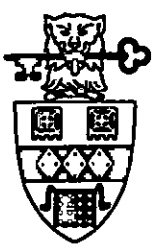
VECTOR

APL88: Conference Reports and Photos

- **Reviews of PL/PC and Debugger**
- **Using quad-NA to Eliminate WSFULL**
- **David Ziemann on Efficiency**
- **Dissembling with Allan Gay**
- **Full Index to Back Numbers**



The Journal of the British APL Association

A Specialist Group of the British Computer Society

Vol.4 No.4 April 1988

Contributions

All contributions to VECTOR should be sent to the Editor at the address given on the inside back cover. Letters and articles are welcomed on any topic of interest to the APL community. These do not need to be limited to APL themes nor must they be supportive of the language. Articles should be submitted in duplicate and accompanied by as much visual material as possible, including a photograph of the author. Unless otherwise specified each item will be considered for publication as a personal statement by its author, who accepts legal responsibility that its publication is not restricted by copyright. Authors are requested wherever possible to supply copy in machine-readable form ideally text files on a 5¼" IBM-PC compatible diskette. For other standards, please contact the Editor beforehand. Program listings should indicate the computer system on which they have been run. APL symbols should be displayed on a separate line and not embedded in narrative. Except where indicated, items published in VECTOR may be freely reprinted with appropriate acknowledgement.

Membership Rates 1986-87

Category	Fee p.a. £	Fee p.a. $	VECTOR copies	Passes
Nonvoting student membership	5		1	1
UK Private membership	10		1	1
Overseas private membership	18	27	1	1
Supplement for airmail (not needed for Europe)	8	12		
Corporate membership (UK)	85		10	5
Corporate membership (Overseas)	140	210		
Sustaining membership	360		neg	5

The membership year runs from 1st May to 30th April. Applications for membership should be made on the form at the end of the journal. Passes are required for entry to some Association events and for voting at Annual General Meetings. Applications for student membership will be accepted on a recommendation from a course supervisor. Overseas membership rates cover VECTOR surface postage and may be paid in £UK or $US.

Corporate membership is offered to organisations where APL is in professional use. Corporate members receive multiple copies of VECTOR and are offered group attendance of Association meetings. Partaking individuals need not be identified but a contact person should be nominated for all communications.

Sustaining membership is offered to companies trading in APL products; this is seen as a method of promoting the growth of APL interest and activity. As well as receiving public acknowledgement for their sponsorship, sustaining members receive bulk copies of VECTOR, and are offered news listings in the editorial section of the journal and opportunities to inform APL users of their products via seminars and articles.

Advertising

Advertisements in VECTOR should be submitted in typeset camera-ready A5 portrait format with a 20 mm blank border. Illustrations should be black-and-white photographs or line drawings. Rates are £250 per page. A6 and A7 sizes are offered subject to layout constraints.

Deadlines for advertisement bookings and receipt of camera-ready copy are given beneath the Quick-Reference Diary.

Advertisements should be booked with and sent to Cathy Dargue, whose address is given beneath the Index of Advertisers.

CONTENTS

Editorial: Keep your Eyes on the Prize

by Adrian Smith

For me, Sydney was a wonderfully refreshing experience! Not just the smell of salty air from the harbour ferries, or even the 116n.o. from Chris Broad on the first day of the Bicentennial Test Match: more the feeling from almost everyone there that we knew where we were going. In any grouping of like-minded individuals, it is essential to have such gatherings, lest we begin to drift around in the kind of short-sighted fog which seems to beset all human endeavours which involve more than 7 people, or last more than 7 years!

APL86 was just the same ... and I would guess that those who returned from Dallas also felt this clearing of vision; the problem is to keep it going in the day to day business of running companies, getting hassled by users, cleaning up after the children, getting VECTOR out on time, and so on. Of course you can't, at least not for all that long, so that is why events like APL88 are so vital in assuring our collective future.

In short, we need to 'keep our eyes on the prize' as long as we possibly can. Then we need to gather ourselves together to renew our confidence, and to re-affirm our long-term commitment to APL, rather than to some particular APL project that we happen to be working on. During APL88, I felt that we really did hold certain truths to be self-evident; let me try and get them on paper before I too forget!

> APL is first and foremost a notation; it is the best way yet devised of doing mathematics on a computer. Many things which are currently hidden in the fog of conventional notation are crystal clear (apologies to Don McIntyre for the pun) when written down in APL. The fact that you have on your desk a 'native speaker' of this notation is a marvellous bonus; it is not the reason we are here. The reason we need APL is to help us see things more clearly. If we then had to code them in FORTRAN it would be a nuisance, but it would not be the end of the world!

> APL needs to pull together. We have done ourselves an incredible disservice by allowing the divergence into two fundamentally different flavours. Regarding APL as no more than a quick way of generating computer systems, such a split hardly matters. Regarding APL as a movement towards a more consistent and effective notation, it is an absolute disaster. We must get our act back together if we are to have any chance of gaining acceptance of anything beyond plain 'VS APL'.

> I-APL is our big chance; we must not fluff it! If we screw this one up we may never get APL accepted where it really belongs ... in the junior schools, the high schools, the Technical Colleges, and the Universities. It will retain a shrinking base in a few big companies, and we shall all get older. In 10 years time there will be rather fewer of us, and our systems will mostly be 10 years old, and due for replacement. Will we have the energy and the resources to fight for anything outside our own little patch? I rather fear that we shall not.

I know that there is always a temptation to keep a good thing to yourself. The medieval craft guilds were built around just this principle! I know that I have a bit of the reputation of a magician in some circles in my company; I could trade on this by keeping all my nice APL knowledge to myself, or I could try to broaden the base of APL experience and consequently undermine my (quite unjustified) reputation.

In my case, I know that the decision is an easy one: I have no desire to be a magician, and I really do want to encourage the growth and spread of APL. That is exactly the feeling I got from the overwhelming majority of delegates at APL88 (and indeed from APL86). Let's just make sure we do 'keep our eyes in the prize'; it's a long time to APL89, and it is all too easy to put tomorrow's personal advantage before APL's future.

Dates for future issues of VECTOR (note 5.2 dates)

	Vol.5 No.1	Vol.5 No.2	Vol.5 No.3
Copy date	4th June 88	1st Oct 88	1st Dec 88
Ad booking	10th June 88	10th Oct 88	12th Dec 88
Ad Copy	24th June 88	24th Oct 88	23rd Dec 88
Distribution	July 88	November 88	January 89

Quick Reference Diary

Date	Venue	Event
29 April 1988	London	BAA AGM and Meeting
16-20 August	Syracuse Univ. New York	ACM SigAPL workshop on APL in Expert Systems.
28-30 September	Canterbury	APL-Ication the Practical State of the Art APL Conference
21 October	London	BAA Meeting
18 November	London	BAA Meeting
13 January 1989	London	BAA Meeting
17 February	London	BAA Meeting
17 March	London	BAA Meeting
21 April	London	BAA Meeting

Please note that the date of the AGM was wrong in both VECTOR 4.2 and 4.3. The above date is the correct one (honest).

All British APL Association meetings are to be held at the Royal Over-Seas League, Park Place, near Green Park Tube station. Meetings begin at 2.00pm and close at 5.30pm.

ACM SigAPL is holding a tutorial workshop on APL Techniques in Expert Systems at Syracuse University from August 16th - 20th 1988. The workshop will have a faculty of about 5 and capacity limited to 50. The conference fee will be $495, including both classroom sessions and hands-on work. Further information from:

Ms D. Chimen
University College
Syracuse University
Syracuse
New York 13210
USA

Why don't YOU contribute to VECTOR?

by the VECTOR Working Group

VECTOR is produced by the editor, with the help of a working group; he shouldn't have to write it as well - not even with their help! Please write letters to the editor, and send them to Adrian Smith. Don't let us think there is nothing in VECTOR that you disagree with. Don't fail to let us know when you think we're doing it right. All the addresses you need are shown below!

Please write reports of anything newsworthy, or any forthcoming event that's worth previewing, and send them to Adrian Smith. Please let Cathy Dargue know of anything to go in the diary, e.g. an APL paper at a non-APL conference. Please send Anthony Camacho press cuttings about APL with source and date of publication on them. You would be doing the APL world good if you also wrote to the magazine or newspaper about it. They too like and print letters.

Please write general articles and send them to Adrian Smith. You may feel your code won't stand up to the scrutiny of the experts, but that shouldn't stop you from helping those who haven't reached your level. We need introductory material so that beginners find something of value to them in VECTOR.

Please send more technical articles to David Ziemann or Jonathan Barman. VECTOR wants to consolidate its position among the world's APL journals, so we need useful functions carefully explained to help APL professionals improve their performance.

We would also like to hear from people who are willing to review books and APL products. Val Lusmore gets material from all over the world; sometimes you get to keep the book or software, and you always get to keep the knowledge!

Have you an idea for a competition? Send it to Jonathan Barman or David Ziemann. Will you judge it? You are likely to see some very clever code if you do; marking a competition with a very large entry is a really good way of learning something new about APL.

This is your magazine! Help it to prosper!

Adrian Smith	Brook House, Gilling East, York YO6 4JJ Hm 04393-385, Wk 0904-653071 x3382
Cathy Dargue	60 Downhall Ley, Buntingford, Herts SG9 9TL Hm 0763-73155; Wk 01 436-9481 x2418
David Ziemann	Flat 3, 63 Queen's Cres.,Chalk Farm, LONDON NW5 4ES Hm 01 267-8032; Wk 01 436-9481
Jonathan Barman	Cocking & Drury Ltd, 155 Friar St, Reading RG1 1HE Wk 0734-588835
Val Lusmore	17 Barton St, Bath, Avon BA1 1HQ Wk 0225-62602
Anthony Camacho	2 Blenheim Rd, St Albans Herts AL1 4NR Hm 0727-60130

APL Training Courses

(Prices quoted are per course, unless otherwise stated)

Dates shown cover the period from April to July 1988. For confirmation of dates and/or further details please contact the vendor directly.

Level	Company	Days	Price(£)	Dates
Beginners	Cocking & Drury	3	445	12/4,26/4,17/5,7/6
	MicroAPL	1	poa	28/4,23/6
	Uniware	5	poa	call
	Mercia	3	390	26/4,7/6,12/7
Intermediate	MicroAPL	1	poa	5/5,30/6
ditto/PC	Mercia	2	260	11/4,15/6
	MicroAPL	1	poa	19/5,14/7
Advanced	MicroAPL	1	poa	12/5,7/7
	Uniware	5	poa	call
System Design	Cocking & Drury	4	700	9/5,4/7
	Mercia	3	420	17/5,6/7
Statgraphics	Cocking & Drury	2	280	6/4,22/6
	Uniware	2	poa	call
U'ware Toolkit	Uniware	2	poa	call
R.BASE 5000	Uniware	5	poa	call

The following vendors run courses, details of which may be obtained directly from the vendor:

APL People
Inner Product
Parallax

General Correspondence

The VECTOR working group welcomes correspondence on any topic affecting the APL community. All letters should be addressed to the Editor, and should indicate whether they are for the general or technical section. (Letters containing APL code will normally be classed as 'technical'.) The editor reserves the right to edit any letter unless the writer states that it is to be published in full or not at all.

The Future of VECTOR

From: C.E. Williams 12th February 1988

Sir,

PLEASE, PLEASE use all your influence on maintaining VECTOR in its present format. It is most professional and a book one can show with a certain amount of pride. To reduce it to a tatty few badly printed pages is the road downhill - fast!

I think the contents must have more for the beginner if I-APL is to prosper, but equally it must still cater for the advanced programmer, or again we will go downhill - fast. Surely one hiccup in the production of VECTOR is not a valid reason to introduce radical changes? Surely this is PANIC not a re-appraisal of the problem?

So PLEASE four issues as at present with the same format externally, at least. Let the Committee get things in perspective and re-arrange the production methods. There must be members willing to help. Let us hear what is required and assess the response.

Finally thanks a lot for 'Toward a Better Basic'. We need more of this type of article to encourage beginners. Thanks a lot too for all the work on I-APL. I was just about to give up on BAA, as I came to the conclusion that it was a secret society pledged to withhold all information on APL from all those not in the clique! I-APL came into being and, at last, I thought, a change of heart.

Yours sincerely,

C.E. Williams
Cold Keld
Loweswater
Cockermouth
Cumbria CA13 0RR

APL2 Available over JANET

From: Paul Barnetson 3rd February 1988

Sir,

For some time now we have been working on making an APL2 service available to British academics and researchers at little cost to them. We have at last succeeded in this and, as you will see from the information sheet below, such a service is now available out of Warwick University. I am sure that you will agree that this is a big step forward in making APL more available to academics and researchers who have previously been put off by its cost.

Would YOU like to have access to APL2 over JANET
and at little cost to yourself???

GLOSSARY

APL. A Programming Language. The famous algorithmic notation that was designed by Dr. Ken Iverson in the early 1960s, and developed as APL for IBM computers a few years later.

As Ken knew little about computers when he designed the notation, the resultant programming language turned out to be radically different from all other existing languages, which had been designed by "computer people". These differences caused the newly-announced APL to be hated by some computer die-hards, loved by many, while it even had a "cult" following amongst some who claimed, not without reason, that "APL forced the computer to work in the way that humans thought, while other languages forced humans to think in the way the computers worked".

The maturer APL has now taken its rightful place amongst all available programming languages, and is acknowledged to be a superb tool for data manipulation and function definition.

APL2. A Programming Language, version 2. The latest version of APL, now available on IBM mainframes. To the original glittering brainchild of Dr. Iverson has been added a host of new facilities, making APL2 even better than APL. These include:

a. the ability to call external packages such as Fortran programs; graphics routines; SQL and DB2 databases, etc.;

b. error trapping;

c. "arrays of arrays", in which any element of a normal APL multi-dimensional array can now be another multi-dimensional array in its own right.

JANET. Joint Academic NETwork. The computer network that joins all British universities, and many polytechnics and other institutions of further education as well. JANET has recently been connected to EARN (European Academic and Research Network), a "super network" which connects academic networks in most European countries, while EARN in turn is connected to BITNET, the academic network used in North America and Eastern Asia.

Thus, through JANET, a British academic or researcher can communicate with, and send files to, fellow academics and researchers in the rest of the world.

THE PROBLEM

Many British academics and researchers have been keen to try APL2 but have been unable to do so because of the cost involved.

THE SOLUTION

This problem has now been overcome by IBM Academic Programmes Department (an Educational Trust) making APL2 freely available to Warwick University, one of the considerations being that the University then make APL2 freely available to selected academics and researchers over JANET.

This is now available.

If YOU are an academic or researcher who would like to try APL2 over JANET, this is your opportunity. An Open Day was arranged to tell interested academics about the service and what is available on 18th April 1988, at Warwick University, including speakers from the University and IBM, with lunch provided courtesy of IBM. For further information contact Mr. Mike Hunt at Warwick University, see below.

FURTHER INFORMATION SOURCES

APL2. First, the IBM Technical Co-ordinator in your university or polytechnic. Second, your local IBM representative. Third, Paul Barnetson at IBM United Kingdom Ltd., PO Box 41, Northern Road, Portsmouth, Hants. PO6 3AU.

JANET. First, the computing department in your university or polytechnic. Second, Dr. Paul Bryant at the Rutherford Appleton Laboratories.

Warwick University. Mr. M. Hunt, Director of the Computer Unit, University of Warwick, Coventry CV4 7AL.

IBM's Programme of Academic Donations. Dr. John Axford, IBM United Kingdom Ltd., Sheridan House, 41-43 Jewry Street, Winchester, Hants. SO23 8RZ.

IBM's Products for Academics and Researchers. First, the IBM Technical Co-ordinator in your university or polytechnic. Second, your local IBM representative. Third, Mr. Brian Whitaker, Academic Systems Marketing, IBM United KIngdom Ltd., Unit 125, Cambridge Science Park, Milton Road, Cambridge CB4 4BH.

Yours sincerely,

Paul Barnetson
Advisory Academic Systems Specialist
IBM United Kingdom Ltd.
PO Box 41, Northern Road
Portsmouth, Hants. PO6 3AU.

British APL Association News

by Graham Parkhouse

BAA Gives Further Support to I-APL

A further £2,000 has been given to I-APL, the project which is producing the free APL interpreter for school and home computers. This was agreed at the BAA committee meeting of 25th February 1988 and brings the BAA's total contribution to I-APL to £8,000. The meeting had been arranged especially to consider I-APL Ltd's application, made in January, for a further £5,000. All the committee were present except for Dave Ziemann.

Anthony Camacho, "Old World" chairman of I-APL Ltd and former Secretary of the BAA, came to the meeting to describe progress to date and to present his case. The meeting heard how I-APL was becoming a success story, how the mailing list was 1000 long and growing at 40 to 50 per week. Distribution was already being cared for by national organisations in six countries.

David Eastwood, the BAA's projects officer, was in the chair. He dismissed a suggestion made by Anthony that the BAA had not always been wholeheartedly behind I-APL; he said that surely we must presume this cannot be the case in view of the £6,000 already given. Norman Thomson proposed the motion that "the committee approves of the progress of the I-APL project so far and supports its current aims", and this was carried unanimously.

Anthony said he needed and welcomed help, and the meeting considered the possibility of the BAA taking responsibility for marketing and distribution of I-APL in Britain. If the BAA were to take this on, I-APL Ltd would be free to manage international co-ordination of the project and to maintain and extend the written material associated with it. Marketing was discussed at some length. The difficulties of introducing APL to education could not be over-estimated, it was said. It might be sensible to employ a company already experienced in marketing educational software, who already had good contacts with the decision makers in the educational system.

Mel Chapman and Norman Thomson both emphasized the importance of getting APL to be accepted as a language that may be used for answering examination questions, because until knowledge of APL counted towards examination results it could only appeal to enthusiasts. Mel pointed out that school computing as currently taught was regarded as a hurdle in higher education; lecturers in computer science at universities and polytechnics universally wish their students had never been taught what they now tend to learn about computers in schools. Summing up, David Eastwood said it looked

like the BAA were offering to do the marketing and Anthony Camacho offered to help the BAA in every way possible to do it effectively.

Concerning financial help, Anthony explained that I-APL Ltd's overdraft stood at £2,000, which was incurred by the recent printing of the encyclopaedias, tutorials and the manuals for the PC version. While there was already income from their sale, there were manuals for the BBC and CBM 64 to be printed at £500 per set, as well as on-going expenses. Each mail-shot was now costing about £300. To put these figures in context, he referred the committee to the notes of the AGM of I-APL Ltd, held at Sydney during APL 88, which gave the total donations collected as £21,950, of which £20,700 had been spent on programming.

Mel Chapman suggested that the BAA should donate a further £2,000 to pay off the overdraft and that David Eastwood should prepare a recommendation for what additional help should be given to I-APL Ltd, regarding existing and near-future financial needs, and regarding getting I-APL into schools. These suggestions were put to the vote. That David should prepare a recommendation was passed unanimously, that a donation of £2,000 should be made straight away was passed 6 to 1. Anthony left the meeting with a cheque.

Graham Parkhouse,
Hon. Sec. BAA,
Dept. of Mechanical Engineering,
University of Surrey,
GUILDFORD GU2 5XH.

News from Sustaining Members

Compiled by Cathy Dargue

APL People Ltd

Due to growing demand among our customers, we are pleased to announce that we are now expanding our specialist recruitment agency to cater for the placement of C programmers in addition to those with APL skills.

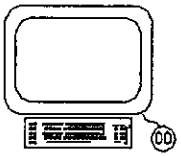
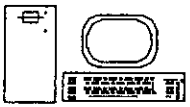
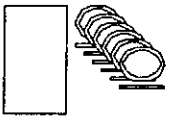
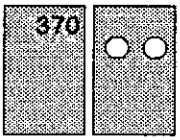
Over the past year or so, more and more of our customers have begun to ask us if we had anybody on our books with C and APL or just with C alone, and so we became aware that there was a growing need for people with C skills which was not being satisfied. In this time, we also took on a C programmer of our own to work on the software being developed by one of our sister companies (Applied Production Logic Limited), so we know that C is the type of language which fits nicely with APL and is often found in the same sort of environment. This led us inexorably to the conclusion that we should expand our business from being solely APL orientated to include the placement of C staff.

In many ways, placing people with APL skills is similar to placing those with C, in that demand for suitably experienced and qualified people outstrips supply. As many companies have found to their cost, advertising has little or no effect. The way we have cornered the APL market, and the way we intend to do it with C is through our extensive contacts in and knowledge of the market. Most people involved in APL have heard of us and know what we do. They also know that we provide an individual service, quite unlike that of other, bigger agencies. We are always happy to spend time with people advising them on their careers, even if they are not actively looking for another job at that moment.

We are confident that we can achieve the same success with placing C programmers as we have with APL, by building up a reputation for providing sound, reliable advice about the suitability of jobs, and by building up a good knowledge of the market and an excellent reputation as an agency to be trusted. People know that we will do our utmost to match them with jobs which really suit them.

As with APL, we will also be providing consultants in C - we already have a few C contract programmers on our books. Currently, we are working on a number of projects in APL, including further extensions of the IPLS software we are developing in conjunction with British Aerospace, and an exciting new challenge assisting one of the UK's biggest furniture retailers in finding the optimum locations for new stores.

A single source for APL*PLUS

APL*PLUS PC		for PCs and the PS2
APL*PLUS UNX		for UNIX systems
APL*PLUS PRO		for the PS2 model 80
APL*PLUS VMS		for the VAX range
APL*PLUS VM & MVS		for IBM mainframes

COCKING & DRURY LTD.

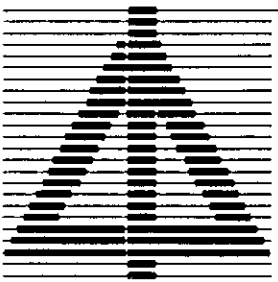
THE APL PROFESSIONALS

180 Tottenham Court Road,
London W1P 9LE
Tel: 01 – 436 9481

Course Schedule

Statgraphics	6 Apr – 7 Apr	£280
APL Fundamentals	12 Apr – 14 Apr	£445
APL Fundamentals	26 Apr – 28 Apr	£445
APL System Design	9 May – 12 May	£700
APL Fundamentals	17 May – 19 May	£445
APL Fundamentals	7 Jun – 9 Jun	£445
Statgraphics	22 Jun – 23 Jun	£280
APL Fundamentals	28 Jun – 30 Jun	£445
APL System Design	4 Jul – 7 Jul	£700
APL Fundamentals	12 Jul – 14 Jul	£445
APL Fundamentals	26 Jul – 28 Jul	£445

All prices exclude VAT. To book, or for further information, please call Beverley Satterley at:



COCKING & DRURY LTD.

THE APL PROFESSIONALS

180 Tottenham Court Road,
London W1P 9LE
Tel: 01 – 436 9481

Dyadic

Dyadic is pleased to announce Dyalog APL Version 5.1. This is a major new release of Dyalog APL, and contains a number of important enhancements. Foremost among these is the provision of user-definable input/output translate tables. These are designed to support National Language Character sets, to make it easy for users to configure new devices, and to allow a 'soft' keyboard. With this facility, users are even able to define their own atomic vector for special applications. Initially Dyalog APL Version 5.1 is available on the IBM 6150, but it will be ported to all the other systems supported by Dyalog APL in the next few months.

The full-screen data manager, which is modelled on AP126, has also been extended. Several new field types have been added, the number of function keys has been increased to 255, and colour terminals are now fully supported.

Following the delay in SCO CGI, Dyadic expects to be shipping its graphics auxiliary processor with Dyalog APL/386 by early April. CGI provides very powerful graphics capabilities, which coupled with a nested array APL, offer all sorts of new and exciting possibilities.

Dyadic is pleased to announce that as of February 1st, Oasis B.V. has taken over as the exclusive Dyalog distributor in the Netherlands.

MicroAPL Ltd

MicroAPL is now releasing Version 7 of APL.68000 on each of the machines it supports. The new version of APL.68000 continues MicroAPL's policy of progressive enhancements and optimizations. On the speed front, the new release includes substantial improvements to mixed monadic and dyadic functions such as rho and catenate, and specific optimization of some commonly used expressions with integer and boolean arguments. New features include improved built-in function/variable editor, and new Quad functions to ease migration from APL*PLUS/PC and APL2 environments to APL.68000.

The new interpreter is being issued initially on the Mirage and Macintosh versions, and other implementations will be issued over the next few months. At the same time as enhancing the APL interpreter itself, we are also making significant improvements to the machine-specific portions of each version. The new Macintosh product shows major enhancements including improved multi-window output handling, user-definable event control, up to four simultaneous function edit windows, full support of vertical and horizontal scroll bars, improved terminal emulator (running in its own window simultaneously with local APL operations), and support for MultiFinder. In addition, a

separate version specifically for the 68020-based Mac II is being released, with highly optimized in-line support of the 68881 floating-point processor.

In the UK, MicroAPL is strengthening its applications consultancy department. Projects are undertaken in all major APLs, especially APL*PLUS/PC, VSAPL, APL2 and APL.68000, and the range of application fields covered is increasing all the time.

APL Impetus Ltd

Although APL Impetus Ltd was formed only in July 1987, when it was decided that 'Impetus' no longer fitted into Boeing's product strategy, the product has been used by customers for 18 months and the concept goes back to 1976 when TABAPL, its mainframe timesharing predecessor, was first marketed.

When the Impetus development was started in 1984, the challenge was to provide all the functionality of the mainframe product in a machine with only 640K memory. The successful result is based on APL*PLUS/PC, enhanced by 30 or so assembler routines for such crucial activities as formatting, scrolling and panning, and displaying without snow on colour and shaded monitors.

Demonstration diskettes are available for consultants and users who would like to know more about this alternative, APL-based, approach to solving those complex modelling problems which are beyond the scope of spreadsheets.

Mercia Software Ltd

It's been another busy quarter at Mercia concentrating on our industrial APLications. Installations of LOGOL now stand at 4 and work is continuing on a number of programming projects for local industry - ranging from lubricants to paper and quality control to product data bases.

STSC products are still best sellers with STATGRAPHICS way out in front - surely the most used piece of APL based software in the world today.

For the APL Purist, STSC will soon be launching the new 80386 based interpreter, which will see the end of 'PC' limitations such as workspace size (providing you can get your hands on the RAM during the great world shortage). Extra features include nested arrays and 1-bit booleans which should please all the PC programmers out there - they've been No. 1 on the list for the last 5 years!

Cocking and Drury Ltd

We are pleased to announce the completion of the design phase of our DB2 interface for the APL*PLUS mainframe interpreter, and are hoping to start implementation soon. The project is likely to involve about one and a half work-years of effort in total. Although the development is based on the MVS environment, only one module will need to be replaced in order to produce a version for use with SQL/DS under VM. Little more would be required to adapt it to any database that has an SQL interface, and thus the work could form the basis of a variety of products. A technical article on the design has been submitted to VECTOR, and will appear in VECTOR 5.1.

We have acquired our first copy of PC Focus, with a view to developing a number of hybrid systems. Cocking and Drury are currently helping clients to migrate their applications between APL and Focus.

Cocking and Drury have developed a series of short seminars related to APL which are available for presentation to clients. Topics currently include; APL and Efficiency, APL and C, APL and SQL, APL and Spreadsheets, and a Management appreciation of APL (an empty array joke?). Please contact us if you are interested in any of these topics.

Our involvement in the European APL market is increasing. Not only are we running more in-house APL2 courses, but we are also doing more APL consulting work. In addition, there has been a recent APL*PLUS Compiler purchase in Scandinavia, and APL*PLUS mainframe software is being considered in other sites in Europe.

On the product side, we have had a number of successful Compiler installations, showing excellent financial and performance benefits in each case. Demand for the APL*PLUS mainframe system (enhancements and sharefile) continues. Release 6 of APL*PLUS mainframe includes the new right-bracket command processor; an extremely useful facility which allows you to invoke a non workspace-resident APL program merely by preceding its name by a ']', or by supplying it as a right argument to a new system function. This tool is used to provide programmer services such as workspace searching without the usual baggage of APL functions.

We are currently testing the new APL*PLUS interpreter for the new range of 80386 based machines, including the IBM PS/2 model 80 and the Compaq 386. The new system uses full 32-bit addressing, allowing workspace and variable sizes that are limited only by the installed memory. We are already receiving a number of enquiries.

Statgraphics and APL*PLUS/PC sales continue to flourish, in particular demand for upgrades to APL*PLUS/PC Release 7 has been heavy, mainly due to the extended memory features and the function monitoring facilities.

HMW Computing Ltd

HMW Computing Limited is now in its sixth trading year and has been involved in consultancy and bespoke systems work since its inception. The systems designed have usually had a financial flavour and it is without surprise that we now find our market is 90 percent within the banking and finance community. It is worth noting (and perhaps this will be removed by the editor) that our Jackson Structured Programming team has recently left the city for a greenfield banking site. COBOL to our company is not a word we joke about???

HMW has also moved a jump ahead, and this year, every employee has a machine at home (from XEN's for the power crazed to pc's for the unfortunate few). This in theory should have made timesheet and expenses forms return to the office on time but instead has lead to increased floppy disc flow of useful 'goodies' through the post.

Marc Griffiths has recently rejoined us after spending 3 years in the academic world gaining an MSc., teaching and working on an Esprit OA project. He will continue with his PhD and augment our all consuming team of banking specialists and system developers.

This year sees the launch of 4XTRA our front end dealing system for the Banking community. 4XTRA comprises a combination of software packages, for example, FX Dealing and Position Keeping, Arbitrage calculators, Basket Currency Calculators. The system is flexible and designed to be tailored easily. For more information please call Stan Wilkinson on 01-286-7068.

MicroAPL for ...

APL on the widest range of micros

PCs

MultiAPL - APL.68000 under PCDOS

Sample configurations:

68000/1MB	£ 995
68000/1MB/Maths processor	£1295
68020/1MB/Maths processor	£2995
68020/4MB/Maths processor	£5495

Prices include: Plug-in card, APL.68000, Keyboard stickers, Manual, STSC emulation software

STSC's APL*PLUS £450

Single user micros

APL.68000 for the Apple Macintosh
APL.68000 for the Atari ST — All versions £86.91
APL.68000 for the Commodore Amiga

Prices include: APL.68000, Keyboard stickers, Manual, Reference card

Multi-User systems

APL.68000 runs under the following operating systems:
Unix, CP/M68K, Wicat MCS, Mirage
on computers which include:
Sun, NCR, Wicat, HP200/300, Spectrum, Aurora, Stride
Prices range from £1200 to £2000

All prices quoted exclude VAT

MICRO APL

MicroAPL Ltd.
South Bank Technopark
LONDON, SE1 6LN
Telephone: 01 922 8866

Access and Barclaycard Orders accepted

The Education Vector

by Norman Thomson

The health and progress of APL and of the British APL Association depend on achieving growth within education. From the other side of the picture the case which must be put before educationalists is that there is no language which can claim such strong educational merit in its own right as APL can.

Movement into education should be seen in the broader perspective of APL's niche-finding progress in the computing world at large. The following is an extract from a letter from a consultant written to the San Francisco Area APL Users Group:

> "Market acceptance in the larger corporate settings (outside actuarial departments) has not been wonderful. Although there are companies with solid products which use APL as a base and which are continuing to grow, they are for the most part smaller operations."

Here is another quote from the same source, this time from a meeting report:

> "APL provides limited access to machine specific functions. For all its power and utility, these two limitations mean that some things cannot be coded in APL, and others crawl."

Although these items appear to herald the demise or at least contraction of APL, arguably they point to an entirely healthy situation. When computers meant mainframe, performance scarcely mattered in choosing between languages. Now, however, that the small PC has become the most popular point of computer contact the difference between a fast system and a slow one really counts.

In the first flush of APL enthusiasm all fields of computer activity were game for the APL programmer. For APL, maturity consists of abandoning those fields in which its performance is worse or no better than other languages, and concentrating on those areas in which it is unsurpassed. PCs have hastened thereby the process of focussing APL on its regions of excellence. If C makes the screen image move faster, then let us use C to move the screen image.

The words in parentheses above are significant. Californian actuaries are clearly in no doubt about the excellence of APL on its own territory, and so too are (some) statisticians, numerical analysts, accountants, and others whose professional business is to handle numbers in the mass. The core of such applications demands that calculating power per statement is a more important consideration than performance. Now ask in what other field this is true. The answer of course is education, not just in the subjects listed above, although these score a double bonus, but education in a wide range of subjects, for which

the computer only has value as a teaching aid if the calculating power per statement is high, and performance is a minor consideration. Absence of computer jargon too is important for teachers - in what systems other than APL can freedom be guaranteed from the profitless wrestle with system technicalities from which none but the brightest hackers are immune.

For teachers, it all adds up to APL, and I-APL gives an opportunity to add education to the list of professions which APL is best suited to serve. The other professions in this list have discovered it, at least in part, and many practitioners cannot afford to be without it - education is a larger field than any and now can afford to have it.

In the UK, there are plans to hold two courses this summer, one at King Alfred's College in Winchester which was last year's venue, and the other at University College, Swansea. In both cases, I-APL will be used and a copy of the interpreter given to participants. Instruction at Swansea will be on old IBM PCs, whereas at Winchester it will be on BBC micros. We expect therefore that the Winchester course will appeal predominantly to school teachers, and the Swansea one to teachers in higher education. In airline parlance, we have doubled the capacity on the route and we hope that the response will be that the traffic will also double!

Of course a leisure weekend is a precious commodity, and BAA members can help by persuading their teacher friends that such a sacrifice will be richly rewarded. The target is to have APL securely based at a selection of schools, universities and polytechnics, not primarily for mundane reasons of increasing employment prospects or qualifications, but because of its power through the computer to enhance understanding, appreciation, and enjoyment of subjects to which it can naturally be applied.

I-APL: Report of 1st Annual General Meeting

by Anthony Camacho

The AGM of I-APL Ltd was held in public in the school of Architecture in the Wilkinson building, Sydney University, Australia on 2nd February 1988. Directors present were: A J Camacho, H A Peelle, D M Ziemann, S M Camacho, P G H Chapman and L Alvord. Apologies were received from E M Cherlin. There were several interested spectators.

Report

1. As this is the first AGM there were no minutes to be read.

2. The company was incorporated on 30 September 1986. The articles of association were available for inspection there and until 8 Feb in room 3224 of International House.

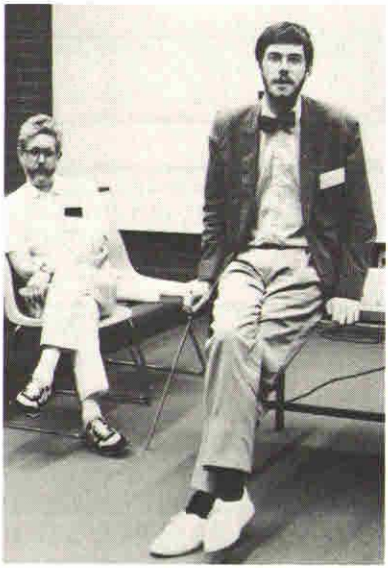
3. The draft accounts were presented to 31 January 1988. It is intended to have the company's financial year end on 31 March 1988 which is why the accounts are not final. The total of donations is £21,951.57 of which £20,700 has been spent on programming and £825.81 on disks, printing, bank charges etc. There are outstanding cheques for £2,920 to pay for printing the I-APL Encyclopedia, Tutorial and Instruction Manual and there are several hundred pounds of expenses which project members will claim and some minor items of freight and mailing to pay for. Overall the project will be about £1500 overdrawn by Feb 29 and when/if the VAT on the programming can be reclaimed, £1,200 in credit. Products have been priced so that we will recover their cost when two thirds of the stock is sold.

The Chairman's report was as follows:

I am glad to report that the project has had a successful 17 months. Paul Chapman's I-APL/PC port is now available with an instruction manual, a Tutorial thanks to Norman Thomson and Linda Alvord and an APL Encyclopedia thanks to Garry Helzer. Ports to several other machines are well advanced; in particular:

R C Piccoline	Kim Andreasen
BBC 'B' & Master	Tony Cheal
Acorn Archimedes	Tony Cheal
Sinclair QL	Tony Cheal
Amiga	David Angier
CBM 64	Andrew Baillie
Microbee	Ian Shannon
CP/M (generic Z80)	Paul Woods
Unix	Paul Chapman

We do not have recent details of the status of the ports to the other machines.



Anthony Camacho and Paul Chapman at the AGM of I-APL

We have at least two more books in course of preparation (by Tama Traberman edited by Norman Thomson and by Linda Alvord). We need help with porting and promoting APL to educational institutions.

It was recorded that Norman Thomson had resigned from the board of I-APL as soon as there was any danger of a conflict of interest with his terms of employment at IBM and that Paul Chapman and Linda Alvord had agreed to join the board.

The report, the provisional accounts and the changes to the board were accepted nem. con.

APL in a Japanese High School

by Makoto Kikkawa

Makoto Kikkawa arrived in Sydney with some superb material to illustrate her paper; she used some of this to accompany the talk she gave to the teachers on Education Day. The following is an edited version of her printed notes: it is a pity that current technology is still not cheap enough for us to reproduce her slides and colour printed overheads. They were of magnificent quality, but they also illustrated in the most direct way that children can produce inspired work if competently taught and given flexible tools.

APL in a Japanese High School

Good afternoon, ladies and gentlemen, I am very happy to be with you here to tell you what my fellow teachers and I are doing in using APL at Shokei Jogakuin Senior High School in Japan.

First of all, I would like to tell you about the current situation in the use of APL in Japanese high schools. I must confess that it is a very poor situation. 93% of Japanese high schools own computers of some sort, but 51% of these schools own only one computer. Among them as much as 79% are using only BASIC machines. Only half of the computer-equipped high schools use the system for subject instruction. There are some disturbing factors in the use of PCs for subject instruction. The biggest problem lies with the teachers.

In Japan only Miyagi Prefecture high schools use APL for math instruction and are trying to develop educational materials for APL-assisted math classes. 44 math teachers from 33 schools out of 106 in Miyagi Prefecture established the Computer Study Group of the Society of Mathematics Education in 1970. They have a meeting once a month to study APL and exchange information to develop educational materials in APL. Their contribution includes making text books such as "APL Text" and "Computer Assisted Mathematics: Elementary Analysis" and the presentation of papers to the parent Society.

Shokei Jogakuin is one of the most active members of this group. The school began offering math classes using computers in both regular and elective courses in the early 1970s. In 1982 we shifted the system from FORTRAN to APL. Since then we have been working on the system very effectively, especially in using APL graphics. This was the turning point in the use of computer education in Shokei. Currently 15 personal computers are used with a ratio of 3 students per machine in regular classes, and a ratio of 1 to 1 in elective classes. At present, teachers are working on a formula manipulation system with APL graphics for their classes.

School Year	All Mathematics Courses			Computer Related Education		
	Required Subject	:Elective :Subject	hrs/ week	Contents	Textbook	hrs/ year
1st year (Ages: 15-16)	Math 1	: : : :	5	Introduction; Visit Computer center	0-1: Illustrated Introductory Computer Guide	1 4 (half day)
2nd year (Ages: 16-17)	Elementary Analysis	: : : :	4	Using graphics to teach math concept	Computer Assisted Mathematics: Elementary Analysis	20
	Math 2	:	4		None	est.
		:Basic Math	2		None	0
3rd year (Ages: 17-18)	Algebra and Geometry	: : :	2	Using graphics for linear transformation	None	5 (est.)
	Math 2	:	2	Statistics	None	8
		:Differential :Integral Calculus	3		None	0
		:Math 1	3		None	0
		:Basic Math	3		None	0
		: :The Computer	2	Learning APL and Drawing graphics	APL TEXT Graphics In APL	70

Curriculum Table

The curriculum is divided into 3 major fields for the use of APL in the classroom.

1) An introduction to computers for all freshmen.

2) Using APL in math classes.

3) Using computer graphics by both teachers and students.

Every year before the summer vacation we have given our "Introduction to Computers" seminar to all freshmen. They are introduced to the computer system at our school, watch the video tape "Computer", talk about the textbook "0 and 1: An Illustrated Introductory Computer Guide" published by Shokei, and see computer demonstrations. During the summer vacation teachers take interested students and their parents to the Computer Centre of Miyagi Prefecture.

I will show you what we are doing during these introductory lessons. Using the iota function we show them the speed and memory of the computer, which is characteristic of

computers in general. Using the domino function we show them how we can solve simultaneous equations with the computer.

```
      ⍳100
1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
     30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
     54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77
     78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

      A←⍳100
      A×A
1  4  9 16 25 36 49 64 81 100 121 144 169 196 225 256 289 324 361 400 441 484 529
     576 625 676 729 784 841 900 961 1024 1089 1156 1225 1296 1369 1444 1521
     1600 1681 1764 1849 1936 2025 2116 2209 2304 2401 2500 2601 2704 2809
     2916 3025 3136 3249 3364 3481 3600 3721 3844 3969 4096 4225 4356 4489
     4624 4761 4900 5041 5184 5329 5476 5625 5776 5929 6084 6241 6400 6561
     6724 6889 7056 7225 7396 7569 7744 7921 8100 8281 8464 8649 8836 9025
     9216 9409 9604 9801 10000

      +/A
5050
      +/A×A
338350
```

2) Using ⌸ (domino) function

(1) $2x-y=-1$, $2x+3y=12$

$x=1$, $y=2$

```
(1)   A←2 2⍴3 2 1 4
      A
3 2
1 4
      B←7 9

      B⌹A
1 2
```

(2) $3x+2y=7$, $x+4y=9$

$x=\frac{9}{8}=1.125$, $y=\frac{13}{4}=3.25$

```
(2)   A←2 2⍴2 ¯1 2 3
      A
 2 ¯1
 2  3
      B←¯1 12

      B⌹A
1.125 3.25
```

(3) $3x+2y-z=20$, $x+y-2z=4$, $4x-2y-3z=14$

$x=6$, $y=2$, $z=2$

```
(3)   A←3 3⍴3 2 ¯1 1 1 ¯2 4 ¯2 ¯3
      A
 3  2 ¯1
 1  1 ¯2
 4 ¯2 ¯3
      B←20 4 14

      B⌹A
6 2 2
```

Students' responses are very quick. When we presented such a demonstration one of the students said "Why do you give us these problems when the computer can solve them?" At that time she was having a hard time solving the same problems by herself in the math lesson. Another student remarked "That computer is an impertinent fellow even though it's no human".

When using APL in math classes we teach them only the few APL symbols necessary to solve problems. For example we use APL for a 3rd year statistics class involving 4 class

periods; it is a study of the heights of students. The 1st period is used for naming data, input and assignment to memory. Then the data is expressed as a matrix and is sorted. In the 2nd period the students obtain data characteristics such as: number of subjects, maximum, range, sum total, mean, standard deviation. In the 3rd period they study how to make random numbers and how to make a sample of student heights using random numbers. The 4th period is dedicated to getting the frequency distribution program and using it, the students getting the frequency distribution of actual heights and of heights simulated with random numbers. For these lessons the students have to learn only 7 symbols.

In the 3rd year in the elective "The Computer" class, APL itself is taught. Teachers use a text "APL Text" which was published in 1980 by the Computer Study group.

While teaching APL we have found some problems. First, in APL the order of execution is from right to left. This feature is completely different from the order of calculation with which the students have been familiar since their childhood. Accordingly, if teachers do not clearly tell students that the execution order is based upon the idea of composite functions they often make mistakes such as:

```
3 x 4 - 5   >>>>   (5 - 4) x 3
```

Secondly APL employs unusual notations. I will show you an example. Matrix product is expressed:

```
AB   >>>>   A+.xB
```

but AB is not a matrix product in APL, A x B is the product of respective elements.

Third is index origin, which should be taught to be 1 in the same way as the origin of natural numbers. However, when the teachers say this origin is changeable some students think it is changeable to numbers other than 1 or 0. Accordingly,

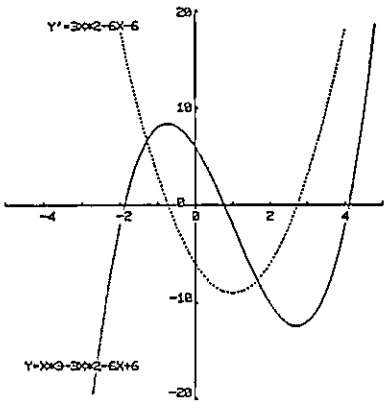
```
5 6 7 8 .....15   >>>>   ⎕IO←5
```

and then try iota 11.

The most important use of computers in our school is APL graphics for teaching mathematics. We have set our computer beside a blackboard and the graphics are displayed on a 26-inch TV screen. At any time teachers can illustrate problems from elementary analysis or algebraic geometry. The advantages are great: static math concepts become dynamic images and theoretical expressions become real pictures. Linear transformations of matrix data used to involve so many calculations that only the simplest examples could be used; but the computer can display linear transformations quickly and easily. As a result, students are able to understand difficult problems and

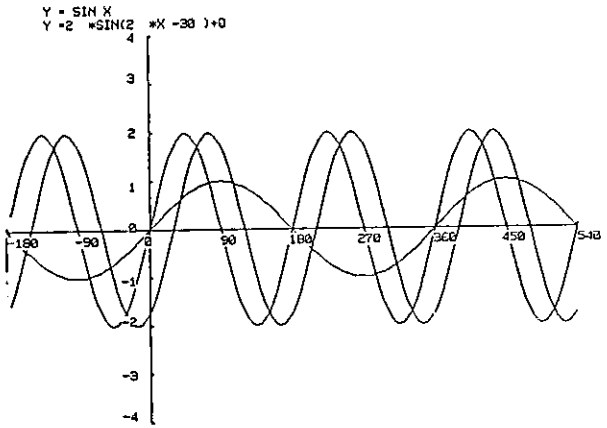
concepts unexpectedly fast when using APL graphics. In other words, APL has great value as a tool for displaying math concepts.

Here is a graph of $Y = X^3 - 3X^2 - 6X + 6$, and its derivative $Y' = 3X^2 - 6X - 6$

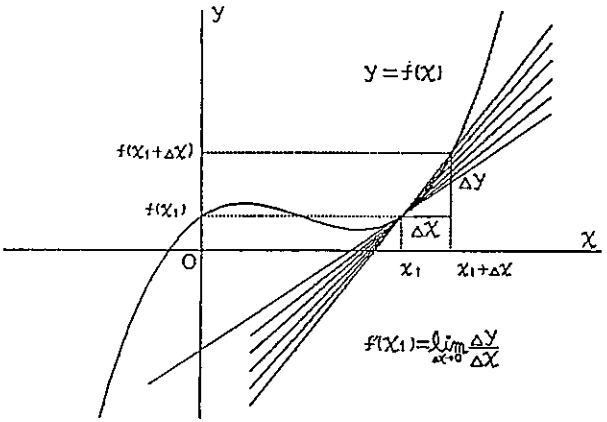


This shows parallel movement of a sine curve:

Y = sin X, Y + 2 sin 2X and Y = 2 sin (2X - 30degrees).

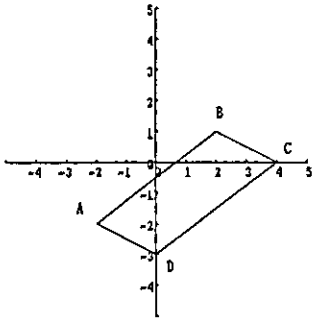


This explains the concept of a tangential line. For explaining differential calculus, the concept of a tangential line can be most effectively explained by the continuous movement of the lines.



After several such lessons, students expressed the wish to do the same kind of graphic work. The teachers realized that this would be of great help in studying math. The students' creation of their own graphics is certainly a good exercise in the algebra and geometry that they have already learned in the classroom. It is mainly in "The Computer" class that students start learning APL and then doing their own graphics work.

With the help of fellow teachers I wrote "Graphics in APL" for students to use in working on graphics. This is a hand-written text of 50 pages specially designed for personal computers (IBM JX). Students learn how to use standard commands or standard methods such as movement of patterns, transformation of patterns, animation and so on.



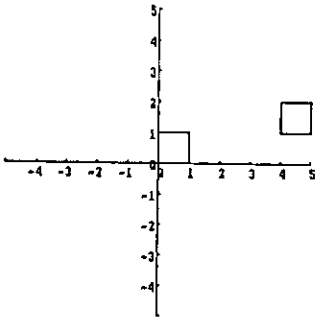

```
      A  B  C  D  A
D←5 2⍴¯2 ¯2 2 1 4 0 0 ¯3 ¯2 ¯2

      D ←
 ¯2 ¯2  ←--- A
  2  1  ←--- B
  4  0  ←--- C
  0 ¯3  ←--- D
 ¯2 ¯2  ←--- A
```

(data matrix)

```
DRAW D ←---- Execution
```

For example, in making a pattern I first tell them to draw the desired pattern on graph paper to find all the pairs of co-ordinates of its points. Then, using the rho function, the pattern can be described in an n x 2 matrix. The numbers on the right show the input data used here.

Movement of Pattern



```
V←5 2⍴0 0 1 0 1 1 0 1 0 0

      V
0 0
1 0
1 1
0 1
0 0

      ⍴V
5 2
```

Value of movement

x ----> 4, y ----> 1

```
      V                (⍴V)⍴4 1
0 0       4 1
1 0       4 1
1 1   +   4 1
0 1       4 1
0 0       4 1

      V←(⍴V)⍴4 1
4 1
5 1
5 2
4 2
4 1
```

For the movement of patterns, data such as this is used. By adding the desired values of movement to the first and second columns we can easily achieve the task.

For continuous movement, we must use looping algorithms, or the function definition said to be defined "recursively". For students there seems no difference between the two methods: they can use both techniques with ease.

Transformation is achieved by the following technique:

$$\begin{cases} x' = ax + by \\ y' = cx + dy \end{cases} \qquad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

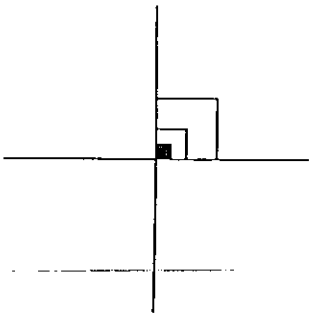
For convenience we rewrite them as follows:

$$(x,y) \begin{pmatrix} a & c \\ b & d \end{pmatrix} = (ax + by,\ cx + dy) = (x',y')$$

Then we express it as:

(n x 2 data matrix) +.x (2 x 2 matrix of linear transformation)

Similarity (Expansion, Contraction)



```
      DRAV V
      DRAV V+.×HO

        V
0 0
1 0
1 1
0 1
0 0

        HO
1 0
0 1

DRAV V+.×HO×2
          2 expansion
FILL V+.×HO×0.5
          1/2 contraction
```

Similarity, expansion and contraction, symmetry, rotation and shape-change are done by using the same techniques.

Using such techniques students did really splendid work. Here are examples of teachers' demonstrations and work done by 3rd year students in "The Computer" classes. [Here Makoto showed slides of some delightful pictures created by her students.]

To conclude my talk I would like to offer some practical advice for all of you who are interested in APL-assisted math classes.

Teaching Math through graphics:

a) Viewing speed should be harmonised with the students' rate of understanding and thinking.

b) Let them take notes or draw graphs by hand. Then visual understanding becomes long-term comprehension.

c) When giving assignments, specify the technique to be used for them and if need be the formula to be used. Evaluation of the work done by the students should be limited to whether or not the required technique is well used. This is math instruction, not fine art class.

As to school facilities, we must have a good computer system with many functions and capabilities. This is the most critical requirement of a computer-assisted class. APL is the best, I think, but unfortunately it is very expensive. It is hard to fit into the school budget. I hope the vendors will think this over and do something for teachers and students.

Thank you.

APL Product Guide

Compiled by Cathy Dargue

VECTOR's exclusive APL Product Guide aims to provide readers with useful information about sources of APL hardware, software and services. We welcome any comments readers may have on its usefulness and any suggestions for improvements.

We do depend on the alacrity of suppliers to keep us informed about their products so that we can update the Guide for each issue of VECTOR. Any suppliers who are not included in the Guide should contact me to get their free entry - see address below.

We reserve the right to edit material supplied for reasons of space or to ensure a fair market coverage.

The listings are not restricted to UK companies and international suppliers are welcome to take advantage of these pages. Where no UK distributor has yet been appointed, the vendor should indicate whether this is imminent or whether approaches for representation by existing companies are welcomed.

For convenience to readers, the product list has been divided into the following groups:

* Complete APL Systems (Hardware & Software)
* APL Timesharing Services
* APL Interpreters
* APL Visual Display Units
* APL character set printers
* APL-based packages
* APL Consultancy
* APL Training Courses
* Other services
* Vendor addresses

Every effort has been made to avoid errors in these listings but no responsibility can be taken by the working group for mistakes or omissions.

Note: 'poa' indicates 'price on application'.

All contributions to the APL Product Guide should be sent to:

Cathy Dargue,
60 Downhall Ley,
BUNTINGFORD,
Herts SG9 9TL.
Tel: 0707-325161 ext 2418

COMPLETE APL SYSTEMS

COMPANY	PRODUCT	PRICES(£)	DETAILS
Analogic	The APL Machine	¤60,000+	AP500 array processor, 4 Mb data memory, 80 Mb disk drive.
Dyadic	APL Coprocessor	3,500+	32-bit coprocessor board for IBM PC. NS32000 cpu with FPP, up to 4Mb RAM, 16Mb virtual memory. Software includes Unix V.2, Dyalog APL, graphics support, DOS interface. Provides multi-user Unix/DOS environment.
	IBM 6150	15,000+	Multi-user Dyalog APL system with Fast 32-bit RISC processor, FPP, up to 8Mb RAM, 210Mb Disk, 16 users. Interface to SQL, graphics and APL support for standard IBM peripherals.
	Altos 3068	25,000+	Multi-user Dyalog APL system with MC68020 cpu & MC68881 FPP. Also features a LAN which supports IBM PCs as Dyalog APL terminals.
	Sun 3	15,000+	Multi-user Dyalog APL systems which can be configured as a network of workstations and or a traditional time-sharing cpu. With its 25MHZ 68020 cpu, the Sun 3/200 is the fastest APL microcomputer on the market.
Gen. Software	Myriade	poa	TI computer APL & APL operating system
Inner Product	IBM PC	2,000-6,000	IBM PCs supplied for turnkey applications
M.B.T.	MBT Series 10	poa	UNIX/68010 based multi-user APL system
	TORCH	poa	68000/Z80 multiprocessor
MetaTechnics	-	poa	Details on application -- IBM PC compatible
MicroAPL	Aurora	20,000+	Multi-user APL computer using 68020 CPU. Std. configuration 2Mb RAM, 16 RS232 ports, 68 Mb hard disc, 720K diskette
	SPECTRUM	7,000+	Expandable multi-user APL computer using Motorola 68000. Std. conguration 1 Mb RAM, 12/36 Mb disc, 12 ports.
	Atari 1040ST	799	1 Mb Mono/Colour System, includes 1 Mb disc drive & mains transformer built into Console.

APL TIMESHARING SERVICES

COMPANY	PRODUCT	PRICES(£)	DETAILS
Mercia	APL*PLUS	poa	STSC's Mainframe Service + MAILBOX etc.
I.P. Sharp	SHARP APL	poa	International Network application systems and public databases.
Uniware	APL*PLUS	call	STSC's mainframe service

APL INTERPRETERS

COMPANY	PRODUCT	PRICES(£)	DETAILS
APL Software	Dyalog APL	1,000-8,000	See Dyadic Systems entry
Cocking/Drury	APL*PLUS/PC Rel 7	475	STSC's full featured APL for IBM PC, PC/AT and PS/2. Now with virtual workspace, quad-MF, GSS graphics, Network and VGA support.
	Upgrade 6 to 7	poa	
	Upgrade 5 to 7	poa	Extension upgrades to release 7.
	Run-time	poa	Closed version of APL*PLUS/PC which prevents user exposure to APL.
	APL*PLUS UNX	poa	STSC's 2nd generation APL for IBM PC/AT, DEC, AT&T and other Unix computers.
	APL*PLUS VMS	poa	Integrated 2nd generation APL for DEC m/cs under VMS
	APL*PLUS mainframe	poa	Component files, quad-fns, nested arrays for VS APL under VM/CMS and MVS/TSO
Dyadic	Dyalog APL	795-10,000	2nd gen. APL for UNIX systems, e.g. IBM 6150, Sun, Vax, NCR, HP9000, AT&T, Altos, Apollo, Whitechapel, Sperry, etc.

Gen. Software	APL*MYRIADE	poa	Runs on Texas Instruments TI990 range.
IBM UK	IBM PC APL	poa	Event-handling & APs for full-screen I/O disks, diskettes, asynch. comms.
Inner Product	VIZ::APL	250-350	8-bit Zilog Z-80 CP/M
	APL*PLUS/PC	600	See under Cocking & Drury
M.B.T.	Dyalog APL	poa	See Dyadic Systems entry
	MBTAPL	poa	Enhanced Dyalog APL for MBT hardware.
	VIZ::APL	poa	Customized for TORCH hardware
Mercia	APL*PLUS/PC Rel 7	475	STSC's full-feature APL for IBM PC, and compatibles. No 64K object size limit.
	Upgrades 5 to 7	325	
	Upgrades 6 to 7	145	
	APL*PLUS/UNIX	poa	Interpreter for UNIX systems: WICAT, CADMUS, CALLAN, FORTUNE 32:16, HP, 9000/500, OLIVETTI 382, SUN etc.
	APL*PLUS/MAC	350	APL for the Macintosh. Big workspaces, big objects, mouse, icons etc
MetaTechnics	APL*PLUS Rel 6	475	Discount on quantity.
MicroAPL	APL.68000	1,000-2,000	Full implementation with component files, error trapping etc. for SPECTRUM, HP300, SUN, NCR etc.
	QL/APL (keyword)	87	Full keyword APL for QL with many extra features.
	QL/APL (APL chars)	87	VSAPL compatible APL for QL with many extra features.
	APL.68000 Apple Macintosh	87	
	APL.68000 Amiga	87	Full APL interpreters with support for windows, mouse, graphics etc.
	APL.68000 for Atari ST	87	
	APL.68000 for IBM-PC	995	
	APL*PLUS/PC - REL 7	450	
Portable	PortAPL	¤195	IBM PC Software
		¤275	Mackintosh
		¤2,995	DEC VAX
I.P. Sharp	Sharp APL/PCX	2,575	For IBM XT/AT
		1,000+	For IBM mainframes
	Sharp APL/PC	325	For IBM PC or PC/XT
Uniware	APL*PLUS/PC	495	STSC's full feature APL for IBM PC/XT/AT, Compaq, Olivetti
	Run-Time	call	Closed version of APL*PLUS/PC which prevents user exposure to APL
	APL*PLUS/UNX	call	STSC's full feature APL for UNIX based computers.
	PortaAPL	280	PORTABLE SOFTWARE's APL for APPLE MACINTOSH.
		2,995	PORTABLE SOFTWARE's APL for the DEC VAX.

APL VISUAL DISPLAY UNITS

COMPANY	PRODUCT	PRICES(£)	DETAILS
Dyadic	Lynwood j300	1,560	Monochrome ANSI 3.64 APL vdu, 15-inch high quality screen, Tek graphics, local macro keys.
	Lynwood j500	2,295	Colour ANSI 3.64 APL vdu, 15-inch high quality screen, Tek graphics, local macro keys.
	IBM 3163	791	Low-cost Monochrome APL vdu. Supports downloaded Dyalog APL font.
	IBM 3164	1,093	Low-cost Colour APL vdu. Supports downloaded Dyalog APL font.
Farnell	Tandberg TDV 2221	995	Ergonomic design APL terminal, 50-19200 baud, 15f8if5 anti-reex screen, low prole keyboard
	Tandberg TDV 2271	1,195	Combined APL/ANSI ergonomic terminal as above.
Gen. Software	Mellordata	400	Second-hand Elite 3045A
M.B.T.	various		Contact MBT for details
Meta Technics	IBM EGA compatible	299	Emulates EGA & Hercules, Half Card

MicroAPL	Insight VDT-1	795	Inexpensive APL VDU
	Insight GDT-1	1,450	With monochrome graphics
	Concept201	1,295	APL VDU with 8 page memory
	Concept 201G	1,650	Graphics VDU
Shandell	HDS2010	1,215	ANSI364 DEC VT52/100/220 compatible. 15" tilt/swivel screen, low profile keyboard 8 page memory, windows, viewports, 80/132 columns, full overstrike, 2 or 3 comms, ports, 55 PF keys, NVM storage.
	HDS2010G/GX	1495+	As above plus Tektronix 4014, Retrographics VT640/D0640 and Visual 500 compatible. 1024 x 390 or 1024 x 780 resolution.
Tektronix	4114B	13,500+	19" D.V.S.T.:Graphics: 3120 x 4096 displayable; Intelligent: up to 800K memory; APL keyboard (option 4E)
	4125	21,550+	19" 2D colour graphics; Workstation (1280 x 1024);Intelligent: up to 800K memory; APL keyboard (mod AP)
	4128	26,822+	As 4125 plus 3D wireframe

APL PRINTERS

COMPANY	PRODUCT	PRICES(£)	DETAILS
Datatrade	Datasouth DS180+	1,295	180 cps matrix printer with 4K buffer, 9 x 7 dot matrix and APL option.
	Datasouth DS220	1,695	Letter quality; graphics capability, APL option (both available with IBM Twinex or Coax interface).
Dyadic	IBM 4201 Proprinter	poa	100, 200, 40(nlq) cps, matrix printer, with graphics. Supports downloaded Dyalog APL font.
	Toshiba P351	poa	24 pin high-quality matrix printer 100 cps letter quality, 192 cps draft.
Inner Product	Epson FX80	500	Soft char. set, 160 cps, 80 column
	Anadex 9620	1,150	200 cps., 132 col., tractor feed
	Siemens PT88	620	180 cps., 80 col., silent
	TGC Sterwriter	1,180	40 cps., letter quality
M.B.T.	Facit 4565	poa	40 cps letter-quality
	Facit 4510/11/12	poa	Matrix printers
MetaTechnics	Quen-data	295	Low-cost APL Daisy-wheel printer
MicroAPL	Datasouth DS180+	1,295	See Datatrade entry
	Philips GP300	1,924	Matrix printer with letter & draft quality and APL.
	Qume Letterpro20	549	APL/ASCII Daisy-wheel printer

APL PACKAGES

COMPANY	PRODUCT	PRICES(£)	DETAILS
APL-385	APL-385	50(PC),125(mf)	including ...
	FSM-385		Screen development
	DRAW-385		Screen design
	DB-385		Relational W.S.
	GEN-385		Miscellaneous Utilities
APL Software Ltd (mainframe)	AFM/AP	11,035	Interprocess Software for VM/CMS & MVS/TSO.
	- Keyed Access	2,650	Component File Management System (VSAPL/APL2)
	- Interactive Link	1,325	
	- Mail Exchange	2,650	
	CALL/AP	4,030	Non-APL program execution (VSAPL/APL2)
	APLPRINT	2,205	Output to high speed line printer or 328x devices (VSAPL/APL2)

	ENHANCED FORMAT	2,205	Extends Format operator to full "Quad-FMT" status (VSAPL/APL2)
	ISP	750	Input and Output Stack Processors for manipulating terminal I/O
	OSP	2,205	with facilities for Error Trapping (VSAPL)
	DISPLAY CAPTURE	poa	Allows terminal output to be collected and held for retrieval by an APL function (APL2)
	UCF	poa	User Communication Facility for data transfer between users (APL2)
	RDS	poa	Relation Data Base System
	PANEL	poa	Fullscreen management system
	PFS	poa	Program File System - APL Systems development aid
	IPLS	poa	Project Management System
	REGGPAK	poa	Regression Analysis Package
(microcomputer)	POWERTOOLS	295	Assembler written replacement function for commonly used CPU-consuming APL functions, includes a Forms Processor.
	REGGPAK	poa	Regression Analysis Package
	RDS	990	Relational Database System
Beta-plan	BETA-FONT	poa	Multiple font PC character generator. Dealers required for non-Scandinavian countries.
APL IMPETUS	Impetus	poa	Hierarchical Planning System
Butel	Merlin	5,000	Mainframe APL spreadsheet runs under VM/CMS, TSO, VSPC
	Merlin/PC	poa	Version for APL*PLUS/PC
Cocking/Drury (for VSAPL)	E'MENTS & SHAREFILE	poa	Component files, quad- functions & nested arrays for IBM VSAPL under VM/CMS & MVS/TSO
	SHAREFILE only	poa	
	ENHANCEMENTS only	poa	
	COMPILER	poa	First APL compiler. Available with APL*PLUS enhancements and Sharefile under VM/CMS & MVS/TSO
	FILEPRINT	poa	Print APL component files
	FILESORT	poa	Sort APL component files
	FILECONVERT	poa	Convert non-APL files to APL files
	FILEMANAGER	poa	Extends APL primitives to database management
	TOOLS + UTILITIES	poa	APL Software development tools
	DATAPORT	poa	Powerful Information Centre spreadsheet incorporating data exchange between APL and FOCUS, IFPS, SAS, APL/DI, ADRSII, LOTUS123, VISICALC, MULTIPLAN, DIF files
(for APL2)	SHAREFILE/AP	poa	STSC's sharefile for APL2
	FMT	poa	Full featured FMT for APL2
	WSDOC	poa	
	FILEMANAGER	poa	
(microcomputer)	STATGRAPHICS Rel 2.6	645	Integrated Statistics and graphics on IBM PC, AT, PS/2 and compatibles
	Update 2 to 2.6	140	
	Update 1 to 2.6	330	
	APL*PLUS PC Tools	325	Utilities including: RAM disk, full screen data entry, menu input, report generation, file documentor, exception handling and games.
	IRMA Module	120	327x IRMA support.
	Fin & Stat. Library	350	Financial & statistical routines
	SPREADSHEET MGR	195	APL-based spreadsheet for APL*PLUS/PC. Cell arithmetic; transfers to ASCII, LOTUS
E&S	PROTOPAK		Packages for prototyping management information systems - consisting of: PC & mainframe modules
	RMS		Relational databases.
	AMS		Multi-dimensional arrays

	RAMS		Combined RMS & AMS.
	BMS		Dynamic nancial modelling & forecasting
	FMS		Full-screen handler for APL*PLUS/PC. (AP124-based)
	CMS		Communications package.
	SOS	poa	Scheduled ordering and stock control.
FASTCODE	FASTCODE, MONITOR	¤239	Assembler written monitor for APL applications in APL*PLUS/PC
Gen. Software	PROPS	500+	Spreadsheet system for Product and/or Project Planning.
H.M.W.	INPUT	poa	Matrix manipulation package for data entry & report generation
	PRINTPAK	poa	Block printing for VM/CMS
	VIEWPAK	poa	AP124 Protocol emulator for IBM/PC
	4XTRA	poa	Front-end Foreign Exchange dealing / pos keeping
	Arbitrage	poa	Arbitrage modelling
	Basket	poa	Basket currency modelling
	Menu-Bar	poa	pull-down menu for APL*PLUS/PC
Inner Product	Viewcom	150	Control Viewdata from APL
	APL/dBASE II	150	Interface APL with dBase II
	APL/dBASE III	150	Interface APL with dBASE III
	APL/LOTUS	150	Interface APL with Lotus
	APL/WORDSTAR	150	Interface APL with Wordstar
	APL/MULTIPLAN	150	Interface APL with spreadsheet
	CEMAS	3,500	EEC monetary and agrimonetary analysis.
M.B.T.	RHOMBUS	poa	Integrated Ofsu3suce System
	HASLEMERE	poa	Hotel Accounting System
Mercia	STATGRAPHICS 2.6	635	Integrated stat. graphic system for PCs.
	Upgrade 1.0 to 2.6	295	
	Upgrade 2.0 to 2.6	145	
	EXEC*U*STAT	395	Easy to use Statistics for management.
	APL*PLUS tools		
	- VOL 1	225	IBM PC Utilities:IRMA3270 comms, full screen, RAM Disk report generator
	- VOL 2	125	File documentation, screen editing. Exception handling.
	FINANCIAL AND STATISTICAL LIB.	325	Financial and Statistical analysis
	APL Spreadsheet	195	APL spreadsheet - links to popular spreadsheet software.
	MICROSPAN	250	Comprehensive APL tutor
	LOGOL	poa	Logistics management system for PC, Forecasting, Inventory Control, Scheduling, Distribution, etc
MetaTechnics	MetaScreen	99	Full-screen handler for APL*PLUS/PC, based on VSAPL AP124
	MetaPack	495	Comprehensive utilities package for APL*PLUS/PC. Includes: MetaScreen, MetaWS, Browse, Toolbox, Numeric Editor.
	APL-IEEE488	99	Controls IEEE488/GPIB Bus from APL*PLUS/PC.
	PLOT/PC	99	2D & 3D Graphics package. Includes interactive diagram Editors.
	Browse	99	Scrolling of DOS files, large APL variables.
	ADAPTA DLS	poa	Production & purchasing scheduling for process manufacturing.
	ADAPTA MSP	poa	Job-shop loading & scheduling for multi-stage production.
MicroAPL	MicroTASK	250	Product development aids
	MicroFILE	250	File utilities and database
	MicroPLOT	250	Graphics for HP plotters etc
	MicroLINK	250	General device communications
	MicroEDIT	250	Full screen APL editor
	MicroFORM	250	Full screen forms design
	MicroSPAN	250	Comprehensive APL tutor

	MicroGRID	poa	Ethernet & other networking
	APLCALC	400	APL spreadsheet system
	MicroPLOT/PC	250	For APL*PLUS/PC product
	MicroSPAN/PC	250	For APL*PLUS/PC product
	PC TOOLS Vol 1	295	
	STATGRAPHICS Rel 1	495	
	STATGRAPHICS Rel 2	535	
Parallax	ExecuCalc	¤5,000	Mainframe-based electronic spreadsheet for VM/CMS & MVS/TSO with links to micro products.
	ExecuPlot	¤5,000	Mainframe-based colour graphics with micro links.
I.P. Sharp	ACT	poa	Actuarial system
	APS	poa	Financial Modelling
	BOXJENKINS	poa	Forecasting technique
	CONSOL	poa	Financial Consolidation
	COURSE	poa	APL Instruction
	EASY	poa	Econometric Modelling
	FASTNET	poa	Project Management
	GLOBAL LIMITS	poa	Exposure management for banks
	MABRA	poa	Record maintenance/reporting
	MAGIC	poa	Time series analysis/reporting
	MAGICSTORE	poa	N-dimensional database system
	MAILBOX	poa	Electronic Mail
	MICROCOM	poa	Mainframe to micro link
	SAGA	poa	General graphics, most devices
	SIFT	poa	Forecasting system
	SNAP	poa	Project management
	SUPERPLOT	poa	Business graphics
	VIEWPOINT	poa	4GL - Info centre product
	XTABS	poa	Survey Analysis
Sugar Mill	Stat 1	¤129.95	Statistical toolbox, menu driven
Uniware (mainframe)			
	STSC's ENHANCEMENTS	10,715	Quad-functions & nested arrays for IBM VSAPL under VM/CMS and MVS/TSO
	STSC's SHAREFILE	10,715	Component files for IBM VSAPL under VM/CMS and MVS/TSO and for IBM APL2
	PROGRAMMER TOOLS & UTILITIES	5,715	
	FILEPRINT	5,715	
	FILESORT	5,715	
	FILECONVERT	5,715	
	FILEMANAGER (EMMA)	5,715	STSC's database package.
	APL*PLUS COMPILER	21,430	First APL compiler. Complements APL*PLUS enhancements and Sharefile under VM/CMS and MVS/TSO.
	EXECUCALC	3,995	Mainframe spreadsheet compatible with VISICALC and part of LOTUS 1-2-3 under VSAPL (VM or TSO).
(microcomputer)			
	STATGRAPHICS	725	Statistics & Graphics for PCs.
	STATGRAPHICS FCA	140	An add-on module to STATGRAPHICS: Factorial Correspondence Analysis.
	APL*PLUS/PC TOOLS		
	- VOL1	325	Incl. 327 x IRMA support, RAM disk, full screen data entry, menu input, report generation, games.
	- VOL2	125	Incl. File documentor, screen editor, exception handling.
	SPREADSHEET MNGR	250	APL spreadsheet with built-in ASCII, LOTUS and SYMPHONY interfaces.

APL*PLUS/PC FIN. & STAT.LIBRARY	350	Collection of financial and statistical utilities.
POCKET APL	140	Smaller version of APL*PLUS/PC.
UNIASM	275	Collection of assembler routines for APL*PLUS/PC users.
UNITAB	240	APL*PLUS/PC spreadsheet-like data entry and validation system.
The APL DEBUGGER	105	First released APL*PLUS/PC debugger.
OVERLAYS	250	Fast assembler routines to handle overlays in APL*PLUS/PC.
R:BRIDGE	380	Interface between APL*PLUS/PC & R.BASE 5000.
DMA	380	A version of EMMA (APL database manager) for APL*PLUS/PC users.
APL2C	295	Interface between APL*PLUS/PC and DATALIGHT C language.
ADAPTA/DLS	33,333	Production & purchasing scheduling for process manufacturing.

APL CONSULTANCY

(prices quoted are per day unless otherwise marked)

COMPANY	PRODUCT	PRICES(£)	DETAILS
APL People	Consultancy	poa	Project management financial applications relational databases. Difficult problems solved. Management consultancy. Links to non-APL systems. From consultant level to managing consultant. Documentation a speciality.
APL Software Technology	Consultancy	poa	Technical & business systems, micros, networking & communications a speciality.
Buckland Management Systems	Consultancy	poa	Business and Technical systems in commerce and industry. Designing, programming and implementing applications of O.R and statistics.
Camacho	Consultancy	poa	Specialising in programming & manual writing.
Chapman	Consultancy	150-300	24-hour programmer: APL, C, assembler, graphics; PC, mini, mainframe, network.
Cocking/Drury	Consultancy	120-150	Junior consultant
		140-200	Consultant
		185-300	Senior consultant
		275-400	Managing consultant
Peter Cyriax	Consultancy	100-150	Junior Consultant
		120-200	Consultant
		160-300	Senior Consultant
Delphi	Consultancy	poa	Specialising in management reporting systems and APL on microcomputers.
Dyadic	Consultancy	poa	APL system design, consultancy, programming & training for Dyalog APL, VSAPL, APL*PLUS, IPSA APL etc.
E & S	Consultancy	150-250	System prototyping: all types of information system.
FASTCODE	Consultancy	poa	Specialise in improving performance of APL applications on micros & mainframes.
Gen. Software	Consultancy	100+	
H.M.W.	Consultancy	poa	System design consultancy, programming. HMW specialize in banking and prototyping work.
Inner Product	Consultancy	200	On-site micro-mainframe APL, PC/DOS & Assembler
Lloyd Savage	Consultancy	poa	Decision support, particularly specialising in Sales & Marketing systems.
M.B.T.	Consultancy	poa	
Mercia	Consultancy	poa	APL*PLUS & VSAPL consultancy.
MetaTechnics	Consultancy	poa	Management Information & Production. Engineering. APL - C/Assembler custom programming
MicroAPL	Consultancy	poa	Technical & applications consultancy.

M.T.I.	Consultancy	poa	Specialise in Maintenance and development of existing APL systems
Parallax	Consultancy	¤750	Introductory APL, APL for End-user & Advanced Topics in APL
QB On-Line	Consultancy	200	Specialising in Banking, Financial & Planning Systems.
Rochester Group	Consultancy	poa	Specialise in MIS using Sharp APL
I.P. Sharp	Consultancy	poa	Consultancy & support service world-wide.
Uniware	Consultancy	call	Junior to managing consultancy in APL.

OTHER PRODUCTS

COMPANY	PRODUCT	PRICES(£)	DETAILS
APL People	Employment Agency	poa	Permanent employees placed at all levels. Contractors supplied for short/long-term projects, supervised.
Mercia	ALLCARD	495+	Memory management unit, allowing 952K under DOS - extra 312K APL*PLUS/PC workspace.
	MULTI-APL	195+	Multi-task / Multi-user / Network APL*PLUS/PC with file locking, etc.
I.P. Sharp	Productivity Tools	poa	Utilities for systems, operations, administration & analysts; auxiliary processors, comms software, international network.
	Databases	poa	Financial, aviation, energy and socioeconomic.

VENDOR ADDRESSES

COMPANY	CONTACT	ADDRESS & TELEPHONE No.
Analogic Corporation	Denise Favorat	8 Centennial Drive, Centennial Industrial Park, Peabody, Mass. U.S.A. 01961 Tel: 617-246-0300
APL 385	Adrian Smith	Brook House, Gilling East, York. Tel: 04393-385
APL Impetus Ltd	Cedric Heddle	Rusper, Sandy Lane, Ivy Hatch, SEVENOAKS, Kent TN15 0PD Tel: 0732-885126
APL People	Val Lusmore	17 Barton Street, Bath, Avon. Tel: 0225-62602
	Jill Moss	17 Barton Street, Bath, Avon. Tel: 0225-62602
APL Software Ltd	Phil Goacher	27 Downs Way, Epsom, Surrey KT18 5LU Tel: 03727-21282
	David Alis	Jubilee House, Chapel Rd, HOUNSLOW TW3 1XT Tel: 01 577-3541
APL Software Technology	John Hagger	14 Rosewood Avenue, Alveston, Bristol BS12 2PP Tel: 0454415737
	Per Hultin	46 Vicarage Road, South Benfleet, Essex SS7 1PB Tel: 0374550501
Beta-plan APS	Kim Andreasen	Stengrade 75 , DK-3000 Helsingor, Denmark. Tel:45 2 21 48 48
Buckland Management Systems	John Buckland	Westwood, 9 Grove Road, Camberley, Surrey, GU15 2DN Tel: 0276 684327
Butel Technology Ltd.	Mike Munro	Butel House, 3 Great West Rd., London W4 5QJ Tel: 01-995-1433
Anthony Camacho		2 Blenheim Road, St. Albans, Herts AL1 4NR. Tel: St. Albans (0727) 60130
Paul Chapman		18, Trevelyan Road, London, SW17 9LN Tel: 01-767 4254
Cocking & Drury Ltd.	Romilly Cocking	180 Tottenham Court Road, LONDON, W1P 9LE16 Tel: 01436 9481
		155 Friar Street Reading RG1 1HE. Tel: 0734-588835
Datatrade Ltd.	Tony Checkseld	38 Billing Road, Northampton, NN1 5DQ. Tel: 0604-22289
Delphi Consultation	David Crossley	Church Green House, Stanford-in-the-Vale, Oxon SN7 8LQ. Tel: 03677-384

Dyadic Systems Ltd.	Peter Donnelly	Park House, The High Street, Alton, Hampshire. Tel: 0420-87024
E & S Associates	Frank Evans	19 Homesdale Road, Orpington, Kent BR5 1JS. Tel: 0689-24741
Farnell International Instruments Ltd.	R. Fairbairn	Jubilee House, Sandbeck Way, Wetherby, W. Yorks. Tel: 0937-61961
	Roger Attard	Davenport House, Bowers Way, Harpenden, Herts. Tel: 05827-69071
FASTCODE	Andrew Dickey	P.O. Box 281, Croton-on-Hudson, New York 10520, U.S.A. Tel:(914) 271-3200
General Software Ltd.	M.E. Martin	22 Russell Road, Northolt, Middx. UB5 4QS. Tel: 01-864 9537
H.M.W. Computing Ltd.	Ken Jackson	142 Feltham Hill Rd, Ashford, Middx. TW15 1HN. Tel: 07842-41232, 01 286-7068
IBM UK Ltd	Chris Sell	PO Box 32, Alencon Link, Basingstoke, Hants. RG21 1EJ. Tel: 0256-56144
Inner Product Ltd.	Dominic Murphy	Eagle House, 73 Clapham Common Southside, London SW4 9DG. Tel: 01-673 3354
Lloyd Savage Ltd	Philip Johnson	Cambridge House, Oxford Road, Uxbridge, Middx, UB8 2UD.Tel: 0895-59826
Mercia Software Ltd.	Gareth Brentnall	Aston Science Park, Love Lane, Birmingham B7 4BJ. Tel: 021-359 5096
MetaTechnics Systems	John Stanbridge	Unit 216, 62 Tritton Road, London, SE21 8DE. Tel: 01-670 7959
MicroAPL Ltd.	Bernadette Leverton	South Bank Technopark, 90 London Road, LONDON SE1 6LN Tel: 01-922 8866
M.T.I.	Ray Cannon	7 Pine Wood, Sunbury-on-Thames, Middx. TW16 6SH Tel: 09327 80848
Paraflax Systems Inc.	Kevin Weaver	60 West 9th Street, New York, New York 10011, U.S.A. Tel: 212-475-4001
Peter Cyriax Systems	Peter Cyriax	213 Goldhurst Terrace, London NW6 3ER Tel: 01-624 7013 (Answerphone) 0860-377963 (Mobile)
Portable Software	Richard Smith	60 Aberdeen Ave, Cambridge, Mass. U.S.A. 02138. Tel: 617-547-2918
QB On-Line Systems	Philip Bulmer	5 Surrey House,Portsmouth Rd Camberley, Surrey, GU15 1LB. Tel: 0276-20789
The Rochester Group	Robert Pullman	164 Pinnacle Rd., Rochester NY 14620 Tel:716-461-3169
Shandall Systems Ltd.	Maurice Shanahan	12 High Street, Chalfont St. Giles, Bucks HP8 4QA. Tel: 02407-2027
I.P.S.A. Ltd.	David Weatherby	10 Dean Farrar Street, London SW1. Tel: 01-222 7033
Sugar Mill Software Corp.	Lawrence H. Nitz	1180 Kika Place, Kailua, Hawaii 96734 Tel: (808) 261-7536
Tektronix UK Ltd.	Paul Morgan	Fourth Avenue, Globe Park, Marlow, Bucks SL7 1YD. Tel: 06284-6000
Uniware	Eric Lescasse	15 Rue Erlanger,75016 Paris, France Tel: (1) 45-27-20-61 Telex: 648348F UNIWARE

APL Press Review

by Anthony Camacho

To help with future press reviews, please send any cuttings which mention APL (or photocopies of them) with the date and full name of the publication written on the edge to Cuttings file, Anthony Camacho, 2 Blenheim Rd, St Albans Herts AL1 4NR. Thanks to those who have already been doing this.

The last press review appeared in April 1987 and included items up to February 1987. In theory there should be a lot more material but I have not taken it upon myself to act as cuttings agency or to look through old magazines. The items described below are just those that found their way into the Editor's cuttings file. For a more complete account next time we have to rely on you to send in the cuttings.

To summarise everything would take up far too much of Vector. We believe not many people would be interested in knowing much more than that APL is beginning to get rather more mention than it used to do. I propose to describe each cutting as briefly as it seems to deserve in approximate date order.

1987 Computer User's Year Book - Programming languages entry

"APL ... has spread by its intrinsic merits ... surprisingly, in view of its mathematical background, it is quite popular as a means of implementing financial and managerial problems ... arrays ... large repertoire of functions ... terse ... special characters."

1987 Excerpt from a Hewlett-Packard magazine

The article is called " ... Selecting a Programming Language made Easy" and begins "... we have prepared a chart that matches programming languages with comparable automobiles." Thus: "Assembler - A formula I race car. Very fast, but difficult to drive and expensive to maintain." or "COBOL - A delivery van. It's bulky and ugly but it does the work." or "Pascal - A Volkswagen beetle. It's small but sturdy. Was once popular with intellectuals." or "Logo - A kiddie's replica of a Rolls-Royce. Comes with a real engine and a working horn." or "APL - A double decker bus. It takes rows and columns of passengers to the same place all at the same time. But it drives only in reverse gear and is instrumented in Greek."

13 Feb 1987 Computerworld EXTRA - 55 col-in article by Alan John

Gives a little history and describes Sharp Australia's International Portfolio Management System, mentioning IBM financial services using APL and C A Reed

Associates. The article makes it clear that APL is used and liked by Australian Guarantee, Ord Minnett, Morgan Stanley, Westpac, Dominguez Berry, NSW Intitute of Technology and others although it appears that Australian Bank and Commonwealth Bank may be intending to phase it out.

30 Mar 1987 Computing Australia - full page article by John Lenarcic

"To the uninitiated APL could be as clear as hieroglyphics but John Lenarcic throws light on its real possibilities." It begins with a little history "30,000 users [of IBM, Sharp and STSC APLs] in US including Coca-Cola and Exxon." There is then a clear exposition of some APL features, a discussion of the extended character set; advantages (brevity, no need to translate) and disadvantages (too concise for easy legibility, need special attention to provide them) and some expectation that with the arrival of cheap APLs on powerful micros APL may have found a niche. APL is taught at The University of NSW, the NSW Institute of Technology and at Swinburne Institute of Technology. The prestigious APL88 International Conference will be held in Sydney from 1 to 5 February 1988.

14 May 1987 Computing - review of Murray & Pappas book

The reviewer, David Albury, says the book is well written and easy to follow, concluding "Certainly APL is an interesting development and it will be interesting to see just how fast it catches on." There is another review in Vector 4.3.

May 1987 Acorn User - 2.5 col-in on I-APL

This gives the bare details and address to write to if you are interested on I-APL for the BBC.

May 1987 Micro User - 9 col-in on I-APL 'APL version for BBC'

"The preferred language for internal development at IBM for 20 years, APL can now be shoehorned into a [school] microcomputer. I-APL will be demonstrated at Dallas this month. It will conform to the ISO standard and we hope it will lead to some of the more go-ahead schools using it. There will be no charge for software - only for manuals and copying.

11 June 1987 Computing - 10 col-in stimulated by Vector Editorial

"A British Computer Society working group is fighting a lonely battle to enlighten the world on the benefits of the APL programming language." Mainly negative comment - "got a bad reputation", "expensive", "takes much longer to use it".

25 June 1987 Computing - BAA summer school and I-APL

The British APL Association is helping to pay for a weekend summer school to encourage wider interest in APL among UK maths teachers. "I-APL has raised £20,000 to pay for an interpreter on a number of popular microcomputers ... BBC, Spectrum and Atari ST as well as IBM PCs and compatibles."

29 June 1987 Datalink - front col 'APL challenges BASIC in schools'

This describes an ambitious attempt to replace BASIC as the preferred language for schools, with assembler specialist Paul Chapman "holed up in Tooting" writing the interpreter. "The free interpreter is a key response by the APL community to the marketing failure which, they admit, has cast a shadow over the future of the language. "I-APL may be a breakthrough in awareness in schools."

20 July 1987 Datalink - full page article by Anthony Camacho

Not just another programming language, APL is a notation and it allows people to think in parallel. The search for a precise way to specify processes using ordinary English will fail just as the law has failed. It will always be necessary to think. The article restates some APL advantages while acknowledging that bad APL can be as horrible as bad BASIC. APL is simply the best way of doing mathematics on a computer.

Sept 1987 Practical Computing - 3.5 col-in assessment of APL

A really incompetent piece of work. I wrote to the Editor but he didn't print my letter. "Excellent data type specification. Does not handle high volumes of data very well." The author made it up because there wasn't a book in the library.

Sept 1987 Personal Computer World - 2.5 col-in about I-APL

Headed "Enthusiasm wins" this gives the bare facts and the address to write to for a copy of I-APL.

Nov 1987 Comms of the ACM - 10 col-in letter from J P Pastinelli

A response to an article in which GOTO is considered harmful, this letter gives a one line solution to a problem from the article - find in a matrix how many all-zero rows (if any) there are before the first non-zero row (or the end). The solution is wittily explained.

1 Jan 1988 Times Ed Supp - 4 column-inches about I-APL

"I-APL claims to be the first all-machine truly portable language." BBC B programs wil produce the same results on IBM Mainframe, Sinclair Spectrum, Amiga, AppleII, Amstrads, Atari ST, Archimedes, PC clones and CP/M machines. "A gallon in a pint pot and not a drop spilt." I-APL fits into 26K and will be free: the only charges are for copying and the manuals. Address for enquiries given.

9 Feb 1988 The Australian - 15 column inches about APL & Iverson

"In one of the twists of irony so often seen in the computer industry, a computer language primarily developed for teaching mathematics was shunned by academics and taken up by the financial industry." Ken Iverson was interviewed about his efforts to introduce APL into Canadian schools - he got in a good plug for his method of teaching APL and the I-APL project also gets a mention.

Feb 1988 Computer Education - 1.5 page article by Jill Moss

APL is different because it was originally a notation. It uses symbols for functions like mathematics and handles vectors and matrices. It is easy to learn and people who have used it a lot find they can think with it. APL is the best way of teaching maths on a computer. It is not just academic: APL is used by half of the top 100 companies - and Jill Moss gives some simple examples of what such companies use APL for. I-APL gets a mention (with the address) and so does a BAA scheme to provide mentors (free advisers from the association) for schools. The recent summer school at Winchester, arranged and run by Norman Thomson, is described with a picture of Norman and Sue Fratter who won the free APL computer (an IBM 5110) at the course. The BCS address is given for queries and follow-up.

Feb 1988 Electronics and Wireless World - 8 col-in about I-APL

Under the heading "International programming language" this puts the I-APL press releases into Electronics and Wireless world's own words. The ideas of portability, a notation, home and school machines, fast programming and ISO conformance are all covered. Professors Hawkes and Spence are mentioned and also the fact that we charge for manuals and copying but the software is free.

25 Feb 1988 Computing - 3.5 col-in about APL as a control language

Richard Sharpe's column is about the results of a Grand Prix for fourth generation languages. Problems arise when there is no clear best choice. "A second problem ... is that the language APL did rather well. 217 points from the tests of flexibility - more than

four of the 4GLs. Its strength in the Grand Prix highlights a central problem about languages. The constructs and how you can use them in APL are very powerful. However there is nothing in APL that forces structured coding, remarking or explicit data typing and verification." [I'm afraid the world thinks such attributes are valuable - without them, as he says, you cannot distinguish a coding mistake from an error in specification.]

7 March 1988 Datalink - 8.5 col-in front page 'Sharp looks to Unix APL'

Under a picture of Ken Iverson, the article says "A new phase in the evolution of the APL language has been initiated by leading APL firm I P Sharp. ... I P Sharp has unveiled APL/UX ... targeted at unix workstations ... the most powerful APL yet written ... it is the closest ... to the ideal APL as defined by the language's inventor Ken Iverson."

17 March 1988 Computing - 30 col-in letter from A J Camacho & picture

Under the heading "Call for robust self-discipline" this thanks Sharpe (25 Feb) for reporting a favourable analysis of APL and suggests that the gain in flexibility from weak typing is worth more than any benefit which may arise from strong typing. It says that there is no language which forces structured coding, but, if you stick to direct definition in APL (available 'free' from I-APL), that makes it almost impossible to write Spaghetti-BASIC programs. The address for order forms is given. The picture is a plate of spaghetti with the caption "Direct definition in APL could turn even the most basic spaghetti into a structured program" - if only it could!

April 1988 Micro User - 5.5 col-in about I-APL

The first all-machine truly portable computer language - I-APL - is nearing completion. Mainframe APLs are over 400K bytes of code; I-APL runs in less than 26K. The magazine is, of course, interested mainly in the BBC B version.

THE APL DEBUGGER

reviewed by Alasdair Richardson

All programmers have to debug their programs, and APL programmers are no exception. I use APL*PLUS/PC, and up to last year I was quite happy that an interpreted language, together with STOP and TRACE, makes this an easier job for me than for programmers in most other languages. Then at APL 86 I saw a product demonstrated which seemed to offer the possibility of making debugging even easier. This was the APL DEBUGGER from a French company UNIWARE. Shortly afterwards I ordered a copy, and have never looked back.

So what does this marvellous piece of software do? The beauty of the debugger is that you can, with the minimum of effort, actually watch your function executing line by line, see the result of each line, and home straight in on the line which is causing a problem. The time saved when debugging can be very substantial, and the simplicity of use can save a lot of swearing and cursing when you forget to put a STOP on the right line, remove STOPS before)SAVEing etc.

You are supplied with a workspace called DEBUGGER.AWS, which contains 15 functions and one variable. The functions are called deltaDEBUG, deltaDEBUG1 and so on up to deltaDEBUG14, and the variable DEBUGFNS is simply a list of the names of these functions. To use the debugger you first)COPY the workspace into your own workspace (the variable DEBUGFNS just makes it easier to get rid of them afterwards by quadEX DEBUGFNS). Then you insert 'goto deltaDEBUG' as the first line of the function you wish to debug. It doesn't matter if you are debugging a stand-alone function or one which is embedded in the middle of a complete system. The function can be recursive, and you can debug more than one function at a time.

When the function you are debugging is called, a flashing 'WAIT' appears at the top right of the screen, and after a short time a window opens up at the right of the screen showing that function. The current line is highlighted in reverse video. Pressing RETURN takes you to the next line. As that line is executed, another window opens up at the top containing the full line of APL, and underneath the final result if there is one. If the result is too large to fit in the window, the cursor keys will scroll it so you can see the whole result. Pressing RETURN again takes you to the next line. Diamondised statements are executed one at a time, so unlike TRACE you can see the result of each statement. Comments are just ignored.

When a statement writes to or reads from the screen, the debugger will recognize this and withdraw all its windows so you can see exactly what your application screen looks like, then pressing RETURN again takes you back into the debugger. Pressing Scroll Lock opens up an 'immediate execution' window at the bottom left of the screen, the size of

which you can alter if you so wish. In this window you cannot use any APL commands or branch expressions, but you can look at or alter any variables, test variations of APL statements etc. You cannot use quadEDIT to alter your function, but if you find a variable which has been given the wrong value you can alter it and carry on, or you can enter a naked branch to take you straight out to APL.

There are a few (well documented) restrictions on the use of the debugger, and I will just mention the most important here. It cannot be used for debugging graphics functions, which is a pity as the poor graphics programmer could really do with a tool like this. Your application quadELX does not work when debugging. If you are using your error handling purely for error trapping of course this doesn't matter, but if you are using it for cutting back the stack or something you need to be aware that it's not going to work. Finally if you want to use quadSEG or quadPOKE in the immediate execution window, you should read the supplied documentation carefully.

I have been using the DEBUGGER now for 9 months, and have come across only a few very minor irritations. Pressing Ctrl/ENTER will cause your function to go on executing as if you were continually pressing ENTER, but the result of each line is still shown so it takes some time. It would be nice if you could indicate that you want to stop displaying each line as it's executed after you have located your problem and if you don't want to branch straight out to APL. Occasionally using the debugger does odd things to the cursor, which have always been irritating rather than important. Thus very occasionally when branching out to APL I have been left with no cursor at all, necessitating going to look up the correct POKE in the manual to get it back again. When using the debugger in conjunction with POWERTOOLS/PC (an extremely useful combination), returning to menu fields you find that a cursor appears where normally it wouldn't.

All in all the APL DEBUGGER is a tool I would highly recommend to any APL programmer using APL*PLUS/PC for non-graphics applications. If Uniware manage to produce a release 2 allowing debugging of graphics functions and the use of quadEDIT on the currently executing function it will be just about perfect.

PL/PC - "APL for All" ?

Reviewed by N.J. Small

Ever since APL was first implemented as a computer language there have been many and varied suggestions as to how it might be improved. However, the introduction of changes in the language has been slow, at least as perceived by the end user (i.e. the programmer). This has been so for a variety of reasons, among which are: small user base; concern about the "purity" of the language; complacency.

It would be only charitable to assume that the last given reason is why there is no sign of a full-screen editor in certain implementations. Similarly, the desire to avoid adulterating the language explains why there are available few of the special functions needed to do in O(n) steps (as immediately perceived as possible by the ex-FORTRAN programmer) what requires O(n*2) steps in APL (not to mention the saving in storage). Again, there are plenty of user-defined functions that require more than two parameters - and whilst using nested arrays may be fine for this in APL philosophy, they are hardly computationally efficient; (sometimes it is nice to be able to pass arrays by address, too).

PL/PC is a new variation on the theme of APL. Its main enhancements over APL - control structures (resembling those of Modula-2: for, while, if, etc.) and compilation - are hardly new in concept; see for example Harris,1973 [1]. However, time has not eroded the benefits obtainable from these features. Indeed, control structures have become more valuable, owing to their ease of programming, flexibility, and clarity, because (i) the problems that programmers are expected to solve have become more difficult and (ii) the proportion of project costs attributable to programming and program maintenance is increasing. The advantages of compilation are clear, but difficult to quantify without a non-compiling version. PL/PC performs what might be described as "run-time compilation", in the sense that a subroutine is compiled only on the first time that it is invoked after the workspace has been loaded.

Other additional features include: a built-in numeric editor; a number of new primitive functions including Bessel operators and fast Fourier transforms; integrated numeric types, including single-bit logical values, single-byte integers, and complex variables; string as well as character data types.

There are some deviations from standard APL practice: keywords are used in place of special symbols; evaluation is from left to right; index origin is always zero; embedded assignments are not permitted; the "execute" function accepts only primitive functions in its argument - strings containing the names of user-defined functions produce an error; the state indicator is cleared when a workspace is saved, so one cannot interrupt a process, save the workspace, and resume tomorrow. The first two of these, and possibly

the third, may be regarded as features of the language, but the remainder are definitely restrictions.

The graphics facilities offered include turtle graphics functions, functions to draw conic sections and polygons, and functions for "painting". Thus, plotting data, drawing a picture, or doing dynamic graphics should not present too much of a problem; there is not much help, however, for the programmer wishing to do the kind of full-screen management required for menus, etc., and as actually used by the numeric editor (do-it-yourself multiple windows are necessary).

In immediate execution mode, PL/PC is considerably less friendly than it might be. In particular: (i) the results lines are not cleared of any previous display, which can lead to confusion when a command line is edited and re-executed; (ii) facilities for cursor movement and the erasing of fields are rudimentary, comparing unfavourably with those available in editing mode; (iii) there is no insert mode - only single character insertion (in contrast to the editors described below); (iv) there is no scrolling memory. Also, it is frustrating to have the screen cleared every time that an editor is invoked or a workspace loaded.

There are two editors in PL/PC, a text editor for functions and a data editor for both character and numeric data. The former is reasonably friendly and in particular has tabbing to give indentation, allowing a marked improvement in the readability of functions (there is no automatic formatting of functions); most of the facilities are controlled by function keys and, unfortunately, these are not screened from any values assigned to them using the DOS PROMPT command, which can be a problem. This text editor makes use of a temporary disk file, which always contains the last subroutine edited (in ASCII format); this last edited subroutine unexpectedly becomes fixed in the workspace if one issues the edit command without an object name, say to edit a new function or a text file; if there is no temporary file and a text file is edited, error messages are issued on leaving the editor as the interpreter attempts to fix as a subroutine the contents of the (empty) temporary file.

The data editor has a few quirks as well. It displays the contents of the variable being edited on the main part of the screen, with a field at the bottom of the screen for the contents of the current element displayed for editing. For numeric data, the format specifications for these two field types differ, which can be confusing; also, with sufficiently many decimal digits it is possible to make a number fractionally smaller and yet induce rounding up. A more serious bug is that the advertised use of break does not enable one to abort an edit session. The editing of character variables is awkward as there is no automatic movement of the cursor when a new character is typed; for string variables, editing is restricted by the size of the edit field.

The use of the two editors would be made easier if they were more similar in their use of special keys.

The manual is rather thin considering the ground that it has to cover, but detail is generally on the right side of adequacy, though someone without prior knowledge of APL might find the going heavy. In one or two cases the explanation is definitely inadequate. For example, in describing the use of the "file" command, for transferring data to or from a file, the syntax is not explained and there is an unhelpful reference to the appendix; fortunately, an example that works is given. Incidentally, this file access method is well-suited to direct access use.

There is no atomic vector in PL/PC, though the expression string (index 255) is equivalent. What would be useful in the manual is a list of the character set, if only to avoid loss of time in finding substitute keys.

As might be expected, PL/PC takes up a lot of core, leaving some 300k bytes free on a PC with 640k bytes RAM, though there is some storage reserved for subroutines; the documentation states that workspaces can be up to 330k bytes. Object sizes are limited to 64k bytes (except for virtual objects used for file access).

In conclusion it may be said that the control structures of PL/PC make it a lot easier and neater to perform certain tasks than it is in APL - tasks such as (forgive my mentioning it) looping or performing conditional calculations. Its additional functions and graphics capabilities will prove useful for certain applications. However, ease of use is all-important in the market at which this package is apparently aimed, and some improvements will be required in this direction; but most of the deficiencies described above are of a fairly minor nature and could easily be overcome with a little "polishing" of the system. Then, perhaps, PL/PC will match the claim in the blurb of being "APL for all".

Reference [1] L.R.Harris. A logical control structure for APL. APL Congress '73, pp203-210 (North Holland, 1973).

RECENT MEETINGS: APL 88

This section of VECTOR is devoted to members' notes and impressions of APL 88 in Sydney, Australia.

The notes are by Anthony Camacho and Adrian Smith, and the photographs by Anthony Camacho and David Ziemann. Where possible, readers are referred to the APL 88 Proceedings for more detail.

The APL 88 Proceedings are available (prepaid) from:

ACM Order Department
Waverly Press
P.O. Box 64145
Baltimore, MD
USA 21264

Price to members $21.00
Standard price $28.00

ACM Order number: 554880

Impressions of Sydney

by John Sullivan

Sydney is very much a mixture of styles. On first sight it appears very British: cars drive on the left, Sydneysiders can spell, they play cricket, some of the place names are very familiar, and there's a very good public transport system; but then you gradually notice the American influence: words-only road signs, traffic lights go straight from red to green, etc.; and then there's the bits that fit nowhere: most pubs shut at 8 p.m. on Sunday, the people are very friendly, and it's safe for anybody to walk alone through the park at night. All-in-all, Sydney is a very attractive place. I like Sydney.

View of Sydney from Victoria Park
(Between the School of Architecture and the Plenary Lecture Theatre)

I went to Sydney for the APL conference, but it seemed such a waste to go all that way and not immerse myself in the culture and surroundings. I say 'culture' and I mean 'culture' because, contrary to what we are led to believe by the media, Australia does have a flourishing culture, both of European origins and its own. On the day after I arrived there was a free classical concert in the Domain (a large park in the centre of Sydney) which was so oversubscribed that it was impossible to find anywhere decent to sit within reasonable hearing distance of the orchestra. At the other end of the scale, after the Conference had finished I decided to go to hear the 'Magic Flute' at the Sydney Opera House, and this was absolutely delightful. I was also impressed by the excellent air-conditioning they have in the Opera House, it was completely silent!

Pubs in Sydney generally are nothing like those in England. First of all they are a man's world, usually the only ladies present are those serving behind the bar. Secondly the Aussies like to drink their 'beer' out of very small glasses. Presumably this is because

their beer is chilled so that you can't taste it, and they don't want to give it the slightest chance of warming up. (The other thing I didn't like about Australian beer was the excessive fizziness, making it look and taste like the soda pop I used to drink when I was very young.) Thirdly, Aussies don't go in for 'rounds' like we do: although one person may order for everyone, they all pay for their own.

Having complained about the typical Australian boozer, I am now going to recommend one! Close to International House was a pub called the 'Duck and Swan', which sells a range of imported beer from all round the world, in addition to the local brew. They also do a very nice line in British-style beers and stout (very tasty once they had been allowed to warm up!) that are brewed in Fremantle, W.A., and are sold by the pint. The staff were very friendly, too, and I was presented with a large Australian-style beer glass (yes there are a few large ones about) as a going-away present 'for being such a good customer' (make of that what you will!). The food in International House reminded me of the food at Manchester, so I decided to eat out and sample several of the restaurants in Glebe Point Road, about half a mile away the other side of Victoria Park. There is a great variety of restaurants there, from Vietnamese to Italian and German. Not one of them was really expensive: you could eat a meal for $15 (about £6) and feel full. I also sampled a couple of the restaurants in Kings Cross (those of you who saw Whicker's World will remember that Kings Cross is the red-light district), where the food is, if anything, even cheaper and just as good.

When a couple of my colleagues heard that I had a ticket for the Bicentennial Test Match they got down on their knees and prayed for rain. Sure enough, at ten past four on the Sunday, bad light stopped play, and the rain started at five o'clock. I didn't really mind, however, because in the play that we did have, seven Australian wickets fell (including one dubious LBW decision by their umpire!) and a good time was had by all.

There isn't enough room in a short non-technical article like this to tell of all the things there are to see and do in and around Sydney. I went to the Blue Mountains, so-called from the blue haze exuded by the eucalyptus trees; I tasted wine from the Hunter Valley which could put the equivalent wines from several European countries to shame; I travelled on the double-deck commuter trains in Sydney, I hugged a koala in the wildlife park. There wasn't enough time to do all the things I wanted to do. I have decided: if the Australians will have me, I am going back there for a holiday sometime soon.

Impressions of APL 88

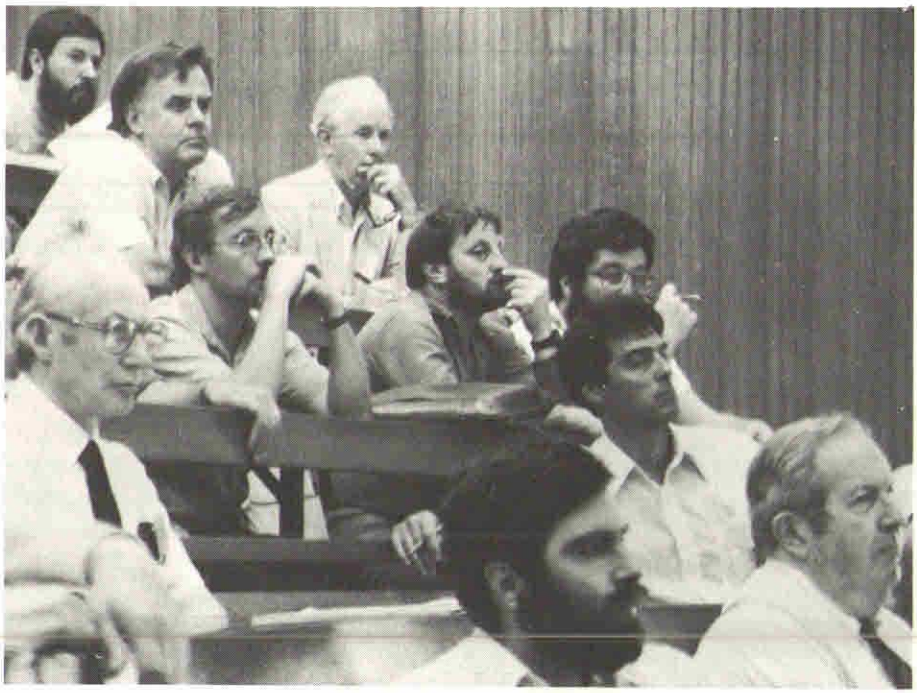
by Anthony Camacho

APL 88 was held from 1-5 February 1988 in Sydney, Australia. Delegates were accommodated in three students' halls: International House, Wesley College and St. Andrews College. Ordinary sessions were held in three lecture theatres in the Wilkinson building of the School of Architecture. Plenary sessions were held in the Stephen Roberts lecture theatre, five minutes' walk across Victoria Park.

The Entrance to the School of Architecture

The Wilkinson building is modern and the lecture theatres are smart, comfortable and air-conditioned. (February in Australia is the equivalent of August in the northern hemisphere and Sydney is about the same distance from the equator as Morocco. It was hot. It rains quite often so the grass stays green and the parks beautiful.) There is also a large lobby where tea, coffee and orange juice were served, and a large open space for the exhibition and hospitality suites. The software exchange, with several PCs, was set up in a room off the lobby.

There were 46 presentations of submitted papers, four panels, two plenary panels, a plenary for SigAPL, a special invited plenary on crystallography, and opening and closing plenaries. The day before the conference there was an Education Day. Over thirty Australian secondary teachers turned up and were addressed by Ken Iverson, Howard Peelle, a teacher from a Japanese school, Makoto Kikkawa, and the I-APL team introducing the hands-on sessions. Each teacher went away with "An APL Tutorial", "An Encyclopedia of APL" and, if they had access to a PC, I-APL/PC and its instruction manual. The hands-on sessions were held in the Madsen building by courtesy of Professor Ken Sinclair.



Joey Tuttle Eugene McDonnell
Rob Hodgkinson Mel Chapman John Sullivan
Donald MacIntyre Howard Peelle
Dave Weintraub George Cherlin

Part of the Audience at a Plenary Session

Below you will find short descriptions of some of the presentations in APL 88 proper. The papers themselves may be read in the proceedings - available from SigAPL. There are

also longer reports of sessions not covered by printed papers. The selection depends on the VECTOR reporters' choice of sessions to attend.

The Conference committee had organised a smart yellow satchel with green APL 88 logo and green binding for each delegate, with a name tag to help avoid inadvertent exchanges, stuffed with:

a welcome from Sydney APL UG
the programme
the proceedings and photocopies of some extra papers
an envelope with tickets for T-shirt, theatre and banquet
a BAA leaflet
an I-APL newsletter (no. 6)
an APL 89 badge
a SigAPL leaflet
maps of Sydney and the University
leaflets about public transport and tours
a pad of paper and a pen

Vector 4.3 arrived too late to be stuffed in the satchels but there was a copy for every delegate as promised.

The T-shirts were yellow and of three sizes. The APL 88 logo was printed in green on the front and the size in APL on the back. The three sizes were 'min over iota zero', 'omega domino omega to the power zero', and 'power max over iota zero'.

The tickets were for the Everest Theatre at the Seymour Centre, next door to International House, and the play was 'Faces in the Street' - a dramatisation by Frank Hardy, himself a maverick, of the life of Australia's radical socialist poet Henry Lawson (1867-1922). 'Faces in the Street' is also the title of Lawson's most famous poem, about the poverty and despair of life in Sydney written in 1888 - another centenary. The plot was somewhat contrived (the principal character was a mental patient who thought he was Lawson) to allow the distinction between fact and fiction to be blurred - this may have helped with the dramatisation but it reduced the value of the evening as a lesson in Australian history. Nevertheless it was an ideal subject for our group, most of whom were visiting Australia for the first time.

The banquet was in the refectory of the Holme building, which is decorated with murals by Lo Schiavo. One of them ,"Mankind", is the largest mural in the southern hemisphere - it runs the whole length of the refectory which must be over a hundred feet. The setting was magnificent and the banquet itself matched the setting. There was a display of cold food with glazed turkeys, hams, huge decorated fish, a cornucopia of fruits and gateaux, and a centrepiece which was a three-foot-high seahorse modelled in ice at the top of a small hill of crabs, lobsters and prawns. It not only looked delicious, it was. While we ate,

a consort of early musical instruments played. It was led by Peter Petocz who had also presented a paper on Monday. The SigAPL outstanding achievement award was awarded to Allen J. Rose, APL 89 and 90 were advertised, Linda Alvord appealed for us to draw back into the fold APLers who had allowed their membership to lapse, and money was raised for I-APL distribution in Australia by auctioning some of Donald McIntyre's models of regular solids (Phil Benkard made a convincing auctioneer) and raffling the master and first copy of I-APL/PC. I have never attended a better banquet; it will be remembered as outstanding for many APL conferences.

Overall the conference ran smoothly. From Warren Julian's opening to the final paper by Ken Iverson nothing significant went wrong. The display equipment always worked, the refreshments were always on time and there was even a substantial amount of press coverage, some during and some after the conference. Even the university accommodation and food were acceptable - only the really fussy went out for meals. Those who wanted could go swimming before breakfast or watch the fruit bats (they have wingspans of two or three feet) eating figs from the trees in the park after supper.

Our verdict is that APL 88 was a best buy among APL conferences and that the people who ran it, and the many who helped, deserve our sincere thanks and congratulations.

Workshop: Secondary School Mathematics using APL

notes by Adrian Smith

The Development of Notation
Neville Holmes

This was a really effective introduction to the day. Neville was at pains to place APL in the context of the long history of notational development, from Babylonian times to the present day. He made it clear that these things take time, and that APL is just another step forward in a continuing process.

He started with the classic example of the invention of a symbolic notation (writing) around 3000BC. In ancient Babylon from 8000BC onwards, you can find scattered clay tokens. These were the 'despatch notes' of the time; if you sent off a consignment of goods with a courier, you needed some reliable way of ensuring that there was no 'shrinkage' along the way. What you did was to make an equivalent set of clay tokens (one per item) and roll the whole lot into a clay ball. This was then shipped with the goods; the receiver broke the ball and could check the tokens against the physical goods delivered.

Spot the flaw! A clever courier could just break the ball, remove the appropriate tokens, and reassemble it to match the reduced cargo. To deter this, they took to pressing matching tokens into the outside of the ball, so that the intermediary could not easily rebuild it. Some 1000's of years later - the breakthrough!! They ceased to put the tokens in at all, and then some bright spark had the idea of just scratching the marks with a pointy stick. So was born the notion of writing.

Counting has gone through much the same pattern. The Romans used a complex set of hand signals to represent the numbers from 1 up to 100. These were transcribed into the familiar Roman Numerals, and were used, together with simple tally sticks, for hundreds of years. In fact decimal notation only drove out the tally-stick from official treasury records as late as 1826.

In both these examples, we see an evolution through:

- Rhetorical. Things have names, or individual symbols. The clay tokens were images of their real counterparts; the Roman hand signals gave the names of the numbers.

- Syncopated. The symbols are made easier to record; pictograms turn into stylized groups of lines.

- Symbolic. The whole thing is clarified into symbols which follow consistent rules and hence can be reliably manipulated. So it is with an alphabet (rather than simplified pictures of sheep), and with the decimal notation (rather than with simplified pictures of Roman hand-signals).

APL is to current algebra what the decimal system was to Roman counting, or a rational alphabet was to Hieroglyphs. It takes the notation forward from its current synchopated form to something which is genuinely symbolic. In the past, symbols have crept gradually into the system (+ - around 1489, = in 1557, |A| in 1841, and ∧ in 1933); APL is the first attempt to give consistency and coherence to a whole mish-mash of arbitrary rules.

Neville closed by just re-emphasising those dates! If we get APL accepted in less than 300 years, we have done pretty well!

Reforming Maths Notation
Howard Peelle

Some 300+ colleges and universities are using APL. Only a dozen or so high-schools are, and few (if any) elementary schools. Yet in APL you only need to write the mathematics; other languages force you to carry so much excess baggage. Why is APL taking so long to become accepted where it is potentially most useful?

Some of the issues are:

- does APL provide you with too much? 60-odd primitives; are they too powerful? In fact does APL do the maths for you?
- is it too symbolic? There are enough 'math-phobics' in the world already.
- there are quite a few common 'bugs' in learning APL. Sometimes one symbol (such as iota) is used in two very different ways; does this cause problems?
- we all talk about 'APL thinking'. We still don't know what this means, or really how APL changes the way you think.
- does APL interact badly with the established world of Basic, Pascal etc?

These are things which must be tackled not only by the APL professionals, but by teachers as well if we are to see a real spread of APL into elementary education.

Hands on

This was the fun part of the day! 20 + teachers were grouped around 8 IBM PC's, and let loose on I-APL. At the end of the session, the copy of I-APL (complete with full documentation) was theirs to keep.

Peter Petocz explaining a point

They were helped and supported by 8 willing volunteers, who just about managed to keep their collective hands off the keyboards for the duration of the session! It was (as usual) quite hard to credit just how fast the teachers took to the systems, and indeed how little of a barrier the keyboard was. In fact by the end of the morning I was just about convinced that I-APL have got it right in going for their unique visual mapping, rather than adopting some variant of the 'standard' VS APL layout.

Education Workshop Helpers: Ian Shannon and Rob Hodgkinson standing

We broke for lunch (dragging away reluctant pupils and masters alike) and resumed with Makato Kikkawa's splendid account of APL in a Japanese high school. As this is documented in the Education VECTOR (page 21), I shall say no more here, except to record the warm enthusiasm with which it was received. A further hour's practice passed quickly, before we returned to the lecture theatre for Ken Iverson's talk.

APL in Canadian Schools
Ken Iverson

Ken has been using APL in teaching since his days as a junior faculty member (in Computer Science) in 1955. He had found traditional notation very helpful, but not adequate; hence the evolution of APL which he used for over 8 years before it was ever implemented on a computer.

In particular, the formal description of the IBM/360 (in APL) was a marvellous tool for teaching, and was indeed used for debugging the real thing! Finally in 1981 Ken was able to set up 'a glorious week' with 15 APL terminals and 30 maths professors. He was already moving away from computer science towards mathematics, and when he retired in 1987 he chose to devote all his time to the use of APL in maths and related subjects.

Ken Iverson with a Visitor to the Education Workshop

In the Province of Ontario, APL was apparently given a head-start when 'somebody sneaked it in' to the standard offering of 5 languages on the schools micro. Are any schools using it? "Not as far as I could find out". Clearly, easy availability of an APL system is necessary, but is by no means a sufficient condition for success. I-APL be warned!! The distributers of computer equipment are primarily in contact with the people teaching computers, not the people teaching mathematics.

For maths teachers, APL provides a coherent approach to teaching maths - not a concoction of games and puzzles as typically used to teach programming! It prevents the students escaping from high school thinking that algebra is all about letters, when it is really all about names. It is easy to learn - as is any language when you have a native speaker to help you try it out! In APL's case the native speaker is sitting on your desk.

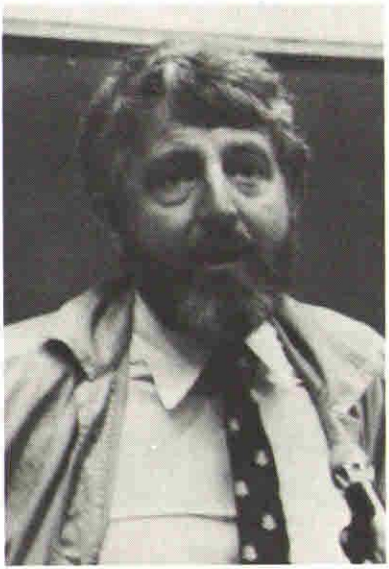
APL also gives you new insights, for example it makes it clear why "0 to the power 0" should be defined as 1; or why 1 is not a prime. Ken concluded with some clear advice, in part as a reaction to the obvious enthusiasm of most of the 20 teachers present: "Don't go back and start teaching APL. Learn it first. Then teach it!"

APL 88 Opening Plenary Session: Computing at the University of Sydney

by Warren Julian

Dr Julian generously made the University of Sydney School of Architecture available as the venue for most of the sessions of APL88. The School of Architecture has one of several impressive computing laboratories which we saw around the campus and Dr Julian set himself to describe how computing in the University had changed over the 30 years from the early 1950s mainframe called SILIAC to today's banks of PCs and Macintoshes.

Dr Julian recounted how, in the High and Far Off Fifties, the then Professor of Physics raised the $50,000 (estimated) and the $75,000 (actual) needed to buy the new-fangled machine from Illinois. The authors believe that it is his name that is now carved in stone under the eaves of the Physics School in the august company of Newton, Einstein and other luminaries.



Warren Julian

In 1964 those original, precious 1024 words were replaced with the help of some Texan money by a KDF9 and 4096 words and ALGOL. In due course IBM got a foot in the door and by 1967 there was a 7040 and FORTRAN on cards. In the School of Architecture APL was introduced to economise on programmers but in the Computer Science Department it is still, as so often, a novelty with which few students are involved.

Gradually the emphasis within the School of Architecture is shifting towards knowledge engineering, some of it based on bought-in packages but including APL on Macintoshes. A great deal of the work done arises out of the modern bureaucratic nightmare of codes and regulations which constrain all those who would build in a crowded city.

It is interesting that the University is experiencing the same dilemma as faces many other computer users who have until recently been content with a time-sharing service based on a central mainframe. Now that every Department has a bank of microcomputers what role is left for the once mighty machine? We gather that in Sydney, as elsewhere, this debate is still active.

Iterative Scaling of Marks

Peter Petocz

This talk described a consulting job, where APL's matrix features had been used to advantage to get quick and accurate results. The basic problem is how to compare scores on different tests when the candidature varies. For example students take exams in 5 out of 20 topics offered; different subjects attract subjects of differing ability, so you cannot just standardize the scores to the same mean and S.Devn. and add! How do you give a fair score to the minority who take the harder subjects?

The solution is a form of inter-subject scaling, using the standard of candidature in each subject. Each subject is adjusted based on how the students did in all the other subjects (....) until the whole set of marks converges. Typically 40,000 pupils take the N.S.W. tertiary selection - so it matters to get it right!

For the code and the details of the algorithm (it fits neatly on one page) see the APL88 Proceedings Page 263. Typically 3 or 4 iterations settles it (rounding to the nearest integer helps!) and a 'pressure test' such as adding 10% to all marks in a particular subject has no effect on the final outcome.

Peter's final comment was that (in a topic which has aroused vehement public debate) it is nice to see the entire process expressed in a couple of simple APL functions, rather than in pages of incomprehensible text in the Educational Supplements!

Linda Alvord noted that this would be a wonderful one to let the kids loose on ... it is something they really care about! You would certainly get some thoughtful reactions to its fairness.

APL Applied in Music Theory

by Michael Kassler

The regularities of Western music have not yet been made completely clear. This presentation described the work in progress by Michael Kassler to convert theories of tonality proposed by Kollman (1756-1829) and Schenker (1868-1935) into APL. To do this it is necessary to make the vague parts of the theories precise.

The representation of polyphonic music naturally suits a matrix where the rows are the parts and the columns the passage of time: all notes in a column sound simultaneously.

It was fascinating to see how the theories add decorations to a simple phrase. The examples were shown in beautiful laser-printed music notation: a pity none of them was played or sung to us.

Would it be possible to establish the APL representation of music as a computer standard? Are there any rivals to it?

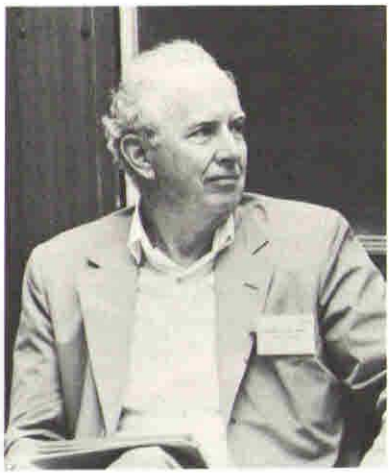
Life: Nasty Brutish and Short

by E E McDonnell

The man whose life is discussed is John Horton Conway. 'Life' is the game with cellular automata. In APL it is possible to write functions to calculate the successive generations very concisely.

Donald McIntyre's version manages to be both concise and easy to understand. Some are impenetrable. Only those with some interesting lesson for us were shown.

Defining globals as 'nasty' and exhaustive testing of all possibilities as 'brutish', the author worked his way through a series of shorter and shorter LIFE programs ending with one only nine tokens long, which is certainly short (only twelve characters plus the function name and colon). The global is a vector of nested arrays each three by three. The point of the exercise was to show the importance of the new operators in APL. The nine-token solution uses both 'cut' and 'rank'.



Eugene McDonnell

The whole was presented with a dry wit: the audience was far from solitary and nothing about the occasion was poor.

Symmetries of the Firing Squad Synchronisation Problem Revealed in a Nested Array.

by J Philip Benkard

The problem is to get a line of automata or soldiers all to fire synchronously after the order to fire is given to a soldier at one end of the line.

Each soldier can only communicate with his immediate neighbours. Plainly if all are to fire at once the message must be passed at least to the other end of the line and back again.

For example the message one way could be the count of the position of each soldier/automaton in the line, which could be stored by each of them and then used on receipt of the return message to begin a count down, one count per message time, to the firing time. This solution calls for many message types and rather complex automata - they must be able to count up to the length of the line among other things.

J Philip Benkard

Phil Benkard presented a solution which works on unlimited length lines with twelve-state automata and a single message type described as a shoulder-tap. There are many interesting symmetries involved, not least that of the matrix used to determine the next state of each automaton from its current state and incoming messages. When the matrix is expressed as a nested array there is a large saving in space.

Interactive Simulation Modelling using APL

Tatsuo Aonuma

Prof. Aonuma described the use of APL in teaching management skills, especially in the use of simulation. He uses APL in 2 ways: first he gives all students 10hrs APL teaching so that they can do the basic statistical analysis; then he uses his simulation generator to let them play with complex dynamic models without the need for a high level of APL programming skill.

Typically a multi-period planning model can be devised, tested and run with only enough APL skill to define the relationships between the variables in the system. The simulation generator takes care of all the hard bits (rather like a spreadsheet, only much more flexible) such as detecting circularity, and getting the equations in the right order. There are lots of handy built-in functions to define lagged relationships, exponential growth and so on.

Networking APLs

Gary Sullivan (on Zeidner panel)

Gary described the process of making IBM chips as 'a bit like etching a truck from a solid block of steel'. Typically there are some 300 stages in the process, and the same chip is often competing with itself for the same tool set at different stages. The challenge was to cut costs and improve quality; they needed an 'expert systems' approach as there was no well-understood model of the plant.

The system was described as a 'proactive embedded' expert system; it takes the transaction stream from the plant (some 240,000 transactions per day) and enhances this to yield a state array of the plant. This can be compared with the plan to locate trouble spots. The system uses both MVS and VM hosts, and PC's to handle the interaction with the shop floor. It has improved tool throughput by between 20% and 80%.

Plenary Session: The Past of APL

Ken Iverson gave the opening address, explaining that he would not repeat the credits already given to so many of those who had contributed technically to the development of APL but would try to cover those in other categories such as administrators, secretaries, organisers, teachers and do-it-yourselfers. The Vector reporter apologises for any names he missed.

Ken reminded us that when he began working with Adin Falkoff some years before implementation, APL was seen as a tool for developing and expressing thoughts.

Ken Iverson, Curtis Jones and Eugene McDonnell

Among the executives and administrators he mentioned A K Watson, who found resources for the work, John McPherson, who was a vice-president of IBM, and John Lawrence, the editor of the IBM Systems Journal through whose articles Larry Breed, R H Lathwell and Roger Moore were attracted to the project. Among secretaries and operators Colleen Conway was mentioned. The first conference organiser was Garth

Foster and the first course organiser was Al Rose who led on to teachers such as Linda Alvord. In Europe Reyner Cokeham and Yves Le Borgne, who attended the first course outside IBM which was at NASA, Per Gjerlov, Gustav Tollet and Timo Seppala were mentioned. Ken ended by quoting Fred Brooks, his co-author of "Automatic Data Processing", who after spending some time in Australia said that the book "A Programming Language" was better known here than in the USA.

Neville Holmes then spoke about APL in Australia. He had learned APL in the early 60s from a Frenchman who had brought it back from the SRI. Then the Australian Bureau of Statistics adopted it for specifications. The Sydney APL Users Group was founded in 1977 and has had as many as 100 members; there's also a Melbourne group. Neville hoped that APL 88, the Education Workshop and I-APL would make it possible to get APL into use in schools.

Dick Bowman, for the British APL Association, said that the Association had been set up by Romilly Cocking. It holds six meetings a year and publishes VECTOR - 128 pages - four times a year. The BAA ran "APL Business Technology 83" at Loughborough, APL 86 at Manchester and will run "APL-Ication" at Canterbury in Autumn 88.

Kyosuke Saigusa said that in Japan there were two groups. One is the APL Association, founded in 1977, which publishes an APL Journal, has 120 members and Ken Iverson as Hon. Chairman; it is rather academic.

The other is the APL Club sponsored by IBM as part of its marketing program and with 200 companies, mainly users of IBM machines, as its members. It too has a magazine, called APL-Club. As IBM seems to be losing interest in APL, now would be a good time for the groups to amalgamate and consolidate their resources.

Kyosuke Saigusa

Marilyn Pritchard explained how the rules of the ACM had to be changed, with Alan Perlis' help to allow the formation of Sigplan Technical Committee on APL or STAPL. She asked us to attend the SigAPL plenary later that afternoon if we wanted to know more about its current state; at present it is losing members at the rate of 100 a year and is spending more than its income. It has 1600 members and $30,000 in the bank so eclipse isn't imminent. Linda Alvord has taken on the job of recruitment and re-recruitment.

Marilyn Pritchard

Gitte Christensen

Gitte Christensen began APL in 1968 at a summer school taught by Iverson. By 1973, Denmark had five APL groups and the highest density of APL use in the world. In 1978 the APL subgroup of the Danish Data Processing Association was formed and it holds meetings and conferences and will host APL 90 in Copenhagen. The Danish APLers are working hard to get APL into schools and hope, with I-APL, to succeed. They are greatly increasing their emphasis on "ground level" APL and on helping people to use APL better.

Curtis A. Jones said the Bay Area User Group was formed by the people who ran APL 81 in San Francisco. It has meetings at the Sharp offices in Palo Alto. Membership statistics are untrustworthy but the group may be growing; it is certainly adding officers as now there is a proper committee, whereas last year there was only Curtis Jones himself.

Eugene McDonnell spoke about the past of the standardisation effort. The original impetus arose on one of Garth Foster's courses when Clark Wiedmann asked "What is the result of this expression on your system?" In practice, first APL/SV and then VS/APL were the definition of APL, but it was unacceptable to have a notation defined by the results from an implementation of it. In 1975 at Pisa an attempt was made to define APL in Backus normal form. IBM tried to produce an internal standard published at APL 79.

This may have been what sparked off Raymond Tisserand to try to get ISO to take over the IBM standard. That attempt roused the other US manufacturers to make great efforts to have the standard set via ANSI. There have been 29 meetings of the committee called X3; the first 24 or 25 produced the standard which is soon to be the ISO and for which Raymond Tisserand, Alex Morrow and Clark Wiedmann were given the outstanding

achievement award at APL 86. Since then meetings have been doing preliminary work on a standard for extended APL. The chairman is currently Lee Dickey. Gene is now Recording Secretary of the ANSI group.

SigAPL Plenary Session

Marilyn Pritchard opened the session. SigAPL's officers are:

Chairman Marilyn Pritchard
Secretary Garry Helzer
Treasurer Andrew K Dickey

The budget is $45,000 p.a. and last year it was over-spent. The fund balance peaked at $65,000. Membership is now declining.

Linda Alvord spoke about halting the decline and asked for help in bringing back old members who had allowed their subscriptions to lapse.

Lee Dickey spoke about Quote-quad. Each year it produces three issues and the proceedings of the conference as a fourth. Recent editorial initiatives have produced some "monolithic" issues such as Iverson's Dictionary of APL. Quote-quad needs more letters - the 'year' contest produced the record response so far. It also needs a new products editor - any volunteers? So far 189 algorithms have been published and when the total reaches 200 Lee hopes to publish "200 Algorithms from Quote-quad". He ended by asking for contributions which are acceptable on floppy disk, via bitnet, usenet, csnet, edunet or you-name-it net or even on paper.

Lynne Shaw spoke about conferences. The next will be a workshop at Syracuse 15-20 August 1988. APL 89 will be in New York in July. IFIP 89 will be in August/September. APL 90 will be in Copenhagen. Conferences for '91, '92 and '93 are not finally fixed but '91 may be in Baltimore and '93 may be in Los Angeles.

Function Rank

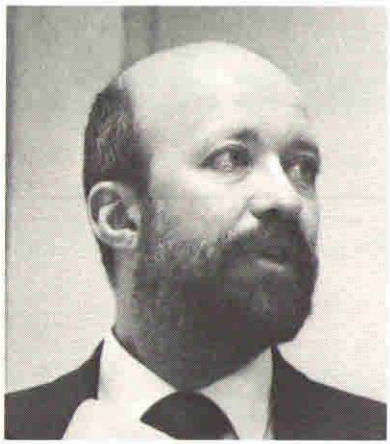
Bob Bernecky

This was for me (ACDS) one of the most enjoyable, and most frustrating, talks at the whole conference. The ideas Bob described are by no means new, but this was a superbly coherent exposition of just how you can use the Rank operator to simplify and improve your code. The examples he gave were all things I felt I could immediately take home and use.

The frustration arose from the fact that I have no chance at all to use the Sharp APL system in my day to day work! OK, I can play with their excellent public domain PC/APL, but I want this Rank thing, and I want it now!! Why did it make such an impact? I shall try to summarize as briefly as I can.

The first major insight was to see that the principle of scalar extension could quite easily be applied to functions of higher natural rank. For example Domino applies naturally to rank-2 arrays; if it is given a rank-3 array it should simply apply itself to each rank-2 subarray, and re-assemble (laminate) the results into a set of planes. For some functions, such as ravel, there is no natural rank, as it applies equally well to any array.

What the Rank operator does is to restrict it to the rank that you want. For example "ravel at rank-2" on a 2 by 3 by 4 array will yield a 2 by 12 result. It has ravelled the rank-2 arrays and re-assembled the result!



Bob Bernecky

Of course you can do the same with defined functions. Say you have a simple routine WCOUNT to count the words in a text vector; to count the words in each row of a text matrix you just do "WCOUNT at rank-1" to the array. It does each rank-1 object, and assembles the resulting scalars into a vector of results, with one element per line of the original matrix.

The thing I really liked about all this is that it works without any magic tricks using enclosed arrays. In other words you can use it to cut out loops, and generally to simplify your existing code, operating on your existing data-structures. See the APL88 proceedings on page 39 for a lot more examples.

Data Base Management with APL2

Stockbridge and Eisner

This talk discussed the transition from a home-grown database to the use of DB2 via AP127. It was interesting in the way the authors had 'rediscovered the past' in finding that simple arrays (one variable per column of data) offered a far lower storage overhead and a dramatic performance improvement, in the access to DB2 data.

They had expended a deal of effort getting round the SQL restriction that tables are updated one row at a time. Compared with the power of the SQL 'Select' command, the authors had found it really frustrating to resort to a sequential serial process in the APL workspace (rather than having AP127 take care of the looping internally).

Their solution to reducing the overhead was to define a separate update table, put the changes on this, then delete the affected rows from the base table and insert the new versions at the end. They had defined a detailed 13-step procedure, involving flags to ensure integrity at all stages, to control this. See APL88, page 314 for all the details. There conclusion was that it worked, and had saved a lot of CPU cost, but they would much rather not have had to do it!

Application-SQL Interaction

Stephen Deerhake

This paper was concerned with a way for the inexperienced SQL programmer to access DB2 effectively, and hence to isolate SQL from the APL application code. Typically it allowed APL to get data easily from 5 or 6 tables, fiddle about in the workspace, and reflect the changes back into the database.

The SQL knowledge needed was limited to 'Select ...', and the APL2 knowledge to arrays of vectors. As with the above paper, the interface handles the update problems for you, and the same comments apply "Insert and delete are easy .... update is a real pig!".

Again some limitations in SQL were apparent; there is no way to get a random selection of rows without first retrieving all the keys and then picking the randomly chosen rows with a second 'Select'; there is also no way to get the first (or last) 10 matching entries.

The final comment (in answer to a question) was that in a single user system, this all worked very nicely, but "if you get into a multi-user system things get real interesting - fast"

Plenary Session: The Present of APL

The purpose of this session was to bring out the differences in the various interpreters, and (more importantly) why they had been designed that way in the first place. In general I felt that the speakers achieved this aim rather well, and that this was one of the more successful joint sessions of the conference.

James Wheeler

James Wheeler (STSC and APL*PLUS)

STSC's objectives for APL are:

1. To provide a compatible APL system across most popular computing environments. IBM mainframe (TSO/VM); DEC Vax (VMS/Ultrix);IBM PC; Mac.

2. To build what the users want. They use 'focus groups' and market research to drive frequent updates of the system.

3. To supply a computing environment, not just the APL language. This means decent editors, file systems and good interfaces to the host machine. These are provided with quad functions, rather than APs, for speed, reliability, ease of coding and more readable programs.

They have moved to '2nd generation' features, but these will not be available under MS DOS or on the Mac. It is no longer the obsolete NARS experiment, but is now based on the same axiom system as Dyalog and APL2. They provide similar functions and operators, as well as the 'strand' notation. There are some differences in the detailed syntax of this, but the long-term plan is for convergence with APL2.

Jim Brown (IBM and APL2)

Some of IBM's objectives for APL2 were:

- all functions are equal. They can all be applied with operators, even FORTRAN ones.

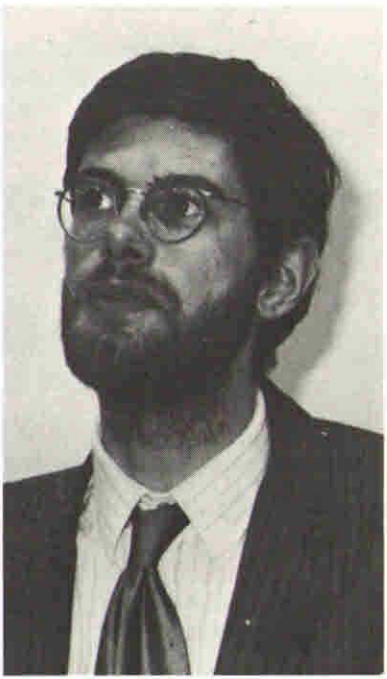
- you can assign whatever you can select.

- connectivity. You can use APL2 alongside all the other things on the system. SPF, DB2, FORTRAN and so on. Lets let all the people who like these things find out about APL!

- portability. SAA does not include APL, however there are experimental PC and RT versions of APL. No sign as yet of any interest from system 36/38 users

Jim suggested that the attitude of IBM to APL is continually changing; just at the moment they have found that APL sells machines. "That is why I'm taking a year in marketing!"

Jim Brown



Paul Chapman

Paul Chapman (Himself and I-APL)

I-APL is portable and small; it will run on the sorts of machines you find all over the world. The extensions to the ISO standard are minimal, except for direct definition which is built in to the product. Interfacing to the environment is kept to a minimum so that I-APL will run the same way on all machines.

Panel session on "APL Extensions in Perspective".

Panel: James Wheeler, Bob Bernecky, Paul Chapman,
Phil Benkard, Martin Gfeller, Eugene McDonnell

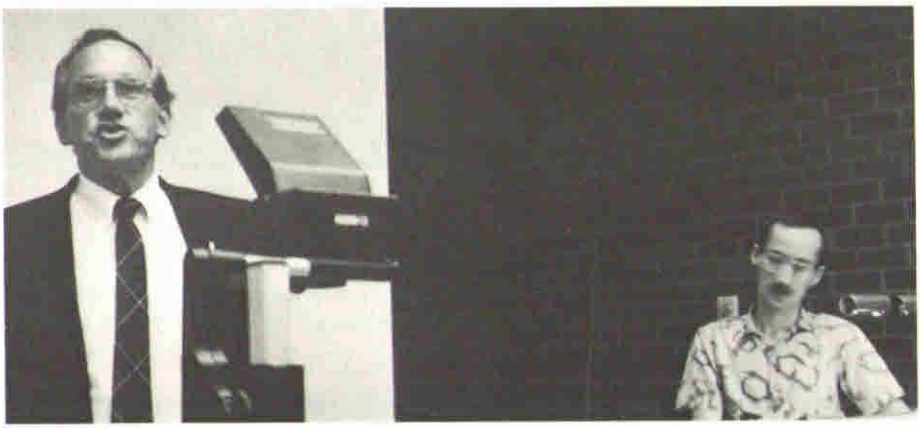
David Ziemann was in the chair and asked each member of the panel to make a preliminary statement about his objectives for his interpreter and his criteria for selection of extensions.

David Ziemann and James Wheeler

James Wheeler said that the ISO standard was OK for portable algorithms but not for portable applications. In order to be able to offer customers portable applications STSC had written their own internal standard. The extensions STSC had added were mainly to improve the power and ease of use for commercial programmers and the main route that they had chosen to take for non-structural enhancements was the quad function. The nested array enhancements were largely compatible with APL2. STSC were moving towards full compatibility with APL2.

Eugene McDonnell, Martin Gfeller, Philip Benkard and Paul Chapman

Bob Bernecky said that Sharp APL was ISO and ANSI compatible. Extensions to the notation were to improve its value as a tool of thought, for example the 'cut' operator. Sharp's main effort was put into changes which would bring benefits to their customers. Some of these were concerned with improvement of performance and with facilities for measuring performance and making faster ways of doing things accessible. Recent additions have been the line timer and provision for hybrid machine-code/APL functions. A second area of improvement is in the size of system that can be handled - to some of their customers a system with a thousand users on line at a time is small! Eugene McDonnell interpolated that Sharp also wish to include some of the good things from APL2.



Jim Brown and David Ziemann

So Jim Brown began by saying that he would like to include some of the good things in Dictionary APL, such as 'rank'. He spoke about the advantages of using PC implementations as front-end systems to the mainframe; programming development could be carried on without causing delays to mainframe work (or being delayed by it). He spoke of extending the number of uses for which APL2 was seen as a suitable language. For example there was to be a conference next August on AI and APL at Syracuse, where he would like to see us all. There are plenty of places where the current picture of APL2 is not complete and he is also exploring these, for example what would a nested left argument to 'reshape' do? The other area of possible expansion is arrays of functions about which there have been several papers at APL conferences.

Paul Chapman said that he was the only author of an ISO conforming APL interpreter who had used the ISO draft standard as his blueprint. He believed that I-APL was the only APL which could pass all the conformance tests. I-APL had to be conforming because it was intended for use in schools who would find an international standard product much more acceptable and might be reluctant to endorse any particular vendor's product by adopting it for educational use.

The enhancements to the standard he had included were therefore few and only those which were common and very likely to be agreed as part of the core of the next standard such as replicate and monadic grade of character vectors.

Direct definition was a conforming enhancement: it replaced what would otherwise be error messages. It had been included for its elegance and because so much educational writing about APL used it. He had also taken one enhancement out, after it was working and tested, because it would have made programs which used it incompatible with other APLs.

Paul Chapman

Phil Benkard said ..... your reporter was so fascinated he forgot to take any notes and it is now completely forgotten. Let this be a lesson: either issue a written account of what you say or put in some dull patches in which people can make notes. (Perhaps reporters should take tape recorders).

Martin Gfeller arrived late because, as he said, the Pacific had taken his glasses and he had had to get another pair at very short notice (half an hour) from an optician in George street. In Martin's opinion by the year 2014 we would probably not be using a version of

APL which was upwards compatible from what we have now. He thought we were a bit too satisfied with what we have.

Eugene McDonnell spoke about the past achievements and the current intentions of the standards group. He listed some of the proposals made to the group to show how much they had to consider. They had chosen to exclude anything which was not a consistent extension, not commonly available, not stable or which was machine dependent. The purpose of the group is to make programs and programming skills portable and to extend the applicability of APL. It is, as he said, too much work. The next meeting is at La Hulpe in Belgium in April 1988.

Some comments from the audience:

Maurice Jordan, implying that APL had fallen so far out of the mainstream that it was not even used for things it would be good at, asked "Why isn't relational database theory expressed in APL?" He didn't get any answer.

Jim Lucas felt that Maurice was unfair to the extensions of APL. They could do the job and the reasons they were not used were not related to the language standards. He also asked Bob Bernecky whether 'cut' was complemented by a 'paste'. Bob replied that it was and the function was called 'raze'.

Ray Polivka mentioned that at Minnowbrook there had been a transfer standard to allow APL code to be passed from one machine to another and asked if it was still alive. Bob Bernecky replied that the ISO standard has the extended transfer form as an appendix. Ray then asked whether all manufacturers would have it, and although not a direct answer, Eugene was able to tell him that it was approved in the committee by a vote of eleven to zero.

Tom Pritchard spoke about simplicity and felt that this was a desirable aim which some extensions seemed to rate not highly enough.

Jim Lucas complained that this was too simplistic. English has no case endings, Rumanian has cases: but which is simpler depends on arguable criteria.

Dave Ziemann had asked the panel members to supply him with questions which they would have liked to ask if they had been in the audience. He read out the following: "When are you guys going to standardise quote-quad input and output?"; "What are you doing to reduce strife between versions of APL?"; "When you consider an enhancement what does the enhancement have to be for you to include it?".

Both Jim Brown and Eugene McDonnell answered the last question; for Jim it had to be Dynamic and Array; for Eugene it had to be Parallel and Equipedal.

Rob Hodgkinson asked whether it was possible to use a common modelling system to allow each vendor to review other people's extensions. Was there, for example any chance of placing the Sharp, STSC or IBM development models into a software library?

Shadows Cast by Buildings.

Dr Warren Julian

Professor Julian has a long-standing interest in daylight and artificial lighting. He finds APL very suitable for this work - things like multiple reflections work out cleanly. He has used APL since 1972, introduced it to students as soon as it was available and has been a member of the Sydney APL Users Group since it was established.

He was disappointed that so few people were interested in specifying their own systems. The work on shadows grew out of his keenness do specify new work and his interest in lighting. The effect of a building on the microclimate of adjacent land is complex.

Warren Julian

In Sydney there are many new tall buildings each of which adds a new pattern of shade. The effect at different times of day and year all may be important when it comes to planning laws and appeals from neighbouring landowners. It is worth a lot off effort negotiating about about exact position, shape and size to avoid legal processes.

He reviewed how the system works. It distinguishes the shadows of interest in solid from incidentals which are hatched. He explained the way the printed functions work together. Since the system uses latitude and longitude, it is not restricted to Australian use.

The parts of the system which it was hardest to get working were those concerned with Macintosh facilities. The APL is complex but it is logical and consistent, lends itself easily to correction and gives good clues about what is wrong. Getting the Macintosh tricks to work robustly ought to have been easier but lacked these advantages.

Crystallography

Donald MacIntyre

Fortunately, the organisers of APL 88 had had the foresight to set a videotape running for this entire session. I hope we can successfully transcribe enough of it to make sense in a future VECTOR; otherwise we shall just have to twist Don MacIntyre's arm until he lets us have a written copy!

In essence, this was a marvellous paper, and was superbly delivered with the aid of numerous props (mostly assembled out of ping-pong balls). His use of APL to express many of the fundamentals of crytallography was wonderful to see. Just to give you a flavour:

> if you make an 8-row triangle out of ping-pong balls
> then the number in each row is `+\8⍴1`;
>
> if this is the plane of a close-packed tetrahedron
> then there are `+\+\8⍴1` balls in each successive plane;
>
> if you make a crystal out of successive tetrahedra,
> then there are `+\+\+\8⍴1` balls in each tetrahedron!

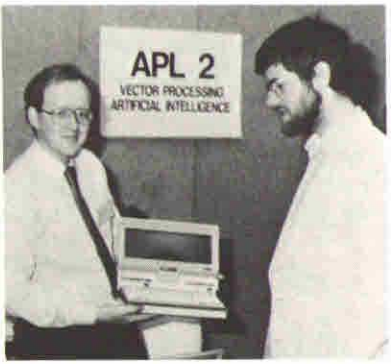
The tape of this talk is available from the APL 88 committee for the cost of copying and postage; otherwise we shall do our best to give a much fuller account in VECTOR 5.2. Watch this space.

Notes on the Exhibition

This was very much a home-grown affair. The IBM and STSC stands were the main vendor represention, while C.A. Read Associates and Lois Hill (Interactive Barchart Scheduling) were available to demonstrate their APL software packages.

From the general APL community point of view, the two most interesting stands were placed handily opposite one another: IBM running APL2/PC on a PS/2 Model-60; STSC running APL*PLUS/386 on a PS/2 Model-80.

The IBM offering may never see the light of day as a commercial product, so this was a fascinating chance to get in and 'have a go'. STSC on the the hand are committed to a mid-88 launch of the 386 product, so this was really a preview, rather than a once in a lifetime opportunity.



David Selby with Paul Chapman

The thing that was amazing everyone about the IBM offering (including the author of I-APL pictured above!) was how on earth you get a complete APL2 interpreter into roughly the same space as APL*PLUS/PC. The first thing I tried in a clear workspace was ⎕WA; sure enough back it came with 397136, so far so good! The basic interpreter speed (FN 1000 = 13 sec) was very comparable with other APLs on similar hardware; the ⎕AV looked pretty ASCII (although I noticed that many of the line-drawing characters had been reused for APL symbols); the session manager was very mainframe-ish (no type-ahead and a tendency to say 'INPUT' while it was still running).

Some bits of APL2 were missing, such as stranded assignment. It was also possible to do some rather odd things like '⎕IO←'A' 'B' 'C''; after much persistent trying David Ziemann finally managed to crash it with a depth-4219 enclosure of ,1. I don't think this represents a serious problem in real life! In summary, this had all the makings of a really good APL2 for small machines. If IBM choose to market it I'm sure a lots of mainframe APL2 sites will take it on as a development/training environment, and possible as a serious application delivery vehicle.

APL*PLUS/386 is a full implementation of the STSC mainframe APL product, with all the PC enhancements included. As its name implies, it is specific to the 80386 chip and uses the full memory addressing capability available. Typically you would run in

workspaces of 2 - 4 Mbytes. Upward compatibility from APL*PLUS/PC was promised, although most of the ⎕POKE commands were not implemented in the version on display.

There is very little I can say about this APL, except that it worked, and worked remarkable fast. FN 10000 ran in 27 sec (note the extra zero!) and floating point appeared to approach mainframe speeds. I can think of few mainframe applications which would not transfer to this environment and show a marked increase in overall execution speed. In particular it re-inforces my view (VECTOR 4.3 page 43) that the 'virtual workspace' of APL*PLUS/PC Vn7 is a temporary aberration, and that the 386 implementation is the real way forward.

Closing Plenary: A Commentary on APL Development

Ken Iverson

As Ken's paper was not included in the APL 88 Proceedings, we have asked his permission to print it in full in VECTOR; I am delighted to say that he has agreed to let us do this, with the agreement of the editor of Quote-quad who clearly had first refusal on all APL 88 related material. This way, you will have a chance to see the complete text as Ken wrote it; so there would be little point in filling up this VECTOR with extensive notes on what we think he said!

The main topics covered were: the proposed 'Yoke' operator, which produces the conjunction of two functions, and its use in expressing the common set manipulations; function arrays; the use of the 'box' function as a universal scalar encoding.

Closing Remarks: The Future of APL

Anthony Camacho

Most children are prone to asking their parents questions like "What are you doing, Daddy?"; maybe there is some significance in the fact that in the Camacho household the question was more often phrased "Daddy, what are you trying to do?!". Anyway, let's look at some of the things the APL community might be trying to do:

- recover old members. Obviously a valid activity, but a purely short-term aim.
- recruit new members among APLers. Another valid aim, but surely we wouldn't be happy even if we achieved 100% success.

- try to get the computing community to use more APL and less FORTRAN, PL/1 and the rest. Unfortunately we aren't going to get the DP department to replace COBOL, which from their point of view probably does a better job anyway. (At this point there was a vigorous interjection from the audience - Joey Tuttle - along the lines of "Stop apologising for APL!")

- increase the number of people using APL. What we need is people aged 20 who already have 10 years APL experience! There is only one way to do this; get APL widely used in education.

If you start by regarding APL primarily as a Programming language, then the fact that it also happens to be a notation gives you very little. On the other hand if you regard it first as a notation, then it is quite extroadinary. How many other notations have been used by so many people in the lifetime of their author? APL has one quite overpowering advantage - it can be executed directly on a computer!

Here then is one final thing the APL community might try to do - set about replacing conventional maths notation. When the 'Ant and Bee' books in the Infant schools have APL in them, we will have succeeded.

Postscript

At the final plenary session Neville Holmes paid tribute to the great efforts of Rob Hodgkinson, John Searle and Chris Craddock who had made APL 88 such a success. The audience obviously agreed. As the applause was dying down, Dave Weintraub remarked in a loud voice that he didn't think that Neville had been properly thanked yet.

Applause began, and went on and on as gradually the audience got to its feet to give Neville a spontaneous standing ovation. It was a good way to end.

TECHNICAL SECTION

This section of VECTOR is aimed principally at those of our readers who already know APL. It will contain items to interest people with differing degrees of fluency in APL.

Prize Competition: Rules

Entries must be in legible English or APL as appropriate and should preferably be machine produced.

Entrants must declare the type of computer and the version and release level of the APL interpreter on which any functions were written.

The date and the entrant's full name and address must appear on each sheet of the entry.

Entries should be physically separate from other contributions such as letters, and should be clearly marked "Competition Entry".

All submissions should be sent to the Editor.

Members of the B.A.A. committee, activities working group or journal working group are ineligible.

DOS format diskettes containing APL*PLUS, IBM or SHARP APL workspaces are acceptable; diskettes will be returned.

Unless otherwise stated, entrants should submit only one entry. We encourage submission of alternative approaches, but the entrants must indicate clearly which one answer is the entry in the competition.

Non-members of the B.A.A. are encouraged to enter the competition. If they should win, then part of their prize will comprise free B.A.A. membership for the current year.

Late competition entries may be accepted if the competition has not yet been judged.

Prize Competition: Missionaries and Cannibals

by Anne Wilson

Several years ago we entertained a visiting minister who had been a missionary in Papua New Guinea. Amongst the tales he told were ones of talking to old men who had been aware of their elders eating missionaries. The solutions to this problem are dedicated to him.

The scenario is familiar. There is a party of cannibals and missionaries travelling from one village to the next. The missionaries are safe so long as they are not outnumbered by the cannibals. Luckily there are equal numbers of cannibals and missionaries!

Between the two villages are several rivers which the party crosses by raft. These rafts are made by the cannibals, and their size depends on the trees in the surrounding jungle.

As the rafts can not usually accommodate all the people in one crossing the missionaries have to work out how many to send across on each journey of the raft. They obviously want to keep the number of journeys to a minimum because they are anxious to reach the next village quickly.

What the missionaries need is an APL program which will take as input the number of missionaries and cannibals, and the size of the raft. It will return a 2 column matrix showing the number of missionaries and of cannibals on the raft for each journey. At no place must the number of cannibals be greater than the number of missionaries, neither on the bank or on the raft.

The program will be of the form:-

```
RAFT number-of-people, raft-size
```

where the number-of-people is the number of missionaries, and the number of cannibals (as they are the same).

```
RAFT 2,3

      missionaries cannibals
                1          1
                1          0
                2          1
```

Technical Correspondence

Direct Definition

From: John Sullivan 16th Feb 1988

Direct definition seems to me an admirable vehicle for displaying algorithms in APL (which may be why conference papers use it so much), but as everybody who has ever written an application knows, applications contain far more than just algorithms. For instance, any function that operates on supplied data should contain all of the following three items:

1. Documentation (comments)
2. Data validation
3. Algorithm

and unless some convoluted logic is used it is pretty difficult to fit data validation and the algorithm into a one-line function. Add to this the need to keep function lines short (everybody in the BAA is exhorting me to do this these days!) and you are in trouble. And if you can't see the need for data validation, try executing FACTORIAL 2.5 using the function on p.100 of Vol 4 No 3. You cannot assume that the users are going to get the data right every time, and so you have to provide coding to deal with data errors.

Anthony Camacho's MAT (Vol 4 No 3 p.100) provides a good example of direct definition used to excess. This function is really only a utility, yet it consists of five directly- defined functions, some of which seem to do excess work for no good reason (see below for my version of MAT). Multiply that by the number of other functions you need to get even a simple application working (even if you can use some of the 'subroutines' more than once) and you will easily get 200 or more functions in your workspace. I well remember the reaction from the audience at the Royal Over-Seas League when the slide displayed on p.62 of Vol 4 No 3 was shown. It seems to me that we are being told to do two conflicting things here: reduce the number of names in the workspace in order to make maintenance etc. easy; and increase the number of names in the workspace for the sake of using direct definition.

I'm sure most of our members have workspaces of utilities, mainly because most APLers come packaged with at least one. As an example of these utilities look at Vol 3 No 4 pp.127-8 [1]. Would anybody even consider converting these to direct definition (or using direct definition for them in the first place)? I wouldn't: I've got too much work to do. Having said that, I'm sure there are occasions where a good set of directly-defined functions could come in handy: for example, a suite of data-verification functions on the lines of those shown at APL86 by Alan Graham.

All of this direct definition mullarkey reminds me of an article I read some years ago [2] which attempted to give the reader an introduction to LISP. I know that some people are trying to write AI applications in APL, but that is no reason for writing APL in the style of LISP.

As for writing MAT in an elegant way, here is my four penn'orth.

1. Get the right algorithm. Repeated screen input needs a (expletive-deleted) loop, but you shouldn't do ANY processing inside the loop.

2. You've got a workspace of utilities, so use them. The utility for converting a segmented string (delimited vector) into a matrix is so well known it needs no introduction. It comes in a variety of flavours, and the one I am using here (ROWNAMES) appears in the standard APL 'textbook'[3]. This function has a left argument that defines the rowlength of the final matrix (use the null vector for a default), which can be used to make MAT monadic.

This example uses direct definition as far as it goes because that is the reason for its existence.

```
MAT:    ⍵ ROWNAMES GETSCR ''
GETSCR: GETSCR ⍵,⎕AV[⎕IO],V : 0=⍴V←⍞ : ⍵
```

Again, this suffers from the lack of a data validation routine (try MAT 2.5) but this could be overcome by making MAT pseudo-niladic and replacing ⍵ by (⍳0). More elegant than Anthony's code? Maybe, but in a production environment who cares about elegance? This code is 'better' than Anthony's because it removes all but the barest essentials from the loop (recursion is looping with the mechanism changed to confuse the innocent!), but it's probably not the best that could be written. The question remains, is it worth it? Why waste half a week being ever-so-clever when you can get the results in half an hour using other methods?

To sum up: direct definition is a useful addition to the tools of the APL programmer, but it is not the be-all-and-end-all. An excessive desire to see everything coded via direct definition can lead to a waste of time and effort.

References

1. David Piper, 'Using Name Association for Data Transfer', VECTOR Vol 3 No 4.

2. Douglas Hofstadter, 'Metamagical Themas', Scientific American Vol 248 No 2 (February 1983) p.14.

3. Leonard Gilman & Allen J Rose 'APL, an Interactive Approach', John Wiley & Sons, 3rd. edition, 1984.

Yours faithfully,

John Sullivan
Research Manager,IT Systems & Library Support
Market Intelligence Dept, Business Development Div.
National Westminster Bank
LONDON EC3V 3NN

APL 88 Code Quality

From: David Piper 15 March 1988

I have just been reading the proceedings of the APL88 conference and have been made extremely angry at the quality of some of the published code. In today's systems development environment, when conpliance to standards and quality of code is given the highest priority, how can the APL community ever hope to be taken seriously when it publishes such appalling code examples?

Superficially it is reasonable to argue that there are no standards in the APL world; there are, however, widely agreed guidelines for writing APL. The following are just a few that are broken:

- Widespread use of global variables
- Use of execute (even worse with 'IF')
- Pepper-wise use of EACH
- Non-use of arguments and explicit results
- Lack of comments (especially descriptive comments)
- Gluing of lines of code

The scale of these violations is indicated by the following example, analysing the code of a single operator:

- 10 uses of execute (6 with 'IF')
- 2 occurrences of output suppression, using

```
→0⍴expression
```

- 4 occurences of line gluing by strand notation, using

```
→0 expression
```

- 2 global variables

Or again, in a single function:

- 20 lines of code
- 13 global variables set
- 38 uses of each (up to 12 in a line, 4 together)
- No arguments
- No explicit result

Two major problems arise from the publication of such bad code. Firstly, the perception of APL as a 'gauche' language is perpetuated amongst those on the fringes of the APL community who perhaps disapprove of it anyway. Secondly, and I believe more importantly, it encourages new users of APL to adopt the same habits, thinking them to be acceptable practices.

As a solution, I would like to suggest that papers submitted for publication in journals and at conferences are scrutinised more closely, with equal weight (or even more weight) being given to the quality of the code and the content. In order to make the authors work easier, a set of general coding guidelines coul be published. This would allow the assessment of code to be automated, papers violating the acceptable metrics being rejected. Even more helpful would be a report sent to each author (even of accepted papers) showing bad points in their code.

I would like to stress that not all was bad in the APL88 proceedings. Many of the papers and much of the code was of a high standard. Some even presented hope for a more professional APL methodology. Notable here was the paper by Bob Bykerk describing the development toolbox developed for use within the ECC.

In future, let's have more of the high quality code and less (preferably none) of the bad.

Yours faithfully,

D.B.Piper
41, Sandown Drive
Rainham
Kent ME8 9DT.

In Defense of Dyalog

From: Iain Hayward 31st March 1988

Concerning your review of Dyalog APL on the IBM 6150 in VECTOR 4.2, I feel it necessary to put the record straight for the benefit of prospective users of Dyalog APL, if not just out of fairness to the suppliers.

My attention was first drawn to this article by an indignant colleague who took exception to your description of the 'vi' editor. Now, I wouldn't argue with that because you were expressing an opinion based on limited experience of Unix, however further reading did provoke me.

Firstly, you complained about the excess of system functions - how can you say that! It seems to me that Dyadic have kept these to a minimum, preferring instead to allow you to make your own, either by executing a Unix command from within APL, or else by writing an auxiliary processor (just how much time it takes to write an AP is another story). If you had taken the trouble to get to know the ones that they have supplied you might have realised how valuable they are. For example have you thought how useful the ⎕NR (nested representation) would be when writing a documentation system? As for ⎕VR, it was the first function I had to write when I took delivery of MicroAPL's APL.68000, just to be able to produce a satisfactory listing of functions on the printer.

Secondly, you complain that the 'external variables' facility only serves to clutter the documentation, and imply that the conventional component file system is quite adequate. I for one think that the conventional component file system is the pits! Dyalog APL lets you use real files, ones which other systems can read and write to. Overall Dyalog APL's facilities for transferring data to and from the outside world are superb and unparalleled. APL will never be widely accepted without this capability.

The impression I get is that you didn't have time to get to know this product and that you reacted in the way that many a beginner does when faced with a new, large, and complicated system - complain that it is too different and criticize the manual.

Yours faithfully

Iain Hayward
9 rue de Clairefontaine
1341 Luxembourg
Grand Duchy of Luxembourg

In defence of VIA ...

From:Bob Bykerk 30th March 1988

Dear Sir,

Referring to Dr Peter Branson's letter in VECTOR 4.2; to remove all the duplicated rows from a matrix, he should try the following:

```
(1=+/R∘.=⍤R)⌿R
```

Adrian Smith's review of Dyalog APL was interesting, but I think it is unfair to print scathing criticism of a feature of a product that the author admits to having little experience with (i.e. the VIA editor).

I suggest that he takes a little more time to learn a product before he criticizes it. There is one thing that can be stated: the VIA editor is a different approach to editing, and can be AT FIRST difficult to master, especially after using other editors such as)XEDIT, SPF and ∇EDIT. However, after the short initial learning time, this editor can be very powerful indeed, and certainly becomes intuitive. It is only a shame that there are not more features of the Unix 'vi' editor available.

Yours sincerely

Bob Bykerk
55, Rue Charles Arendt
Luxembourg

Editor's note:

sorry via! It bit me rather badly, to the tune of around 15 minutes lost work, quite early on. I'm afraid I never forgave it sufficiently to have a really serious go. In fact with the Dyalog session manager ∇ is really quite adequate, much more so than on most other systems.

I stand by my comment that you don't need three different ways to convert functions to characters. If you have ⎕CR you can easily program ⎕VR and ⎕NR and so on! APL used to be all about orthogonality; it upsets me to see such a high degree of overlap between supplied functions.

Maybe I'm just getting old and crabby! Maybe I'm just upset that no-one will give me a Unix box to play with? Please keep the letters coming - anything except direct abuse gets published, honest.

From Dyadic Systems ...

From:Peter Donnelly 11th April 1988

Thank you for your interesting review of Dyalog APL in VECTOR 4.2. I would like to make one or two comments about the article, which your readers might find informative.

1. Dyadic chose Unix because it offered portability, speed, multi-user support and unlimited memory. Just the things for an effective APL environment. I agree with you that there is resistance to Unix, but there is always resistance to change, particularly when so many vested interests are involved. Today, virtually all of the major hardware developments in computing (RISC, parallel architecture, vector-processors) are Unix based. In my opinion, DP managers who choose to ignore Unix are paying through the nose for their MIPS!

   Yes, DEC may continue to 'lock users in' with VMS, but Hewlett-Packard, NCR and Unisys are firmly committed to Unix, as is IBM. IBM recently claimed that it spends as much time on its Unix product AIX, as on any other of its operating systems, and underlined its long-term Unix strategy with the announcement of the 'AIX Family Definition' in parallel with SAA.

2. You certainly don't like out editor, do you! Rightly or wrongly, Dyadic decided to emulate the standard Unix editor 'vi'. This has the advantage that users only have to learn one set of editor commands; not one set for APL and another for the rest of the system. I agree that 'vi' can be difficult at first, but like many other powerful tools (e.g. APL?) it more than repays the effort of learning. Lots of our users swear by at.

3. We too can provide instantaneous screen response. Dyalog APL/386 on an IBM PS/2 Model-80 or a Compaq-386 is every bit as fast as APL*PLUS/PC in terms of screen handling. The particular screens you used in your evaluation are also instantaneous when used in full-screen or graphics mode. They are however comparatively slow at scrolling. I expect that this was your problem.

4. I have to disagree with you about the error trapping. Having used many different versions of APL I am convinced that Dyalog APL's error trapping is more powerful and easier to use than any other I have encountered.

   For example `⎕TRAP←9 'E' '→ERR'`

   means that if a DOMAIN ERROR occurs in my function, the program will branch to ERR. Surely that's simple enough!

   If I say `⎕TRAP←0 'C' '→ERR'`

it means that if any error occurs in a function called by this one, the stack will be 'cut back' automatically to here before executing the branch.

Using these two cases in combination, you can quite simply set up very comprehensive error handling, without having to resort to complex lexical analysis of ⎕DM.

Dyalog APL also has ⎕SIGNAL, which is very similar to ⎕ERROR, but that's all that's needed for error-handling. There is no heap of obscure ⎕FNS.

5. We introduced ⎕NR because a vector of text vectors is so much easier to handle than either a matrix or a line-list. We retained ⎕CR and ⎕VR for compatibility with the ISO standard, and with other APLs.

6. Really Adrian, there are only 54 system functions, variables and constants in Dyalog APL (including 20 for the component file system) compared with 144 (a gross) in APL*PLUS/PC. I don't think it is Dyadic who should be accused of taking the ⎕KITCHENSINK approach!

7. Yes, you're partly right about modified assignment. Everyone who uses Dyalog APL (including myself) falls over the same thing, i.e. that the 'pass through' values of modified assignment is the thing on the right, not the result of the assignment. The problem is that what is theoretically correct is (initially at least) intuitively wrong. Incidentally, we were not the first to implement this feature; Burroughs APL had it too.

9. You will be please to know that since your evaluation, we have enhanced our graphics APs to take full advantage of nested arrays. Basically, the output functions now accept nested arguments so you can draw more than one object at a time. This means that you can avoid writing loops, so your graphics applications run faster. The most complex of the demo charts you tried is now calculated and displayed in under 3 seconds.

10. I am glad you liked our interface to Oracle. This seems to be very popular with users. As you noted, you can improve on the standard database management facilities by writing your own tools in APL. Several of our users have already done so, and one customer has even built a fully relational IC1 look-alike.

Finally, a note on performance. The IBM 6150 Model 20 you were using was superseded last year by the model 125. This is 2 - 4 times faster than the old model, and outperforms the Sun 3/140.

Yours sincerely

Peter Donnelly
Operations Manager, Dyadic Systems Ltd
Park House, The High Street,
Alton, Hampshire GU34 1EN

Dyalog APL on the IBM 6151: A Reply

From: Allan Prys-Williams 8th April 1988

As someone who has been using this system in workstation mode for eighteen months now, perhaps I could be allowed to expand on Adrian Smith's review in Vector 4.2. Obviously, my view is coloured by my coming to this system from mainframe rather than PC use, but I thought that his review was unfairly critical in some ways, while missing one or two actual problems.

Unix

Obviously, any system that gives real flexibility to the user is going to frighten DP managers, but the AIX implementation is already based on a hierarchy of users.

- The superuser can do anything, but no-one logs in as superuser except for specific purposes. It is too dangerous even for the administrator to use routinely.

- The system group can do many things that are fairly dangerous, such as altering the status of printers. On my system, I normally work within this group. On a DP-manager-friendly system, most commands would be moved to have this restriction, and only the managers would belong to it.

- Ordinary users only need a very small set of commands, especially if their profile causes them to wake up in APL. On my system I have a 'demo' user configured so that colleagues can try things out without causing any damage.

I have to say that every time I find myself on e.g. a DEC mainframe I can't wait to get back to UNIX. Only, don't let yourself get posted as the superuser of any UNIX system that doesn't have explicit superuser's manuals. UNIX provides a friendly environment for ordinary users by loading all the awkward bits onto the superuser, which can be terrifying. The IBM6150/1 has good manuals, but the Cadmus I was on before had not.

Editors

The 'vi' editor is pretty horrible, but the really good full-screen editor and file manager provided within AIX - 'INed' - is not compatible with APL as it uses the 'alt' key to control editing functions. Indeed, if you start this up after running APL from the console, it won't

work at all. In practice, I have found that using the full-screen cut and paste functions in the session manager, combined with the del-editor, is excellent for generating new functions, as you can test code in bits and then capture it into the function. Vi works rather well for documenting things and going through changing names of variables and things like that, so one ends up using both. Many of the session-managing keys work within vi, though the manual omits to say so. At least I have been able to give up using things like '///4 A' and other del-incantations.

The APL

External variables are marvellous. Of course, they are only high-class component files (as you find out if the system goes down at the wrong moment) but they have the tremendous virtue of transparency. For database work, I can develop the system using just basic APL and then convert any variables that are going to be changing, into external variables of the same name. These are then sharable; backed up independently; enable me to keep the workspace absolutely fixed; and I don't need to know where (exactly) everything is. It is also, in emergency, trivial to bring them back into the workspace as standard variables (a single left-arrow does it) when they can be hacked around ad. lib.

The ⍝KITCHENSINK comment is really an attack on the documentation. The different ways of representing a function all do non-trivially different things, but you have to work out the crucial differences for yourself. Personally, I find that the presence of minority-interest goodies makes it much easier to get help from Dyadic over the phone at the practically important level of things like interfacing with Unix files or printers.

Windows

In workstation mode, it is practical to have separate full-screen windows open on several different jobs (the operating system for when something goes wrong; the APL under development; the editor for writing it up; a routine data-handling job that has to be done at the same time) and flip between them. This makes it practical to document what you are doing as you do it, instead of as an afterthought. Not only does this save a lot of hassle near deadlines, but you get new insights by trying to explain to an unknown reader what you are up to, and then just flip back to the APL window to try them out. The INed editor allows windows into 3 or 4 files at once with text capture from one to another.

Conversely, it is most desirable to run APL in a window, not the main screen, so that you haven't disabled the alt-keys when you run your next Unix application. You can even run APL inside the mouse-driven Usability Services shell, though I find point-and-grunt systems dreadfully slow.

Personally, I have only noticed delayed response when the machine is definitely busy elsewhere (multi-tasking plus a virtual system means that you can run monstrous background jobs while thinking what to do next). This is a function of a single chip doing many jobs, and as the 386 machines come in even PC users are going to have to get used to it. Certainly the Prefect full-screen I/O utilities run with exemplary efficiency, with the only delays being caused by heavy data-crunching.

But...

There are quite a few I/O problems. While direct communication with the Proprinter is exemplary for sending character arrays, capturing APL session output is hairy. If you split the output to a monitor file (the standard Unix technique) the result doesn't have carriage returns at the end of lines and tends to print continuously across the page. Personally I use monitored material inside files sent to the 'mm' word-processing package, which gives full pagination and scope to comment, as well as eliminating unprintable characters and adding CRs, but this is the sort of elaboration that gets Unix buffs a bad name.

Similarly, the console doesn't act as an APL terminal at all. If you view a file with APL characters, they aren't translated. So if you have a monitor file going, all you get is ASCII on the screen and no session-manager. A terminal emulator is supposed to be on the way, but it will be just my luck if it works splendidly with the VT100 emulator, but not with basic paging through a file.

Summing Up

My unimproved workstation is vastly better than the University's new Pyramid mainframe with umpteen MIPS (and no slower, in real life) and I will fight anybody to keep it, but some of its idiosyncrasies seem to demand a dedicated user. However, this also applies to every PC I've ever observed.

Allan Prys-Williams

Department of Management Science and Statistics
University of Wales
Swansea.

Dissembling

by Allan Gay

If, like me, you find some of the APL examples in Vector tough going, you might enjoy this account of what happens when high-level APL meets a low-level Assembly Language.

There's no doubt that there's a slightly loony flavour to the application described. Please bear in mind that it was thrown together in less than an hour, used briefly and, having served its purpose, jettisoned unceremoniously.

Recently, I had occasion to write some System/370 Assembler. A client required a routine to convert IBM mainframe TOD clock values embedded in accounting records to something printable.

The TOD - or time of day - clock is simply a 64-bit counter, the bits being numbered from 0 on the left to 63 on the right. The clock steps at a rate equivalent to incrementing bit 51 once per microsecond. A zero value corresponds to 00:00 hours GMT on January 1st 1900. The clock will wrap round to zero again after approximately 143 years. Come to think of it, there's probably a Vector competition in there somewhere.

I used MSDOS Edit on my Olivetti M21 to prepare the System/370 Assembler program statements. I knew the client had an IRMA card, so I'd be able to pump the code through to his IBM mainframe from a diskette. The trouble was that I couldn't run a System/370 assembly on the 8086-based Olivetti to verify statement syntax and I couldn't test the program either.

I had APL*Plus PC Release 6.2 and, at first, I toyed with the idea of programming the algorithm in APL. Cogitation revealed that this would prove almost nothing. The APL code would be at such a high level that it would have little in common with the Assembler solution. It was evident that the real issue was the detailed mechanics of the Assembler program; would I, for example, accidentally overwrite a register value which I needed later? Besides, APL*Plus PC already provides the FTIMEREP function for date conversion. Writing another FTIMEREP would prove nothing.

FTIMEREP would, however, be useful for checking results. Admittedly, the TOD clock value is not in microseconds but I could get my routine in sync with FTIMEREP by inserting a 12-bit register shift in the real Assembler code after I'd verified it. What's more, APL*Plus PC's WSTS quad function would be a useful source of testdata since it yields the workspace timestamp in microseconds.

The next idea was that I code a simulation of the System/370 Assembler in APL to produce some fake object code. I could then write some sort of executor to gallop through this invoking APL functions which simulated individual System/370 instructions such as Divide, Load Address and what-have-you.

That would have been fun but I feared the disapproval of my more favoured colleagues who have had the benefit of C&D's System Design Course. There would have been some dreadfully infra dig module couplings in such a system. If I were not to be debagged by a screaming horde of outraged purists I would have to be very circumspect. For Heaven's sake don't tell them I thought of doing such a thing.

The fact that the Assembler manual runs to over 400 pages had absolutely nothing to do with my decision to abandon this idea. For the time being.

In the end, of course, I cheated. I wrote a function to simulate the System/370 Divide instruction and transliterated the relevant Assembler statements into a crude APL function which called it. Another function was then placed atop the lot to exercise the code, convert the resulting date via a utility function which I already had to hand, and report the converted date and the timestamp.

To simulate System/370 registers, I just used APL integer variables. Because I was going to throw the workspace away after use, I didn't bother to localise them. If I needed a scalar, I'd just wang a vector into line without bothering why it was a vector in the first place.

When I ran the DATETEST function, to my infinite gratification, I got an incorrect result. It turned out that, although the Assembler code correctly implemented the algorithm, the algorithm was faulty. Without going into the humiliating details, I'll just confess that I had failed to distinguish between cardinal and ordinal numbers. There's probably another Vector competition in that, too.

Finally, we come to the APL code itself. When I reviewed it today, I formed the view that code of such surpassing rambunctiousness should not be permitted to foul the august pages of Vector. For this reason, I'd have preferred to leave it as an exercise for the reader. On the other hand, bad code can have a heartening effect. It gives courage to beginners and makes experts split their sides.

The System/370 Assembler code has yet to be tested on a mainframe, so don't accept the extract below unreservedly. Here goes, and don't say I didn't warn you ...

```
      DATETEST ⎕WSTS ⍝          SAMPLE EXECUTION OF THE APL SIMULATION.
02OCT87 14 5 25.800000
      FTIMEREP ⎕WSTS ⍝          A CROSSCHECK ON THE RESULTS.
1987 10 2 14 5 25 800

    ∇ R←DATETEST MS;⎕IO;⎕IO
[1]   ⎕IO←1 ⍝                          USE ORIGIN-1 IN ALL FUNCTIONS.
[2]   ⎕IO←MSCONV MS ⍝                  RUN THE SIMULATED CONVERTER.
[3]   ⎕IO← 2↓⍕ 1000 1000 ⊥⎕IO ⍝        GET YEAR & DAY AS 'YYDDD' TEXT.
[4]   R←UTTSVC ⎕IO ⍝                   CONVERT 'YYDDD' TO 'DDMMMYY'.
[5]  ⍝                                 APPEND TIME OF DAY TO RESULT.
[6]   R←R,' ',(⍕ 60 60 60 T(⍳0)⍴R9),'.',⍕R11
    ∇


    ∇ R←MSCONV MS
[1]  ⍝ MICROSECONDS CONVERTOR - SIMULATED SYSTEM/370 ASSEMBLY LANGUAGE.
[2]  ⍝
[3]   R7←MS ⍝                  MICROSECONDS IN.
[4]   D 6,1000000 ⍝            R7←ELAPSED SECONDS.  R6←MICROSECONDS OVER.
[5]   R11←R6 ⍝                 R11←MICROSECONDS OVER.
[6]  ⍝
[7]   D 6,86400 ⍝              R7←ELAPSED DAYS. R6←SECONDS OVER.
[8]   R7←R7-365 ⍝              (FOR 1900).
[9]   R9←R6 ⍝                  R9←SECONDS OVER.
[10]  R7←R7+1 ⍝                CHANGE ELAPSED DAYS TO DAY NUMBER.
[11] ⍝
[12]  D 6,1461 ⍝               R7←ELAPSED QUADYEARS. R6←DAYS OVER.
[13]  R10←4×R7 ⍝               R10←ELAPSED YEARS IN THE QUADYEARS.
[14] ⍝
[15]  R7←R6 ⍝                  R7 IS DAYS OVER (A PART-QUADYEAR).
[16]  D 6,365 ⍝                R7←YEARS OVER. R6←DAYS OVER.
[17] ⍝
[18]  R10←R10+R7+1900+1 ⍝      R10←FINAL ADJUSTED YEAR NUMBER
[19] ⍝                         (+1900 FOR ID.  +1 CONVERTS FROM ELAPSED).
[20]  R←R10,R6                 2-ELEMENT VECTOR RESULT: YEARNO AND DAYNO.
    ∇


    ∇ D NN;REG;BY;EVEN;ODD
[1]   REG←1↑NN
[2]   BY←1↓NN
[3]  ⍝
[4]  ⍝ SIMULATE SYSTEM/370 EVEN/ODD REGISTER-PAIR DIVISION
[5]  ⍝ ('REG' IS EVEN-REGISTER-NUMBER.  'BY' IS DIVISOR')
[6]  ⍝
[7]   EVEN←'R',⍕REG ⍝          DEVISE NAME OF EVEN-NUMBERED REGISTER.
[8]   ODD←'R',⍕REG+1 ⍝         DEVISE NAME OF ODD-NUMBERED REGISTER.
[9]   ⍎EVEN,'←',ODD ⍝          START BOTH WITH A COPY OF THE INPUT.
[10] ⍝
[11]  ⍎EVEN,'←BY|',EVEN ⍝     THE EVEN REGISTER GETS THE REMAINDER.
[12]  ⍎ODD,'←L',ODD,'÷BY' ⍝   THE ODD REGISTER GETS THE QUOTIENT.
    ∇
```

APL CONSULTANTS

LONDON & READING

Account Managers	(6 years+)	to	**25K**
Senior Consultants	(4-6 years)	to	**21K**
Consultants	(2-4 years)	to	**17K**
Junior Consultants	(1-2 years)	to	**13K**

Are your APL skills and potential being recognised and rewarded?

Cocking & Drury consultants have been implementing successful decision support applications for 10 years, with clients who appreciate the productivity benefits of APL.

In our professional team you will experience a range of APL environments - APL*Plus, VSAPL, APL2 and Unix, on both mainframes and micros. You will also be developing systems which, increasingly, need to interface with non-APL Information Centre products.

Of course as the leading APL consultancy, in a rapidly expanding market, we offer a rewarding career with first class benefits - profit sharing, free health insurance, and a non-contributory pension.

For further details call Ralph Wilson on 0734 588835

COCKING & DRURY LTD.

155 Friar Street, Reading, RG1 1HE

Using Quad-NA to Eliminate Workspace-Full Problems

by David Piper

Introduction

Much has already been written about using quad-NA to speed up APL2 function code. A lesser known (indeed undocumented) series of recent enhancements have been implemented which allow workspace size problems to be tackled using quad-NA. These enhancements are unsupported in the current release of APL2 (APL2 V1.R2).

Using Quad-NA

The Quad-NA system function is used to establish linkage with an object outside the APL2 workspace. Typically this object is a function coded in a language other than APL2. The right argument gives the function name with which linkage is to be established. The left argument specifies the name class and attached processor through which the linkage is made. The result is a boolean scalar, 1 if the association is established successfully, 0 otherwise.

```
      3 11 ⎕NA 'FOO'          ⍝ Associate with function FOO
1                             ⍝ Linkage established
```

The clue to the newly implemented features lies in the left argument. By changing the specified name class, objects other than functions can be associated with:

```
      4 11 ⎕NA 'OP1'          ⍝ Associate with operator OP1
1                             ⍝ Linkage established
```

It must be emphasised at this point that the associated object takes only a minimal amount of workspace. When an association is formed with a function or operator, only about 12 bytes of workspace are used. This memory overhead is used to establish pointers to the correct code outside the workspace.

Saving Workspace

The largest objects in a given workspace are normally the variables to be used during processing. Clearly, opportunities for saving workspace will be maximised by storing variables outside the workspace, then associating with them as required:

```
      ⎕WA                  ⍝ Current work area
1987103
      FIRST                ⍝ Reference non-existant variable
VALUE ERROR
      FIRST
      ^
      2 4 ⎕NA 'FIRST'      ⍝ Establish association
1
      ⍴FIRST               ⍝ What is its shape
200 80 100
      ×/⍴FIRST             ⍝ Number of items
1600000
      ⎕WA                  ⍝ Work area with variable
1987091
```

Note that in the above example, only 12 bytes of work area are used to store an array containing 1.6 million items.

Just as in the examples above with associated function, the associated variable acts exactly as a normal APL2 variable. The only difference is that the variable is stored externally to the workspace. Associated arrays can of course be nested:

```
      ≡FIRST
5
```

Existing variables can be placed external to the workspace simply by associating the variable with the associated processor after it has been assigned. Once associated, any assignment to the variable updates the associated version rather than re-creating a copy in the workspace.

```
      ⎕WA
1987091
      SECOND←(10 8 6⍴480?1000)(15 5⍴'ABCDE')
      ⎕WA
1985056
      2 4 ⎕NA 'SECOND'       ⍝ Externalise the array
1
      ⎕WA
1987079
```

The space saving that can be achieved using associated variables is obviously tremendous. The enhancement to Quad-NA which implements this facility is referred to as the Fully Integrated Reserved Storage Table (FIRST).

Saving More Workspace

Now that functions, operators and variables can be placed external to the workspace and accessed via name association, the only memory usage within the workspace is that required by the interpreter. In order to minimise memory usage, quad-NA has been further enhanced to allow the storage used by the interpreter to be stored externally also. These areas of storage are then accessed via name association.

```
      )CLEAR
CLEAR WS
      ⎕WA
2074824
      ¯1 4 ⎕NA 'APL2'
1
      ⎕WA
3187688
```

Note the use of a 'dummy' name and name class on the call to quad-NA. The use of the name APL2 causes the entire interpreter storage to be removed from the workspace. Using other names allows different portions of the interpreter to be removed:

```
      ¯1 ¯4 ⎕NA 'APL2'      ⍝ Get the interpreter storage back
1
      ⎕WA
2074824
      1 4 ⎕NA '⎕FNS'        ⍝ Only system functions moved
1
      ⎕WA
2215580
```

Note the use of the negative processor number in the above example. This is used to dissolve the name association, and relocate the interpreter storage back within the workspace.

The enhancement to Quad-NA which allows the interpreter storage to be moved out of the workspace is referred to as Associated Processor Remote Interpreter Linkage (APRIL).

Using No Workspace At All

Once all the defined objects have been removed from the workspace, and the interpreter storage is also relocated, the only remaining use of memory in the workspace is for the associated object definitions (12 bytes per object).

A third enhancement to quad-NA allows even these headers to be moved out of the workspace. Referred to as Full Operation Off-Line (FOOL), this allows your APL systems to run without using any memory at all.

Conclusions

The three enhancements to quad-NA described (FIRST, APRIL and FOOL) allow applications to be written with only very small memory overheads. The ability to store variables outside the workspace reduces the required workspace for a given application.

Memory usage is reduced still further when the interpreter is stored outside the workspace. If the quad-NA headers are stored outside the workspace, memory usage vanishes in a SPOOF of smoke.

Efficiency in APL

by Dave Ziemann

Introduction

This article is based on a presentation with the same title given at a meeting of the British APL Association in 1987. It is by no means intended to be an exhaustive study of the topic of efficiency in the APL environment, but rather a collection of ideas from many sources suffused with some recent work experience optimising APL systems. I apologise for any mistakes or omissions, and welcome any correspondence on this subject, either through the pages of this journal or personally at the Cocking and Drury London office.

First, let me briefly review the contents of this article, so that you can decide now if you want to read it or not; we will be covering a number of areas: after a brief look at what 'efficiency' could possibly mean, and why it is relevant, a review will be made of the tools available for locating problem areas in APL systems. We will then look at some sample techniques that can be used to optimise those systems. I also hope to demonstrate the value of 'knowing your interpreter' in the optimisation effort.

What is Efficiency?

When I was asked to give this talk, it seemed clear what the subject matter was, but then of course I started to think about it, and all was lost. In a vague attempt to regain my footing I reached for a pocket dictionary (not Ken Iverson's); apparently 'efficiency' is the 'ratio of useful work done to energy expended'. First we have to decide what the object of efficiency is. It could be the development of a software system, the operation of the system, the execution of the system and probably others too. For the purpose of this article let us take the restricted view, and agree that what we are primarily interested in here is program execution efficiency.

We should also assume that the program does in fact do 'useful work', although as we will see later the removal of redundancy is a valid optimisation technique. What 'energies' are expended then, during program execution? Certainly electricity is consumed, and possibly also such human resources as are required by the program's interaction with its environment. Again, let us take a restricted view; most of us do not concern ourselves with the quantity of electricity that our beloved programs consume (although there were rumours that during the early days of development of STSC's APL compiler the lights went dim all over Rockville). What we are concerned with are the time and space opportunities which once lost are gone forever. In APL application terms we are talking about:

- CPU time
- Workspace storage
- Elapsed time
- External storage

We are probably all familiar with the CPU time and workspace storage resources. Elapsed time and external file storage considerations can also make or break an APL application - these are often determined by higher-level design choices as well as by questions of run-time efficiency. The main emphasis of this article will be on the first two, and in particular on minimising CPU time.

Why Make Programs Efficient?

So now we have arrived at what was probably obvious to you all along - that reducing CPU time increases program efficiency. But before we get stuck into the nitty-gritty of optimisation, it is worth looking briefly at why optimising a program may be worthwhile. Actually, efficiency can be seen as just one out of a whole set of 'program quality characteristics', each of which contributes to the overall quality of the program.

Figure 1 puts efficiency into this context.

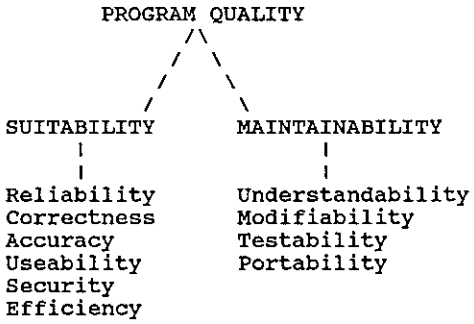


Figure 1: Progam quality characteristics

So increasing efficiency is just one way of increasing the quality of a program. The catch of course, is that many of the quality characteristics compete, i.e. increasing one may cause others to decrease. Your program is probably too efficient once you can no longer understand it. (You may have had the experience of asking a colleague why a particular piece of dense, uncommented APL code was written in such a compact way; the answer probably was "oh yes, well, I had to write it like that to be efficient.") In fact it is not necessary to sacrifice understandability for efficiency, if the right approach is adopted.

One of the first jobs the application developer should do when extracting the users' requirements is to gain a clear understanding of which of the quality characteristics are perceived as important. This simple task is often overlooked in the rush to 'start work'.

How do programs get to be efficient? Either by writing them that way because efficiency was declared to be a critical quality factor, or more usually, by making an existing program more efficient, which we can call optimisation. Ironically of course, it is often harder to do things the second way, especially if storage efficiency is a factor.

Even if you cannot afford the time to directly consider the efficiency issue at the start of a project, there is something you can do at this stage to make life easier later on. Making a program more efficient is merely a kind of program modification, albeit an unusual one which does not change the input or output. Any type of program modification can be made more easily, quickly, cheaply and correctly when the program is highly modifiable, and high modifiability is one of the attributes of a good system design. A system design technique well-suited to APL is described in [1].

So a useful rule is: if you are expecting to make changes to your program, do a good design. It is so much more effective to put early effort into creating a modifiable system than to waste the time later heart-searching about what type of looping construct to use, or whether to truncate variable names to three characters, or other such nonsense.

The fact of the matter is, however, that most of the code people want to optimise has already been written, and so we do need to consider alternative coding techniques, about which more later.

Perhaps it is worth briefly reflecting on why efficiency in APL has become more of an issue recently. APL is now expected to compete with packages and languages that it could not have hoped to in the past. Users are not willing to put up with a language that is insular, but expect it to interact with its environment and to take full advantage of external facilities. Consider as an example STSC's Statgraphics package, which competes very successfully with other PC and mainframe statistical and graphics systems. Although written largely in APL, the performance that users expect can only be delivered by a combination of efficiently-coded APL, C and Assembler. On the mainframe side too, users want to exploit the particular advantages of their environment, access fast Fortran routines and carefully monitor their APL applications, to name just three examples at random, with a view to reducing the CPU loadings on their machines.

Tools for Optimisation

Once you have decided to optimise a system, the first thing you have to do is decide what to optimise. It can be very tempting to think that speeding up everything is the best way to proceed, but do not be fooled. It is similarly a waste of time to say 'OK, I'll rewrite function XYZ because it uses a really inefficient algorithm' when function XYZ only accounts for 1% of the total CPU time in a run.

A better approach is to optimise just those areas that need it most. There are two main advantages to this method: first, time is not wasted working on areas which will not yield benefits; and second and perhaps more critical, the overall maintainability of the system is not compromised. The goal is to achieve a large increase in efficiency while retaining the flexibility of an easily readable, modifiable, testable and portable system.

These observations underline the need to carefully and accurately monitor application execution time. You cannot start optimising until you know what to optimise, and you cannot know what to optimise until you know what is slow. So let us review the tools available to the APL optimiser. Figure 2 is a list of execution monitoring tools available to users of different APLs - apologies for any omissions.

APL interpreter	Tools available
VS APL	- APAT (APL Performance Analysis Tool)
APL2 R1,R2	- Migrate APAT
APL2 R2,R3	- ⎕IB
APL2 R3	- TIME external function
APL*PLUS Mainframe	- ⎕MF
APL*PLUS PC R6	- Fastcode monitor
APL*PLUS PC R7	- ⎕MF
SHARP APL (experimental)	- ⎕FM

Figure 2: APL execution monitoring tools

IBM's APAT product is rarely seen; it will give the CPU time for each function line, and list the functions consuming most time in descending order. APAT is itself written in APL, and works by modifying the APL application on which it is used. A review can be found in [14]. Users of APL2 Release 2 or 3 could try to migrate APAT, or use the quad-IB system function, as described in [2]. Correctly used, quad-IB will yield the number of times each line of code is executed, and the CPU time per line. The quad-IB facility actually provides a whole range of useful facilities, of which function execution tracing is but one. (APL veterans may be amused to learn that 'IB' apparently stands for 'I-Beam'.) Release 3 comes with an external function called TIME for measuring the performance of APL2 programs.

Users of APL*PLUS Mainframe or APL*PLUS PC Release 7 can use the built-in monitoring facility accessed via quad-MF. This system function will monitor the execution of all the functions named in its right argument. A typical use is given in Figure 3.

```
1 ⎕MF ⎕NL 3 ⍝ Monitor ALL functions in ws
RUN         ⍝ Run the application to completion
)COPY 3 MFFNS GRPCUMSUM GRPLTIMES
CUMSUM  ⎕NL 3     ⍝ Display CPU info for each function
LTIMES 'FOO'      ⍝ Display CPU info for each line of FOO
```

Figure 3: Use of APL*PLUS Mainframe Monitor Facility

This article is not the place for a full description of the outputs of the CUMSUM and LTIMES functions, but a synopsis of the information produced is useful. The CUMSUM function produces a listing which includes the following pieces of information for each function in its argument:

- The function name
- The total CPU time spent in the function
- The function's CPU time as a % of the whole run
- The cumulative % CPU time
- The number of times the function was called
- The maximum number of times any one line was executed
- The number of lines in the function
- The function size in bytes

The above information is displayed on a single row per function, ordered so that the highest CPU consumer appears at the top of the list. The LTIMES function applies to a single function name, and yields information relating to each line of code in the function. The display includes the function listing with the following data appended to each line of the listing:

- The CPU time spent on the line, excluding sub-functions (HERE)
- The CPU time spent on the line, including all sub-functions (ALL)
- The number of times the line was executed

The HERE time ignores the time spent in any defined sub-functions called by the line, and so identifies code lines which are intrinsically expensive. The ALL time accumulates the time spent in all of the sub-functions invoked as a result of executing the line, and is crucial for identifying CPU-critical 'function trees' in the workspace. This encourages the optimiser to recast the algorithm of a function with high ALL time so that it no longer needs the expensive sub-functions.

Optimisation Procedure

The process of optimisation involves the following steps:

- Time the application with specific, known inputs
- Make optimising changes
- Re-time the application with the same specific, known inputs

Clearly one can iterate on the last two steps until the objective has been achieved, or until resource limitations preclude further study.

In large, complex APL applications it is especially important to identify the objects of optimisation. It is initially tempting to view the object of optimisation as the entire ws; after all, we want to speed up the slow functions in the workspace don't we? In fact there may be more than one top-level function in the ws, and in any case there are likely to be many possible 'execution paths' in a large system.

So the focus of attention should be a specific 'theme' in the workspace, combined with a given set of data. I am using the word theme here to refer to exactly one execution path through the application. This path is determined by all of the inputs to the APL interpreter and the application during the run, which must remain unchanged for the timing runs performed before and after optimising changes have been made. The job of 'optimising the ws' may therefore be a larger task involving the optimisation of multiple themes.

Once you know the distribution of CPU time in the application theme under study you can start optimising. Most unoptimised programs obey the apparently ubiquitous 80-20 rule, which says:

- 80% of the CPU time is spent in 20% of the code

In some cases you find an even stronger skew, with over 90% of the time going into a single function or even a single line. Optimised systems usually show a more even spread of CPU time among the CPU intensive functions. In general, the greater the skew, the easier the work of optimisation. The golden rule is:

- Concentrate your effort where it will do most good

Sometimes it can be hard to know when to stop optimising a system. This depends on the original distribution of CPU over the critical functions, but once a reasonable job has been done on the hogs, it is probably time to stop. The point is that it is not usually worth trying to squeeze out an extra few milliseconds once you have halved CPU time. It is interesting to reflect on the fact that:

> A speedup factor of 2 saves more CPU time than a subsequent speedup factor of 1000

Optimising changes can be classified according to modifications made at three distinct levels:

- Line level
- Function level
- Function-group level

An example of a line level change is the replacement of one code fragment by another. In a function level change, the whole function is rewritten, or completely replaced by a new one. The last type is more subtle and can often give the most gains. What happens is that a certain function appears high on the CPU-hog list, and one is tempted to immediately concentrate a lot of effort on the function. This may not always be the most successful approach.

In these cases it is useful to look at the calling function - the function that invoked the CPU-hog. Of course, the calling function may not itself be a hog, but it may be possible to rewrite it so that the reference to the offending function is no longer required. Effectively you are rewriting the algorithm, and in doing so replacing one group of functions by a new group. In such cases it is essential to have a clear understanding of what the code is doing, i.e. to understand the application.

Techniques for Optimisation

Now that we know what to optimise, we should review the techniques available for effecting the optimisation. This depends on which APL you are using, but the options can be classified by the following categories, in no particular order:

- Recode using APL
- Recode to use fast-functions
- Recode to use an alternative programming language

- Recode to use operating system services
- Modify application to use IBM files more efficiently
- Modify application to use an APL component filing system
- Modify application to use more appropriate data structures
- Use APL compiler
- Upgrade your APL

The first on the list - recode using APL - is one of the most commonly used, and detailed examples will be saved until after the other techniques have been briefly addressed.

The phrase 'fast-function' is used here to refer to a function which has the external attributes of an APL function, but is actually implemented using a lower-level language. Generally fast-functions implement commonly used and well-defined APL utilities, e.g. table lookup functions. Many vendors now supply workspaces of fast-functions with their interpreters, while others offer them as add-ons.

Here is a summary of some of the software available for some popular APLs:

APL interpreter	Available fast-functions
VS APL	- third-party functions
APL2 R2	- supplied external functions - third-party external functions - third-party ⎕FMT external function
APL*PLUS Mainframe	- supplied FASTFNS worskpace - extra FASTFNS with APL*PLUS compiler
APL*PLUS PC	- supplied ASMFNS workspace - third-party assembler functions
Dyalog APL	- external functions

Figure 4: Available APL fast-functions

The following example shows how the fast-function technique is used in practice. In a recent job, a VS APL system was migrated to APL*PLUS Mainframe with a view to optimising it. (Many of the examples in this article come from the APL*PLUS Mainframe system, because much of my recent work experience has been with that system.) After monitoring the CPU intensive application, which was designed to analyse and print multi-level product-tree information, it was discovered that more than 95% of the total CPU fell on a single line. The offending line was merely doing a table-lookup between two large integer matrices of product code numbers, where all the codes in one table were known to exist in the other. The notoriously CPU intensive 'and-dot-equals' method was employed. The line was recoded to use a supplied fast-function for performing numeric table lookup. This simple change yielded a 90% speedup over the whole application theme.

Figure 5 summarises the original and replacement code lines.

```
      ⍴MAT
1000 10
      ⍴CODES
100 10

      I←1++⌿~v⍷MAT∘.=⍉CODES    ⍝ Original line

      I←MAT MATIOTAN CODES     ⍝ Replacement line
```

Figure 5: Using a fast-function to speed up numeric table lookup

Users of APL*PLUS/PC are well-advised to study the contents of the distributed ASMFNS ws, which contains a host of useful fast-functions. The workspace also contains 'segmented-string' fast-functions which permit the processing of delimited character vectors more quickly, and in less space, than when using the traditional character matrix representation. Other good sources af partition-function techniques include [3] and [4].

When the fast-function you need does not exist, and you are feeling particularly brave, you might consider trying your hand at developing your own low-level programs. Figure 6 summarises available foreign language execution facilities for a variety of APL interpreters.

APL interpreter	Facility
VS APL	CALL/AP
APL2	⎕NA
APL*PLUS UNX	⎕CALL, ⎕XP1
APL*PLUS VMS	⎕NA
APL*PLUS PC	⎕CALL, APL2C
APL.68000	AP1
Dyalog APL	⎕SH
IBM PC APL	⎕PK, AP2

Figure 6: Foreign language execution facilities from APL

In one VS APL application developed by a colleague before the days of fast-functions, the critical part turned out to be a numeric table lookup similar to the case described above. It was known that the rows of both argument matrices were in sorted order, however. The technique used was to write a small 360 assembly language program which used a fast binary-search algorithm to do the lookup. The assembler program was invoked from VS APL via a shared-variable link to the Interprocess CALL/AP auxiliary processor. The original APL algorithm was never implemented because it would simply have taken too long to run [5].

APL2 users can now write specialised programs in Assembler, Fortran, Rexx, PL/1, (and now even APL2 itself), and interface to them via the quad-NA Name Association facility. A similar facility is available in the APL*PLUS VMS system. APL.68000 foreign language routines can be accessed via the AP1 auxiliary processor via shared variables, and Dyalog APL external functions can be executed via the quad-SH (shell) function.

APL*PLUS PC users can write their own assembler programs and invoke them via the quad-CALL system function. Those interested in developing this skill should review the excellent tutorials [6] and [7]. For jobs too large to attempt writing assembler code, or those involving floating-point arithmetic, the APL2C interface from Lauer Software may prove suitable. A colleague has produced excellent results interfacing in this way to C programs, in the area of processing tabular control-break data prior to producing reports in APL*PLUS PC [8].

The host operating system can sometimes offer a solution to efficiency problems. On mainframes, IBM's System Sort utility or Computer Associates' CASORT utility can be used to sort large files more efficiently and easily than an equivalent APL program. At the other end of the scale, DOS commands can be used to great benefit from the PC APL environment. For example, the DOS PRINT command is an asynchronous process, and so when invoked from within APL*PLUS PC, large files may be printed without delaying subsequent APL processing. This is particularly beneficial on 80286/80386 machines.

A good file design is essential for both ordinary IBM files accessed via shared variables, and for APL component filing systems. Techniques for improving IBM file access times usually involve changes that minimise file i/o. For example, this might be done by playing around with file record lengths or by introducing an in-workspace file buffer. (See references [9 10 11]). An APL component filing system can provide excellent opportunities for the efficient processing of APL data, especially when multi-user access is required.

Choosing appropriate structures for the major data objects in an APL system is also crucial, both for time and space efficiency. This will be briefly covered later.

If an APL compiler is available, recoding CPU-intensive functions for compilation is another avenue open to the optimiser.

But for now, we should return to a most popular optimisation technique, the one we will concentrate on most - recoding in APL.

Recoding in APL

Four techniques can be identified when using pure APL to recode for efficiency:

- Change the idiom
- Remove redundancy
- Use dynamic function-fixing
- Use a better algorithm

Changing the idiom is a common technique. The use of fast-functions falls into this category. In a recent project a line of code was sped up by 26% by replacing the original idiom by a different one. The original line, which catenated two large numeric matrices (see Figure 7), was buried inside a long loop, where the boolean compresssion vector B had been pre-computed outside the loop.

```
Z←(B/MAT1),MAT2 ⍝ Original idiom, B pre-computed outside loop
Z←MAT1[;I],MAT2 ⍝ New idiom, I pre-computed from B outside loop
```

Figure 7: Example change of idiom under APL*PLUS Mainframe

The new idiom uses a pre-computed index vector I to index the columns of MAT1, rather than compress them. This is a faster operation in VS APL and in APL*PLUS Mainframe.

In the same system, the original author was able to rewrite a sub-function in order to avoid transposing the argument and the result, as shown in Figure 8. The new function used a modified algorithm to unpack the un-transposed codes matrix. Removing the transposes saved 29% of the CPU time, and this saving was increased to 47% by using fast-functions. [5].

```
Z←⍉UNPACK⍙CODES ⍉CODES     ⍝ Original line
Z←UNPACK⍙CODES⍙UP CODES    ⍝ New line
```

Figure 8: Avoiding two transposes saved 29%

A good understanding of the application can often allow redundant code to be identified and removed. In one system for example, the line of code in Figure 9 was being used to remove the rows of matrix MAT which were entirely zero. Not only is the inner-product algorithm expensive, but it turned out that the whole line was redundant; the operation had already been performed at an earlier stage of the processing. Removal of the line produced a significant benefit.

```
Z←(MAT∨.≠0)⌿MAT
```

Figure 9: Remove expensive redundant code

In the same system it turned out that about 10% of the CPU time was going into the calculation of control totals on the reports. When questioned, the user admitted that he never looked at these totals, and offered the opinion that they were "something the programmers wanted". Removing the control total calculations entirely might be a bit drastic, but modifying the application to provide the user with an option to omit them was an acceptable solution.

Savings can also be made by ensuring that invariant operations are removed from loops - 'loop hoisting'. Again, this is only worth doing if the looping function proves to be a large consumer of CPU time. In critical cases a vector of case-statement labels formed by repeated catenations inside a loop can be replaced by a line that uses a pre-computed vector of labels, as in Figure 10.

```
LP:→(L1,L2,L3,L4,L5,L6)[CASE] ⍝ Old line
...

BRVEC←L1,L2,L3,L4,L5,L6        ⍝ New lines
LP:→BRVEC[CASE]
...
```

Figure 10: Loop hoisting in critical cases

Redundant operations can also be removed from within interpreter loops. For example, I recently came across a line of code similar to the following one, in which the dyadic-iota was consuming quite a lot of CPU time:

```
I←BITVEC/SHORTVEC⍳LONGVEC
```

A reduction in processing time was achieved by performing the compression before the expensive lookup, without changing the result:

```
I←SHORTVEC⍳BITVEC/LONGVEC
```

It is better to apply any selection operations as soon as you can in order to avoid unnecessary processing.

The technique of using quad-FX to dynamically fix a function is a type of 'compilation' which all APL users can exploit. I hesitate to mention it however, because it has one very serious drawback. The general idea is to build a character matrix which contains the lines of code that are to be executed, but using run-time knowledge to minimise the processing. The character matrix is then fixed to create the function. The approach may be indicated when the following conditions arise:

- Heavy processing inside a long loop
- Conditional code, much branching
- Execute is being used to execute constructed expressions

The benefit is faster code; there are no conditions, branches, executes or constructed expressions of any kind. The very serious drawback is that the code is usually much more difficult to write, debug, maintain, modify or optimise, because it is harder to understand.

In one VS APL optimisation effort this technique was taken to the limit; no less than 19 functions were dynamically fixed by the system, many of which still appeared at the top of the CPU list. In this particular case, all other avenues had been explored, and only this approach made the application feasible. (This was before the days of fast-functions and compilers.) This is a very rare case - in general the disadvantage far outweighs the benefit for applications programming, and the method should be avoided.

It is easier to justify in APL 'systems programming' or for building utilities of a special nature. [13].

Use a Better Algorithm

Before you can use a better algorithm you have to know what is wrong with the current one, and how to make improvements.

Because APL is an interpreted language, the efficiency of 'small array' problems can be considered distinctly from that of 'large array' problems. When applied to small arrays all the APL primitive operations take about the same amount of CPU time, because all the CPU is spent analysing syntax and managing memory. This means that the total CPU time is determined by the total number of distinct operations. On the other hand, for large arrays the interpretive overhead is relatively small compared to the time spent processing the elements, and the CPU time is determined by the type of operation being performed. To summarise:

- SMALL ARRAYS
- All primitives equivalent
- Most CPU in the syntax
- ==> CPU determined by the NUMBER of operations
- LARGE ARRAYS
- Interpretive overhead small
- Per-element time large
- ==> CPU determined by the TYPE of operations

This begs an interesting question; how large is a large array? On the APL*PLUS Mainframe system, the expression V+V takes about 700 machine instructions to analyse the syntax and perform setup operations, and only 7 machine instructions to actually add each pair of elements. This situation is summarised in Figure 11.

V+V

	Number of elements in V		
	1	100	10,000
Overhead	700	700	700
Addition	7	700	70,000

Figure 11: Number of machine instructions to perform V+V on APL*PLUS MF

Notice that for a singleton vector the overhead dwarfs the time for the addition, whereas for 100 elements the CPU spread is balanced between overhead and productive processing. As the array grows larger the percentage of the time spent doing useful work increases, reaching 99% for 10,000 elements. This is why it is generally better to process arrays of data in APL rather than loop round processing scalars.

So, to get back to the subject, when optimising small array problems, the approach is to remove as much syntax and redundant baggage as possible, thereby producing as concise a solution as possible. For large array problems the techniques are different; we must look at the type of the operations being used, and in particular, the 'order' of the algorithm.

The 'order' of an operation is a way of describing how the processing time is expected to increase as the number of items to be processed increases. All of APL's scalar functions are order-N, or linear, operations, that is to say the time to complete the operation is linearly proportional to the number of elements processed (N) when the arrays are relatively large. For example, we would expect the time for the expression V+V to double when the number of elements in V is doubled. Other primitives are associated with higher orders, the worst offenders including the inner and outer product, and the cases of index finder and membership where the interpreter performs a naive search.

The order of common APL primitives are listed in Figure 12.

ORDER	FUNCTION
N	Scalar functions, eg + - × ÷ Some cases of ⍳ and ∈ in advanced systems
NlogN	⍒⍋ Some cases of ⍳ and ∈ in advanced systems
$N^{1.5}$	Inner products, ⌸
N^{2}	Outer products Some cases of ⍳ and ∈ in advanced systems All cases of ⍳ and ∈ in VS APL

Figure 12: The order of some common APL operations

In VS APL the membership and index finder functions are notorious, achieving no better than N-squared time. In fact it is usually possible to write cover functions to these search operations which are actually faster than the primitives themselves. [15 9]. In more

advanced APL systems, such as APL*PLUS Mainframe, these functions employ sophisticated algorithms the behaviour of which depends upon multiple factors, so that log and even linear time can be achieved.

The overall order of a complex algorithm may be determined by looking for the highest order of operation in the algorithm. One approach to optimisation could therefore attempt to recast the N-squared operations as lower-order algorithms, with a view to reducing the overall order.

As an example, consider the following example, taken from an STSC Newsletter article by Jim Weigang [12]. Figure 13 shows three variants of the COUNT function, which simply counts the number of 1s in its argument, the number of 2s, etc. COUNT1 uses an obvious APL approach - but the outer product makes it an N-squared algorithm. COUNT2 embodies a more subtle algorithm, using grade-up to sort the argument - this makes it order NlogN. Finally, COUNT3 employs a very naive 'BASIC-style' looping solution. No operation in COUNT3 is worse than linear, so the function as a whole is expected to have a linear order.

```
      COUNT1 4 7 7 5 2 1 1 5 1 9
3 1 0 1 2 0 2 0 1


    ∇ Z←COUNT1 W
[1]   ⍝ Outer product makes this an N-squared algorithm
[2]     Z←+/(⍳⌈/W)∘.=W
    ∇

    ∇ Z←COUNT2 W;L;I;F
[1]   ⍝ Sort, find last occurrence of each number, compute frequencies
[2]   ⍝ ⍋ makes this oder NlogN
[3]     W←W[⍋W] ◇ L←W≠1⌽W ◇ L[(0≠⍴W)/⍴W]←1
[4]     I←0,L/⍳⍴L ◇ F←(1↓I)-¯1↓I
[5]     Z←(0⌈⌈/W)⍴0 ◇ Z[L/W]←F
    ∇

    ∇ Z←COUNT3 W;I
[1]   ⍝ Uses <W> as an index to increment appropriate result element
[2]   ⍝ Scalar operations make this order N, a linear algorithm
[3]     Z←(0⌈⌈/W)⍴0 ◇ I←⍴W ◇ →(I=0)/0
[4]   LP:Z[W[I]]←Z[W[I]]+1
[5]     →(0<I←I-1)/LP
    ∇
```

Figure 13: The three COUNT functions

The relative APL*PLUS Mainframe CPU timings as reported by Jim are listed in Figure 14. The first column is the number of elements in the argument, and the other columns show the function used. (The last column gives the results obtained for running the COUNT3 function compiled with the STSC APL Compiler.)

The results may surprise those belonging to the parallel-is-always-best school of APL thought. Let's ignore the compiled function for a moment. For small arguments COUNT1 is the fastest because it uses the fewest operations, however, for larger arguments the order of the operation determines the time taken, and COUNT2 is the fastest. Even though it is linear, COUNT3 cannot hope to beat COUNT2 because it processes scalars. (You could say that COUNT3, although linear, has a very high coefficient of linearity.)

You may be surprised to see that the looping COUNT3 is at least twice as fast as the parallel COUNT1 for anything up to about 1,000 elements. Overall though, the COUNT2 function is probably the best one to store away in your utility workspace.

The high coefficient of linearity can be removed by compiling COUNT3. Freed of the interpretive overhead, the compiled version of COUNT3 reveals the great potential speed of the linear algorithm.

N	COUNT1	COUNT2	COUNT3	COUNT3 compiled
1	7	19	14	6
10	10	24	71	8
100	179	50	622	19
1,000	15,740	357	6,327	136
10,000	WS FULL	4,113	62,752	1,490
Order:	N-squared	NlogN	N	N

Figure 14: Relative CPU times for the COUNT functions

As another example, consider the Index GENerator utility function IGEN, which produces a vector of index sets catenated together (see Figure 15). The timing results are shown in Figure 16. Also included are the times for IGEN4 compiled, and for the equivalent vendor supplied hand-crafted fast-function OIGEN. Some of the solutions are sensitive to the magnitude of each element, and some to the total number of elements. Timings were therefore performed on two different sets of data which produce the same length of result.

```
      IGEN 3 0 4 2
1 2 3 1 2 3 4 1 2
```

```
     ∇ Z←IGEN1 W;L;B
[1]   ⍝ Nested array solution
[2]    Z← ⊃,/⍳¨W
     ∇
```

```
      ∇ Z←IGEN2 W;L;B
[1]   ⍝ ISO APL solution using outer product
[2]     L←⍳⌈/W
[3]     B←(W-~⎕IO)∘.≥L
[4]     Z←(,B)/,(⍴B)⍴L
      ∇

    ∇ Z←IGEN3 W;W1;V
[1]   ⍝ ISO APL solution using indexing
[2]     W1←(W≠0)/W
[3]     Z←(+/W1)⍴1
[4]     V←1-¯1↓(~⎕IO),W1
[5]     Z[¯1↓+\⎕IO,W1]←V
[6]     Z←+\Z
    ∇
```

```
    ∇ Z←IGEN4 W;I;J;N;C
[1]   ⍝ ISO APL solution using looping
[2]     Z←(+/W)⍴W
[3]     I←⎕IO
[4]     J←⎕IO+''⍴⍴W
[5]     N←0
[6]   LP:→(I=J)/0
[7]     C←W[I]
[8]     Z[N+⍳C]←⍳C
[9]     N←N+C
[10]    I←I+1
[11]    →LP
      ∇
```

Figure 15: The four IGEN functions

To put these timings into some perspective, consider that the nested array approach, when applied to the 1,000-element vector, takes nearly 1 CPU Second. It is of course much easier to write and to read (once you have mastered 'iota-each' and 'catenate-reduction'). As usual, if you are willing to put in a little more effort you can do much better. The preferred solution in the absence of the fast-function or the compiled function, is IGEN3.

In more complex algorithms the overall order is not always as obvious as these examples would imply. 'Catenate loops' and 'take/drop loops' are good examples of subtle N-squared algorithms, for instance. Consider the code fragment in Figure 17, which shows a simple loop in which records are being appended to the end of a matrix. Maybe the function PROCESS is reading a vector from an IBM file via a shared-variable.

		?100⍴100	?1000⍴10
⍳¨	IGEN1	22	360
∘.≥	IGEN2	9	10
[]	IGEN3	5	6
loop	IGEN4	14	117
IGEN4 compiled	IGEN5	2	8
fast-function	OIGEN	1	1.5

Figure 16: Relative CPU times for the IGEN functions
Timed on APL*PLUS Mainframe R5
Each unit = 2.7 Msec.

```
      ...
      MAT←(0,W)ρ0
 LP:  ...
      REC←PROCESS
      MAT←MAT,[1] REC
      ...
      →LP
END:
```

Figure 17: A typical catenate loop

Every time you go through the loop, a copy of MAT is made, a copy of REC is made, the new result is formed, which is then assigned to MAT again. This is unfortunate for two reasons: firstly a lot of data is moving around. If MAT starts off empty, and ends up with N rows, we can say that the average number of rows of data being moved by the interpreter will be half N. But we will go round the loop N times, so the total amount of data being shunted around will be half-N-squared. Hence we have an N-squared algorithm.

Secondly, this data copying places demands on the interpreter to continually allocate new blocks of storage, which is expensive and may well cause a fragmented workspace. Don't worry, this is not as painful as it sounds, but it could result in a costly garbage-collection operation when a new block of storage is required each time through the loop.

The problem can be reduced to linear time by replacing the catenate loop by an indexed-assignment loop instead. The simplest case, when you know how many rows there will be in advance, is shown in Figure 18. In other cases it is still possible to use this technique, but you will have to extend the matrix once the available rows have been filled.

```
      ...
      MAT←(N,W)ρ0
 LP:  ...
      REC←PROCESS
      MAT[I;]←REC
      ...
      I←I+1
      →LP
END:
```

Figure 18: An alternative to the catenate loop using indexed assignment

To give you an idea of the saving possible, I recently performed such a replacement in a system which was reading integer vectors via IBM's AP 110, in a situation where a block-

read was not a possible solution. The resulting function ran 21 times faster. (Subsequent compilation gave an overall 50-fold speedup.)

A similar problem occurs in a 'take/drop' loop, a template of which is shown in Figure 19.

```
      ...
 LP: →(0∈⍴V)/END
     PROCESS V[1]
     V←1↓V
     →LP
END:
```

Figure 19: A typical drop loop

Beneficial results are again obtained by replacing the code by an indexed assignment method. I would like to thank Clark Wiedmann for these examples, taken from his excellent APL*PLUS Compiler course.

The aim is to reduce the overall order of the algorithm as much as possible, hopefully achieving log or even linear time. In many cases you can replace a search type algorithm by a linear indexing operation. For example, suppose that you want to remove the duplicates from a set of integers in a known range. You may use the common utility called DUPSOUT or NUB, shown in Figure 20. The dyadic-iota in NUB can take a lot of time, especially if the vector is a very long one. If you do not mind getting the unique elements in ascending order, then you can use the alternative function SNUBRANGE, which uses a linear indexing operation. You have to know what the largest element in the set will be, and pass it in as the left argument.

```
    ∇ Z←NUB W
[1]  ⍝ <Z> is the NUB of the elements of the vector <W>
[2]
[3]    Z←((W⍳W)=⍳⍴W)/W
    ∇

    ∇ Z←A SNUB∆RANGE W;B
[1]  ⍝ <Z> is Sorted NUB of integers in set <W>
[2]  ⍝ Elements of <W> must be in the RANGE ⎕IO to <A>-~⎕IO
[3]
[4]    B←A⍴0
[5]    B[W]←1
[6]    Z←B/⍳⍴B
    ∇

      ⍝ These timings made on APL*PLUS PC R7 on an Olivetti M21

      I←?5000⍴10000
      Q1←NUB I                    ⍝ 26.4 Seconds
      Q2←10000 SNUB∆RANGE I       ⍝  1.6 Seconds
      Q1[⍋Q1]≡Q2                  ⍝ SNUB∆RANGE result is sorted
1
```

Figure 20: Alternative NUB function for integers in a known range

Another example is the use of dyadic grade to replace inner-product table lookup algorithms by equivalent order NlogN code. Figure 21 shows a typical function for removing the duplicate rows from a character matrix. Dyadic grade-up on the character matrix is used to bring matching rows together prior to deleting the duplicates. If your APL system has this facility, you can use it to develop a complete set of fast character matrix manipulation utilities.

```
    ∇ Z←MATNUB∆IP W;I;U;B
[1]   ⍝ Remove duplicate rows of matrix <W>
[2]
[3]   Z←(∨⌿<\W∧.=⍉W)⌿W
    ∇

    ∇ Z←MATNUB W;I;U;B
[1]   ⍝ Remove duplicate rows of character matrix <W>
[2]   ⍝ Uses dyadic grade for speed
[3]
[4]   I[⍋I]←+\∨/U←⍉1⌽U←W[I←⍋AV⍳W;]
[5]   B←(⍳⍴I)=I⍳I
[6]   Z←B⌿W
    ∇

      ⍝ These timings made on APL*PLUS PC R7 on an Olivetti M21

      A←⎕NL 3
      ⍴A←A[?100⍴1↑⍴A;]
100 13
      ⍴Q1←MATNUB∆IP A            ⍝ 54.9 Seconds
18 13
      ⍴Q2←MATNUB A               ⍝  0.9 Seconds
18 13
      Q1≡Q2
1
```

Figure 21: An alternative to inner product in MATNUB

Know your Interpreter

The APL philosophy largely protects the user from the idea of data-types, although under the covers APL interpreters are not all the same. Most systems use different internal representations for each type of data.

The more you know about your system the better, especially in terms of the data representations, and how they behave under common operations. How much storage does your APL take to represent an integer? A decimal number? A boolean value? Also, an understanding of the primitives and how they may be used to produce alternative solutions to problems is valuable.

As an example, look at the common problem of selecting records from a large matrix, consisting of many 'records' and few 'fields'. This operation is typically performed prior to special processing on the data, and may even occur within a loop, where the records to be selected have been pre-computed before entering the loop.

The choices before you are whether to use indexing or compression, and what orientation of your matrix to choose, i.e. whether to select rows or columns. To do some timing tests I set up a random 1000 by 20 matrix MDEEP and two boolean vectors BR and BC which can compress rows and columns of MDEEP respectively. From these were computed MWIDE, the equivalent 'wide' form of the same matrix, and IR and IC, vectors of indices corresponding to BR and BC, which can be used for indexing. The results for timings under APL*PLUS Mainframe R5 are summarised in Figure 22.

MDEEP←	Integer ?1000 20⍴10	Floating .9×?1000 20⍴10	Boolean 1000 20⍴1 0
MWIDE←⍉MDEEP	18.8	32.2	12.7
IR←BR/⍳⍴BR	1.3	1.4	1.4
IC←BC/⍳⍴BC	0.1	0.1	0.2
BR⌿MDEEP	5.6	10.4	5.3
MDEEP[IR;]	16.6	20.9	15.2
BR/MWIDE	22.4	16.4	33.0
MWIDE[;IR]	16.3	21.1	19.1
BC/MDEEP	22.0	25.1	29.9
MDEEP[;IC]	13.2	16.7	14.8
BC⌿MWIDE	2.2	4.3	1.2
MWIDE[IC;]	10.6	14.0	9.5
NB:			
BC/MDEEP	21.5		
⍉BC⌿MWIDE	15.3 !!		

CPU time in Msec. using APL*PLUS Mainframe R5.

Figure 22: Timings for alternative matrix selection expressions using APL*PLUS Mainframe R5.

Similar timings were observed with VS APL, the predecessor to APL*PLUS Mainframe. However, the following list of observations relates to the specific set of timings above, and should not be assumed for other APL systems, or for different environments. APL2 users beware: you may be surprised to find these results reversed. Interesting points to notice include:

- Transpose is expensive
- Compressing rows is faster than indexing them
- Indexing columns is faster than compressing them
- The same pattern is observed when selecting 'wide' records
- Selecting rows of the wide matrix is very fast
- No extra surprises with the floating data
- Compressing columns from a boolean matrix is very slow

We can surmise why some of these results pertain. The fast selection of rows, especially long ones, could be caused by the use of a 'block move' instruction for transferring contiguous bytes of memory from subject to object matrix, possible because arrays are stored in their 'ravel' order.

The comparative slowness of the boolean column selection operations can be explained by realising that the interpreter is engaging in much tedious unpacking and repacking of the 8-to-a-byte bits.

A similar experiment on APL*PLUS/PC R6, with 2-byte integers and 8-byte reals, gave the results in Figure 23. Indexing columns was again faster than compressing them (about 14%), but this time indexing rows was also faster (about 8%) than compressing them.

	Integer	Floating
MWIDE←	?10 1000⍴99	.5×?10 1000⍴99
I←B/⍳⍴B	0.1	0.1
B/MWIDE	1.0	2.1
MWIDE[;I]	0.9	1.8
MDEEP←⍉MWIDE	0.8	0.9
B⌿MDEEP	1.0	1.9
MDEEP[I;]	1.0	2.1

CPU time in Seconds, using APL*PLUS PC R6 on an Olivetti M21 with 8087 maths coprocessor.

Figure 23: Timings for alternative matrix selection expressions using APL*PLUS PC R6.

Have you ever thought of setting ⎕CT to zero? (All those with virulent ⎕CT phobia should quickly skip the next few paragraphs.)

When performing certain comparison operations APL shows tolerance when deciding whether two quantities are equal. This comparison tolerance usually applies for floating point numbers and large whole numbers, and is controlled by the setting of the system variable ⎕CT. Most APL systems put a limit on the largest value of ⎕CT in order to ensure that no two integers can be tolerantly equal. This allows the interpreter to skip the tolerance calculation for integers, producing the answer more quickly. For floating point numbers however, a tolerant calculation is performed.

But this applies even to whole numbers which are not integers, for example in APL*PLUS/PC, a number which exceeds the 2-byte limit of 32,767. Consider the lookup expression V iota V, performed on a vector V of one thousand 8-byte whole numbers, perhaps a set of packed dates or times. Figure 24 summarises the findings for three APL systems.

Tolerant Comparison

V←?1000⍴1E10

	V⍳V Default ⎕CT	V⍳V ⎕CT←0	Speedup
APL*PLUS PC R7	87secs	78secs	10%
APL*PLUS MF R5	25msecs	5msecs	80%
VS APL	792msecs	796msec	

Fig24: The effect of ⎕CT←0 when comparing large whole numbers
APL*PLUS PC timing on Olivetti M21 with 8087 co-processor

With the default setting of ⎕CT, APL performs the tolerance calculation. When ⎕CT is set to zero, some systems will skip the calculation (or perhaps some machines perform a hardware multiply-by-zero very quickly). The poor performance of VS APL is due to the naive dyadic-iota algorithm. Savings can often be made, with particular impact in the PC environment.

By the way, do not forget to reset ⎕CT to the default value afterwards, or you might encourage your application to behave differently thereafter. The proper way to restrict the effect of resetting ⎕CT is to localise it in an appropriate sub-function.

I apologise for the absence of results from other systems. I am not attempting to sell the merits of any particular APL in this article - I just make timings on those systems available to me. Perhaps someone will send VECTOR similar results for other systems, particularly APL2.

Perhaps a more common example is the comparison of alternative methods of justifying a character array. The two code fragments of interest are:

```
Z←(-+/∧\⌽' '=W)⌽W  ⍝ Right-justify: rotate booleans
Z←(-+/∧\' '=⌽W)⌽W  ⍝ Right-justify: rotate characters
```

It turns out that VS APL and APL*PLUS MF are faster rotating booleans than characters, whereas APL*PLUS PC is faster rotating characters than booleans (actually 2-byte integers).

It is worth repeating that system-dependent optimisations of this kind should be made with care, since the benefits may not transport between different APL interpreters, or even between different releases of the same interpreter. [9].

Data and File Design

This was not supposed to be an article about data design or file efficiency specifically, but the presentation on which it was based included a foil containing these efficiency tips relating to the design of data structures in APL:

- Do a data design for the major data structures
- Keep logically distinct data in separate structures
- Do not mix different data types together in one structure
    - it wastes space
    - is hard to modify
    - it causes the interpreter to work hard packing/unpacking
    - a nested array may be effectively used to store logically related data of different type in a single structure

- Consider the orientation of matrices
- Consider data-packing techniques
    - using base value/representation (decode/encode)
    - by modifying data types
- Imagine you are the interpreter, doing the work

And these relating to efficiency with files:

- Do a file design
- Structure files - have a directory
- Estimate CPU and elapsed time requirements
- Minimise file reads by
    - increasing file block sizes
    - 'caching' data in the workspace
- Use host services where possible
- Accumulate file update requests and apply them at once
- Store frequently accessed data in the file directory

Some aspects of improving file efficiency in the IBM environment are covered in references [10 11].

Efficiency with Nested Arrays

No talk on efficiency would be complete without reference to this topical and contentious subject. The foil from the presentation contained the following hard-learned lessons:

- Do a careful data structure design
- Keep logically distinct data in separate arrays
- Use the simplest structure possible
- Avoid using 'mixed' arrays for the major data structures
- Use the 'each' operator wisely

It is tempting to believe that packing every single piece of data into one great big nested array is somehow 'efficient'. If you do this you will undoubtedly regret it; it is hard to understand and modify, difficult to write appropriate data access code, and can lead to serious performance problems, in particular with regard to file i/o. It usually results from an inappropriate (or absent) data design.

Similarly, do not get suckered into using a 'deep' array when a 'shallow' one will do; why use a vector of vectors when a matrix is just as good? (If the 'inner' vectors are all the same length then it is certainly inappropriate.)

I learned this the hard way - my first excursion with APL2 resulted in a depth five data structure, each level of which corresponded to one level of corporate consolidation. As the system developed it became harder and harder to traverse the structure and to perform arbitrary operations at the level I wanted. Modification to the structure (e.g. adding a new intermediate consolidation level) became unthinkable. Luckily for me the system was built for demonstration and teaching purposes only (a purpose for which it is now very well suited).

Later on it turned out that the whole lot could be replaced by a depth two matrix. This had a number of advantages: it was simpler, easier to understand, easier to 'traverse', much easier to modify (adding a new consolidation level now meant adding a new row to the matrix). In addition it became apparent that an identical structure could be used for another part of the system, and that a common set of access functions could be written.

Last year, a colleague achieved a threefold speedup of a customer's application by replacing a depth four structure by a depth two structure, and of course modifying the access functions.

For most commercial applications a depth two nested array is deep enough.

This is no reason to shy away from 'second generation' APL code however. Nested array facilities can be very effective, rewarding us with excellent productivity gains, in the area of systems programming (testing, debugging, one-off utilities) as well as in applications.

If you do have nested arrays, then use them freely. Part of the system testing phase should include a monitoring exercise; you may well find that the CPU bottleneck is in some nested array code. At that stage you can take the time to recode the offending fragment into traditional APL. In this way the best tradeoff between productivity and efficiency can be obtained. The trick is never to let the real hog code reach the user.

The APL2, APL*PLUS and Dyalog 'each' operator can be a rather good little CPU-gobbler if used carelessly, however. It really is just a hidden loop - something that most

APLers would normally recognise as undesirable. A small example will demonstrate the perils of 'each'.

Suppose you have a vector of character matrices, of varying sizes, and you want to left justify each one. You might think "Ah yes, first I have to find where the blanks are". Fine - so you write on a piece of paper:

```
V=' '
```

"Hmmm, because 'equals' is pervasive (you cleverly remember), I now have a vector of boolean matrices, so I need to do an and-scan on each one - I'd better use 'each'." So you write:

```
∧\¨V=' '
```

"Right - now to add up the number of ones in each row of each boolean matrix", you continue:

```
+/¨∧\¨V=' '
```

"All I need to do now is rotate each matrix by each of the integer vectors I've just produced":

```
LM←(+/¨∧\¨V=' ')⌽¨V
```

Then you proudly sit back and look at the result. You type it in. It works! (So it must be right.) Maybe you even embed it into a utility function called LJUSTEACH.

What you have just done of course is to commit at least two cardinal APL sins; you looped three times and you re-invented the wheel. The nasty specks all over the screen are what Alan Graham of IBM referred to as 'pepper' in a presentation at APL87. They may not make you sneeze, but are just as irritating.

What you should have done of course is to remember the LJUST utility you wrote ages ago. It looked like this:

```
    ∇ Z←LJUST W
[1]  ⍝ <Z> is character matrix or vector <W> left justified
[2]    Z←(+/∧\⌽' '=W)⌽W
    ∇
```

It works fine on ordinary old character matrices, so you can apply it to each character matrix in the vector of matrices:

```
LM←LJUST¨V
```

This approach has some advantages:

- It exploits an existing utility
- It is easier to write, read and understand
- It is easier to modify
- It executes faster because it uses only one 'each'

This emerging coding style with 'each' can be summarised as follows:

- Solve the problem for SIMPLE arrays first
- Put your simple array solution into a defined function
- Use the defined function with the each operator

Finally, I would like to write about the benefits of using an APL compiler. I would like to, but this article is already overlong (it could certainly do with optimising). If there is interest, this topic rightfully deserves an article of its own.

If I may finish by stealing another Alan Graham line, I should remind you that if your program has not yet been optimised, then it must be pessimised.

References and Bibliography

[1] Glenford J Myers, Reliable Software through Composite Design, Van Nostrand Reinhold Company, N.Y.

[2] David Piper, Technical Correspondence, VECTOR Vol 3. No 4. p109, April 1987.

[3] Jonathan Barman, APL and Partitioned Data, VECTOR Vol 1 No 1 p127, May 1984.

[4] Bob Smith, A Programming Technique for Non-Rectangular Data, APL79 Conference Proceedings p362-367, Rochester N.Y.

[5] Jonathan Barman, Cocking & Drury Ltd, private communication.

[6] P Wortham, APL*PLUS PC and Assembler (Part 1), VECTOR Vol 4 No 2 p107, October 1987.

[7] P Wortham, APL*PLUS PC and Assembler (Part 2), VECTOR Vol 4 No 3 p109, January 1988.

[8] Romilly Cocking, Cocking & Drury Ltd, private communication.

[9] Eugene McDonnell, Technical Correspondence, VECTOR Vol 3 No 2 p94, October 1986.

[10] John Sullivan, Faster File Input in TSO, VECTOR Vol 4 No 1 p99, July 1987.

[11] Lori D McNichols, Fast I/O - Efficient file Processing, APL88 Conference Proceedings.

[12] Jim Weigang, Understanding Execution Time, STSC Newsletter, 1987.

[13] David Piper, Using quad-FX to Facilitate the Use of APs, VECTOR Vol 3 No 2 p123, October 1986, and VECTOR Vol 3 No 3 p130, January 1987.

[14] Bernard King, APAT: First Impressions, Software Review, VECTOR Vol 1 No 2 p44, October 1984.

[15] Phil Last, Technical Correspondence, VECTOR Vol 2 No 3 p97, January 1986.

Other Useful References

David Piper, Using Name Association for Data Translation, VECTOR Vol 3 No 4 p109, April 1987.

David Piper, A Command Driven Interface for BDAM and QSAM APs using APL2 under TSO, VECTOR Vol 3 No 3 p125, January 1987.

Clark Wiedmann, Efficiency in the APL Environment - A Full Arsenal for Attacking CPU Hogs, APL85 Proceedings, Seattle, Washington.

Jim Lucas, Making APL Cheap - Saving Time and Money, APL88 Conference Proceedings.

British APL Association
Software Library

Order Form

To: Dave Ziemann
Flat 3, 63 Queens Crescent
London NW5 4ES, ENGLAND

PLEASE SUPPLY THE DISKS CIRCLED BELOW:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53
54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99	100							

TOTAL NUMBER OF DISKS ORDERED (please double-check): ------

at 2 Pounds per disk : ------ BAA Member's rate

OR at 3 Pounds per disk : ------ Non-member's rate

I want to join the BAA, and enclose
the 10 Pounds annual membership fee : ------

Postage and handling : 1.00

Add 2 Pounds for orders outside the UK: ------

Total order - remittance included: ======

Your signature: _____

Your name: _____

Full address: _____

_____

_____

Please make cheques payable to the British APL Association.

Payment can be made in US Dollars - pay 1.5 US Dollars for each Pound.

All orders must include payment - allow 30 days for delivery.

BAA membership year runs from 1 May - 30 April. People joining part way through the year will receive appropriate back numbers of Vector.

VECTOR - Index to Back Numbers
Vols 1 - 4 inclusive

Title	Author(s)	Vol	No	Page
"Decision Support Systems"	David Preedy	2	2	42
"Diagnosing the system"	David Preedy	2	3	33
Dyalog APL on the IBM 6150	Adrian Smith	4	2	53
"Expert Systems:Concepts & examples"	Robert Bittlestone	1	3	42
ForteGraph & XT/286	Adrian Smith	4	1	31
"Fuzzy Sets & Decision Analysis"	Robert Bittlestone	1	4	51
"Graphics ... in Office Systems"	David Preedy	2	3	36
H-P 7550 plotter	David Preedy	2	1	40
Hercules-plus Graphics Card	Adrian Smith	4	1	44
I-APL for the IBM PC	Dave Weatherby	4	3	37
"Introduction to APL" (Peelle)	Romilly Cocking	3	3	38
"Intro to APL for the PC" (Murray&Pappas)	Anthony Camacho	4	3	32
Keyword APL/QL	David Ziemann	1	4	121
LaserJet printer	David Preedy	2	1	42
"Learning & applying APL"	Anthony Camacho	1	3	39
"LISP: a gentle introduction ..."	Robert Bittlestone	1	4	51
MicroSpan	Anne Wilson	2	4	41
"Mr. Babbage's Secret"	Ole Franksen	1	2	42
NEC V20 Processor	Adrian Smith	4	2	51
PC Graphics Cards	Martyn Adams	3	1	51
PL/PC	Nick Small	4	4	52
PortaAPL	Normand Montour	3	1	45
"Powers of TEN"	Robert Bittlestone	1	4	53
U-REG - regression package	Teresa Jones	1	3	45
"Visual display ..."	David Preedy	2	3	34

Index to Advertisers

All queries regarding advertising in VECTOR should be made to the advertising editor, Cathy Dargue, at the following address:

Cathy Dargue,
60 Downhall Ley,
Buntingford,
Herts SG9 9TL

Tel: 0707-325161

BRITISH APL ASSOCIATION

Membership Application Form

Please read the membership information in the inside front cover of VECTOR before completing this form. Use photocopies of this form for multiple applications. The membership year runs from 1st May – 30th April.

Name: _____

Department: _____

Organisation: _____

Address Line 1: _____

Address Line 2: _____

Address Line 3: _____

Address Line 4: _____

Post or zip code: _____

Country: _____

Telephone Number: _____

Membership category applied for (tick one):	87/88	
Non-voting student membership (UK only)	£ 5	
UK private membership	£ 10	
Overseas private membership	£ 18	$ 27
Airmail supplement (not needed for Europe)	£ 8	$ 12
Corporate membership	£ 85	
Corporate membership Overseas	£140	$210
Sustaining membership	£360	

For student applicants:

Name of course: _____

Name and title of supervisor: _____

Signature of supervisor: _____

PAYMENT

Payment should be enclosed with membership applications in the form of a UK sterling cheque or postal order made payable to "The British APL Association". Corporate or sustaining member applicants should contact the Treasurer in advance if an invoice is required. Please enclose a stamped addressed envelope if you require a receipt.

Send the completed form to the Treasurer at this address:

Mel Chapman, 12 Garden Street, Stafford ST17 4BT, UK.

The British APL Association

The British APL Association is a Specialist Group of the British Computer Society. It is administered by a Committee of officers who are elected by the vote of Association members at the Annual General Meeting. Working groups are also established in areas such as activity planning and journal production. Offers of assistance and involvement with any Association matters are welcomed and should be addressed in the first instance to the Secretary.

1987/88 Committee

Chairman:	Dick Bowman 01-634-7639	CEGB, 85 Park Street, LONDON SE1.
Secretary:	Graham Parkhouse 0483-571281(x2379)	Dept. of Mech. Eng., University of Surrey, GUILDFORD GU2 5XH.
Treasurer:	Mel Chapman 0785-53511	12 Garden Street, STAFFORD ST17 4BT.
Activities:	Phil Goacher 03727-21282	27 Downs Way, EPSOM, Surrey.
Journal Editor:	Adrian Smith 0904-653071	Brook House, Gilling East, YORK.
Education:	Norman Thompson 0962-54433	IBM Mail Point 188, Hursley Park, Winchester Hants, SO21 2JN
Publicity:	Jill Moss 0225-62602	17, Barton Street Bath, Avon
Technical:	David Ziemann 01-436 9481	Cocking & Drury Ltd, 180 Tottenham Court Road, LONDON, W1P 9LE
Recruitment:	Val Lusmore 0225-62602	17, Barton Street Bath, Avon
Projects:	David Eastwood 01-735 3806	Sahara Software Ltd, Unit 5.11 Bondway Business Centre, 69-71 The Bondway, LONDON SE1 6LN

Journal Working Group

Jonathan Barman	0374-588835
Anthony Camacho	0727-60130
Cathy Dargue	0707-325161
Val Lusmore	0225-62602
Adrian Smith	0904-653071
David Ziemann	01-436 9481

Activities Working Group

Phil Goacher	03727-21282
Maurice Jordan	01-562 3090
David Parker	01-834 2333

Typeset using MS Word 3.0 on Xerox 3700
Printed in England by Short-Run Press Ltd, Exeter

VECTOR

VECTOR is the quarterly Journal of the British APL Association and is distributed to Association members in the UK and overseas. The British APL Association is a Specialist Group of the British Computer Society. APL stands for "A Programming Language" - an interactive computer language noted for its elegance, conciseness and fast development speed. It is supported on many timesharing bureaux and on most mainframe, mini and micro computers.

SUSTAINING MEMBERS

The Committee of the British APL Association wish to acknowledge the generous financial support of the following Association Sustaining Members. In many cases these organisations also provide manpower and administrative assistance to the Association at their own cost.

APL People
17 Barton Street
BATH, Avon
Tel:0225-62602

Cocking & Drury Ltd
180 Tottenham Court Rd
LONDON, W1P 9LE
Tel:01-436 9481

HMW Progr. Consultants Ltd
142 Feltham Hill Rd.
ASHFORD
Middlesex TW15 1HN
Tel:07842-41232

Inner Product Ltd
Eagle House
73 Clapham Common Southside
LONDON SW4 9DG
Tel:01-673-3354

MetaTechnics Systems Ltd
Unit 216, 62 Tritton Road
LONDON SE21 8DE
Tel:01-670-7959

I.P. Sharp Associates
10 Dean Farrar St.
LONDON SW1H 0DX
Tel:01-222-7033

APL Software Tech.
14 Rosewood Avenue
Alveston, BRISTOL BS12 2PP
Tel:0454-415737

Dyadic Systems Ltd
Park House, The High Street
ALTON, Hants
Tel:0420-87024

APL Impetus Ltd
Rusper, Sandy Lane
Ivy Hatch, SEVENOAKS
Kent TN15 0PD
Tel:0732-885126

Mercia Software Ltd
Aston Science Park
Love Lane
BIRMINGHAM B7 4BJ
Tel:021-359-5096

MicroAPL Ltd
South Bank Technopark,
90 London Road,
LONDON SE1 6LN
Tel:01-922 8866

Peter Cyriax Systems
213 Goldhurst Terrace
LONDON NW6 3ER
Tel:01-624-7013
Mobile:0860-377963

The British Computer Society, 13 Mansfield Street, London W1M 0BD.