. dependence on separate elements being executed on parallel
   asynchronous processors. The complete set of Safety Critical
   Software modules should run in a single processor;
6. multi-processing on a single processor;
7. object code patching;
8. software architectures that are re-configurable under
   application program control;
9. dynamic memory management.
```

Figure 1: text of section 21 of the draft standard

4. **It is dangerous to ban assembly language programming until a verified compiler is available** (see above).
5. The great success of CCS, CSP (and its implemented subset, occam) is that they provide a design method, mathematical proof tools and associated programming language for constructing secure concurrent systems – both for parallel processing and for multiprocessing within a processor. These tools are *much more reliable* than using complex sequential programs to achieve the same effect. They have been developed over many years *specifically* to overcome the difficulties of constructing secure real-time systems.
6. See above.
7. Object code patching should clearly be prohibited.
8. It is not clear what 'reconfiguring the software architecture' means.
9. Dynamic memory management – this is probably best prohibited for the moment.

Summary

The above remarks can be summarised:

▷ The theoretical tools *do* exist to allow mathematically verified software to be constructed. Although the existing techniques can not cope with large programs, they *could* be employed for the kind of safety critical programs described in the standard.

▷ Existing standard languages and their compilers do not form a suitable basis for mathematical verification. Much smaller, simpler languages are needed. They must be designed with verification in mind *from the outset.*

▷ Much *folklore* has grown up about which programming techniques are unsafe. In particular, there is a tendency to regard all forms of concurrency and non-determinism with suspicion. This view is wrong. Secure, high-level programming techniques, supported by rigorous mathematical verification techniques, now exist for programs incorporating parallelism and non-determinism. Indeed, such programs are often *easier* to reason about and test than the alternative (a relatively complex sequential program). It is important not to prohibit the very design techniques which make mathematical verification possible!

▷ A substantial investment will be needed to create – and standardise – the language(s) and tools (proof checkers, verified compilers) needed. It will also be important to consider the relationship with mathematically verified hardware (the specification of a machine instruction set is needed in order to verify the compiler). See the appendix to this note.

Lastly, it should be noted that although the UK has considerable strengths in the theory and practice of program verification, a very substantial investment in training (and re-training) will be needed. It is unlikely that enough people are available to *teach* these techniques: perhaps the most important first step is a major investment in higher education focused around existing centres such as Oxford University Programming Research Group, Manchester University, Edinburgh University Laboratory for the Foundations of Computing Science and Cambridge University.

Appendix

The SAFEMOS project

The construction of fully verified systems is currently a research activity. However, commercial interest in formal verification is growing rapidly in view of the increasing use of microprocessors in real-time control applications. The UK IED project SAFEMOS (collaborative project 1036: Inmos, SRI Cambridge, Oxford University, Cambridge University) involves the construction of languages, compilers and hardware for verified systems.

The SAFEMOS project uses the occam language developed together with the Inmos transputers. Of all the languages designed to support verification, occam is probably most widely used – and primarily in embedded real-time applications. This language is designed to support both sequential and concurrent programs. It has a rich mathematical semantics facilitating program transformation and proof. Its compilers provide extensive static checking including full alias-checking and disjointness-checking for concurrent programs. There is some interest in standardising occam as a language for formally verified systems software.

The SAFEMOS project aims to develop a program verifier, verified compiler and verified processor allowing mixed hardware and software systems to be designed and verified. As far as possible, mechanised proof-checking will be provided by the HOL theorem prover. The processor will be similar in design to the Inmos transputer, allowing concurrent systems to be constructed and verified.

David May
Inmos Limited
1000 Aztec West
Almondsbury
Bristol BS12 4SQ
United Kingdom

AN OPTIMAL TOPOLOGY FOR MULTICOMPUTER SYSTEMS BASED ON A MESH OF TRANSPUTERS

A. Arruabarrena[1], R. Beivide[1], E. Herrada[2], J. L. Balcázar[2], C. Izu[1]
[1]Euskal Herriko Unibertsitatea, Donostia, Spain
[2]Universitat Politècnica de Catalunya, Barcelona, Spain

In this paper we present the construction of an effective interconnection scheme to tie any number of transputers together, in order to implement a multicomputer system. Also, we present one way to solving the problem of routing messages through this kind of network on a transputer-based multicomputer.

I. Introduction

A multicomputer architecture [18] is a parallel computer system that consists of N computers, called *nodes*, connected by a message-passing communication network. In this paper we consider multicomputers with nodes based on processors like the T800 transputer [19].

One of the important problems to be addressed in such architectures refers to the interconnection scheme which ties all of the system nodes together. The most essential function of the interconnection network is that it allows an efficient message interchange between processes which execute on the nodes. The characteristics of the interconnection topology directly affect the expected performance of the global system.

To obtain a high performance in a multicomputer system, the structure of the interconnection scheme must accomplish several conditions. Different performance metrics, based on graph theory, are commonly used in order to characterize the merits of the proposal. One generally accepted set of network parameters which is adequate for this purpose includes the following: total number of links, diameter, average distance, link- and node-connectivity, symmetry, embeddability of algorithms and extensibility. Let us consider briefly each of these network parameters.

The network *degree* refers to the maximum number of links incident with a node. Making constant and small the network degree signifies a simplicity for the routing policy, as well as a reduction in the cost of nodes and links. The network *diameter* and *average distance* are measures related to the maximum and average delays during transmission of messages. To keep the values of these parameters as small as possible is desirable in order to obtain a high system throughput. The *link-* and *node-connectivities* refer to the minimum number of links and nodes, respectively, whose removal results in a disconnected network. These two parameters represent a measure of the robustness exhibited by the network. Node *symmetry* makes the network look the same when viewed from any node, and is related to the reduction in the complexity of designing distributed routing algorithms, as well as to programmability issues. Good *embeddability* of many important parallel computation graphs – such as rings, trees, meshes and others – allows the topology to attain an efficient matching to the communication structure of many well-known parallel applications. Finally, the *extensibility* of a network is an important property which allows a graceful scaling of the network size.

In previous work [20, 21], the authors have proposed an intercommunication topology based on a mesh with wrap-around links. It turns out that this scheme is especially suitable for connecting any arbitrary number of transputer nodes configuring a multicomputer system, since it provides a good trade-off in the network metrics described above. To clarify this suitability, let us summarize here the main features exhibited by this kind of interconnection topology.

These networks are optimal with respect to two distance parameters simultaneously, namely diameter and average distance, among all 2D meshes with wrap-around links, and so the maximum and average transmission delays are minimized. We therefore call them *midimew* networks (**mi**nimum **di**stance **me**sh with **w**rap-around links). They are also regular, node-symmetric and maximally connected and have degree four. One such optimal midimew network exists for any given number of nodes.

In section II we review the construction procedure given in reference [21], in order to introduce midimew networks. Their distance properties are also stated. See reference [21] for proofs and additional properties. In section III, we consider the problem of routing messages between nodes through the network. Assuming that the processor of the system node is a transputer, we provide a distributed routing algorithm written in occam 2 that conveys messages between nodes following a shortest path.

II. Midimew networks

We present in this section the essentials of a systematic method, fully described in reference [21], for constructing midimew networks with any number of nodes $N > 2$. We provide also analytic expressions to compute diameter and average distance parameters for this class of networks. Finally, we compare briefly the topological characteristics of midimew networks with other well-known static interconnection topologies.

Procedure

(Figure 2 may help in understanding the procedure.)

Step 1 Find the following parameters:

$$b = \left\lceil \sqrt{\frac{N}{2}} \right\rceil, \quad r = \left\lceil \frac{N}{b} \right\rceil b - N, \quad h = b + r, \quad v = \left\lceil \frac{N}{b} \right\rceil - r$$

Step 2 Design a rectangular grid of width h and height v.

Step 3 If $r \neq 0$ and $v \neq b-1$, then discard from the drawn grid a leftmost upper rectangle of width r and height $v - b + 1$.

Step 4 Establish the wrap-around links: Connect each node $(i, 0)$ to the top node of the column $(i + r) \bmod h$, and connect each node $(h-1, j)$ to the leftmost node of the row $(j + b - 1) \bmod v$.

That completes the construction of the midimew network. The networks in figures 3 and 4 have been obtained using this algorithm, and the nodes have been labelled in

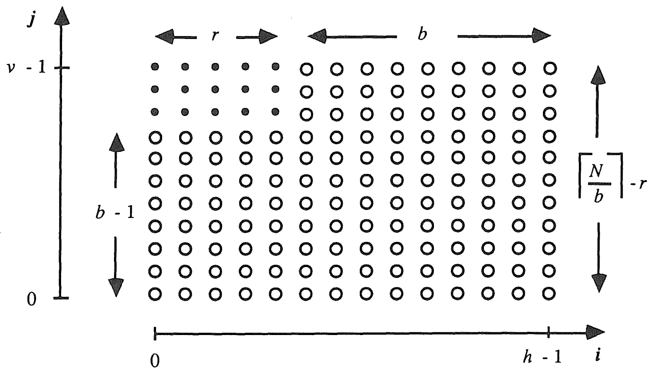


Figure 2: dimensions of midimew networks

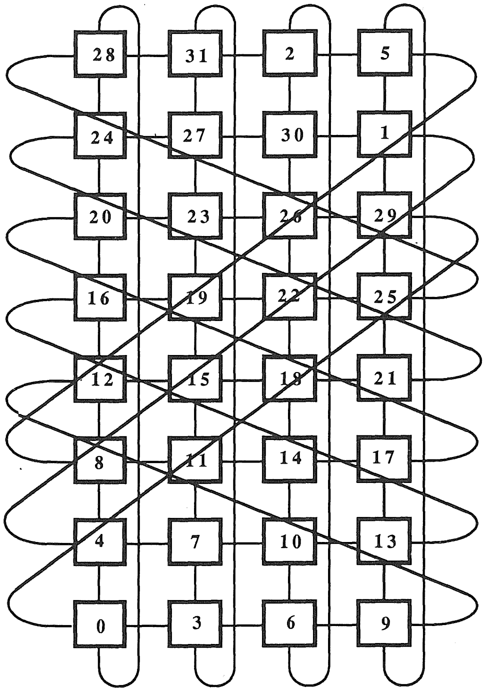


Figure 3: midimew network with $N = 32$ nodes

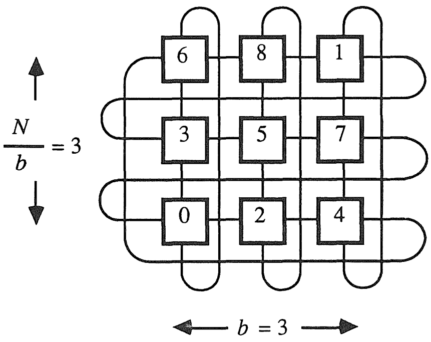


Figure 4: square midimew with $N = 9$

the following way: every node i, with $0 \leq u \leq N - 1$ is adjacent to the nodes

$(i + b) \bmod N$	(North displacement)
$(i - b) \bmod N$	(South displacement)
$(i + b - 1) \bmod N$	(East displacement)
$(i - b + 1) \bmod N$	(West displacement)

We label nodes in this way in order to preserve a graph isomorphism between midimew networks and a family of optimal circulant graphs described in reference [21]. This isomorphism allows us to obtain analytic expressions for the network diameter and the average distance, to be described in this section, and is the basis for the routing algorithm that we will present in section III.

Rectangular and square cases

It can be seen from figure 2 that fully rectangular midimew networks are obtained in each of the following cases:

1. When $b + r = b$, i.e. $r = 0$. This case occurs when N is a multiple of b, and then the dimensions of the network are $h = b$ and $v = N/b$. For each b, exactly four such full rectangles appear, with sizes $v = 2b - 3$, $v = 2b - 2$, $v = 2b - 1$ and $v = 2b$, corresponding to $N = 2b^2 - 3b$, $N = 2b^2 - 2b$, $N = 2b^2 - b$ and $N = 2b^2$.
2. When $b - 1 = \lceil N/b \rceil - r$, i.e. $r = \lceil N/b \rceil - b + 1$. This case occurs when $N = (\lceil N/b \rceil + 1)(b - 1)$ and then the dimensions of the network are $h = N/(b - 1)$ and $v = b - 1$. For each b, exactly one such full rectangle appears, namely that with $h = 2b - 1$, that is, when $N = 2b^2 - 3b + 1$.

Remark that, out of the $4b - 2$ midimew networks existing for each value of $b > 2$ ($N > 8$), only five of them are rectangular. On the other hand, only two square midimew networks exist, corresponding to the cases where $N = 4$ or $N = 9$. This can be easily seen from the fact that a square midimew can be obtained only if $b = N/b$ or $b - 1 = N/(b - 1)$, and that the second equality never holds. Some manipulations show that the first case implies that $N < 4\sqrt{N} - 2$. Solving for $\sqrt{N}$ yields that $N < 12$, and thus the only perfect squares fulfilling this condition are 4 and 9. Figure 4 presents the square midimew $N = 9$.

A generally desirable characteristic is to achieve high density meshes (large number of nodes) yet minimizing the number of wrap-around links, since they may raise some implementation difficulties. Square meshes with wrap-around links present an advantage over similar networks with rectangular or 'L' shapes, because a square is the rectangle with minimum the perimeter for a given area. The number of wrap-around links in midimew networks is equal to the value of the network semiperimeter, and the area corresponds to the total number of nodes. Midimew networks have been chosen by looking for the minimimum perimeter.

Diameter and average distance in a midimew network

We present here the analytical expressions that give the diameter and the average distance for midimew networks. These expressions can be derived from theorems 1 and 2 of reference [20].

The value of the diameter k can be expressed, as a function of N, as

$$k = \begin{cases} b-1 = \left\lceil\sqrt{N/2}\right\rceil - 1 & \text{if } n \leq 2b^2 - 2b + 1 \\ b = \left\lceil\sqrt{N/2}\right\rceil & \text{if } n > 2b^2 - 2b + 1 \end{cases}$$

The value of the average distance can be expressed, as a function of N, as

$$\bar{k} = k\left[1 - \frac{2(k^2-1)}{3(N-1)}\right]$$

Both values are lower bounds for the diameter and the average distance of any 2D mesh topology with wrap-around links [20]. Due to the simultaneous minimization of both parameters, the midimew networks are an optimal family of 2D meshes with wrap-around links. Moreover, these topologies can be directly implemented with transputers, exploiting all of their connectivity.

In order to estimate the suitability of midimew networks we have compared several well-known static interconnection networks (2D and 3D mesh, 2D and 3D torus and hypercube) and the alternative proposed here, midimew [21]. As an example, figure 5 illustrates the comparison of all these networks with respect to a typical cost measure [22], namely the product of diameter and degree for a number of nodes up to around 2200. Present medium-grain multicomputers fall in this range.

III. Routing strategy

Let us consider now the problem of routing messages in a transputer-based midimew network. We simplify this problem by considering two processes running in parallel on each node: a computing process and a routing process, communicating through a buffer (see figure 6). We are interested in the routing of messages between nodes through the network, excluding the passing of messages between processes in the same node.

Every node can receive and send messages from/to any of the four channels which connect it with its neighbour nodes. We have named these channels *North*, *South*, *East* and *West* respectively, as can be seen in figure 6.

The process that must take routing decisions reads information from any of the five channels, one from each neighbour and one from its own local computing process. If the received message has reached its destination, the routing process sends the

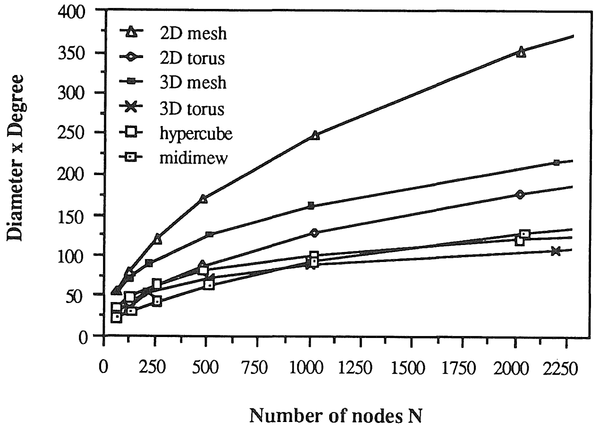


Figure 5: diameter times degree as a function of N

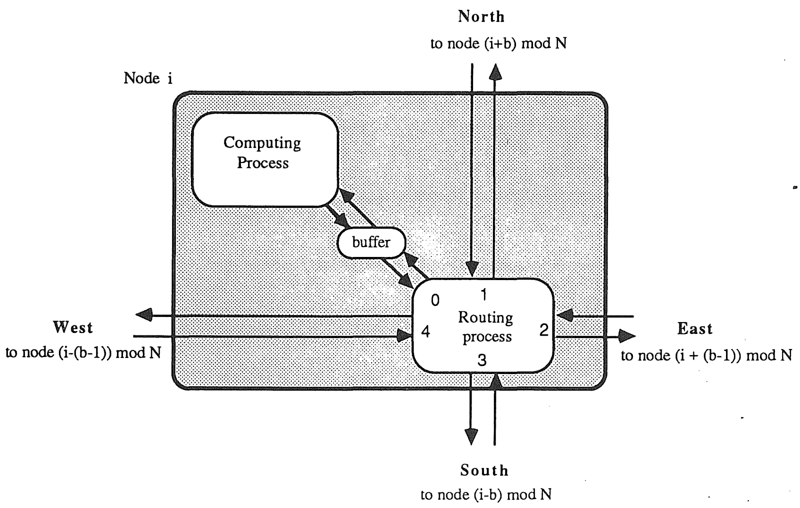


Figure 6: channels and processes of a node

message to the local computing process. Otherwise, it will take the appropriate routing decision sending the received message in an appropriate direction.

The routing algorithm is quite simple and can be formally derived from the topological properties of the midimew networks [21]. Messages are sent from source node to destination node using one shortest path of the network joining those nodes. The routing strategy sends messages first in the E–W direction (hops of length $b-1$) and next in the N–S direction (hops of length b).

The message format must include the length of the message, the destination address and the message itself. The addresses of the nodes correspond to the labelling proposed in section II.

Message length	Destination address	Data byte 1	Data byte 2	...

If we want more than one process to be executed in the same network node, sharing the processor time, we must add a process identifier to each message passed between processes.

In our approach, when a parallel program is being executed on this kind of multicomputer all the nodes must perform the same routing algorithm. When a message is read, the router calculates the difference, m, between the destination address and its own address. If $m = 0$, then the message has reached the destination node and it is sent to the local computing process. Actually, the message is transferred to a buffer that stores it for further processing, so as to try to avoid blocking of the router in case the computing process is not ready to accept the message. Otherwise, some decisions are taken in order to send the message in the appropriate direction. If the message has not yet completed its path along the first direction (E–W), it is routed again in this direction. Otherwise, it is sent to the second direction (N–S).

An occam 2 implementation of the routing process for a transputer-based midimew network is shown in figure 7. The procedure uses two arrays of five channels, *input* and *output*. The channel numbered 0 links the routing process and the local computing process, and channels numbered 1, 2, 3 and 4 represent *North*, *South*, *East* and *West* links respectively, connecting to neighbouring routing processes.

	0	1	2	3	4
Input channels	Local	North	East	South	West
Output channels	Local	North	East	South	West

Let us examine an example of the routing mechanism for the network shown in figure 3. A message that goes from node 19 to node 9 will be sent first through the *West* channel, to node 16. The routing process in node 16 will reroute it in the same direction to node 13. At this node the routing process will send the message to the *South*, and it will finally arrive at its destination, node 9, where will be consumed.

Another way to route messages through the network consists of computing, at the sending node, a *routing record* with the number of hops the message must make in each direction, (x, y) [21]. This routing record is sent as the header of the message instead of the destination address. Then, the intermediate nodes route the message first in one direction, namely y, updating this element of the routing record. When the message has accomplished all the hops in this direction the routers will send it in the x direction towards the destination node. To perform this strategy, the routing process must be divided in two parts: one to compute the routing record, for those

```
VAL INT N IS 32 :          -- Global variables in function of the network size
VAL INT b IS 4 :
VAL INT max_length IS 200 :                      -- maximum buffer size

PROTOCOL message IS INT::[]INT :
PROC  routing (VAL INT source, [5] CHAN OF message input,
                               [5] CHAN OF message output)
  INT m, remainder, cond, sign, length:
  [max_length] INT buffer :
  WHILE TRUE
    ALT  i = 0 FOR 5
      input[i] ? length::buffer    -- read a message from an input channel
        SEQ
          m := buffer[1] - source  -- calculate dest. - source (current node)
          IF
            m <> 0
               SEQ
                 sign := -1
                 IF
                   m > (N/2)
                      m := N - m
                   r > 0
                      sign := 1
                   m > (-(N/2))
                      m := -m
                   m > (-N)
                      SEQ
                        sign := 1
                        m := N + m
                 remainder := m REM b
                 cond := (m/b) - (2*remainder)
                 IF
                   remainder > 0
                     IF
                       ((b - cond) * sign) > 0
                         output[4] ! length::buffer  -- send to the west
                       ((b - cond) * sign) < = 0
                         output[2] ! length::buffer  -- send to the east
                   remainder = 0
                     IF
                       sign = 1
                         output[1] ! length::buffer  -- send to the north
                       sign = (-1)
                         output[3] ! length::buffer  -- send to the south
            m = 0
              output[0] ! length::buffer        -- message at destination
:
```

Figure 7: routing algorithm for a midimew network with 32 nodes

cases in which the message is originated at the node itself, and the other to route messages. This technique could be more suitable to implement an adaptive routing policy, as can be seen in reference [23].

References

[18] W. C. Athas, C. L. Seitz *Multicomputers: message-passing concurrent computers*, IEEE Computer, August 1988, pp. 9–24.

[19] *IMS T800 transputer*, Inmos Product Data, 72 TRN 116 00, February 1987.

[20] R. Beivide, E. Herrada, J. L. Balcázar and J. Labarta, *Optimized mesh-connected networks for SIMD and MIMD architectures*, Proc. 14th Int. Symp. on Comput. Archit., June 1987, pp. 163–170.

[21] R. Beivide, E. Herrada, J. L. Balcázar and A. Arruabarrena, *Optimal distance networks of low degree for parallel computers*, Research Report FISS-I-33.1-ATC-89, Informatika Fakultatea, EHU 1989 (Submitted for publication).

[22] P. W. Dowd and K. Jabbour, *Spanning multiaccess channel hypercube computer interconnection*, IEEE Trans. on Comp., Vol. 37, Nº 9, September 1988, pp. 1137–1142.

[23] C. R. Jesshope, P. R. Miller and J. T Yantchev, *High performance communication in processor networks*, 1989 Int. Symp. on Comp. Arch., pp. 150–157.

A. Arruabarrena, R Beivide and C. Izu
Informatika Saila
Euskal Herriko Unibertsitatea
649 p.k.
20080 Donostia
Spain

E. Herrada and J. L. Balcázar
Facultat d'Informatica
Universitat Politècnica de Catalunya
Pau Gargallo 5
08020 Barcelona
Spain

agustin@gorria.if.ehu.es

AN EXPERIMENT IN PARALLELIZING EDGE-DETECTION

Fabrizio Imelio, Università di Pisa, Italy

I have parallelized four algorithms for edge detection

- ▹ Canny: Gaussian filter followed by second directional derivative;
- ▹ Marr–Hildreth: Gaussian filter followed by Laplacian operator;
- ▹ Nevatia–Babu: template matching followed by edge point selection;
- ▹ Bartliff: structure shown in figure 8;

each on a three-dimensional cube of T800 transputers. The transputers are allocated on a B012 motherboard with a software-programmable crossbar-switch.

There were two objects in parallelizing the algorithms:

- ▹ a balanced division of work, so that each transputer executes the algorithms on the same quantity of data;

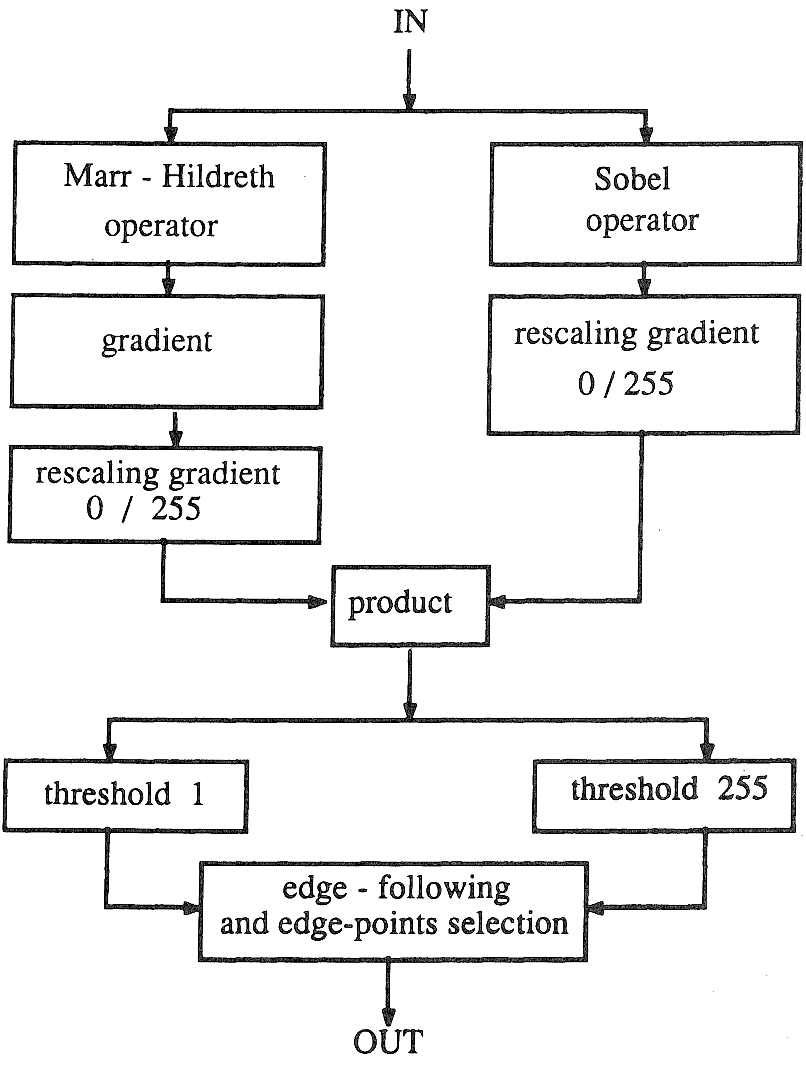

IN
Marr - Hildreth operator
Sobel operator
gradient
rescaling gradient 0 / 255
rescaling gradient 0 / 255
product
threshold 1
threshold 255
edge - following and edge-points selection
OUT

Figure 8: Bartliff edge detector

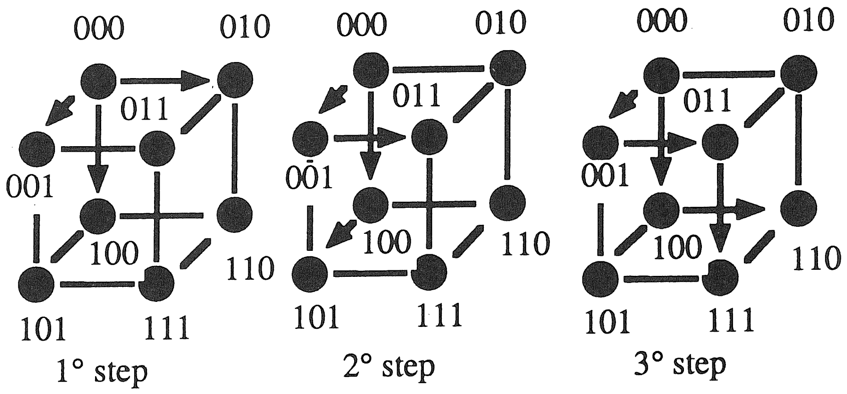


Figure 9: steps in distributing·the work in a cube

▷ efficient distribution of work, to minimize the number of steps required to send the data from the root to all of the nodes of the machine and vice versa.

Figure 9 shows the data flow during the sending of work to all the nodes or the cube. This operation is performed in three steps, that number being the dimension of the cube.

All the algorithms are implemented on a MicroVAX (in C), on a single transputer, and on the cube. The execution times and efficiencies are as follows:

algorithm	MicroVAX	1 × T800	8 × T800	efficiency
Canny (7 × 7)	218.1	39.3	6	0.82
Marr–Hildreth (15 × 15)	350.7	62.6	8.51	0.92
Nevatia–Babu	374.26	63.35	8.94	0.88
Bartliff	597.6	107.4	17	0.79

The efficiency value is $t_1/(nt_n)$ where t_1 is the execution time on a single transputer and t_n is the execution time on a cube of n transputers.

These experimental results show that the algorithms have been parallelized with an efficiency greater than 80%.

Fuller details can be found in the paper: *Real-time processing architectures in ground surveilance systems*, by S. Bottalico, F. de Stefanis and F. Imelio, presented at the 5th International Conference on Image Analysis and Processing, Positano, Italy (1989).

Fabrizio Imelio
c/o Selenia S.p.A.
via S. Maria, 83
56100 Pisa
Italy

REALLY EFFICIENT MULTIPLE BUFFERING IN OCCAM AND EFFICIENT FAIR ALTS

Piers A. Shallow, Marconi Maritime Applied Research Laboratory, Cambridge[1]

Although I was originally going to reply to G. Jones' article on *Carefully scheduled selection with ALT* [24], it was the use of the word 'efficient' in his article on *Efficient multiple buffering in occam* [25] which has enticed me to write and put forward the methods I have been using.

In both articles, I feel that G. Jones has forgotten to stress the enormous processing time wasted in setting up the replicator constructs; indexing into arrays (both channel and data arrays); the use of Boolean guards on channels; and the use of the remainder operation to obtain the modulus of the index pointer, all of which I have deliberately avoided for purpose of performance, at the cost of memory space.

The way in which I have been implementing a 'fairer `ALT`' with a small n, is by replicating the code of the `ALT` (`PRI ALT`) and its alternatives n times by using the 'attach file' utility (method 1). Within each `ALT` all n alternatives are rotated by one position (up or down) such that the bottom alternative becomes the priority one (or vice versa). I have 'abbreviated' the channels and data arrays as single elements and have defined them once outside of the '`WHILE` *condition*', removing any indexing or repetitive abbreviations. The use of data arrays is questionable and I use, whenever possible, only one 'data type' variable for all the alternatives.

The method I have been using for the 'multiple buffer' is an unwrapped replicated `SEQ`, where the format of the code is repeated three times but the variables used are rotated around (method 2) [26, without the procedure calls]. The deliberate use of unique variable names and channels has meant that the variables and channels are never shared between parallel processes within a `WHILE` construct.

In both cases I have also avoided the use of procedure calls when ever possible as they take time to initiate by embedding the code into the processes with the 'attach file' utility.

The methods mentioned above have reduced the amount of processing overheads required to implement the programs by a factor of about three. The approximate number of processor cycles [28] (excluding the amount of time it takes to transfer data across a channel and the amount of 'process' code) was reduced from 1608 cycles [24, favourite code] to 495 cycles (method 1), and from 1063 cycles [25, figure 11] to 386 cycles (method 2) for the 'fairer `ALT`' (with $n = 4$) and the 'multiple buffer' respectively. The compiled size of the code (excluding the code required for the 'process') for implementing the 'multiple buffer' programs are the same, however my implementation of the 'fairer ALT' program is two and a half times as large with $n = 4$ and 24 times as large with $n = 10$. (Incidentally, the D700D TDS will not compile the program in figure 11 of reference [25] as legal occam.)

I have used the word 'fairer' as the examples given are far from fair. Take my implementation of the `ALT`, assume that only the channels from G onwards are always ready and start with alternative A as the favourite. As channels A to F are not active, alternative G is selected and processed. At the start of the next `ALT` the favourite alternative now becomes B. As B to F are still inactive and because G has become

[1]This document expresses the opinion of the author and not that of the company.

ready, it will be selected and processed again. The favourite alternative now becomes C on the next `ALT`. As alternative G is always ready at the start of each `ALT` it will always be selected until the favourite alternative becomes H. Alternatives I to N will then be selected in turn as the favourite progresses through to N. At this point the favourite alternative becomes A again, where upon alternative G is reselected. As a result alternative G is effectively selected seven times to every single selection of alternative H to N, which is far from fair. This problem is also inherent in G. Jones' favourite solution [24].

In reference to R. Peel's problem [27] of ensuring that it is possible for the buffers to be emptied, there are two very simple programs (figures 10 and 11) which achieve this property. Needless to say these programs do not support the subtlety incorporated in R. Peel's program which has the flexibility of allowing the processing to start after the data has been read in and allowing the data to be transmitted once the processing has finished, i.e. the simple programs run alternately two of the three processes at once and then only one, instead of always running two of the three processes.

Again it is possible to increase the performance of R. Peel's program by expanding the program and removing the data indexing (figure 12) and can be condensed down into to two parallel processes (figure 13). Unfortunately the variables are still shared between the parallel `WHILE TRUE` processes, making the program illegal occam.

In attempting to reduce the number of internal communications, the solution I obtained (figure 14) ended up sharing the external channels between two parallel `WHILE TRUE` processes, which again is illegal occam, although their use is strictly controlled by the synchronisation of the internal channels. In rearranging the program, to make it legal occam, I have arrived with an alternative (figure 15) which has the same basic characteristics as R. Peel's program, but with fewer parallel processes and no internal communications. This program only takes about 196 processor cycles (excluding the time taken to transfer data over a channel and to run the 'process' code) instead of 691 processor cycles required for figure 14 (a factor of about three).

I would also like to bring to light an underlying problem of using protocols within a multiple buffering environment. Take for example a simple buffering program, G. Jones' program [25, figure 9], where a simple protocol has been used. If we attempt to replace the protocol with a tagged protocol we rapidly become stuck because the `CASE` statement has to be used. Should there not be an easier way of running parallel input and output channels without the need to use an additional flag (tag) which has to be set upon the input tag option and used as the means of selecting the output data (figure 16)?

References

[24] G. Jones, *Carefully scheduled selection with ALT*, OUG newsletter Nº 10, January 1989.

[25] G. Jones, *Efficient multiple buffering in occam*, OUG newsletter Nº 11, July 1989.

[26] Phil Atkin, *Performance maximisation*, INMOS Technical Note 17.

```
PROC solutionA (CHAN OF p.data.type Chanin,
                CHAN OF p.data.type Chanout)

  data.type   dataA, dataB:
  SEQ
    Chanin ? dataA
    WHILE TRUE
      SEQ

        PAR
          Chanin ? dataB
          SEQ
            ...  process dataA
            Chanout ! dataA
        PAR
          Chanin ? dataA
          SEQ
            ...  process dataB
            Chanout ! dataB
:
```

Figure 10: simple solution A

```
PROC solutionB (CHAN OF p.data.type Chanin,
                CHAN OF p.data.type Chanout)

  data.type   dataA, dataB:
  SEQ
    Chanin ? dataA
    ...  process dataA
    WHILE TRUE
      SEQ
        PAR
          SEQ
            Chanin ? dataB
            ...  process dataB
          Chanout ! dataA
        PAR
          SEQ
            Chanin ? dataA
            ...  process dataA
          Chanout ! dataB
:
```

Figure 11: simple solution B

```
PROC peel.solution (CHAN OF p.data.type Chanin,
    CHAN OF p.data.type Chanout)
  CHAN OF INT c.1.2, c.2.3, c.3.1:
  data.type   dataA, dataB       :
  PAR
    {{{  input data
    INT    data.validA, data.freedA:
    INT    data.validB, data.freedB:
    SEQ
      WHILE TRUE
        SEQ
          Chanin ? dataA
          PAR
            c.1.2 ! data.validA -- dataA now valid
            c.3.1 ? data.freedB -- wait for free dataA
          Chanin ? dataB
          PAR
            c.1.2 ! data.validB -- dataB now valid
            c.3.1 ? data.freedA -- wait for free dataA
    }}}
    {{{  process data
    INT    data.validA, data.readyA:
    INT    data.validB, data.readyB:
    SEQ
      c.1.2 ? data.validA
      WHILE TRUE
        SEQ
          ...  process dataA
          PAR
            c.1.2 ? data.validB -- wait for valid dataB
            c.2.3 ! data.readyA -- dataA now ready
          ...  process dataB
          PAR
            c.1.2 ? data.validA -- wait for valid dataA
            c.2.3 ! data.readyB -- dataB now ready
    }}}
    {{{  output data
    INT    data.freedA, data.readyA:
    INT    data.freedB, data.readyB:
    SEQ
      PAR
        c.3.1 ! data.freedB     -- dataB now free
        c.2.3 ? data.readyA     -- wait for ready dataA
      WHILE TRUE
        SEQ
          Chanout ! dataA
          PAR
            c.3.1 ! data.freedA -- dataA now free
            c.2.3 ? data.readyB -- wait for ready dataB
          Chanout ! dataB
          PAR
            c.3.1 ! data.freedB -- dataB now free
            c.2.3 ? data.readyA -- wait for ready dataA
    }}}
:
```

Figure 12: Peel's solution expanded

```
PROC my.condensed.solution (CHAN OF p.data.type Chanin,
                            CHAN OF p.data.type Chanout)

  CHAN OF INT c.1.2, c.2.1:
  data.type   dataA, dataB:

  PAR
    {{{  input data
    INT    data.validA, data.freedA:
    INT    data.validB, data.freedB:
    SEQ
      Chanin ? dataA
      c.1.2 ! data.validA       -- dataA now valid

      WHILE TRUE
        SEQ
          Chanin ? dataB
          PAR
            c.1.2 ! data.validB -- dataB now valid
            c.2.1 ? data.freedA -- wait for free dataA

          Chanin ? dataA
          PAR
            c.1.2 ! data.validA -- dataA now valid
            c.2.1 ? data.freedB -- wait for free dataB
    }}}
    {{{  process and output data
    INT    data.validA, data.freedA:
    INT    data.validB, data.freedB:
    SEQ
      c.1.2 ? data.validA       -- wait for valid dataA
      ...  process A

      WHILE TRUE
        SEQ

          PAR
            SEQ
              c.1.2 ? data.validB       -- wait for valid dataB
              ...  process dataB
            SEQ
              Chanout ! dataA
              c.2.1 ! data.freedA       -- dataA now free

          PAR
            SEQ
              c.1.2 ? data.validA       -- wait for valid dataA
              ...  process dataA
            SEQ
              Chanout ! dataB
              c.2.1 ! data.freedB       -- dataB now free
    }}}
:
```

Figure 13: condensed solution

```
PROC temp.solution (CHAN OF p.data.type Chanin,
                    CHAN OF p.data.type Chanout)

  CHAN OF INT c.1.2:
  data.type   dataA, dataB:
  PAR

    INT inchan.free, outchan.free:
    WHILE TRUE
      SEQ
        Chanin ? dataA
        PAR
          c.1.2 ! inchan.free   -- free input channel
          ...  process dataA
        Chanout ! dataA
        c.1.2 ! outchan.free    -- free output channel

    INT inchan.free, outchan.free:
    WHILE TRUE
      SEQ
        c.1.2 ? inchan.free     -- wait for input channel
        Chanin ? dataB
        PAR
          ...  process dataB
          c.1.2 ? outchan.free  -- wait for output channel
        Chanout ! dataB
:
```

Figure 14: temporary solution

```
PROC my.peel.solution (CHAN OF p.data.type Chanin,
                       CHAN OF p.data.type Chanout)

  data.type   dataA, dataB:

  SEQ
    Chanin ? dataA

    WHILE TRUE
      SEQ
        PAR
          SEQ
            ...  process dataA
            Chanout ! dataA

          SEQ
            Chanin ? dataB
            ...  process dataB

        PAR
          Chanin  ? dataA
          Chanout ! dataB
:
```

Figure 15: my Peel solution

```
PROC variant.protocol (CHAN OF p.variant Chanin, CHAN OF p.variant Chanout)

  data.typeA   dataAA, dataBA:
  data.typeB   dataAB, dataBB:
  ...
  data.typeN   dataAN, dataBN:
  BYTE flagA:
  BYTE flagB:

  SEQ
    {{{  input  A
    Chanin ? CASE
      tagA; dataAA
        flagA := 'A'
      tagB; dataAB
        flagA := 'B'
      ...
      tagN; dataAN
        flagA := 'N'
    }}}
    WHILE TRUE
      SEQ
        PAR
          {{{  input  B
          Chanin ? CASE
            tagA; dataBA
              flagB := 'A'
            tagB; dataBB
              flagB := 'B'
            ...
             tagN; dataBN
              flagB := 'N'
          }}}
          {{{  output A
          CASE flagA
            ('A')
              Chanout ! tagA; dataAA
            ('B')
              Chanout ! tagB; dataAB
            ...
            ('N')
              Chanout ! tagN; dataAN
          }}}
        PAR
          ...  input  A
          {{{  output B
          CASE flagB
            ('A')
              Chanout ! tagA; dataBA
            ('B')
              Chanout ! tagB; dataBB
            ...
             ('N')
              Chanout ! tagN; dataBN
          }}}
:
```

Figure 16: variant protocol

[27] R. Peel, *Issues raised while implementing a layered protocol using occam and the transputer*, in Proceedings of OUG technical meeting Nº 10, ed. André Bakkers, *Applying transputer based parallel machines*, IOS, 1989.

[28] INMOS Ltd, *Transputer instruction set – a compilers writers guide*, Prentice Hall, 1988.

P. A. Shallow piers@uk.co.mmarl
Marconi (MMARL)
Unit 33
Cambridge Science Park
Milton Road
Cambridge CB4 4FX
United Kingdom

TWO IMPLEMENTATIONS OF SEMAPHORES IN OCCAM

Geoff Barrett, Inmos Ltd, Bristol

This paper presents two implementations of semaphores in occam. The aim is to design an algorithm which has a non-busy and efficient implementation on a communicating process architecture.

Introduction

Semaphores and hand-shaking have come to be the two most common models of concurrency in theoretical circles. It has been shown that point to point handshaking communication has an efficient implementation [32]. It has also been known for some time that semaphores can be implemented in machines with only read and write instructions [29]. We set out to exhibit a more efficient implementation of semaphores.

Abstractly, we can think of a semaphore as a natural number variable which is incremented by a signal operation and, if positive, is decremented by a wait operation; otherwise, the process executing the wait operation is suspended until the variable is made positive again by a signal operation. A binary semaphore is one whose variable only ever takes on the values 0 and 1. Such semaphores were used in one of the first solutions to the problem of providing mutual exclusion. For more examples of the use and theory of semaphores see reference [31]. Semaphores are therefore a paradigm which is most useful on shared memory architectures. In particular, we shall be interested in an implementation for a single processor.

Notation

The language which is used to describe the algorithm is like occam except that we abuse the fact that the transputer implementation has some properties which are not required by the language definition. The extent of this is that we assume that channel communications achieve the desired synchronisation whenever only one process attempts to input or output at any one time; and we assume that a process can only interrupt another whenever the interrupting process is of higher priority or else the two processes are of the same priority and the interrupted process has reached the end of the body of a `WHILE`-loop.

We introduce some further constructs into the language as follows:

1. Type abbreviations are allowed with a self-explanatory syntax:

   ```
   TYPE name
     type
   ```

2. The class of types is augmented with records:

   ```
   RECORD
     { declaration }
   ```

3. There is a new type of lists which has syntax

   ```
   LIST OF type
   ```

   The empty list is denoted $\langle\,\rangle$, a singleton by $\langle x \rangle$ and catenation of lists by $\frown$.

4. To deal with lists we introduce a new control construct:

   ```
   MATCH exp
     branch
   ```

   where

   $$branch ::= \begin{array}{l} pattern \\ \quad process \end{array}$$

   Patterns are expressions with free variables and are matched by an expression if there is an assignment to the free variables of the pattern such that the pattern is equal to the expression. The free variables are available to the process as constants bound according to the match. For instance, the pattern $\langle c \rangle \frown q$ will match the list $\langle 1,2,3 \rangle$ binding c to 1 and q to $\langle 2,3 \rangle$. The construct

   ```
   MATCH ⟨1,2,3⟩
     ⟨⟩
       STOP
     ⟨c⟩ ⁀ q
       head, tail := c, q
   ```

   has the effect of assigning 1 to `head` and $\langle 2,3 \rangle$ to `tail`.

Only one priority

The simple case of the implementation is when there is only one level of process priority. In this case, we may assume that any `WHILE`-free segment of code can only be interrupted at a communication. This does require a slightly different implementation of the conditional construct from that recommended in the compiler writer's guide [30] so that no unconditional jumps are used.

The essence of the implementation is to maintain a queue of processes which are waiting for the semaphore. Because this queue cannot be maintained directly, we keep a list of channels over which the processes are waiting to communicate. The `WAIT` instruction will be implemented by decrementing the semaphore variable if it is positive or, if the semaphore variable is 0, appending a channel to the end of the semaphore queue and waiting on that channel. The `SIGNAL` instruction will increment

```
PROTOCOL UNIT
  CASE
    unit
:
TYPE SEM
  RECORD
    INT v
    LIST OF CHAN OF UNIT q
:
PROC INIT (SEM s, INT n)
  s.v, s.q := n, ⟨⟩
:
PROC WAIT (SEM s)
  IF
    s.v > 0
      s.v := s.v-1
    s.v = 0
      CHAN OF UNIT c
      SEQ
        s.q := s.q ⁀ ⟨c⟩
        c ! unit
:
PROC SIGNAL (SEM s)
  MATCH s.q
    ⟨⟩
      s.v := s.v+1
    ⟨c⟩ ⁀ q'
      SEQ
        s.q := q'
        c ? CASE unit
:
```

Figure 17: single priority interrupts

the semaphore variable if the queue is empty and otherwise remove the first element of the queue and complete the communication along the channel, thereby releasing a waiting process. The semaphore variable is initialised to the initial value of the semaphore and the queue is initially empty. The code appears in figure 17.

To verify the correctness of the implementation, we need the observation that if c is in the semaphore queue, then there is precisely one process waiting to output along the channel. The abstract behaviour of the semaphore (described in the introduction) considers the WAIT and SIGNAL operations to be atomic. We have given implementations which may be interrupted. To see that the implementation is correct, we define a map from the interrupt points of the implementation to processes of the specification. A SIGNAL process is only possibly interrupted when it comes to output along the channel from the queue. At this point, we map it to SKIP because the communication is guaranteed to terminate within some finite amount of time as there is a process waiting to output along the channel. A WAIT process can only be interrupted when it comes to output. This point is mapped to a WAIT process

```
TYPE SEM
  RECORD
    INT v
    LIST OF CHAN OF UNIT q
    SIM_SEM lo
    SIM_SEM hi
    BOOL lo_using
    CHAN OF UNIT hi_chan
:
```

Figure 18: type of a semaphore in the two-priority implementation

in the specification because the process still has to wait for some other process to increment the semaphore variable. This argument can be formalised to prove that the implementation is safe in that it does no actions which are not allowed by the specification.

We must still ensure that the implementation is live in the sense that it can perform enough of the actions which the specification requires. In other words, if there is a SIGNAL process of the specification which can make progress, then there must be one in the implementation which can, and similarly with the WAIT processes. The former assertion is obvious and the latter can be deduced from the fact there is a process waiting on each of the channels in the semaphore queue and otherwise the WAIT process proceeds as normal.

Two priorities

Next, we consider how to implement the semaphore when there are processes of two priorities. The implementation is complicated because we cannot assume that lower priority processes are only interrupted at communications. What we shall do is to combine two single priority sempahores to make a semaphore which works properly between processes of different priorities. Access to the semaphore state will be protected by a binary semaphore on each priority. This means that we only have to implement a semaphore which arbitrates between a single process of each priority. Since it is only the low priority process which may be interrupted, the problem is to exclude the high priority process from the semaphore state while the low priority process has access.

A smart solution

The way in which this is achieved is with a flag and a channel (figure 18). Whenever the low priority process has access to the semaphore state, the flag is set. If a high priority process requires access to the state and the flag is set, then it waits to output along the channel. Initially, the flag must be unset. In the code in figures 18 and 19, the semaphore and procedures from the single priority implementation are prefixed with SIM_.

We will prove a number of assertions which lead to the fact that at most one process is attempting to access the state of the semaphore at any one time (by state we mean the variable and queue). This is all that is required for a proof of correctness

```
PROC INIT (SEM s, INT n)
  SEQ
    SIM_INIT (s.lo, 1)
    SIM_INIT (s.hi, 1)
    s.v, s.q, s.lo_using :=
                n, ⟨⟩, FALSE
:

PROC HI_SIGNAL (SEM s)
  SEQ
    SIM_WAIT (s.hi)
    IF
      s.lo_using
        s.hi_chan !  unit
      NOT s.lo_using
        SKIP
    MATCH s.q
      ⟨⟩
        s.v := s.v+1
      ⟨c⟩ ⌢ q′
        SEQ
          s.q := q′
          c ?  CASE unit
    SIM_SIGNAL (s.hi)
:

PROC LO_SIGNAL (SEM s)
  SEQ
    SIM_WAIT (s.lo)
    s.lo_using := TRUE
    MATCH s.q
      ⟨⟩
        s.v := s.v+1
      ⟨c⟩ ⌢ q′
        SEQ
          s.q := q′
          c ?  CASE unit
    s.lo_using := FALSE
    PRI ALT
      s.hi_chan ?  CASE unit
        SKIP
      SKIP
        SKIP
    SIM_SIGNAL (s.lo)
:

PROC HI_WAIT (SEM s)
  CHAN OF UNIT c:
  BOOL waiting:
  SEQ
    SIM_WAIT (s.hi)
    IF
      s.lo_using
        s.hi_chan !  unit
      NOT s.lo_using
        SKIP
    IF
      s > 0
        waiting, s.v := FALSE, s-1
      s = 0
        waiting, s.q := TRUE, s.q ⌢ ⟨c⟩
    SIM_SIGNAL (s.hi)
    IF
      waiting
        c !  unit
      NOT waiting
        SKIP
:

PROC LO_WAIT (SEM s)
  SEQ
    SIM_WAIT (s.lo)
    s.lo_using := TRUE
    IF
      s ≠ 0
        waiting, s.v := FALSE, s-1
      s = 0
        waiting, s.q := TRUE, s.q ⌢ ⟨c⟩
    s.lo_using := FALSE
    PRI ALT
      s.hi_chan ?  CASE unit
        SKIP
      SKIP
        SKIP
    SIM_SIGNAL (s.hi)
    IF
      waiting
        c !  unit
      NOT waiting
        SKIP
:
```

Figure 19: two priority implementation

for the new implementation is essentially the same as the old but with some extra scheduling control.

1. Only one low priority process has access to the semaphore state, the *lo_using* variable and the output end of the *hi_chan* channel at any one time. No low priority process has access to the input end of the *hi_chan* channel.
   (Use of low semaphore.)
2. Only one high priority process has access to the semaphore state and the input end of the *hi_chan* channel at any one time. No high priority process has access to the output end of the *hi_chan* channel nor assigns to the *lo_using* variable at any one time.
   (Use of high semaphore.)
3. If *hi_chan* is ready for output, then *lo_using* is `FALSE`.
   (The only process which may make *hi_chan* ready for output is the low priority process which has just assigned `FALSE` to *lo_using*.)
4. When *hi_chan* becomes enabled, *lo_using* is `TRUE`.
   (The only process which may enable the channel is the high priority process which has just tested *lo_using*; the process cannot be interrupted during this sequence of actions.)
5. If a low priority process attempts to access the semaphore state then *lo_using* is `TRUE`.
   (Only low priority processes set it, by part 1 only one of them has access at a time, and then it is obvious from the way in which the assignments bracket the access.)
6. If a high priority process attempts to access the semaphore state then *lo_using* is `FALSE`.
   (Either *lo_using* was `FALSE` at commencement of the test in which case the process cannot be interrupted during its access to the semaphore state or else *lo_using* was `TRUE` but as soon as *hi_chan* becomes ready, *lo_using* is `FALSE` and the process is rescheduled and cannot be interrupted during access.)

Parts 5 and 6 together mean that at most one process has access to the semaphore state at any one time. We need only verify that once a high priority process suspends on *hi_chan* and the low priority processes are not starved by high priority actions, it eventually receives a communication. This is immediate since the high priority process will only suspend on the channel when some low priority process is accessing the semaphore state and will, if not starved, eventually communicate along *hi_chan*.

An easy solution

The easy way in which to ensure that a process is atomic is to give that process the higher priority. So that, for instance, `LO_SIGNAL` would be

```
PRI PAR
  SIM_SIGNAL (s)
  SKIP
```

Comparisons

There are three sorts of comparisons which can be made between the two implementations. The first is in terms of efficiency. The second is in the way in which the algorithms treat processes which are waiting to access the semaphore. The third is the effect on the environment.

Efficiency

Both methods have a common subset of actions which are necessary in each access. We study the extra actions needed otherwise. In the less smart algorithm, there is the cost of spawning a high priority process for each low priority access and that is all. For the smart algorithm, the overheads vary depending on whether other processes of the same or different priority are attempting to access the semaphore. In the best case, when no other process is attempting to access the semaphore, there are four extra assignments and three extra tests for a low priority process, and two extra assignments and three extra tests for a high priority process with an extra assignment and test in both cases if the operation is wait. In the worst case, there are, on top of the overheads already mentioned, two communications (we count only one of the communications for the `SIM_WAIT` and `SIM_SIGNAL` because otherwise each communication would be counted twice).

Since it rare that two processes attempt to access a semaphore simultaneously, the smarter algorithm is usually more efficient.

Queuing

Suppose there is a set of processes waiting to access a semaphore. Both implementations choose arbitrarily between high priority processes and if there are none, between low priority processes. High priority processes always overtake low priority processes. The queue of processes waiting to acquire the semaphore is always dealt with in a first accessed first served fashion.

With regard to the effects on the transputer scheduler queue, the less smart implementation causes each low priority process to reschedule on the queue thereby reducing its effective time-slice period and increasing the chance of contention for the semaphore.

Environment

The smart implementation has the advantage that access to semaphores by low priority processes does not affect the interrupt latency of the whole program. Reasoning about real time programs is very much simplified by only having to consider high priority processes when calculating interrupt response times.

Conclusions

The reasoning employed to show the correctness of the algorithms needs intimate knowledge of the scheduling mechanism of the underlying process model. This leads to a very intricate and complicated proof. The provision of semaphores and the bundling of synchronisation into hand-shaking communication gives a much more suitable level of abstraction in which to reason about such systems.

However, the fact that it is possible to provide semaphores on a single machine allows a more efficient implementation of some forms of replicated arbitrations. For instance, if the only use for input of the array of channels c is in:

```
ALT i= 0 FOR SIZE c
  c[i] ?  x[i]
    P(i)
```

Then the array of channels may be collapsed to a single channel and a binary semaphore thus the channel declaration becomes:

```
CHAN OF IPROT c:
SEM s:
SEQ
  INIT(s,1)
  P
```

where IPROT is the protocol INT; PROT if PROT is the protocol of c. The arbitration becomes:

```
c ?  i; x[i]
  P(i)
```

and output c[i] ! e is replaced by:

```
SEQ
  WAIT (s)
  c !  i; e
  SIGNAL (s)
```

Note also that the implementation is fair because the output processes are queued as they become ready so that none are infinitely overtaken. Another variation on the same theme is the 'first come, first served marriage bureau' in which each end of the channel is protected with a binary sempahore.

Acknowledgements

Thanks to Roger Shepherd, Geraint Jones and Tony Fisher for comments on earlier drafts of the paper.

References

[29] E. W. Dijkstra, *Solution of a problem in concurrent programming control*, Commun. ACM, Vol. 8, No. 9, Sep. 1965.

[30] Inmos Ltd, *Transputer instruction set: a compiler writer's guide*, Prentice-Hall, Hemel Hempstead, 1988.

[31] A. Martin and J. L. A. van der Snepscheut, *Design of concurrent programs*, in Proc. International Summer School on *Constructive methods in computing science*, Marktoberdorff, 1988.

[32] D. May and R. Shepherd, *The transputer implementation of occam*, Inmos Ltd, Bristol, 1987.

Geoff Barrett
Inmos Limited
1000 Aztec West
Almondsbury
Bristol BS12 4SQ
United Kingdom

geoffb@uk.co.inmos

MEASURING THE BUSYNESS OF A TRANSPUTER

Geraint Jones, with contributions from Andy Rabagliati,
Klaus Zeppenfeld, Michael Goldsmith and others

Something which transputer users often want to do, it seems, is to measure how much time a processor spends not doing anything. On the face of it, this is quite hard, because one of the things hidden by the process abstraction of occam is the way that a processor is shared by processes.

Some time ago, Andy Rabagliati from Inmos' American Central Applications posted to the 'transputer' electronic mailing list a code fragment for making just this measurement. It used knowledge about the way the Inmos occam compiler behaved, and the way the transputer scheduler behaved, in order to estimate how often the processor had nothing to do.

Since I have never seen this trick in print, and since a number of people have recently asked me for code to perform just this sort of measurement, I am writing it up here. I had to struggle a bit to understand how it worked, and I have modified the code a bit in the hope of making it easier to understand.

Interface

The code consists of a procedure *supervisor* which must be run, at low priority, in parallel with the code of the program being measured. It measures how busy the subject program would have kept the processor, by counting how often it finds the transputer idle, and returns and integer value representing the percentage utilization of the transputer's processor.

There are two phases: a calibration phase in which the *supervisor* measures how fast the scheduling mechanism is – a function of processor speed and whether it is running in internal or external store. In order to work, this *must* happen when all other processes are idle because they are waiting for communications or timers. When the *supervisor* starts, it waits for a *supervisor.calibrate* signal on its *command* channel, indicating that it can start calibration. When the calibration is over – after about a millisecond or two – it acknowledges this by a second communication on the same channel. This indicates that it is safe for other processors to be run without compromising the calibration.

To start a measurement, the *supervisor* is sent a *supervisor.start* command. Thereafter, sending a *supervisor.read* command causes the utilization since the most recent *supervisor.start* to be calculated and returned on *return*.

The *supervisor* terminates when sent a *supervisor.stop* signal over the *command* channel.

```
PAR
  supervisor(command, return)
  SEQ
    command ! supervisor.calibrate
    command ! supervisor.ack
    ...
    command ! supervisor.start
    ... subject code
    command ! supervisor.read
    INT busyness :
    SEQ
      return ? busyness
      Write("The processor was busy ", busyness, "% of the time")
    ...
    command ! supervisor.stop
```

Figure 20: Example of a one-shot use of *supervisor*

Algorithm

The *supervisor* works by counting the number of times it finds no processes in the low-priority process queue of the transputer. The transputer scheduler maintains a queue of processes that are not blocked waiting for a communication or for a timer: these are 'ready' to be run. Leave aside for the moment the possibility that there are high-priority processes. While there is a ready low-priority process, the processor executes one of these ready processes. Periodically the currently executing process is descheduled and placed at the back of the ready queue, to the replaced by a process removed from the head of the queue.

When the *supervisor* is running, it is always ready. This means that – apart from any disturbance to the execution of the program caused by the supervisor – the transputer would have been busy running the rest of the program, without the supervisor, if and only if there is something else in the ready queue when the supervisor is being executed. When it is executed, the supervisor process makes a note of the time and deschedules itself, placing itself at the back of the ready queue.

The next time this process comes to the head of the queue and is executed, it compares the present time with its last noted time. If these times are sufficiently close, the assumption is that the ready queue was empty – the processor would have been idle, were it not for the supervisor – and that the supervisor was resumed as soon as it had been descheduled. The proportion of idle – or conversely busy – time is obtained by comparing the number of times the queue is found to be idle in a given period with the number of times it was found to be idle in the calibration period.

The mechanism is able to cope reasonably well with high-priority processes. Whenever there is a ready high-priority process it is executed, and no low-priority activity takes place. Time taken by high-priority process in the subject program is therefore effectively lost to the *supervisor*, and so is measured as though it had been consumed by some additional ready low-priority processes.

Implementation

The *supervisor* is implemented by a pair of parallel processes (figure 21). One of them handles the communication with the user code, and calculates the result (figure 22). The other process is the one that counts the number of times that it finds the processor idle (figure 23). There are a number of 'tricks' in the implementation of this idle counter, code which is not strictly speaking occam since it is not guaranteed to work by the language definition. Of course, there have to be such tricks, because the behaviour of an occam process is meant to be unchanged by sharing a processor with another process.

The first problem is that of implementing the descheduling operation. It is necessary to run the body of a loop, P say, and to ensure that after P has terminated any process in the ready queue is given a chance to execute before the next iteration of the loop. In the code given here, this is achieved by writing

```
WHILE condition
  PAR
    SKIP
    P
```

So long as the compiler does not optimise the code by omitting the SKIP, this will work. Since the loop is executing on a single transputer, both the branch of the parallel which executes the P, and the branch executing SKIP have to be executed on the same processor. Consequently, each time the loop body is executed two processes are ready and at least one of them has to be put at the back of the process queue. Every process in the queue has therefore to be executed before the parallel in the loop body can terminate, and this must happen before the next iteration of the loop can start.

The second problem is that, although the *supervisor* must run at low priority, it must have access to a timer with adequate resolution to tell whether two executions of the loop body are really not separated by a few instructions from another process. In the case of the transputer, this means being able to read the high-priority timer, accessible only to high-priority processes. The trick here is an idiom:

```
PRI PAR
  P
  SKIP
```

which executes P as a high-priority process, whatever its context. It is not at all plain to me (although I know that others disagree with me) what the language definition says PRI PAR is meant to mean, except where it is the outermost PAR of a program, or the outermost PAR of a branch of a PLACED PAR. Be that as it may, it happens that this trick works with the Inmos occam compiler for the transputer.

The parameter *threshold* should be just large enough that one execution of the body of the loop in the counter process can be executed in that number of ticks of the high-priority clock. Thus, whenever the clock has moved on by more than this value it can be deduced that some other code was run between iterations of the loop. I see no reason to suppose that the same value of *threshold* would work for any member of the transputer family, so it might be that it is necessary to resort to studying the data-sheet for a particular transputer, and the instructions that the

```
PROTOCOL SUPERVISOR.COMMAND
  CASE
    supervisor.calibrate -- first command
    supervisor.ack       -- acknowledges end of calibration
    supervisor.start     -- starts a measurement
    supervisor.read      -- returns percentage idle time
                         --     since the last supervisor.start
    supervisor.stop      -- last command: terminates SUPERVISOR
:

PROC SUPERVISOR(CHAN OF SUPERVISOR.COMMAND command, CHAN OF INT return)

  VAL period IS 25 :    -- number of low pri clock ticks in calibration
  VAL threshold IS 20 : -- just enough high pri ticks to reschedule

  PROTOCOL IDLE.COUNTER.COMMAND
    CASE
      idle.counter.start
      idle.counter.read
      idle.counter.stop
  :

  CHAN OF IDLE.COUNTER.COMMAND calculator.to.counter :
  CHAN OF INT counter.to.calculator :

  PROC idle.counter
    ...
  :

  PROC cpu.load.calculator
    ...
  :

  PAR
    cpu.load.calculator
    idle.counter
:
```

Figure 21: code skeleton of the complete supervisor

```
PROC cpu.load.calculator

  TIMER low.pri.timer :
  INT idle.loops.per.period :
  SEQ
    command ? CASE supervisor.calibrate
    INT now :
    SEQ
      calculator.to.counter ! idle.counter.start
      low.pri.timer ? now
      low.pri.timer ? AFTER now PLUS period
      calculator.to.counter ! idle.counter.read
      counter.to.calculator ? idle.loops.per.period
    command ? CASE supervisor.ack

    BOOL cpu.load.calculator.needed :
    SEQ
      cpu.load.calculator.needed := TRUE
      INT start.time :
      WHILE cpu.load.calculator.needed
        command ? CASE

          supervisor.start
            SEQ
              calculator.to.counter ! idle.counter.start
              low.pri.timer ? start.time

          supervisor.read
            INT read.time, actual.count :
            SEQ
              calculator.to.counter ! idle.counter.read
              counter.to.calculator ? actual.count
              low.pri.timer ? read.time
              VAL elapsed.time IS read.time MINUS start.time :
              VAL maximum.count IS
                      (idle.loops.per.period * elapsed.time) / period :
              CASE maximum.count
                0             -- elapsed time below resolution
                  return ! 0  -- return arbitrary result
                ELSE
                  return ! ((maximum.count - actual.count) * 100) /
                                                          maximum.count
          supervisor.stop
            SEQ
              calculator.to.counter ! idle.counter.stop
              cpu.load.calculator.needed := FALSE
:
```

Figure 22: the process that controls and calculates results

```
PROC idle.counter

  BOOL idle.counter.needed :
  SEQ
    idle.counter.needed := TRUE

    INT idle.count, now, then :
    WHILE idle.counter.needed
      PAR
        SKIP     -- dummy process to force rescheduling
        SEQ
          PRI PAR
            TIMER high.pri.timer :
            high.pri.timer ? now
            SKIP -- dummy low priority process
          IF
            (now MINUS then) < threshold
              idle.count := idle.count PLUS 1
            TRUE
              SKIP
          then := now
          PRI ALT

            calculator.to.counter ? CASE
              idle.counter.start
                idle.count := 0
              idle.counter.read
                counter.to.calculator ! idle.count
              idle.counter.stop
                idle.counter.needed := FALSE

            TRUE & SKIP
              SKIP
:
```

Figure 23: the process that detects idleness of the processor

compiler generates for the body of the loop, in order to determine an appropriate value for any processor. The value of twenty is probably a little high for a T800.

Each execution of the body of the loop in the counter process polls, with the construct

```
PRI ALT
  calculator.to.counter ? CASE
    ...
  TRUE & SKIP
    SKIP
```

for a command from its controlling calculator process. If there is no command pending, it goes on to reschedule itself at the back of the queue. This loop is 'busy': it is always ready to be executed by the processor, so it appears at first sight to be consuming valuable processor time. Notice however that only one execution of the loop body can happen for every time slice of every other ready process in the program. This means that whenever the program has anything to do, very little time is consumed by the *supervisor*. The time overhead of running this code has been measured in practice to be of the order of one percent.

Warnings

Although the code in the *supervisor* procedure 'looks and feels' like occam, this is misleading. Correct execution of the *supervisor* relies on details of the implementation of both the compiler and the transputer outside what is guaranteed by the semantics of the language. For example, in occam is is the case that

```
PAR
  P
  SKIP
```

is the same as the process P, but the correct execution of the *supervisor* relies on the compiler not optimising the former by compiling code only for the latter.

The *supervisor*, as written, cannot be used to measure performance over periods longer than about two thousand seconds; otherwise there will be an arithmetic overflow (assuming thirty-two bit INTs). The calculation of the return value could be done in floating point, if necessary, but eventually the problem arises of representing an *idle.count* bigger than the biggest integer.

In the code given here, *idle.count* is incremented by the cyclic addition, PLUS, to avoid causing an arithmetic overflow if the *supervisor* runs for a very long time. I make no attempt to check that the count has not exceeded the largest representable integer. This means that it is safe to leave a *supervisor* running indefinitely, although its calculated result can only be trusted for start/stop pairs that are not too long apart.

Similarly, in a program with a vast number of ready processes it might be that the high-priority clock would count through half its range of ticks in the time taken to schedule and execute a time slice for each ready processes. The idle loop counter would mistakenly count this as an idle cycle, because *now* MINUS *then* would be negative and so less than *threshold*. Because this would happen very infrequently – no oftener than half the clock's maximum cycle time – it is unlikely to make a significant difference to the accuracy of the calculated result.

Acknowledgement

The idea of a process that continually put itself on the back of the low-priority queue came from Andy Rabagliati, at INMOS Colorado Springs in 1987; the clean separation into a pair of process communicating over channels was suggested to me independently by Michael Goldsmith and Klaus Zeppenfeld; and the experiments were done for me by Klaus, who also explained to me what the real transputer scheduler does. (I really did not understand Andy's code when I first saw it.) Any failure to understand which might manifest itself in errors in the program or the article is of course entirely my own.

Geraint Jones Tel: +44 865 273851
Programming Research Group geraint.jones@uk.ac.oxford.prg
11 Keble Road
Oxford OX1 3QD
United Kingdom

IMPROVEMENTS(?) TO OCCAM

G. A. Wilson, Department of Computer Science, Sheffield University

In this short note we show how the functionality of the occam IF construct can easily be achieved with the more general ALT construct, and suggest that this encourages a more parallel mode of thinking.

Additionally, we suggest that extending abbreviations to datatypes and literals will save much laborious typing, and also provides a simple named type facility.

ALT instead of IF (and CASE)

Remove CASE constructs from the language, by transforming into the equivalent IF:

```
CASE e            IF
  v1                e = v1
    p1                p1
  v2        ⇒       e = v2
    p2                p2
  ELSE              TRUE
    p3                p3
```

The degenerate cases (i.e. with no conditions at all) are both the same (the process behaves like STOP).

Remove IF constructs from the language, by transforming into a PRI ALT:

```
IF                PRI ALT
  b1                b1 & SKIP
    p1                p1
  b2        ⇒       b2 & SKIP
    p2                p2
  TRUE              TRUE & SKIP
    p3                p3
```

Again, the degenerate `PRI ALT` behaves like `STOP`. To save typing, the extra `& SKIP` component can be made implicit by the compiler, in a similar manner to that currently implemented for `TRUE &`, i.e. 'Guards which do not have a boolean conjunct to them have `TRUE &` added' [33].

EFFICIENCY need not be compromised, since the compiler can be extended to recognise `IF`-equivalent or `CASE`-equivalent constructs, and efficiently compile them as before.

OBJECTIONS? One point raised by a member of our group is 'when I see an `IF`, I do not need to enter the (folded) body of the construct to determine if there may be any communication with other concurrent processes'. True, but does this matter?

Extending abbreviations ...

... TO DATATYPES. Implicit abbreviations currently exist for integers. Making these explicit might help alleviate the problems encountered when creating programs of mixed T2/T8 processors, where only `INT` types have been specified. A suggested syntax is `INT IS INT32` for the T8, and `INT IS INT16` for the T2.

... TO LITERALS. A string of digits is currently implicitly of type `INT`. The abbreviations `REAL IS REAL32`, or `REAL IS REAL64` might similarly introduce a default type for any numeric literal of 'real' form (e.g. `1.2`, `3.E4` etc.), and enable us to avoid the infuriating `1.(REAL32)` syntax.

... AS SIMPLE USER-DEFINED TYPES. An article by this author [34] shows some implementations of a *Set* datatype using bits within integers. Usage would be much clearer and more convenient if the implementation of the set could be hidden by an abbreviation e.g. `SET IS INT32`. Note that such an abbreviation might be implemented in two ways:

simple – the textual instances of each new type are simply substituted by the text of the abbreviation;

complex – each abbreviation is a new datatype, and variables of the new type are not compatible with those of the abbreviated type.

References

[33] Section 6.10.3 of: Inmos Ltd, *Transputer instruction set – a compiler writer's guide*, Prentice Hall, 1988.

[34] G. A. Wilson, *Comparative timings of three set implementations in occam*, Software: Practice and Experience, 19(3), 273-281, March 1989.

G. A. Wilson
Department of Computer Science
Sheffield University
Sheffield S10 2TN
United Kingdom

ac1gaw@uk.ac.sheffield.primea

TRANSPUTER AND OCCAM BASED CONTROL SYSTEMS IN ELECTRONIC CONTROL OF MACHINES

Martin Törngren, The Royal Institute of Technology, Stockholm

This paper is a partial translation of the original paper *Transputern och Occam, applikationer i Datorstyrd Mekanik*, written in Swedish, which discusses the use of the transputer and occam in electronic control of machines.

The transputer is considered as an interesting component because of the following facts:

- ▷ Multi-transputer systems are easily built and programmed.
- ▷ The transputer concept is a step towards integration of hardware, operating system and software.

In this paper aspects on using the transputer and occam in a real-time control system are presented. The first part describes limitations in the transputer design which either makes the design of a transputer system difficult, such as few priority levels; or may introduce non-deterministic behaviour into the system, such as instructions with a variable execution time. The second part describes transputer I/O interfacing. The reader is assumed to have a basic knowledge of the transputer and its scheduler.

Real-time properties – transputer based control systems

A control system generally has to handle both periodic and aperiodic tasks. A control system controlling the motion of a mechanical structure has to behave deterministically, i.e. it has to solve the periodic tasks and at the same time guarantee that the response times for some important aperiodic tasks are fulfilled. This should be done with a high processor utilization. To be able to construct such a control system a detailed knowledge of its components (hardware, operating system and software) and their characteristics concerning response times, time measurement and scheduling algorithm is needed.

When constructing a transputer based control system two solutions are available:

Use of a bare transputer. This requires knowledge of the transputer scheduler and the transputer hardware. The limitations in the transputer design presented below has to be dealt with.

Use of a real-time operating system executing on the transputer. This real-time operating system could be coded as a high priority transputer process and control application processes working at low priority level, possibly manipulating with process queues. The real-time operating system should use scheduling algorithms more suited for real-time applications than the algorithm used by the transputer scheduler.

Two priority levels – a clear limitation in a control system

With few priority levels the response times are dependent on other processes' execution times. The following example of a system consisting of the following processes demonstrates this:

- ▷ Process 1 (for example a control algorithm), executing periodically at 1 kHz.

▷ Process 2 (for example data sampling), executing periodically at 10 kHz.

▷ Process 3 (for example emergency stop), an aperiodic process.

With three priority levels the assignment of priorities to processes would be simple: process 1 – lowest priority; process 2 – medium priority; and process 3 – highest priority. The response time for process 3 would not be dependent on the execution time of the other processes.

With only two priority levels the assignment of priorities to processes is more difficult: process 1 and process 2 cannot both be given low priority if the execution time of process 1 exceeds 100 μs. If process 2 and process 3 are assigned high priority the response time for process 3 will be dependent of the execution time of process 2. Code changes in process 2 which increases the execution time might result in an unacceptable response time for process 3.

The response time for a low priority process is nondeterministic

The response time for a process with low priority in a program with n low priority processes and no high priority processes is given in reference [35] to be $2 \times T_{slice} \times (n-1)$, where T_{slice} is the fixed time slice period that the scheduler uses. If a low priority process has executed for this time the scheduler will deschedule it. This can however only be done at certain instructions, namely the following: timer input, channel communication (internal end external) and the *jp* and *lend* (loop end) instructions [36].

The actual response time is $2 \times T_{slice} \times (n-1) + T_?$, where $T_?$ is the time it takes the process to get to the next descheduling point. Most program contains lots of descheduling points. But you cannot be sure of that, for example when using an optimizing compiler.

The response time for a process with high priority is determinable but is dependent of the execution times of other high priority processes. If there is only one process with high priority the response time has a determinable upper bound dependent of the transputer model, execution-frequency and use of on-chip RAM. On a T414-20 transputer this time is about 3 μs. Let this time be denoted by $T_{schedule}$ and the execution time for process i by $T_{exec(i)}$. The worst case response time for process $(i+1)$ in a system with two high priority processes becomes $2 \times T_{schedule} + T_{exec(i)}$.

Priority inversion

The priority inversion problem arises when a high priority process wants to perform synchronized communication with a low priority process. It may then become blocked if the low priority process is not ready for communication meaning that another low priority process can be allowed to execute. As the high priority process is blocked and a low priority process is executing this phenomenon is referred to as priority inversion. The priority inversion problem can be avoided by using a dynamic priority scheduling algorithm [38]. In reference [37] modifications to the transputer's scheduler to incorporate this algorithm are presented. Priority inversion is often solved by inserting high priority occam processes as buffers. This alternative however demands verification every time the system is changed and also increases the response time for other high priority processes.

The unfairness of the ALT construction

The `ALT` construction is, in current occam compilers, implemented in the same way as the `PRI ALT`. Both `ALT` constructs can cause starvation for processes waiting for communication. In the `PRI ALT` the programmer explicitly assigns priorities to alternatives. One could argue that the scheduler should take into account the priority of the communicating processes. In a real-time system the `PRI ALT` can be used, but has to be carefully designed to prevent starvation. In a less time-critical system the same degree of care has to be exercised with pure `ALT` constructs.

Execution time of the ALT and timer input instructions

Execution of an `ALT` in a process P_A involves the following steps: All components of the `ALT` are enabled. Process P_A is descheduled. When P_A starts executing again, the components of the `ALT` are polled to detect which component is ready for communication. This means that both the setup time and the polling after scheduling are dependent of the number of components in the `ALT` and lead to variable execution time.

When a process executes a timer input instruction the process can be descheduled. An example in occam code (declarations are omitted):

```
clock ? time                      -- Reading actual time
clock ? AFTER time PLUS period    -- Delaying the process =>
                                  -- descheduling of the process
```

Descheduling means sorting the process onto the the appropriate timer queue. This sorting obviously depends on the number of processes already waiting on that timer queue and thus the execution time of the *timer input* instruction is variable.

Execution of a program on one or several transputers

One of the major advantages of the transputer is the ability to develop a program for a multi-transputer system on one transputer. Any external hardware and external systems can simulated as software processes. The program can then be moved to a transputer network by adding configuration statements. The question is, does the program behave in the same way?

In a real-time application there are two sources for different behaviour. The first is the true parallelism inherent in executing a program on more than one transputer. Even though the synchronization points (channel communication is synchronized) between processes will be the same, and so the synchronization graph of the program still will be the same, the execution of processes will now have other timing relations with each other. This is important when dealing with the outer world as in the case of control systems.

The second is the implementation of the `ALT` construct. As described above the `ALT` consists of a number of transputer assembler instructions. Firstly each component of the `ALT` is enabled and then the `ALT` process is descheduled. Secondly

```
PRI ALT
  c2 ?
    kod2
  c1 ?
    kod1
  c0 ?
    kod0
```

Figure 24: process P_A

```
PAR
  P0
  P1
  PA
  P2
```

Figure 25: one transputer

```
PAR
  P0
  P1
  PA
```

Figure 26: first transputer

```
P2
```

Figure 27: second transputer

when the process is scheduled it searches through its channel components to see which one is ready for communication. The possibility of different behaviour follows from the fact that channel communication between processes on different transputers can be done in parallel with execution because the links work in parallel with the processors. In a one processor system internal channel communication is sequential with execution.

The following example shows different program behaviour, due to the ALT construct, when one process is moved to another transputer. The example program has four processes: P_0, P_1 and P_2 are processes that each have one channel, called c_0, c_1, and c_2 respectively, to the fourth process called P_A (figure 24), which contains a PRI ALT construction with three components. All four processes are supposed to be executed repeatedly, perhaps in a WHILE loop.

Firstly the program is executed on one transputer (figure 25) and secondly on two transputers (figures 26 and 27) where process P_2 is moved to the other transputer. The following details a possible execution scenario of the program. Executing the program on one transputer:

- ▷ P_0 executes and is descheduled when it starts communication with P_A,
- ▷ P_1 executes and is descheduled when it starts communication with P_A,
- ▷ P_A executes its PRI ALT instruction so kod_1 will execute,
- ▷ P_A executes its PRI ALT instruction so kod_0 will execute,
- ▷ P_2 executes and is descheduled when it starts communication with P_A,
- ▷ P_0 executes and is descheduled when it starts communication with P_A,
- ▷ P_1 executes and is descheduled when it starts communication with P_A,
- ▷ P_A executes its PRI ALT instruction so kod_2 will execute,

and so on. Executing the program on two transputers (P_2 is moved to another transputer):

- ▷ P_0 executes and is descheduled when it starts communication with P_A,
- ▷ P_2 executes in parallel with P_0 and is descheduled when it starts communication with P_A,
- ▷ P_1 executes and is descheduled when it starts communication with P_A,

- ▷ P_A executes its `PRI ALT` instruction so kod_2 will execute. P_2 starts executing and is descheduled when it starts communication with P_A,
- ▷ P_A executes its `PRI ALT` instruction so kod_1 or kod_2, if P_2 and the link already has performed the communication, will execute,
- ▷ P_A executes its `PRI ALT` instruction so kod_0, kod_1 or kod_2 will execute.

I/O interfacing

Very compact transputer 'calculation modules' can be constructed but the transputer lacks I/O for applications in electronic control of machines. Interfacing can be constructed by using the memory interface or the link-adaptor circuit.

MEMORY INTERFACE Fast I/O is possible – four bytes transferred per three processor cycles, provided zero wait-state memory is used. (On the T801 only two cycles are needed.) The speed is not a continuous speed because the memory interface is also used for instruction fetching. Communication with I/O is asynchronous.

LINK ADAPTOR Slower peak speed for I/O transfers than with the memory interface. The link speed of 20 Mbits/s implies about 1·74 Mbytes/s (one way communication) which can be sustained through a continuous transfer made by the link unit. Four bytes can be transferred in about 2 μs. Interfacing fits very well into the programming model. Communication with I/O is synchronous.

An appealing alternative that reduces the number of chips in the system is to use something like a microcontroller as an I/O processor. An even better alternative would be the appearance of a new transputer processor, a 'transputer embedded controller' containing such I/O as ADC and digital ports on chip. After all, reference [35] describes the transputer architecture as consisting of a core and an application specific interface. The M212 is so far the only transputer to contain an application specific interface on chip.

References

[35] Inmos Limited, *Transputer reference manual*, 1987.

[36] Inmos Limited, *The transputer instruction set – a compilers writers guide*, 1987.

[37] A. Burns and A. J. Wellings, *Occam's priority model and deadline scheduling*, in OUG-7, ed. Traian Muntean, *Parallel programming of transputer based machines*, IOS, 1987.

[38] Lui Sha, Ragunathan Rajkumar, John P. Lehoczky, *Priority inheritance protocols: an approach to real-time synchronization*, 1987.

Martin Törngren
Damek Research Group
Department of Machine Elements
The Royal Institute of Technology
100 44 Stockholm
Sweden

Tel: +46-8-790 7849
Fax: +46-8-723 1730
martin@se.kth.damek

GETTING ALONG WITHOUT WORKSTATIONS

Paul Healy, Computer Vision Group, Trinity College Dublin

We describe methodologies developed during the evolution of a parallel Virtual Image System running on a network of transputers, hosted by a PC compatible computer.

The environment

Although a PC hosted transputer development system is probably the most common variety found in research and industry, it is often looked down on as in some way less capable or powerful, when compared with a Unix box such as a Sun workstation. The purpose of this note is to highlight some of the more useful techniques and tools that we utilised during almost two years working with our transputer vision system VIS [39, 40]. While principally directed at users of the 3L Parallel C compiler, many points have a wider applicability, and a large number can be applied directly to one variety of occam program development using TDS. The commercially available products that we make use of include: the 3L Parallel C compiler, TDS, the Brief text editor, the Microsoft 80x86 C compiler (MSC) and finally the DESQview multitasking system from Quarterdeck.

Program maintenance

Although Brief has facilities to allow you compile a file without exiting to an MSDOS shell, we found it beneficial to produce a makefile. The MSC make utility is used to compile and link our transputer vision system VIS. The batch file which starts the make also preloads the compiler, saving repeated loads if there are many files which need to be recompiled. This batch file checks for a successful make, and can optionally change directory, load our vision system or restore the last edit session depending on the result. The compiling and linking process is done on a RAM-disk to speed up the process by up to thirty percent. Finally in order to maximise general file i/o to the transputer, a file cache facility is used.

Accessing host facilities from the transputer

We have extended the alien file server protocol [41] in a number of directions. The first facility, has been found useful to monitor the activity of an apparently dead program in the moments leading up to its locking up. This takes the form of a debug option on the afserver command line, which when enabled causes information from strategic areas of the alien file server to be echoed to the user. This has proved invaluable in writing transputer programs to control PC hosted frame grabbers. An optimised fast console put string command has resulted in a speed increase of about six, as compared with the standard transputer C output library routine. Routines to implement non-buffered input from the console and serial port access have also been used successfully. We are at present using a facility to provide an image display with sixty-four grey levels on a colour or monochrome VGA monitor. A fast file transfer utility, from host to host, using transputer links is also in use. The command to send a group of files down link 1 is simply '`tsend 1 *.*`'. Our occam programming is by necessity done to conform to the the alien file server arrangement. Very simply, we need to have direct access to the PC hardware, something that TDS on its own does not give us. As a fortunate byproduct of this state, we have a common platform on which to build resources which can be used by either C or occam. Then finally we now have windows.

Windows and multitasking with DESQview

We have recently added three new commands to the alien file server. We can now open, write to and close DESQview windows [42, 43] while running transputer VIS from within the DESQview environment. This support is available from the transputer while running C, or while using occam TDS programs written to use the alien file server protocol. The C function to open a DESQview window is:

```
int new_window(name, length, depth, x, y)
char *name;
int length, depth, x, y;
```

This function opens a window called 'name', with dimensions 'length' and 'depth' and at screen position 'x', 'y'. It returns a handle to the window which is used on future operations to the window. The C function to write to a window is:

```
void win_puts(win, str)
int win;
char *str;
```

This function writes the character string 'str' to the window 'win', where 'win' is the handle returned by the new_window function.

Finally the C function to close a window is:

```
void win_free(win)
int win;
```

This function closes the window 'win', where 'win' is the handle returned by the new_window function.

This facility is used to allow multiple processes access to screen resources in a relatively sane manner. Overlapping windows behave as expected, with underlying windows being automatically restored when occluding windows are closed. A debug facility allows the traffic though windows to be written to files for later inspection. Each item written to a file is time-stamped, in order to make comparisons of progress through a program. An equivalent set of routines has been written, and used successfully for the occam TDS, when using the alien file server protocol. We have also used the DESQview facility for screen management to switch between a program text screen and a sixty-four grey level VGA graphics screen with a resolution of 320×200.

A very obvious manner to scale our system lies in the ability of transputer development boards to reside at different addresses on the PC bus. The Inmos B008 TRAM motherboard can for example be used at three different addresses. Up to three alien file servers can be run in different windows to provide a very simple manner of run time debugging.

We intend to investigate other simple usages for the DESQview multitasking resources. This includes running transputer applications such as a make or VIS in the background, while continuing to edit in the foreground.

We are also looking at the feasibility of using DESQview's mouse input facilities to provide input to VIS, and also a way for the user to select and move transputer created windows.

Conclusion

Some very innovative software tools are available for the PC, which can be used with a little effort to expand the capabilities of transputer based software.

References

[39] D. Vernon and G. Sandini, *VIS: a virtual image system for image-understanding research*, Software: Practice and Experience, Vol. 18(5), 395–414, May 1988.

[40] P. Healy and D. Vernon, *Very coarse granularity parallelism: implementing 3-D vision with transputers*, in Proceedings of Image Processing '88, London, 1988.

[41] 3L Ltd, *File service protocol definition*, 3L Technical Note Nº 3, 02703 May 16 1988.

[42] *DESQview API C Library*, Quarterdeck, 1988.

[43] *DESQview API Reference*, Quarterdeck 1988.

Paul Healy
Computer Vision Group
Department of Computer Science
Trinity College
Dublin 2
Republic of Ireland

phealy@cs.tcd.ie
Tel: 0001-772941 x1765
Fax: 0001-772204
Telex: TCD 93782 EI

DE VERMIBUS (ON WORMS)

Tony Fisher, Department of Computer Science, The University of York

In Inmos's terminology, a *worm* is a program which propagates through a network of transputers, determining the topology as it goes. I had thought that occam worms were a recent invention; but the following passage from the writings of William of Occam shows that our patron knew something about transputer worms:

> ... *Sol producit vermem cum verme et sine verme... Hoc patet: Quia vermis generatus per propagationem et putrefactionem sunt eiusdem speciei, manifestum est; et tamen vermis productus per propagationem producitur ab omnibus causis essentialiter ordinatis simul; vermis autem productus per putrefactionem producitur a sole sine actione vermis.* ...
>
> *Quaest. in lib. I Physicorum Q. cxxxiv*
>
> ... The sun can produce a worm both with and apart from a worm... This is clear: Worms generated by propagation and by putrefaction are of the same kind, as is manifest. Nevertheless, the worm produced by propagation is produced by all the essentially ordered causes together, whereas a worm

produced by putrefaction is produced by the sun without the action of a worm.

Note the reference to the Sun-based TDS.

The current Inmos worm programs are clearly of the 'propagation' type. Has Inmos investigated worms produced by putrefaction, as suggested by William of Occam?

Tony Fisher — Tel: +44 904 432738
Department of Computer Science — fisher@uk.ac.york.minster
The University of York
York YO1 5DD
United Kingdom

REVIEWS

TRANSPUTER DEVELOPMENT SYSTEM

Inmos Limited, pub. Prentice Hall, July 1988,
pp. xx+491, pb. £25·95, ISBN 0 13 928995 X

This substantial tome is the authoritative documentation on the D700D (D800D) release of Inmos' Transputer Development System, replacing the large-format preprints which early customers received. Unlike some of the other titles in the Prentice Hall series on transputer technology, which betray their origins as compilations of former technical notes and data-sheets, this volume appears to have been written as a coherent whole. It will no doubt prove indispensable to all users of the TDS; most of the material is (very properly) so specific to the particular software package that it is not likely to prove of great interest to those who are not, at least in prospect, users.

The book is divided into two major sections: the first quarter is devoted to a 'User Guide' which gives an overview of the system and a descriptive account of the user's interactions with the folding editor, the compiler and filer utilities, running programs and using the debugger. Included in this part are several examples, which are generally helpful in illustrating the point in question, and also more specialised sections for those who will be using the TDS to develop stand-alone embedded systems or need to step outside the occam model of concurrency to handle communication faults and other operating system concerns. It is the authors' intention that the relevant chapters of this section should be read by any user before starting to use the system. I believe that those who do so, who work through the editor tutorial file supplied with the software and who obtain a good text for learning the occam 2 language, should be reasonably prepared to develop simple programs using the TDS.

The bulk of the book is taken up by a 'Reference Manual', which provides terser reminders of much of the material in the User Guide, more detailed descriptions of the user interfaces to the utilities, to the various libraries supplied with the distribution, to the software tools, and to the host system. As its title suggests, this part is intended more for looking things up in than for reading through from start to finish; it is supplemented by a number of appendices, tabulating such data as the names defined by the software, values of system constants, file formats and a glossary. Finally, there is a reasonably comprehensive index to both parts, something which was sorely missed by users of the preprint version.

The criteria by which to judge software documentation are necessarily different from those against which other technical writing is measured; for instance, fidelity to what has actually been implemented must override abstract notions of technical elegance, and it is unrealistic to expect a manufacturer to be too overtly critical of his own product. Given this, overall I was quite favourably impressed by the way the novel features of the folding-editor-based integrated development system are described for an audience to most of whom they might be disturbingly unlike previous experience.

There are a couple of areas I would single out for adverse criticism: in one or two places the user guide prescribes what should be done to avoid problems, but gives no help in how to get out of the difficulties if human fallibility leads you to break that rule. For instance, if you use the $\boxed{\text{SUSPEND TDS}}$ key to return to DOS, both the user guide and the reference manual caution that you should ensure that the working directory is changed back to the value it had at the time the TDS was suspended before returning to the TDS with a DOS exit command; neither gives any advice on what to do if you don't. (I believe you can get away without any ill effects if you immediately $\boxed{\text{SUSPEND TDS}}$ again and change to the correct directory; if you do any editing, then you find some filed folds in one directory and some in the other – there must be a reasonably straightforward recovery procedure that someone with access to the source could work out.) Similarly, the user guide gives the dire warning that if you edit the contents of a filed fold which has been marked with $\boxed{\text{WRITE PROTECT}}$, then the TDS 'will be unable to write back the changes'; you have to turn to the reference section to discover that you have not just wasted an hour's work, and that you get the fold back, but unfiled. Indeed, a general discussion of the TDS filing system, the advantages and disadvantages of $\boxed{\text{ATTACH/DETACH}}$ making multiple references to the same DOS file (I can find no explicit warning that deleting a fold which is one of several to which a file is attached deletes the physical file, unless write protected), and the interactions between the TDS idea of the file system and DOS manipulations of it (what happens if you use $\boxed{\text{SUSPEND TDS}}$ and copy files over ones the TDS knows about?) would have been a useful addition.

The library mechanism has some infelicities, and the documentation does not always help as much as it could. There is apparently no way to make a 'header' library (one containing only constant and protocol definitions) depend on constants declared in another: a '#USE' directive within the library causes a compilation error in the program which uses it; only experimentation or word-of-mouth reveals that, as validation actually makes no checks at all on such libraries, they need not be self-contained, but may have 'free' occurrences of other constants, which are resolved in the scope current at the place of the '#USE'. We are told that there is no real distinction between 'header' and 'code' libraries, but that 'in practice it is useful to separate them out'; that this is because any separately compiled units in the library are totally unable to refer to the header declarations (for the SC fold delimits the scopes that can be seen inside it, and the library cannot be referred to unless it is validated, and cannot be validated without compiling the SC) is left to us to discover. We are told that libraries must be compacted if they are to be used from a directory other than the one it was developed in, because 'the compiler is unable to read filed folds nested within a file in another directory'; this is, in general, false – the problem lies with resolving filenames given relative to the current directory, and giving an absolute path from the root to the source files results in the descriptor and code files having an absolute name too, and the *compiler* is then quite happy. (Unfortunately, the *debugger* seems to have difficulty locating to a line within such libraries in some circumstances, so the workaround is not perfect!)

As the relatively petty nature of these criticisms may suggest, I don't find anything very seriously wrong with this book. If you have (or intend to have) a TDS, and you cannot get Inmos to give you one for nothing, it is well worth buying.

Michael Goldsmith, Programming Research Group, Oxford University

COMMUNICATING PROCESS ARCHITECTURE

Inmos Limited, pub. Prentice Hall, October 1988,
pp. xii+170, pb. £20·95, ISBN 0 13 629320 4

TRANSPUTER TECHNICAL NOTES

Inmos Limited, pub. Prentice Hall, December 1988,
pp. xiv+246, pb. £20·95, ISBN 0 13 929126 1

The two books *Communicating Process Architectures* and *Transputer Technical Notes*, which were published by Inmos this year, hold few surprises for anybody who has been working with transputers and occam over the last few years – they consist of papers previously published in the Inmos *Technical Reports* series. Although one may have assumed that Inmos would have taken this opportunity to review the material and bring common parts together, this has unfortunately not been the case.

Communicating Process Architectures includes a fairly comprehensive description of the main parts of the occam 2 language with some justification for its current form (e.g. why recursion was not included and why pointers are not feasible). A number of the papers describe how occam is implemented on the transputer, how it can be (and was) used to go from parallel algorithms to microcode in a mathematically rigorous way (for example in the design of the FPU of the IMS T800), and how it can be compiled into silicon, and these are used to demonstrate how a language with simple semantics can be an advantage. There is a useful paper on real-time programming with the transputer and another on the IMS T800 architecture, and these are followed by a number of applications of concurrency to a number of graphics based tasks.

Transputer Technical Notes is more hardware orientated than *Communicating Process Architectures*; unfortunately, it includes several papers which contain too much technical information for a general reader (for example, the crosstalk on long link connections) but not enough for a system designer (who is referred by Inmos to the *Transputer Reference Manual* for more information). The book does, however, contain what is probably the most useful paper for those who are interested in achieving the best performance from the transputer (*Performance Maximisation*) with detail on how to write occam in such a way to make best use of the hardware. A paper to encourage those who doubt the power of the transputer is the honestly named *Lies, Damned Lies and Benchmarks* which shows how the transputer compares to a number of other chips while casting doubt on the amount of importance that should be given to these figures. Other topics covered are use of the IMS C004 crossbar switch (with a very dubious CSP 'model' which bears only a minimal resemblance to the actual operation of the chip), a number of papers on link usage (how to explore a network of transputers, how to recover from failed communications and how to connect links over long distances), a couple of very similar papers on the transputer based navigation system and an interesting paper on a distributed graphics display.

The main criticism of both of these books is the amount of duplication. For example, in *Communicating Process Architectures* we have the basic process blocks of occam described briefly six times; it would seem to require very little editorial time to replace five of these entries by references to an earlier section. One also wonders why it is necessary to show the block diagram of the transputer *quite* so many times. A further annoying feature of both books is in the way in which they are typeset; they are printed using a ridiculously small sans-serif font which after the first ten pages becomes irritating, and which after a hundred and fifty becomes painful.

If Inmos had spent only a little longer on the typesetting and structure of these books I would be pleased to recommend that people who are (or want to be) interested in occam or the transputer should consider buying these books. As it is, I would suggest asking your library to buy them as a better alternative than having lots of loose bound technical reports, and for you to buy a magnifying glass.

Phil Richards, Programming Research Group, Oxford University

PROGRAMMING IN OCCAM2

Geraint Jones and Michael Goldsmith,
pub. Prentice Hall, August 1988,
pp. x+317, pb. £16·95, ISBN 0 13 730334 3

When considering a new book, the sympathetic reviewer must be wary of sparing the author's blushes at the expense of misleading the reader. If the author (or co-author) is also the editor of the publication in which the review is to appear, then the effort must be doubled. Fortunately, for this reviewer and this book the task is easy – *Programming in occam 2* is to be whole-heartedly recommended.

The book's fundamental strength is that its contents match its title. There are no sections on the transputer or parallel architectures. There is no discussion of the relative complexities of various parallel algorithms for solving problems on different machines. Instead, the reader is introduced to occam 2 in the context of programming tasks for which its features make it especially suitable. We are shown that thoughtful use of concurrent processes and channels with appropriate protocols can lead to clear and clean solutions to traditionally tricky problems.

My own favourite example is in the section on formatting output. Initially, we are invited to tut disapprovingly at Pascal's *write* and C's *printf* and their slackness in allowing an arbitrary number of parameters. Then, we see that the same effect can be achieved without resort to such ad-hoc tactics, by passing the string (with embedded placeholders, in the *printf* style) to the new write process directly and concurrently passing the corresponding items (to be embedded in the string) into the process over a discriminated protocol channel. Later, we find our old friend parallel merge sort introduced not as end in itself, but as a framework upon which to demonstrate the suitability of the occam model in programming software monitors (in the sense of observing behaviour).

Elsewhere we find the concurrent programmer's traditional diet of interrupt handling, synchronisation and buffering as well as Huffman coding and Conway's *Life*. All the examples are carefully developed with plenty of explanatory prose and examination of a variety of solutions. The complete programs are gathered together at the end of the book although, as the authors acknowledge, reading a substantial occam program on the page can never be as convenient as exploring it with a folding editor.

The main chapters on the examples are preceded by an introduction to the basic building blocks of occam 2, with great attention paid to detail. For my liking, there is a little too much concentration here on the detailed meaning of various bit level operations and numeric data types. On the other hand, this material is only a very small part of the whole book, and certainly no-one having read these sections could complain that they had not been told! The occam 2 notation is usefully summarised in its own twenty-five page chapter.

Finally, there is a short annotated bibliography with tempting explanations as to why we should be rushing to the nearest research library to follow up the referenced works.

Regular readers of the oug newsletter will not be surprised to discover that the style is both lucid and entertaining throughout.

Murray Cole, Department of Computing Science, Glasgow University

DIGITAL SIGNAL PROCESSING

Inmos Limited, pub. Prentice Hall, August 1989, pp. xvi+266, pb. £19·95, ISBN 0 13 212804 7

This volume of Prentice Hall's INMOS documentation series contains both the full data sheets for a number of the INMOS digital signal processing chips and support products, and a number of documents that have previously appeared as application notes.

The data sheets on the A100, A110, A212 are indeed full, and quite long: 'engineering data' is the description in the introduction. This includes an algorithm-level description, a logical signal specification, as well as the details of electrical and timing specifications of the signals, and packaging details. There are also a few lines on possible applications for each of the chips, although the application note chapters are more use for this. These chips are principally intended to be resident in the address space of – and be configured by – a transputer or similar microprocessor.

The A100 is thirty-two stage one-dimensional transversal filter, the sort of thing you might use for convolution, filtering, Fourier transforms in communications or radar applications; although more remarkable suggested applications are as a matrix multiplier co-processor, or a programmable waveform generator.

The A110 is a significantly more complicated device containing a configurable array of multiply accumulators and long shift registers, apparently intended to be thought of as 'naturally' being divided into three; together with a back-end processor whose complexity defeated me at first reading. This device is intended for real-time image processing applications, capable of two-dimensional convolutions and a variety of the sorts of data transformation of the sort that are found in image understanding and image enhancement applications.

The A121 is a more specialized device capable of calculating the discrete cosine and similar transforms on an eight by eight block of twelve-bit signed numbers, and doing this at signal rates of 20 MHz.

The B009 data sheet is a very brief description of a PC add-in TRAM motherboard carrying four A100s and a T2, intended for experimenting with the A100. The D703 data sheet is an equally brief description of a development system – actually an add-on to a D700 transputer development system – capable of controlling A100 applications and containing amongst other things an occam 2 emulation of the A100 and of the B009, as well as a driver for a B009.

The remaining two-thirds of the book consists of application notes

- ▹ Digital filtering with the IMS A100
- ▹ Discrete Fourier transform with the IMS A100
- ▹ Correlation and convolution with the IMS A100
- ▹ Complex (I&Q) processing with the IMS A100
- ▹ Hardware considerations with the IMS A100
- ▹ Image processing with the IMS A100
- ▹ Cascading the IMS A110
- ▹ The IMS A110 back-end processor

These abound in examples of the uses of the DSP products and abound in specific examples.

Some of the longer notes, those about particular transforms and filtering algorithms, are principally about the algorithms that you might use these chips to implement. For example the Fourier transform note explains the discrete Fourier transform, and some way that particular size transforms would be factorised into parts that could be implemented by A100s. It does not, however, tell you in any sort of detail how to get the parameters into a system of A100s, or how to plug them together.

There are other notes which do things just the other way. In fact the *Hardware considerations* chapter, the tenth in the book, turns out to be what I would have called the user manual of the A100. I might have been less surprised by it, had it had been next to the data sheet rather than nine chapters away!

This book is an interesting collection of material about the INMOS DSP chips, but it is clearly just that: a collection of material. No thought seems to have been given to indicating which parts of the book are about DSP, and which parts are about the INMOS products. Neither, apparently, have any changes been made to the contents of the application notes; for example the references in one chapter will be to application notes by their numbers, rather than page or chapter numbers, giving no hint that these very notes appear elsewhere in the same volume.

Manfred Maurenbrecher

THE HELIOS OPERATING SYSTEM

Perihelion Software Limited, pub. Prentice Hall, April 1989,
pp. xii+510, pb. £19·95, $35·95, ISBN 0 13 386004 3

To design a reliable multiuser, multiprogrammed, multiprocessor operating system for a chip with no support for memory protection or relocation, no 'privileged' execution mode, and no mechanism for handling internal exceptions other than halting the processor, presents a daunting challenge. To choose the right set of primitive operations – rich enough to build into real applications, yet simple and predictable enough that writing software can be a creative act rather than a struggle through a thicket of complexity – is a challenge of a higher order. The Helios system, described here by its joint authors at Perihelion Software Ltd, is an impressive attempt at meeting these challenges, only partly subverted by the dubious wish to present itself as yet another Unix lookalike.

The book is in three parts, providing an introduction and command language guide for users, a system library reference for software developers, and a technical description of some of the internals of the system to fill in details glossed over in the other two sections.

The first section begins unpromisingly with a description of the Helios command processor. This is a reimplementation of the egregious Berkelix 'C shell', taking pains to duplicate baroque and idiosyncratic features like the history list (surely an anachronism in an era of bitmapped windows and on-screen editing), but leaving out a few simple and useful ones like reverse-quote command substitution. The stated intention is 'for Unix users to feel at home'; but the list of commands in the next chapter suggests that experienced Unix users will find themselves not so much at home as in a rather sparsely furnished flat. Aside from commands built in to the shell, a total of 30 Unix-(almost-)compatible utilities are provided, plus another 15 which are specific to Helios. (Compare this with the 135 commands listed in the 1979 Seventh Edition Unix manual, or the 356 commands – at last count – found in the standard directories of a recent SunOS release.)

The second part of the book becomes more interesting, starting with an overview of the real system behind the Unix mask. Helios is described as a 'software backplane' for a network of transputers, providing a distributed infrastructure for dynamically managing hardware resources (processors, memory, and links) and software objects (semaphores, access capabilities, message ports, streams, etc.), loading and controlling tasks (groups of processes sharing a memory address space), and mediating intertask communication. A network-wide tree-structured name space gives names to processors, loaded programs and active tasks as well as to files and devices. The filing system is implemented by distributed server tasks – as indeed are all the operating system services above the message-passing level.

This section introduces one of the novel contributions of Helios: the Component Distribution Language (CDL) which is used to describe the configuration of 'task forces' (collections of tasks on one or more processors, connected by communication streams). An elegant generalisation of the Unix shell pipeline syntax permits the specification of trees of tasks as well as linear chains, without explicit naming and assignment of channels; and other high-level constructors provide for common idioms such as subordination, and farming of replicated tasks. The Helios shell can accept CDL scripts and pass them to the Task Force Manager, which automatically assigns tasks to available processors using heuristics (unspecified) to match stream connections to the actual topology of the network.

There follows a list of the routines in the 'resident library' (the Helios equivalent of system calls), giving for each the C language calling sequence and a brief description of its purpose. In theory this should be enough for programmers to make use of all the facilities of the system; but the functional descriptions are sufficiently sketchy that they will need to turn frequently to the third part of the book, which gives details of the implementation of many system components, and defines the data structures they work with. Fortunately there is an excellent and thorough index, which should enable diligent and determined readers to find answers to most of their questions without recourse to the source code. They will also be assisted by the example programs (five pages of C code) which illustrate simple uses of Helios routines – although a lot more (and more legible) examples would have been welcome.

There is a similar list of the routines in the 'Unix compatibility' library, which implements functions 'largely compatible with the proposed Posix standard'. Those wishing to 'port' Unix software would have liked to see a concise list of divergences from the standard; in its absence they will need to read between the lines and guess at the features which would be awkward or impossible to simulate within the Helios programming model.

Who will want to read this book? It will be essential for Helios users and programmers, since it comprises the main documentation for the system (necessary but not sufficient – it fails to mention, for example, how to create home directories and passwords for new users). As a user manual, it is no worse than most and considerably better than many. The book will also hold some interest for students of operating system design. For them it will not be easy reading; since much of it consists of alphabetical lists of commands or library routines, abounding in forward references to other chapters, it is pretty much necessary to read all of the book before properly understanding any of it.

Richard Miller, Programming Research Group, Oxford University

PARALLEL PROCESSING: TECHNOLOGY AND APPLICATIONS

ed. H. Neishlos, pub. IOS B.V., 1989,
pp. viii+141, pb. Dfl.100, about $55, ISBN 90 5199 021 9

These slim proceedings contain most of the papers presented at a symposium held in Johannesburg in October 1988 by the University of the Witwatersrand with the IFIP Committee of the Computer Society of South Africa and the South African Section of IEEE. I was surprised to find that the first four papers' authors had American addresses, but as you might expect I later found that with the solitary exception of a paper from Inmos Bristol the other two thirds of the papers quoted South African addresses.

It will not surprise anyone at all familiar with current research in this field that hypercubes appear often in the American paper; nor sadly that the bulk if not all of the references to transputers are in the other papers. One of the signs that the field is not yet mature is the number of papers that one sees proposing architectures for more or less general purpose parallel machines, and this collection is no exception. The same is – unsurprisingly – true of ways of designing parallel programs, and this collection talks about a number of radically different paradigms and techniques for implementing them. There are, gratifyingly, also papers about real applications: one as real as the control of induction motors. It turns out that what I had expected to dismiss as heavy electrical engineering is something that requires a deal of real-time computation; and that calculations which might otherwise have used a number of special-purpose digital signal processing chips and floating point processors can reasonably be handled by a couple of transputers. That's what we like to hear.

Inevitably, in a collection as diverse as this, there are unlikely to be a great number of papers that will appeal to any particular reader, but going by what has been presented in the past about a half of these papers fall within the subject area of occam user group meeting technical meetings. The Inmos paper, by the way, is Atkin and Ghee's description of *A Transputer Based Multi-user Flight Simulator* which looks remarkably like the Inmos technical note of the same title, which appears in the *Communicating process architecture* book.

A small thing for which I thank the editors and the authors of one of the papers is a delightful typographical error in the name of the T@!@ transputer.

Manfred Maurenbrecher

PRODUCTS, SERVICES AND ANNOUNCEMENTS

INMOS MARKETING UPDATE

Mark Jones, Inmos, Bristol

As those of you who attended the last OUG meeting in Edinburgh will have seen, Inmos has seen some important changes and made significant progress in 1989.

In September 1989, together with Pasquale Pistorio the President and CEO of SGS-Thomson, we started a worldwide press tour to promote the transputer. This has been backed up by a worldwide advertising campaign totalling a million dollars over the last three months. To date, this has generated over 5 000 direct enquiries and increased the profile of the transputer worldwide.

Our messages of 'Multi-processing Made Simple' and low cost 32 bit processors, have broken down a lot of the barriers and overcome the misconceptions that were held about the transputer family. The 1990s will see the transputer become one of the major microprocessor families and an industry standard.

Some of the highlights of the launch are detailed below.

T400 – the lowest cost 32 bit microprocessor!

The T400 was launched in September at $2 per MIPS. It is the lowest cost 32 bit microprocessor on the market at under $100 even for one-off quantities.

We launched the T400 to enable new users to evaluate the transputer and to demonstrate that multiprocessing is cost effective. As a device for mass production, there are no longer any reasons for not choosing the transputer.

The T400 is available now in a ceramic PGA package and will be produced in surface mount PLCC and QFP packages too. It is software- and pin-compatible with the T425 and T800 and comes in a single speed version of 20 MHz.

The T400 has 32 bit CPU, 2 kbytes on-chip memory, timers, 4 Gbyte external memory interface and two serial links.

This device has opened up many new applications in the embedded application arena and attracted interest from users of 8 bit and 16 bit microprocessors.

H1 – the next generation

H1, as it is code-named, is the next generation transputer. We have pre-announced this product to demonstrate Inmos's commitment to the future and intention to remain at the forefront of technology.

At this stage, we have declared the major features of the H1:

▷ a single processor with over 100 MIPS and 20 MFLOPS performance on a single chip,
▷ code compatible with existing T800 transputers,
▷ large on-chip cache,
▷ increased link bandwidth of 80 Mbytes/sec; there will be a new protocol and conversion devices for communication to existing products,
▷ hardware support for message passing and virtual channels.

H1 will be available in 1991 and further details will be published in 1990.

iQ Systems – a new business

Based on the success of the TRAM and motherboard concept, Inmos has launched a separate business venture called iQ systems to market board level products and software. The hardware includes motherboards, computational TRAMs and application TRAMs such as a Vector processing module, Ethernet and SCSI interfaces.

Software development will concentrate on joint third party developments such as an Ada compiler and Postscript interpreter from Harlequin. Inmos has set up a strategic alliance program called STRAP to encourage developments in this area and if you are interested in any further details, then contact your Inmos sales representative or contact Inmos directly.

New TRAM modules

B411	T800 with 1 Mbyte DRAM, size 1
B415	Differential link TRAM, size 1
B416	T222 with 64 kbyte SRAM, size 1
B417	T800 with 4 Mbytes DRAM, size 4
B418	T222 with 256 kbyte FLASH ROM, size 2
B419	G300 based graphics TRAM, size 6 (see figure 28 on page 112)
B420	vector processing TRAM, size 4 (1 k FFT under 2 ms)
B421	IEEE 488 GPIB interface TRAM, size 4

B422 SCSI interface TRAM, size 2

D700E – Occam TDS

The next revision of the TDS, D700E, will be available 1Q90. It includes:

- ▷ occam compiler updated to D705 software toolset revision,
- ▷ new Iserver included,
- ▷ editor includes new functions such as multiple keystroke macros,
- ▷ supports new processor types e.g. T425,
- ▷ new tools to create files for EPROMs.

Contact your sales representative or distributor for availability.

Mark Jones
Inmos Limited
1000 Aztec West
Almondsbury
Bristol BS12 4SQ
United Kingdom

OCCAM PROMOTION INITIATIVE

Michael Poole, Inmos Limited

Inmos is considering embarking on an initiative to promote the occam language in its own right in addition to reinforcing its current position as the preferred language for transputer systems.

In addition to its support for concurrency, important attributes of the language which make it worthy of promotion include its simplicity and consequent ease of learning, its unambiguous semantics and hence appropriateness for reasoning about programs, and the ability to construct compilers which maximise compile time checking and hence minimise development time.

In addition to projects where concurrency is of the essence, important application areas include numerical computations and safety critical systems.

For a language to be acceptable in such systems it is necessary for there to be an internationally accepted standard, and possible routes to such standardisation are being investigated. Any standard will need to be accompanied by rigorous acceptance procedures for new implementations. It is hoped that formal definition methods will contribute significantly to such a process.

In addition to standardisation activity, which will inevitably include the consideration of possible language enhancements, the initiative would encourage new implementations of the language for targets other than transputers. Inmos has already announced its intention to make available the source of an occam 2 compiler written in C to people considering such implementations.

The placement of articles on occam in appropriate journals will be encouraged. Any members who would like to write such articles, or to get actively involved in any other promotion activities are invited to make themselves known to me. The more enthusiasts there are outside the company the easier it will be to get the necessary support from Inmos management and elsewhere. Do not miss this opportunity to help to ensure the future of occam.

Michael Poole
Inmos Limited
1000 Aztec West
Almondsbury
Bristol BS12 4SQ
United Kingdom

Tel: +44 454 616616
mdp@uk.co.inmos

OCCAM 2 AND TRANSPUTER ENGINEERING COURSE

Computing Laboratory, University of Kent at Canterbury

Course Objectives	To acquire technical knowledge, insight and practical experience of parallel system design using *occam* and *transputer* networks.
Further Details	Harnessing the potential processing power of *transputer* networks requires the development of a fluency in parallel systems design equal to our traditional skills for sequential logic. *Occam* is a simple, small but powerful language which enables such fluency. Software engineering principles, load-balancing techniques, real-time applications and various embedded and super-computing issues will be covered. The strengths, weaknesses and likely future developments of *occam* and *transputer* technologies will be discussed.
Course Members	If you are thinking of using parallel computing to engineer high-performance high-security systems, this is the course for you. If you have picked up basic *occam* syntax and semantics and are wondering how best to exploit its power, come along. If you have never seen any *occam* before, so much the better! Hardware engineers are especially welcome. *C* programmers beware – this course will change your life!! *[Since September 1986, this course has attracted over 200 participants from Industry and Academia worldwide.]*
Course Methods	Informal lectures with a large proportion of 'hands-on' experience being provided through practical exercises and a 'mini-project'. Practical work will be on the MEiKO Computing Surface and will be supervised at the ratio of one tutor for every six attendees. The MEiKO provides a multi-user multi-transputer development and applications environment. Our system will support up to 30 simultaneous users, each with dedicated access to a private network of transputers including at least two T800s. The full system comprises over 131 transputers (including 97 T800s) with a gigabyte distributed file store and three high resolution graphics workstations.
Length & Cost	Five days @ £450 (including lunches and light refreshments).
Dates	Course Nº 14: 26–30 March 1990, Course Nº 15: 25–29 June 1990.

Contact	For a full syllabus, application forms, fees, special arrangements and accommodation, please contact:
	Professor P. H. Welch, Computing Laboratory, The University, Canterbury, Kent CT2 7NF, England — Tel: +44 227 764000 x7695, Fax: +44 227 762811, Telex: 965449 UKCLIB, Email: phw@uk.ac.ukc
EEC Recognition	This course is one of the foundations for a series of courses and technical workshops entitled '*Training for Transputer Technologies*'. These are being developed under contract with the EEC as part of the Communities Action Programme for Education and Training for Technology (COMETT).

occam is a trade mark of the INMOS Group of Companies;
MEiKO and the Computing Surface are trade marks of Meiko Limited.

CODE – A REVOLUTIONARY PLATFORM FOR LEARNING OCCAM

Centre for Development of Advanced Computing, Pune, India

CODE is a tool for the scientific community to migrate seamlessly to the CSP paradigm and overcome the reluctance and inertia in learning a new computing paradigm. It is a full implementation of occam 2. It does not require any expensive hardware and eliminates the need for any add-in transputer cards which are normally required to execute an occam 2 program. CODE helps users learn occam by writing programs, checking for errors and debugging during execution. All this at a price that is absolutely unbeatable.

Key features

- ▷ Requires no additional and expensive transputer hardware in the form of plug-in cards (as IMS B004) to execute and occam program.
- ▷ Integrated user-friendly environment with pull-down menus for user interface.
- ▷ Full Inmos occam 2 implementation and support.
- ▷ A folding editor compatible with the Inmos TDS folding editor which naturally helps users adopt a top-down approach for program writing.
- ▷ Syntax checker to help the user in writing an error-free program.
- ▷ Interpreter to enable program execution in the debug and non-debug optional modes.
- ▷ Extensive support for debugging unlike any other post-mortem debuggers; hence helps check the program logic.
- ▷ Assures deadlock detection and allows monitoring of the deadlock cause by setting a watch on the channel.
- ▷ Helps secure the trace information for the program.
- ▷ Gives simulated 'compute and communication' times for any process.

- ▷ Provides process status information like the number of waiting processes, the current process status, etc.
- ▷ Allows setting of break-points and program execution in either 'step-through' or 'step into' modes.
- ▷ Lister for source listing of programs.

Operating environment

PC based MSDOS version 3.0 onwards. IBM PC-compatible machine with colour monitor and a hard disk and floppy drive or two floppy drives.

Using CODE

CODE is an integrated environment for developing and learning occam. The user can write an occam source code for a single transputer using the folding editor, check it for syntax errors and later execute it.

A program can be executed in any of two optional modes – debug and non-debug modes – when the logic is correct.

In the non-debug mode, the user can further select the trace ON/OFF facility depending on whether it is required or not.

Components of CODE

Folding editor

Inmos TDS compatible folding editor which provides 'to write' and 'to edit' occam programs using folds in CODE. The source file generated on this editor is Inmos TDS compatible.

Syntax checker

It checks syntax errors in the occam code and reports the errors encountered. It is responsible for generating the object file for the interpreter.

Interpreter and debugger

When executed in the debug mode, it helps the user to keep watch on variables and channels and observe the process status, such as process waiting for input, output, altinput, etc.

Lister

It helps to get the source line listing of a program with or without line numbers. It opens all nested folds and lists the programs.

Ordering information

CODE version 1.0 is available on one floppy disk along with a user manual.

C-DAC reserves the right to make changes in specifications at any time and without notice. The information furnished here is believed to be accurate. However, no responsibility is assumed for its use, nor for any infringement of patents or other rights of third parties resulting from its use.

C-DAC, Centre for Development of Advanced Computing, is a Scientific Society of the Department of Electronics, Government of India.

Centre for Development
of Advanced Computing
Pune University Campus
Ganeshkhind Road
Pune 411 007
Maharashtra
India

Tel: (0212) 332461, 332479
Fax: (0212) 337551
Telex: 0145-615 CDAC IN

INMOS, IMS and occam are trade marks of the INMOS Group of Companies.
Copyright for CODE has been applied for.

PARALLEL PROCESSING AND THE TRANSPUTER – A VIDEO TAPE

Microelectronics Educational Development Centre, Paisley, Scotland

The video provides a short introduction to the concept of parallel processing and the Inmos transputer, hopefully answering some of the most common questions which arise.

WHAT IS PARALLEL PROCESSING? The introduction examines the restrictions inherent in existing computer architectures and introduces the concept of parallel processing as a possible solution.

WHAT IS A TRANSPUTER? This segment introduces the architecture of the transputer family of microprocessors, highlighting the three main types of processors available (T800, T414, T212). It also looks at the unique design features of the chip: communication and memory.

HOW DO I PROGRAM THE TRANSPUTER? The software segment introduces the occam programming language which provides the programming model for the design of the transputer.

WHAT CAN I USE IT FOR? The application section of the video looks at two different applications of transputer based systems: the first is the National Engineering Laboratory, East Kilbride, who have extensive experience and success using transputers in the field of image processing; the second is the Edinburgh Concurrent Supercomputer project based at the University of Edinburgh.

The video is produced jointly by the Microelectronics Educational Development Centre, a Scotland-wide training agency for higher education and industry in microelectronics, microcomputing and information technology, and the National Engineering Laboratory, East Kilbride, a national body with an unrivaled reputation in applied research.

Availability

The tape is available in VHS format, and lasts 18 minutes. It is aimed at the Further and Higher Education sector in Scotland, and is distributed free of charge to these institutions, but may also be of interest to others.

Unfortunately, funding does not permit free distribution outside the Scottish education sector but surplus copies are available to academic institutions in Europe at a nominal cost: £35 in the UK, £40 in the rest of Europe. All requests must be accompanied by payment in Sterling.

Carol Higginson	Tel: +44 41 887 2158
MEDC	Fax: +44 41 889 9755
8/14 Storie Street	Telecom Gold: 81:MEC001
Paisley PA1 2BX	Telex: 778951 PCT LIB
Scotland	

MEGA-LINK TRANSPUTER BOARDS

Robert A. Sang, SANG-Computersysteme GmbH

MEGA-Link01 plus, multi-transputer farm-card for PC

The *MEGA-Link01 plus* is a farm-card to build up high performance networks, consisting of four T425 or T800 transputers (20, 25 or 30 MHz), each with a local RAM of 1, 2, 4 or 8 megabytes.

The four transputers on the board are interconnected in a pipeline structure, leaving at least two links of each transputer for manual configuration via cables or electrical configuration with the *MEGA-Link-Switch.*

A fast DMA-interface (B008-compatible) combined with the optimized *MEGA-server* allows communication with the host computer at a rate of 300–400 kByte per second.

Because of its extremely high packing density (up to four transputers and thirty-two megabytes of RAM into a single PC slot) and its flexible amount of memory the *MEGA-Link01 plus* is designated as a high-performance farm-card to build up large, reconfigurable transputer networks.

Hosts IBM PC/AT and compatibles, Commodore Amiga 2000, Atari ST, or *MEGA-Link-Station.*

MEGA-Link02, high performance transputer graphics board

The *MEGA-Link02* is the first commercially available board using the new Inmos G300 Colour Video Controller. It has been developed to allow the transputers a powerful display of their graphics data. High resolution with few (256) colours and true-colour display in TV-resolution can be readily realized.

The *MEGA-Link02* is a PC/AT plug-in board, consisting of a T425 or T800 transputer (20, 25 or 30 MHz), with 1, 2, 4 or 8 megabytes of main memory plus 1 or 2 megabytes of dedicated dual-ported video-RAM, which the transputer can access more than 95% of the time without any restriction.

An Inmos G300 Colour Video Controller manages the video RAM and the generation of the video signals. The G300 allows either a display of 16·7 million colours simultaneously, or 256 out of a palette of 16·7 million colours.

Some of the possible resolutions are:

▷ 1 024 × 768, 72 Hz non-interlaced, 256 out of 16·7 million colours,

▷ 1 280 × 1 024, 67 Hz non-interlaced, 256 out of 16·7 million colours,

▷ 640 × 480, 72 Hz non-interlaced, 16·7 million colours simultaneously.

A fast DMA-interface (B008-compatible) allows communication at a rate of 300–400 kByte per second between the transputer and the host computer. The links of the transputer can be used to connect the *MEGA-Link02* with other boards via cables or the new *MEGA-Link-Switch.*

The application spectrum of the *MEGA-Link02* ranges from a stand-alone transputer graphics board (e.g. to have a low cost station running Helios with X-windows) to a high-performance graphics server in networks consisting of other *MEGA-Link* boards.

Hosts IBM PC/AT and compatibles, Commodore Amiga 2000, Atari ST, or *MEGA-Link-Station.*

MEGA-Link03, transputer board with from 1 up to 32 megabytes of DRAM

The *MEGA-Link03* has been developed to cope with memory-hungry tasks using transputer systems.

It puts a RAM which can range from 1 to 32 megabytes – depending on the users' requirements – at the disposal of a T425 or T800 (20, 25 or 30 MHz). Additional memory will be supplied at the price-difference.

An unlimited number of transputer boards can be connected via the four links including graphics output (*MEGA-Link02*), farm-cards (*MEGA-Link01 plus*), electronic configuration units *MEGA-Link-Switch*) or a very fast SCSI hard disk controller (*MEGA-Link04*, to be available October '89).

The fast DMA-interface and the new *MEGA-Server* allows data transfer rates of 300–400 kByte per second with the host computer.

The *MEGA-Link03* can be used as an inexpensive start into the 'transputer world' as well as a front-end board in powerful, reconfigurable transputer farms.

Hosts IBM PC/AT and compatibles, Commodore Amiga 2000, Atari ST, or *MEGA-Link-Station.*

MEGA-Link-Switch, PC board with programmable crossbar switch

The *MEGA-Link-Switch* is a short plug-in card featuring an Inmos C004 programmable crossbar switch with thirty-two link connectors and a control link, allowing the user to change dynamically his network topology (e.g. pipelines, arrays, trees).

An unlimited number of transputer boards can be connected using the *MEGA-Link-Switch* in any topology you will need.

Hosts IBM PC/AT and compatibles, Commodore Amiga 2000, Atari ST, or *MEGA-Link-Station.*

MEGA-Link-Interface, interfaces to Commodore Amiga 2000 and Atari ST

The *MEGA-Link-Interface* boards for the Commodore Amiga 2000 and Atari ST allow the users of these computers to use the power of the *MEGA-Link* series of transputer boards.

Included with the interfaces is the modified version of the *MEGA-Server*, allowing it to use Parallel C, Parallel Fortran and Parallel Pascal compilers from 3L and the OCS and Toolset from Inmos.

The server software needed for the Helios operating system is currently under development.

Hosts Commodore Amiga 2000 or Atari ST.

Robert A. Sang
SANG-Computersysteme GmbH
Am Wuennesberg 13
4300 Essen 1
West Germany

Tel: +49-201-71 01 191
Fax: +49-201-71 04 10

THE TOPEXPRESS TRANSPUTER LIBRARIES

F. W. Wray, Topexpress Ltd

Over recent years the demand for high performance computing has risen dramatically. This has been particularly so in the area of scientific and engineering computing where hardware has rapidly advanced to satisfy that demand. In parallel with these improvements in hardware has come the development of routines that solve standard numerical problems. These routines have been highly optimised and are usually written in a high level language, such as Fortran, to guarantee portability. Many libraries of such routines are now available. Most sacrifice performance for portability by failing to take advantage of specific features of the hardware. This is particularly important in the case of the transputer where for routines involving floating point operations, sections of code written in assembler can often outperform equivalents written in Fortran or occam by a factor of two or more.

The Topexpress Parallel Processing Group, a highly skilled team of professional programmers and mathematicians, has realised the need for high performance numerical algorithms to run on transputer systems. In response to this requirement, Topexpress has successfully executed the following design strategy: firstly, we have written a set of highly optimised assembler primitives which achieve maximum numerical performance on a single processor; secondly, we have written a library of well understood numerically efficient algorithms, running on a single transputer and making extensive use of the optimised primitives. Finally, we have converted many of these single transputer routines to run efficiently on a fixed topology pipeline of processors. The primitives, single and multi-processor routines are callable from Fortran, C and occam and enable the non-specialist user to obtain the benefits of the transputer's power whilst programming in a serial manner in a high level language.

The Topexpress vector library (VecLib)

The performance of scientific and engineering application programs can be dramatically enhanced by using the Topexpress Vector Library. The Library, which contains over 100 single precision, double precision and complex vector primitives, is written in optimised T800 assembler. VecLib routines are callable from Helios, Idris and 3L languages and occam. The library has a well established user base and is supplied with full documentation. Included with the library is an executive which maximises performance by dynamically loading library code into on-chip RAM. As an example of the typical performance increases the library can offer, the following megaflop rates are for a single precision vector multiply (VMUL) and for a single precision vector scalar multiply and add (VSMADD). The operand and result vectors are all off-chip. Performance figures are given for a T800-20 with four-cycle RAM.

	FORTRAN	VecLib
VMUL	0.18	0.65
VSMADD	0.33	1.05

The VecLib peak Megaflop rates for other important routines, under the same conditions as above are:

Vector dot product	1.52
Vector sun of elements	1.67
Vector magnitude squared	1.77
Vector scalar multiply	0.82
Vector add	0.73
Vector scalar add	0.84
Vector square	0.77
Vector divide	0.56
Vector maximum element	0.64
Complex vector multiply	1.27
Complex vector divide	1.33
Complex vector scalar multiply	1.34

Typically, with an $n_{1/2}$ of less than 10, vectors of length 25 are operated on at 75% of peak performance, those of length 50 at more than 95% of peak performance. Routines are available both for contiguous vectors and for vectors with non-unit stride.

The Topexpress mathematical procedure library (MathLib)

The Topexpress Mathematical Procedure Library comprises over 50 numerical algorithms important for many scientific and engineering applications. The procedures are numerically efficient, have been optimised for the T800 transputer using assembler where necessary, and are available in single and double precision versions. The library is available for Helios, Idris and 3L languages and occam. Included in the library is an executive which can maximise performance by dynamically loading code into on-chip RAM. This library has a well established user base, full after-sales support, and is supplied with detailed documentation.

The library contains the following:

- ▹ general matrix procedures,
- ▹ sparse matrix solvers,
- ▹ Toeplitz matrix procedures,
- ▹ symmetric positive definite matrix procedures,
- ▹ eigenvalue/eigenvector procedures,
- ▹ fast Fourier transforms,
- ▹ Bessel, gamma, Legendre and error functions,
- ▹ sorting procedures,
- ▹ optimisation procedures,
- ▹ curve fitting procedures,
- ▹ polynomial root finding procedures.

As an example of the performance of the library routines, the following timings are for the solution of a set of linear equations using Gaussian elimination with partial pivoting (EQSLV), the inversion of a matrix (INVERT), the calculation of all the eigenvalues/eigenvectors of a symmetric matrix by the QR method (EVSPD), a complex to complex fast Fourier transform (FFT), and an integer merge sort. All timings, in milliseconds, have been made on a T800-20 with four-cycle RAM using single precision versions of the routines

Matrix Dimension	10	30	60	100
EQSLV	3	39	225	905
INVERT	8	131	809	3180
EVSPD	47	912	5590	23850
Transform Size	128	512	2048	8192
CFT (without lookup tables)	7	34	163	752
CFT (with lookup tables)	6	30	145	677
Sort Length	1000	2000	4000	8000
IMSORT	57	127	278	604

The Topexpress parallel library (ParLib)

The Topexpress Parallel Library contains a set of concurrent numerical algorithms optimised to run efficiently on an array of T800 transputers. The procedures are available in single and double precision versions and make use of assembler, where necessary, to maximise performance. The library is supplied with an executive which, on a call to a library routine by a user program, will load the appropriate code and handle all subsequent data input/output. The simple and effective user interface permits the processing power of transputer networks to be harnessed whilst programming in a sequential manner. The library, available for Helios, Idris and 3L languages and occam, includes the following:

- ▹ fast Fourier transform routines,
- ▹ general matrix routines,
- ▹ eigenvalue/vector routines,
- ▹ symmetric matrix routines,
- ▹ sparse matrix routines,
- ▹ iterative equation solvers,
- ▹ sorting routines.

As an example of the performance of the library routines, the following timings are for the solution of a set of linear equations with partial pivoting for a single right hand side (GAUSSP), the inversion of a general matrix using the same procedure (GAUSSP), the solution of a symmetric positive definite matrix equation with a single right hand side (CHLSLP), the calculation of all the eigenvalue/eigenvectors of a symmetric matrix by Sturm sequences (STURMP), a two-dimensional real to complex fast Fourier transform (R2DFTP) and an integer sort routine (ISORTP). All timings, in milliseconds, have been made on an array of four T800-20s with four-cycle RAM, 20 Mbit/s links using single precision versions of the routines, and include time for transfer of data to and results from the array

Matrix Dimension	50	100	200
GAUSSP (1RHS)	84	372	2057
GAUSSP (inversion)	147	855	5601
CHLSLP	26	115	697
STURMP	579	3160	18493
Transform Size	64×64	128×128	256×256
R2DFTP	49	194	757
Sort Length	2048	4096	8192
ISORTP	38	85	183

These routines are highly parallel and run on an array of any number of transputers.

Whilst MathLib and ParLib do not compare in size with large well-established numerical libraries, they already include most commonly used algorithms, and are growing rapidly. Furthermore, in many cases, their performance is a factor of two or more times better than that of other libraries. This is particularly so in the case of the Parallel Library whose performance is better than that of its competitors by a substantial margin. For further information telephone or write to:

Kate Ramsey — Tel: +44 223 462121
Topexpress Ltd
Poseidon House
Castle Park CB3 0RD
United Kingdom

CONCURRENCY: PRACTICE AND EXPERIENCE

an international journal from John Wiley and Sons, for users and designers of concurrent applications, hardware and software

Readership

Primarily, *Concurrency: Practice and Experience* aims to be a forum for cross-fertilisation of ideas on developing algorithms and software from users and designers of concurrent machines.

Concurrency: Practice and Experience will be a major attraction for computer scientists who are involved in the design of concurrent computers and the associated languages and systems software.

The practical approach will appeal strongly to people from a wide variety of scientific disciplines – amongst them physicists, molecular biologists, and chemists who are applying the technology in their field.

Aims and goals

Recent developments in technology have stimulated the development of concurrent computers. These machines consist of a collection of processors connected in a network – or alternately a collection of processors sharing access to a common memory. These include both general purpose MIMD and SIMD architectures and special purpose systems such as neural networks. Optical and dataflow hardware can be expected in the future. There are now several commercially available concurrent computers and an increasing number of microprocessor chips specifically designed to permit the construction of parallel computers varying in size from PC add-in boards with a few processors to 64 000 processor supercomputers.

These machines are being successfully applied in a wide range of application areas especially in science and engineering. This is producing a substantial amount of practical experience in those problems which parallelize well and the features of hardware and systems software needed to use concurrency effectively. There are also new computational methods, such as cellular automata and massively parallel neural networks, which are particularly suited to concurrent execution. At present, there is no journal that brings this work together. Results, if published at all, are scattered through specialized technical journals.

This journal will therefore focus on practical experience with concurrent machines, especially:

▹ Concurrent solutions to specific problems
▹ Concurrent algorithms and computational methods
▹ Programming environments, operating systems, and tools
▹ New languages
▹ Performance design, analysis, models and results

Note for contributors

We encourage papers from a broad range of authors and points of view and will attempt to choose referees who will judge papers on quality and not approach. If you are interested in submitting a paper please contact an editor. Full notes for contributors will appear in each issue and are available from the editors and the publisher.

Editor

Geoffrey C. Fox
California Institute of Technology
Mail Code 206-49
Pasadena, California 91125
United States of America

ARPANet: gcf@hamlet.caltech.edu
BITNet: gcf@caltech
Tel: +1 818 356-3765
Fax: +1 818 584-5917

Associate editors

Paul Messina
California Institute of Technology
Mail Code 158-79
Pasadena, California 91125
United States of America
ARPANet: messina@zephyr.caltech.edu
BITNet: messina@caltech
Tel: +1 818 356-3907

Tony Hey
Department of Electronics
and Computer Science
University of Southampton
Southampton SO9 5NH
United Kingdom
Tel: +44 703 559 122 x2029

International editorial board

Dr Donald M. Austin, Department of Energy, Washington; Dr Jacob Barhen, Jet Propulsion Laboratory; Professor Enrico Clementi, IBM, New York; Dr Erik DeBenedictis, Ansoft Corporation, Pittsburgh; Dr Dennis Duke, Florida State University, Tallahassee; Dr W. M. Gentleman, National Research Council of Canada, Ottowa; Dr Milton Halem, Goddard Space Flight Centre; Professor Per Brinch Hansen, Syracuse University, New York; Professor Ken Kennedy, Rice University, Houston; Professor H. T. Kung, Carnegie Mellon University, Pittsburg; Dr Lennart Johnsson, Thinking Machines Corporation, Cambridge, Massachusetts; Dr David May, INMOS Ltd, Bristol; Professor Oliver McBryan, University of Colorado, Boulder; Dr Raul Mendez, Institute for Supercomputing Research, Tokyo; Dr Steve Otto, University of Southampton, England; Dr George Paul, IBM Thomas J. Watson Research Centre, New York; Professor Dan Reed, University of Illinois, Urbana; Professor Joel Salz, NASA Langley Research Centre, Virginia; Professor Martin Schultz, Yale University, Connecticut; Dr Danny Sørensen, Argonne National Laboratory, Illinois; Professor D. J. Wallace, University of Edinburgh, Scotland; Dr Andy White, Los Alamos National Laboratory, New Mexico; Dr Daniel F. Wiener II, University of Virginia, Charlottesville; Dr Hans Zima, Universität Wien, Österreich.

COMMERCIAL PARALLEL PRODUCTS USE EXPRESS

Larry Lesser, ParaSoft

The EXPRESS parallel environment has been used to create parallel versions of:

- ▹ COSMOS finite element program, developed by Structural Research Corporation of Santa Monica, California
- ▹ NeuralWare Professional Works II by NeuralWare of Pittsburg, Pennsylvania

Also, the Strand88 parallel language has been ported to EXPRESS and will be available wherever the EXPRESS system is available.

The parallelization of COSMOS and NeuralWare Professional Works II was done as joint efforts between ParaSoft and the software authors, while AI Ltd, the author of Strand88 did the porting to EXPRESS with essentially no assistance from ParaSoft.

Performance characteristics for each product are better than expected. As these are all commercial products available for sale, performance was critical. The successful use of an environment, EXPRESS, should help end some of the concerns about overheads that may be associated with such systems, though EXPRESS provides the fastest performance with the least overhead and is being chosen over alternative systems when performance is an issue.

ParaSoft provides tools at each stage of the parallelization process to help create completed applications as easily as possible. The typical steps are:

- ▷ Parallelization – EXPRESS provides global functions, automatic data distribution, scalability and load balancing.
- ▷ Debugging – ParaSoft has created NDB a parallel source level debugger that is similar to SUN's dbx.
- ▷ Performance – ParaSoft provides PM a parallel performance monitor to help find where time is being wasted.

EXPRESS is available on most transputer systems, including PC, Mac or Sun based products as well as NCUBE and Symult parallel computers. Questions on EXPRESS or the other products can be addressed to

Larry Lesser
ParaSoft
27415 Trabuco Circle
Mission Viejo, CA 92692
United States of America

Tel: +1 714 380-9739
Fax: +1 714 380-9739

VIDEOPAINT SOFTWARE USING TRANSPUTER IMAGE PROCESSING

Peter Kruger, Digithurst Ltd

A highly effective video paint package, DHPAINT, has been developed by Digithurst Ltd for use with its Transputer-based MicroEye TC image grabbing and processing board for PC-AT and compatible computers.

DHPAINT is to be supplied with MicroEye TC, giving users a means of becoming quickly familiar with the board's exceptional capabilities and functions, and as a valuable tool for image capture, retouching, presentation and output.

DHPAINT works with images of 720 × 512 pixels, with 24 bit (16·7 million) colour information, and supports VGA display format.

The package is operated from an easy to learn graphic icon menu using a mouse, digitising tablet or other pointing device. It allows users to grab an image, and then to modify or retouch it in virtually any way conceivable, while displaying the effect as the work is done. The transputer processor ensures that flood, shading (vignette) and pattern fills occur very quickly indeed.

DHPAINT has a full complement of freehand, line, Bezier curve, polygon, rectangle, circle and text drawing tools with 32 preset brush shapes, and the ability to modify a brush to any 32 × 32 pixel pattern. Colours can be picked from the screen, from a palette, or mixed by the user from Hue, Saturation, Intensity or Red, Green, Blue values. There are 11 logic modes available when drawing, including Tint and Blend, achieving a huge variety of possible effects and textures.

The resulting display can be 'colour balanced' under user control by graphically programming the way that pixel values held on the MicroEye TC actually map onto the colours displayed. This makes it possible to achieve anything from selective lightening, darkening or intensifying of colours in a grabbed image when retouching a photograph, to a complete rearrangement of the colours giving 'psychedelic' or pseudo-colour effects on a moving image. Pattern drawing, spraying or fill is possible by selecting a pattern area from the screen image. When a pattern is generated from an area of a grabbed photographic image, it allows the creation of completely realistic retouching effects, output from which is difficult to distinguish from the quality of the original video image.

A cut-out function allows an arbitrary-shaped area of the image to be picked and saved, and then repositioned, resized and redrawn on the same, or another, image. This is essential for applications such as photofit and hair-styling, where a library of standard images can be quickly called up from disk, and tried in position until the desired effect is achieved.

One of the most exciting and powerful features of DHPAINT is a sophisticated stencil, or masking, function. By using the two picture stores on the MicroEye TC board, the stencil function offers a variety of image effects by combining the two images.

For example, store and X-ray image of a hand in the reference store, and grab a real image of a hand into the first store. An outline stencil can be cut in the 'real' hand, and filled with the same portion of the X-ray hand, to achieve the effect of a cut-away showing the bone structure below the flesh. Alternatively, the X-ray image can be sprayed onto the 'real' hand to achieve a ghostly effect of the bones showing through the hand. This powerful technique has many uses in creating cut away and exploded views from real objects, or visualisation, or combinations of the two to show, for example, the effect of a new development on the landscape.

DHPAINT is a valuable image processing tool for creating picture libraries, graphic designs and visualisations, storyboards and so on. Digithurst have also modified the package for vertical market applications such as hair-styling, landscape gardening, architectural visualisation and photofit, where a more specific user interface is needed. DHPAINT will load and save images in TIFF and TGA formats, for compatibility with other applications such as slide making and electronic publishing software.

For further information, contact:

Peter Kruger, Managing Director — Tel: +44 763 242955
Digithurst Ltd — Fax: +44 763 246313
Newark Close
Royston, Herts SG8 5HL
United Kingdom

MEIKO IN-SUN COMPUTING SURFACE HARDWARE

The computing surface is a scalable, multi-process distributed memory architecture. Individual processors execute sequential threads of application programs within their own dedicated memory systems. They co-operate when required by exchanging information via high performance inter-processor message links. Embedded microcode schedulers manage task synchronisation while threads co-operate.

Processors are used independently for sequential applications, or in groups for parallel applications. The Computing Surface message link switch allows the optimum network topology to be used for every application. The message link switch, constructed with proprietary VLSI routing chips, enables high speed, low latency message paths to be established between co-operating processors under software control – effectively allowing application software to determine its own parallel machine architecture.

The scalability of the Computing Surface architecture has allowed hundreds of machines to be constructed, ranging in size from a few processors, yielding supermini performance of a few megaflops, to hundreds of processors with gigabytes of distributed random access memory able to sustain supercomputer performance in the gigaflops region.

- ▹ The In-Sun Computing Surface provides a range of board level alternatives for ready integration within Sun Workstations and servers. (Parts MK200, MK201, MK202 are illustrated on the front cover.) Multiple boards may be combined utilising the expansion slots of these industry standard host machines. Further, unlimited expansion is achieved by attaching Computing Surface modules to the external interfaces of In-Sun boards.
- ▹ The individual processors are Inmos T800 transputers, each delivering five VAX MIPS sustained performance. They are purpose built for distributed memory message passing multi-processor construction.
- ▹ Transputer message passing links are connected under application software control using Meiko's proprietary low latency message switch. Total flexibility in parallel application topologies giving optimum match of machine to problem.
- ▹ System health monitoring and run-time diagnostics are directly supported by Meiko's proprietary System Supervisor VLSI chip incorporated with each individual processor.
- ▹ High performance interfacing to the Sun windows and fileserving environment is ensured by the use of further dedicated T800 transputers with dual ported shared memory in the Sun VME address space.

Meiko Scientific
650 Aztec West
Bristol BS12 4SD
United Kingdom

Tel: +44 454 616171
Fax: +44 454 618188

M212 TRAM BASED DISC SUBSYSTEM

Centaur Computer Systems

Centaur computer systems announce the launch of an ST506/412 disc driver.

This size 4 TRAM offers a 100 MIPS M212 16 bit transputer and 64 kbytes of SRAM for program storage or sector buffering.

The CTDM2-20 is designed to control four drives in any combination of floppies ($3\frac{1}{2}''$ or $5\frac{1}{4}''$) and Winchesters, utilising either mode 1 or mode 2 operation.

For more information, contact:

Paul Esmail
Centaur Computer Systems
Triona House
London Road
Bracknell, Berkshire RG12 5AB
United Kingdom

Tel: +44 344 423695

IF/PROLOG ON THE TRANSPUTER

Oskar Bartenstein, InterFace Computer Japan Ltd

InterFace Computer GmbH, München, West Germany and InterFace Computer Japan Limited, Tokyo, Japan are pleased to announce that IF/Prolog V3.4.4 is now available for the INMOS transputer IMS T414 and IMS T800 family of parallel RISC processors.

IF/Prolog is an industrial grade implementation of the Artificial Intelligence programming language Prolog.

It offers a complete programming and testing environment with exception handling, floating-point arithmetic, module concept and screen oriented debugger. An incremental compiler allows interpreter and compiler to call each other.

Designed as an open system, IF/Prolog features a bidirectional interface to the C language, supporting all Prolog data types, unification, and control structures including cut, fail, and backtracking.

Complementing the system programming languages 'C' and 'occam', IF/Prolog is probably the first compiler based very high level programming language supported worldwide for PCs, UNIX machines and mainframes that is available for the transputer in a fully compatible version.

The following gives an overview of IF/Prolog on the INMOS transputers T414 and T800.

Software requirements

IF/Prolog can be used on any environment which is supported by 3L Parallel C V2.0. IF/Prolog is available for both the T414 and the T800.

IF/Prolog comes with its full functionality as on UNIX workstations. However, since there is no shell available on the transputer you cannot invoke operating system commands.

Hardware requirements

Each transputer processor in a system supports its own IF/Prolog process. IF/Prolog can be used in single transputer add-on boards and on multi-processor systems.

Minimum memory recommended is 2 Mbyte for each transputer, so IF/Prolog can be used on TRAM modules also. IF/Prolog object size is about 200 kbyte.

IF/Prolog is available for IBM PC and NEC PC hosts, distribution media is 5″ 2HD floppy disks. We recommend use of a hard disk if you intend to do serious work. IF/Prolog itself does not require the use of a hard disk. A system that is hosted by a Sun is under preparation.

Special features on the transputer

IF/Prolog is delivered compatible with IF/Prolog for workstations and minicomputers. You can use all communication facilities of the transputer via IF/Prolog's C language interface using 3L Parallel C V2.0, as far as they are supported by Parallel C.

Each IF/Prolog process runs on one transputer, with its own unshared Prolog data base. Thus there is no communication overhead unless explicitly requested by the programs.

Performance

On a IMS T800 we measured 8·2 kLIPS for the execution of the naïve reverse benchmark. This is slightly better than the performance of a MicroVAX II for the same application.

IF/Prolog on the transputer

For embedded systems and mass products, InterFace Computer offers the ROM based version of IF/Prolog. We can also assist with system integration.

Applications

IF/Prolog is a general purpose programming language and the transputer is a fairly general purpose machine, so lots of different applications are to be expected.

There are several distinct types of typical IF/Prolog-on-transputer configurations.

The most spartan one is the use of IF/Prolog on an add-on board in an IBM PC, NEC PC or compatible, mainly for two purposes:

▷ evaluation of the transputer technology,

▷ speed within a low cost hardware system and within MS-DOS, but without MS-DOS memory limitations.

The second major configuration consists of few transputers running concurrently, each supporting a time independent process. For example in an embedded real time expert system one processor each for the user interface, for the inference process and for the sensor/actuator interaction. This is an example for large grain parallelization. The actual system can still be configured on an add-in board on a PC. Products to be announced by INMOS in the near future will allow run time systems to be burned into ROM, e.g. for volume products or for very rough environments.

The third type of configuration is the large processor array. This kind of system is used for brute force computational tasks and for pure research e.g. on neural nets or parallel computing.

The first IF/Prolog supplied for the transputer was delivered to the College of Engineering of The University of Tokyo, for a machine consisting of eight T800 and sixty-four T414 transputers. IF/Prolog on this machine is used for advanced robotics and intelligent vision research.

Further information

Request information on IF/Prolog on the transputer from either

Dr Oskar Bartenstein
InterFace Computer Japan Limited
205 High City Hongo 3-23-5 Hongo
Bunko-ku, Tokyo, 113, Japan
Tel +81 3-818-5826
Fax +81 3-818-5829

Annette Kolb
InterFace Computer GmbH
Garmischer Straße 4
D-8000 München 2
West Germany
Tel +49 89-510-8655
Fax +49 89-510-8628

FAST9 UPGRADE MAINTAINS QUINTEK EDGE

Maintaining its commitment to leading edge, high performance parallel processing, Quintek Limited of Bristol has now upgraded the established FAST9 transputer board to include the latest technology and processing power.

With up to 4 Mbyte of DRAM on each of the nine processors, the FAST9 is designed to fit in a single slot on IBM PCs or compatibles. The latest FAST9 provides greater than 100 MIPS of total processing power (200 MIPS peak) and, using the Inmos T800-25 floating point transputer, greater than 32 MFLOPS peak. For the future, the FAST9 will also be able to use the Inmos T800-30 transputer when available. For users who may wish to extend the processing power, the FAST9 can be supplied initially with just four processors and later upgraded to nine.

Flexibility has remained the keynote of the FAST9 since its introduction in 1988, now both in terms of memory, where 1 Mbyte or 4 Mbyte can be supplied on each processor and in the number of transputers initially fitted. A particularly useful configuration is one with 4 Mbyte on the master transputer and 1 Mbyte on each of the eight slaves. Similarly, there is flexible software control of link configurations using the C004 link exchange, with a number of boards being easily connected together to provide large parallel processing resources.

The FAST9 is established as an extremely reliable and cost-effective product, used in large numbers throughout the world. It is also particularly versatile, capable of operating with a variety of standard languages including Fortran 77, C, Pascal, Occam, and Ada with their associated debugging, profiling and vectorizing tools. The FAST9 can provide the central processing resource for the Helios Unix-style parallel operating system with X-windows. Applications where the FAST9 is providing high performance parallel processing include:

- ▷ image processing,
- ▷ fingerprint recognition,
- ▷ signal processing,
- ▷ parallel searching and sorting,
- ▷ computational fluid dynamics,

- ▷ optical design optimisation,
- ▷ finite elements and difference software,
- ▷ many applications through the SERC/DTI Load Pool.

These latest enhancements ensure that for existing applications and for new products or research, the FAST9 – one of the family of FAST boards from Quintek – remains at the forefront of innovation and performance.

INDUSTRIAL I/O FOR TRANSPUTER SYSTEMS

Cambridge RISC Machines Limited

The Cambridge RISC Machines CRM402H – the first in a range of I/O boards for transputer systems – is a digital input/output board in the standard (3U, non-extended) Eurocard format. It provides sixteen input and sixteen output lines, arranged as eight-bit ports. Each port may be configured for handshake or free-running operation, and appears to the software as a single channel, communicating over an Inmos link at up to 20 Mbits/s.

The buffered signals can be conditioned by other boards via an industry-standard Signal Conditioning bus. Off-the-shelf signal conditioning boards include opto-isolators, Darlington drivers, solenoid drivers and relays. Industrial quality screw terminals are available for connection to plant.

Therange of I/O boards froms part of the Cambridge RISC strategy, which aims to provide a complete service for the development of real-time control systems based on the Inmos transputer. Complementary products already available from Cambridge RISC include processor boards, an EPROM board, gateways to VME and other standard buses, and fibre-optic industrial networking. All of these products are in Eurocard format which facilitates the construction of rugged control systems.

Software support for PC and VMEbus OS–9/68k host systems is available from Cambridge RISC. This consists of 3L's range of parallel compilers for C, Pascal, and Fortran. Complete turn-key systems can be built by using 3L's ROM configurer, blowing developed code into EPROM and running it on Cambridge RISC's hardware.

For further details contact:

Simon Roberts
Cambridge RISC Machines Limited
Unit 10
Cambridge Science Park
Milton Road
Cambridge CB4 4FG
United Kingdom

Tel: +44 223 420787
Fax: +44 223 420572

COMMERCIAL FORMAL METHODS AND TOOLS

Formal Systems (Europe) Ltd

Formal Systems Design and Development Inc., a US company which brings leading-edge academic research to the market-place, has just opened its European office in Oxford, to specialise in applications of the theory of concurrency. Its staff is drawn largely from former members of the Oxford University Programming Research Group, including a number of the original team responsible for the occam Transformation System, and several current members of the group serve as directors and consultants.

As well as developing software tools for supporting occam programming – watch future issues for product announcements! – the company offers collaboration and consultancy on all aspects of parallel systems and formal methods, with particular strengths in the formal development of very reliable concurrent systems. Short courses can also be arranged, either off-the-peg or tailored to clients' specific needs, in any of these areas.

For further details, request an information pack from:

Formal Systems (Europe) Ltd — Tel: +44 865 728460
Unit 7, The S.T.E.P. Centre — Fax: +44 865 793165
Osney Mead
Oxford OX2 0ES
United Kindgom

LINKEYE – AN INTERFACE FOR LINE-SCAN CAMERAS

Transputer Systems Group, SI, Oslo, Norway

The Transputer Systems Group at SI, Oslo, Norway has recently developed a new building block for industrial vision systems.

Linkeye digitises the signal from a line-scan camera and multiplexes it line by line onto a number of transputer-links. This means that any network of transputers can be utilized at the link end of the interface. The multiplexer will work on any number from one to sixteen links. At maximum the interface will handle 800 kpixels/s on each link (1.6 Mpixels/s if Inmos ever supply link adapters with overlapped ack.) With sixteen links this adds up to 12.8 Mpixels/s performance if your algorithms and transputers are able to swallow the data.

Linkeye has many features to support debugging, fault tolerance and general ease of use. Some of these are:

- ▷ control of reset and various modes of the interface by the first transputer
- ▷ event signalling of data overrun or new-line,
- ▷ lots of LEDs on front panel indicating state of the interface,
- ▷ electronic aid for adjustment of aperture and focus,
- ▷ adjustable gain and offset on video signal.

Linkeye should be well suited for applications with hard real-time demands like industrial inspection and quality control systems. We have developed the board in cooperation with SIUs Vision Systems Group and they will use it in their ongoing development of industrial inspection systems. We are interested in cooperating with and making Linkeye available to interested companies and research institutes.

Arne Sommerfelt — Tel: +47 2 45 20 10
Senter for Industriforskning — (direct) +47 2 45 27 97
(Centre for Industrial Research) — Fax: +47 2 45 20 40
Box 124 Blindern — sommerfelt@no.uninett.si
N-0314 Oslo
Norway

IMMEDIATE SOLUTION TO DATA STORAGE PROBLEMS

T2 Systems Ltd

Following the recent launch of their Paradise-1 SCSI/TRAM T2 Systems Ltd have released version 0·00 of their 'I-connect' software module for this TRAM. I-connect is so called because it allows the Paradise-1 to be used as a general purpose SCSI 'initiator' able to access any SCSI peripheral device.

I-connect has been designed with four main requirements in mind:

- ▹ It should be completely general purpose, able to execute any SCSI command.
- ▹ It should offer a simple well defined user interface that can easily be incorporated into any language or programming environment.
- ▹ It should provide a very high speed data transfer rate.
- ▹ Large data blocks should be transferred with a single command.

I-connect meets all these requirements giving a system which is easy to use, high performance, and inherently upgradeable. Applications for the Paradise-1/I-connect system include

- ▹ File Servers
- ▹ Image storage
- ▹ High speed tape backup systems
- ▹ Virtual Memory
- ▹ Database support

T2 Systems Ltd	Contact:	Patrick Pope (T2 Systems Ltd)
62 Longmead Avenue		+44 703 641276
Bishopstoke		Sandu Hellings (Rapid Silicon)
Eastleigh, Hants SO5 6ET		+44 494 442266
United Kingdom		

SPLASH DISPLAY SUBSYSTEM APPLICATION ACCELERATOR

Tektite Limited, Felixtowe

The Tektite SPLASH is an advanced applications accelerator and display system for the IBM PC, PC-AT, and compatible computers. The SPLASH implements the processing, drawing, and display functionality of a traditional workstation, while comfortably exceeding the performance of all but the most advanced workstation systems. SPLASH has the power needed to run advanced windowing and other display software. Typical SPLASH applications include computer-aided design, high performance windowing systems, publishing systems, and true-colour animation and simulation.

- ▹ Workstation quality one million pixel display system – resolutions up to 1152×900 at 8 bits/pixel.
- ▹ 17 MIP RISC processor (Inmos T425-17 transputer) with 2 Mbyte or 5 Mbyte total memory.
- ▹ Fully programmable display resolution, including interlaced or non-interlaced displays. Fully compatible with all common analogue monitors. Video master and video slave modes. Normal and tessellated (TV) syncs.

- ▹ Palette allows display of 256 colours from a selection of over 16 million.
- ▹ True colour mode (24 bits/pixel) allowing all 16 million colours to be displayed.
- ▹ Two standard 20 Mbits/s expansion channels for attaching additional processors for compute-intensive applications.
- ▹ Fully equipped for expansion into a multiprocessor system as either the system master or a slave. Multiple SPLASH boards can be installed in a single PC.
- ▹ IBM PX-XT half-card size with a standard PC-AT edge connector.
- ▹ Outstanding price-performance ratio.

In an IBM PC form factor, SPLASH provides a unique combination of Sun screen resolution, a 17 MIP general-purpose processor with 1 Mbyte or 4 Mbyte of workspace, and 24 bit/pixel (true colour) capability.

Typical non-interlaced display resolutions include $512 \times 512 \times 24$, $640 \times 480 \times 8$ and $1152 \times 900 \times 8$. All these resolutions can be selected under software control.

TT300-A models of SPLASH incorporate an Inmos G300-A colour controller. TT300-B models incorporate a G300-B with the extra features of gamma correction in 24 bit/pixel mode, tessellated sync pulses, and the ability to slave to external video. Tektite can upgrade TT300-A cards to the TT300-B model.

For more information call

Andrew Talbot	Tel: +44 394 672117
Tektite Limited	Telex: 987458 TKTITE G
PO Box 5	
Felixtowe	
Suffolk IP11 7LW	
United Kingdom	

THE YARC PROTRAN™ SYSTEM

a higher-performance PC/AT plug-in transputer system

Yarc Systems designs and manufactures co-processor systems for Macintosh, PC/AT and PS/2 personal computers, using AMD 29000 and Motorola 68020/68030 processors. At the second NATUG conference Yarc introduced its first parallel processing product: the PROTRAN™ system, which uses the INMOS T800 transputer.

The PROTRAN™ board is currently available for the AT bus. In addition to the C012 host interface, the board can accommodate a maximum of four transputers, operating at either 25 MHz or 20 MHz. A 30 MHz version of the board will be offered soon.

The first transputer (the root) can be equipped with 1, 2, 4, 8 or 16 Mbytes of RAM; the other three with 1, 2, 4 or 8 Mbytes. The most powerful configuration thus has four transputers with 40 Mbytes of RAM. Every transputer on the board has a full subsystem implementation, and can be individually reset and analyzed.

The PROTRAN™ system retains full compatibility with the INMOS B004 interface, and therefore runs all standard PC transputer software. In addition it features a high-speed interface mode that makes use of the string I/O instructions on the 286 processor. This high speed interface is *5 to 8 times faster* than the B004 interface, achieving a maximum sustained data rate of *over 1 Mbytes/sec*. This is close to the highest data-rate possible with the C012 link adapter chip.

To interconnect different transputers, the PROTRAN™ system combines all interconnection signals (including subsystem, reset and analyse) on a proprietary pin-field array. In production a pre-configured header would be used to determine the interconnections. In the development stage they can be made either with jumpers, jumper cables or wire wrapping. Inter-board connections are achieved with one or two standard flat cables, neatly positioned at one end of the boards.

The PROTRAN™ uses the same memory technology that Yarc developed for their RISC 29000 products. Wait states are only generated when needed, giving the transputer an external memory access between three and four cycles. Even with standard DRAM this makes the PROTRAN™ system between 10% and 25% faster than boards that use a fixed number of wait states. A very important additional advantage of this memory subsystem is its reliability. All PROTRAN™ boards are extensively tested at 60° Celsius prior to shipping.

Yarc offers a version of the Logical Systems C Toolkit that has been modified to take advantage of the high-speed host interface. A version of TopExpress Fortran will soon be available.

Although Yarc is a relatively new name in the transputer world, the people behind the PROTRAN™ System are not. The board was designed by Trevor Marshall, chief engineer of Yarc. Trevor originated the Definicon Transputer board, back in 1987. Bernt Roelofs, Yarc's parallel processing product manager, has a Master's degree from Twente University in the Netherlands, with over four years of transputer hardware and software experience.

Yarc is currently considering applying its Microchannel and Nubus design experience to developing transputer products for these platforms. We welcome any comments or suggestions.

For more information, contact:

Yarc Systems Corporation	Tel: +1 818 889 4388
27489 West Agoura Road	Fax: +1 818 889 2658
Agoura Hills, CA 91301	
United States of America	

REFERENCE

There is no list of new members in this issue of the newsletter because (at least at the time of going to press) it is intended that a consolidated list of members should be distributed as a separate publication in the same mailing.

BIBLIOGRAPHY UPDATE

Articles authored by INMOS personnel

Image compression using a discrete cosine transform image processor, D. J. Bailey and N. Birch, Electronic Engineering, July 1989.

Porting to the Transputer, Andy Hamilton, .EXE Magazine 4(3), August 1989.

OCCAM et le transputer: Le parallelisme au present, Jego, Bruno, Minis & Micros, N° 322/15 Mai 1989 (pp. 45–50).

Le Transputer et la qestion du temps reel (I), Jego, Bruno & Hersemeule, Richard, Minis & Micros, N° 327/18 Sept 1989.

Software simulation on the transputer, Ray Knagg, Microsystem Design, July/August, 1989.

The influence of VLSI technology on computer architecture, David May, Phil. Trans. R. Soc. Lond., A326, pp. 377–393 (1988).

Articles authored by others

Transputer arrays for solving partitioned systems of linear equations, Al-Turaigi, M., Afifi, M., El-Azhary, I., Excell, P., Int. J. Electronics, 1989, Vol. 66 (5), 789–800.

On implementing parallel GKS, Arnold, D. B. & Hinds, M. R., Computer Graphics Forum, 8(1), pp. 13–19, 1989.

Analogue I/O strategies for transputers, Barlow, M. I., Konnanov, P. & Burge, S. E., Microprocessors & Microsystems, 13(6), July/August 1989.

The Future of High Performance Computers in Science and Engineering, Bell, Gordon, Communications of the ACM, 32(9), September 1989.

Parallel Object System on Transputer-Based Architectures, Ciampolini, Anna; Coradi, Antonio; Leonardi, Letizia, Microprocessing and Microprogramming, 27, pp. 339–346, 1989.

An Environment for Developing Concurrent Software for Transputer-based Image Processing, Crookes, D.; Morrow, P. J.; Sharif, B.; McClatchey, I., Microprocessing and Microprogramming, 27, pp. 417–422, 1989.

An array processing language for transputer networks, Crookes, D.; Morrow, P. J.; Milligan, P.; Kilpatrick, P. L.; Scott, N. S., Parallel Computing, 8, pp. 141–148, 1989.

A Survey of Synchronization Methods for Parallel Computers, Dinning, Anne, Computer, July 1989, pp. 66–76.

Very-high-performance multiple-instruction multiple-data applications, Elliott, C. J., Phil. Trans. R. Soc. Lond. A326, pp. 471–479 (1988).

Parallel computing comes of age: supercomputer level parallel computations at Caltech, Fox, Geoffrey C., Concurrency: Practice and Experience, 1(1), 63–103, September 1989.

Transputer-based implementation of the Radon transform, Hall, G.; Terrell, T. J.; Senior, J. M.; Murphy, L. M., Microprocessors and Microsystems, 13(7), September 1989.

Phase 1 of the Development and application of a low cost high performance multiprocessor machine, Harp, J. G.; Jesshope, C. R.; Muntean, T.; Whitby-Strevens, C., Esprit '86: Results and Achievements, Directorate General XIII (Editors), Elsevier Science Publishers, pp. 551–562, 1987.

The use of a methodology in control applications of Transputers, Hasnain, S. B.; Linkens, D. A., IEE Colloquium on Recent Advances in Parallel Processing for Control, 7/1–10, 1988.

Real-Time System Implementation – the Transputer and Occam Alternative, Hull, M. Elizabeth C.; Zarea-Aliabadi, Adib, Microprocessing and Microprogramming, 26, pp. 77–84, 1989.

Transputers and switches as objects in OCCAM, Jesshope, Chris, Parallel Computing, 8, pp. 19–30, 1988.

Transputer based digital signal processing unit for a 3-D vision system, Jokitalo, P.; *et al.*, Microprocessing and Microprogramming, 27, pp. 143–146, 1989.

A Flexible Transputer Network for Numerical Applications, Luo, J.; Bruggeman, F; Reijns, G. L., Microprocessing and Microprogramming, 27, pp. 405–412, 1989.

Study of Dynamic Routing Algorithms using a High-Speed multiprocessor, simulator, Nichols, S. J.; Clarke, R. T.; Mars, P., Second IEE National Conference on Telecommunications, pp. 161–166, 1989.

The Transputer: T414 Instruction Set, Nicoud, Jean-Daniel; Tyrrell, Andrew Martin, IEEE Micro, pp. 60–74, June 1989.

TLS: a System for Building and Controlling Transputer Networks, van Peursem, M. A.; Knoppers, P.; van der Goor, A. J., Microprocessing and Microprogramming, 27, pp. 739–746, 1989.

Occam II, Pountain, Dick, Byte, pp. 279–284, October 1989.

Parallel Processing in Machine Automation, Rautiola, Kyosti; Nayha, Tuomo; Kaarela, Kari, Microprocessing and Microprogramming, 27, pp. 723–730, 1989.

Synchronous Operations as First-class Values, Reppy, J. H., Proceedings of the SIGPLAN '88 Conference on Programming Language Design and Implementation, Atlanta, Georgia, June 22–24, 1988 (pp. 250–259).

Acceleration of Circuit Simulation on a Parallel Transputer Workstation, Reus, Thomas, Microprocessing and Microprogramming, 27, pp. 731–738, 1989.

Translating from PARLOG to occam 2: a methodology, Scott, Robert B.; Trehan, Rajiv, Concurrency: Practice and Experience, 1(1), pp. 105–134, September 1989.

Hardware voter for fault-tolerant transputer systems, Standeven, J., Colley, M. J., Lyons, D. M., Microprocessors and Microsystems, Vol. 13 (9), November 1989, 588–595.

Scheduling and Parallel Operations on the Transputer, Tyrrell, A. M.; Nicoud, J. D., Microprocessing and Microprogramming, 27, pp. 175–185, 1989.

Computers for Symbolic Processing, Wah, Benjamin W.; Lowrie, Matthew B.; Li, Guo-Jie, Proceedings of the IEEE, 77(4), April 1989 (pp. 509–540).

Parallel tracks to standard processing, Watts, Susan, New Scientist, 12 August 1989 (pp. 44–47).

Textual and chemical information processing using parallel computer hardware, Willett, Peter, Journal of Information Science, 15, pp. 223–236, 1989.

In languages other than English

Occam: maitriser le parallelisme, Cosnuau, Alain, Micro-Systemes, Juillet/Aout, pp. 161–165, 1989.

Le transputer passe du laboratoire a l'industrie, Catier, Eric, Electronique Industrielle, Nº 163/01–06–1989 (pp. 42–46).

386 ou Transputer: Le choix d'une solution pour le calcul scientifique, Gravier, Gilles, Micro-Systemes, Oct 1989 (pp. 119–122).

Microprocesseurs: La grande glisse en parallele, No author given, Electroniques, Techniques et Industries, Nº 69, 19 Mai 1989 (pp. 30–41).

Prozessorientierte Programmierung: Occam, Baumann, Ruedi, Elektroniker, Nr. 6/1989 (pp. 97–103).

OCCAM 2 und ADA, Schabernack, J.; Schutt, A., Informatik Spectrum, 12(1), pp. 3–18, 1989.

Occam, Husken, V., Informatik Spectrum, 11(6), pp. 325–6, 1989.

Jezyk Occam (I & II), Blaszcazak, Slawomir, Informatyka, Nr. 7 & 8, 1988.

CONTACTS FOR RELATED GROUPS

Australian Transputer and Occam User Group

John Hulskamp
Department of Communication
and Electrical Engineering
Royal Melbourne Institute of Technology
GPO Box 2476V
Melbourne 3001 Australia

Tel: +61 3 660 2453
Fax: +61 3 662 1060
rcojh@au.oz.rmit.gecko

French Transputer Users Working Group

Traian Muntean
IMAG-LGI
b.p. 68
38402 St Martin d'Heres CEDEX
France

traian@fr.imag.imag

Deutschen Occam-Interessengemeinschaft der Transputeranwender

DO IT can be contacted through its secretary:

Heinz Ebert
Im Heidigen 3
5206 Neunkirchen-Seelscheid 2
West Germany

The president is:	and vice presidents are:	
Joachim Stender	Frank Heinemann	Peter Eckelmann
c/o Brainware GmbH	c/o Fraunhofer-Institute	c/o Inmos GmbH
Gustav-Meyer-Allee 25	Kleiststraße 23–26	Danziger Straße 2
1000 Berlin 65	1000 Berlin 30	8057 Eching b. München
West Germany	West Germany	West Germany
		+49 89 319 10 28

Occam User Group Japan

Contact the Secretary:
Mr Kazuto Matsui
Technical Marketing, INMOS Division
SGS-Thomson Microelectronics K.K.
4F Nisseki-Takanawa Building 2-18-10
Takanawa Minato-ku Tokyo 108
Japan
Tel: +81 3 280-4125
Fax: +81 3 280-4131

The chairman is:
Prof. Tosiyasu L. Kunii
Department of Information Science
University of Tokyo
7-3-1 Hongo, Bunkyo-ku
Tokyo 113
Japan
+81 3 505 2840

Latin American Transputer Users' Group

For further information, contact the Chairman:
Rafael D. Lins
Av. Dr José Rufino 656
Estância
50.781 – Recife – PE
Brazil
Tel: +55 81 251 0713
Fax: +55 81 326 4880

New Zealand Transputer Users' Group

The NZTUG is still only a small organisation. The Chairman in Bob Hogson, Professor of Production Technology at Massey University. Contact the secretary and treasurer:
Dr Ian Graham
Department of Computer Science
University of Waikato
Private Bag
Hamilton, New Zealand
Tel: +64 71562889 x8204
Fax: +64 71384066
CSNet: i.graham@waikato.ac.nz
JANet: i.graham@nz.ac.waikato

Swedish Transputer User Group

The purpose of STUG is to act as an information exchange forum for transputer users in Sweden, and to stimulate discussion concerning related areas such as parallel programming and parallel processor systems. STUG arranges seminars and publishes a newsletter, supported by Gösta Bäckström AB, who represent INMOS in Sweden.
Martin Törngren
Maskinelement
Kungl. Tekniska Högskolan
100 44 Stockholm, Sweden
Tel: +46-8-790 7849
Fax: +46-8-723 1730
martin@se.kth.damek

North American Transputer Users Group

NATUG have a permanent organization with a committee of about fifteen members, which receives secretarial support from Inmos Colorado Springs. Contacts for this group are: the Chair, Dyke Stiles; the Secretary of the North American Transputer

Users Group, care of Mark Hopkins at Inmos Colorado; and the local agent for newlsetter submissions, who is Lyle Bingham. Their addresses appear on page 114.

A WORD ABOUT NAMES AND NUMBERS

I have tried to be reasonably consistent about addresses and telephone numbers in the newsletter. Human fallibility excepted, the telephone numbers are all given in the international form: so for example a UK caller should replace the +44 of my number by an initial nought, and in the USA you would just drop the +1 from Lyle Bingham's number.

Would that electronic mail was as simple! Again I have tried to be reasonably consistent: UK addresses are quoted big-end first, but in other parts of the world `geraint.jones@uk.ac.oxford.prg` for example, would be given little-end first as `geraint.jones@prg.oxford.ac.uk` and in the UK they prefer American addresses like `csa@adam.byu.edu` the other way, in this case as `csa@usa.edu.byu.adam`. If you can tell whether you need to reverse any address from this newsletter, then you are an expert; but if you cannot, I am afraid you will need the help of an expert.

I have been told that if you are at a BITNET site, turning a big-endian address around does not work for all UK addresses, and in particular that it does not work for addresses at `uk.co.inmos`. It ought to be the case that all UK commercial domain addresses are known at Canterbury – `uk.ac.ukc` – so you may be able to render, for example `oug@uk.co.inmos`, as `oug%uk.co.inmos@ukc.ac.uk`. *gj*

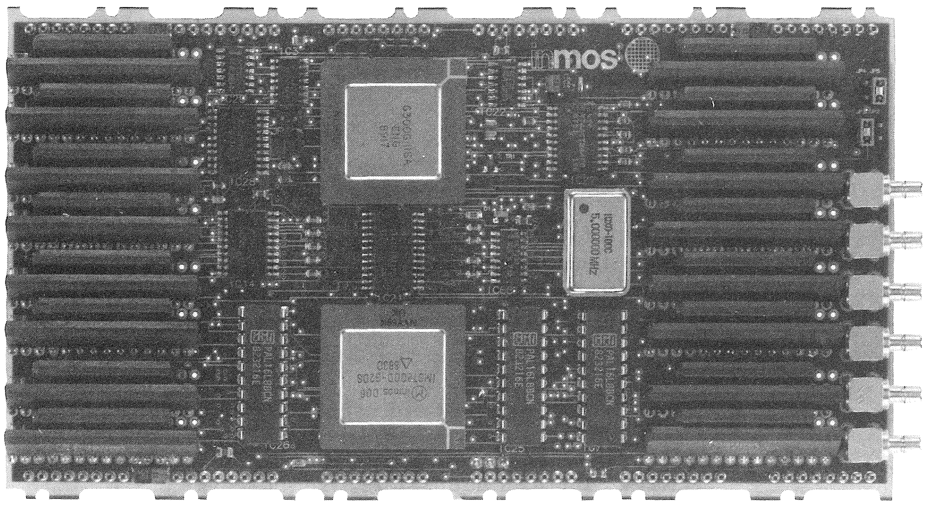


Figure 28: IMS B419 – a size 6 G300 based graphics TRAM which carries a T800-20, 2 Mbytes DRAM and 2 Mbytes VRAM.

SPECIAL INTEREST GROUP CHAIRMEN

Artificial intelligence

Note a change of chairman

Joachim Stender
c/o Brainware GmbH
Gustav-Meyer-Allee 25
1000 Berlin 65
West Germany

Image processing and vision

Neil Carmichael
KSEPL – Shell Research
Volmerlaan 6
2288 GD Rijswijk ZH
The Netherlands
Tel: +31 70 11 39 11
Fax: +31 70 11 39 10

Environments

Gordon Manson
Department of Computer Science
University of Sheffield
Sheffield S10 2TN
United Kingdom
+44 742 768555 x5580

Education and training

Roger Peel
Department of Electrical Engineering
University of Surrey
Guildford
Surrey GU2 5XH
United Kingdom
+44 483 509284
roger@uk.ac.surrey.ee

Formal methods

Bob Stallard +44 25 672 3911
Racal Milgo Ltd
Bartley House
Station Road
Hook
Hants RG29 9PE
United Kingdom

Numerical methods

Derek Paddon
Department of Computer Science
University of Bristol
University Walk
Bristol BS8 1TR
United Kingdom
+44 272 303030 x4336
derek@uk.ac.bristol.compsci

Hardware

Denis Nicole
University of Southampton
Department of Electronics
and Computer Science
The University
Highfield
Southampton SO9 5NH
United Kingdom
+44 703 787167
dan@uk.ac.soton.ecs

Graphical program development tools

Mike Roberts
Centre for Information Engineering
City University
Northampton Square
London EC1V 0HB
United Kingdom
M.ROBERTS@uk.ac.city

NATUG STEERING COMMITTEE

Dyke Stiles
Electrical Engineering Department
Utah State University
Logan, UT 84322-4120
Chair
+1 801 750 2806
dyke@opus.ee.usu.edu

Mark Hopkins
INMOS Corporation
PO Box 16000
Colorado Springs, CO 80935-6000
Secretary
+1 719 630 4000

Lyle Bingham
Computer Systems Architects
950 N. University Avenue
Provo, UT 84604
Newsletter contributions
+1 801 374 2300
csa@adam.byu.edu

Jim Favenesi +1 205 837 5282
c/o SPARTA, Inc.
4901 Corporate Drive
Huntsville, AL 35805

Jim Newhouse
FMC Advanced Systems Center
1300 South Second Street
Minneapolis, MN
+1 612 337 3242

David L. Fielding
Cornell University
265 Olin Hall
Ithaca, NY 14853
+1 607 255 8686
fielding@tcgould.tn.cornell.edu

Colin Whitby-Strevens
INMOS Limited
1000, Aztec West
Almondsbury
Bristol BS12 4SQ
United Kingdom
+44 454 611500
colin@inmos.co.uk

Linda Pollard +1 503 222 7080
Regis McKenna Inc.
220 NW 2nd, #1150
Portland, OR 97209

Gerald C. Johns
Computer Systems Laboratory
Washington University
724 S. Euclid Avenue
St. Louis, MO 63110
+1 314 362 3123
gerald@wuibc.wash.edu

Gordon Harp +44 684 894824
RSRE jgh@rsre.mod.uk
St Andrews Road
Great Malvern
Worcs WR14 3PS
United Kingdom

Ernest Miller
Computer Science Dept.
East Stroudsburg University
East Stroudsburg, PA 18301
+1 717 424 3447

John Board
Electrical Engineering Dept.
Duke University
Durham, NC 27706
+1 919 684 3123
jab@dukee.egr.duke.edu

Paul Smith
University of California at San Diego
Center for Research Language
CRL-C-008
La Jolla, CA 92093
+1 619 534 2695
Paul@amos.VESD.edu
PSSmith@UCSD.bitnet

Gerd Beckmann
Rensselaer Polytechnic Institute
Associate Director
Center for Manufacturing
110 8th Street
Troy, NY 12189
+1 518 276 6010

INFORMAL OCCAM USER GROUP COMMITTEE

Peter Welch
Computing Laboratory
The University
Canterbury
Kent CT2 7NF
Chairman
+44 227 764000 x3629
phw@uk.ac.ukc

André Bakkers
University of Twente
PB 217
7500 AE Enschede
The Netherlands
+31 53 892790
elbscbks@henut5.earn

Richard Beton
Plessey Electronic Systems Research Ltd
Roke Manor
Romsey
Hants SO5 0ZN
+44 794 833433

Gordon Harp
RSRE
St Andrews Road
Great Malvern
Worcs WR14 3PS
0684 894824
jgh@uk.mod.rsre

Geraint Jones
Programming Research Group
Oxford University Computing Laboratory
11 Keble Road
Oxford OX1 3QD
Newsletter editor
+44 865 273851
geraint.jones@uk.ac.oxford.prg

Jon Kerridge
Department of Computer Science
University of Sheffield
Sheffield S10 2TN
+44 742 768555 x5580
acljmk@uk.ac.sheff.primea

Derek Paddon
Department of Computer Science
University of Bristol
University Walk
Bristol BS8 1TR
+44 272 303030 x4336
derek@uk.ac.bristol.compsci

Roger Peel
Department of Electrical Engineering
University of Surrey
Guildford
Surrey GU2 5XH
+44 483 509284
roger@uk.ac.surrey.ee

Michael Poole
Inmos Limited
1000 Aztec West
Almondsbury
Bristol BS12 4SQ
Secretary
+44 454 616616
oug@uk.co.inmos

Simon Turner
MEiKO Limited
650 Aztec West
Almondsbury
Bristol BS12 4SD
+44 454 616171

Stephen Turner
Department of Computer Science
University of Exeter
Prince of Wales Road
Exeter EX4 4PT
+44 392 264048

Colin Upstill
Plessey Electronic Systems Research Ltd
Roke Manor
Romsey
Hants SO5 0ZN
+44 794 833339

Hugh Webber
RSRE
St Andrews Road
Great Malvern
Worcs WR14 3PS
Program exchange
+44 684 894728
hcw@uk.mod.rsre

John Wexler
Edinburgh University Computing Service
The King's Buildings
Edinburgh EH9 3JZ
+44 31 667 1081 x2635
J.Wexler@uk.ac.edinburgh

. . . continued from front cover

SPECIAL INTEREST GROUPS 18
- Education and training 18
- Hardware 19
- Formal methods 20
- Graphical program development tools 21

TECHNICAL CONTRIBUTIONS 22
- Safety first 22
- David May on Draft Military Standard 00–55 27
- An optimal topology for multicomputer systems 31
- An experiment in parallelizing edge-detection 39
- Really efficient multiple buffering in occam 42
- Two implementations of semaphores in occam 49
- Measuring the busyness of a transputer 57
- Improvements(?) to occam 64
- Transputer and occam based control systems 66
- Getting along without workstations 71
- De vermibus 73

REVIEWS 74
- Transputer development system 74
- Communicating process architecture 76
- Transputer technical notes 76
- Programming in occam2 77
- Digital signal processing 78
- The Helios operating system 80
- Parallel processing: technology and applications 81

PRODUCTS, SERVICES AND ANNOUNCEMENTS 82
- Inmos marketing update 82
- Occam promotion initiative 84
- Occam 2 and transputer engineering course 85
- CODE – a revolutionary platform for learning occam 86
- Parallel processing and the transputer – a video tape 88
- MEGA-Link transputer boards 89
- The Topexpress transputer libraries 91
- Concurrency: Practice and Experience 94
- Commercial parallel products use EXPRESS 96
- VideoPaint software using transputer image processing 97
- Meiko In-Sun Computing Surface hardware 98
- M212 TRAM based disc subsystem 99
- IF/Prolog on the transputer 100
- FAST9 upgrade maintains Quintek edge 102
- Industrial I/O for transputer systems 103
- Commercial formal methods and tools 103
- Linkeye – an interface for line-scan cameras 104
- Immediate solution to data storage problems 105
- SPLASH display subsystem application accelerator 105
- The Yarc PROTRAN™ system 106

REFERENCE 107

·O·U·G·

Facilities to prepare this newsletter were provided by Oxford University Computing Laboratory; printed and distributed on behalf of the Occam User Group courtesy of INMOS Limited.