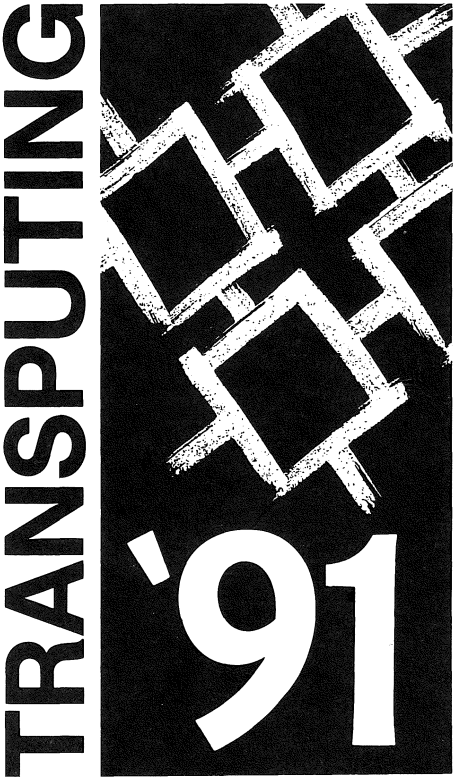
·O·U·G·

occam® user group · newsletter

for all users of occam and the transputer

№ 14 January 1991

Contents



continued on back cover

EDITORIAL

This coming April all the various user groups associated with this *Newsletter* are organising a joint meeting. This is quite a venture – for organisations as informal as ours, our own local meetings are often quite an undertaking – but it promises well. There is a full programme of technical presentations arranged for the middle of the week, as well as tutorials and workshops for the beginning and end of the week, and there will be an exhibition running alongside the meeting. Details appear on page 5, and in leaflets which you will doubtless find in mail with this *Newsletter*.

The bulk of the technical papers in this issue would appear to be devoted to familiar topics: the discussion of semaphores and of `ALT` follow on from earlier contributions on the same subjects. I am glad to be able to draw your attention to some new threads of discussion, though! It is nice to see a hardware paper once in a while.

Through technical difficulties – and not least for fear of injuring too many postmen – I have had to hold back one paper reacting to the proposed alterations to the occam language (see page 27). Now that Geoff Barrett's paper has appeared in the proceedings of the latest (York) OUG technical meeting, I expect that there will be more of us wanting to join in the discussion that he invites, and I urge you to put your thoughts in order and send them to this *Newsletter*.

Let me finally draw your attention to some new addresses in the back of the *Newsletter* (see page 102) giving contacts for related groups. In notices of forthcoming meetings you will also find contacts in Poland which were new to me.

... and farewell

It's been great fun, honestly – well, it has for me anyway. However, a few weeks ago the men in suits paid me a visit, and after they left I found there was a loaded revolver on my desk. This is my last *Newsletter*: after ten years in the post – well, 10 (in base two) years – I think perhaps I would make a really good back-seat driver. In years to come this will be talked of as the end of an era, but I firmly believe that Stephen Turner is the candidate to unite the party.

Geraint Jones, 14 December 1990

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 easier for the editor if you were able to submit material (particularly longer contributions) by electronic mail to `steve@uk.ac.exeter.cs`; or on an IBM PC compatible floppy disk to the editor at the address below. LaTeX format is preferred, but unformatted ASCII files are also acceptable. If these methods cannot be used, clean camera-ready copy should be sent to 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 originals to the A5 page size of this booklet. (This means: make sure your type is not too small!) Pictures are welcome as black-and-white prints, and will be subjected to the same reduction in size.

Copy for the next edition must arrive by *Friday, 17th May 1991.*

Dr Stephen Turner
Department of Computer Science
University of Exeter
Prince of Wales Road
Exeter EX4 4PT
United Kingdom

steve@uk.ac.exeter.cs
Tel: +44 392 264048
Fax: +44 392 264067
Telex: 42894 EXUNIV G

PROCEEDINGS OF GROUP MEETINGS

Proceedings of the meetings of the *occam user group*, the *North American transputer user group*, and other sister groups are now published for the groups by the IOS Press, and should be ordered directly from the publishers.

The following volumes are available: in the Occam User Group Progress Reports series (prices in Dutch guilders/US dollar prices in the USA and Canada only)

- OUG–7, 14–16 September 1987, Grenoble, *Parallel programming of transputer based machines*, ed. Traian Muntean (pp. x+480, fl.230/$110).
- OUG–8, 27–29 March 1988, Sheffield, *Developments using occam*, ed. Jon Kerridge (pp. vii+214, fl.92/$50).
- OUG–9, 19–21 September 1988, Southampton, *Occam and the transputer – research and applications*, ed. Charlie Askew (pp. vii+176, fl.92/$50).
- OUG–10, 3–5 April 1989, Enschede, *Applying transputer based parallel machines*, ed. André Bakkers (pp. viii+318, fl.130/$65).
- OUG–11, 25–26 September 1989, Edinburgh, *Developing transputer applications*, ed. John Wexler (pp. x+206, fl.115/$55).
- OUG–12, 2–4 April 1990, Exeter, *Tools and techniques for transputer applications*, ed. Stephen J. Turner (pp. vii+244, fl.130/$69).
- OUG–13, 18–20 September 1990, York, *Real-time systems with transputers*, ed. Hussein Zedan (pp. viii+351, fl.130/$69).

and in the NATUG Progress Reports series

- NATUG–1, 5–6 April 1989, Salt Lake City, Utah, ed. G. S. Stiles (pp. 166, fl.120/$60).
- NATUG–2, 18-19 October 1989, Durham, North Carolina, ed. John A. Board, Jr. (pp. 462, fl.230/$115).
- NATUG–3, 26–27 April 1990, Santa Clara, California, ed. Alan Wagner (pp. x+352, fl.190/$95).

- NATUG–4, 10–11 October 1990, Ithaca, New York, ed. David L. Fielding (pp. viii+250, fl.130/$69).

IOS also publish the most recent proceedings of others of the user groups,

- Proceedings of the 3rd transputer/occam international conference, Tokyo 1990, ed. Tosiyasu L. Kunii and David May (pp. x+308, fl.170/$89).
- ATOUG–3, Proceedings of the 3rd Australian transputer and occam user group conference, Melbourne, 1990, ed. T. Bossomaier, T. Hintz and J. Hulskamp, (approximately 200 pages, fl.120/$60).

and the proceedings of the international conferences on the applications of transputers organised under the auspices of the UK SERC/DTI 'Transputer Initiative'.

They are available from bookshops or direct from the publisher, either individually or on continuation orders.

IOS — Fax: +31 20 22 60 55
Van Diemenstraat 94
1013 CN, Amsterdam
The Netherlands

or in the USA and Canada

IOS Press Inc. — Fax: +1 703 250 47 05
Postal Drawer 10558
Burke, VA 22009-0558
United States of America

or on the US West Coast

Computer Literacy Bookshops, Inc. — Fax: +1 408 435 0689
2590 N. First Street
San Jose, CA 95131
United States of America

or in Japan

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

SIG publications

The proceedings of last year summer's international conference organised by the Artificial Intelligence SIG have appeared as the first volume of the *Wiley Communicating Process Architecture Series*, series editor David May.

- *Parallel Processing and Artificial Intelligence*, ed. Mike Reeve and Steven Ericsson Zenith, (pp. xvi+291; £29·95); John Wiley & Sons, 1989.

The proceedings of the OUG AI SIG's second meeting are also available from IOS, through bookshops and the agents listed above.

- Proceedings of the 2nd International Conference of the OUG Artificial Intelligence SIG (London, UK, 1st October 1990), ed. Joachim Stender and Tom Addis (pp. viii+252, fl.120/$65).

TRANSPUTING '91

THE WORLD'S LEADING CONFERENCE ON MULTIPROCESSING

Organized by

The World Transputer User Group Committee

Sponsored by

SGS-Thomson Microelectronics, ESPRIT,
Parsytec, Paracom, Parasoft, Transtech Devices,
JMI Software, Yarc Systems Corporation,
Distributed Software Ltd, 3L and others.

Transputers are the revolutionary microprocessors for multiprocessing which have outsold all other 32-bit RISC architectures. They are fuelling explosive growth in science and engineering by extending the ability to solve complex problems fast, elegantly and at reasonable cost.

If you're investigating the use of parallel or multiprocessing in any application, be sure not to miss TRANSPUTING '91. This international event includes five days of selected presentations, tutorials and workshops on the powerful concepts of parallel computing based upon communicating process architectures, and will feature a full technical disclosure of the next generation transputer, codenamed H1.

Invited speakers include representatives from Fujitsu Labs, Harvard University, and IBM's T. J. Watson Research Centre. A concurrent exhibition will support the conference themes of performance and scalability, porting existing systems, parallelization paradigms, formal methods and security, programming languages, support environments, standards, and applications.

Applications discussed and demonstrated at TRANSPUTING '91 will include embedded real-time systems, workstations, supercomputers, laser printers, disk arrays, image processing, global positioning by satellite, artificial intelligence, databases, and the testing of scientific and mathematical theories.

So... plan to find out what transputing is all about. See state-of-the-art application demonstrations and product exhibitions from around the world. Gain an in-depth understanding of the new software and hardware technologies enabled by the transputer. Discover H1. Learn why transputers make sense – today and for the future.

Endorsers of this conference include:

- ▷ ESPRIT
- ▷ The Institute of Physics
- ▷ The Institute of Electrical Engineers
- ▷ The British Computer Society
- ▷ The Parallel Processing Connection
- ▷ The New Zealand Computer Society

- The Oregon Advanced Computing Institute
- The Edinburgh Parallel Computing Center

and others

Special invited speakers

MR. RIICHIROU TAKE AND MR. YASUO NOGUCHI Researchers, Artificial Intelligence Laboratory, Fujitsu Laboratories Ltd.
An architecture for parallel database computing The paper proposes an MIMD computer aimed at database processing and general purpose applications. An experimental system has been made using 32 T800s as processing nodes and 32 for Dragon Net, a binary n-cube structured network which can optimally process all-node-to-all-node communication based on a simple rule.

Mr. Take and Mr. Noguchi are researching into concurrency control for distributed database management systems, and parallel architecture and algorithms for speeding up database operations. Mr. Take has a degree in Mathematical Engineering and Information Physics from the University of Tokyo. Mr. Noguchi has a Masters degree in Pharmaceutical Science from the University of Tokyo.

MR. JONAH MCLEOD Editor, Electronics
An industry-wide perspective on parallel and multiprocessing Jonah McLeod has been with Electronics and Electronic Design since 1979, writing on the whole spectrum of electronic equipment. Prior to this he managed the Apple Computer and Intel Corp. accounts at Regis McKenna PR, before which he was west coast editor of Computer Design. He has written several books on computers, computer peripherals and CAD, and has a Batchelor of Science from the University of Texas.

DR DAVID MAY Manager, Transputer Architecture and Development, Inmos
Towards general purpose parallel computers David May is currently working at Inmos on the architecture of a new product range for introduction in 1994. He graduated from Cambridge University with a degree in Computer Science and has published about 45 papers and 15 patents. He is Visiting Professor of Engineering Design at Oxford University and has an honorary DSc from Southampton University for his contributions to the development of parallel computing.

PROFESSOR LESLIE VALIANT Harvard University
Bulk-synchrony: a bridging model for parallel computation The success of the Von Neumann model of sequential computation is attributed to the fact that it is an efficient bridge between software and hardware. This paper argues that an analogous bridge between software and hardware is required for parallel computation if it is to become more widely used, introduces the bulk-synchronous parallel model as a candidate for this role, and supports the suggestion by giving a number of results that quantify its efficiency.

Professor Valiant is currently Gordon McKay Professor of Computer Science and Applied Mathematics at Harvard University. His current research interests are computational complexity, machine learning, and the theory of parallel algorithms and architectures. In 1986 he received the Navanlinna prize for theory of information processing from the International Mathematical Union.

MR. DENNIS G. SHEA Modular Microsystems Group Manager, IBM T. J. Watson Research Center

IBM Victor V256 Victor is a family of partitionable transputer-based multiprocessors that have been designed at IBM Research to provide researchers with a platform for experimentation in the area of highly parallel message passing MIMD machines with distributed memory. Applications running on Victor cover a variety of scientific and engineering topics such as VLSI waveform relaxation based circuit simulation and Quantum Monte Carlo simulations for exploration of high temperature superconductors.

Dennis Shea's research focuses on the design and development of high performance distributed memory parallel processors and their use in solving real applications. He is a Ph.D. candidate in the Department of Computer and Information Science at the University of Pennsylvania, and has a Master's degree in Computer Systems from Florida Atlantic University.

H1 transputer disclosure

IAN PEARSON Director of Technology, Inmos Wednesday afternoon's session will feature a complete technical disclosure of Inmos' next generation transputer, codenamed H1.

The key features of H1 are a high performance pipelined superscalar processor and major support for multiprocessing applications. Peak performance will be more than 150 MIPS and 20 MFLOPS. Inmos, a member of the SGS-Thomson Microelectronics group, is the recognized leader in parallel and multiprocessing, and H1 represents a major advancement in parallel computing and high speed communications.

The transputer architecture is unique in providing hardware support for process scheduling and specific instructions for interprocess communication. As the computational loads on embedded processors increase, the ability to produce scalable multiprocessor systems is crucial. H1 represents a significant advance in the transputer's already proven capabilities in parallel and multiprocessing.

The design goals for H1 were to enhance the transputer's position as the premier multiprocessing microprocessor, and to establish a new standard in single processor performance, while maintaining compatibility with existing transputer products. This session will disclose the means by which these have been achieved, and time has been scheduled for Mr. Pearson to field questions from the audience.

Calendar of events

Monday April 22

REGISTRATION

TUTORIALS on the fundamental principles underlying transputer technologies and various design paradigms for exploiting them.

- ▷ Communicating process architectures and transputer overview
- ▷ Designing parallel – going sequential
- ▷ Parallelizing existing code
- ▷ Transputer programming environments

- ▷ Practical use of formal methods
- ▷ Mixed language programming on the transputer

EVENING RECEPTION

Tuesday April 23

TECHNICAL SESSION Morning and afternoon: papers for these sessions are currently being refereed.

Wednesday April 24

TECHNICAL SESSION Morning

H1 TECHNICAL DISCLOSURE

CONFERENCE DINNER

Thursday April 25

TECHNICAL SESSION Morning and afternoon.

AWARD DINNER

Friday April 26

WORKSHOPS on transputer applications and advanced techniques

- ▷ Embedded real-time control systems
- ▷ Real-time kernels for C programmers
- ▷ Designing parallel – a hands-on workshop
- ▷ Image processing – a hands-on workshop
- ▷ Scientific computing
- ▷ Office automation applications
- ▷ Communications applications
- ▷ Artificial intelligence
- ▷ Helios operating system – a hands-on workshop
- ▷ Logic programming on transputers

Registration, accommodation and travel

Main conference 23–25 April including three nights hotel accommodation $650 US
Main conference 23–25 April without hotel accommodation $450 US
Extra night's lodging $100 US

These hotel rooms have been made available at specially discounted rates and will sleep up to four occupants.

Student discount – deduct $225 US
Early registration before 31st January 1991 – deduct $100 US
(Only one discount allowable per person)
One half-day tutorial/workshop on 22 or 26 April $100 US
Two half-day tutorials/workshops on 22 or 26 April $150 US

International airfare available through American Express +31 20 520 77 77 or Brian Raines (number given below).

Registration forms will be circulated in December but early registration is welcome. Forms are available from Brian Raines (below).

Some exhibitors booths are still available.

All enquiries about the above should be directed to

Brian Raines
Executive Meeting Management
PO Box 434
Camp Hill, PA 17001
United States of America

Tel: +1 717 731 9295
Fax: +1 717 731 9295 and
push ⋆ after tone

FORTHCOMING

WORKING SEMINAR ON PARALLEL COMPUTING AND TRANSPUTERS

10 am – 4 pm, 17th January 1991
Tamka str 38, 1st floor, Warsaw, Poland

Organisers

Systems Research Institute, Warsaw, Poland
Microvex Ltd, Warsaw Poland
Quintek Ltd, Bristol, United Kingdom
Katal Co, Warsaw, Poland
(Participation of other foreign firms is also expected)

The aim of this seminar is to bring together researchers, practitioners, and business people active in Poland in the field of parallel computing and making use of the transputer technology.

This will be the second meeting of this kind. The previous one was organised by the Systems Research Institute and Microvex Ltd and held in autumn 1988 in Warsaw. There were about forty participants.

We hope that the seminar will give all the participants a good opportunity to exchange, disseminate or collect information on the subject.

A preliminary programme includes:

- ▷ concise presentations of recent activities of the seminar organizers in the field of parallel computing and transputers,
- ▷ presentations prepared by the participants,
- ▷ demonstrations of transputer hardware, software and applications,
- ▷ discussion.

A call is made to all interested persons and institutions to prepare short (approximately ten minute long) presentations reporting scientific results, scientific or application projects, problems to be solved or any other matter related to the subject of the seminar. The title and an abstract of the presentation should be sent to the

contact person, dr inż. Leon Słomiński, at the address below, before 10th January 1991.

The seminar room is conveniently located in the centre of Warsaw. It can be easily reached from the Central Railway Station in 15 min by bus 175 and a short walk.

dr inż. Leon Słomiński
Systems Research Institute
PAS
01–447 Warsaw
Newelska str 6
Poland

Tel: +48 22 36 41 58
Fax: +48 22 37 27 72
Tlx: 812 397

WORKSHOP ON ABSTRACT MACHINE MODELS FOR HIGHLY PARALLEL COMPUTERS

Organised by the British Computer Society Parallel Processing Specialist Group
University of Leeds, United Kingdom, 25–27 March 1991

Call for papers

Aim of the workshop

The workshop aims to provide a forum for the discussion of abstract machine models for highly parallel computers. Particular attention will be given to the specification, implementation and application of such models and to the identification of key issues for future research.

Workshop format

A number of invited speakers will give keynote presentations. Working groups will focus on major issues identified through position papers. The main purpose of the workshop is to provide a forum for discussion and as many participants as possible will be given the opportunity to present a position paper. The workshop will be restricted to 60 delegates and it is expected that all participants will be active researchers in relevant fields. In the event that more than 60 applications are received, selection will be on the basis of position papers submitted.

Invited speakers include:

- Dr J. P. Banatre, IRISIA;
- Prof. C. Jesshope, Surrey University;
- Dr D. May, INMOS Ltd;
- Prof. R. Milner, Edinburgh University;
- Dr D. Nicole, Southampton University;
- Prof. E. Odijk, Philips Research;
- Prof. S. Peyton Jones, Glasgow University;
- Prof. I. Watson, Manchester University.

Position papers

Potential participants should submit a position paper or extended abstract of around 1500 words indicating the current direction of their research. The position papers of all participants will be made available at the workshop. It is hoped that fuller versions of selected papers will be published more formally.

Suitable topics for position papers include, but are not limited to:

- ▷ low level abstract machine models,
- ▷ impact of machine models on operating system design,
- ▷ models for general purpose parallel computing,
- ▷ performance models,
- ▷ high level programming models,
- ▷ parallel system architectures,
- ▷ application-specific language interfaces,
- ▷ hardware support for abstract machine models,
- ▷ portable distribution runtime environments,
- ▷ open standards.

Position papers should be submitted no later than 31st January 1991 to:

Mrs J. A. Thursby
Abstract Machine Workshop Secretary
School of Computer Studies
The University
Leeds LS2 9JT
United Kingdom

Cost

The basic cost of the workshop will be approximately £100, which includes lunch and light refreshments on each day, and a copy of the position papers.

Optional costs will be approximately: bed and breakfast (24 to 26 March) £20 for each day, dinner (24 to 25 March) £9, workshop dinner (Tuesday 25 March) £25 (including wine).

Vegetarian meals are available on all occasions.

Intending participants should send no money at this stage. Payment forms will be sent with acceptance of position papers.

Programme committee

Prof. P. M. Dew, Leeds University (co-chair); Dr Tom Lake, GLOSSA (co-chair); Mr J. R. Davy, Leeds University (local organiser); Dr R. Allen, SERC Daresbury Laboratory; Dr J. P. Banatre, IRISA; Prof. J. Gurd, Manchester University; Prof. A. Hey, Southampton University; Prof. C. Jesshope, Surrey University; Dr D. May, INMOS Ltd; Prof. R. Milner, Edinburgh University; Mr M. Platt, IBM UK Scientific Centre; Dr C. P. Wadsworth, SERC Rutherford Appleton Laboratory.

For further details contact the Workshop Secretary, whose postal address is given above or contact the Local Organiser Mr J. R. Davy by electronic mail to `davyjr@dcs.leeds.ac.uk`

PARALLEL PROCESSING: THE FUTURE FOR COMPUTING

A British Council course

12–22 May 1991, Abingdon, Oxfordshire, England

Parallel processing is already well established as the most effective method of obtaining massive computational power at an affordable price. In future it will be the only means of achieving the speeds demanded by advances in computing applications.

The course is designed to bring together senior scientists, directors and administrators who make or influence decisions on computer strategy. Current achievements in parallel processing and future directions will be discussed with an emphasis on the interchange of views and experience. Participants will be asked to provide information in advance on computing activities and needs in their own countries and to highlight areas where parallel processing is most likely to be important.

The course will include seminars, discussions and panel sessions. There will be technical visits provisionally to the Inmos silicon fabrication facility and to the Rutherford Appleton Laboratory. An optional two-day workshop will be provided to enable participants to gain hands-on experience of a parallel processing system at the Rutherford Appleton Laboratory.

The course will be directed by *Professor Tony Hoare,* FRS, Professor of Computation, University of Oxford, and *Dr Mike Jane*, Co-ordinator, Transputer Initiative, Rutherford Appleton Laboratory. The Oxford University Computing Laboratory was amongst the recipients of The Queen's Award for Technological Achievement 1990.

The following topics are planned:

Hardware
- Physical limits to sequential computing
- Survey of parallel architectures
- Shared memory multi-processors
- Communication networks (distributed memory MIMD)
- Connection machines (SIMD)

Applications
- Medicine
- Communications
- Finance and commerce
- CAD/CIM
- Petrochemicals
- Environment
- Engineering and scientific research
- Expert systems
- Safety critical

Software
- Running a parallel processing service
- Migration aids
- Portability and standards
- Languages and libraries
- Tools and operating systems

Contributors to the programme are expected to include: Professor I. Barron, Founding director of Inmos Ltd and chairman of Division Ltd; Mr David Barrow, Director, Chiron Business and Technology Counsellors Ltd; Dr N. Carmichael, Shell UK Ltd; Mr Miles Chesney, Managing Director, Meiko Ltd; Professor L. M. Delves, University of Liverpool; Professor A. de Pennington, University of Leeds; Professor T. Durrani, University of Strathclyde; Dr M. Guest, Daresbury Laboratory; Professor C. A. R. Hoare, University of Oxford; Dr A. J. G. Hey, University of Southampton; Professor Peter A. Lee, University of Newcastle upon Tyne; Professor A. Linney, University College London; Dr Geoffrey Manning, Chief Executive Officer, Active Memory Technology Ltd; Dr Philip Mattos, Consultant Engineer, Inmos Ltd; Dr D. May, Inmos Ltd; Professor R. H. Perrot, Queen's University of Belfast; Mr D. Talbot, Head of Division Software Technology and Advanced Information Processing, Commission of the European Community, Brussels; Mr Kenneth R. Johnson, University of Edinburgh; Professor D. J. Wallace, University of Edinburgh; Professor P. Welch, University of Kent; Dr C. Whitby-Strevens, Inmos Ltd.

The course will conclude with a British Council keynote lecture on *Future Plans and Prospects for Parallel Processing*, which is expected to be given by Mr David Talbot.

Optional two-day workshop

The main course will be followed by an optional two-day workshop for participants who wish to gain hands-on experience from 22–24 May 1991.

Those interested in taking part should mention this when applying for the main course.

Qualification of members

The course is aimed at senior scientists, engineers, directors and administrators who make or influence decisions on computer strategy.

Numbers

There are vacancies for 30 participants.

Fee and accommodation charges

Course fee: £720
Accommodation charge: £415
Total fee: £1135

The course is residential only. The total fee includes academic programme, accommodation and all meals.

We are unable to accept non-residential participants and regret that no requests for reduction or refunds can be accepted.

Optional two-day workshop fee: £220 (this includes academic programme, accommodation and meals)

Location and accommodation

Participants will be accommodated in single rooms at The Cosener's House, Abingdon, where the course sessions are to take place. A few bedrooms have en suite

bathrooms and these will be allocated to the earliest applicants to send their forms to us. The remaining bedrooms have their own hand washbasins but share bathroom facilities.

Applications

Applicants are advised to apply before 18 January 1991. Application forms may be obtained from your nearest British Council Office or from

Courses Department
The British Council
10 Spring Gardens
London SW1A 2BN
United Kingdom

Tel: +44 71 389 4406/4264/4252

TRANSPUTER APPLICATIONS 91

The third international conference on the applications of transputers
28–30 August 1991, Glasgow, Scotland

Sponsors
SERC/DTI Transputer Initiative

Hosts
Scottish Transputer Centre

Transputer Applications 91 is the third in a series of International Conferences sponsored by the SERC/DTI Transputer Initiative. The conferences are widely acclaimed as highly successful international events, and the accompanying exhibitions are increasingly seen as curtain raisers for new products, systems and tools.

Transputer Applications 91 offers a unique platform to application developers from industry, government and academia for the presentation of results and experiences arising from the use of transputer based systems.

Transputer Applications 91 represents an important forum for the review of progress in this rapidly growing field, and for the identification of current trends and future directions.

The conference

The conference programme will include keynote presentations by leading experts, and will span over several parallel sessions of contributed papers. Papers may be presented in either oral or poster form.

To reflect the applications flavour of the Conference, submissions are invited from the following non-exclusive list of topics:

- real time control,
- industrial and commercial applications,
- image processing and pattern analysis,
- software tools and programming environments,
- signal processing,
- standards,

- ▷ computational fluid dynamics,
- ▷ molecular modelling,
- ▷ applications in communications,
- ▷ application on proprietary operational systems.

Submission of abstracts

Authors are invited to submit papers on original and unpublished work which can be described as a 'transputer application'. Submissions must adhere to the following instructions:

- ▷ Three copies of an extended abstract of approximately four pages should be submitted for consideration by referees.
- ▷ Two copies of a short abstract of *at most* one page should also be submitted for publication in the Transputer Applications 91 edition of the SERC/DTI Transputer Initiative Mailshot.
- ▷ Where possible, all material should be typeset on an A4 page in 12 point Times Roman fount.
- ▷ For papers with multiple authors, the presenter should be identified.
- ▷ Full mailing address, telephone and fax numbers of the author to be notified should be indicated.
- ▷ Proceedings of the conference will be published prior to the event.
- ▷ Abstracts and enquiries should be addressed to:

  TA 91 Conference Abstracts
  Scottish Transputer Centre
  Exchange House
  229 George Street
  Glasgow G1 1RX
  Scotland
  United Kingdom

Deadlines

The following deadlines for the submission of abstracts and papers must be observed:

29 March 1991 Submission of abstracts.

3 May 1991 Notification of acceptance.

14 June 1991 Submission of camera-ready papers for inclusion in Transputer Applications 91 Conference Proceedings.

Scottish Transputer Centre
Exchange House
229 George Street
Glasgow G1 1RX
Scotland
United Kingdom

Tel: +44 41 552 4400 x2499
Fax: +44 41 552 2487

FOURTEENTH OCCAM USER GROUP TECHNICAL MEETING

Monday 16th September to Wednesday 18th September 1991
Loughborough University of Technology, England

The occam user group invites all those interested in the programming and application of transputer based architectures to attend its 14th technical meeting at the Loughborough University of Technology. The conference includes invited and refereed lectures, an exhibition, a panel session with key speakers, and meetings of the Special Interest Groups.

The conference will be concerned with both the applications of transputers and the techniques and tools that allow transputers to be used effectively.

The meeting and exhibition will take place on the University campus which is situated less than two miles from the mainline railway station, six miles from East Midlands International Airport and less than a mile from the town centre. Accommodation will be available on campus. There will be a conference dinner on Tuesday 17th September.

Call for papers

Authors are requested to submit *five* copies of an extended abstract of between 1000 and 1500 words outlining the content of their paper. Abstracts should be sent to the conference organiser to arrive by Thursday 25th April 1991. As well as the name and title of all authors, each paper should include the full *name* and *title* of the individual(s) giving the presentation, a telephone number and, if possible, e-mail address and fax number where the presenter can be reached. Submitted abstracts will be refereed by three members of the OUG committee.

The emphasis of the conference is both on the applications of transputers and the techniques and tools that allow transputers to be used effectively. Contributions are invited in all relevant areas of interest, for example: simulation, real time control, image and signal processing, graphics and CAD, databases, neural networks and AI, parallel programming languages, functional and object oriented languages, environments, software tools, formal methods, design methodologies and performance evaluation.

Papers will be presented at the meeting in half-hour sessions (including time for questions).

Deadlines

Abstracts must be received by: 25th April 1991
Acceptance Notification by: 23rd May 1991
Submission of Complete Papers: 20th June 1991

Accepted papers will appear without subsequent revision in the proceedings, which will be published by IOS, Amsterdam, and will be distributed at the meeting. Papers not received by this date will *not* be included in the proceedings.

Conference organiser

All enquires should be made to the Conference Organiser:

Janet Edwards
Department of Computer Studies
Loughborough University of Technology
Loughborough
Leicestershire LE11 3TU
England

Tel: +44 509 222677
Fax: +44 509 610815
J.Edwards@uk.ac.lut

AUSTRIAN CENTRE FOR PARALLEL COMPUTATION

First international conference
30 September – 2 October 1991, Salzburg, Austria

Call for papers

The Austrian Center for Parallel Computation (ACPC) is a co-operative research organization founded in 1989 to promote research and education in the field of software for parallel computer systems.

The first international conference of the ACPC will take place at the University of Salzburg and is intended as a forum for both researchers and practitioners in the field.

We invite submissions of papers presenting original research in topics including the following areas:

- ▷ Algorithms
- ▷ Languages
- ▷ Compilers
- ▷ Programming Environments
- ▷ Applications

for parallel computation including high-performance computing.

Contributors are invited to send five copies of a full paper not exceeding 12 pages in English to

Prof. Hans P. Zima
Institute for Statistics and Computer Science
University of Vienna
Rathausstrasse 19/3
A-1010 Vienna
Austria

Tel: +43-1-401032788
Fax: +43-1-4089250
A4424DAJ@AWIUNI11.BITNET

Submissions must arrive before 1st March 1991. Authors will be notified of acceptance or rejection by 15 May 1991. The final versions of accepted papers must arrive in camera-ready form before 30 June 1991, to ensure the availability of the proceedings at the conference. The proceedings will most probably be published in the Springer Verlag Series Lecture Notes in Computer Science.

Program committee

Chairman: H. Zima (Vienna); G. Balbo (Turin); A. Bode (Munich); J. Boyle (Argonne); B. Buchberger (Linz); M. Chytil (Prag); G. Haring (Vienna); U. Hofmann (Dresden); W. Kleinert (Vienna); I. Plander (Bratislava); P. Schuster (Vienna); U. Trottenberg (Bonn); C. Ueberhuber (Vienna); J. Volkert (Linz); M. Wolfe (Portland); P. Zinterhof (Salzburg).

If you intend to attend the conference, please send your name and address and telephone and fax numbers and electronic mail address, together with an indication of whether or not you intend to present a paper, as soon as possible to:

Mag. R. Schiller
Research Institute for Software Technology
University of Salzburg
Hellbrunner Strasse 34
A–5020 Salzburg
Austria

REPORTS

TRANSPUTER USER GROUP ARGENTINA

The first meeting of TUGA, with the purpose of formalizing the creation of the group, was held on Wednesday 25th April 1990 with around twenty people attending. Group co-ordination was decided as follows:

Co-ordinator: Lic. Guillermo Navas
Academic Committee:
Dr Enrique D'Atellis – Numerical Control
Dr Armando Menendez – Electromagnetism
Dr Roberto Perazzo – Neural Networks
Lic. Gustavo Sanchez Sarmiento – Computational Mechanics
Dr Hugo Scolnik – Combinatorial Optimization
Ing. Jorge Sinderman – Electronics and hardware
Secretary: Sr Guillermo Marshall – Numerical Methods
Relations with Industry and Commerce:
Ing. Mario Frigerio
Secretary: Ing. Daniel Lamela

For the time being the group's activities are being supported by KEYDATA S.A. and her sister companies KEYSOFT S.A. and KEYTECH S.A., members of the CNEA (National Atomic Energy Commission), and the CONICET (National Science Council).

Planned activities centre around informing probable users about this new technology and include: an inaugural meeting in Buenos Aires in the summer, a set of tutorials and courses about occam and the transputer, a series of conferences during the Winter Computing School at the University of Buenos Aires in August, and

possibly an event during the September Latin American Meeting of the CLAIO, Latin American Conference on Informatics and Operations Research.

Esteban R. Gillanders
Keydata S.A.
Crisólogo Larraide 1801
(1429) Buenos Aires
Argentina

Tel: +54 1 70-4467/3281
Telex: 23096 KEYSA AR
Fax: +54-1-11-2426

FIRST NORDIC TRANSPUTER SEMINAR

Turkku, Finland, 6th and 7th October 1990

Lars Estreen and Martin Tørngren

After having had two seminars in Sweden, in May and November last year, the Swedish Transputer User Group, STUG, wanted to try gathering more people and more experience, even though 40–50 people attended the last seminar.

As we had heard about a lot of activities in the other Nordic countries, a natural way to expand the STUG seminars was to arrange a Nordic Transputer Seminar. A committee with two members from each of Sweden, Norway, Denmark and Finland was formed by STUG. Administrative parts of the arrangements were handled by Inmos and SGS Thomson's representatives in the Nordic Countries.

The initial expectations of 20 persons from each country in the end turned out to be realistic as around 95 persons visited the seminar.

A successful seminar

Most people attending the seminar gathered Friday evening in Stockholm to take the ferry to Turkku in Finland. After a pleasant pub session at sea the seminar began Saturday morning at Åbo Akademi in Turkku, Finland.

It was about time that the transputer users in the Nordic countries met! There were representatives from several Universities, Technical Institutes and companies in the Nordic countries, most of them having a lot of experience with transputers. The informal contacts were very rewarding.

A contributing reason for the success of the seminar were the pleasant arrangements made by our hosts at Åbo Akademi.

The keynote speakers were Richard Forsyth from Inmos who talked about the H1, the next generation transputers, Bart Veer from Perihelion software who presented Helios, Philip Mattos who presented a transputer based GPS (Global Positioning System) in the perhaps most appreciated talk during the seminar. Finally Mark Jones from Inmos talked about the current status of the transputer. He reported a strong position for the transputer compared to other RISC processors like M88000 and AMD29000. However, he did not mention the 680x0, 80x86 or the National family of processors. Can it thus be deduced that Inmos is 'the fourth bestseller' in the realm of 32 bit micros?

The general impression of the Inmos talks was that there was not much news, either about the H1 or occam 3. However, people enjoyed hearing the small pieces of information available.

The other talks (15 in total) were generally very interesting, too. The main field of interest seemed to be speed-up efforts, followed by vision applications and development/debugging tools.

One part of the seminar that did not work out so well was the workshops, because they were scheduled for Sunday afternoon. Another thing, where were all potential exhibitors? Only three showed up. These are things that can be improved.

The overall impression is that the seminar was a success. Look out for a new one next year.

The Swedish Transputer User Group aims at continuing publishing a newsletter. The newsletter will from now on be published to an audience in the Nordic countries. We think that there is a need for such a newletter (complementing the OUG newsletter).

Peter Ygberg is publishing coordinator and contributions to the newsletter (in English or Swedish) should be sent to him before 1st February 1991.

Enquiries or contributions to the newsletter should be directed to:

Peter Ygberg
Fridhemsgatan 8, 5 trappor
11240 Stockholm
Sweden

or by electronic mail to Martin Tørngren at stug@damek.kth.se.

EDUCATION AND TRAINING SIG WORKSHOP

National Transputer Support Centre, Sheffield, 3rd July 1990

This workshop was attended by ten educators or transputer users, who normally are associated in some way with parallel computer training in the commercial, industrial or educational fields.

The intention of this workshop was to provide the opportunity for these educators to come together and design a real-time control application, and then to discuss the results and how such programming experiences could be passed on to students.

The task provided by Dr Jon Kerridge was to write a photocopier simulator which would run on one of the NTSC's T222 boards. These are equipped with a range of peripheral interfaces capable of emulating the buttons and displays of a photocopier. Following an introductory session on the facilities available, the attendees split into two groups, and tackled the task in two very different ways.

One group, keen to try out the hardware provided, immediately embarked on a bottom-up design exercise, attempting to write driver code for the various peripheral interfaces. During the three or so hours available, they made good progress, investigating the complexities of key switch debouncing, matrix switch encoding and 7-segment LED display multiplexing. No real effort was made, however, to produce interfaces to these routines which would be suitable for the code of the main simulator.

The second team embarked on a much more ambitious design exercise, based around the hope that the various elements of the copier (controller, input paper feeder, scanner and output fuser) could be implemented as a string of independent

parallel processes. Much effort was spent designing all the interfaces to the parallel processes controlling the peripheral devices, in order to avoid deadlock. The major area of heated debate was how a single reset switch (provided on most photocopiers to recover from paper jams and other failures) could be interfaced to several pipelined functional units when only one of these might have failed, and others were still buffering partly-processed copies! Needless to say, no time was left for implementation, but the design was completed successfully.

Before the workshop, Jon had already produced his photocopier simulation. Partly as a consequence of initially testing the peripheral hardware using individual processes, his solution just harnessed these processes with a single sequential controller, which requested that each perform its action in turn.

The author has since implemented the parallel, pipelined design; it worked without modification, and provides a user interface identical to that of Jon's code.

In the discussion which followed the three group presentations, it was clear that a purely top-down design could produce problems at the device-driver process level unless they are successfully anticipated. In contrast, the initial coding of the lowest-level processes independent of the design of the higher-level ones was shown to be inviting deadlock problems. A middle approach – not surprisingly – was considered optimum.

It was agreed that the workshop had been a stimulating event, and that seeing other people's design processes at work was useful. The workshop material will form the basis of a tutorial at the forthcoming TRANSPUTING '91 conference, where the principles of teaching parallel programming methodologies will be examined from the viewpoint of those setting up new courses as well as those looking to improve current ones. *Roger M. A. Peel*

NORTH AMERICAN TRANSPUTER USERS GROUP FALL MEETING

Ithaca, New York, 11–12 October 1990

The Fourth Meeting of the North American Transputer Users Group was held in Ithaca, NY on 11–12 October. There were 82 attendees and 23 presentations. Vendor exhibits included Transtech Devices Ltd., Yarc, MicroWay, and Inmos.

The meeting opened with the keynote address entitled 'Portable parallel programming' by Dr Geoffrey C. Fox, currently the Director of the Northeast Parallel Architectures Center at Syracuse University, and recently the Director of Caltech's Concurrent Computation Program. Dr Fox discussed future directions in parallel computing and ideas for a machine-independent programming environment.

The technical presentations focused on applications which dealt with a wide variety of topics including graphics 'Parallel radiosity methods', industrial control 'An application of transputers to a metal milling machine', medical treatment 'Interactive dose calculation for 3-D radiation treatment', industrial visualization 'Real-time turbine engine visualization' just to highlight a few of the many fine technical papers. The proceedings from this meeting, 'Transputer research and applications 4,' are available through IOS Press.

The presentations were excellent, and at the conclusion of the meeting atten-

dees voted for the top presentations. The award for the best presentation went to William T. Carter, Jr. from GE Corporate Research and Development for his talk on 'Finite element analysis on a transputer system.' The second place award went to Alan G. Chalmers of the University of Bristol for his talk on 'Parallel radiosity methods'.

The meeting's banquet dinner was held on Thursday evening, following which there were a number of vendor announcements. Martin Shumway, from the Inmos Division of SGS Thomson, highlighted the finer points of the upcoming H1, including details of H1 routing and H1 instructions. Tanya Schmah, from JMI Software Consultants discussed a real-time kernel for the transputer, and Bernt Roelofs, of YARC, discussed their transputer products along with their product line based on the AMD 29000 RISC CPU. Judging from the numerous questions from attendees the after dinner announcements were of great interest to attendees and a nice addition to the meeting.

The fall meeting here in Ithaca was quite successful in all aspects except for the weather. The beautiful colors of a fall in the fingerlakes region and the historical Cornell University campus nearby more than made up for any slight downturn in weather. The weekend following the meeting was spectacular, if this is any consolation.

The desire for portable parallel programming has existed for a period of time, Dr Fox stressed its importance in making parallel computing more appealing to industrial concerns. While some level of portability has been achieved by the transputer software environments available today, each has been developed independently and vary in their degrees of portability. In the future I would like to see these groups come together to look at standards being developed for multiprocessor systems and to make their own significant contribution to future standards. We can look forward to more on portability at upcoming meetings.

Excitement surrounding the H1 was clearly evident here in Ithaca and can only be expected to pervade at the world conference next April where unveiling of the H1 is expected. I look forward to seeing you all in Santa Clara at the first world conference.

The technical sessions were as follows:

- *Fluid dynamics on express: an evaluation of a topology-independent parallel programming environment*, Lon-rong Hu and G. S. Stiles, Department of Electrical Engineering, Utah State University;
- *Topology-independent mapping for transputer networks*, Mahendra Ramachandran, Manas Mandal, and Prasad Vishnubhotla, Department of Computer and Information Science, The Ohio State University;
- *Interrupt driven transputer file server*, R. M. H. Cheng, S. C. L. Poon, Ramesh Rajagopalan, Center for Industrial Control, Department of Mechanical Engineering, Concordia University;
- *Mapping rings onto arbitrary transputer networks*, Raju Karia and Alejandro Teruel, Departamento de Computacion, Universidad Simon Bolivar, Caracus, Venezuela;
- *Tmon: a real-time performance monitor for transputer-based parallel systems*, J. Jiang, A. Wagner and S. Chanson, Department of Computer Science, The

University of British Columbia;

- ▷ *A distributed algorithm for the 3D compressible Navier-Stokes equations*, Kwan-Liu Ma and Kris Sikorski, Department of Computer Science, University of Utah;
- ▷ *Real-time preprocessing of multiple images using an unconventional pipeline approach*, Hamid R. Arabnia, Department of Computer Science, The University of Georgia;
- ▷ *TransLogic: a transputer based brute force digital logic simulator*, G. J. Porter, B. M. Hutson and S. N. Lonsdale, Department of Electrical Engineering, University of Bradford;
- ▷ *A generalized FFT algorithm on transputers*, Herman Roebbers, University of Twente;
- ▷ *Finite element analysis on a transputer system*, William T. Carter, Jr., GE Corporate Research and Development;
- ▷ *An application of transputers to a metal milling machine*, James C. Irrer, Control-O-Mation Inc.;
- ▷ *Evaluating communication overhead in Helios*, Wouter Joosen, University of Leuven;
- ▷ *Occam configuration as a task assignment problem*, Howard E. Motteler, Department of Computer Science, University of Maryland, Baltimore County Campus;
- ▷ *Software controlled shared virtual memory management on a transputer-based multiprocessor*, Sanjay Raina, Department of Computer Science, University of Bristol;
- ▷ *A dataflow implementation of functional language FP on transputers*, L. Yang and L. Jin, Department of Computer Science, California State University – Fresno;
- ▷ *A parallel genetic algorithm for the graph partitioning problem*, Gregor von Laszewski, Gesellschaft für Mathematik und Datenverarbeitung;
- ▷ *Static vs. dynamic scheduling in cellular automaton*, M. S. Laghari and F. Deravi, Department of Electrical and Electronic Engineering, University College of Swansea;
- ▷ *Parallel radiosity methods*, Alan G. Chalmers and Derek J. Paddon, Department of Computer Science, University of Bristol;
- ▷ *Implementation of the dynamic time warping algorithm for speech recognition on a transputer network*, Varma R. Manthena and John A. Board, Jr., Department of Electrical Engineering, Duke University;
- ▷ *Real-time turbine engine data visualization*, Theodore Bapty, Ben Abbott, and Janos Sztipanovits, Department of Electrical Engineering, Vanderbilt University;
- ▷ *Interactive dose calculation for 3D radiation treatment planning*, Fred U. Rosenberger and John W. Matthews, Washington University, Institute for Biomedical Computing;
- ▷ *Transputer performance in digital signal processing applications*, G. M. Guisewite, HRB Systems;
- ▷ *Topological network of vector processors*, Runping Qi and Ying Zhang, Department of Computer Science, University of British Columbia.

Also contained in the proceedings of NATUG 4 are:

- ▷ *Transputers at work: real-time distributed robot control*, Louis L. Whitcomb, Yale Robotics Laboratory, Department of Electrical Engineering;

▷ *A parallel implementation of robot control equations on IMS T414 transputers*, Ruiguang Zhang, Eduardo B. Fernandez, and Jie Wu, Department of Computer Engineering, Florida Atlantic University;

David L. Fielding

WHEN IS A RAZOR NOT A RAZOR?

The joke session at the technical meeting held in Wentworth College in York was dominated – as was the end of the meeting itself – by the muted outrage which Geoff Barrett's language proposals caused. Perhaps the most restrained comment was that

Occam's razor seems to be suffering from the distinct lack of shaver sockets at Wentworth!

There was the usual attention to the competition:

Q: What does the i stand for in i860?

A: It is the imaginary part of the performance figures.

as well as Inmos:

Q: What is the real meaning of 'INMOS'?

A: **I***ndustry* **N***eeds* **M***ore* **S***ources.*

Oh, and did I mention Geoff's proposals?

Q: What's the methodology behind the increasing version numbers of occam?

A: The version number is about $\log_{10}$ *of the number of keywords.*

Of course 'real-time' is the buzz-word these days:

There was a real-time programmer called Walt,
Who had one too many guards in his ALT*,*
The world wouldn't wait,
He was scheduled too late,
And now the population is nought!

I happened to be in the SIG where this one was heard:

Quote from Inmos Diplomacy department:-
"Inmos will provide tools to prevent people screwing themselves...
... and switches so that they may enjoy themselves."

And then there was the obsession with Geoff's occam development paper again:

The Ada 90.x Standard is definied as a subset of occam 3.

Of course there are some favourite targets:

There were several transputers in York,
In deadlock they always would baulk,
C was the way,
But it crashed the first day,
In occam, however, they'd talk.

Then there was the reaction to Inmos' plans for a H-series of transputers, and the talk of the open-microprocessor project, codenamed the E-series:

There's a wonderful fam'ly of transputer:
Each is a splendid computer.
The T eight is great,

The H1 is late,
And the E's far away in the future.

In fact, as Geoff Barrett let slip the following day, the rationale behind these names is just that Inmos wanted to become T-H-E microprocessor manufacturer. And talking of Geoff reminds me of the occam development paper...

A transputer user and gent,
Found his programs all bugged and bent,
He tried a `PRI PAR`*,*
An `ALT, SEQ, TYPE, STREAM, JOIN, BUS, RECORD,`
`CLASS, RESOURCE, FINAL, SERVE, CLAIM` *and* `PAR`*,*
Gave up and went – spent – back to Kent.

That was the winner: written by Chris Jones of British Aerospace, as were a number of the others. *gj*

SPECIAL INTEREST GROUPS

REAL TIME SIG

There was a meeting of the SIG at the York user group meeting, but the presentation on occam 3 the previous day did cast its shadow ahead on the workers in the Real Time and embedded systems. This OUG meeting did have a definite emphasis on real time systems. The contending scheduling papers by Welch and Sunter indicated that something is still missing. Too bad we missed the paper on Trans-RTXc by Verhulst. It shows one implementation that allows rate monotonic scheduling on the transputer. For the rest there was not much hope for Real-Time systems. Although the paper on the assessment of the use of occam for dependable real-time systems by Burns made suggestions that there might be different occam 3s like: one for real-time systems; one for Object Oriented Programming etc.

One of the conclusions that came out of the discussions was that occam is a good base for RT systems. However a straight realization of a real-time kernel will probably not result in a state of the art system because many current RT kernels don't use up to date theory. Most RT kernels only have fast context switches and priorities that will guarantee scheduling in finite time but not before certain deadlines. It was doubted whether current RT kernels can handle transputer processes.

Alan asked: What are the scheduling hooks? Inmos won't tell although Roger Shepherd said previously that the hooks will probably not be sufficient for an all inclusive RT system. Also one is led to believe that occam already contains the necessary things for fixed priority scheduling namely `PRI PAR`. Besides that the transputer has already the necessary hardware to do deadline driven scheduling: the *timer queue*. Both papers mentioned before have showed that this is not sufficient for deadline driven scheduling. What is needed for this is exactly known and proposals for implementation have been made whereby the high priority remains intact with lightning fast pre-emption time. Low priority pre-emption takes longer but will be

fast enough. However to quote Roger Shepherd: We don't make what is best for them, we make what they know.

A large part of the market is the embedded systems market. Why is Inmos not actively supporting RT systems? Routing chips are pretty fast, but is there an upper bound for the delay? Otherwise RT kernels are not going to use them. The same thing for the cache, the cache of the H1 will be turned off in RT systems. Real time constructs do not have to have zero probabilities. Probabilities equal to the probability that the hardware fails are low enough to live with.

When priorities can be changed by other processes, priority inheritance becomes possible. Deadline scheduling is the basic layer from which the higher layers of RT systems can be built. Also it was reported that the formal semantics of a multi priority CSP has been worked out in Italy. *André Bakkers*

HARDWARE SIG

A meeting of the hardware SIG was held on 18th September 1990 during the York OUG meeting. As usual, the meeting consisted of several short presentations about items of current interest.

Virtual reality

Neil Miller of Division Ltd described his company's prototype 'virtual reality system' which was demonstrated at the SIG-GRAPH meeting the previous month. A simulated environment is offered to a user equipped with

- a pair of goggles with a video screen for each eye,
- headphones,
- a position detector on her forehead, and
- a glove to record hand movements.

The system as demonstrated offers a three-dimensional cursor which flies around a room, giving the ability to pick up a table, chair or teapot. They 'clunk' and 'clink' as they are moved. The user is even offered a menu which allows her to enter a room with 'Boris the spider'.

We were amused to learn that, as the company had forgotten the necessary calibration software, Stephen Gee had to spend eight hours a day for three days locked in this virtual world.

Proposed uses for the system, which was to be formally launched the next week at the Royal Society, include:

- An executive toy.
- A training aid for dangerous environments.
- A flight simulator – for those of us who fly without an aeroplane.
- Previewing architectural projects.

The image rendering system for virtual reality ran on sixteen transputers, but Neil confessed that the image quality was not very high as they have insufficient processing power.

C104 emulator

Mark Hill of Southampton University described a message routing system that is under development for the Esprit *PUMA* project. The system uses a cluster of four T222 transputers, linked in a square by dual-port memories, to provide a sixteen-link message routing engine. Eight such engines can provide complete single hop routing for a thirty-two transputer machine, such as a fully populated Supernode backplane.

While there is some small improvement in message latency, Mark claimed that the principal advantages of the system were

- ▷ improved security for a multi-user system,
- ▷ improved message throughput, and
- ▷ greatly reduced message processing load on the 'compute' transputers.

The system is fully integrated with a virtual occam compiler that provides unlimited inter-processor occam channels.

Prototype cards

Roger Peel of the University of Surrey described some transputer prototyping cards that he had designed and was willing to make available. These single extended Eurocards will accommodate a T4 or T8 processor and two to eight megabytes of DIL DRAM memory, with a PAL providing full address decoding and memory timing. Four subsystem ports are also provided on the board. External interfaces include an extended B003-type connector and a 96 way expansion connector which carries latched transputer addresses and all other appropriate transputer signals. The whole board consumes 300 mA from a 5 V supply.

Jon's boards

Jon Kerridge promised to write a note for this *Newsletter* about his transputer prototyping boards... *Denis Nicole*

OCCAM DEVELOPMENT SIG

Partly because a number of people had been pressing for a forum to discuss development of occam, and largely because Geoff Barrett presented a proposal for substantial additions to the language, we held a SIG at the York meeting which was variously known as an *occam development SIG* and a *Geoff Barrett SIG*. Judging by the vast attendance there was a great deal of special interest either in Geoff Barrett or – more likely – in the development of occam.

Geoff summarised the main proposals from the paper [1] which he had presented the previous day, calling for four classes of addition to the language: some new data-structuring; some assistance with code structuring, re-use and encapsulation; support for sharing of objects; and improved error-handling.

There was general acceptance of the relatively modest data-structuring proposals, essentially the re-introduction of the records that briefly flitted into and out of the occam 2 proposal. The mechanisms equivalent to enumerated types seemed

cumbersome to some of us. Classes were where the argument started; several people were concerned that there were too many different ways of structuring code in much the same way. Peter Welch and Chris Jones were variously concerned about separate compilation units, libraries and classes all being similar but different. In defence of classes, Geoff said that separate compilation was essentially a separate concern (he really did, hones' injun) and that libraries are stateless, whereas classes will not be and capture initialisation of their state and so on.

The proposed mechanism for sharing – something like, but more general than an arbitrarily wide ALT – matches 'claim's and 'accept's rather as though they were communications on shared channels, but governing a region in which the shared resource 'knows' to which process it is allocated. There was some discussion of whether it is possible to hide obscure interactions between process in the claim/accept mechanism for extended rendezvous. The assertion is that since claims are forbidden in the critical region governed by a claim there is no interesting way to obscure a deadlock. (Anyone recognise a challenge when they see one... ?)

The server is a particularly common construction for which Geoff had proposed a language cliché, bundling a great deal of housekeeping into a standard package. When Jon Kerridge claimed not to understand the termination of servers, Alan Burns raised a laugh by explaining it in terms of Ada. There was common concern that the proposal was for a much larger language – certainly one with a great many more key-words. Perhaps the absence of printed copies of the paper (the proceedings not having been published in advance of this meeting) made this worry greater than it would otherwise have been.

As the timetable bore us on inexorably into a much needed coffee break, it was concluded that what we needed now was a 'serious' example of the proposed language in use.

References

[1] G. Barrett, *The development of occam: types, sharing and modules*, in H. Zedan (ed.), *Real-time systems with transputers*, Proc OUG TM-13, York (September 1990), IOS, 1990.

Geraint Jones

FORMAL METHODS SIG

For the second time running, a baker's dozen of people attended the meeting, which was held during the 13th OUG technical meeting at York in September. There was again quite a cosmopolitan mix among the attendees (four countries, and a fairly even split between industrial and academic organisations).

There was very little news of interesting research being started in the area. Given the Technical Meeting's theme of real-time concurrency, there was quite a lot of interest in how well the established formal methods for untimed analysis and design extend to deal with real-time aspects: how, for instance, could one treat deadline scheduling in a formal framework?

Much of the meeting was spent discussing how the SIG could best promote and

promulgate formal methods. It became apparent that there was a wide range of prior experience and knowledge among the attendees, and presumably among the potentially interested world at large: ranging from people who want to know what formal methods are, and what they are good for, to those with considerable expertise who want to exchange techniques with other experts.

It was decided that it would be good to try and organise some more technical meetings, in addition to the general chat during the SIG sessions of full OUG meetings. Two subjects in particular attracted support: a fairly broad survey of formal methods for the non-expert, and a guru session for those into real-time methods such as Timed CSP and Timed CCS. The first of these will form the topic of one of the tutorial sessions at TRANSPUTING '91, and may very well be repeated (probably in an abridged version) on this side of the Atlantic at some future date. It was hoped that the second might be organised for early in 1991, but at the time of going to press it has not yet proved possible to arrange; this probably means that it will slip to the end of next summer. Any details will be circulated to anyone on the SIG's list of interested parties, and sent to the occam and transputer electronic mailing lists, as well as appearing in the *Newsletter* if time allows.

Once again, I opened the meeting with a statement of good intentions, to try and make more happen in the Formal Methods SIG; once more, I close the report with the reminder that, in the way of the world, things are much more likely to occur if I am given encouraging prods from time to time, or even constructive suggestions. Anyone who can come up with either is most welcome to contact me at the address on page 105. *Michael Goldsmith*

EDUCATION AND TRAINING SIG

Members at the York technical meeting started by summarising the various hardware and software environments which are suitable in a teaching environment. The discussion examined the various hosting possibilities (PC, software-only schemes on multi-user machines, transputer access from multi-user machines), as well as the benefits of TDS versus standalone toolkit products, folding editors and so on. The problems of how to resource large groups of students were aired, but not solved!

The meeting also heard a report on the Education and Training SIG Workshop at the Sheffield Transputer Support Centre in July, for which see page 20. This was followed by a discussion on the educational benefits of the exercise, and how the aspects of top-down and bottom-up design which had been seen could best be demonstrated in a teaching context. *Roger M. A. Peel*

GRAPHICAL PROGRAMMING TOOLS SIG

Although many programmers would like to see usable tools for software visualisation and graphical programming, only eight people attended to the SIG meeting at the 13th OUG Meeting at York University.

It was anticipated if there was any complete or partial products for occam program development with graphical tools. Alan Knowles from Manchester University

mentioned that his colleague Adrian West is working on a software environment which also includes graphical tools. Indeed Adrian gave a talk on the subject in the 12th OUG meeting at Exeter. With this, pictorial representation of the programs and editing with icons would be possible. However the tool could not have a general purpose usage as it was designed for T-rack machine in Manchester.

Ed Hart from Polytechnic of Central London announced that they are selling Transim/Gecko package. Transim simulates the behaviour of a program on a transputer network. Gecko visualises the activity of the program. Transim/Gecko combination need Sun workstations. Transim alone can also run on the IBM-PC. However, the development of these products ceased a long time ago.

At Kent, I have been developing graphical representation and manipulation of program objects as a by-product of a Version Management System.

In order to enhance the discussions, it was suggested that this SIG could be combined with the Environments SIG. *S. Vedat Demiralp*

TECHNICAL CONTRIBUTIONS

SENDING OVER A TRANSPUTER LINK WITHOUT ACKNOWLEDGE

Dieter Homeister, Universität Stuttgart, Germany

In this article an algorithm is presented to send data trough a transputer link without the need of acknowledge packets for each byte. If the bytes are lost, no error occurs. This is useful to resynchronise disconnected links.

A star topology with transputers

The algorithm was developed for a master-slave system in star topology. For a time critical application a star network was necessary to avoid a multihop message passing. The realisation of a star network with more than four slaves is difficult, because Transputers have only for links. So a reconfigurable network with a C004 crossbar switch was used to simulate a star network in time multiplexing. If the slave has finished a task, it cannot send the result to the master, because in this network a slave is disconnected from the master most of the time. The slave would run into a deadlock. Before sending, a synchronisation between master and slave must be performed and the master has to reconfigure the network.

Resynchronisation of disconnected links

The first solution was a polling algorithm. The master processor establishes a direct connection to all slaves in a round-robin fashion. A byte is sent to wake up an interrupt process in the slave, telling the master whether the task is finished or not. If a result is ready, the slave sends it now. Otherwise, the next slave is tested. If the master polls the slaves too fast, the interrupt process on the slave consumes a lot of

time and the working process slows down. Too slow a polling speed causes long idle times on the slaves.

To avoid these disadvantages a better algorithm with no need for an interrupt process was developed. When the task is done, the slave tries to send any data to the master. This fails in most cases and the data bytes are lost. The slaves get no link acknowledge, but a timeout avoids a deadlock. The slave tries to send again in a loop until a synchronisation with the master succeeds. To poll the slaves, the master connects to a slave and 'listens' into the line. If the line is quiet the slave is still working on the task. The master now switches to the next slave. If the master receives data on the line, an idle slave has been found. The master sends a byte to stop the continuous data stream from the slave. The slave now knows that the connection to the master is established and sends the result of the task. The master now may send a new task to this slave.

While working on the task the slave is never interrupted. The whole CPU capacity is used for the task. When the task is finished the slave is idle and it does not matter how much CPU time the synchronisation consumes.

The master has to listen for a time interval including at least one byte from an idle slave. This time interval may be decreased if the slave sends the bytes without acknowledge as fast as possible. The master is able to poll more slaves in the same time and the synchronisation becomes faster.

How can a transputer send bytes without acknowledge?

The link receivers use a one byte buffer. A link is ready to receive data if this buffer is free. After the reception of one byte the DMA controller writes this byte into the memory and sends an acknowledge packet back to the sender. The link sender now knows that the receiver is ready for the next byte. The sender is blocked until the acknowledge arrives. A lost acknowledge or a lost data byte causes a deadlock in the sending or the receiving process or in both communication partners.

One method to send bytes without waiting for the link acknowledge uses the occam procedures described in reference [2]. The procedure `OutputOrFail.t` was designed to send over an unreliable channel with timeout. `OutputOrFail.t` is repeated in a loop until the master transputer sends a stop byte. A byte is sent without acknowledge every 65 microseconds. A transputer with the old (slow) link protocol needs 2·4 microseconds per byte in the regular operation with acknowledge. (All results are measured on a system with a link speed of 10 Mbit/s.) This solution was not fast enough for the application.

The analysis of `OutputOrFail.t` shows why this is not the fastest solution. (I had no sources, but disassembling and understanding the procedure was a good exercise to learn transputer assembler and the implementation of an `ALT` statement.) To implement an `OutputOrFail.t` procedure an `ALT` statement for a timer and an output statement would be needed. This could look like this:

```
ALT.OUTPUT -- this language construct does not exist
  unreliable.chan ! data
    ok := TRUE
  timer ? AFTER timeout
    ok := FALSE
```

The non-existent output alternative has to be transformed in an input alternative:

```
PAR
  SEQ
    unreliable.chan ! data
    help.chan ! any.data -- this simulates the ALT.OUTPUT
  ALT
    help.chan ? dummy
      ok := TRUE
    timer ? AFTER timeout
      ... GUY RESETCH -- reset link and test if still hanging
      IF
        hanging
          SEQ
            ... restart hanging process with GUY RUNP
            help.chan ? dummy
            ok := FALSE
        SKIP
          SEQ
            help.chan ? dummy
            ok := TRUE -- or FALSE if this should be an error
```

There are two parallel processes. The first one sends to the unreliable channel, and after the successful termination it sends an end message to the second process. In this second process an ALT waits for a timer or the input from the first process. This is (with the help of the internal channel *help.chan*) the desired output ALT. An input from *help.chan* tells the second process that the communication via the unreliable channel succeeded, the procedure ends without an error message. The timer input causes a RESETCH-instruction [3]. The instruction returns an indication of whether the communication is hanging or was finished in the meantime. If the communication is hanging, the procedure ends with an error message. In the procedure OutputOrFail.c the timer alternative is replaced by a stopper channel.

There is another way to synchronise a master and a disconnected slave. To test if the slave's task is finished the master sends a byte to the slave. A busy slave ignores this byte and the master is blocked. After a timeout the master knows that the slave is busy. If the slave answers, the master has found an idle slave. This variant was implemented in reference [4] under Helios.

A fast algorithm

The fastest possibility to send without acknowledge is based on two twin processes in the style of OutputOrFail.c. One process sends a byte and hangs, because it receives no acknowledge. If no other processes are running, the second twin process is scheduled immediately. The first action of this process is to abort the failed communication in the other process with a RESETCH, so that the hanging process is appended at the end of the process queue. Now the active process sends a byte, hangs, and so on. A third process waiting for the stop byte from the master transputer is used to stop this mechanism, if the synchronisation between master and slave is completed.

```
stop = 0; /* clear the shared variable */
parallel
   {
   process: /* sends FF(hex) until a stop byte comes from the master */
   process: /* second, identical process sharing the same code */
      /*$NOSCHEDULE*/ /* compiler switch: avoid descheduling points */
      for ( ; !stop ; )
         {
         process_id = RESETCH (link_to_master);
         if (process_id != MINT())  /* if other process hangs */
            RUNP (process_id); /* restart hanging process */
         if (stop)
            break;
         OUTBYTE (link_to_master, 0xFF);
         /*****************************************
         ** at this point the process hangs and     **
         ** the scheduler gives the CPU to the twin **
         ** process to "reanimate" this process     **
         *****************************************/
         }
      /*$SCHEDULE*/
   process: /* stopper process */
      {
      IN (&dummy, link_from_master, 1);
      stop = 1; /* set the shared variable */
      }
}
```

Figure 1: the fast algorithm, coded in parallel C

The following listing shows the principle function of the two twin processes. Uppercase words are assembler instructions, and *stop* is a shared variable, only writable by the stopper process not shown here. Each process runs until it reaches the OUTBYTE instruction. At this moment the process is descheduled and the other process continues with the instruction after OUTBYTE. You need GUY statements to write this in occam.

```
loop:                                  loop:
RESETCH                                RESETCH
if hanging                             if hanging
  RUNP                                   RUNP
if stop                                if stop
  exit_loop          deschedule          exit_loop
OUTBYTE to_master  <==========>  OUTBYTE to_master
if not stop                            if not stop
  goto loop                            goto loop
```

An implementation of the program written in parallel C is shown in figure 1.

Results

The parallel C implementation sends a byte every 6·2 microseconds. In assembler this time could be reduced to 5·5 microseconds. After placing the code and all variables (including the shared variable) into the on-chip RAM a data rate of 327 900 Bytes per second (at 10 Mbit/s) was reached. A byte is now sent every 3·05 microseconds. This is about 75% of the data rate of a T414-20 at 10 Mbit/s with acknowledge.

This speed is only possible if there are

- ▷ no process creations and terminations,
- ▷ no internal channels (they are replaced by the shared variable), and
- ▷ no `ALT` statements (the occam compiler generates 22 instructions for an `ALT`).

The control between the processes is switched by the hardware scheduler, not with channels. This unstructured implementation is useful in this case because the synchronisation must run as fast as possible, otherwise the system performance is not optimal. The number of instructions between the sending of two bytes is reduced from about 100 (for the `OutputOrFail` solution) to 15 (for the two twin processes). The program works in high or low priority, because there are no other processes running at this time. The program is extensively tested and runs without problems.

If the reconfiguration of the C004 is done at the moment when the slave sends a data packet, the master can receive a fragment of this packet. A single one-bit is interpreted as a link acknowledge. The link hardware tolerates a single unexpected acknowledge. The best bit pattern to use is `FF`(hex). Other bit combinations containing two or more single one-bits can confuse the receiver link hardware. Before polling the next slave the master link hardware has to be put into a defined state with a `RESETCH` instruction.

The disadvantage of this algorithm is that the waiting time between two tasks cannot be used for a reserve task because this synchronisation needs the whole CPU capacity. To use these waiting times, extra hardware has to catch the synchronisation bytes from the slave and tell the master which slave is ready. In this case the slaves send only one byte without acknowledge. During the waiting time a reserve task may work on the slave until the master fetches the result and sends a new task.

Experiments [5] show that this algorithm works faster than the polling algorithm with interrupts. Conventional message-passing implementations on a static tree network are significant slower in case of short tasks. Only implementations using the gaps between the tasks are faster. On 24 slave transputers a speedup of 23 is reached with an average task size of 30 milliseconds. The speedup of the interrupt polling is 20·5. The message-passing program spends most of the time on the communication, the speedup is only four at this task size.

If your transputer system will not boot any more

If you try to restart your system and a transputer sends without acknowledge in the direction of the root transputer, booting may be impossible. The host boots the root node with a hardware reset followed by the boot code on the link between host and root. Transputers are very fast, in most cases the uncontrolled sending slave transputer sends bytes into the gap between reset and boot code. The root transputer interprets the bytes from the slave as boot code and ignores the host. A

program made of lots of hex FFs followed by the old memory contents is not very useful and the root transputer does undefined things. So you first have to send a hardware reset to the uncontrolled sending slave.

References

[2] Roger Shepherd, *Extraordinary use of transputer links – Inmos technical note 1*, Bristol, 1987.

[3] Inmos Ltd, *The transputer instruction set – a compiler writers' guide*, Bristol, 1987.

[4] Jürgen Dammert, *Paralleles Ray-Tracing*, Diplomarbeit, Fakultät Informatik der Universität Stuttgart, Jan 1990.

[5] Dieter Ebhart, *Entwurf und Vergleich von Auftragsverteilungsalgorithmen für punktweise entkoppelte Probleme auf parallelelen Transputersystemen*, Diplomarbeit, Fakultät Informatik der Universität Stuttgart, Oct 1990.

Dieter Homeister
Universität Stuttgart
Institut für parallele und verteilte Höchstleistungsrechner
7000 Stuttgart 1
Azenbergstraße 12
Germany

Tel: +49 711-121-1342
Fax: +49 711-121-1424
homeister@informatik.uni-stuttgart.dbp.de

USING THE HIGH PRIORITY TRANSPUTER TIMER FROM WITHIN LOW PRIORITY PROCESSES

G. De Pietro and U. Villano
Centro di Studio sui Calcolatori Ibridi, Napoli, Italy

One of the most common problems for transputer users is the measurement of time intervals with a fairly high degree of accuracy. As is well known, every transputer provides free-running internal timers which can be straightforwardly used for local time measurements, i.e. for measurements concerning the activities of processes running in the same processor, as well as for suspending the execution of a process for a given interval of time. In particular, every transputer contains two timers, one that can be read only by high priority processes and another that can be accessed only by the low priority ones. The important thing to remember is that they have different resolutions: the hi-pri timer is incremented every 1 μsec, while the low-pri one every 64 ms. The rationale of the choice of a 'coarser' tick for the low priority timer is that a fairly high resolution would typically be of little use for low priority processes, as these run in quasi-concurrence time-slicing processor time, and can hence be descheduled several times during execution. It is clear, for example, that a descheduling which precedes a timer reading by a low priority process may introduce a non-negligible error in a time measurement. The

time needed for the measuring process to regain CPU control, spent waiting in the queue of active processes (*active process list*), can in fact be quite long and cannot be estimated in advance. A substantially analogous problem, affected by similar difficulties, is the suspension of a low priority process for a given interval of time. A low priority process suspended on a timer is placed onto the active process list as soon as the instant of time indicated in the delayed input construct is reached. Nevertheless, its actual wake-up comes with an unpredictable delay, which is highly dependent upon the number of processes preceding it in the list [6].

It could be concluded that a low priority process should never be used when a high degree of time measurement accuracy is essential. However, the above mentioned problems very often do not exist (e.g. because there is only one low-pri process running), and the utilization of a high priority process is not desirable or possible. In these cases it is possible to make use of some tricky solutions for reading the high priority timer from within a low priority process. The only way for a low priority process to access the high priority processes' timer by means of pure occam code is to execute a `PRI PAR` construct, as in the following scheme

```
SEQ
  ...
  PRI PAR
    ...  read timer
    SKIP
  ...
```

Most people (including ourselves) are not completely sure of the actual meaning of a `PRI PAR` construct which is not the outermost level of a program. However the occam compiler currently available supports this use of `PRI PAR`, which can be useful in many situations, as shown above. Nevertheless, some limitations on the use of the construct do exist, since `PRI PAR`s cannot be nested. This essentially precludes the reading of the high priority timer from within a low priority process running in parallel with one or several high priority processes. In this case the only possibility is to use a suitable assembler code that can temporarily raise the priority of the running process. This is the solution discussed in this article.

In the following sections we will first of all introduce some preliminary considerations on the transputer management of concurrent processes, showing the machine code that makes it possible to alter the priority of the current process. We will then present a function, written partly in occam and partly in transputer assembler code, that returns the current value of the high priority timer to a low priority calling process. Finally the adopted method will be extended to delayed inputs, illustrating a simple procedure that allows a low priority process to be descheduled for an interval of time measured by means of the hi-pri timer.

The transputer management of concurrent processes

In the first part of this section we will briefly recall the principal notions required for understanding the code presented below. However, for obvious space limitations we will assume that the reader has some familiarity with the transputer model of

execution and machine code programming. For any further detail the reader is referred to reference [7].

Every transputer process has a reserved area of memory, called its workspace, which is allocated at compile time and is used for holding the process local variables. The workspace is organized as a falling stack with end-of-stack addressing: during the execution of a process the workspace pointer (W_{ptr}) points to the bottom of the workspace, and the local variables are addressed as positive offsets from W_{ptr}. On the other hand, a number of locations with small negative offsets from W_{ptr} are directly used by the transputer hardware for scheduling, communications and timer input purposes. In particular the word of memory at location $W_{ptr} - 1$ is used to store the instruction pointer (I_{ptr}) of the process whenever this is held in the list of the processes waiting for CPU control or is unable to proceed because it has to wait for a communication/timer. It should be remembered that when a low priority process is interrupted by a high priority one that has become ready, the computation status (including the current value of I_{ptr}) is saved in particular memory locations at the bottom of the transputer memory map. The interrupted process will be restarted using this information as soon as the high priority process relinquishes CPU control, following its termination or a wait for an event (a future time, or a communication). According to this mechanism, at interrupt time the interrupted process is not placed onto the active process list, nor is its instruction pointer saved at location $W_{ptr} - 1$.

Every transputer process is completely identified by a descriptor, composed of the process workspace pointer W_{ptr} and the process priority (0 for high priority processes, 1 for low priority ones). Since workspaces are word-aligned in memory and hence the least significant bit of W_{ptr} is always low, the descriptor of a process can be obtained by storing its priority in the least significant bit of W_{ptr}.

The transputer instruction set provides several instructions for starting and terminating processes. The `runp` and `stopp` instructions are of particular interest for our purposes, as they allow a new concurrent process to be initiated at either priority level and to be terminated, respectively. When a `runp` is executed, the register A_{reg} should contain the descriptor of the process to be started, and the location at offset -1 from the W_{ptr} supplied in A_{reg} the starting address of the process code. The priority level of the newly started process can be determined by setting or clearing the least significant bit of the descriptor supplied in A_{reg}. A `stopp` terminates the executing process and saves the current value of the instruction pointer (i.e. the address of the instruction that immediately follows the `stopp`) at location $W_{ptr} - 1$.

The transputer provides no instruction for changing the priority of a running process in a straightforward way. The only possibility is to spawn a new process at different priority with a `runp`, and to terminate the current one with a `stopp`, as shown in greater detail below. Let us firstly suppose that we want to lower the priority of a hi-pri process. The sequence of instructions

```
ldlp 0
adc  1
runp
stopp
```

firstly builds in A_{reg} the descriptor of a process that has the same workspace as the current one (`ldlp 0` loads in A_{reg} the content of W_{ptr}) and low priority (`adc 1` sets

the least significant bit of A_{reg}). The `runp` places this new process onto the low-pri active process list, for which no starting address has yet been provided at location $W_{ptr} - 1$. The initial value of I_{ptr} for the spawned process is supplied by the `stopp` instruction, which terminates the high priority process and saves the current value of the instruction pointer at address $W_{ptr} - 1$ (remember that the activating and the activated process share the same workspace). The overall effect of the previous code is to lower the priority of the executing process, which is just what we intended to do. It should be noted that the spawned process has no possibility to run before the `stopp` has suitably loaded the memory location $W_{ptr} - 1$, because the process that performs the priority change cannot loose control of the CPU until the `stopp` is executed. There are two reasons for this: firstly because it is a high priority process, and secondly because the above code does not contain any descheduling instructions.

The dual problem, i.e. that of raising the priority of a low-pri process, is more complicated. The instruction sequence

```
ldlp 0
runp
stopp
```

is completely analogous to the one shown above, except that the priority bit of the descriptor built in A_{reg} is left equal to 0. However this code does not work properly: as soon as the `runp` is executed the spawned hi-pri process gains CPU control interrupting the activating process, which is not placed onto the scheduling list. So – unlike the previous case – the activation of the high priority replica of the executing process precedes the loading of $W_{ptr} - 1$, leading to unpredictable results.

It can be easily seen that even the following sequence for changing from low to high priority is not correct

```
ldc L
ldpi
stl -1
ldlp
runp
stopp
L:
```

In this case the high priority process is correctly initiated, since its activation is preceded by the loading of the address `L` of the instruction that immediately follows the `stopp` at the location $W_{ptr} - 1$. However, if the hi-pri process is descheduled before terminating (e.g. if it has to wait for an event) the value of its I_{ptr} at the time of suspension is stored at location $W_{ptr} - 1$, and the processor returns to the pre-interrupt state. Then the `stopp` of the activating process is executed, which saves the address `L` in $W_{ptr} - 1$, overwriting the information stored at that location, indispensable for the future re-activation of the high priority spawned process. In other words, this time not only does the `stopp` not load the location $W_{ptr} - 1$ before the actual activation of the process replica, but it is also dangerous, as the same workspace is shared between the high and the low priority 'twin' processes.

The simplest method for avoiding the drawback of the above code is to move the low-pri process W_{ptr} by using the `ajw` instruction. The shift of the pointer should be

such that the memory location $W_{ptr} - 1$ could be overwritten by the `stopp` without producing any undesirable effects, for example as shown below

```
ldc L
ldpi
stl -1
ldlp
runp
ajw S-Wptr+1
stopp
L:
```

where $S - W_{ptr}$ is the displacement (in words) between the bottom address of the process workspace, held in W_{ptr}, and a 'safe' location S.

An alternative – and more complex – solution requires the effect of the 'delayed' `stopp` to be considered beforehand. Sometimes, if suitable coding is adopted, it is possible to exclude any dangerous erasing of useful data. An obvious solution, for example, is to use (when possible) an instruction sequence such that the address stored by the `stopp` at location $W_{ptr} - 1$ is equal to the one that is overwritten, i.e. the one that was saved for successive re-activation of the spawned process. However, there is no general rule for doing this, and finding the right code to be used is a challenging exercise.

Input and delayed input using the high priority timer

In this section we will propose the code for performing inputs and delayed inputs using the hi-pri timer from within a low priority process. This code is fundamentally based upon the priority changing techniques shown above, and therefore all the readers who survived the previous section should be able to understand quite easily how it works. All the others can include the proposed routines in their occam programs without worrying any further. The *hi.time* function, shown in figure 2, returns the current value of the high priority timer to a low priority calling process. When the function is executed, the instructions

```
ldc 0
cj  change.priority
```

cause a jump to the code corresponding to the label *change.priority*. Note that this coding, as opposed to the use of a single jump instruction – `j change.priority` – prevents the process from being descheduled while executing the function [7, p 22]. The code

```
ldc -10
ldpi
stl -1
ldlp 0
runp
```

builds the descriptor of a hi-pri process with the same workspace as the current one, prepares at location $W_{ptr} - 1$ the pointer to the instruction corresponding to the label *read.timer* which is located 10 bytes before the instruction that follows the `ldpi`, and spawns the new process. This high priority process is immediately executed, interrupting the low priority one. It reads the (hi-pri) timer, stores the

```
INT FUNCTION hi.time()
  INT time:
  VALOF
    SEQ
      GUY
        LDC 0
        CJ .change.priority
        :read.timer
        LDTIMER
        STL time
        STOPP
        :change.priority
        LDC -10
        LDPI
        STL -1
        LDLP 0
        RUNP
    RESULT time
:
```

Figure 2: the *hi.time* function

result in the variable *time*, and stops (`stopp`). After the pre-interrupt state has been restored, the low priority process returns the value contained in *time* to the calling process. It should be noted that during the execution of the *hi.time* function the running process cannot be descheduled (it is interrupted for a short time, but this is substantially different) and hence the calling of the proposed function is functionally equivalent to executing an occam timer input.

The procedure *hi.delay* shown in figure 3 can be used for suspending a low priority calling process on the high priority timer for *delay* ticks of time (i.e. for *delay* microseconds). The instruction sequence

```
ldc 8
ldpi
stl -1
ldlp 0
runp
```

as usual spawns a hi-pri process with the same workspace as the current one. The twin hi-pri process is executed starting from the instruction with label *use.hi.timer* (which is located 8 bytes after the instruction that follows `ldpi`). It is activated immediately, as for the above shown function. The substantial difference between the two cases is that for positive (and sufficiently high) values of the input variable delay the spawned process is suspended on the delayed input before terminating, thus returning control to the low-pri twin process. However, the `stopp` of the low priority process does not overwrite the value of I_{ptr} that was previously saved, since its execution is preceded by the movement of the (low-pri) process workspace pointer (`ajw 1`), in such a way that $W_{ptr} - 1$ points to the address of the variable

```
PROC hi.delay (VAL INT delay)
  TIMER time:
  INT   now:
  INT   dummy:
  SEQ
    GUY
      LDC 8
      LDPI
      STL -1
      LDLP 0
      RUNP
      AJW 1
      STOPP
      :use.hi.timer
    time ? now
    time ? AFTER now PLUS delay
    GUY
      LDLP 0
      ADC 1
      RUNP
      STOPP
:
```

Figure 3: the *hi.delay* procedure

dummy. Hence at the end of the timer wait the high priority process will be correctly reactivated, and will lower its priority by executing the sequence

```
ldlp 0
adc 1
runp
stopp
```

which, as discussed in the previous section, spawns a low-pri twin process and terminates the hi-pri one. If, on the other hand, the value *now* PLUS *delay* is before the value of the hi-pri timer when the delayed input is executed, the high priority process is not descheduled, and terminates itself placing a further low priority replica process onto the active process list. Then the pre-interrupt state is restored and CPU control returns to the 'original' low-pri process, which terminates overwriting the contents of the location corresponding to the *dummy* variable.

It should be noted that the code in figure 3 adopts the first of the two approaches discussed in the previous section for raising the priority of a process. The procedure in figure 4, on the other hand, is based upon the second one, since the effect of the stopp executed by the low priority process before the termination of the suspended hi-pri twin is 'controlled'. However, it would be quite hard to explain how this code works, and it will not therefore be illustrated here.

Both versions of the *hi.delay* procedure are functionally equivalent to an occam delayed input, as performed by a low priority process. In other words, at wake-

```
PROC hi.delay (VAL INT delay)
  TIMER time:
  INT   now:
  SEQ
    GUY
      LDC 0
      CJ .change.to.hi
    time ? now
    time ? AFTER now PLUS delay
    GUY
      LDC 0
      CJ .change.to.low
      :change.to.hi
      LDC -15
      LDPI
      STL -1
      LDLP 0
      RUNP
      STOPP
      :change.to.low
      LDLP 0
      LDC 1
      OR
      RUNP
      STOPP
:
```

Figure 4: alternative *hi.delay* procedure

up time the waiting process is placed onto the low-pri active queue, and will be scheduled only after an unknown interval of time, with all the obvious consequences. Nevertheless, it should be borne in mind that any time-critical action can be executed before lowering the priority of the hi-pri spawned process, i.e. immediately after the delayed input statement contained in the proposed procedures.

References

[6] Martin Tørngren, *Transputer and occam based control systems in electronic control of machines*, OUG Newsletter Nº 12, January 1990, pp 66–70.

[7] Inmos Limited, *The transputer instruction set – a compiler writer's guide*, Prentice-Hall International, Hemel Hempstead, UK, 1988.

G. De Pietro and U. Villano
Centro di Studio sui Calcolatori Ibridi
via Claudio 21
80125 Napoli
Italy

Tel: +39 81 636683
Fax: +39 81 611826

A HARDWARE RANDOM NUMBER GENERATOR FOR TRANSPUTER SYSTEMS

Leslie Smith and Frank Kelly, Department of Computing Science, Stirling

A hardware random number generator based round a noise generating diode is presented. Although fast, the spread of numbers produced is not as smooth as hoped. Some suggestions are made for its improvement.

Introduction

Research in Neural Networks at the University of Stirling has frequently required large numbers of random numbers for generating noise for signals and stochastic pseudo-neurons. In simulations of neural nets processing time-varying signals, many thousands of such numbers may be required each second, making fast random number generation important. There is no requirement that precisely the same sequence of random numbers be repeated, so that we need not be restricted to seed-based random number generation. Hardware random number generation was perceived as a relatively straightforward task, something not really borne out by later experience.

The generator

Figure 5 shows a block diagram of the generator. Noise generation is achieved by the reverse biased noise generating diode (NC202). This is guaranteed to produce white noise from 20 Hz to 25 MHz. This very small signal (about 0·1 μv) was then amplified by a four stage amplifier based on the Signetics NE5539 wideband operational amplifier. The resulting analogue signal is converted to TTL levels by one channel of a DS26LS32 line receiver, followed by a Schmitt trigger. The resultant 'digital' noise signal is then fed to an 8-bit shift register clocked at 10 MHz. This gives a theoretical limit to the time for number generation of 0·8 μs. This part of the circuit runs continuously.

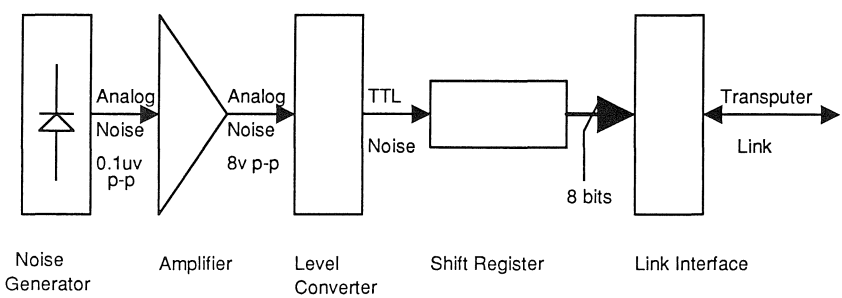


Figure 5: noise generator block diagram

The shift register is interfaced to a transputer link by a C011 link interface chip, running as a parallel interface. It is set up so that the receipt of a byte on the

link causes the generator to send out a stream of random bytes, stopping only when another byte is received on the link. When the C011 is ready to send out a new random byte, it waits for the shift register to receive a new set of eight bits, and then causes the output of the serial to parallel converter to be latched. The data is then sent out by the C011. Running the link at 10 MHz, it has been found that the remote transputer receives a new random byte every 2·74 μs. A more detailed circuit description is available from the authors.

Testing the generator

Three tests have been tried on the generator: frequency checks on single number generation, simple frequency of ones and noughts, and frequency of pairs of numbers rising, falling, or staying the same. Additionally, the frequency of recurrence of single bytes, of pairs of bytes, and of sequences of three bytes have been checked.

The generator performs worst on frequency of single number generation. Experimentally, it has been found that certain numbers, notably those with strings of ones, occur more frequently. This sort of systematic error results in very high values of χ^2 when large numbers of random bytes are produced. The actual effect is most pronounced on the number 255, which occurs about 1·8 times as often as it should; the results for other numbers are much less pronounced, with 127 being produced 1·2 times as often, though values for 63 and 191 vary widely.

The results for simple frequency of ones and noughts are better; the system is slightly biased towards ones. On a sample of 100 000 bytes (i.e. 800 000 bits) the χ^2 value produced varies between 3·5 and 11·2, when it should be between 0·00016 and 6·635. Again, this value rises on larger samples.

For the test of pairs of numbers rising or falling, or staying the same, the results are better: using 2 560 000 numbers, values of χ^2 of between 0·35 and 8·49 were found, when reference [8] suggests 0·0201 to 9·21. We also looked at recurrence of numbers: these were found to occur at about the correct frequency for single bytes (i.e. about once per 2^8), for double bytes (about once per 2^{16}) and triple bytes (about once per 2^{24}).

Comments and conclusions

Many difficulties were encountered in the construction of the equipment. Largely, these came from an underestimation of what was involved in the mixing of sensitive analogue and standard digital circuitry in one box. Eventually, the analogue circuitry was completely enclosed, and used an entirely independent power supply. There were also problems with oscillation due to the large amount of wideband gain required. These problems can be laid at the door of our lack of experience in non-digital construction. Before trying the noise-generating diode, we tried a number of ordinary and zener diodes. The magnitude of the noise signal generated by these was an order of magnitude smaller, making the problems of amplification much greater.

The results are a little disappointing, although good enough for the application itself. We believe the problems causing the systematic lack of randomness come from two sources: the level conversion, and the bandwidth of the noise signal. The level conversion circuit is originally a balanced line receiver: it is perhaps not as

accurate in its detection of a 0 V signal as required. This could lead to the slight preponderance of ones over noughts. The noise generating diode is supposed to give out noise evenly over a very large band: however, we believe that the noise falls away a little towards the high frequency end of the spectrum, leading to numbers which contain strings of the same digit. We do find numbers with strings of ones, but do not find numbers with strings of noughts. Further, the imbalance between noughts and ones is very small, so we remain rather uncertain of the precise nature of this problem.

Since the results were good enough for the application, and since the primary aim of the project is in neural net simulation, not random number generation, we have not pursued the matter further. Were we to do so, we would try a higher bandwith noise generating diode, plus some filtering to achieve an optimal spread of noise frequencies (though we note that we are not certain of what that should be).

References

[8] D. E. Knuth, *Seminumerical algorithms*, Addison-Wesley, 1981.

Leslie Smith and Frank Kelly
Department of Computing Science
University of Stirling
Stirling
Scotland
United Kingdom

TRANSPUTER HARDWARE STANDARDS SURVEY

James Kidd, Altis

There are many designers and manufacturers who have been assembling systems containing several transputers. Since most designers have been unaware of the work of others, there are now almost as many ways of connecting these systems together. To help rectify this situation, this document collects together information about existing methods of building transputer based systems.

It is intended that this document should be used by designers of transputer based systems as and aid to selecting suitable methods for connecting transputers. It is recommended that, where necessary, readers write or phone the relevant contact (given on page 46) in order to obtain the most complete and up-to-date information.

At the Hardware SIG session of the 12th technical meeting of the Occam Users Group at Exeter it was felt worthwhile to conduct a survey of existing transputer standards. Two areas of compatability were identified as important to allow interworking between equipment. Firstly, hardware issues such as transputer link buffering, control signalling and connector arrangement. Secondly, software issues such as the port maps of the transputers and any host computer. Many designers and manufacturers were then sent questionnaires requesting information about these issues. The replies have been assembled to form this document.

Any readers wishing to contribute to this survey are invited to contact James Kidd at Altis for further copies of the questionnaire. All replies will be included in

Organisation	Altis
Address	9 Brackley Street, Warrington WA4 6DY, UK
Telephone	(+44) 925 601 735
Engineering Contact	James Kidd
Position	Proprietor
Organisation	Computer System Architects
Address	950 N. University Avenue, Provo, Utah, USA 84604
Telephone	(+1) 801 374 2300
Organisation	Gnome Computers Ltd
Address	16 Histon Road, Cambridge CB4 3LE, UK
Telephone	(+44) 223 461 520
Engineering Contact	Steve Temple
Organisation	Impuls Computer-systeme GmbH & Co KG
Address	A-1232 Wien, Jochen Rindt-Strasse 9, Austria
Telephone	(+43) 222 616 24 01-0
Engineering Contact	DI Herbert Widner
Position	Head of Development
Organisation	Inmos Ltd
Address	1000 Aztec West, Almondsbury, Bristol BS12 4SQ, UK
Telephone	(+44) 454 616 616
Engineering Contact	Paul Walker
Organisation	Meiko Ltd
Address	650 Aztec West, Almondsbury, Bristol BS12 4SD, UK
Telephone	(+44) 454 616 171
Organisation	Paramax Corporation
Address	101B Halmar Cove, Georgetown TX 78628, USA
Telephone	(+1) 512 869 0115
Engineering Contact	David Smith
Position	Engineering Manager
Organisation	Parsys Ltd
Address	Boundary House, Boston Road, London W7 2QE, UK
Telephone	(+44) 81 579 8683
Engineering Contact	Stuart Kennedy
Position	Design Engineer
Organisation	SJ Research Ltd
Address	Intercell, 1 Coldhams Lane, Cambridge CB1 3EP, UK
Telephone	(+44) 223 461 406
Engineering Contact	Kim Spence-Jones
Position	Managing Director, Technical
Organisation	T2 Systems Ltd
Address	62 Longmead Avenue, Bishopstoke, Eastleigh, Hants SO5 6ET, UK
Telephone	(+44) 703 641 276
Engineering Contact	Patrick Pope
Organisation	Telmat Informatique
Address	BP12, route de l'Industrie, 68360 Soultz, France
Telephone	(+33) 89 76 55 55
Engineering Contact	Mr Bertrand Blum
Position	Supercomputer Department Engineer
Organisation	White Cross Systems Ltd
Address	90 London Road, London SE1 6LN, UK
Telephone	(+44) 71 922 8824
Engineering Contact	Roger Gaskell
Position	Technical Manager

Summary of transputer link buffering

		Signal levels				Line term	
		TTL	422	ECL	Oth	TXr	RXr
1	T2 Systems Ltd	x	.	.	.	x	.
2	T2 Systems Ltd	x	.	.	.	x	.
3	T2 Systems Ltd	x	.	.	.	x	.
4	Paramax Corporation	x	.	.	.	.	.
5	Gnome Computers Ltd	x	.	.	.	x	.
6	Impuls Computer-Systeme	.	x	.	.	.	x
7	Telmat Informatique	x	x	x	.	x	x
8	Parsys Ltd	.	.	x	.	.	x
9	Parsys Ltd	.	x	.	.	.	x
10	Parsys Ltd	x	.	.	.	x	.
11	White Cross Systems Ltd	.	.	x	x	.	x
12	SJ Research Ltd	.	x	.	.	.	x
13	Altis	.	.	x	.	x	.
14	Altis	x	.	.	.	x	.
15	CSA	.	x	.	.	.	.
16	Meiko Ltd	.	.	x	.	x	.

Remarks: 1 Tx:47ohm series, Rx:4k7ohm pull down
2 Compatible with TRAMs/B008
11 Optical links

TTL = TTL; 422 = RS422; ECL = ECL; Oth = Other
TXr = Transmitter; Rxr = Receiver

Summary of control signalling

		Configuration				Signals					Levels			
		U/D	SSs	MSs	Oth	Res	Ana	Ers	Erh	Oth	TTL	422	423	Oth
1	T2 Systems Ltd	.	.	.	.	x	.	.	.	.	x	.	.	.
2	T2 Systems Ltd	x	x	.	.	x	x	x	.	.	x	.	.	.
3	T2 Systems Ltd	.	x	.	.	x	x	x	.	.	x	.	.	.
4	Paramax Corporation	.	.	.	.	x	x	x	.	x	x	.	.	.
5	Gnome Computers Ltd	x	x	.	.	x	x	x	.	.	x	.	.	.
6	Impuls Computer-Systeme	.	x	.	.	x	x	x	.	.	.	x	.	.
7	Telmat Informatique	x	x	.	.	x	x	x	x	.	x	.	.	.
8	Parsys Ltd	x	.	.	.	x	x	x	.	.	x	.	.	.
9	Parsys Ltd	.	.	.	.	.	.	.	.	.	.	.	.	.
10	Parsys Ltd	.	.	.	.	.	.	.	.	.	.	.	.	.
11	White Cross Systems Ltd	.	.	x	.	x	x	.	x	x	.	x	.	x
12	SJ Research Ltd	.	.	.	.	.	.	.	.	x	x	.	.	.
13	Altis	.	.	.	.	.	.	.	.	.	.	.	.	.
14	Altis	x	x	.	.	x	x	x	.	.	x	.	.	.
15	CSA	x	x	.	.	x	x	x	.	.	x	.	.	.
16	Meiko Ltd	x	.	.	.	x	x	x	x	.	.	.	x	.

Remarks: 1 TRAM compatible
4 Also provide O/C Error
5 Conform to Inmos TN49
11 Optical signals

U/D = Up / Down; SSs = Single Subsystem; MSs = Multiple Subsystem; Oth = Other
Res = Reset; Ana = Analyse; Ers = Error (software); Erh = Error (hardware)
TTL = TTL; 422 = RS422; 423 = RS423

Summary of connectors and cabling

		Config			Use		
		Com	Sep	Lnk	Wth	Btw	Remarks
1	T2 Systems Ltd	.	.	.	.	.	
2	T2 Systems Ltd	.	x	.	x	.	DB37
3	T2 Systems Ltd	.	x	.	x	.	
4	Paramax Corporation	.	.	.	.	.	SIMM
5	Gnome Computers Ltd	.	x	.	x	.	Inmos style 5-way
6	Impuls Computer-Systeme	x	.	.	.	x	96w DIN 41612 connector
7	Telmat Informatique	.	x	.	x	x	several standards
8	Parsys Ltd	.	.	x	x	x	16w IDC
9	Parsys Ltd	.	.	.	.	.	
10	Parsys Ltd	x	.	.	.	x	DB37 (as B008)
11	White Cross Systems Ltd	x	.	.	x	x	SMA/Optical
12	SJ Research Ltd	.	.	x	.	x	DB9/MiniDIN
13	Altis	.	.	x	.	x	DB9
14	Altis	.	x	.	x	.	2-row r/a PCB header
15	CSA	x	.	x	x	x	MiniDIN
16	Meiko Ltd	x	.	x	.	x	DB9/DB37

Com = Combined links and control; Sep = Separate links and control; Lnk = Links only
Wth = Within equipment; Btw = Between equipment

Summary of transputer port maps

		I/O ports				Remarks
		SSs	MSs	RPL	Oth	
1	T2 Systems Ltd	x	.	.	.	
2	T2 Systems Ltd	x	.	.	.	
3	T2 Systems Ltd	x	.	.	.	
4	Paramax Corporation	.	.	.	.	No current ParaSIMMs have a subsystem port
5	Gnome Computers Ltd	x	.	.	.	B004/B008 compatible
6	Impuls Computer-Systeme	x	.	.	.	
7	Telmat Informatique	x	.	.	.	
8	Parsys Ltd	.	.	.	x	Supernode control bus interface
9	Parsys Ltd	.	.	.	.	
10	Parsys Ltd	.	.	.	.	
11	White Cross Systems Ltd	.	.	.	x	
12	SJ Research Ltd	.	.	.	.	
13	Altis	x	.	.	x	B004/B008 compatible superset
14	Altis	.	.	.	.	
15	CSA	.	.	.	.	
16	Meiko Ltd	.	.	.	.	

SSs = Single subsystem; MSs = Multiple subsystem; RPL = Reset only per link; Oth = Other

Summary of host port maps

		Host machine							
		ISA	MCA	Mac	Sun	Apo	VAX	Arc	Oth
1	T2 Systems Ltd	.	.	.	.	.	.	.	.
2	T2 Systems Ltd	x	.	.	.	.	.	.	.
3	T2 Systems Ltd	.	.	.	.	.	.	.	.
4	Paramax Corporation	.	.	.	.	.	.	.	.
5	Gnome Computers Ltd	.	.	.	.	.	.	x	.
6	Impuls Computer-Systeme	x	.	.	.	.	.	.	.
7	Telmat Informatique	x	.	.	x	x	.	x	.
8	Parsys Ltd	x	x	.	x	.	.	.	.
9	Parsys Ltd	.	.	.	.	.	.	.	.
10	Parsys Ltd	.	.	.	.	.	.	.	.
11	White Cross Systems Ltd	.	.	.	.	.	.	.	x
12	SJ Research Ltd	.	.	.	.	.	.	x	.
13	Altis	x	.	.	.	x	.	.	.
14	Altis	.	.	.	.	.	.	.	.
15	CSA	.	.	.	.	.	.	.	.
16	Meiko Ltd	.	.	.	.	.	.	.	.

Remarks: 4 Do not mnf mothercards
11 Self-hosted
13 B008 compatible superset

ISA = PC/XT/AT (ISA bus); MCA = PS/2 (MCA bus); Mac = Macintosh; Sun = Sun workstation; Apo = Apollo workstation; VAX = DEC VAX; Arc = Acorn Archimedes; Oth = Other

later versions of this document.

It must be said that not all replies received so far have been of a high standard or contained much relevant information. The summaries in this document contain extracts from all replies but, where the replies contained little or no further information, they have been omitted from the main body of the report and relegated to appendices in the full document [editor's note: only the summaries appear in this article]. Interested readers are invited to write or phone the relevant contact in order to obtain any further details.

Paul Walker replied on behalf of Inmos stating that they would prefer that users refer to the published literature rather to an abbreviated summary. Computer System Architects and Meiko Ltd were unable to complete the survey forms but instead supplied copies of relevant parts of their literature.

Worthy of special attention are the Nexus standards supplied by Kim Spence-Jones of SJ Research Ltd and which are currently supported by Atari, Active Book Company, Microrobotics and Capricorn Computing. These standards include, not only the connectors and signalling, but also the link, network and transport protocols. Full details are available from Kim.

Although it is not really covered by this survey, details of a board level standard were supplied by David Smith of Paramax Corporation. These boards are similar to TRAMs in concept but are based on the SIMM connector often used for DRAM modules. Full details of their standards and products are available from David.

The list on page 46, in alphabetical order, provides details of the organisations

that have supplied information to this survey.

James Kidd +44 925 601 735
Altis
9 Brackley Street
Warrington WA4 6DY
United Kingdom

TRANSPUTER IMPLEMENTATION OF GENERAL SEMAPHORES

Murat Tayli, Mohamed Benmaiza, King Saud University

This article presents a dynamic and general solution for the implementation of predictable and high performance primitives operating on general semaphores. The solution is coded in assembly, using occam as the harness language.

Introduction

Dijkstra's concurrency control paradigm, the semaphore [9], has ever been the ultimate building block for the implementation of synchronization mechanisms in shared memory systems. Systems based on the *Communicating Sequential Process* (CSP) model [10] adopted the point-to-point handshaking communication scheme as the basic synchronization mechanism. Despite the duality of these models, there are instances where systems built on CSP model may still resort to the use of semaphores. A major incentive is the restrictions exerted by a static interpretation of CSP model. Another is a more efficient implementation of some concurrency control structures using semaphores [12].

Our interest in semaphores stemmed from the need to define dynamic, fast, and predictable synchronization primitives for a real-time kernel, implemented on a network of transputers [11]. Design objectives of this system required the adoption of a dynamic process management model and provisions for dynamic concurrency control mechanisms, capable of handling ever changing populations of processes.

In what follows we will first analyze implementation problems, review a number of proposed solutions, and then present an alternative approach that satisfies stringent requirements of our real-time environment.

Presentation of the problem

Primitives operating on semaphores are defined as indivisible actions [9]. Their correctness rely on the atomicity of the operations that preserves the integrity of shared objects (i.e. semaphore's process queue, counter, lock etc.). Implementation of predictable and general semaphore operations on the unconventional transputer hardware represents a real technical challenge. Difficulties stem not only from the lack of direct hardware support to implement indivisible operations, but also from a blurred vision of the system through high-level programming languages.

Atomicity of the operations in a system is only endangered by the pre-emption of the CPU on the occurrence of uncontrolled events. A brief analysis of transputer

architecture reveals that it mainly uses non-preemptive process management policies. For instance, high priority processes relinquish the CPU only voluntarily (on blocking requests), and low priority processes are descheduled to the benefit of each other only on limited occasions (on blocking requests, iterations and unconditional jumps). Consequently, for processes on the same priority level, the atomicity is naturally guaranteed, if the use of descheduling and blocking instructions can be avoided in critical sections. Assuming that this constraint can be easily met, at least at machine code level, the only uncertainty left stems from high priority processes that are allowed to interrupt low priority processes at any time. This problem can be overcome either by executing critical sections at high priority level, or by devising adequate serialization mechanisms that guarantee exclusive access to critical sections. Previous issues of this *Newsletter* included several implementation proposals that can be analyzed to gain better understanding of the problem area.

Barrett proposed in [12] an elegant solution for processes running at the same priority level. Though his primitives were defined in an occam-like language, he exclusively used the properties of the underlying hardware to guarantee the integrity of his operations (i.e. priority levels restricted to high and low, uninterruptibility guarantee for high priority processes, non-preemptive scheduling policy between low priority processes, synchronized and non-busy channel communication). Barrett's single priority solution has a number of static or dynamic implementation alternatives, depending on the interpretation of the abstract data type `LIST OF CHANNELS`. However, a static implementation would be too restrictive, and would render its usage inadequate for the dynamic context defined in the introduction. Unfortunately, Barrett's multiple priority implementations do not match the efficiency and elegance of his single priority solution. This is because he disregarded the properties of the transputer hardware, which he exploited so well in the single priority solution. The resulting primitives are expensive, and introduce coding disparities between processes of different levels.

S. W. Lau and F. C. M. Lau's single priority solution [14] is a successful implementation of Barrett's ideas. They efficiently used the duality of communication and process management primitives to build a dynamic `LIST OF PROCESSES`, as a substitute for the original `LIST OF CHANNELS`. However, their multiple priority solution is still prone to the same drawbacks as Barrett's proposal.

Homeister [13] used the 'Reset Channel' (`RESETCH`) instruction as an effective substitute of traditional 'test-and-set' type instructions to guard his critical sections. His approach falls into 'algorithmic solutions to the critical section problem' category [15], rather than being a direct implementation of semaphores. Consequently, proposed solutions are neither general nor dynamic. Moreover, the 'Busy Implementation' does not work properly for multiple priority processes as claimed. For instance, a high priority process that pre-empts a low priority one, in a critical section, will monopolize the CPU and cause a deadlock.

Proposed solution

Solutions analyzed in the previous section have all adopted complex synchronization schemes to implement atomic actions. Also, they considered single and multiple priority cases separately. In our view, transputers provide sufficient architectural

support for the implementation of atomic actions. Thus, the use of heavy algorithmic mechanisms (that already rely somehow on the hardware characteristics) is neither necessary nor desirable. Moreover, adequate solutions to the problem should be general enough to address both the single and multiple priority cases. Furthermore, these solutions should incur similar and predictable costs for all priority levels. Finally, they must be efficient.

Given the criteria we defined and the characteristics of the target system, it is clear that satisfactory solutions cannot but be machine dependent. Therefore, the proposed primitives will be coded at transputer instruction level, in order to avoid the unaccounted overhead that could be introduced by high level language compilers, as well as to exclude all possibility of generating blocking and descheduling actions in critical sections. As for the serialization mechanism (that is needed to solve high versus low priority contentions), it will simply differ the execution of conflicting operations until the interrupted process is pushed out of the critical section. In other words a high priority process pre-empting a low priority process in a conflicting situation will resume the interrupted sequence, exit from the pre-empted critical section, and then proceed with its own operation. This approach has the merit of being free of extra process synchronization idiosyncrasies, and preserves the real-time sequencing of the events.

Implementation details

The implementation of proposed primitives, namely the `P` (for resource-request/wait) and the `V` (for resource-release/signal) operations, is given in figure 6. Primitives are coded as occam procedures, using the TDS-PC environment. Figure 7 corresponds to the context switching sequence common to both primitives. The semaphore object is implemented using an integer array of four elements. The variable *Sem.lock* guards the access of the semaphore, while it is in use. The `RESETCH` operation (described in detail in [13]) is used as an indivisible operation to test and set the lock variable. The value 0 represents a free semaphore, and the smallest integer (`MININT`) a busy semaphore. Processes waiting on the semaphore queue are linked following the conventions used in other transputer lists, such as 'active process lists' and 'timer lists' [16]. In the workspace of each process, the location $Wptr - 2$ points to the workspace of the next process in the queue. Queues are organized in FIFO, but a priority based ordering can also be easily implemented. In `P` and `V` procedures, the number and placement of local variables are context sensitive. A high priority process can pre-empt a low priority process in a `P` (or a `V`) operation, and invoke a `P` (or a `V`) primitive itself, accessing the same semaphore. Therefore, a `P` (or a `V`) may have to finish a pending `P` or `V` primitive. Consequently these primitives should have similar workspaces to cope with this constrained type of re-entrancy. Moreover it is essential that the variable *Low.Iptr* is assigned to the location $Wptr + 0$ (that is what the TDS occam compiler generates for the code provided in figure 6) to start properly the interrupted process (see the `ALTEND` instruction in the next paragraph).

The context switching sequence (figure 7) consists of the following steps:

- compute and save the restart address of the high level process (it may be either in a `P` or `V` operation),
- restore the working context (workspace) of the interrupted process,

```
-- definition of semaphore structure
VAL Sem.lock   IS 0          : -- Lock to guard semaphore access
VAL Sem.val    IS Sem.lock+1 : -- Semaphore value
VAL Sem.q.head IS Sem.lock+2 : -- Process wait queue head pointer
VAL Sem.q.tail IS Sem.lock+3 : -- Process wait queue tail pointer
--
-- definition of the workspace frame
VAL Wptr       IS 0          : -- Word used by ALTEND instruction
VAL Wdesc.iptr IS Wptr-1     : -- Iptr save area
VAL Wdesc.next IS Wptr-2     : -- Pointer to next process in queue
--
-- process save area layout
VAL WdescIntSaveLoc IS 11    :
VAL IptrIntSaveLoc  IS 12    :
VAL AregIntSaveLoc  IS 13    :
VAL BregIntSaveLoc  IS 14    :
VAL CregIntSaveLoc  IS 15    :
--
-- miscellaneous constants
VAL mint            IS MOSTNEG INT  :
VAL low.priority    IS 1            :
VAL sem.unlocked    IS 0            :
VAL sem.locked      IS mint         :
--
--------------------------------------------------------------
--           Initialization of Semaphore
--------------------------------------------------------------
PROC Init.Sem ( []INT Sem , VAL INT n )
  INT Sem.lock  IS  Sem[Sem.lock] :
  INT Sem.val   IS  Sem[Sem.val] :
  SEQ
    Sem.val    := n             -- set semaphore count
    Sem.lock   := 0             -- set semaphore access to "free"
:
```

Figure 6: implementation of the semaphore operations

```
------------------------------------------------------------
--              Request Resource (Wait Event)
------------------------------------------------------------
PROC P ( []INT Sem)
  INT Process.to.awake, Original.priority :
  INT Save.High.Wptr, Save.High.Iptr      :
  INT Low.Iptr                            :-- must be AT Wptr+0

  SEQ
    GUY
      LDPRI
      STL   Original.priority -- save priority of calling process
      LDLP  Sem               -- test and set semaphore lock
      RESETCH                 -- Sem.lock <- MINT ; Areg<-Sem.lock
      CJ     .itisfree        -- jump if (Sem.lock was = MINT)

      ... switch to the context of pre-empted low priority process

      :itisfree               -- semaphore access is free
      -- sem.val := sem.val - 1
      LDLP    Sem
      LDNL    Sem.val
      ADC    -1
      LDLP    Sem
      STNL    Sem.val
      --  Test Sem.val
      LDC     0
      LDLP    Sem
      LDNL    Sem.val
      GT
      CJ    .accessgranted    -- jump if Sem.val >= 0
      -- ifcase : Sem.val < 0 block the process
      LDLP    Sem             -- test if process queue is empty
      LDNL    Sem.val
      EQC    -1
      CJ     .nonemptyqueue   -- jump if (Sem.val <> -1)
      -- empty queue => enqueue at the beginning of the list
      LDLP    Wptr
      LDL     Original.priority
      SUM
      LDLP    Sem
      STNL    Sem.q.head      -- update process queue head pointer
      LDC     0
      CJ      .updatetail
```

Figure 6: implementation of the semaphore operations (P)

```
:nonemptyqueue
LDLP Wptr
LDL  Original.priority   -- restore the original priority
SUM
LDLP  Sem
LDNL  Sem.q.tail
STNL  Wdesc.next         -- [tail->Wptr.next]<-current Wptr
:updatetail
LDLP  Wptr               -- update process queue tail pointer
LDLP  Sem
STNL  Sem.q.tail
-- stop process
LDPRI
LDL     Original.priority
DIFF
CJ     .nopreemption
-- preemption
LDC  11             -- reinitialize Iptr of pre-empted process
LDPI                -- to restart at 'STOPP' instruction
:buildcontext
MINT
STNL    IptrIntSaveLoc
LDL     Save.High.Wptr
GAJW                -- restore original Wptr
LDL     Save.High.Iptr
GCALL         -->>>>>>> resume the execution of high priority
              -->>>>>>> process alone in the critical section
:nopreemption
LDC     sem.unlocked
LDLP    Sem         -- free  semaphore access
STNL    Sem.lock
         -- <<<<<< restart point of preemted low priority
STOPP    -- <<<<<< process which is being queued
J      .endP
-- ifcase : Sem.val >= 0
:accessgranted
LDPRI
LDL  Original.priority  -- if original priority=current pri.
DIFF                    -- then
CJ   .passingsemaphore  -- no pre-emption occurred
LDC  5              -- reinitialize Iptr of pre-empted process
LDPI                -- to restart at endP
J    .buildcontext
:passingsemaphore
LDC     sem.unlocked
LDLP    Sem             -- free the semaphore access
STNL    Sem.lock

:endP
```

Figure 6: implementation of the semaphore operations (P continued)

```
------------------------------------------------------------
--                Release Resource (Signal event)
------------------------------------------------------------
PROC V ( []INT Sem )
  INT process.to.awake, Original.priority :
  INT Save.High.Wptr, Save.High.Iptr      :
  INT Low.Iptr                            :-- must be AT Wptr+0
  SEQ
    GUY
      LDPRI
      STL  Original.priority  -- save priority of calling process
      LDLP    Sem             -- test and set semaphore lock
      RESETCH                 -- Sem.lock <- MINT ; Areg<-Sem.lock
      CJ   .itisfree          -- jump if (Sem.lock was = MINT)

      ... switch to the context of pre-empted low priority process

      :itisfree             -- semaphore access is free
      -- sem.val := sem.val + 1
      LDLP    Sem
      LDNL    Sem.val
      ADC     1
      LDLP    Sem
      STNL    Sem.val
      -- Test Sem.val
      LDLP    Sem
      LDNL    Sem.val
      LDC     0
      GT
      CJ     .freeprocess   -- jump if Sem.val <= 0

      -- ifcase : Sem.val > 0
      :NoProcessToAwake
      LDPRI
      LDL     Original.priority
      DIFF
      CJ     .exitV          -- no pre-emption to consider
      LDC     25             -- compute displacement to endV
      LDPI
      ------------------> (this_address - afteraltend) = 25 bytes
      MINT
      STNL    IptrIntSaveLoc
      LDL     Save.High.Wptr -- restore the original context
      GAJW
      LDL     Save.High.Iptr
      GCALL          -->>>>>> resume the execution of high priority
                     -->>>>>> process alone in the critical section
```

Figure 6: implementation of the semaphore operations (V)

```
-- ifcase : Sem.val <= 0
:freeprocess
LDLP   Sem              -- get the workspace @ of the first
LDNL   Sem.q.head       -- process waiting in semaphore queue
STL    process.to.awake
LDL    process.to.awake
LDNL  Wdesc.next        -- get next proc. wptr @ in the queue
LDLP  Sem
STNL  Sem.q.head        -- update wait queue head pointer
LDL   process.to.awake -- awake dequeued process
RUNP
LDC     0
CJ     .NoProcessToAwake
:exitV
LDC     sem.unlocked
LDLP    Sem
STNL    Sem.lock        -- free the semaphore access

:endV
:
```

Figure 6: implementation of the semaphore operations (V completed)

- save the workspace reference of the high priority process in the context of the interrupted process,
- restore registers of the interrupted process,
- start executing the remaining code of the interrupted critical section (as the execution continues in high priority, no other pre-emption can occur).

The transfer of control to the interrupted code is achieved using the ALTEND instruction. ALTEND is just a conventional relative jump instruction. It also has the interesting property of preserving the contents of CPU register, which is not the case of the dynamic procedure call, the GCALL instruction. Note that ALTEND uses the location $Wptr + 0$ to get the displacement for the jump relative to the instruction pointer. Thus this location has to be reserved and not used by the P and V operations. Jump (J) instructions that are deliberately used in both primitives will not cause any descheduling since they will be only executed by high priority processes.

Correctness proof

The correctness of the proposed solution is based on:

- the atomicity of the execution of P and V operations. By construction, any semaphore operation comes to a completion before another operation on the same semaphore can start. This is true independently of the priorities of the processes. Consequently, processes can be simultaneously in the same critical section.
- free progress in the P and V operations. Both operations are loop-free and do change neither the status nor the priority of racing processes. Any process will eventually leave or will be pushed out of the critical section.

```
--
-- Switch to the context of low priority process being pre-empted
--
LDC   28              -- restart address of the high priority
LDPI                  -- process is 'afteraltend'
---------------->   (afteraltend - this_address) = 28 bytes
STL     Save.High.Iptr
MINT
LDNL  WdescIntSaveLoc -- Areg<-Wdesc of low priority process
ADC   -1              -- remove priority bit
GAJW                  -- exchange workspaces of low and high
                      -- priority processes

STL   Save.High.Wptr  -- save Wdesc of high priority process
MINT                  -- compute entry point of pre-empted process
LDNL  IptrIntSaveLoc
LDC   13
LDPI
---------------->   (afteraltend - this_address) = 13 bytes
DIFF
STL   Low.Iptr    -- [Wptr+0] <- (Saved_low_priority_Iptr-13)
--
MINT
LDNL  CregIntSaveLoc  -- restore Creg
MINT
LDNL  BregIntSaveLoc  -- restore Breg
MINT
LDNL  AregIntSaveLoc  -- restore Areg
--
ALTEND          -- relative jump to (NextInst+[Wptr+0])
                -- (preemtion point of low priority process)
:afteraltend    -- high priority process resumes here
```

Figure 7: context switching sequence common to P and V

Performance analysis

The table in figure 8 summarizes execution costs for P and V primitives, expressed in transputer cycles and in microseconds (assuming 50 nanosecond machine cycles). Listed figures correspond to the performance of the primitives under various circumstances, and do not include procedure invocation costs. In the TDS occam implementation, a call of a P or V may take from 9 to 16 cycles, depending on the linkage sequence and the size of the program. Assuming an even distribution of blocking and non-blocking cases, average costs (including invocation sequence) are approximatively 4 microseconds for a P, and 3·6 microseconds for a V operation. Minimum execution times for both operations is around 3 microseconds. The worst and unlikely case is the pre-emption of a low priority process in a P or V operation, using the same semaphore as the pre-empting process. Worst case figures come to

		Non-preemption		Preemption and contention	
		cycles	μsec	cycles	μsec
P	non-blocking	49	2.45	99	4.95
	blocking	88	4.4	136	6.8
V	without activation	46	2.3	95	4.65
	with activation	75	3.75	122	6.1

Figure 8: performance of semaphore primitives

7·5 microseconds for a P, and to 6·8 microseconds for a V operation.

Conclusion

The proposed solution meets general criteria observed in the implementation of synchronization tools. It ensures mutual exclusion and free progress of concurrent processes; it does not cause starvation (although mechanisms built with it can); it induces non-busy waits; and it is fair. In addition, it encompasses both single and multiple priority cases, and offers a unique system interface. More importantly, it addresses stringent requirements of our real-time environment. In particular, execution costs of P and V primitives are:

- ▷ low (averaging less than 4 microseconds),
- ▷ independent of process priorities, and
- ▷ predictable, since they do not rely on asynchronous events (e.g. channel synchronization, process scheduling) to ensure the serialization.

References

[9] E. W. Dijkstra, *Solution of a problem in concurrent programming control*, Commun ACM Vol.8, Nº 5, September 1965, p 569.

[10] C. A. R. Hoare, *Communicating sequential processes*, Commun ACM, Vol.21, Nº 8, August 1978, pp 666–677.

[11] M. Tayli et al, *RT-DOS – a real-time distributed operating system*, in Proceedings of OUG-13, 18–20 September 1990, IOS.

[12] G. Barrett, *Two implementation of semaphores in occam*, OUG Newsletter, Nº 12, January 1990, pp 49–57.

[13] D. Homeister, *Semaphores at the transputer instruction level*, OUG Newsletter, Nº 13, July 1990, pp 46–49.

[14] S. W. Lau, F. C. M. Lau, *More on the implementation of semaphores in occam*, OUG Newsletter, Nº 13, July 1990, pp 50–54.

[15] J. L. Peterson, A. Silberschatz, *Operating systems concepts*, Addison Wesley, 1985, pp 327–344.

[16] J. Nicoud, A. M. Tyrell, *The transputer T414 instruction set*, IEEE Micro, June 1989, pp 60–75.

Murat Tayli F60C002@SAKSU00.BITNET
Mohamed Benmaiza F60C023@SAKSU00.BITNET
College of Computer and Information Sciences Tel: +966 1 4676580
King Saud University Fax: +966 1 4675630
P.O. Box 51178
Riyadh 11543
Saudi Arabia

AN ALTERNATIVE TO ALT

H. Tempelman and D. Millot
Institut National des Telecommunications and Universite Paris-Sud, France

The problem is well-known: there is that very fast transputer (network), with a fast programming language called occam. It all runs fine unless... an ALT is used. We propose an 'enqueuing mechanism' as an alternative to an ALT-multiplexer (as might be encountered in a routing layer, for instance). It provides both speed (especially when there are many data-sources) and flexibility (dynamically varying number of sources). We assume some knowledge of the transputer instruction set [17, 18].

The cost of ALT

A frequent use of ALT is for multiplexing purposes, like for example in an operating system where more applications can be loaded on one transputer. The different sources (applications) get access to the operating system through an ALT-construct:

```
WHILE TRUE
  ALT i = 0 FOR N
    input[i] ? x
      out ! x
```

This ALT block will be compiled to something like:

```
clk     again:
2         ALT
            -- enable channels
5/17      ALTWT
            -- disable channels
4         ALTEND
4         j again
```

The 'clk' column indicates the number of clock cycles required for each instruction. The 17 is for the situation when no guard was ready during enable, so descheduling occurs with ALTWT. Remark that this also implies another process has to be scheduled, which takes about 19 clock cycles! This means the 'body'-cost of an ALT is

- ▷ 15 clock cycles if a guard is ready (high load),
- ▷ 46 clock cycles if no guard was ready (low load).

For each input there is the need for an enable/disable pair:

```
ENABLE:
   2/3        ldl channel
   1          ldc 1   -- boolean true
   5/7        ENBC

DISABLE:
   2/3        ldl channel
   1          ldc 1   -- boolean true
   2          ldc process.offset
   8          DISC
```

which brings 22 clock cycles on average for each guard. (There are some variations depending on a guard being ready and the offset to *channel* and *process*.) For N channels (guards), the ALT-cost becomes:

- $15 + 22N$ under high load,
- $46 + 22N$ under low load.

The use of ALT has the following disadvantages:

1. The ALT-construct is proportionally slower when there are more input channels (guards).
2. The number of guards is fixed at compile time. Because of (1) one tends to make the number of inputs small, whereas for maximum flexibility the number of inputs should be high.
3. An ALT is not fair.

Reference [19] suggests to split a big ALT into a tree of smaller ones, which partly solves the problem of large ALTs – the time-per-packet in relation to the number of inputs being logarithmic instead of proportional. Reference [20] improves the ALT itself by not enabling channels (guards) when one has already been found to be ready. This is a bit better for high-load situations (and worse for low-load) and also deals with unfairness, using inline assembly code. Other (occam)-solutions are known to problem 3, but the total performance remains bad.

Enqueuing mechanism

In the following, we assume that several applications want to get access to a link. These applications (sources) are shielded by communication tasks (CT) which will try to get access to the link if the sources want to send something to it. Otherwise the CTs remain inactive. When the source sends something to the CT, the CT will try to input a *link.is.free* signal from the last CT using the link. In so doing, it may be descheduled on this communication, appending itself at the end of a communication chain. This chain is handled as a queue. The first CT in the queue is using the link. When it has finished, it will send a *link.is.free* signal to a *kick-channel*. The signal is received by the second CT in the queue (assuming there is a second one). This second one will therefore be put in the scheduling list of the processor, and can now use the link. You might sat that the second CT got a 'kick'.

The last CT in the queue will try to output to a non-existing CT, so nothing happens. If some time later a CT wants to use the link, it will do an input from this channel, and get kicked immediately (because the queue was empty). The trick is

that there is no central queue-handler, and that the on-chip communication-scheduler is used. Because it is dangerous to depend on applications to take actions correctly, they should be shielded by CTs, ready-to-use processes, part from the operating system, and linked to the application at load-time. These CTs can also do end-to-end flow-control, so that the network can never get blocked. They can also be responsible for routing: the CTs can directly enqueue for a link, and special CTs on every input link can send an incoming packet further to another link – by getting access to it using the enqueuing mechanism, in competition with the CTs of the applications – or pass the packet to a local process when the destination processor is reached.

The enqueuing mechanism can also be used to manage access to a (local) operating system shared service. Reference [21] shows it is also suitable for implementing broadcast services.

The main points:

- A process wanting to use the link appends at the end of the queue, otherwise it has no effect on the queue. In the queue it will not spoil any processing power. Therefore the processing cost per packet is independent of the number of inputs.
- Because it is a queue it is fair.
- There are no limitations on the number of inputs.

Implementation

After using the link, a CT kicks the next CT by writing some byte to a channel. If there is no next CT, the current CT will be descheduled. Now suppose the same application wants to use the link again. This is not possible for the CT will not accept the request before the 'kick' has been given and vice-versa. Therefore the kick should be given by a separate process. Now the CT can handle a request from the application and give a kick at the same time, so it can also kick itself if it is the only user of the link. The *kick.channel* of a CT is defined in its local workspace. In order to be able to input the kick from the previous CT, it has a pointer *last* pointing to the *kick.channel* of the previous CT. Additionally, every CT has a pointer to a global workspace (of the operating system), where at some location a '*back-pointer*' (`BPTR`) is stored, pointing to the last CT in the queue. This enables new CTs to append to the queue, and change the back-pointer to point at themselves. In order to start the enqueuing mechanism the first time, the operating system should set the `BPTR` to point to a location – a dummy channel – to which the system then outputs an initial kick.

This results in the implementation shown in figure 9 using the transputer instruction set. Because for the kick-task another process has to be (de)scheduled, 19 clock cycles have to be added bringing the total to 109 clock cycles overhead per packet for the enqueuing mechanism. This means this mechanism is better than the repeated `ALT`-construct for values of N greater than

$(109 - 15)/22 = 4{\cdot}2$ (more than 4 channels for high load),

$(109 - 46)/22 = 2{\cdot}8$ (more than 2 channels for low load).

Note that this implementation does not allow a mixture of low and high priority communication tasks. Low priority CTs cannot interfere with each other, because the transputer can only deschedule on the instructions `J` and `LEND`, and of course on channel communication and process management.

```
        mint; stl kick.channel          -- initialise kick.channel
    again:
        -- wait for a request from the application
        -- do flow control
 4      ldl  global_wptr; ldnl BPTR     -- get back-pointer which points
 1      stl  last                       -- to the last CT in the queue.
 1      ldlp kick.channel
 4      ldl  global_wptr; stnl BPTR     -- modify back-pointer: now this
                                        -- is the last CT in the queue
 ---------------------------------------- wait for kick:
 1      ldlp dummy.buffer               -- pointer to store kick
 2      ldl  last                       -- input from previous CT
 1      ldc  1                          -- input 1 byte
21      in                              -- if the previous CT is not
                                        -- ready, this CT deschedules
                                        -- now.
        -- Now scheduled again (kick received), so use the link.
        -- After using the link start up a separate task kicking
        -- the next CT in the queue.
 1      ldc  kick-base                  -- offset to kick.task
 2      ldlp -offset                    -- workspace for kick.task
12      startp
    base:
 4      j again                         -- handle next request
 ---------------------------------------- kick process:
    kick:
        ldlp kick.channel
 1      ldc  any
23      outbyte                         -- kick next CT
11      stopp                           -- and abort this task
```

Figure 9: implementation of the enqueing mechanism

Problems

1. The kick is generated by a separate task. This task will not execute immediately, but will be appended at the end of the schedule list. Then it kicks the next CT, which will also be appended to the end of the schedule list. This means that between two successive link accesses, the time which elapses is twice the round-robin time of the schedule list. In this 'dead-time' the link or service remains unused. If n processes are already waiting on the schedule list, then the time to wait becomes:

$$\textit{dead-time} = 2 \times n \times (2 \times 5120/(5\,\text{MHz})) = 4n\,\text{ms}$$

Remarks:

- ▷ This assumes that each process uses its full time-slice period, i.e. that it does not get blocked on communication or timer input – otherwise this figure is improved.
- ▷ When an ALT deschedules (under low load), there is also a dead time (once

the round-robin time) of $2n$ ms. For high load the `ALT` is better with respect to the dead-time.

- ▷ If these processes are run under high priority however, n will generally be small. The enqueuing mechanism will be optimised below, resulting also in improved dead-time.

2. After the kick-task has given the kick it is descheduled immediately because of the `STOPP` instruction, so we lose 19 clock cycles. This will also be dealt with by the optimisation.
3. There are some problems if the link/service enqueued for is not always the same (for example if a CT also does routing and uses different links; this is not however possible at all using `ALT`): suppose CT-A accesses link 1 and is the only one. At the end it will startup a kick-task saying link 1 is free. Suppose CT-A now uses link 2, and at the end it starts up the same kick-task saying link 2 is free, both times using the same (local) kick-channel address. Now a CT-B uses link 2, it will be kicked by the kicktask of CT-A and reset it to `MinInt` (the `IN` is performed). This means a CT-C will find link 1 blocked for ever. A comparable problem could occur if the task which is supposed to give a kick is aborted (because the application it belongs to was aborted by the user), leaving its kick-channel address in some undetermined state. For the same reason a task may not migrate when it is queuing. These problems are addressed in the next section.

Optimisations and improvements

From the first implementation on four optimisations were done – see reference [21] for details – according to the following considerations:

- ▷ The fact that the queue is empty can be saved in a global flag. In that case a new CT can continue immediately if the queue is empty. It also means that problem (3) is solved, since the 'queue-empty' information is no longer stored in the last CT that used the link. The queue is empty if the back-pointer holds the value `MinInt`. This is possible since it does not need to store the last CT any longer if the queue is empty.
- ▷ The effect of sending a byte on the kick-channel is just to schedule the next CT, which has stored its PID (**p**rocess **id**entifier) in this kick-channel, on the assumption that there *is* a next CT! So if the current CT notices there is another CT waiting (kick-channel holds a non-`MinInt` value), it can also schedule this process using a `STARTP` instruction. If there is no CT waiting, it sets the global BPTR to `MinInt` – see previous consideration.
- ▷ According to the first two considerations, it is no longer necessary to start up a separate kickprocess, which means problem (1) is solved, and problem (2) is partly solved. There is still a dead time however: the next CT is still appended at the end of the queue, so the dead time becomes $2n$ ms, where n is the number of processes in the queue.
- ▷ The `IN` instruction has the only effect that the PID of the process is saved in the kick-channel address of the previous CT! The byte transfer is not used at all, so this code-part can be optimised, and nothing has to be done when a CT arrives on an empty queue (when BPTR is `MinInt`).

The resulting assembly code for the enqueuing mechanism is shown in figure 10. Note that the initialisation of the *kick.channel* is in the loop now. Note that there is no extra (de-)scheduling of other processes now since there is no separate kick-task, so this results in the following processor load:

low load: 41 clock cycles ($46 + 22N$ for an `ALT`)

high load: 62 clock cycles ($15 + 22N$ for an `ALT`)

This means that for low load the enqueuing mechanism is even better than just the *overhead* of the `ALT` construct (`ALT-ALTWT-ALTEND`)! For high load the calculation is:

$N = (62 - 15)/22 = 2{\cdot}1$ (more than two channels for high load)

In practical implementations however a few clock cycles have to be added. In the listing it is assumed that the BPTR can be found at a fixed place; in reality it will be in some table. But the same kind of overhead also applies for practical `ALT`-implementations.

Remark: the enqueuing mechanism is safe (if all the CTs have the same priority) if two CTs try to enqueue at the same time. The reason lies in the fact that the transputer only switches to another process on certain instructions. These are the instructions where this can be expected (channel communication, timer wait, process descheduling), and the instructions `j` and `lend`. The first part of the listing in figure 10 (the enqueue part), contains only load and store instructions, and can therefore be regarded as an atomic instruction. It can be interrupted by a high-pri process however, and therefore a mixture of low- and high-pri CTs is not allowed.

Future transputers

The processor overhead could reduce dramatically if some parts of the listing were supplied in special instructions, thus being coded in microcode. Two extra instructions could be supplied:

enqueue parameters: a pointer to BPTR, and a pointer to the kick-channel of this CT. Ensures the the process will enqueue. If the BPTR was `MinInt`, it will continue directly, otherwise it deschedules itself saving its PID.

kick parameters: a pointer to BPTR, and a pointer to the kick-channel of this CT. If the kick-channel is not `MinInt`, it is the PID of the next CT, which it will append to the end of the schedule list, otherwise it will initialise BPTR to `MinInt`.

The `j again` should not be part of the kick instruction, but be programmed just after the kick instruction. This means that the first occurrence of `j again` becomes a jump in the microcode to the end of the microcode for the kick instruction.

There is a problem deciding which scheduling list it should use: the low-pri schedule list may be too slow, the high-pri schedule list should be reserved for urgent tasks. Here a third kind of schedule list could be introduced, with medium priority, which is reserved for the enqueuing mechanism only. (In current types of transputer it would hardly be possible to use a fine-grain priority – say 255 priority levels – because it would require 255 lists.) This option would be a welcome extension and meet many current demands for finer priority granularity. Because it will always use the medium-pri list for this, the PID does not have to contain the priority, so that the same principle can be used also in 16-bit transputers (where there is only one bit available for storing the priority).

```
     again:
 2     mint; stl kick.channel        -- initialise kick.channel
       -- wait for a request from the application
       -- do flow control
 4     ldl  global_wptr; ldnl BPTR  -- get back-pointer which points
 1     stl  last                    -- to the last CT in the queue
 1     ldlp kick.channel
 4     ldl  global_wptr; stnl BPTR  -- modify back-pointer: now this
                                    -- is the last CT in the queue.
 4     ldl  last; mint; diff        -- was BPTR MinInt ??
 6/8   cj   link_is_free            -- if so, go on and use the link
 ------------------------------------ wait for kick:
     -- (this part is done under high load only)
 3     ldlp 0; ldpri; or            -- get own PID
 2     ldl  last                    -- store in kick.channel ..
 2     stnl 0                       -- .. of previous CT
11     stopp                        -- deschedule now.
     link_is_free:
       -- Now scheduled again (kick received), so use the link.
 ------------------------------------ kick next CT:
 1     ldl  kick.channel
 2     mint diff                    -- is there a next CT ??
 2/4   cj   next_to_come            -- if not, set BPTR to MinInt
 ------------------------------------ next CT already waiting:
 2     ldl  kick.channel            -- get PID of next CT ..
10     runp                         -- .. and start it
 4     j again
 ------------------------------------ was last CT for now:
     next_to_come:
 1     mint                         -- make ..
 2     ldl  global_wptr             -- .. BPTR ..
 2     stnl BPTR                    -- .. MinInt
 4     j again
```

Figure 10: optimized implementation of the enqueuing mechanism

Conclusions

The enqueuing mechanism is a simple idea which results in a very nice performance if some optimisations are done.

Characteristics are:

- ▷ More efficient than any kind of (optimised) `ALT` with respect to the processor load.
- ▷ Processor load is independent of the number of possible input channels, while the `ALT` is linear dependent on this.
- ▷ Unlimited number of input channels, and the number can vary dynamically whereas the `ALT` has a fixed number.
- ▷ Absolutely fair.
- ▷ The same source code can be used for getting access to different links/services (such as routing).
- ▷ Smaller code than `ALT` because `ALT` needs separate code for each guard.

Disadvantages:

- ▷ Cannot be described in occam, but perhaps some `GUY` inline code can solve this problem.
- ▷ Under high load the link/service cannot be used continuously: when the next CT is kicked it is appended at the end of the scheduling list. The link/service remains unused during the time it takes for the CT to get scheduled. This can be solved making the CT a high priority process; the high-pri schedule list is shorter in general than the low-pri list. (The idea suggested for a future transputer could deal with this point.)

Remark: In order to test the enqueuing mechanism (and other assembly subjects), a cross-assembler called TRASM (**tra**nsputer **as**sembler) has been developed, including a debugger (TRDEB) and a loader (TRLOAD). Simple programs have tested the mechanism on all aspects and found it to be working correctly. More advanced tests are necessary however especially where performance is concerned.

This work was carried out by Hennie Tempelman (a student electrical engineering at the University of Twente, the Netherlands) during a six month period at Institut National des Telecommunications, with the assistance of Daniel Millot, teacher/researcher in the DIT Department. It is part of his master thesis report [21] which deals with the considerations for an operating system for transputer networks, using assembly language and sockets for occam SC-modules [22].

References

[17] Inmos Ltd, *The transputer instruction set – a compiler writers' guide.*

[18] D. A. P. Mitchell, J. A. Thompson, G. A. Manson, G. R. Brookes, *Inside the transputer*, Blackwell Scientific Publications, ISBN 0-632-01689-2, 1990.

[19] K. M. Shea, F. C. M. Lau, *On the performance of* `ALT` *in occam*, North American Transputer Users Group: NATUG 3, April 1990, ISBN 90-5199-030-8.

[20] S. W. Lau, F. C. M. Lau, *An efficient and flexible implementation of* `ALT`, North American Transputer Users Group: NATUG 3, April 1990, ISBN 90-5199-030-8.

[21] H. Tempelman, *Operating system kernel for transputer networks having an unknown topology*, master thesis report, University of Twente, Department of Electrical Engineering (BSC), B.P. 217, 7500AE Enschede, The Netherlands; and INT, Department of Computer Science (DIT), 9 Rue Charles Fourier, 91011 Evry Cedex, France.
[22] Inmos Ltd, *Transputer Development System (IMS D700D).*

H. Tempelman
Beukenlaantje 8
7475 SK Markelo
The Netherlands
Tel: +31-5476-1837

D. Millot
INT–DIT
9, Rue Charles Fourier
91011 Evry
France
millot@frint51.bitnet

and
LRI
Université Paris-Sud
91405 Orsay
France
millot@lri.lri.fr

SOME ALGEBRA

and some of the proposed extensions to occam

Geraint Jones, Programming Research Group, Oxford

In a paper at the OUG technical meeting in York [23] there were a couple of proposals for new ways of writing a couple of common idioms. In order that one could write the usual sort of service-type of process that hangs off the side of most user programs, and make it look like a 'declaration' of the service, Geoff Barrett proposes that

```
declarations
INITIAL
  initialisation.process
:
RESOURCE
  resource.management.process
:
FINAL
  finalisation.code
:
user.process
```

should be made to mean the same as

```
declarations
SEQ
  initialisation.process
  PAR
    resource.management.process
    SEQ
      user.process
      finalisation.code
```

the idea being that the *initialisation.process* gets to set the values of any variables in the declarations before they are used, and that the *user.process* gets to talk to the *resource.management.process* until it (the *user.process*) terminates, and then the finalisation.code gets a chance to (tell the *resource.management.process* to) stomp all over the resource before telling the *resource.management.process* to go away. You

keep wanting to write such things, and the proposed style makes it altogether more obvious which part of the code is the *user.process*, and which parts are 'service code'.

What is really neat about this is that as you can see by matching the two bits of code above, the three new 'declarations' can be implemented as independent translations, and this is what he intends:

```
INITIAL                          SEQ
  i.p          is the same as      i.p
:                                  u.p
u.p
```

and

```
RESOURCE                         PAR
  r.p          is the same as      r.p
:                                  u.p
u.p
```

and

```
  FINAL                          SEQ
    f.p        is the same as      u.p
  :                                f.p
  u.p
```

Where have you seen this sort of thing before? Well, if you have seen it before, you probably call it sectioning. The sections of a (binary) operator are the functions you get by fixing one of the arguments. The left section $(a+)$ of the $+$-operator is the function that takes a number, say b, and adds a to it, giving you $a + b$. That is, $(a+)\,b = a + b$, read as 'a-plus applied to b gives you a-plus-b'.

The right-section $(+a)$ is similar except that $(+a)\,b = b + a$. Of course it is just the same as the other section since plus is commutative, but for a non-commutative operator you get different functions, for example $(2/)$ gives you twice the reciprocal of a number, and $(/2)$ gives you half the a number.

Now look at the occam: if we write $P;Q$ for the `SEQ` of P and Q, and $P\|Q$ for the `PAR` of P and Q, then

```
INITIAL
  P          is the same as   (P;)
:
```

and

```
RESOURCE
  P          is the same as   (P||)
:
```

and

```
  FINAL
    P        is the same as   (;P)
  :
```

and since the || operator is commutative, $(P\|) = (\|P)$ so these are all the sections of the ; and || operators.

You can see from the definitions that $(P;)(Q;)R = P;(Q;R)$ which we usually write as $P;Q;R$ because the ; operator is associative, and moreover since $P;$`SKIP` $= P$ you have that $P;Q;R = (P;)(Q;)(R;)$`SKIP`. Indeed in general you can write any

expression in which all the operators are ; as a repeated application of left-sections of ; to SKIP. In other words, we can now dispense with SEQ from the language altogether because

```
INITIAL                          SEQ
  P                                P
:                                  Q
INITIAL          is the same as    R
  Q
:
INITIAL
  R
:
SKIP
```

and similarly we can dispense with PAR because

```
RESOURCE                         PAR
  P                                P
:                                  Q
RESOURCE         is the same as    R
  Q
:
RESOURCE
  R
:
SKIP
```

I hope that you can see what comes next: if you stack up a number of FINAL declarations, you also get a process made up from the bodies of the FINALs by SEQ, but the order of the processes is reversed:

```
FINAL                            SEQ
  P                                Q
:                                  R
FINAL            is the same as    P
  Q
:
FINAL
  R
:
SKIP
```

so you can now get away from this annoying business of starting at the beginning of a program and executing it step by step until you get to the end: at last a way of writing occam programs that start at the bottom and work back to the top.

I don't remember whether Geoff was suggesting replicated INITIAL, RESOURCE and FINAL declarations, but if we allow for the moment things like

```
INITIAL i = b FOR c
  P(i)
:
```

with the (um, er) obvious meaning

```
INITIAL
  P(b)
:
INITIAL
  P(b+1)
:
...
INITIAL
  P(b+c-1)
:
```

then we can dispense with replicated SEQ and PAR as well, because

```
INITIAL i = b FOR c
  P(i)
:
SKIP
```

means

```
SEQ i = b FOR c
  P(i)
```

and similarly for RESOURCE and PAR. What is altogether more fun is that

```
FINAL i = b FOR c
  P(i)
:
SKIP
```

which is that thing everyone has always wanted at some time or other – a replicated SEQ in which the index counts *down* to a given value in steps of minus one – because it is the same as

```
SEQ i = 1-c FOR c
  P(b-i)
:
```

What next? Well, something I have often wanted in occam is a commutative sequential operator: the $\times$ for which $P \times Q$ has the same meaning as

```
ALT
  SKIP
    SEQ
      P
      Q
  SKIP
    SEQ
      Q
      P
```

It has the effect of executing both of P and Q, and only one of them at once, so it is just the thing you need for a pair of exclusive accesses to some shared object in those cases where you do not need – and so do not want – to specify the order of the accesses. In CSP terms, it is the non-deterministic choice between $P;Q$ and $Q;P$.

This is an associative operator so you can imagine a construct with many branches, like SEQ and PAR, but again you only need the binary operation $P \times Q$ where $P \times Q \times R = (P \times Q) \times R = P \times (Q \times R)$ and so on. Now what are the sections of

this operator? Well, they are clearly both the same, because it is commutative, so $(P\times) = (\times P)$, and I suggest we write it

```
SOMETIME
  P
:
```

because what this would mean is that P should be executed, but it can either be executed now, or the machine can put it on one side and execute it after the end of the scope of the 'declaration'. Perhaps something like

```
SOMETIME
  get.work.done
:
edit.newsletter
```

Does anyone have good names for the sections of the ALT and IF operations? There are three of these, perhaps the left section of IF should be PREEMPT.IF, because

```
PREEMPT.IF                          IF
  e                                   e
    P          is the same as           P
:                                     IF
IF                                      ...
  ...
```

and it is the same as the conditional in its body unless e is true; in that case the execution of the conditional is pre-empted by the declaration, and the whole thing behaves like P. The right-section is a sort of if-all-else-fails-then-if, because

```
IF.ALL.ELSE.FAILS.THEN.IF                    IF
  e                                            IF
    P                                            ...
:              is the same as                  e
IF                                                P
  ...
```

and it executes P if and only if both e is true and also the conditional in its scope would otherwise have failed.

Because ALT is commutative, it has only one section (like PAR) and it means something like, 'Oh, and by the way I won't say it in the ALT, but if you happen to hear from this channel then you could do this instead'. There is – as it happens – just such a section lurking in the explanation of SERVER in terms of INITIAL, RESOURCE and FINAL in Geoff's paper[23]. He suggests something like

```
SERVER
  g
    P
  h
    Q
  ...
:
```

for

```
PROTOCOL SIGNAL
  CASE
    signal
:
BOOL more :
CHAN OF SIGNAL user.terminated :
INITIAL
  more := TRUE
:
RESOURCE
  WHILE more
    ALT
      g
        P
      h
        Q
      ...
      user.terminated ? CASE signal
        more := FALSE
:
FINAL
  user.terminated ! signal
:
```

Of course, equipped with the sections of `IF` and `ALT`, and with a `STOP` to get started from you could easily dispose of `IF` and `ALT` from the language.

Oh, I do hope nobody has been taking this article entirely seriously. What am I to do when the *Newsletters* are published three months either side of the first of April? This may be my very last chance to slip something silly past the editor when he is otherwise occupied.

I would not want you, either, to do less than take seriously Geoff's paper. All I wanted to do is to have a bit of fun at the expense of my own style of exposition, perhaps cutting myself on the sharp edge of Occam's Razor.

To be briefly serious, I think these constructs are really rather neat ways of exposing the structure of resources and their users within a program. As I hope I have explained above they have the great virtue that you do not need to add anything new to your concept of what an occam process is.

The only other serious thing I have to add is that right-sections of ; like

```
FINAL
  P
:
```

are called 'continuations', or sometimes 'process continuations', by the sorts of people who do denotational semantics, and now I think about it, I cannot think why they do not seem to have names for the other sections that I have discussed here.

References

[23] Geoff Barrett, *The development of occam: types, sharing and modules*, in Hussein Zedan (ed.), *Real-time systems with transputers*, OUG–13, IOS, 1990.

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

geraint.jones@uk.ac.oxford.prg
Tel: +44 865 273851
Fax: +44 865 273839

MAXIMISING THE PERFORMANCE OF DATA THROUGHPUT IN A SATURATED ROUTING SYSTEM

Piers A. Shallow

Introduction

This article looks at three fundamental areas concerning the optimization of data throughput between processes sharing common memory and describes how the efficiency of a typical buffering system, buffer resource manager and the `ALT` command can be improved. The performance maximisation is targeted at a saturated routing system because it is in a condition that requires the maximum communications bandwidth and the minimum possible implementation overheads.

The implications of modifying the occam model to allow the use of shared variables and memory between parallel processes is not discussed this paper, however the time saved transferring an `INT` index pointing to an array of elements instead of the elements themselves is obvious and forms the fundamental basis of the techniques suggested.

Buffering of data packets

In striving to achieve true parallelism on both input and output communication channels, D. May [24] suggested a very simple solution, in both concept and implementation, by use of I/O processes in parallel (figure 12, figure 11 ex. 1). Unfortunately the solution suffers from an enormous degradation in the communication bandwidth as well as latency in data transfer. Consequently the solution commonly used is one that incorporates the use of a 'request' mechanism (figure 13, figure 11 ex. 2) between two parallel processes. The main process carries out the buffer administration, accepts blocks of data from an input channel, stores them in the buffer and transfers the blocks of data to the second process whenever a request for data is made. The second process only requests data from the main process when it is able to send further data out along its output channel.

As it stands, the 'request' solution only allows the main process either to input data, or accept a request, or output data at any instant, losing the true input and output parallelism desired. The only way to improve the performance is to have the pool of buffers accessible by both processes and only pass between them the index of the data block to be transferred (figure 11 ex. 3), reducing the output transfer

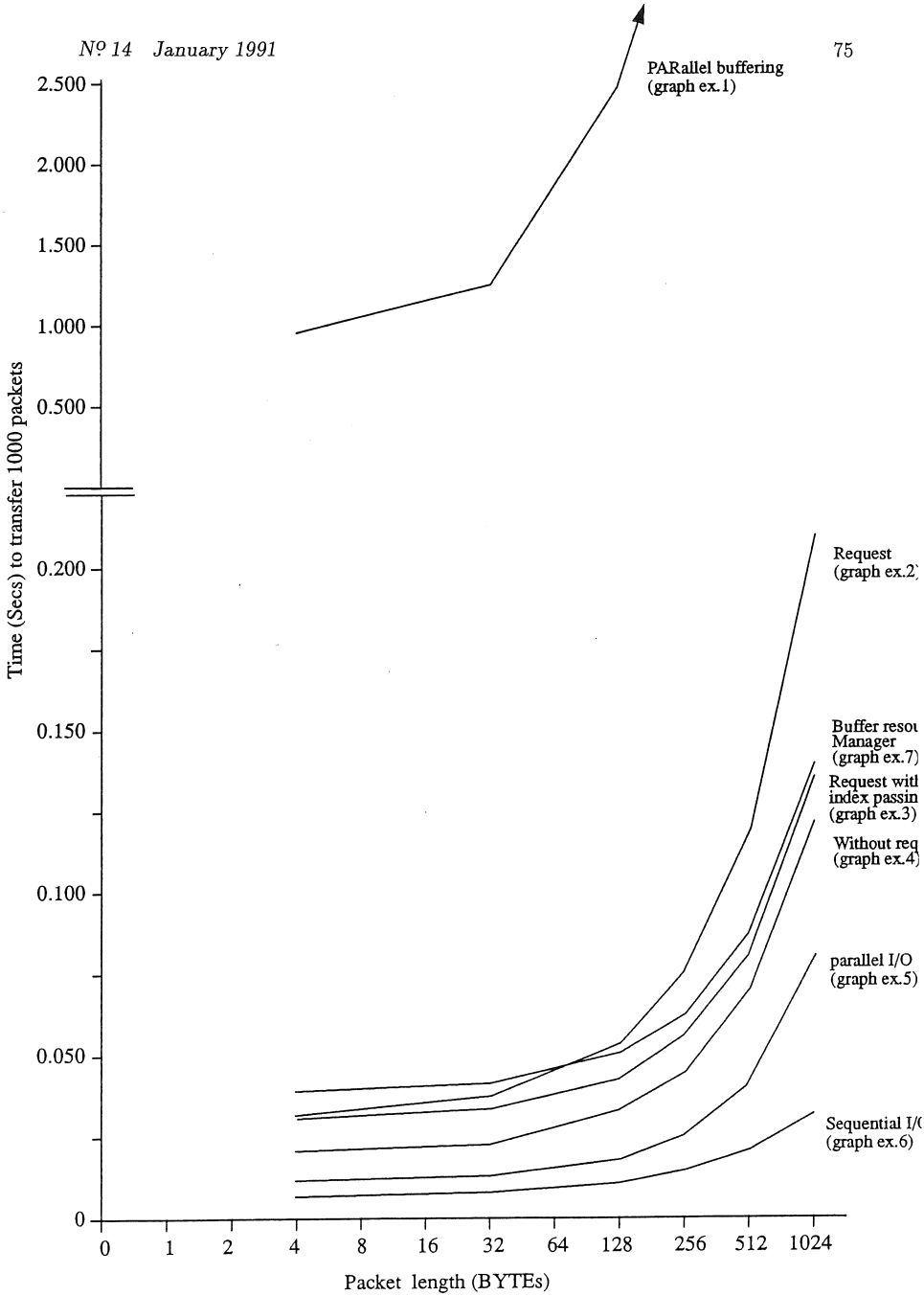


Figure 11: time taken to transfer data using different types of buffers of size sixteen

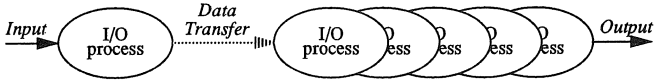


Figure 12: PARallel buffering system

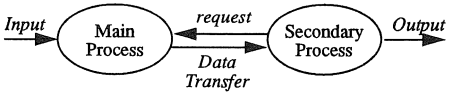


Figure 13: 'request' buffering system

time to that of an INT. The use of the *last.tail* pointer prevents both the processes accessing the *tail* buffer at the same time.

Time is still being wasted with the 'request' mechanism and the need continuously to request the next block of data can be removed by implementing a synchronised communication between the two processes whenever the buffer becomes full or empty (program in figure refprogram, figure 11 ex. 4). The way in which this solution works is somewhat controversial as it is not within the rules of occam 2 [25]. As well as the buffers being accessible by both processes, the head and tail pointers used to ensure that the two process do not access the same buffer at the same time, are themselves accessible by both of the processes. The head pointer is used in such a way that it is 'read only' by the second process and is written to by the main process. The tail pointer is set up to be 'read only' by the main process and written to by the second process.

These performance characteristics and those of a simple parallel I/O routine (figure 11 ex. 5) and a sequential I/O routine (figure 11 ex. 6) can be seen in figure 11, which displays the times taken to transfer 1000 blocks of data continuously through the different types of buffering systems using internal communication channels. The buffers were capable of storing 16 blocks of data.

Buffer resource manager

The buffer resource manager (figure 15, figure 11 ex. 7) is a process that is in control of the pool of shared buffers accessible by all the processes associated with the movement of data. Every time a new resource is required by a process, a request for a resource is made to the buffer resource manager which allocates an available resource by returning a pointer (index) to the requesting process. The index of the resource is then transferred to another process and when that process has finished with the resource it returns the freed resource back to the buffer resource manager which adds it to the pool of free buffers. The great advantage of this type of mechanism is that it is not limited in the number of requesting and discarding processes (figure 16) it can support and that data is efficiently transferred from any requesting process to

```
PROC buffer (CHAN OF p.data.type  channel.in, channel.out)
  -- internal channels
  CHAN OF ANY empty.strobe:
  CHAN OF ANY full.strobe:
  -- global vars
  [buffer.size][data.length]data.type    buffer:
  INT head:
  INT tail:
  SEQ
    head := 0
    tail := 0
    PAR
      -- input process
      INT any:
      BOOL empty:
      SEQ
        WHILE TRUE
          SEQ
            channel.in ? buffer [head]
            IF
              head = tail       -- was empty
                SEQ
                  head := (head PLUS 1) REM buffer.size
                  empty.strobe ? any
              TRUE              -- was not empty
                SEQ
                  head := (head PLUS 1) REM buffer.size
                  IF
                    head = tail -- now full
                      full.strobe ? any
                    TRUE        -- not full
                      SKIP
      -- output process
      SEQ
        empty.strobe ! 0 -- any
        WHILE TRUE
          SEQ
            channel.out ! buffer [tail]
            IF
              head = tail       -- became full while waiting to output
                SEQ
                  tail := (tail PLUS 1) REM buffer.size
                  full.strobe ! 0 -- any
              TRUE              -- not full
                SEQ
                  tail := (tail PLUS 1) REM buffer.size
                  IF
                    head = tail -- now empty
                      empty.strobe ! 0 -- any
                    TRUE        -- not empty
                      SKIP
:
```

Figure 14: parallel I/O buffer without the need for a 'request'

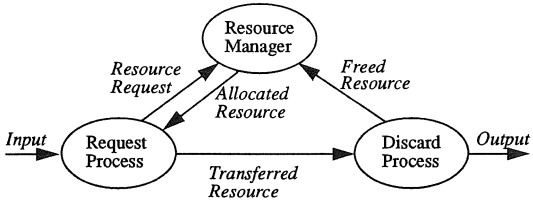


Figure 15: buffer resource manager

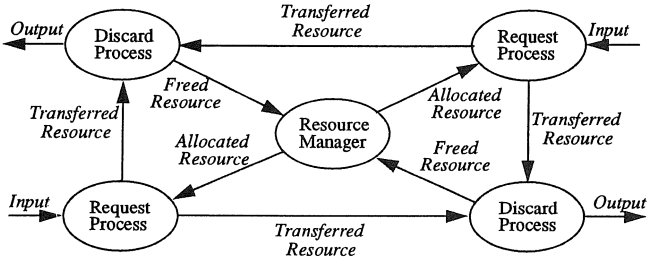


Figure 16: number of requesting and discarding processes

any discarding process by only passing the index.

Although the buffer resource manager is in effect implementing strict control over which process has the right of access to a particular buffer at any instant of time, there is a penalty in its implementation of a degradation in communication performance (figure 11 ex. 7) as the process rapidly becomes the bottleneck in any non-small routing systems. This is because all the requests and discarding processes have to communicate with the resource manager which can only service each process in sequence.

This problem is solved by the removal of the resource manager altogether whilst maintaining the strict control over the buffer allocation to its users. The way in which this can be achieved is based upon the principle that if the discarding process is ready to accept another buffer, it must have just finished with one, and that if the request process is ready to transfer a buffer index to the discarding process then the process must require another buffer to take its place. So, instead of the discarding process passing the freed index to the resource manager and the request process requesting another resource from the resource manager, the discarding process simply passes the freed index straight to the request process (figure 17), which removes the need for the resource manager altogether. This model can now be expanded to cater for applications which have more than one destination process (discarding process), by allowing the input process (requesting process) to only accept the freed index from the discarding process to which it has just transferred a resource (figure 18). In this situation, the returning freed index can act as the 'request' in the buffering system allowing buffering to occur within the input and output processes of the routing system. Instead of each process containing buffer areas to store the blocks of data,

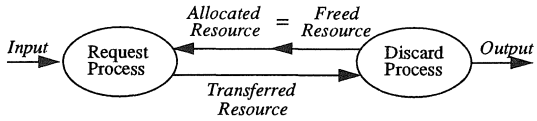


Figure 17: resource manager removed

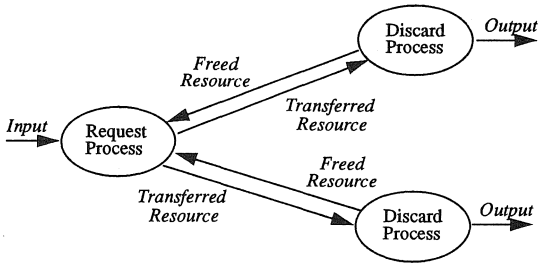


Figure 18: multiple destination processes

it has an array of buffered indices pointing to the buffer locations within the pool of buffers. The model may be extended further to incorporate two or more input processes (figure 19). Again the request process only accepts the freed resource from the discarding process to which it has just transferred a resource. The control of the output route is determined by the request process.

> Intriguing problem – without the use of an ALT set up to use output communications on internal channels, what would the code have to look like if instead of the data being directed to a particular discarding process, as in the case described above, data could be accepted by either discarding process depending on which one happened to ready to accept further data?

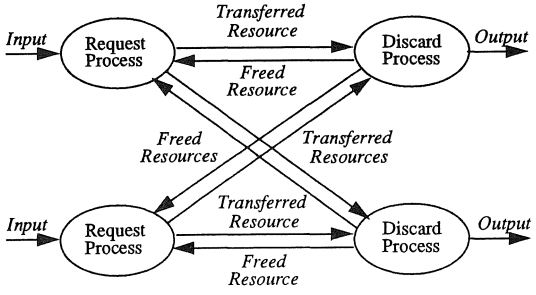


Figure 19: multiple request and discard processes

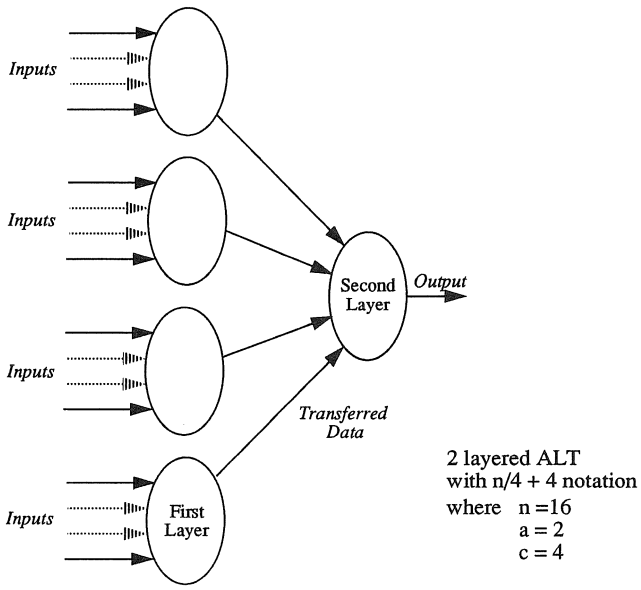


Figure 20: layer distributed ALT

Distributed ALTs

The problem with an ALT in a saturated routing system is that at least one channel of an ALT is always ready and as a consequence a lot of time is being wasted enabling (enbc) and disabling (disc) channels declared after the first ready channel has been ascertained.

There are number of possible solutions that could be implemented in assembler, one of which could include a pre-ALT process which checks the channels of the ALT to see if any of them are ready. If one of them is ready, then the program jumps straight to that channel's associated block of code, otherwise the ALT is set up and implemented in the normal way. Unfortunately this solution is not immediately practical and an alternative solution can be written in occam by reducing the number of channels within a single ALT. This is simply achieved by distributing the channels over a number of ALTs, connected together by further ALTs (figure 20). The maximum reduction in processor cycles [26] is obtained when $n = a^c$ where a is the number of layers of ALT processes, c is the number of channels supported by each ALT and n is the total number of input channels.

The gains obtained can be seen in figure 21, which shows the time taken in processor cycles to transfer an INT against a varying number of total input channels for different ALT configurations. The notation of the configuration of each distributive ALT is given as $A + B + C + \cdots + N$ where the letters represent the number of input channels for each ALT at each layer, i.e. $n/8 + 4 + 2$ is a three layered distributed

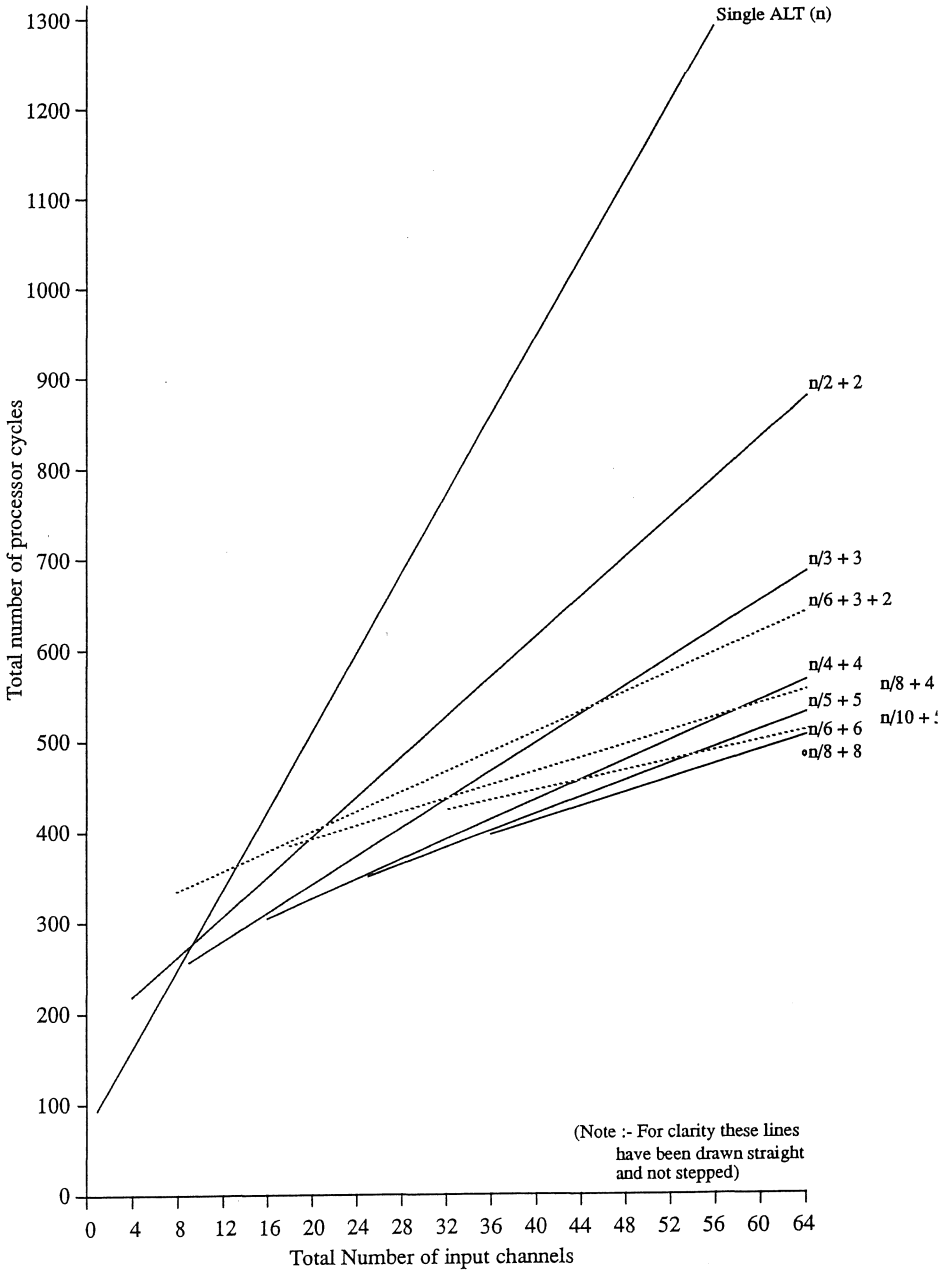


Figure 21: distributed ALT

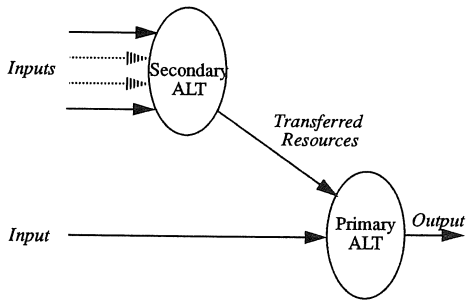


Figure 22: PRI ALT

ALT with 8 ALTs in the first layer each supporting $n/8$ input channels, 2 ALTs in the second layer each supporting 4 input channels and 1 ALT in the third layer supporting 2 input channels.

The real gains of splitting the ALT lie with the PRI ALT where the high priority channel is separated from the remaining channels (figure 22) enabling the PRI ALT to only have a set up overhead of two input channels. Gains are also obtained in situations where there are a number of channels which are very seldom used within the ALT and they can be separated into their own secondary ALT prior to being connected to the primary ALT.

The savings discussed are being obtained because the indices of the buffer locations are being transferred through the network rather than the actual blocks of data. The indices can be either transferred back through the ALTs, which would increase the implementation overheads by having to have a second channel setup and INT transfer (figure 23) or the ALT in the first layer could append a channel reference number to the index which the ALT in the last layer uses to return the freed resource. Again the buffering of indices within processes is still possible.

Summary

The implementation of buffering mechanisms within the I/O processes when using internal channels degrades the communication bandwidth.

The use of a global pool of buffers and passing of buffer indices are fundamental to obtaining high data throughput in routing systems.

Buffer resource managers are the bottlenecks within multi I/O routing systems.

The buffer resource manager only provides buffing for free indices and not for 'used' indices. The buffering of 'used' indices must take place in another process.

The full flexibility of the buffer resource manager can be achieved by carefully controlling which of the processes in the routing system implement the buffering of the indices.

The size of the pool of buffers must equal the total index buffering space plus the number of declared indices used within the other processes throughout the routing system.

Distributing the ALT over a number of ALTs is worth adopting as the reduction

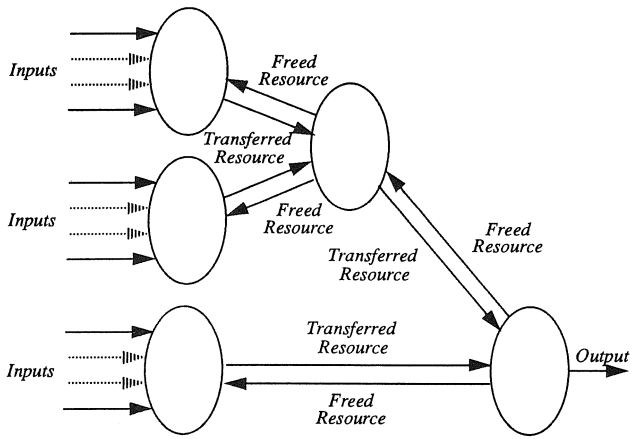


Figure 23: ALT returning the freed resource index

in processor cycles in quite considerable. Enormous gains are obtained by simply splitting the total number of inputs over x ALTs and having a further single ALT with x inputs.

Questions

Although the control of access to variables is implemented in software, should the programmer be conforming to the limitations of the occam compiler or should the occam compiler cater for 'read only' variables?

With the inherent nominal buffering present in the routing system described above, are the overheads in implementing additional buffering within processes justified in light of the reduction in communication bandwidth they cause, let alone the possible side effects such as starvation in one channel whilst another has a data in its buffer, or loss of end to end communication synchronisation which results from buffering?

Should the implementation of the ALT in assembler be altered to improve the performance of channel communication as the communication speed appears to be the transputer's handicap? Does the increase in processor cycles taken in setting up the ALT suggested in assembler really matter when there are no channels ready, as it does not affect the communication bandwidth of the channel?

Should there be an output ALT on internal channels only and not external channels (those using the links)? (Yes I appreciate the query:- when is an internal channel not a external channel? And the answer to the question is:- when the pool of buffers is in local memory and not distributed across a number of processors.)

References

[24] D. Partain and D. May, *A tutorial introduction to occam programming*, Inmos 1987.

[25] Inmos Ltd, *Occam 2 reference manual*, Prentice Hall.

[26] Inmos Ltd, *Transputer instruction set – a compiler writer's guide* Prentice Hall.

P. A. Shallow
c/o Danescourt
Pond Road
Woking
Surrey GU22 0JT
United Kingdom

EXPERIENCES OF A SOFTWARE BASED SCHEDULER

James Kidd, Altis

Background

Previously, when this design work was initiated, the author was employed as Senior Software Engineer in a company involved in the manufacture of Ethernet based equipment. Many of these products contained Zilog Z80 microprocessors; the software, for which was written in assembly language by a small team of programmers. To enhance the software development environment, several tools were created that were based on principles similar to those of occam. This article contrasts this, software based, scheduler with the transputer and occam and describes some of the more useful observations.

Design goal

Since our products had to conform to the *Open Systems Interconnect* (OSI) layered model of communications, much of the software was also written in layers. Consequently there was a desire to reuse, within several different products, the same blocks of tried and tested code. Perhaps typically for embedded system manufacturers, the software design and development procedures had been somewhat informal. Consequently there was also a desire to define more rigorously the interface between modules created by different programmers.

Initial solution

A library of routines was written that formed a multi-tasking executive. The code was very simple and fast. The kernel consisted of the following subroutines, global variable and background task:

- ▷ `SPAWN` – Creates new tasks
- ▷ `SNOOZE` – Invokes a task switch
- ▷ `INPUT/OUTPUT` – Inter-task communication (ie message passing)

- ▷ SLEEP/WAKE – Message-less synchronisation (ie counting semaphores)
- ▷ LOCK/UNLOCK – Critical region protection
- ▷ TIME – Global variable incremented every 1 ms
- ▷ WATCHDOG – Background task to monitor the scheduler

All these constructs operate entirely at runtime and under the complete control of the programmer. Points at which a task switch can occur are explicitly declared. The system supports any number of tasks at a single background priority level plus conventional vectored interrupts.

Aspects relevant to the transputer and occam

Occam had not 'hit the streets' when this scheduler was conceived, so our intention had not been to produce simply a poor-man's occam replacement. Rather, it was an attempt to solve a similar problem; albeit with a somewhat more hands-on approach. There are, therefore, some differences which I do not care to highlight. Instead, I will concentrate here on some of the effects that we found and some of the differences that could possibly be transferred to the occam and transputer world. In fact, information about occam 1 was released while this project was being completed. This caused its name to be changed to *Razor*; standing for *Ruddy Awful Z80 Occam Replacement.*

Although Razor's basic message passing system is similar to the occam's, there is a major difference that could be exploited to create a truly fair ALT construct. Razor's INPUT and OUTPUT routines pass messages between tasks through channels by copying data. This is much like occam, however, Razor's channels are comprised of three words of memory. The first word is counter of the number of tasks suspended on the channel. This counter is zero if the channel is empty, positive if it contains outputting tasks and negative for inputting tasks. The second and third words are, respectively, head and tail pointers to a linked list of suspended tasks. Thus several outputting tasks can share a channel to one inputting task. Tasks which output concurrently are queued on this channel while the inputting task handles one message at a time. Since output messages are always accepted (they are never prevented from entering the queue) and are processed in the order they were received, this technique could be used as a basis for a truly fair ALT.

As any programmer using Razor can, indeed must, explicitly declare each task switch, it is possible to write efficient hardware polling routines. These are usually not as efficient as ones using interrupts, however they are sometimes necessary. A clear occam directive to invoke a task switch would be appreciated.

Razor provides a task called WATCHDOG, which the user may spawn. Using a pair of counters, this task and the 1 ms timed interrupt routine monitor each other. In this way, errant user tasks can be detected and located. A separate hardware watchdog is also handled here. These are common among embedded control systems but sadly one has not been integrated with the transputer.

Observations and enhancements

After extensive use, a latency problem was identified with the INPUT/OUTPUT rendezvous. At any given time, most tasks are to be found waiting in an INPUT guard for some data to process. When a corresponding OUTPUT occurs: the message is

copied by the kernel to the input buffer, the inputting task is added to the end of the scheduling queue and the outputting task is resumed. Unfortunately further processing of this message does not occur until the inputting task is rescheduled, which is only after all the other active tasks in the queue have had their turn. When a message crosses several tasks, as happens frequently in our layered software, a significant and unnecessary delay can be introduced. The solution adopted was to ensure that, no matter which task is the first to arrive at a rendezvous, the inputting task is always the first to be resumed. The outputting task is put to the end of the scheduling queue. This reduces message latency with apparently no ill effects. The transputer appears to handle the rendezvous with the original, slower, technique.

A bitswitch within each unit was assigned as a debug-/run-time indicator. When the software finds an error and hits a trap, a monitor program examines this bitswitch. If this indicates debug mode, it then enters the debug software which allows the programmer to examine the state of the system. If the bitswitch indicates runtime mode, the unit is immediately rebooted and restarted. Inevitably, equipment containing software bugs was installed on site, where user pressure soon made it desirable to know more about these runtime traps. The monitor software was subsequently modified so as to dump this information to the Ethernet bus, before the unit was rebooted. Another unit, in a convenient location, collected this information and transferred it to a printer. Although crude, this technique proved to be vital in locating errors that we could not reproduce in the lab. On a more commercial note, this was also responsible for a regaining some customer goodwill, as he could actually see something happening to solve his problems.

The debugging and testing of many concurrent processes were found to be difficult problems. Small changes were made to the scheduler to record the last task entry point and the address and length of the last message transferred. These are of some assistance after an error trap has occurred. Most effective, however, is the methodology of testing each task individually and in isolation. A general purpose test harness was written. This can be run in parallel with the code under test and provides a window into the running system. With this tool, a programmer can 'on the fly' examine and edit memory, examine the timer and the scheduling queue, send and receive messages with `INPUT` or `OUTPUT` and `WAKE` sleeping tasks. A simple programming language also allows a series of events or messages to be generated.

In practice `LOCK` and `UNLOCK` were rarely if ever used. This is because task switch points are well defined and also, instead of sharing data structures, programmers preferred to pass messages between tasks. Similarly `SLEEP` and `WAKE` were usually only used between interrupt and background tasks, where `INPUT` and `OUTPUT` are obviously unavailable. Two or more shared buffers were often used to transfer the actual data.

Further enhancements

The power and address space of the Z80 processor became limiting and so a new one was sought. Eventually the 32000 series from National Semiconductor was selected. It was necessary to rewrite the scheduler and this prompted us to add some extra features. We also had to modify some of the directives to make them compatible with the type-checking of the Pascal compiler that we intended to use. Consequently,

the following routines were added:

- ▷ `FORK` – Clone current task n times (Pascal-compatible `SPAWN`)
- ▷ `SEND/RECV` – Non-blocking inter-task communication
- ▷ `NEW/DISP` – Access to fixed-size block heap
- ▷ `NEWCHAN` – Obtain a new channel etc. from the heap
- ▷ `TIMEWAKE` – Initiate delayed `WAKE`
- ▷ `TIMEOUT` – Initiate delayed `OUTPUT` (cf. Timer `ALT`)

The routines `SEND` and `RECV` formalise the transfer of memory blocks between interrupts and background tasks. The first word of each block is reserved for use by these directives. Interrupts may use `SEND` to append memory blocks to a linked list, whilst `RECV` may be used by a background task to receive them. In practice, they can be used much like `OUTPUT` and `INPUT` with the exception that the programmer must provide a path to return unused blocks to the consumer task.

Often, we found it necessary for the software to configure itself at runtime. For example, the same code had to run on units with 4, 8, 12 or 16 serial ports and with varying amounts of memory. A fixed size block heap was introduced, which configures itself on initialisation. The directive `NEW` allows tasks to acquire their workspace at runtime. The remaining memory blocks can be obtained and released, as required, with `NEW` and `DISP`. In communication systems, throughput is often directly related to the amount of buffer space available, so this can improve system performance by allowing memory to 'migrate' and supplement the buffers of busy channels.

When debugging software based on Razor, we often encountered a problem that will be familiar to users of the TDS post-mortem debugger. Unless one knows in which channel it is suspended, it is often difficult to locate a lost task. This is because, in both Razor and the transputer, the tasks return address can be at virtually any location within its stack (or workspace). A pointer to the exact location will (or should!) be located in one of many linked lists, which can be tedious to search by hand. All 32000 series processors have two stack pointers and this fact was exploited to overcome this problem. To each task is assigned a *task control block* (TCB) at a fixed location in the supervisor stack space. Details such as the task's return address are always stored at the same offset within each TCB. Thus, knowing the address of a task's TCB, one can quickly establish where it is suspended. Also stored in each TCB is a pointer to the one of the previously executing task. This is updated on every task switch and is invaluable on an error trap. It allows the monitor to list the tasks that were executing up to the point of failure. This list is displayed together with other information such as each task's entry and exit points and times. This often helps to establish the series of events that cause the problem.

James Kidd +44 925 601 735
Altis
9 Brackley Street
Warrington WA4 6DY
United Kingdom

ALT WHILE CONDITION

Piers A. Shallow

In the light of recent articles on the various methods of implementing "fair ALT"s [27, 28, 29] and a lack of enthusiasm by Inmos to produce one [30], I would like to suggest an additional occam command "`ALT WHILE` *condition*" which can be compiled using the existing transputer instruction set [31].

The reason for my suggestion stems from the need to select more than one alternative of an `ALT` and the need to have fairness in the selection, in order to prevent hogging and starvation. Current attempts to obtain this fairness are being implemented in occam by manipulating the use of the existing `ALT` commands within the `WHILE` loop. The fairness is being obtained from the long term effect of cycling through the alternatives and not the individual selection of an alternative of the `ALT`. By writing the solutions in occam one is incurring an enormous penalty due to the amount of processor time wasted in implementing them. The number of processor cycles required to implement these solutions in occam [27] is about a factor of three [29] compared to the simple `ALT` command. It is for this reason that I believe that a fairer `ALT` command should exist as an occam construct and the solution that I am suggesting requires about the same amount of object code and processing time as the existing `ALT`s.

What I am suggesting is that there should be at least two types of `ALT`s, one of which is a single `ALT` and another a repetitive `ALT` – the `ALT WHILE` *condition* of the title. The syntax of alternation [32] would look like this:

```
alternation = ALT
                {alternative}
            | ALT replicator
                alternative
            | PRI ALT
                {alternative}
            | PRI ALT replicator
                alternative
            | ALT WHILE boolean
                {alternative}
            | ALT replicator WHILE boolean
                alternative
alternative = guarded.alternative | alternation
```

The single `ALT` is the one currently implemented in occam where one and only one alternative is selected before the `ALT` is terminated. The present method used for determining which alternative should be selected, is the same for both the `ALT` and `PRI ALT` and is dependent upon the order in which the alternatives are listed in the code. It is the first alternative found to be ready.

The repetitive `ALT` is one which will keep selecting alternatives until the predefined condition is no longer true, whereupon the `ALT` is terminated. The method by which the alternatives are selected works on the principle of the 'round robin'. (I

admit it is not perfect but then neither is the current ALT!) Although there are many ways in which the round robin can be achieved, the method which I am proposing is one based on its simplicity in its implementation in machine code. The ALT always starts testing the first alternative and then the remaining alternatives in turn, selecting and processing those channels found to be ready. In this implementation the condition is only tested once every cycle, however there is no reason why the condition should not be tested after each selected alternative.

Before I can describe my proposal in more detail, I will would like to briefly describe the way the ALTs are currently implemented. Take for example the following program:

```
PROC current.alt (CHAN OF p.data.type chaninA, chaninB, chaninC,
                                      chanout                  )

  PROC p (CHAN OF p.data.type chanout, data.type b)
    chanout ! b -- or some processing of the data
  :

  data.type data:
  BOOL running:
  SEQ
    running := TRUE
    WHILE (running)
      SEQ
        PRI ALT
          chaninA ? data
            p (chanout, data)
          chaninB ? data
            p (chanout, data)
          chaninC ? data
            p (chanout, data)
:
```

the object code of the ALT will take the form of:

```
          alt;
          enable (Ga); ...; enable(Gn);
          altwt;
          disable(Ga); ...; disable(Gn);
          altend;
          process(a); ....; process(n);
END:
```

where

enable(Gx) is the sequence of code: `ldl c; ldl e; enbs;`

disable(Gx) is the sequence of code: `ldl c; ldl e; ldc L; disc;`

process(x) is the sequence of code required to carry out the associated action, followed by a jump to END.

where

e is the boolean

c is the channel

```
START:
        ldl condition;
        cj .END;
        alt;
        enable (Ga); ...; enable(Gn);
        altwt;
        disable(Ga); ...; disable(Gn);
        altend;
TESTA:
        test(Ga); cj .TESTB;  process(a);
TESTB:
        test(Gb); cj .TESTC;  process(b);
TESTC:
        ...;
TESTN;
        test(Gn); cj .TESTEND process(n);
TESTEND:
        j .START
END:
```

where
test(Gx) has the sequence of code:
`mint; ldl c; ldnl 0; xor`, or `ldl -1; stl 0; ldl c;  ldc 1; ldc o; disc.`

Figure 24: implementation of the repetitive ALT

L is the offset between the **altend** instruction and the start of the instruction sequence of its associated **process(x)**

The single ALT is initiated (*alt*); all the channels are enabled (*enbc*); the process is de-scheduled (*altwt*) until data arrives on a channel where upon the process becomes active; all the channels are tested and disabled (*disc*) in turn and the appropriate processing action is taken for the first ready alternative, after the ALT has terminated (*altend*), whereupon the program jumps to the end of the ALT code.

My implementation of the repetitive ALT is quite simple and is very similar to the current implementation of the ALT. The object code takes the form shown in figure 24. At first, the condition is tested and if it is found to be true the ALT is initiated (*alt*); all the channels are enabled (*enbc*) and the process is de-scheduled (*altwt*). When data arrives on a channel, the process becomes active; each channel is tested and disabled (*disc*) in turn and the appropriate processing action is taken for the first ready alternative after the ALT has terminated (*altend*). Once this processing action has terminated the remaining channels are then tested. If any of the remaining alternatives are found to be ready, their associated actions are taken. Once all the channels have been tested and disabled the condition is re-tested and if it is found to be true then the ALT is re-initiated; all the channels are re-enabled and the process is de-scheduled. This ALT process is then repeated until the condition becomes false.

As to how the "ALT *replicated* WHILE *condition*" command could be implemented, I do not intend to say much other than the testing, disabling and processing could take the form shown in figure 25. As the Transputer Instruction Set manual does not say if the are any hidden side effects when using the eleven instructions associated

with ALT, would someone from Inmos like to comment on my suggestion?

* * *

Since having written this article, it is becoming apparent that by removing WHILE loop mechanism from the ALT WHILE suggested, would yield greater scope and flexibility in its use of the command. This would producing an ALT-ish command my.ALT.mechanism which could be nested within the existing WHILE *condition* command.

For example – if a time out was required on a set of channels, then the ALT WHILE *condition* would be neater written as :-

```
WHILE condition
  SEQ
    ALT
      TIMER ? AFTER time.out.time
        ...
      my.ALT.mechanism
        chanA ? data.type.A
          ...
        chanB ? data.type.B
          ...
        ...
        chanN ? data.type.N
          ...
    TIMER ? time
    time.out.time := time + time.out.delay
```

As to what this command should be called and whether it is really an ALT is questionable!

I would like to thank N. George for his comments.

References

[27] Alan Chalmers, *Useful titbits*, OUG newsletter Nº 9, Summer 1988.

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

[29] P. A. Shallow, *Really efficient multiple buffering in occam and efficient fair* ALTs, OUG newsletter Nº 12, January 1990.

[30] David May, during panel session, 9th occam user group technical meeting, Southampton, September 1988.

[31] Inmos Ltd, *Transputer instruction set – a compilers writers guide*, Prentice-Hall International.

[32] Inmos Ltd, *occam 2 reference manual*, Prentice-Hall International.

P. A. Shallow
c/o Danescourt
Pond Road
Woking
Surrey GU22 0JT

```
INIT.DISABLE
        ldc       start;
        stl       i;
        ldc       count;
        stl       (i + 1);

START.DISABLE:
        disable(Gi);
        cj .OMIT;
        ldl i;
        stl temp;
OMIT:
        ldlp      i;
        ldc       (INIT.TEST - START.DISABLE);
        lend

INIT.TEST:
        ldl       temp;
        stl       i;
        ldc       count;
        stl       (i + 1);

START.TEST:
        test(Gi);
        cj .NEXT;
        process(i);
NEXT:
        ldlp      i;
        ldc       (END - START.TEST);
        lend
END:
```

Figure 25: implementation of ALT *replicated* WHILE *condition*

REVIEW

OCCAM 2

John Galletly, pub. Pitman, 1990, pb. £12·99

The stated aim of this book is to provide a gentle and structured introduction to Occam. It succeeds. Eleven chapters lead the reader from primitive processes and basic data types to timers, configuration, and general parallel programming techniques. Throughout the book reference is made to the transputer family, and an appendix deals with the TDS and the folding editor.

The first chapter deals with the three basic processes, going on to SEQ, PAR, and ALT. It introduces two-way communication over channels and then considers deadlock and graceful termination of programs. Large pictures and large fonts help to make the

book easy reading, and discussion of such conceptual novelties as deadlock benefits from this approach. Examples abound, even when discussing simple ideas like the nesting of blocks of code, and so if difficulty is encountered in understanding the raw statement of a problem, a fragment of Occam is available for clarification. Despite this emphasis of clarity over brevity, once or twice I felt that I would have liked one more sentence to tie up a section. For example, the first discussion of `SKIP` and `STOP` mentions that they are useful, but does not say why. It would have been better if Galletly had stated that they can be used as place-holders for yet-to-be-written processes.

The discussions of data types, arrays, and maths operations are exactly as might be expected, after which we come to `ALT`. This chapter is very clear, and includes important points like the fact that the implementation is free to choose the same guard repeatedly if more than one input is available. But the section on `IF` again leaves one sentence unwritten. It says that `TRUE` may be used as a catch-all guard, but it must be put last. This is true, but it would be easier to remember this if we were told that this is due to the guards being examined in textual order (i.e. from the top of the page downwards). The contrast with the commutativity of `PAR` is not made, which is a pity.

The many Occam fragments here as elsewhere are presumably meant to be 'lifted' by the student, to save needless re-invention. In general they are neat and clean, but occasionally they are simply not the best way of writing something. A `WHILE` loop which copies input to output until a certain character is read is initialised by setting the local variable to be the space character. But maybe we do not want spurious values of this sort – why not simple have an input to the variable, and then enter the loop? But this is a minor quibble.

Functions, procedures, abbreviations and retyping are dealt with in chapter 7, and protocols are considered at length in chapter 8. These chapters, and the following ones on timers, priority, ports and memory allocation are all clear, and show when software can and cannot ignore hardware considerations. After a chapter on placement and configuration, the last chapter in the book deals with approaches to parallel programming. It discusses geometric and algorithmic parallelism, and then introduces pipelines and farming, with examples. It concludes with a section on efficiency, which emphasises the communication overheads involved with the transputer, and the necessity of differentiating between compute-bound and i/o-bound tasks.

In general, this book is a clear introduction to Occam on the transputer. It is very easy going, and does not require the novice to look for other sources of information. This is achieved at the cost of a complete mixing of different levels of abstraction. If you wanted to program in Occam on a non-transputer system, you would find it hard to distinguish between features of the language and features of the transputer implementation. However, as we are not exactly drowning in non-transputer versions of Occam, this is not really a problem (until the H1 comes along).

If you want just one book on Occam, this is not it. It is not sophisticated enough to remain the book you turn while writing your wormhole-routing algorithm. But this was not its aim. If someone finds Jones & Goldsmith too dry, or if they have managed to tie themselves in such knots that they wish to return to basics, this book is a good choice. If you are teaching Occam to sixth-formers, maybe the many

repetitions (formal statement, explanation, two examples) will be useful. I suggest that this is a book to recommend to someone who claims to be flumoxed by the ideas of parallelism. This book will start them off properly, and it is perhaps an achievement if a book is so clear that it quickly makes itself redundant.

Summary: no substitute for the standard works, but a good gentle introduction.

Michael Jampel, Programming Research Group, Oxford University

PRODUCTS, SERVICES AND ANNOUNCEMENTS

NEW INMOS TRANSPUTER VERSION OF C EXECUTIVE

Real Time Systems Limited, Douglas, Isle of Man

RTS has developed, under contract to Inmos Limited, a complete real-time kernel for the transputer. Device drivers for the Inmos iserver and for the transputer's physical and virtual links are included in the package.

C Executive is a multi-tasking, ROM-able kernel already available for many CISC and RISC processors. It provides the transputer with a general-purpose interrupt mechanism and a deterministic pre-emptive scheduler. Multiprocessor systems may be configured with tasks on separate processors communicating via virtual link drivers. The package supports all current transputers and has been designed with a view to the H1.

An optional DOS compatible file system, CE-DOSFILE, is available, providing device drivers for SCSI, memory and Inmos *iserver* disks.

This is a brief summary; detailed information from:

Alan Cleary
Real Time Systems Limited
P.O. Box 70
Viking House, Nelson Street
Douglas, Isle of Man
British Isles

alan@rts-iom.co.uk
Tel: +44 624-661500
Fax: +44 624-663453

DYNET DYNAMIC NETWORK ROUTING CHIP FOR TRANSPUTER LINKS

Dynet is a routing chip composed of eight switches – capable of connecting eight transputer link inputs and eight link inputs – and eight arbiters – each of them being associated with one or several outputs, according to the topology of the network.

On request from one of the input links, the arbiter associated with the requested output will establish the connection. The arbitration is performed independently of the existing connections and has the ability to solve any possible contention problems.

Once the path is established between input and output, the chip is fully transparent and the transputers are directly connected.

And finally when the message gas been transmitted, the arbiter releases the path and makes it available for any further requests. Dynet is fully cascadable to build

networks from eight to 256 transputers. Further more, it is suitable for different topologies such as a torus, hypercube or delta.

Why custom silicon?

The objective was to design a routing device to interconnect up to 256 transputer nodes, on a delta-like topology. The larger a single device would be in terms of switching capabilities, the smaller the number of devices to build a 256 transputer network. Furthermore, as technology improves a shrink and redesign of the ASIC would increase the capacity of the device.

The current 1·5μm ES2 allowed the development of an 8×8 routing chip.

Esprit project Nº 967 – PADMAVATI

The design of Dynet was done by an engineer from Thomson-CSF in collaboration with European Silicon Structures as part of an Esprit project, PADMAVATI. The collaborators on this project are Thomson-CSF, GEC, First, NSL and Prologia, and the aim is to develop a parallel architecture and an appropriate environment for artificial intelligence applications.

Thomson-CSF
Division Outils Informatiques
Direction Commerciale
Parc d'Activité Kléber
160, bd de Valmy
B.P. 82
92704 Colombes Cedex
France

Tel: +33 (1) 47 60 30 00
Fax: +33 (1) 47 60 33 57
Telex: THOM 616 780 F

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.

Background INMOS *transputers* form a family of VLSI components for the elementary construction of multi-processor computing systems. The T800 integrates on a single chip a 10 MIPS 32-bit general-purpose processor, a 1·5 MFLOPS (ANSI–IEEE 754 standard) floating-point processor, eight 20 Mbaud DMA channels, 4 kbytes of fast static RAM and a programmable 32-bit external memory interface. *Transputer* networks of arbitrary size, shape and power may be created by simple wire connections of the DMA channels. A common bus, which would cause contention bottlenecks, is *not* used for combining *transputers*.

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. The model of parallelism provided by *occam* is a central (not 'added on') feature and is directly supported by the INMOS *transputer*. This model reflects major software engineering features such as abstraction, structuring, and information hiding and has important benefits for the life-cycle development costs of large systems (not least through the establishment of libraries of reusable software/hardware components).

The *occam* and *transputer* technologies provide well-engineered, production-quality and commercially available tools for parallel processing applications. They may be used to construct anything from high-performance fault-tolerant real-time embedded systems through to general-purpose super-computers. They represent an early release from the parallel processing research community into practical engineering. This course will emphasise their strengths, weaknesses and likely future developments.

Course Members If you have picked up basic *occam* syntax and semantics and are wondering how to use it to engineer high-performance high-security systems, this course is for you. 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 170 participants from Industry and Academia worldwide.]*

Course Methods Time will be divided equally between informal lectures and 'hands-on' experience through practical exercises and a 'mini-project'.

Course Materials Copies of all OHP slides (approx. 300) will be provided together with example solutions for all set exercises and a large quantity of utility and demonstration *occam* software (approx. 350 kbytes). Various printed notes (e.g. a summary of the *occam* language and relevant papers) will also be distributed. 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 system 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 174 transputers (including 140 T800s) with a gigabyte distributed file store and three high resolution graphics workstations.

Course Length Five days. Our publicly advertised courses would normally be a continuous block of five consecutive week-days. However some companies may prefer to release people for one day per week (over five weeks). Special arrangements can be made – see *Contact* below.

Course Cost £450 per person (including lunches and light refreshments, but excluding accommodation, evening meals or breakfasts).

Dates Course №17: 8–12 April 1991.
Course №18: 24–28 June 1991.

Contact For further details, or to enquire about special arrangements for 'spread-out' courses 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

CRESCO DATA NEWS

Cresco Data has announced its new PC–AT transputer board and a range of TRAMs.

The PC–AT board is a new, B008 compatible motherboard with added features for the IBM PC/AT and Apollo workstations. It supports up to eight transputers per board in a combination of mounted components and TRAMs. The master transputer, the high-speed interface and the link-switch (C004/T222) are integrated on the motherboard.

In the minimum configuration, the CD-TB10/AT forms a reliable and high-performance, cost-effective entry level solution, upgradable for future requirements. The onboard zero wait-state memory varying from 1 to 16 Mbytes, allows even memory intensive applications to be developed and executed without upgrading the board. Though the motherboard is B008 compatible, it features an 8/16 bit bus interface with a FIFO construction, yielding in excess of 600 kb/sec sustained transfer rate.

The new family of high performance, size 2, transputer modules – CD-TRAM are low profile TRAMs, compatible with the Inmos B008 and Cresco motherboards. A choice of either T800 or T425 with 1 to 8 Mbyte of zero wait-state memory are available. The large amount of memory available makes it ideal for running very large programs. Onboard jumpers are provided for selection of either 10 or 20 Mbit/sec link speed and insertion of wait-states. Furthermore, the TRAMs may be stacked.

Cresco Data A/S
Øresundvej 148
DK–2300 Copenhagen S
Denmark
Tel: +45 31 55 42 70
Fax: +45 31 55 01 53

QUINTEK INTEGRATED IMAGE ANALYSIS SYSTEMS

Quintek Limited, Bristol, UK

In the wake of continued popularity of the HarleQuin frame grabber, Quintek proudly announce new image processing products which combine transputer power with Zoran floating-point vector processors for real-time image analysis.

The MosaiQ is a full-colour frame grabber designed to take full advantage of the flexibility of high-performance CCS cameras for special applications. The SeQuin includes high-speed buffered data-ports for capturing and processing data from digital sources. A novel feature of both these AT-format boards is a size4 TRAM slot which enables the user to adapt his system with extra processing or I/O as needed. Both boards are available in a number of memory/processor options.

These two boards form the basis of the Quintek Integrated Image Analysis System. We supply a complete workstation and camera with appropriate software tailored to your application. The system is based on the Zoran fast vector DSP chip, and comprises:

- a colour frame grabber with full colour display, overlay, cursor and plenty of memory;
- a choice of input – either from a 3-colour camera or from and 18-bit digital input port using a FIFO at speeds up to 25 Mbyte per second;
- 2 Zoran ZR34325 DSP chips for pixel-level processing;
- an Intel i860 chip for fast high-level C programs;
- an excellent software environment including the Zoran Development System and NEL's IPLIB parallel processing library;
- a state-of-the-art camera.
- Optional components include a compatible standard graphics board (Splash), a DSP22C TRAM to support existing applications, and fast optical communications between stations using the Onelan MIF card (multiple interface to FDDI).
- The standard CCD camera we supply is capable of asynchronous sampling and triggering and pixel clock-synchronisation with optional low-light intensifier and fibre-optic front-ends. Also soon to be available are high-resolution systems (1024×1024), X-ray sensitive and low-temperature low-light photon integration cameras.

For further information on any Quintek products, please contact

Matthew Page Tel: +44 272 628196
Fax: +44 272 628717

ADVANCED TRANSPUTER ENGINEERING WORKSHOP

Computing Laboratory, University of Kent at Canterbury
Control Laboratory, University of Twente in Enschede, Holland

Course Objectives To go deeper into the process of designing generic *occam* procedures. To acquire a better understanding of the inner working of the *transputer*, so that you can build faster, smaller and more efficient programs. To learn how to really use the post-mortem analyser. A pre-requisite for this course is some reasonable experience in working with occam and transputers.

Further Details In this course, practical sides to the development of *occam/transputer* systems will be shown. Subjects will range from how to design a size-independent matrix library for two-dimensional arrays or the issue of how to do dynamic code loading on a

transputer network. A very simple message passing network kernel with an efficient buffering scheme will be discussed. The techniques for overlapping communication with processing will be presented.

Course Members If you are using *transputers* and you want to know how to squeeze the most out of them, this is the course for you. If you always wanted to know what all these Iptr, Wptr things are that the debugger keeps throwing at you, you've come to the right address. What to do if it keeps telling you 'Invalid Wptr'? We will tell you. If you want to build more generic procedures in *occam*, without sacrificing performance or memory requirements, this course will tell you how to do it. If you want to know how to balance an algorithm over a network of processors, join in. You want to develop a stand-alone application which must run outside TDS/OPS? We show you how you can do it.

Course Methods Informal lectures with lost of examples to look at, exercises to make and a small transputer network to play with. 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 174 transputers (including 140 T800s) with a gigabyte distributed file store and three high resolution graphics workstations.

Length & Cost Two days @ £180 (including lunches and light refreshments).

Dates Workshop Nº 2: 26–27 March 1991,
University of Kent at Canterbury.
Workshop Nº 3: 4–5 July 1991,
University of Kent at Canterbury.

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).

ELECTRONIC ARCHIVE SERVER AT INMOS

The Transputer Archive-Server is an automated data service whose email address is `archive-server@inmos.com`. This server contains bulletins for transputer software and copies of public domain software related to transputers. This server is continually updated. The contents of the Transputer Archive-Server include:

- The *cprot* archive which contains source code for a utility program which
  - converts Occam Constants to C defines
  - generates C code to interpret Occam protocols
- The *Check* archives (*checkocc*, *checkpc*, *checksrc*, *checksun3*, *checksun386i*, and *checksun4*) contain the programs comprising the *Check* utility suite of programs for various host systems.
- The *disassembler* archive contains the source code and a PC executable for a transputer code disassembler.
- The transputer demonstration archive (*demos*) contains demonstration programs for the transputer.
- The *forth* archive which contains documentation, source code, and executable files for a forth interpreter for transputers.
- The *iris* archive contains C source code and a makefile to support the development of a Silicon Graphics Iris host to transputer interface on B014 and Paracom/Parsytec boards.
- The *iserver* archive contains C source code and makefiles for the transputer *iserver*.
- The *Origami* archives (*origami*, and *origamip*) contain source code and PC executable code for the Origami folding editor.
- The PROM loader (*promldr*) archive contains source and executable files for program which lets you test a 'boot from PROM' program on a 'boot from link' transputer.
- The *s706beta* archive contains a software package which will convert toolset code files to a form which can be loaded into eprom programmers. This archive contains a subarchive with the source code.
- The *supervis* archive contains an Occam program which measures how 'busy' a transputer is. This archive contains the source code and documentation.
- The *unixoccam* archive contains a compressed 'tar' format file for a Unix Occam compiler which generates executable code for the DEC 11 series computers. This is the compiler that was written by 'Gil' and posted to the net. We would like to give full credit to the author but our news expired the original messages and we cannot find the author's name on any of the traffic relating to the compiler.
- The *yank* (Yet Another Network Konfigurer) archive contains documentation, executable code, and source code for a utility that generates transputer network configuration files from the output of the Check program (which is also stored in the archive-server).

In addition to these new archives, one new command has been added to the server. The *product* command will retrieve all software bulletins associated with a particular product. For instance the command

```
product D705A
```

will retrieve all software bulletins related to the D705A. Please note that the letters in the product identifier must be upper case if this command is to find the products.

Many of the commands to the archive-server will generate large amounts of data. Therefore we recommend you use the *pack* or *archive* commands to compress the files. The archive server currently supports the Unix *compress* utility and the *zoo* and *lharc* archivers which compress files.

If you need additional information on using the Transputer Archive-Server, send a message with the command

```
help
```

in the message body to `archive-server@inmos.com` and the server will send a file listing the commands and procedures for using this server. *Tony Schneider*

ELECTRONIC GRAPEVINES

If you are an electronic mail user, you may want to know about two electronic mailing lists, carrying discussions on occam and the transputer. These offer you a mechanism rapidly to distribute information, short papers, programs, problems, even gossip about Inmos, to the sort of people who may be interested. You may even want to read this sort of thing. We even have subscribers from Inmos who can sometimes be goaded into authoritative declarations.

Each list has distribution points both in the UK and the USA. To join try making contact with the appropriate address: for the occam mailing list contact

`occam-request@uk.ac.oxford.prg` (in the UK)

for the transputer list contact

`transputer-request@tcgould.tn.cornell.edu` (in the USA)

or `transputer-request@uk.ac.oxford.prg` (in the UK).

Please choose the contact the address that is nearest you, to reduce duplicated traffic across the Atlantic. The transputer list traffic is also available in newsgroup `comp.sys.transputer` on USENET.

Request for help

There used to be an `occam-request` address in Syracuse which administered the distribution of the occam list in the USA. This service has had to be withdrawn so the occam list is not at present being distributed on what I call the other side of the Atlantic. If you would be prepared and able to handle the task, will you please get in touch with `occam-request@uk.ac.oxford.prg`. *gj*

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.

A word may be in order about London telephone numbers: formerly +44 1 (or in the UK, 01) numbers have been divided into +44 71 (in the UK, 071) and +44 81 (in the UK, 081). Mind you, there rarely seem to be that many London numbers in the *Newsletter*.

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@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`. That particular site address, `uk.co.inmos`, ought to be interchageable with `inmos.com`. *gj*

REFERENCE

CONTACTS FOR RELATED GROUPS

Transputer Users Group: Argentina

The purpose of TUGA is to prompte the use of parallelprocessing techniques via the transputer with the occam language as the most natural starting point. The secretary is

Esteban R. Gillanders
Keydata S.A.
Crisólogo Larraide 1801
(1429) Buenos Aires
Argentina

Tel: +54 1 70-4467/3281
Telex: 23096 KEYSA AR
Fax: +54-1-11-2426

Australian Transputer and Occam User Group

The person to contact for details of future activities is:

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@oz.rmit.xx.minyos

Bulgarian Transputer and Occam User Group

Readers in Bulgaria interested in the formation of this group are invited to contact

Dr Simeon Patarinski Texlex: 22628 BANMAT BG*
Bulgarian Academy of Sciences
Block 4, Acad. G. Bonchev St
1113 Sofia
Bulgaria

French Transputer Users Working Group

Traian Muntean traian@fr.imag.imag
IMAG-LGI
b.p. 68
38402 St Martin d'Heres CEDEX
France

Deutschen Occam-Interessengemeinschaft der Transputeranwender

DO IT can be contacted through its secretary:

Heinz Ebert
Im Heidigen 3
5206 Neunkirchen-Seelscheid 2
Germany

The president is:

Joachim Stender
c/o Brainware GmbH
Gustav-Meyer-Allee 25
1000 Berlin 65
Germany

and vice presidents are:

Frank Heinemann
c/o Fraunhofer-Institute
Kleiststraße 23–26
1000 Berlin 30
Germany

Peter Eckelmann
c/o Inmos GmbH
Danziger Straße 2
8057 Eching b. München
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 Tørngren
Maskinelement
Kungl. Tekniska Högskolan
100 44 Stockholm
Sweden

Tel: +46-8-790 7849
Fax: +46-8-723 1730
stug@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 newsletter submissions, who is Lyle Bingham. Their addresses appear on page 106.

SPECIAL INTEREST GROUP CHAIRMEN

Artificial intelligence

Joachim Stender
c/o Brainware GmbH
Gustav-Meyer-Allee 25
1000 Berlin 65
Germany

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
R.Peel@uk.ac.surrey.ee

Formal methods

Michael Goldsmith
Formal Systems (Europe) Ltd
Unit 7, The S.T.E.P. Centre
Osney Mead
Oxford OX2 0ES
United Kingdom
Tel: +44 865 728460
Fax: +44 865 793165
michael@uk.ac.oxford.prg

Image processing and vision

Hugh Webber
RSRE
St Andrews Road
Great Malvern
Worcs WR14 3PS
United Kingdom
+44 684 894728
hcw@uk.mod.rsre

Graphical program development tools

Mike Roberts
The Centre for Information Engineering
City University
Northampton Square
London EC1V 0HB
United Kingdom
+44 71 253 4399 x3889/3877
m.roberts@uk.ac.city

Hardware

Denis Nicole
University of Southampton
Department of Electronics
and Computer Science
The University
Highfield
Southampton SO9 5NH
United Kingdom
+44 703 787167
D.A.Nicole@uk.ac.soton.ecs

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

Real time

André Bakkers
Twente University
EL-BSC Dept.
P.O. Box 217
7500 AE Enschede
Netherlands
Tel: +31-53-892794
+31-53-892790
Fax: +31-53-354003
Telex: 44200 thtes
elbscbks@utwente.nl
elbscbks@henut5.bitnet

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
hopkinsm@isnet.inmos.com

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 833458
rdb@uk.co.rokeman

Gordon Harp
RSRE
St Andrews Road
Great Malvern
Worcs WR14 3PS

+44 684 894824
jgh@uk.mod.rsre

Geraint Jones
Programming Research Group
Oxford University Computing Laboratory
11 Keble Road
Oxford OX1 3QD

+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
ac1jmk@uk.ac.sheffield.primea

Roger Peel
Department of Electrical Engineering
University of Surrey
Guildford
Surrey GU2 5XH

+44 483 509284
R.Peel@uk.ac.surrey.ee

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

Secretary
+44 454 616616
oug@uk.co.inmos

Stephen Turner
Department of Computer Science
University of Exeter
Prince of Wales Road
Exeter EX4 4PT

Newsletter editor
+44 392 264048
steve@uk.ac.exeter.cs

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

Hussein Zedan
Department of Computer Science
University of York
York YO1 5DD

Tel: +44 904 432744
Fax: +44 904 432767
zedan@uk.ac.york.minster

... continued from front cover

This newsletter was prepared using facilities provided by the Department of Computing Science at the University of Glasgow, and by Oxford University Computing Laboratory; printed and distributed on behalf of the Occam User Group courtesy of INMOS Limited.