ICL Technical Journal

Volume 2 Issue 2 November 1980

ICL Technical Journal

Contents

Volume 2 Issue 2

ICL Technical Journal

The ICL Technical Journal is published twice a year by Peter Peregrinus Limited on behalf of International Computers Limited

Editor
J. Howlett
ICL House, Putney, London SW15 1SW, England

Editorial Board

J. Howlett (Editor)	D.W. Kilby
D.P. Jenkins	K.H. Macdonald
(Royal Signals & Radar Establishment)	B.M. Murphy
M.V. Wilkes FRS	J.M. Pinkerton
(University of Cambridge)	E.C.P. Portman
C.H.Devonald	

All correspondence and papers to be considered for publication should be addressed to the Editor

Annual subscription rate: £10 (cheques should be made out to "International Computers Limited", and sent to International Computers Limited, Corporate Communication, ICL Technical Journal Office, ICL House, Putney, London SW15 1SW)

The views expressed in the papers are those of the authors and do not necessarily represent ICL policy

Publisher
Peter Peregrinus Limited
PO Box 8, Southgate House, Stevenage, Herts SG1 1HQ, England

This publication is copyright under the Berne Convention and the International Copyright Convention. All rights reserved. Apart from any copying under the UK Copyright Act 1956, part 1, section 7, whereby a single copy of an article may be supplied, under certain conditions, for the purposes of research or private study, by a library of a class prescribed by the UK Board of Trade Regulations (Statutory Instruments 1957, No. 868), no part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means without the prior permission of the copyright owners. Permission is however, not required to copy abstracts of papers or articles on condition that a full reference to the source is shown. Multiple copying of the contents of the publication without permission is always illegal.

© 1980 International Computers Ltd

Printed by A. McLay & Co. Ltd., London and Cardiff ISSN 0142–1557

The ICL information processing architecture IPA

J.Kemp and R.Reynolds
ICL Marketing Division, Slough, Berks

Abstract

IPA embodies ICL's strategy and plans for information processing over at least the next 10 to 20 years. It is not itself a product but a comprehensive and consistent set of systems concepts, design rules and function descriptions, together with specifications for interfaces and protocols, in accordance with which all existing software and hardware products will evolve and all new products will be developed. In broadest terms its objective is the provision of standard methods for linking together computers, computer systems and terminals via either public or private telecommunication lines or networks, extending to international linkings. The formal structure of IPA is based on the ISO seven-layered model for Open Systems Interconnection. The paper explains the need for such an architecture, describes the essence of the ISO model and shows its close relation to the existing ICL communications architecture Full XBM, from which IPA has evolved. The paper shows also how the principles of IPA can be implemented in various circumstances and describes some of the first products and user services which will be made available.

1 Why is an architecture needed?

The early development of the digital electronic computer was dominated by the concept of the centralised installation. This was largely a consequence of the high cost of the processor and main random-access store with the physical bulk, large power demands and need for a controlled air-conditioned environment contributing. As the possibilities increased for the transmission of digitally encoded information over the telephone lines available from the national telecommunications authorities – the PO in the UK, the PTT's in most other countries and certain regulated companies in the USA – there was a great increase in the use of the computer from terminals remote from the centre, and of increasingly wide variety. But the standard remained, essentially, the central machine though now with possibly very large numbers – such as hundreds – of terminals directly connected.

This type of structure, with simple terminals all connected to a single powerful mainframe machine, does not present any deep problems of system management and co-ordination of data, and the disciplines necessary for successful operation became well understood. There are many systems of this kind all over the world, some on a very large scale, which have been operating very successfully indeed for many years now. But over the past few years the cost of both processing power and

storage capacity have fallen dramatically as a consequence of developments in micro-electronics. The physical size and power consumption of the equipment have been reduced equally dramatically and now only the largest systems need to be housed in air-conditioned or otherwise special accommodation. At the same time, all the world's telecommunications authorities are accelerating the growth of digital communication services: for example, a public packet-switched network TRANSPAC is now operating in France and a corresponding service PSS will be opened by the PO in Britain in the near future. All this has made the dispersal of processing power and data storage from the centre a much more practical and economic possibility and similarly for the devolution of authority and the diversification of electronic applications. The user community, which now includes virtually the whole of the business world, has already seen great value in such developments and there is a real demand that, to put it most simply, it should be possible to connect any piece of equipment to any other and to transfer any kind or quantity of information, with everything kept properly under control and without the user needing to concern himself with anything but the job in hand.

This presents problems of an entirely different order of complexity from those of the simple centralised system with its directly connected terminals. Purely tactical or *ad hoc* solutions are no longer sufficient and indeed we are no longer dealing with simple data processing but rather with the interconnection of a wide variety of tasks aided by computers and associated devices: in fact, with all aspects of information processing and handling in general. There is clearly a need for a firm strategy for linking equipment together if one is to avoid the situation in which every new demand presents a new problem. But the strategy must be based on fundamental and widely accepted principles if the products to which it leads are to meet the needs of such a wide range of users. It must also allow for needs which have not yet been specified. The term 'architecture' has come to be used for the embodiment of such a strategy in the computer world and considerations of this kind have led ICL to the development of IPA.

IPA is not a product in the sense of a particular piece of hardware or software but a coherent set of rules and conventions which provide for the unified and comprehensive interconnection of the main ICL products such as data-processing systems, business systems and terminals, and give also the capability for connecting these to other products either from ICL or from other manufacturers.

2 IPA and the ISO 7-Layered Model for Open Systems Interconnection

The overwhelming majority of linkings between computers and associated devices such as terminals and satellite processors are made over lines provided by the national telecommunication authorities such as the PTT's. At one end of such a link User A despatches some information, which we can call a 'message', together with a destination, which we can call the 'address' of User B at the other end, and it is the responsibility of the PTT to deliver this safely, that is, without errors and to the correct address. How this is achieved, in particular how any errors in transmission are detected and corrected, is no concern of A or B; and what is in the message is no concern of the PTT. Thus there is a clear separation between the information processing function, which is the concern of the users, and the com-

munication function, which is the concern of the communication authority. But the manufacturer of the computing and other equipment must be aware of the requirements of this authority and must ensure that the signals constituting the messages which are put into the communication system conform in every way to the standards and conventions to which that system is built. Therefore whilst there is this separation between the processing and communication functions, the architecture which we have been discussing cannot avoid a close involvement with the architectures of international telecommunications systems.

There has been intensive study by international bodies for several years past of the question of what has become known as Open Systems Interconnection, or OSI. This means that if OSI were achieved in any community – which ideally should be the whole world – then there would be no technical barriers to the user of any computer, terminal or similar device communicating with any other within the community. The telephone and Telex are familiar examples of virtually world-wide OSI. The paper by Houldsworth[1] in the first issue of this Journal discusses this concept and the technical problems which underly its realisation. Recently the International Standards Organisation (ISO) has published[2] a formalisation of the fundamental structure of information processing and communication, which has become known as the ISO 7-Layered Reference Model for OSI. This, like IPA, is a set of concepts. It separates the full processing–plus–communication spectrum into an ordered set of seven subactivities or *layers*, in such a way that only adjacent layers interact with one another. The effect of this is both to make clear and to bound the consequences of a change in the implementation of any layer. In particular, the model makes the vital separation between processing and communication. The model and its relevance to the aims and realisation of OSI are discussed by Brenner.[3] We shall give a brief account of the model in Section 3 of this paper.

IPA and the ISO model are concerned with the same problem and it is evident that their aims have a lot in common. Further, the model is the outcome of truly international studies by groups of acknowledged experts in this field and is an expression of fundamental concepts which have already gained international acceptance. Therefore ICL, who have a general policy of adhering to international standards wherever possible, have incorporated the ISO model into the structure of IPA. It is not entirely accidental that the structure of the current ICL communications protocol Full XBM (also known as ICLC-03) is already quite close to that of the ISO model.

3 The ISO model and full XBM

Fig. 1 gives the structure of the model

The details are discussed at length in the definitive ISO publication[2] and more briefly by Brenner.[3]

The top layers 7,6,5 are concerned with the information processing functions and with ensuring that the messages which are to be transmitted between the communicating parties contain all the information necessary to specify the task to be performed and to control the interchange in whatever way is desired. The top

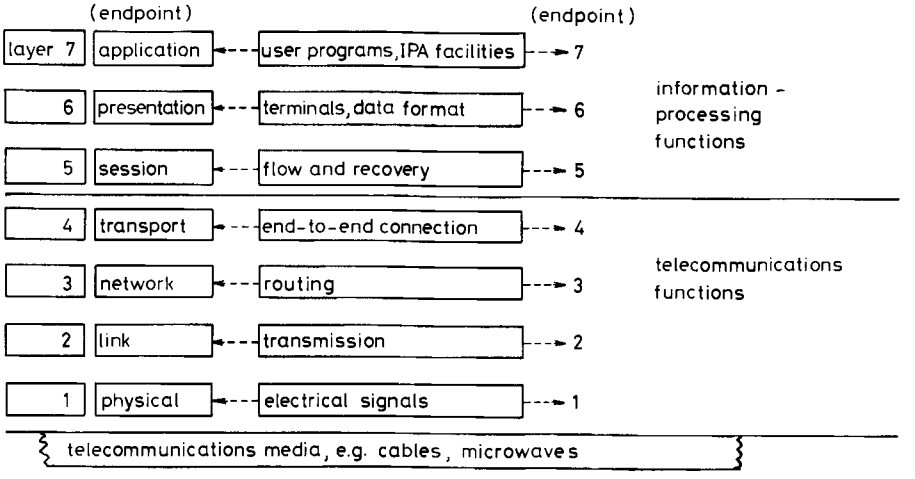


Fig. 1 ISO 7-layered model for OSI

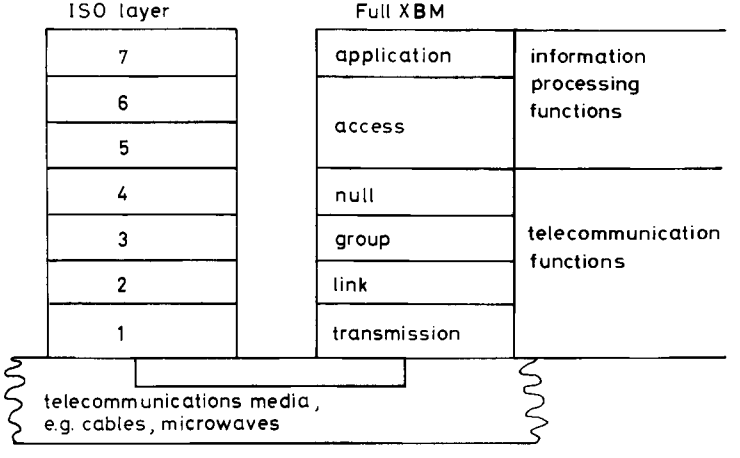


Fig. 2 Comparison between Full XBM and the ISO model

layer 7, the 'applications' layer, concerns the actual work which the users wish to do; they do not need to concern themselves with or even to be aware of the others below. The lower layers 4,3,2,1 are concerned with ensuring that the actual signals input to the communications system – for example, the public telecommunications network – meet the requirements of that system. The basic concept is that information originating in any one layer is handed down through the layers below until it reaches the physical communications medium; after transmission it is handed up through the layers at the receiving end until it reaches the equivalent layer. A principle of fundamental importance is that whatever is 'done' to an information stream as it moves down from one level to the next below at the sending end is 'undone' as it moves correspondingly upwards through the layers at the receiving end. This means that communication is between layers at the same level – 'peer'

layers – and that to any layer the layers below together appear as nothing more than a delay line. For this reason David Ackerman of ICL, in his early work in this field, used the terms 'onion skin architecture' and 'principle of complementary reflection'.

Over the past few years ICL has developed a series of structures for communications of which the latest and most comprehensive before IPA was Full XBM, originally called ICLC-03. XBM means Extended Basic Mode. This has much in common with the ISO model as Fig. 2 shows.

Like the ISO mode, Full XBM embodies the principles of:

separation of information processing from telecommunication
interfaces only between adjacent layers
end-to-end communication only between 'peer' layers.

It was thus a natural evolutionary step to adopt the structure of the ISO model as the basis of the new architecture IPA.

4 Initial implementations of IPA

4.1 Communications processors

The first major ICL hardware product designed according to the principles of IPA is the Multi-Function Communication Processor, MFCP. This expresses in physical terms the important separation between the processing and communications functions. Fig. 3 shows the structure.

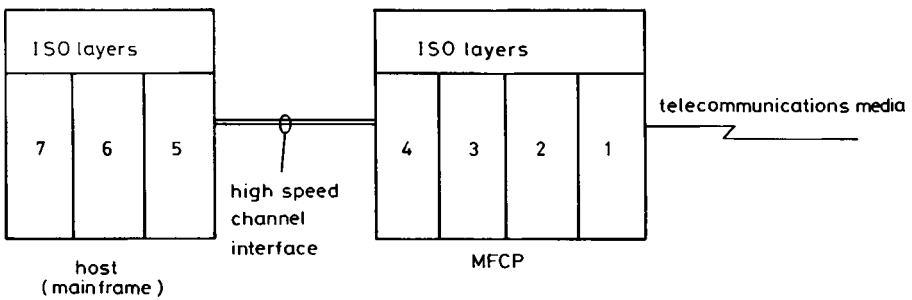


Fig. 3 The ISO Model and the ICL Multi-Function Communications Processor, MFCP

The MFCP provides the functions of layers 4,3,2,1 of the ISO model. Its purpose is to take care of all the operations which have to be performed in order to connect a mainframe (host) computer to a telecommunications system, such as for example a public packet-switched network. The top three layers are in the host machine. Layer 7 is of course the 'real' work which is to be done and is all the user need know about; layers 6 and 5 deal with data format, control of terminals and flow control and are implemented in the system software.

A great benefit resulting from this structure is that the host computer (meaning in effect the user) is insulated from changes in the communication system. Thus if the

latter is a packet-switched network using the X-25 protocol, as is the case for the French TRANSPAC, the British PSS and other important networks, this can be provided in the MFCP by appropriate software or micro-code in layers 3,2,1. But other protocols, such as X-21 for circuit-switched networks, are emerging and these could be implemented if the need developed.

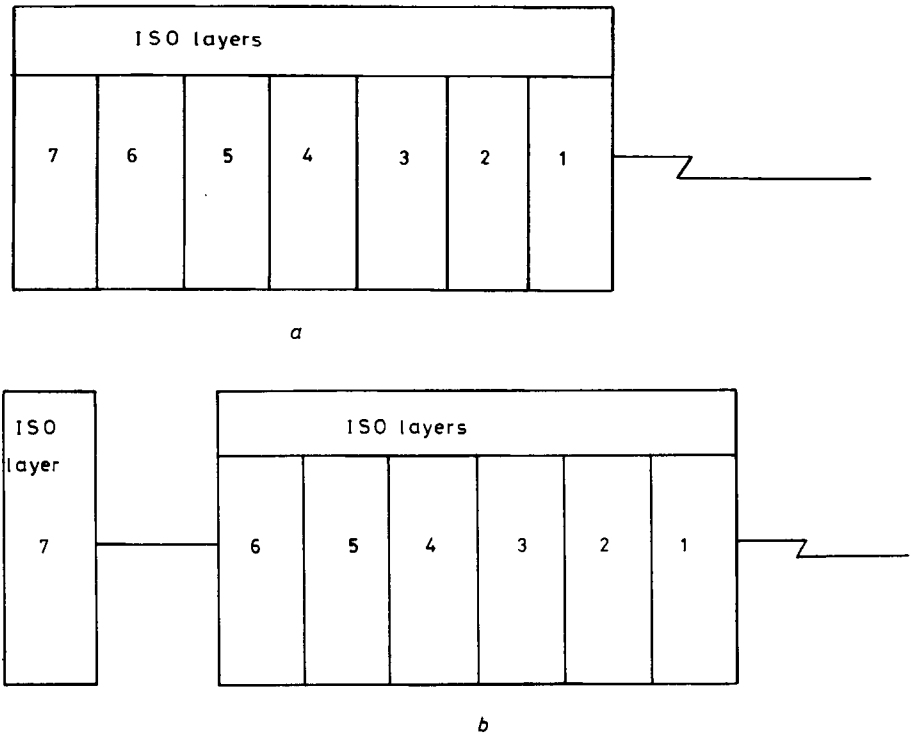


Fig. 4 *a* All ISO model layers in one unit *b* Application in 'back-end' processor

Communication between the host computer and the MFCP is via a high-speed channel and it is important that the host need not be concerned with the physical form of this – for example, whether it carries a single bit stream or is a 32-bit highway. Therefore a new protocol has been defined for the purpose embodying the rules governing how information is to be transferred between layer 5 (in the host) and layer 4 (in the MFCP).

This implementation, with the whole of the transport function of the ISO model in the MFCP and the processing functions in the host, is clearly one of several possibilities. Two extremes are equally possible (shown diagrammatically in Fig. 4*a* and 4*b*):

(*a*) a fully-integrated system with all the processing and transport capabilities (i.e. all seven layers) in one unit

(*b*) a single unit providing the capabilities of layers 1 to 6, acting as a front-end to a machine dedicated to processing; then only layer 7 would be implemented in the latter.

Any other allocation can be made but the order of the layers must not be varied – a basic principle of the ISO model is that the functions of the various layers have been defined and there must be interaction only between adjacent layers. The important point is that if the principles of the model are held to, then there is plenty of scope for flexibility in design of equipment with no risk of loss of coherence, compatibility and standardisation.

4.2 Packet switching terminals

Packet switching is being adopted by many of the world's telecommunication authorities and the relevant X-25 protocol has gained equally wide acceptance. (There is a thorough discussion of the concepts and merits of packet switching, and of the relevant protocols including X-25, in the book by Davies *et al.*[4]) ICL has therefore produced new models of the 7501 and 7502 terminals for connection to such networks. These control all levels of X-25 and thus relieve the mainframe of any need for knowledge of the existence of the network. The implementation does not however include the possibility of polling across the network because this would bring in an expensive overhead.

The implementation is known as PSTS, meaning Packet Switching Terminal Systems. The terminals are already in use in France with the TRANSPAC network and will be available for connection to the British PSS when this opens. Implementations for other X-25 networks will be made available as the need arises.

Fig. 5 shows the X-25 connections with the 7501 and 7502.

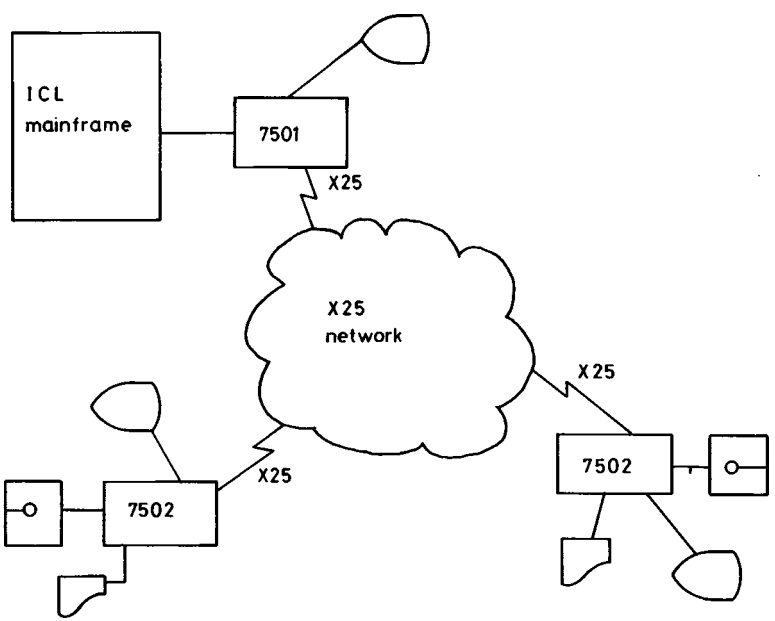


Fig. 5 Packet Switching Terminal Systems, PSTS

5 Some user services

To show what this means at the practical user level, we now describe some of the main services to be made available under IPA.

The following is an example of an entirely realistic situation. User A, with a small machine and a terminal, has some data which he wishes to process by means of a program P running on a large machine M at a distant site to which he has access. This program will need to use some other data held in a file F at third site. He wishes to have the results printed out on his local printer and also to transmit them to user B at a fourth site, for up-dating one of his (B's) files. All these files, programs and so on are known by names to the participants and these are the only identifiers which A wishes to use in initiating the various activities.

The services to be described enable this kind of activity to be carried out as a routine matter. They are all implemented as software running under the main ICL operating systems VME/B, VME/K and TME. DME users also are catered for under the IPA umbrella, as they may assume the role of an 'associate' in an IPA community, so that a DME machine can act as a host providing a service such as MAC to the community. It is the adherence to the principles of IPA which makes it possible to provide the services in such a general form; that is, in standard forms which, so far as the user is concerned, are independent of the equipment and the communications system being used.

The services are:
- Remote Session Access (RSA)
- Distributed Message Router (DMR)
- Distributed Application Facility (DAF)
- File Transfer Facility (FTF)
- Application Data Interchange (ADI)
- Range Remote Job Entry (RRJE)

5.1 Remote Session Access (RSA) (Fig. 6)

This facility allows the user of a terminal connected to an IPA mainframe to access a service resident in another processor to which he is not directly connected. This is essentially a 'pass-through' facility which allows complete transparency through intervening machines to the required service. The link between the terminal user and the machine offering the service is achieved by the input of an appropriate simple message from the terminal keyboard.

An example of RSA in use might be when the terminal operator wishes to use a TP service which is not available on the machine to which he is directly connected. Once connected to the 'remote' TP service the operator is unaware that RSA is being used. Breaking the RSA link is as simple as establishing it; the terminal user then drops back to the machine environment to which he is directly connected.

In this type of organisation the machine 'passed through' is called the Agent and the machine offering the service is called the Server. The RSA connection is on a session basis, not message-by-message.

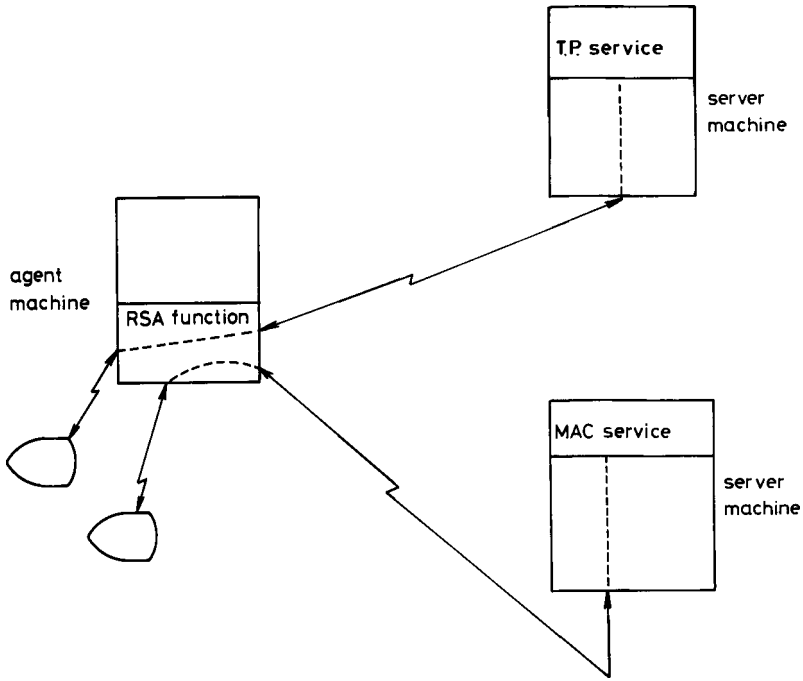


Fig. 6 Remote Session Access (RSA)

5.2 Distributed Message Router (DMR) (Fig. 7)

Some user input messages may relate to files or applications which are not available locally and which therefore need to be transmitted to a remote system for processing. Using DMR, any message input by the terminal operator will be examined for a destination indicator and routed to that destination, which may be the local or a remote mainframe.

This facility is of particular use where adjacent messages, for example in a stock control system, need to be processed against files or applications resident locally or in distant processors. DMR would be used to despatch the messages to the appropriate destinations. Once again the operator is unaware of this routing activity, except perhaps for a variation in response time resulting from the extra transmission delays.

Here, unlike RSA, routing activity is handled on a message-by-message basis.

5.3 Distributed Application Facility (DAF) (Fig. 8)

This enables the processing of different tasks, or different parts of the same task, to be distributed amongst different parts of the system or network. A terminal oper-

ator may send messages to a particular application for which data is not available locally, or which requires treatment by an application residing elsewhere in the processing network. Without operator intervention or awareness the local application will arrange transmission of either the original message or an entirely different one, depending on the requirements, to the appropriate distant application through the use of DAF. When the distant application has processed the message it may reply to the originating application in the local machine or it can optionally reply directly to the terminal operator.

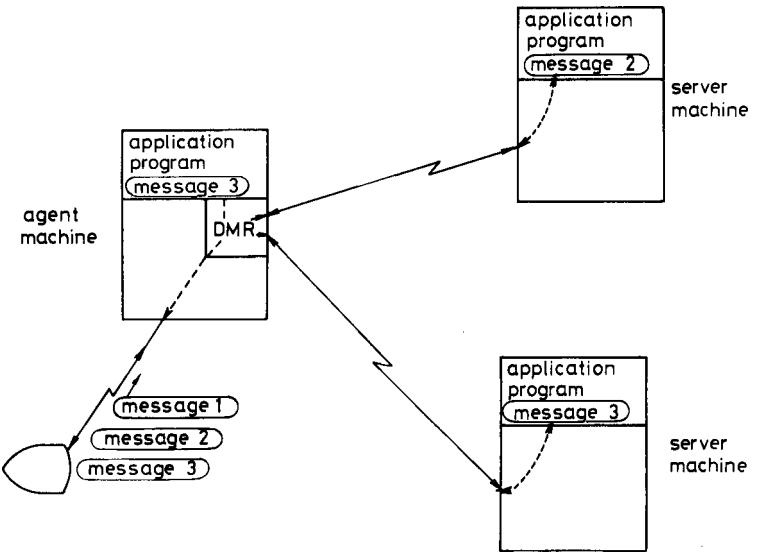


Fig. 7 Distributed Message Router (DMR)

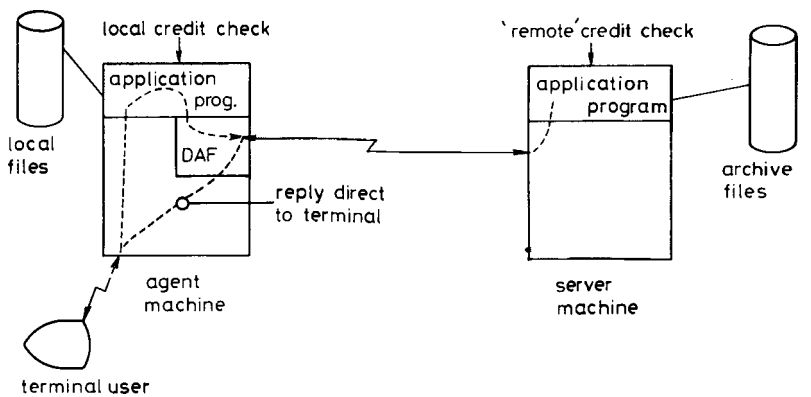


Fig. 8 Distributed Application Facility (DAF)

An example is the request of a credit check for a customer whose account has been idle for some time and whose records have been archived in order to minimise

local storage requirements. The local credit check program recognises that the account is not held locally and accordingly makes the enquiry upon the files held at the central site. When the reply is sent back from the centre the message can go either first to the local application or directly to the terminal user.

5.4 File Transfer Facility (FTF) (Fig. 9)

In conjunction with DAF, FTF will respond to commands in Job Control Language (JCL), for example from a terminal or from within an application program, to transfer a file between two systems.

An example might be the transfer of a summary of a day's local transactions at a branch office or shop to a central location for updating the central files. Another use of FTF could be the transmission of centrally developed source program files to remote systems for testing for local use.

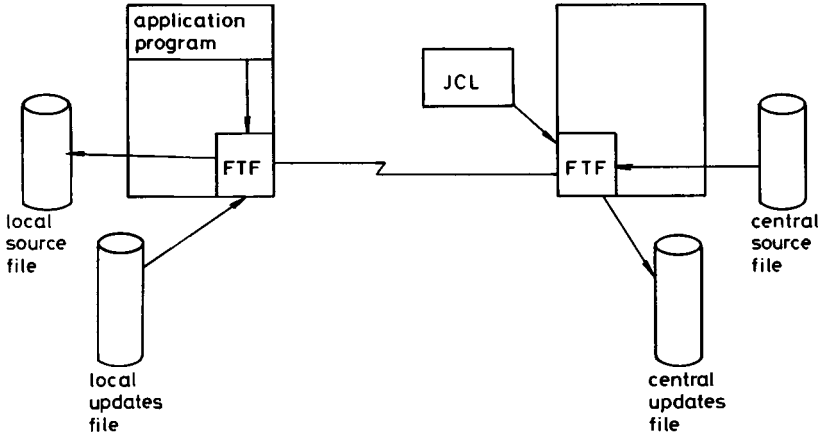


Fig. 9 File Transfer Facility (FTF)

5.5 Application Data Interchange (ADI) (Fig. 10)

This is the facility whereby an application program in one computer can communicate with another program in another computer without involving TP control or software. It is particularly useful when a high level dialogue is required between two related programs which are not operating on a TP message/transaction basis.

An example might be an overnight run of an application program which gathers the accumulated data and statistics of the previous day's work from a number of remote systems. The central and remote programs would have an intimate protocol relevant only to themselves for gathering the data.

Alternatively, and perhaps more important, as ADI offers a transparent 'pipe' through which data passes, it can be used for connection to alien (non-ICL)

machines, when the interpretation of the incoming data can be handled by the application within the ICL mainframe. In this way it is possible to have a meaningful conversation at the application level with other manufacturers' machines.

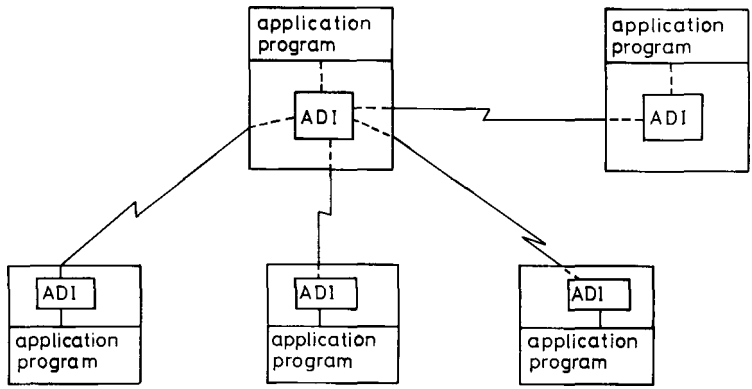


Fig. 10 Application Data Interchange (ADI)

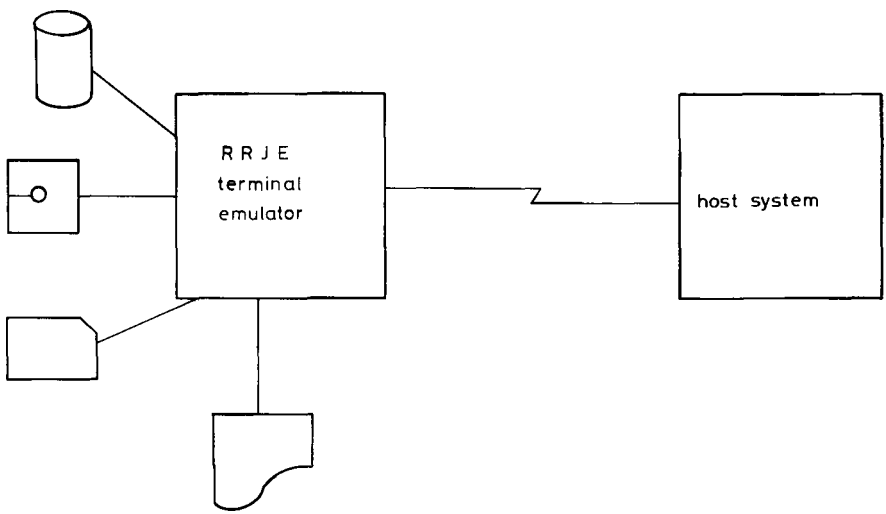


Fig. 11 Range Remote John Entry (RRJE)

5.6 Range Remote Job Entry (RRJE) (Fig. 11)

The RRJE facility offers the ability to input work for processing, including source code, data and JCL, from a machine emulating an RJE terminal to a host system. Output from the host can be received and spooled by the emulating machine, for eventual printing.

An example could be the transfer of jobs from one IPA host to another. Using RRJE, the entire JOB envelope – that is, the JCL, the job itself and any other

relevant information – could be moved from one host to another, executed and the results then despatched to the appropriate site.

6 Concluding remarks

This paper is intended as a broad overview of IPA. We have aimed to bring out the fundamental nature of the principles on which the architecture is based, its power and generality and its scope for unlimited development. As will be clear, a great deal of technical detail is involved in its implementation; it is intended that later papers in this *Journal* will deal with some of this.

Acknowledgments

The authors wish to thank Mr. D.J.Williams of ICL Product Development Group, Bracknell, for his technical help and advice and Dr. J. Howlett the Editor of the *ICL Technical Journal* for his advice of form of the paper.

References

1 HOULDSWORTH, J.: Standards for open network operation. *ICL Tech. J.*, 1978, 1, 50-65

2 ANON.: Reference Model of Open System Interconnection (Version 4 as of June 1979). ISO/TC97/SC16/N227

3 BRENNER, J.: Using Open System Interconnection standards *ICL Tech. J.*, 1980, 2 (1), 106-116

4 DAVIES, D.W., BARBER, D.L.A., PRICE, W.L. and SOLOMONIDES, C.M.: *Computer Networks and their protocols,* New York, Wiley, 1979.

VME/B a model for the realisation of a total system concept

B.C.Warboys
ICL Product Development Group, Northern Development Division, Kidsgrove, Staffs

Abstract

Almost a decade has elapsed since the foundations of the VME/B system design were laid. The paper is an attempt to isolate the fundamental initial design motivations. During the decade most exposure has inevitably been given to the extensive set of capabilities provided by the system. As a result the original motivations tend to be obscured by the, admittedly most important, facility discussions. It therefore seemed appropriate to restate the fundamentals uncluttered by details of such facility support. This may well cause disappointment to those who expect to find a facility overview within; however, I felt that a highly personal statement of the original design motivations would be worthwhile after such a long interval.

'The writers against religion, whilst they oppose every system, are wisely careful never to set up any of their own' (Edmund Burke).

No such precaution was taken by the author and in fact a decade has flown by since the lapse and this paper. The paper is both an attack on the then (1971) state of operating systems and an attempt to provide a positive response to that attack.

Inevitably when attempting to outline the initial design concepts for such a project after such a long interval there is some difficulty in separating current motivations from those of ten years beforehand. The paper is therefore somewhat of a mixture of history, present thoughts and future predictions. Apologies are offered for such self-indulgence; it is hoped that the reader will understand and forgive.

1 Scope

The reader who perseveres will read about system concepts and motivations. Little or no mention is made of the extensive processing and support capabilities of the VME/B system. These are well documented in a series of ICL technical publications to which the interested reader is referred.

Thus there is little or no description of the scheduling mechanisms, the control language, the filestore or the virtual store. It is hoped that some future papers in this series will provide technical background to some of these areas.

The VME/B system is currently packaged and marketed as a large general-purpose system supporting the concurrent operation of batch, multi-access, remote job entry and transaction processing services. It is offered to support large ICL 2900 series systems.

2 Introduction

In any description of an entity it is useful to detail not only what it is but also what it is not. The classical view of a computer operating system 'to *share* the hardware which it controls among a number of users, making unpredictable demands upon its resources ..... efficiently, reliably and unobtrusively' is too narrow to enable a reasonable presentation of VME/B's objectives and philosophy. There are so many preconceptions about the components of an operating system and even more about what its functions should be.

Instead the paper reflects the basic thinking behind the original design intentions that the time had come (in 1971) to challenge the boundaries and dichotomies which had grown up around that mythical entity 'the operating system'; to challenge the concept of a single unit capable of satisfying the diverse needs of all users.

Well, so the King is dead: now what do we do? Clearly the first task is to examine the origins of any legend in order to understand society's dependence on it; to examine the divisions and boundaries that give shape to the legend and distinguish between those that have prospered through the lack of any formulation of theorems, those that reflect the constraints of recent knowledge and practice and those that exist through our present conception of the very nature of the system we are providing.

The paper thus attempts to gradually identify what was perceived as the users needs. It demonstrates that VME/B is not just a 'product' but a unified and comprehensive set of system concepts, design rules and functions. These provide the basis for the development and integration of ICL products of many types, both existing and new. The paper seeks to establish that the 'packaged' VME/B system as currently preferred is a particular usage of the design and thence to differentiate between module capabilities and their packaging, mapping and binding into marketed systems.

Any set of software capabilities must choose how to colonise the wilderness between the rich pastures of available technical possibilities and the forest of users' demands for capabilities.

Such a colonisation requires the identification of the characteristics of the components, the style of their combination, and a supporting methodology allowing the flexible assemblage of components to provide the necessary application services (Fig. 1).

So what were the possibilities, what were the demands?

3 Technical possibilities

The sixties had generated a number of techniques:

structured programming, top-down design, procedures, capability machines, data-bases and communication protocols.

The architecture of the 2900 range had drawn from this base and had reflected the latest thinking into the structure within which module designers could begin to function. The 2900 architecture has been outlined elsewhere[1] and the reader is referred to such publications if further background reading is required.

From the viewpoint of the software module designer the architectural features which framed the basis for a sympathetic software implementation were as follows.

3.1 Architectural support for the procedure

Historically the dichotomy between software packages, in particular those which had come to be known as the operating system and the applications system, had grown to such an extent that the concept of privilege had enshrined a single level boundary between the 'good' guys with wide capabilities and the 'bad' guys with narrow ones. The justification, probably after the event, was that one of the principal roles of the operating system was to hide the details of hardware from the application writers. In practice one highly stylised and complex interface had merely been replaced by another.

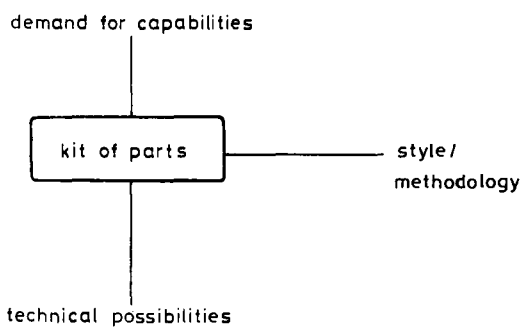


Fig. 1

The advent of high level languages and then of application data interfaces in fact presented the applications developer with a stylised high level interface which now required support from mapping code rather than an abstracted hardware interface. However few people had seemed to notice and a sharp distinction grew up between the mapping support provided by the 'run-time library' and that provided by the 'operating system'. The 360 architecture in fact provided two code-calling mechanisms something like 'BALR 14, 15' for library calls and something like 'SVC 23' for operating system calls.

Further, once within the operating system, it (the operating system) invoked procedures within itself using the library calling convention of BALR 14, 15. Thus three classes of procedures could be identified

(i) User available library procedures (subroutines)
(ii) Operating system interface procedures
(iii) Operating system available library procedures.

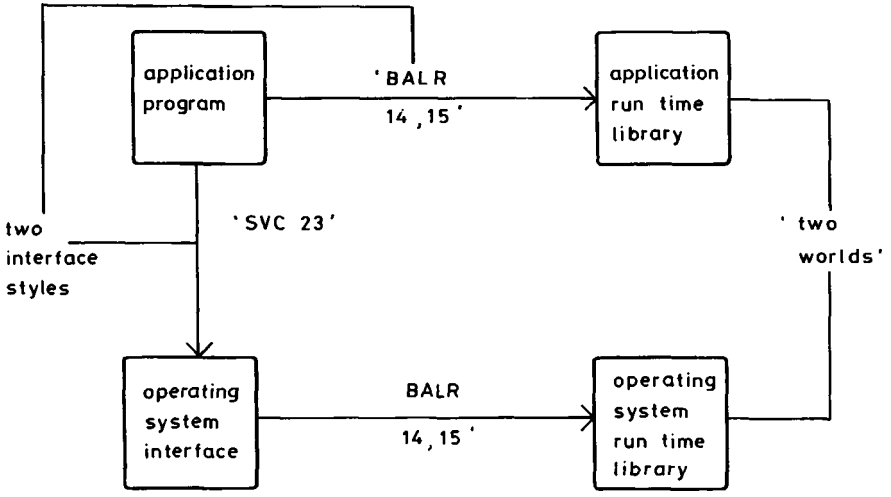


Fig. 2

Not only did this enshrine for the system the level and structuring of support offered by the various libraries but also cut off general access to operating system procedures which were of potential value to the user. Since that time the growth of microcode machines has introduced a fourth level of library: that of the hardware-provided microcode-procedure library.

It was clear that if we were going to exploit to the full the opportunities presented by more flexible hardware machines, the new lessons of data management and the wide use of high level application languages such distinctions would only inhibit the freedom for flexible system restructuring to meet the challenges of the rapid evolution of computing theory, technique and hardware economies.

Thus 2900 offered a number of facilities which were specifically designed in order to ensure that such distinctions were not pre-ordained for the system designer:

(*a*) The call (the procedure invocation) was a universal mechanism independent of the privilege and priority of the calling and called procedures.
(*b*) The binding of a called procedure to its caller was accomplished by the 'insertion' of a pointer (a call descriptor) between the call and the target procedure. This 'insertion' could be delayed, if necessary, until call time

and it was this generalisation of binding mechanisms which offered real opportunities to rethink the mechanisms needed to flexibly assemble a variety of application services.

(*c*) The 'work-space' for a procedure was mechanised, by a hardware stack, such that all procedures could operate in the same way. This was clearly a prerequisite for the delayed binding of procedures into application environments. There was little point in delaying the classification of procedures into levels of privilege if the code body of such procedures was dependent on the level chosen.

In practice, for efficiency reasons most binding is performed at either system generation, system initialisation or module load time as part of the packaging of the product for field release.

Clearly the possibilities available through such a general technique could not be ignored in the resulting software system design.

3.2 Architectural support for virtual machines

Historically the other major distinction which had become cemented into systems architecture reflected the need for multiple users to access the computing service simultaneously. To support such access most machines implemented some form of base-displacement protection mechanism limiting access by a user program to a 'local' store plus a set of operating system interface procedures, i.e. the SVCs mentioned above were included in the base-displacement domain. This mapping did much to enshrine the concept of a 'program' as the basic user processing unit and hence encourage the development of monolithic application programs to fit into these highly stylised protection domains. Further since the user had no access to any form of concurrency apart from the 'program level' a plethora of tasking techniques evolved such as to enable the applications writer to have some access to parallel processing capabilities. The combination of the lack of distinction between the protection of tasks and shared variables within a program led to a number of rather complex parallel language definitions culminating in PL/1-like approaches. Such approaches placed considerable strain on the skills of the applications developer. Indeed since then a number of treatises, books even, have been written attempting to provide aids in mastering such complexity.

2900 offered the possibility of a more thoughtful and general approach to sharing and concurrency through the notion of procedure capabilities within virtual machine domains. However, it must be admitted that the further developments particularly in information processing theory have demonstrated that in the future wider generalisations are going to be necessary. These will basically derive from a better understanding of the nature of naming and relationships in information processing. However, it is already clear that the lifting of many of the artificial barriers has presented better opportunities for developing a coherent approach.

The virtual machine in 2900 is thus viewed as the assembly point for a set of capabilities to provide an applications service. The procedure base offers a flexible

mechanism for mapping real world objects into capabilities and the scene is set enabling us to look at the provision of a set of modules enabling the assemblage of highly specific application-processing services.

However, at this point, it is worth noting the constraints, implicit in our style of business, which were imposed by the large number of established commercial practices, especially since the manufacturers had in the first place been responsible for the establishment of these practices. These included the rather peculiar style of mapping required for FORTRAN and COBOL applications and in ICL's case the style of control language made familiar to users through the George 3 operating system. Thus the virtual machine needed to become a flexible localised container in which opportunities for rationalisations could be exploited whilst allowing familiar techniques to continue to be exploited. Indeed the current style of command language interface, deriving from the George 3 vocabulary, is likely to continue for at least another decade.

4 Capabilities

So what processing services are necessary and what style and methodology is required to provide the framework for such services? To answer this question we must look at the pressures which were growing within the industry which in turn were dictating these requirements. There were two basic areas of pressure, one deriving from the economics of application software development and one from the technological developments in the science and practice of the industry.

4.1 Application development

From the viewpoint of the computer user striving for a cost-effective applications service there appeared to be three problems which were of particular concern and where an increased application of software engineering principles could be of immense benefit to him.

(i) Program duplication – duplication in his own programming because of ignorance of the work of others, differing languages, change of computing system or partial change of requirements, and duplication of system support software, which in the final analysis he has to pay for.
(ii) The poor design and implementation of the images presented to the user and their irrational variation from one application to another.
(iii) The management of large application program suites – getting them written, used and maintained.

4.2 Technology development

From the state of the industry a number of pressures were arising and these can broadly be divided into the three areas of Hardware technology, Information technology and System and Information theory.

4.2.1 Hardware technology:

(*a*) Processing power. The cost of hardware was reducing rapidly which generated the twin challenges of the effective utilisation of more power by the competent user and the utilisation of a powerful system by an increasing population of non-professional users.

(*b*) Accessible storage. Most social and business revolutions have derived from the ready availability of an enlarged information base. The huge explosion in the amounts of accessible storage over the last decade and its even greater explosion in the next posed problems of distributed databases, friendly interfaces and rapid access and protection of very large information banks. Clearly an approach to filestore, database structure and naming was required that both coped with existing skills and practices but at the same time left the door open to deal with new vastly different demands.

(*c*) Novel machines. It was becoming clear that the general purpose machines prevalent in 1971 would rapidly be complemented by a whole 'mish-mash' of special purpose engines. The ICL DAP and CAFS machines are already available and there are a number of 'Reduction' machines which have demonstrated the potential of very high and functional programming languages as directly executable languages. Clearly the system partitioning and hardware mapping would have to take account of this aspect.

(*d*) Networks. The network business was taking off spurred on by a number of influences:

the growth of distributed processing in the market place;

the demands to interlink data-processing systems and components in a flexible way;

the mutterings about automatic offices and electronic mail.

Clearly no longer could communications facilities be treated as add-on goodies to a batch processing service; an integrated approach to both centralised and distributed processing was required. See for example the paper by Kemp and Reynolds[2] on the recently-released ICL Information Processing Architecture, IPA.

4.2.2 Information technology: File organisations had arrived, indexed sequential, keys, records, were accepted jargon. The CODASYL work and the work of CODD[3] were producing implementations of management information systems with immense scope and potential. There was clearly a need both to absorb the lessons on naming conventions, data structuring and relational theory and to produce solutions which allowed freedom of evolution of the applications database independently of the applications suite. In particular many applications would for the first time be changing from tape-based unconnected file-processing approaches to disc-based interconnected integrated management-information systems.

4.2.3 System and information theory: It was the era of 'modularity' – the need to get rid of monoliths and move to off-the-shelf interchangeable applications modules together with a style and a methodology for packaging such modules.

4.3 Summary of required capabilities

(i) Cost-effective applications development environment, in particular the availability of modular programming support techniques
(ii) 'Synergetic' man machine interfaces
(iii) Exploitation of very large information bases and current information theory
(iv) Exploitation of 'novel' machines
(v) Exploitation of distributed systems

5 Response

5.1 Recap on the story so far

The paper has thus far attempted to outline the three main areas of influence.

(i) The existence of historical boundaries and the opportunities offered by the 2900 architecture as a basis for a more flexible approach to technological possibilities.
(ii) The challenges presented by a whole new set of technological possibilities.
(iii) The demands for capabilities from an often embittered user population.

The rest of the paper is an attempt to demonstrate the extent to which the system style has grasped these opportunities and challenges and gone some way to understanding how much of operating system sentiment was fact and how much fiction.

5.2 Functionally-oriented environments

Earlier the paper had identified that the set of capabilities which were needed effectively demanded a style, a set of mechanisms for easing the lot of the developers of applications systems. It had identified that historically monolithic applications development had derived from two main sources.

(*a*) The widespread classification into two worlds – operating system and application program.
(*b*) The lack of support for modular program development systems.

It then identified the basic elements of the 2900 building style, that is, procedures and virtual machines. Particular emphasis was given to the concept of delayed binding of procedures to support the mapping of a set of real-world entities into a set of functionally oriented environments existing within the boundaries of an architecturally supported Virtual Machine. Further in performing this mapping we did not wish to reintroduce unnecessary boundaries but rather allow as much freedom as possible in the selection of procedures to support a given set of application-provided packages. The final requirement was clearly to then provide a generalised

naming and control structure allowing the flexible assemblage of procedures to present the required application images.

In order that such environments could freely exist it was necessary to ensure that:

(*a*) we were not building interpreters on top of interpreters and thereby creating systems with a certain hierachic elegance but very poor performance;

(*b*) the maximum use was made of defaults and naming contexts in order to localise identifiers;

(*c*) once packaged environments had been established they would be re-used (inherited) in order that the minimum amount of time was spent in creating the tailor-made environments. Clearly the object of the exercise was the execution of work in such environments rather than their initial establishment.

(*d*) the mechanisms for the building of a set of 'user-friendly' interfaces were freely available. We took the view that no single definition of user-friendly was likely to serve the needs of all users. Further the state of the industry was such that the definition of 'friendly' was likely to evolve rapidly in the next two decades or so.

The approach taken to ensuring that such objectives were indeed satisfied was:

(i) to provide a single naming system within which all real-world objects could be described, the relations between them could be specified and a set of local (i.e. environment specific) 'currencies' could be used to access such objects once they had been mapped ('selected') from the real world into the processing environment; see section 5.3.5 for further explanation;

(ii) to allow such currencies to identify (and hence directly invoke) primitives which were needed to support the environment and hence avoid the need for intermediate interpreters;

(iii) to provide a block-structured scoping system allowing both the re-use of tailored nested environments and to ensure that such environments were tidily deleted when no longer required. The system adopted was very similar to that used for variables in any Algol-like language. However, all real-world entities when mapped through the naming (and thence procedural support) system were subject to the same scoping rules;

(iv) to provide a 'profile' system allowing for the establishment of a variety of human input command languages which controlled the assembling and thence execution of the set of procedures required to support a given application.

Note that in all of the above there has been no necessity to distinguish between the providers of procedures. We shall however, identify functions which need 'public' access (and hence privilege) in order to support this model. It is such functions which help to identify the classical operating system functions which are 'real' and those which evolved through habit or manufacturer's policies.

Notice also that, as yet, no real mention has been made of the mapping of information and programs onto storage. Indeed this paper will not attempt to describe the various sophisticated techniques which are needed to ensure an effective utilisation of file storage and main store. Clearly much of the system implementation was and is concerned with this mapping but this is seen as of secondary strategic importance to the basic concepts outlined above. Indeed it was imperative that with the rapid evolution of storage technology the mapping of the storage hierarchy should be flexible and distinct from the support given to the general system structure.

5.3 Basic system principles

Thus the system was viewed in concept as supporting four basic activities.

(*a*) System description – involving the description of all of the named objects required to be accessed by the various users and managers of a distributed computer system. Note no distinction has had to be made between operator, system manager, programmer or application user. This system description may turn out to be distributed throughout a network and indeed encompass a variety of different naming systems and standards.

(*b*) User entry – involving the creation and management (and thence deletion) of an environment tailored to meet the demands of this particular user within a set of installation defined user types.

(*c*) Object naming – being the mechanism required for the provision of a local view of the generic system description. In particular in a distributed network it was considered of primary importance that the user was given a consistent and personalised way of naming an object independent of the controlling system managing the object.

(*d*) Object manipulation – once an object had been named, the user needed a local 'currency' guaranteeing his required private access which when subsequently used to manipulate that object would police its conformance to the behaviour which he had established for it (rather like the concept of types, classes and monitors).

Fig. 3 outlines the entities and environments which support these basic activities.

5.3.1 Loading: Before more detailed analysis of these basic system activities there is one point worth reiterating. The basic 2900 approach has been viewed as providing mechanisms for the translation of any activity into a set of procedure capabilities. Clearly the procedures provided from a number of sources are themselves named objects. Thus when visualising named objects one should remember that the procedures supporting them are included in that set. In other words the system is basically providing mechanisms for *name translation* and the naming system copes both with the naming of primary objects (e.g. users, files, devices) and their supporting derived objects (e.g. procedures, events, data areas). The process of providing named procedures to support object selection and manipulation is known as 'loading'. Thus basically whenever a user names and then manipulates an object the system 'dynamically' provides the named procedures for supporting these opera-

tions, hence the importance of the delayed binding and run-time privilege classification mechanisms outlined earlier.

As noted earlier the system generation (packaging) process can of course preload to optimise the process for identified packaged requirements. In fact VME/B, currently marketed as an 'Operating System' product, is mostly packaged this way at present.

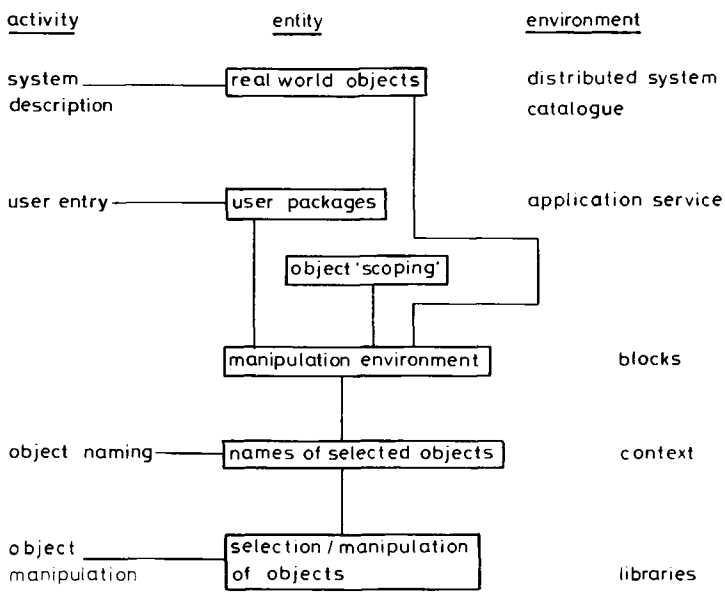


Fig. 3

5.3.2 System description: To support a complete description of a set of named objects the concept of a *names catalogue* was introduced. As a design statement the catalogue was not dependent on any processing centre of a distributed system and indeed might appear in several such centres. It is organised in terms of nodes and relationships. Entries for named objects are located at *nodes* and the nodes are connected by *relationships.* For example a file node is connected to a volume node by a placement relationship.

Privacy relationships define access rights and in particular control the translation (mapping) possible between objects and the implied support procedures. Thus rather than provide a rigid partitioning between operating system capabilities (functions) and user-provided procedures the system provides, through the loader and catalogue mechanisms, a general capability for establishing as part of a system description the mapping between objects and the privilege, priority etc of the underlying and often implied support procedures. Clearly once we have established a global, protected loading and cataloguing system we can generate secure systems.

5.3.3 User entry: User entry to the system is viewed merely as the 'selection' of a user node. Clearly as a result of such a selection the implied support procedure

(normally referred to as LOGIN) can select required (allowed) processing environments and required (allowed) manipulation facilities. The capabilities (provided through named procedures) reflecting the installation's view of this particular user's needs and competence.

However, more is required. The original objective was to hide the complexity of translation between the user's view and the generic description of the network. Further in an 'open' system one might expect to see several different underlying network views and indeed (with the advent of CAFS, DAP and other novel machines) different types of processing capabilities. Thus the system provides two other basic mechanisms:

(*a*) implicit 'loading' and thence 'unloading' of capabilities. This is accomplished as mentioned earlier by giving scope to all loaded capabilities and selected objects. The system adopted was the simplest. That is a block-structured system such as is used to control variables in ALGOL 60. Most 'loading' is implicit since the system objective was to provide the user with access to named objects and the underlying support procedures vary according to the hardware supporting any application. Clearly the support procedures required for mapping to an array processor are different from those required to support a conventional machine.

(*b*) re-use and sharing capabilities. Clearly the establishment (binding) of procedures to support each and every named object takes a great deal of processing time but once established many environments rarely alter (e.g. a cash dispenser terminal in a banking network). Hence the system supports the concept of multilevel sharing and re-use of environments. This allows the installation to produce (system generate) a set of prepackaged environments with selected capabilities; the previously mentioned LOGIN procedure then merely has to link to the appropriate member upon user entry to the network.

5.3.4 Object naming: The paper has so far merely established a general mechanism for naming objects within a network and providing a tailored environment for their subsequent processing. Clearly the user having been presented with a tailored processing environment now needs to access objects using purely local terms. He does not wish to understand other people's naming system nor indeed to know all the names and naming structures in the universe. Thus the system provides a context mechanism. This can be viewed as the glue necessary for relating a purely local name to a fully qualified name within the network catalogue(s).

Thus a series of translation tables are provided allowing the user to gain coherent access to the naming systems within the network and thus to gain independence from any given processing node's naming structure. In a world of ever increasing open system interconnection and thence rapidly expanding information bases it is imperative that the necessary mechanisms be provided for the user to have a single coherent naming system.

5.3.5 Object manipulation: We have now provided the mechanisms for users to select objects into a tailored personalised processing environment. What remains is

to provide processing capabilities. The naming of an object is not the end of the matter; clearly one needs mechanisms for describing the 'correct' behaviour of any entity processing that object. Thus in order to identify the specific selection of a named object for a specific purpose a system of 'currencies' was introduced. A currency is a local pointer to a specific object selection. It thus acts as a 'bookmark' remembering where this user has accessed the catalogue. It further acts as a description of the permitted accesses (including other people's right to concurrent access) afforded by a particular selection or series of selections. And lastly it gives the route to the 'possibly implicitly' loaded procedure capabilities that give the facilities for manipulation of the selected object.

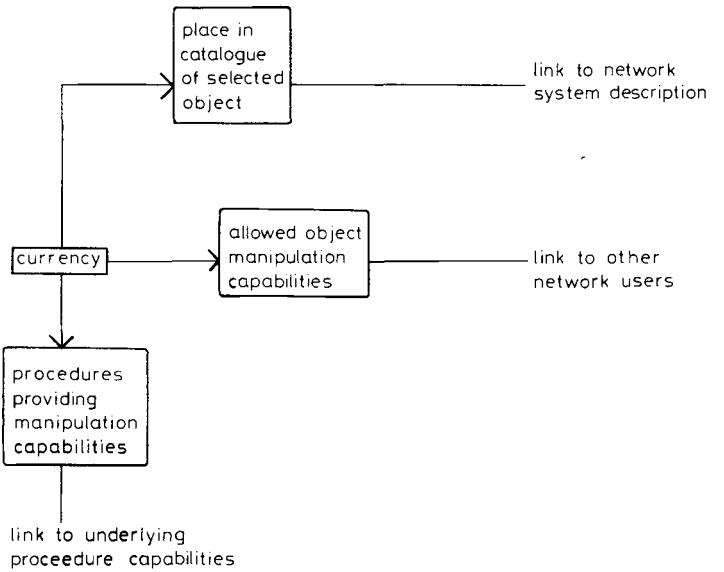


Fig. 4

Thus as an example a user may select a real file and thence a file manager and thence a record. The combination of file, processing agent (i.e. access method), file position and processing buffer resulting in a record currency. In the case of a serial file therefore repeated usage of a record currency might always yield 'the next record' into the processing buffer.

Clearly the support of such currencies is undertaken by a whole series of procedures specifically designed to support a set of required processing capabilities. The set is infinite and the role changing as the hardware interface moves and as the definition of the object types and operations required to support networked information processing is developed.

5.3.6 Applications development: The model established so far has only dealt with naming, name translation into procedure capabilities and environment provision. In passing it has dealt with the demands generated as a result of wishing to exploit

the capabilities of distributed systems and novel machines. It has as yet not sought to specifically satisfy the demands for synergetic man–machine interfaces and cost–effective applications development environments. The model has however identified the mechanisms for the provision of such processing environments and the underlying support necessary for the implicit 'loading' and 'unloading' of the supporting procedure based capabilities.

What additional capabilities are required to support these other demands? Basically the system design took the view that there was little to choose between the requirements necessary for basic system structuring and those required for application structuring. That is to develop applications as a set of modular capabilities, to allow for the flexible assemblage of such modules into services, to provide the mechanisms to create and thence re-use packaged environments and to provide maximum transparency of required capabilities to the end user. The list of requirements is the same as has been identified earlier. Thus the capabilities:

(*a*) to extend the catalogue – system description;
(*b*) to extend environment packaging – user entry;
(*c*) to extend the context mechanism – object naming:
(*d*) to extend the currency (procedure capabilities) mechanisms – object manipulation,

provide the basics for modular applications development. The capabilities are augmented by a System Control Language[4] which reflects

(*a*) the block-structured scoping of objects;
(*b*) the procedure-based capabilities system;
(*c*) the need to provide a single language for all applications binding since we have not distinguished types of user as a basis for system design but rather as a packaging requirement;
(*d*) the need to combine application modules, system procedures into processing services and hence provides good communication (e.g. variables) facilities;

Thus one can see that the control language is not an 'add-on goody' but rather a continuous extension of the basic system concepts.

6 Conclusion

The paper has attempted to identify the causes of monolithic and expensive operating system development and thence monolithic and expensive applications development. It has then identified what the VME/B design identified as the four main system concepts necessary for the support of both distributed and modular system developments. That is, system description, user entry, object naming and object manipulation. It has not discussed many things including concurrency, file-store or hardware management. That is not to say that these are not important, in fact, of course they are the essence of what one might term the operating system procedures contributions. The reader will, one hopes, see that the provision of such manipulation capabilities is secondary to the provision of a coherent naming and

processing capabilities system and indeed that the implementation of such manipulative functions will rapidly evolve as the economics of system construction change over the next decade.

Acknowledgments

Thanks are due to many people: Di Turner for painstakingly preparing the document, Jack Howlett for bullying me into writing it, Peter Wharton and Barry Hopewell for providing inspirational notes, all the members of OSTECH, past and present who provided me with such splendid design support during the past decade, the members of 2900 OSS, past and present, who turned designer's dreams into product realities and finally to those in other areas of ICL who have dealt with the consequences of these product realities.

References

1 BUCKLE, J.K.: 'The origins of the 2900 series', *ICL Tech. J.* 1978, 1, 5.
2 KEMP, J. and REYNOLDS, R.: 'The ICL Information Processing Architecture IPA' *ICL Tech. J.* 1980, 2(2).
3 CODD, E.F.: 'Normalised data base structure', IBM report RJ935, 1971.
4 BRUNT, R.F. and TUFFS, D.E.: 'A user-oriented approach to control languages', *Software practise and experience*, 6, 93, 108.

Birds, Bs and CRTs

I.D. MacArthur
ICL Communications System Segment, Northern Development Division
Kidsgrove, Staffs

Abstract

Nowadays it is popular to criticise Visual Display Units (VDUs) as a possible hazard to their operators. ICL is well aware of the concern in this area and this paper presents the views of one VDU designer. Cathode Ray Tubes (CRTs) have been around for more than 70 years and have been used for data display since the earliest days of computers. It is the author's considered belief that we shall be living with them for many years yet.

The purpose of this paper is to spell out some of the basic facts of VDU design which apply to all VDUs made by all manufacturers. In this way it is hoped to present in a compact form some of the facts which affect the appearance of the VDU screen. It must be stressed at the outset that there are very few absolute truths in the design of VDUs; most of the decisions are subjective and there are few standards to guide the engineer. The main sections of the paper deal with operators and their needs, VDU design, physical CRT parameters and safety and health.

The Bs are all around us: Business – Big Business – Brothers – Big Brothers – Breadwinners – Boredom – Backache. The Birds are of course our delightful operators.

1 The operator

The whole purpose of a VDU is to communicate something speedily to an operator. VDUs do this by displaying alpha-numeric characters and sometimes graphics on the face of a CRT or other device.

If the operator is totally absorbed in the task of debugging a program, designing a bridge or taking over a company then past experience suggests he or she is unlikely to complain about the VDU. However, as VDUs are applied to more and more mundane tasks with less job interest, a bored office person is likely to find fault with her environment and colleagues. I think it is the growing penetration of VDUs into office work that makes them currently a target for criticism.

As a VDU designer I feel obliged to attempt to deflect the operator's wrath from the VDU and redirect it where it properly belongs – against the furniture, canteen facilities, air conditioning and management shortcomings in general. To this end it is a good idea to study the operator's vital statistics and make sure that the VDU is designed to harmonise rather than conflict with her needs.

This subject has been very well researched and a recent report[1] is recommended to the reader who wants more detail. A good summary is also provided in another report[2] which gives general recommendations while omitting much of the detailed background.

It must be stressed that even the best VDU in the world can be rendered unusable if badly installed, particularly in its siting relatively to windows and lights. The design of desks, chairs and office equipment generally is well covered in the above reports.

Whilst most of the operators' requirements are well covered, a few points relating to the design of VDUs are worth considering.

1.1 Environmental needs

Apart from the visual factors already mentioned and the usual office environment considerations of temperature, humidity and ventilation, there are one or two points concerning VDUs and their operators.

(*a*) *Noise:* Although VDUs are nominally silent it is surprising how much noise a proficient typist can make on a 'silent' keyboard. Whereas this may be quite insignificant in a computer room or in a 'data prep' area, it can cause annoyance in a very quiet office. It should be realised however that the quietness of a keyboard is related to its feel, and a very quiet keyboard may feel soft and lack a positive tactile response – one of the many compromises in VDU design must be invoked here. The same applies to fans, which can range from the Kiplingesque to the miniature RB211.

(*b*) *Cooling:* Fans have been mentioned and are generally an unmitigated nuisance. A blast of hot air in the face from a neighbouring VDU will usually result in retaliatory action. The experienced operator can usually overcome this particular problem by sticking some paper over the air outlet with selotape: the reader is left to guess what that does to the internal temperature of the VDU.

All electronic equipment dissipates heat to some extent and the 100W or so from a typical VDU can normally be handled by convection. For comparison, a human body also dissipates about 100W.

It is clearly good design to run equipment as cool as possible and to this end adequate ventilation must be provided; but care should be taken to see that warm air is directed away from the operator.

2 VDU design

2.1 Character shapes and sizes

It is generally accepted that the optimum size of character for viewing subtends an angle of about 20 min of arc corresponding to a size of 3 mm when viewed from a typical distance of 600 mm (2 ft).

Johann Gutenberg is reputed to have invented printing with moveable type about 1448 and printers have been perfecting the technique ever since. It would therefore seem reasonable to assume that after more than 500 years development, printed characters have achieved acceptable proportions. A number of modern type fonts have been analysed (see Appendix A) yielding the quoted proportions, which I believe represent the optimum for viewing by today's operators. Similarly various printed texts have been analysed to yield optimum line spacing (see Appendix B).

From these analyses it would appear that the CRT screen should accommodate a display width of about 240 mm and a height of 160 mm for 25 rows of 80 characters.

These figures are in good agreement with what is achievable on a 15 in diagonal CRT. A 12 in CRT is obviously useable, but will need a closer viewing distance. Fig. 1 summarises the optimum dimensions.

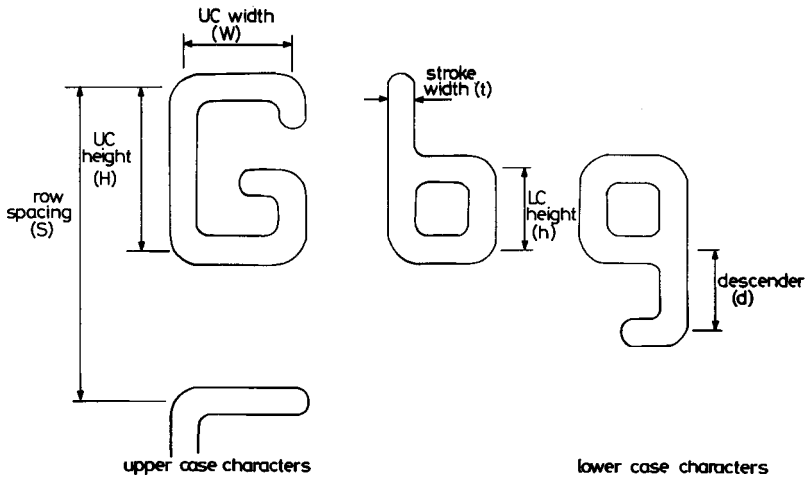


Fig. 1 Optimum character parameters

$\frac{H}{w} = 1{\cdot}63\ \frac{H}{h} = 1{\cdot}49\ \frac{d}{h} = 0{\cdot}43\ \frac{S}{H} = 1{\cdot}9\ \frac{t}{H} = 0{\cdot}13$

2.2 Scanning of character generation

Let us now turn our attention to how such screenfuls of data are to be generated. Unless storage tubes are to be used it is necessary to refresh the display. This means scanning the same data repeatedly so that it appears as a steady flicker-free picture – more on this anon.

Fig. 2 shows a simple representation of a magnetically deflected CRT such as used in virtually all modern VDUs. Since a CRT generates only a single spot of light, characters are generated by deflecting this spot using magnetic fields. There are four well established techniques for doing this.

The oldest, most versatile and unquestionably most expensive technique shown in Fig. 3*a* is the cursive drawing where the electron beam is deflected around the outline of each character. This technique is widely used in large graphical and radar displays. It is capable of very good results but is unlikely to be seen in a display costing less than £20,000.

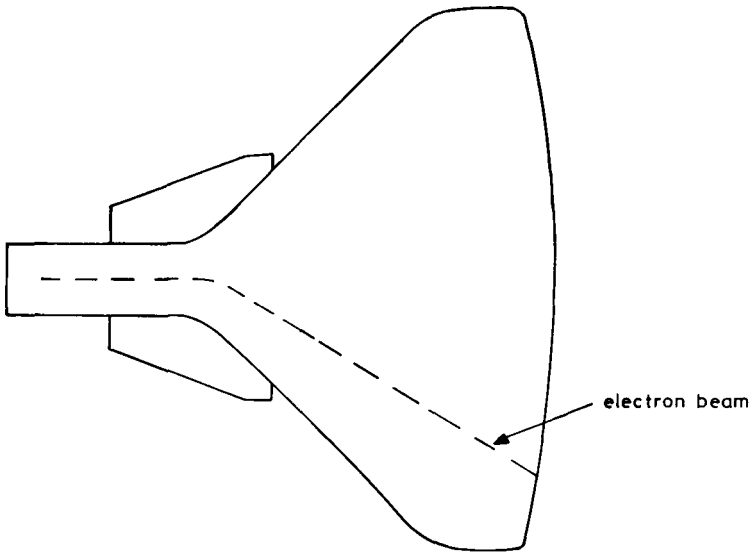


Fig. 2 Magnetically deflected CRT

Jiggle scan (Fig. 3*b*), where the beam is deflected very quickly to scan the character position combined with slower vertical and horizontal scans, is capable of providing a very good display at moderate cost. In the early days of VDUs this scanning technique was much used largely because it required a store access every 10 μs or so, which was easily obtained using delay-line techniques. Nowadays, the availability of faster semiconductor stores and the cost of the additional scanning circuitry have largely caused the jiggle scan technique to be superseded by raster scan displays.

All television (TV) sets use the raster scan (Fig. 3*c*) where the electron beam is deflected by two continuously running scanning circuits normally called the *line* and *field* scan circuits. The line circuit usually deflects the beam horizontally while the field deflects it vertically. I shall have a lot more to say about line and field frequencies, but for the moment it suffices to say that European TV uses a line frequency of 15625 Hz and a field frequency of 50 Hz.

A development of the raster scan is the step scan raster (Fig. 3*d*) where the vertical scan is not uniform but speeds up between rows. Since the amount of speed up can be adjusted, it is now possible to adjust the character height independently of overall height. Table 1 summarises the pros and cons of raster and step scans.

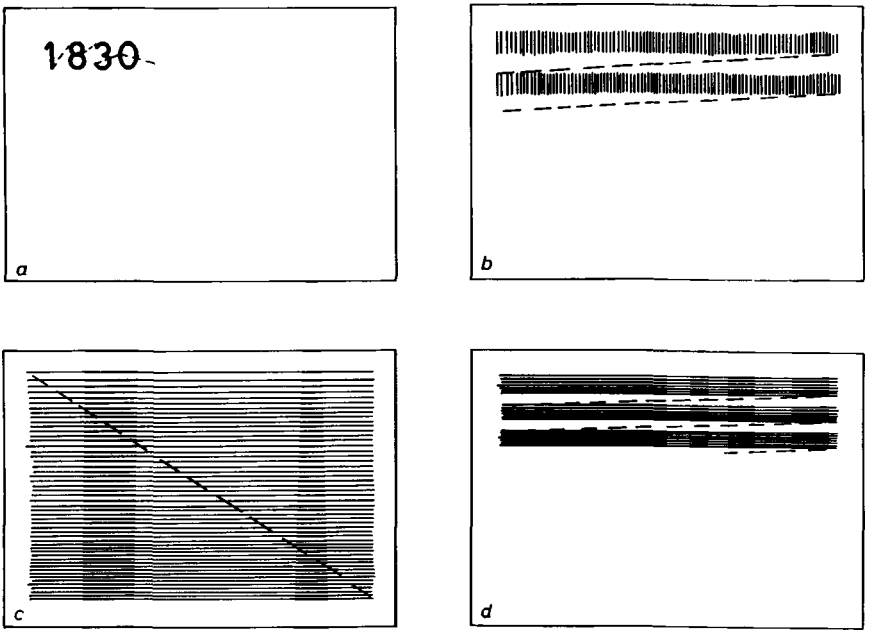


Fig. 3 (*a*) Cursive scan
(*b*) Jiggle scan
(*c*) Raster scan
(*d*) Step scan

2.3 Character generation

Fundamentally a CRT screen is a matrix of possible display dots – something like ¼ million of them and each one must be scanned ***at least*** 50 times per second, allowing less than 100 ns for each dot.

We therefore have a number of interrelated parameters to work with.

(*a*) CRT spot size and amount of defocussing
(*b*) Scan line spacing
(*c*) Number of dots in the character cell
(*d*) Number characters on the screen
(*e*) Line frequency
(*f*) Field frequency
(*g*) Acceptable character quality
(*h*) Acceptable cost.

(*a*) *Spot size:* If the spot size is too large then it will be difficult to read the characters – especially some lower case ones. If the spot size is too small the character will appear dotty and unpleasant to the eye. If the spot size varies much from one part of the screen to another, again the eye will notice this and be distracted.

Table 1 Raster scan pros and cons

Pro			*Con*		
Low cost	–	50 years or so development of TV has eroded the costs of the components to rock bottom.	Timing	–	as the line scan frequency is increased this timing gets more difficult owing to the energy recovery circuits' limited speed.
Low power	–	since the scans are repetitive and of constant amplitude it is normal to use energy-recovery drive circuits.	Coarseness	–	the TV lines are fixed distance apart tending to produce dotty characters on the screen.
Compatible	–	provided line and field frequencies are equivalent a data display can also display TV pictures.	Fixed character height	–	since a finite number of lines must be allocated to each row of characters and also to the inter-row gap, a compromise between spacing and character height must be made.
Transmission	–	well established waveforms for transmission by single coaxial cable.			
Interlace	–	can be used but beware of flicker.			

Step scan pros and cons

Pro			*Con*		
Adjustment	–	adjustment of character height allowing increased matrix size, e.g. 9 x 7 or greater.	Cannot	–	display TV pictures or contiguous graphics.
			Incompatible	–	cannot be displayed on standard TV monitors.
Transmission	–	can be transmitted by coaxial cable just like conventional raster.	Cost	–	more expensive than raster scan adding perhaps 5% to cost of scan circuits.
Better display	–	display quality is noticeably better than for raster scan.			

CRT spots are not clear, round, sharp-edged circles as sometimes thought (see Fig. 4). The light output varies across the diameter of the spot in a nearly Gaussian fashion making the spot appear to have fuzzy edges. Further, the spot may not be circular but become pear shaped due to astigmatism in the electron optical system and defocussed due to the deflection.

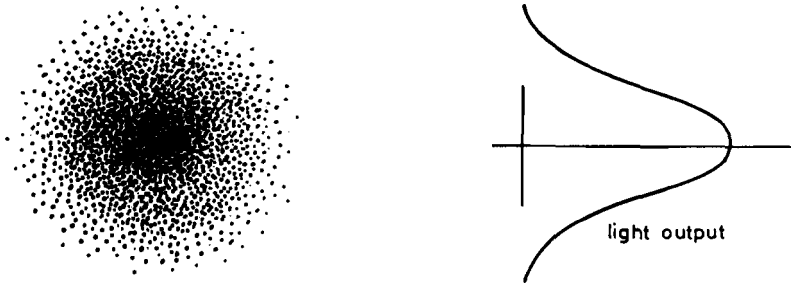


Fig. 4 Typical CRT spot

(*b*) *Scan line spacing:* Here the choice is easier. If a TV raster is used then the spacing is determined by the number of lines and the overall height. If a step scan is used then the spacing is adjustable anyway: this highlights one of the CRT's main advantages over the matrix panel, namely that the spot size can be larger than the matrix of the character generator, thus eliminating the dotty appearance associated with plasma panels and similar devices.

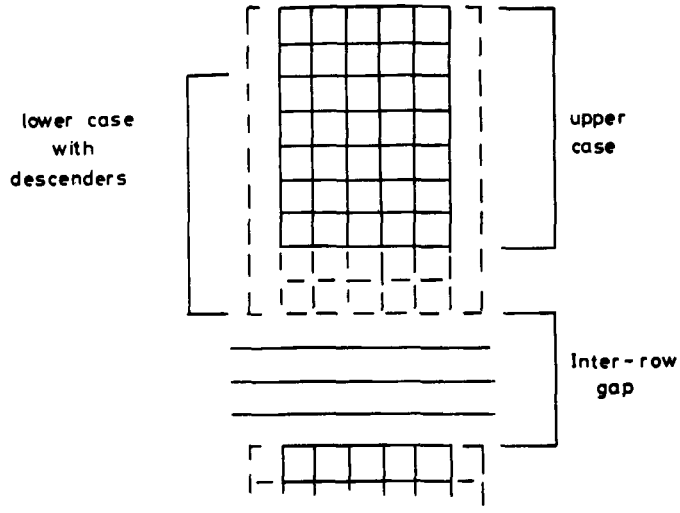


Fig. 5 5 x 7 character cell

(*c*) *Number of dots per character:* In order to cope with characters like A M G E, etc., it is normal to provide a character cell with an odd number of dots in each direction so as to retain the character symmetry. In general the more dots we use

the better the quality of the character, but as I show below additional dots are expensive, especially in terms of bandwidth and light output.

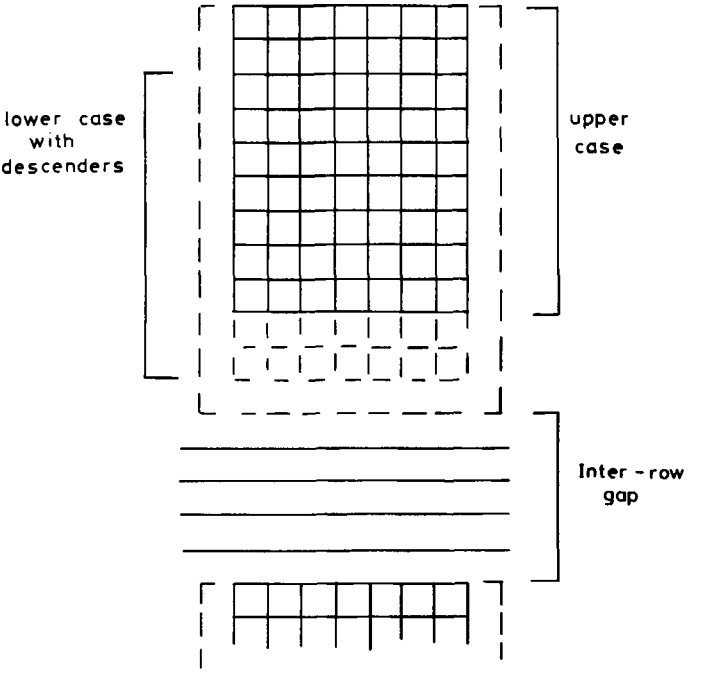


Fig. 6 7 x 9 character cell

Much the most common cell size is known as 5 x 7, meaning an upper case letter cell 5 dots wide and 7 dots high: see Fig. 5. Another popular cell size is 7 x 9, shown in Fig. 6. To provide correctly proportioned characters as defined earlier it is necessary to provide four lines in the inter-row gap for a 5 x 7 and five lines for a 7 x 9, giving a total of 13 or 17 lines per row and leading to the need for a lot of lines per page.

The effect of number of lines on line and field frequencies is shown in Fig. 7: the basis of the derivation is given in Appendix C. This shows the limits imposed by flicker and acoustic noise.

The effect of line frequency on dot time and light output is shown in Figs. 8 and 9 for 5 x 7 and 7 x 9 cells, respectively, and the derivation is given in Appendix D. It will be observed how the light output is reduced by increasing the matrix size and the corresponding line rate.

(*d*) *Interlace:* Interlace is a method of apparently increasing the number of lines per page. The technique is to produce two interleaved fields as shown in Fig. 10. Here for clarity I have reduced the number of lines per field to 5½.

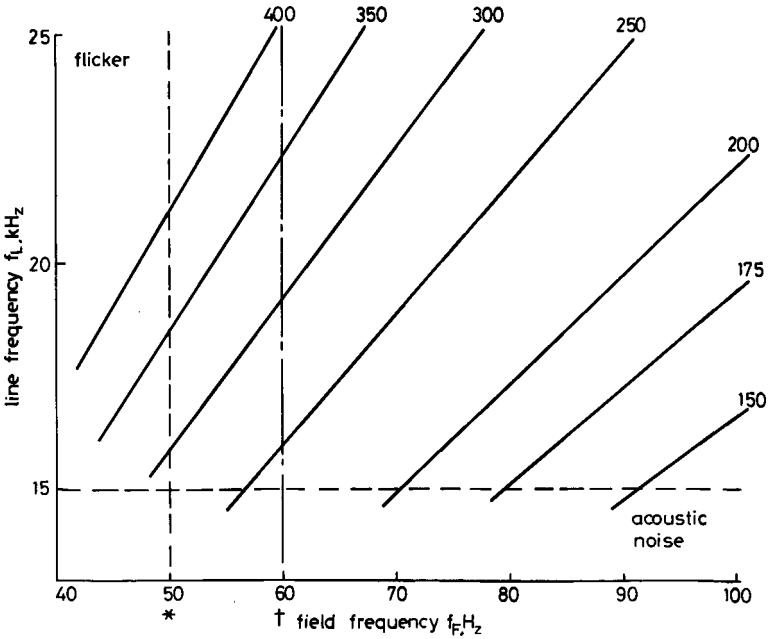


Fig. 7 Number of displayed lines as a function of line and field frequencies. See Appendix C for derivation.
*European mains frequency
† US mains frequency

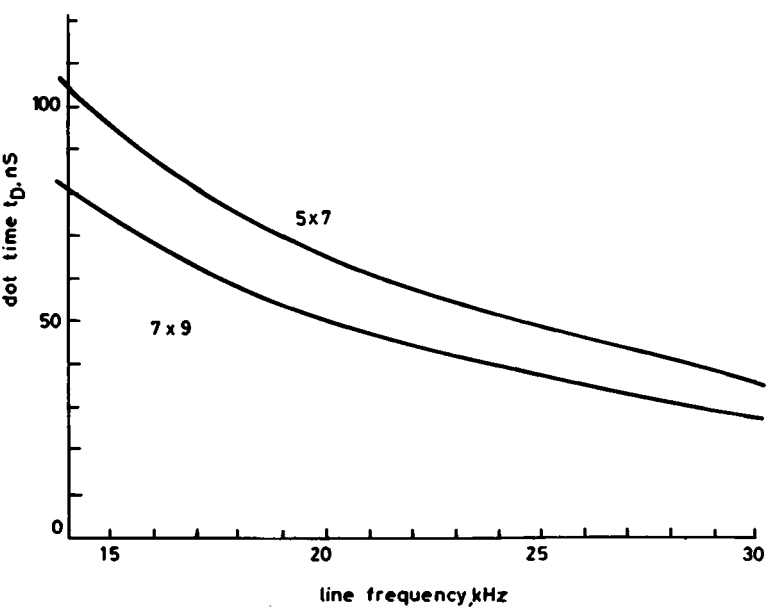


Fig. 8 Showing reduction in dot time (increase in video bandwidth) caused by increasing matrix size and line rate to give better resolution

In European TV for example each field contains 312½ lines, giving a total of 625 lines. Of course some of these are consumed by field flyback so that a typical TV picture actually contains about 525 viewable lines. This seems like good value until one realises that the use of interlace causes increased flicker, since the refresh rate is now effectively halved. This is of no consequence in TV applications but in VDUs, where the viewing distance may be only 500 mm or so, it is crucial. It is especially important to realise that each field could contain exactly the same number of half lines if severe flicker problems are to be avoided. My observations suggest that interlace flicker will be objectionable if the angle subtended by two adjacent TV lines is greater than about 30 seconds of arc.

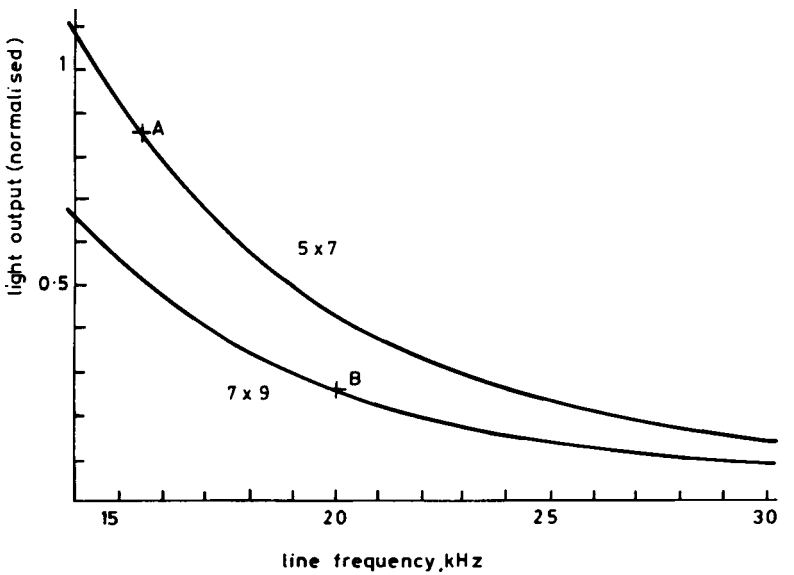


Fig. 9 See Appendix D for derivation of Figs. 8 and 9. Note the decrease in light output for two typical VDU parameters at points A and B as discussed in the text

Some manufacturers overcome the flicker problem by using a long persistence phosphor, but this leads to other problems such as low brightness, short screen life and smearing with changing picture content.

3 Physical CRT parameters

3.1 Problems

It is a sad fact that although the CRT is the best available display device it is not perfect. It suffers from the following problems:

(*a*) It is an emitter of light and is thus prone to being 'washed out' by high levels of ambient illumination. The brightness can be turned up but only so far.

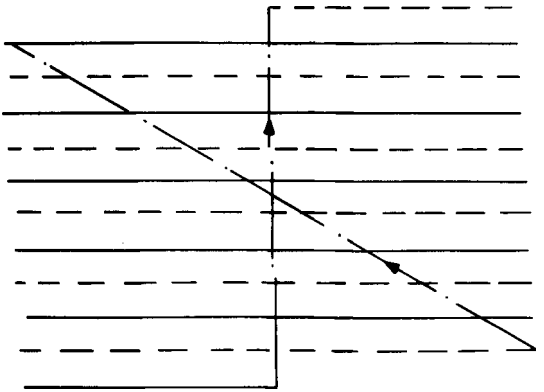


Fig. 10 Interlace

(*b*) Since a CRT works by projecting a beam of electrons from an electron gun, a considerable depth is needed behind the screen (hence the current interest in flat panel displays): obviously, as shown in Fig. 11, the bigger the deflection angle the shorter the tube. The largest deflection angle currently in use is 110° and this is common for VDUs. However, large deflection angles bring their own drawbacks in the form of focussing problems and geometry.

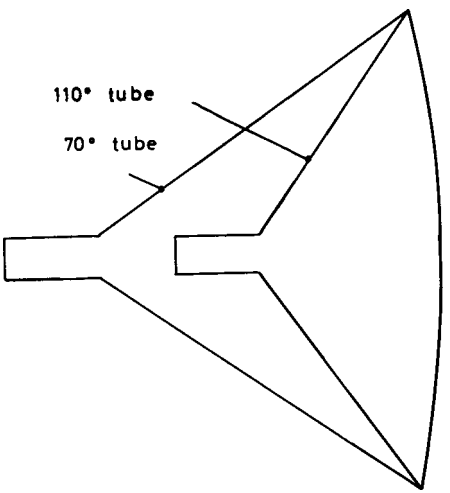


Fig. 11 Tube angles

(*c*) CRTs being full of vacuum need very strong glass envelopes and to achieve this the manufacturers make the screen spherical rather than flat, for the same reason that bridge builders use arches. This produces a very robust tube but brings a few more problems in its wake.

3.2 Distortion

The problem the VDU designer faces is exactly the opposite of that of the cart-

ographer – how do you represent a flat sheet of paper on the spherical surface of the CRT? He cannot, of course, so a compromise is needed. Terms like pincushion, barrel, keystone and trapezium are used (see Fig. 12) and all of them are present to some extent in every CRT display. In order to define a standard, we at ICL relate the display as seen to a rectangular display viewed from a distance of 600 mm normal to the centre of the screen. It is most important to realise that the picture can only be assessed when being viewed from the correct position.

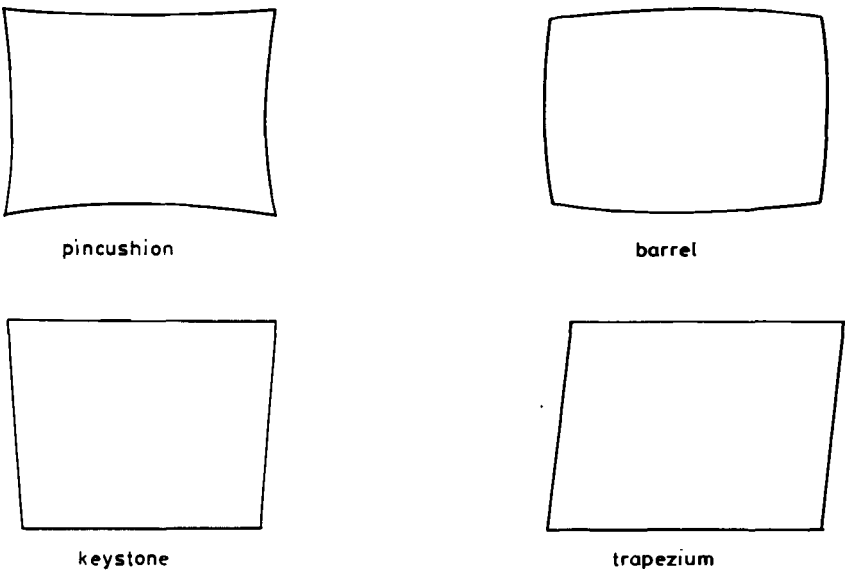


Fig. 12 Distortion

3.3 Stability

It is desirable that the display should appear completely static and flicker-free to the operator; and to achieve this the VDU designer has to steer a path through a minefield of problems.

Probably the most difficult problem is presented by the fact that a CRT will respond to magnetic fields. Permanent magnets are used to correct picture distortion for example, but alternating fields, especially at mains frequency, can cause problems.

Consider a VDU operating at 60 Hz refresh rate running on 50 Hz mains supply. Any stray field at 50 Hz (from a transformer for example) will cause a 10 Hz beat which shows up as an objectionable wobble. Such a field need only be about 1/10th of the earth's magnetic field. Three solutions are possible:

(*a*) screen the CRT with mumetal – pretty effective but very expensive. This method is used in precision displays,
(*b*) run the refresh at the local mains frequency (very commonly done in the

USA where 60 Hz mains are used). However, it is not a suitable solution for 50 Hz mains since 50 Hz will give rise to flicker problems. However many cheaper VDUs do use this method – for the time being!

(*c*) provide adequate shielding and orientation of the mains transformer.

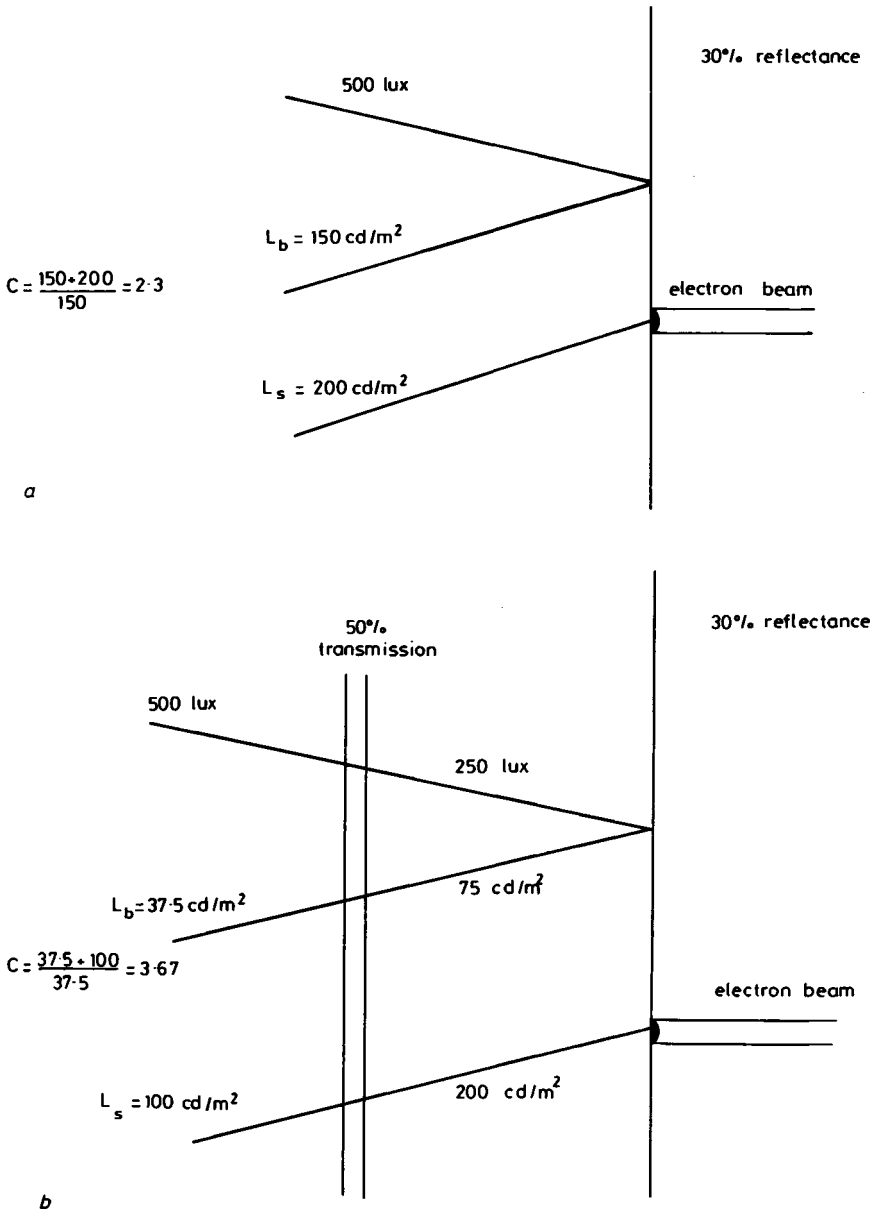


Fig. 13 (*a*) Bare CRT
(*b*) CRT with 50% filter

3.4 Contrast

The CRT creates its display by exciting a phosphor with its electron beam. The phosphor emits light which we see as the display. However most phosphors are whitish granular powders, when unexcited, and of course act as a reflector. As a result the difference between light emitted and light reflected can be quite small.

The apparent brightness of a surface is called its 'luminance' and is expressed in candelas per square metre (cd m^{-2}) (or by the older units 'foot-lamberts').

If L_s is the luminance of the spot
and L_b is the luminance of the background
then contrast ratio is defined as

$$C = \frac{L_s + L_b}{L_b} .$$

As far as legibility of CRT displays is concerned it can be demonstrated that high contrast is more beneficial than high brightness.

The minimum recommended value for contrast ratio is usually 3.

Most VDUs nowadays use a contrast enhancement screen in front of the CRT (see Fig. 13). Even greater improvement is possible with filters whose spectral transfer function matches the spectral response of the phosphor, but these tend to be expensive.

Polarising filters are not much used with CRTs since very little of the light reflected from the phosphor is in fact polarised even after passing through the filter. This is because of the granular nature of the phosphor. Polarising filters are expensive and have shiny surfaces.

3.5 Reflections

The previous paragraphs considered light reflected from the phosphor which being granular gives largely Lambertian reflection. (A uniformly illuminated Lambertian surface appears equally bright from whatever angle it is viewed, see Fig. 14.)

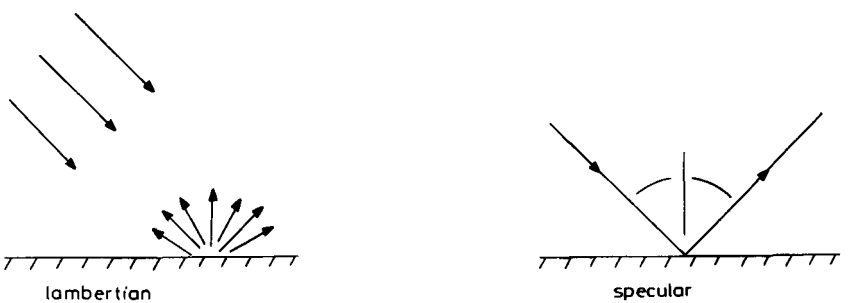


Fig. 14 Surface reflections

A shiny surface exhibits specular reflection as in a mirror, where the majority of the light reflected is along a path which makes an angle to the normal of the surface equal to that of the incident light.

All real surfaces of course exhibit a combination of the different types of reflection and in addition they also absorb and/or transmit light. Untreated glass reflects about 3–4% of the light incident upon it and absorbs or transmits the rest. Reflection from untreated glass is highly specular, which means that clear images of, for example, fluorescent lights or windows may be formed.

The shape of the surface has a very important effect on the nature of the reflected image. A flat surface will produce a 'natural' sized image as in a plane mirror whereas a convex surface will produce a smaller image but one which is visible over a wide angle. These effects are shown graphically in Fig. 15 and the derivation is in Appendix E. A typical VDU might have a spherical surface radius of about 0·65 m.

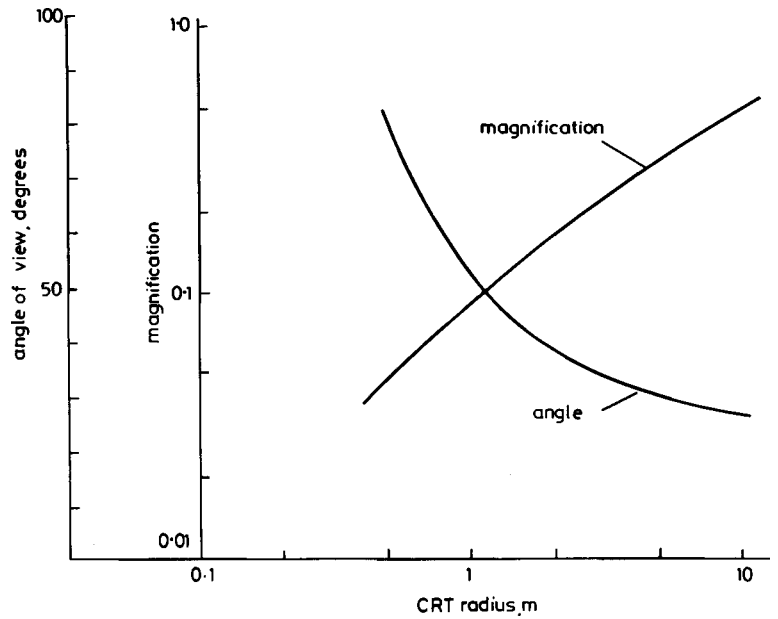


Fig. 15 Magnification and angle of view for 0·6 m viewing distance. See Appendix E for the derivation of these curves.

Subjectively the effect is that reflections, of say a fluourescent light, are smaller and hence less objectionable in a convex surface but being visible over a wide angle are very difficult to eliminate by moving the screen whereas the converse is true with a flat surface.

The choice of flat or convex is subject to much heated debate and each has its advocates. All that I can conclude is that there is no one best surface for all applications. The choice is also complicated by the availability of various antireflection treatments.

3.6 *Antireflection surfaces*

There are basically four ways of tackling reflections and all have their good and bad points:

(*a*) Roughened surface;
(*b*) Angled screen;
(*c*) Dichroic coating;
(*d*) Mesh filters.

(*a*) *Roughened surface:* By roughening the surface, reflections can be diffused and made less objectionable. This treatment can be applied to glass (by etching in acid) and a panel thus treated, bonded onto the face of the CRT, is known as a 'bonded faceplate'.

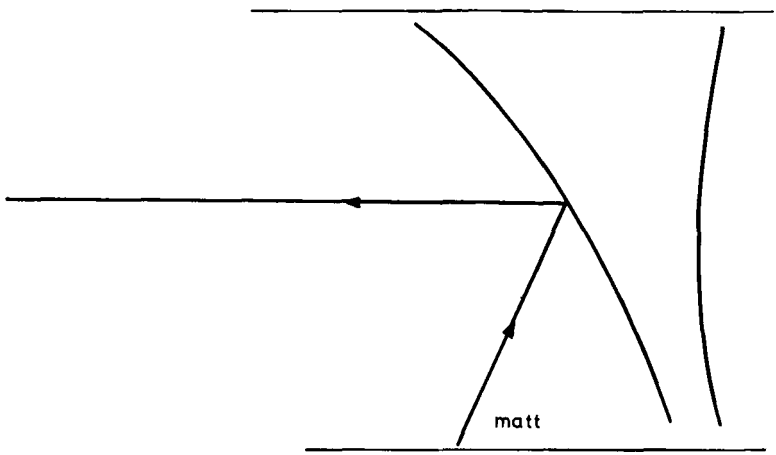


Fig. 16 Curved screen and hood

On plastic panels (flat or curved) a similar effect can be achieved with an applied coating.

This technique for tackling reflections is widely used but does have serious drawbacks.

(*i*) The roughened surface not only diffuses reflected light, it also diffuses transmitted light (e.g. the characters on the CRT screen). This causes a loss of focus and severely limits the resolution.
(*ii*) Bonded faceplates are expensive, mainly because of the difficult bonding operation.

(*b*) *Angled screen:* This is illustrated in Fig. 16. It will be seen that there is effectively a hood over the screen and any reflections reaching the viewer from the curved screen will be of the matt lining of the hood.

This technique has been used very effectively in viewing hoods for oscilloscopes, but would be very bulky when used with say a 15 in CRT. This bulk would I think be unacceptable on a VDU in a general office.

(*c*) *Dichroic coating:* Probably the most effective treatment of all, and certainly the most expensive. This is the type of coating sometimes known as 'blooming' on camera lenses. It can only be applied to glass and being applied under vacuum is a capital intensive operation. The technique is used for aircraft instrumentation but is not economic for large areas such as CRT faceplates at present.

(*d*) *Mesh filters:* Placing a fine mesh in front of the CRT can be effective in reducing reflections. A mesh itself reflects light poorly since the individual wires tend to scatter the incident light and since it casts a shadow on the CRT screen it works well in enhancing the contrast. But although it can be very effective it suffers from a number of drawbacks and I know of only one manufacturer who has used it with a VDU.

The problems are:

(*i*) it collects dust and allows dust to percolate through and to adhere to the CRT screen, attracted by the high voltage. It is difficult to clean;
(*ii*) being fragile it is liable to damage in use and when being cleaned;
(*iii*) it restricts the angle of view.

3.7 Choice of phosphor

The phosphor is a material which emits light when excited by the electron beam and there are a number of useful ones around for use in CRTs. The parameters of most interest are as follows.

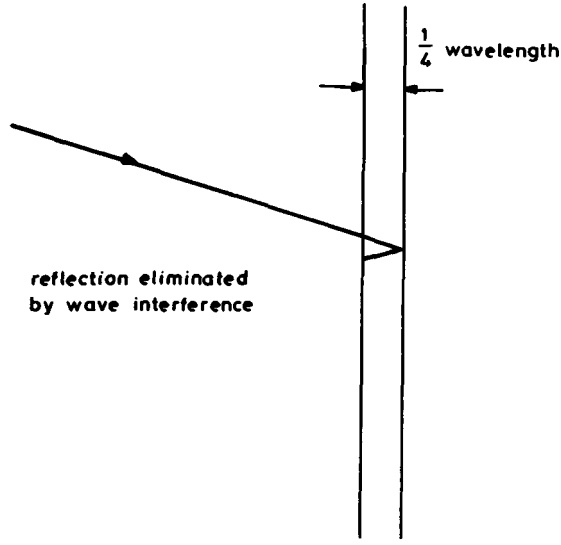


Fig. 17 Dichroic coating

(*a*) *Colour:* Most colours from red through yellow and green to blue and violet are available as are mixtures of colours, e.g. 'white'.

Visually the reds and blue–violet colours are poor. They are necessary of course for multicolour displays (e.g., colour TV) but are not suitable on their own. This leaves the greens, yellows and whites.

Although many VDUs use white and this is generally the best for viewing at distances over a metre or so, there is a growing body of opinion which considers that white is tiring when viewed close to as in many VDU applications.

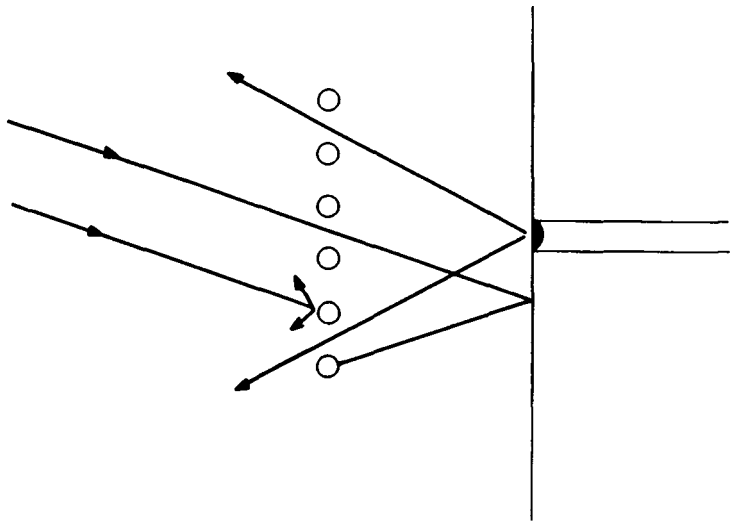


Fig. 18 Mesh screen

(*b*) *Efficiencey (visual):* The efficiency, expressed in lumens/watt, defines how much visible light is given out for a given excitation and since the VDU designer is seldom endowed with too much light this is a significant parameter. Fortunately the greens, yellow and whites are among the best.

(*c*) *Persistence:* The use of a long persistence phosphor is regarded by some as the obvious solution to the flicker problem. *This is a fallacy.* The light output from a phosphor does not cease instantly when the excitation is removed but decays, roughly exponentially.

A typical phosphor characteristic is shown in Fig. 19. It will be observed that the output drops quickly at the beginning and even a long persistence phosphor will produce a noticeable fluctuation if the refresh frequency is reduced much below 50 Hz. The eye is very sensitive to small changes in light output.

As a result the use of a long persistence phosphor has much less effect on flicker than might have been supposed.

In experiments conducted by ICL some years ago the use of a long persistence phosphor was found to be ineffective since the eye was sensitive to changes of as little as 6%.

A long persistence phosphor can be effective in minimising interlace flicker but introduces smearing and tails on a moving cursor and changing picture which can be annoying.

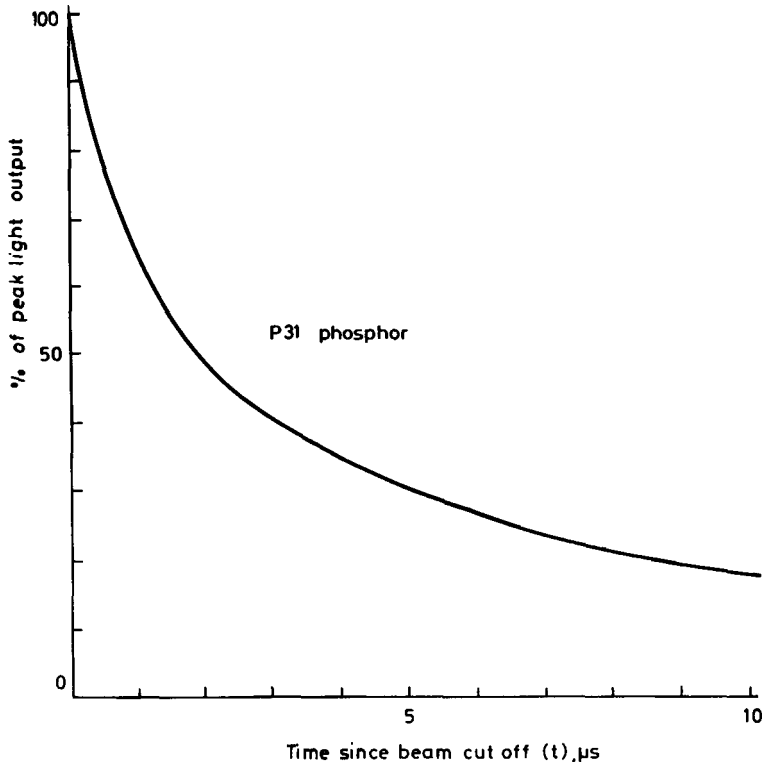


Fig. 19 Phosphor persistence

(*d*) *Resistance to burning:* Most phosphors eventually suffer a permanent loss of light output and this is hastened by operation at high level output. The long persistence phosphors are worst in this respect. It is not uncommon for a brand new tube to show permanent burn marks after only a few seconds use. The problem is made worse because long persistence phosphors are less efficient than the short/medium ones and there is a temptation to run them harder to maintain picture brightness.

4 Safety and health

It can be stated that VDUs present no known health or safety hazard. Easily stated, but how do we justify the statement? This subject has been extensively documented, so I will only summarise the situation here.

The possible safety hazards specifically appropriate to VDUs are as follows.

4.1 Implosion of the CRT

In principle it is possible for the CRT glass bowl to shatter either due to impact from some hard object or due to its own internal stress. The pressure difference due to the internal vacuum could then cause glass particles to be thrown about.

All modern TV sets and VDUs incorporate a protection device known as the tension band (or T-Band) designed to prevent glass flying. Such VDUs have a very good safety record.

Despite the very high level of safety built into all CRTs, many VDUs incorporate additional protection in the form of either a bonded faceplate or plastic screen in front of the tube. This also provides contrast enhancement and anti-reflection treatment.

4.2 Ionising radiation

CRTs are high voltage devices and can generate X-rays. However, the glass used in the CRT construction contains lead which absorbs these rays and in any case the voltage level is very low compared with that required to produce energetic X-rays such as used for X-ray photographs.

In practice the radiation level is well below the accepted safety level. In most cases it is comparable to the background cosmic radiation and all the health authorities agree there is no health hazard from the CRT.

4.3 High voltages

Whilst it is true that VDUs use high voltages (up to 20,000 V), and these figures are often used to impress, it is usually the case that the power supplies generating these voltages produce very little current and in consequence these high voltages are often less dangerous than the normal 240-V mains supply.

All VDUs supplied by ICL have to meet stringent international safety standards as does all data processing equipment.

4.4 Health hazards

These alleged hazards are currently attracting a lot of attention but medical opinion is that there is no known reason to think that VDUs constitute a hazard. Currently speculation is based on two VDU operators in the USA who contracted cataracts in the eye, but investigations have revealed no connection between these afflictions and the operation of VDUs.

Sources of possible hazard to health currently of interest include:

(*a*) visual effects e.g. flicker, contrast, colour, stability, focus etc., which are dealt with elsewhere in this paper

(*b*) heat; every VDU dissipates a certain amount of heat and some concern has been expressed that this could cause dryness of the eyes. This view is refuted by medical opinion and in any case there are many more potent sources of heat: a single bar electric fire (1 kW) releases at least five times more heat than a typical VDU.

(*c*) Non-ionising radiation, e.g. infrared, ultraviolet, radio waves etc.

Certain regions of the electromagnetic spectrum are well researched and understood whilst others are not. The radio spectrum between say 3×10^{4} m and 3×10^{-1} m is well understood and controlled and there are far more potent emitters than VDUs around. Between 3×10^{-1} m and 3×10^{-3} m it gets progressively more difficult to make detectors. Between 3×10^{-3} m and 3×10^{-6} m it is currently almost impossible to make detectors (other than thermopiles which are not very sensitive).

So we have little knowledge of radiation in these wavelengths. Wavelengths below 3×10^{-6} bring us into the region of infrared and optical wavelengths which again are well understood.

Current speculation that VDUs can cause harmful radiation in these unmeasurable wavelengths can thus only be regarded as mischievous unless simultaneously directed with equal force at Hifi, central heating, TV and motor cars.

4.5 Psychological

The very real fears that many people have about their future employment are unlikely to be alleviated by the introduction to their offices of new technologies, of which the VDU is probably the most visible and may consequently become the focus of their discontent.

I think that the best the VDU designer can do is to ensure his VDU is built to acceptable safety and health standards and produces as good a display as possible. He should also be sympathetic and patient in explaining the merits and short comings of his product.

5 Conclusion

A paper like this cannot conclude with a few neat recommendations. The subject is too broad and there are so many solutions that no single set of parameters can be regarded as the final answer.

Time and attitudes change. Had CRT operators in the late 1930s insisted on refresh rates above 50 Hz we would have had no radar. Nowadays people expect better conditions and demand higher standards and rightly so in my opinion.

The problems attributed to VDUs will, I think, diminish as generations of operators accept VDUs as part of their environment rather than 'new technology', provided VDU manufacturers maintain high standards. Appendix F concludes this paper with a few possible solutions to the problem of selecting character and line formats.

The last word will however rest with the buyer of VDUs, who will undoubtedly compare the display quality of my offering with the best available and its cost with the cheapest.

References

1 CAKIER, A., HART, D.J. and STEWART, T.F.M.: 'Visual display terminal manual', IFRA, Darmstadt, 1979.

2 Guides to users of business equipment, including visual display units. Business Equipment Trade Association (BETA), London, 1979.

Appendix A

Analysis of various type fonts

Font Name	Character width (W)	UC height (H)	Stroke (t) thickness	Descender (d)	LC height (h)	$\frac{t}{H}$	$\frac{H}{W}$	$\frac{d}{h}$	$\frac{H}{h}$
Alt. Gothic No. 3	2·0	4·8	0·8	1·2	3·5	0·17	2·4	0·34	1·4
Eras Medium	4·5	5·7	0·8	2·0	4·2	0·14	1·3	0·48	1·4
Erostile Extended	6·4	5·5	0·7	1·8	3·8	0·13	0·9	0·47	1·4
Filio Light	3·5	4·2	0·5	1·2	3·3	0·10	1·5	0·36	1·6
Futura Medium	3·3	5·0	0·7	1·3	3·0	0·14	1·5	0·43	1·7
Helvetica	4·0	5·5	0·8	1·4	3·7	0·15	1·4	0·38	1·5
Unvers 57	2·4	4·9	0·7	1·4	3·2	0·14	2·0	0·44	1·5
Adler Typewriter	1·2	2·3	0·2	0·7	1·6	0·09	1·9	0·44	1·4
Facit Golf Ball	1·3	2·4	1·3	0·8	1·6	0·13	1·8	0·50	1·5
Average						0·13	1·63	0·43	1·49
ICL VDU Type 7561/1 (in terms of dots) (note 2)						0·17	1·5	0·5	1·5

Notes: (*a*) dimensions are measured between the centres of the strokes. See Fig. 1.
(*b*) because height and width are adjustable independently in a VDU the dimensions of the dots may not be the same in the vertical and horizontal directions.

Appendix B

Analysis of row spacing

Source of data	*Row space* LC height (*h*)	*Row space* UC height (*h*)
Typewriter	2·7	1·9
Texas instruments calculator manual	2·1	1·8
Corgi paperback	3·0	1·9
Radio society of Great Britain manual	2·7	1·9
Signetics catalogue	2·7	2·0
Design engineering magazine	2·6	1·8
Average	2·6	1·9
ICL VDU type 7561/1 (in terms of dots)	2·8	1·8

See also notes a and b in Appendix A.

Appendix C

Effect of number of displayed lines on line and field frequencies (derivation of Fig. 7)

It is assumed that field flyback time is constant at 960 μs

Let f_L = line repetition frequency ($t_L = \frac{1}{f_L}$ = line period),

f_F = field repetition frequency ($t_F = \frac{1}{f_F}$ = field period),

N = number of lines available to display data excluding those during field flyback

n = number of lines during field flyback.

Hence $f_F = \frac{f_L}{(N+n)}$ where $n = \frac{960}{t_L}$

f_L (kHz)	15	17·5	20	22·5	25
N					
150	91·2	104·9	118·2	131·1	143·7
175	79·2	92·2	103·0	114·95	125·6
200	70·0	80·7	91·2	101·5	111·6
300	47·7	55·2	62·7	70·0	77·2
350	41·2	47·7	54·2	60·6	66·8
400	36·2	42·0	47·7	53·4	59·0

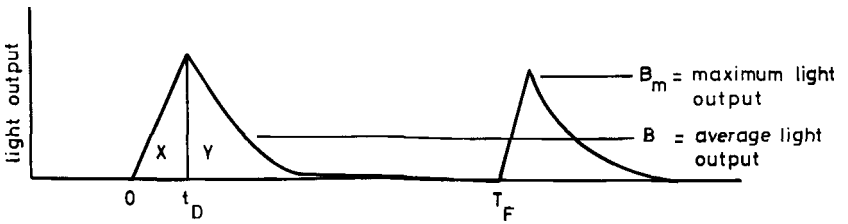
Appendix D

Effect of line frequency (f_L) on video bandwidth and line output (shown graphically in Figs. 8 and 9)

Based upon the following assumptions which are typical of VDU applications

80 characters per row.
7 or 9 dots per character (5 x 7 or 7 x 9 matrix including 2 dots intercharacter gap)
Line flyback blanking time 13 μs
Field frequency 50 Hz
Phosphor time constant $T_P = 10\ \mu s$.

$$\text{Dot period } t_D = \frac{f_L - 13}{80x7} \text{ or } \frac{f_L - 13}{80x9} \quad \mu s$$



If we assume light output rises linearly with t which is reasonable if $t_D \ll T_P$

then area $X = \frac{B_m t_D}{2} = \frac{kt_D^2}{2}$.

If we assume light output decays as $B = B_m e^{-t/T_P}$

then area $Y = \int_{t_D}^{T_F} B_m e^{-t/T_P} dt$ and if $T_P \ll T_F$ area $Y \to T_P$.

Total area under curve $= X + Y = \frac{Kt_D^2}{2} + T_P$

B_0 = Average light output $= \frac{1}{T_F} \frac{kt_D^2}{2} + T_P$

If we normalise this so that

$B_0 = 1$ when $t_D = 100$ ns $T_F = 20$ ms $T_P = 10\ \mu s$

this reduces to $B_0 = 10^{14} t_D^2$ ignoring high order terms or more conveniently

$B_0 = \frac{t_0^2}{10^4}$ where t_D is in ns.

Whence we can calculate the following table of light output.

Line frequency,	7 dots/char		9 dots/char	
f_L	t_D	B_0	t_D	B_0
14	104	1·09	81	0·66
16	88	0·78	69	0·47
18	76	0·58	59	0·35
20	66	0·44	51	0·26
22	58	0·34	45	0·20
24	51	0·26	40	0·16
26	45	0·21	35	0·13
28	51	0·16	32	0·10
30	36	0·13	28	0·08

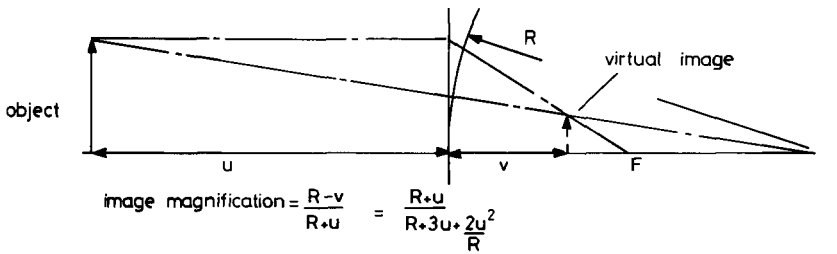
Appendix E

Reflection from CRT face

The nuisance value of a reflection from the CRT face depends upon the size of the image as well as its brightness. The magnifications calculated below compare the size of the image with the size it would have if viewed direct.

Obviously a reflection is only objectionable if it falls within an operator's field of view and to minimise this the angular field of view calculated below should be minimised. A typical CRT has a radius of 0·65 m and it will be seen that a flat screen would be better provided it could be positioned to suit the operator. There is no consensus of opinion on whether a flat or a curved screen is better.

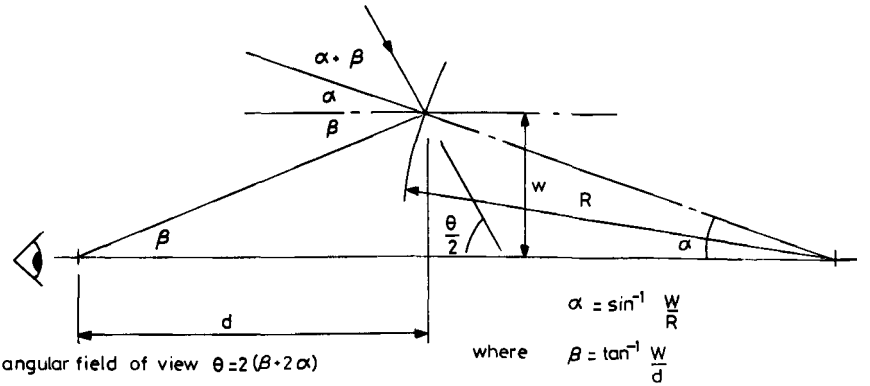
(*a*) *Magnification*



$$\text{image magnification} = \frac{R-v}{R+u} = \frac{R+u}{R+3u+\frac{2u^2}{R}}$$

Hence assume a light say at $u = 5$ m from the screen and calculate magnification for different values of screen radius R.

$R(m)$	magnification
0·5	0·048
0·6	0·057
0·7	0·065
0·8	0·074
1·0	0·091
2	0·167
3	0·231
4	0·286
5	0·333
10	0·5



(*b*) *Angle of view*

Hence for a typical 15 in CRT viewed from 0·6 m
$d = 0{\cdot}6$ m
$w = 0{\cdot}125$ m

R	0°
0·5	81·5
0·6	71·6
0·7	64·7
0·8	59·5
0·9	55·5
1	52·3
2	37·9
5	29·3
10	26·4
Infinity (flat)	23·54

Appendix F

Selecting character and line formats: possible solutions

	1	2	3	4	5
Matrix	5 x 7	5 x 7	7 x 9	5 x 7	7 x 9
Descenders	2	2	2	2	2
Inter row	4	1	1	4	3
Lines/row	13	10	12	13	14
Scan	raster	step	step	raster	raster
No of rows	24	24	24	22	24
No of lines in field flyback	19	15	19	25	14
Line freq. kHz	20	15·625	18·4	15·625	21
Field freq. Hz	60·4	61·3	60	50	60
Dot time, ns	59	84	53	86	49
Graphics	possible	no	no	possible	possible

All the above combinations give possible acceptable pictures. They are by no means exhaustive and all assume 80 characters per row and standard 12 μs line flyback.

Comments:

1 Should give reasonable picture maybe a bit dotty but could be used with continuous graphics
2 Should give good picture but not compatible with graphics.

3 Should give very good picture but not compatible with graphics.
4 Reasonable picture uses 'standard' UK line and field frequencies compatible with graphics and TV but probably not acceptable due to low field frequency causing flicker.
5 Should give good picture compatible with graphics but pushing the technology needing fast scan circuits and character generator. Character quality would not be as good as 3.

Solution of elliptic partial differential equations on the ICL Distributed Array Processor

S.J.Webb
ICL DAP Marketing Unit, London

Abstract

The ICL Distributed Array Processor (DAP) is a radical departure from conventional serial computer architecture which is capable of performing operations simultaneously on many different pieces of data. Conventional methods for solving partial differential equations are tailored for serial computers and do not exploit the parallelism of the DAP. The paper discusses some of the most important of these methods and shows how, by considering the problems with parallel processing in mind, powerful and efficient parallel algorithms can be devised.

1 Introduction

Partial differential equations can be used to describe a very great number of scientific and commercially relevant problems, from the evolution of the weather to the flow of oil in a subterranean reservoir. A discussion of these applications, ranging from purely scientific through engineering and defence to commerce, is beyond the scope of this paper but in general it can be said that the methods of solution have a great deal in common; and, in particular, they all demand vast amounts of computer time.

The Distributed Array Processor (DAP)[1-3] is an array, in the versions now being built, of 64 x 64 identical processing elements (i.e., 4096 in total) each with its own store of 4096 bits. The DAP can perform the same operation simultaneously on many pieces of data and is thus a SIMD – single instruction, multiple data – computer. As we shall see, the methods used to solve partial differential equations on a computer require identical operations to be performed on large numbers of different pieces of data, which is precisely the task to which the DAP is ideally suited.

As is well known, all practical methods for the numerical integration of partial differential equations involve discretisation, that is, replacement of the differential coefficients by finite differences. This reduces the problem to that of solving a (usually large) set of linear algebraic equations; non-linear systems can almost always be made to depend on a linearised system. This algebraic system can then be

solved either by a direct method or by an iterative method. The former will give the solution after a finite, pre-determined number of computational steps; the latter approaches the solution by successive approximations. Direct methods are described in Section 2; these are applicable to one-dimensional problems and also have wider applicability for inverting matrices generally. Iterative methods, which are almost always used for two or three dimensional problems, are described in Section 3.

The paper deals only with *elliptic* equations. These are of very great importance in physical and engineering applications as they describe all steady-state and diffusion-type phenomena. The equations of Laplace and Poisson are the standard forms.

2 Direct methods

2.1 General methods

We start by considering methods which apply to any set of linear algebraic equations, not necessarily arising from the discretisation of a partial differential equation.

The standard method is Gaussian elimination;[4] it will produce an exact solution after a finite, predetermined number of steps. Consider for example the solution of the following set of four equations in four unknowns:

$$\begin{aligned} x + 3y + 7z + w &= 32 \\ x + 4y + 9z + 2w &= 44 \\ 2x + 8y + 19z + 3w &= 87 \\ 3x + 13y + 33z + 4w &= 144 \end{aligned} \qquad (1)$$

The method falls into two distinct parts, a forward elimination followed by a back substitution. In the first, the x terms are eliminated from the second, third and fourth equations by subtracting multiples of the first; the new second equation is then used to eliminate the y and z terms from the new third and fourth, and similarly for the z terms. This results in the equations taking the following triangular form:

$$\begin{aligned} x + 3y + 7z + w &= 32 \\ y + 2z + w &= 12 \\ z - w &= -1 \\ w &= 4 \end{aligned}$$

The second part, the back substitution, gives explicitly the values of each of the unknowns. We do this by substituting the value of w into the third equation to give z, and so on for y and x.

On a serial computer this is the most efficient method for solving a set of linear equations and takes of the order of n^3 operations to solve n simultaneous equations. On a parallel computer such as the DAP the method is not, however, the most efficient.

Consider for example solving the above set (1) on a DAP. We may perform the first

elimination entirely in parallel because we can subtract the first equation from the second, third and fourth at the same time; but when we come to eliminate the y terms we only subtract the second equation from the third and fourth. Thus the degree of parallelism decreases as the forward elimination proceeds until in the last elimination we only have to subtract the penultimate equation from the last. Moreover we must then perform the back substitution which for the same reason as for the forward elimination is not at all suitable for a parallel computer.

Both these problems are overcome if we use a variation of Gaussian elimination known as the Gauss-Jordan method.[4] On a serial computer this is not as efficient as Gaussian elimination, taking approximately twice the number of operations to produce the solution. But on a parallel computer with $n \times n$ processors we can solve a set of n linear equations in a number of steps proportional to n by this method. The difference between the Gauss-Jordan method and Gaussian elimination appears when the degree of parallelism begins to decrease in the forward elimination stage. The first step is the same in both methods and gives:

$$\begin{aligned} x + 3y + 7z + w &= 32 \\ y + 2z + w &= 12 \\ 2y + 5z + w &= 23 \\ 4y + 12z + w &= 48 \end{aligned}$$

At the second step we eliminate not only below the second equation but above also, producing

$$\begin{aligned} x + z - 2w &= -4 \\ y + 2z + w &= 12 \\ z - w &= -1 \\ 4z - 3w &= 0 \end{aligned}$$

We do the same when eliminating the z and w terms, getting finally

$$\begin{aligned} x &= 1 \\ y &= 2 \\ z &= 3 \\ w &= 4 \end{aligned}$$

Notice that by seeking to maximise the parallelism at each step we have also removed the need to perform the back substitution.

Any set of equations

$$A\,T = Q \qquad (2)$$

where the matrix A is non-singular may be solved on the DAP by the Gauss-Jordan method. For the number of equations $N \leqslant 64$ the solution is obtained exactly as described above. The DAP Fortran for the simplified method without pivoting—which means that the process will be numerically unstable for some ill-conditioned matrices A — is given in Fig. 1. The variables *LC, LR* are logical matrices, **LV** is a logical vector, A is a real matrix and **B** a real vector. N is the number of equations.

```
      DO 10      I = 1, N
      LR = ROW(I)                    set row i in LR .TRUE.
      LC = COL(I)                    set column i in LC .TRUE.
      LV =  EL(I)                    set element i in LV .TRUE.

      S  =  1/A(I,I)
      A(LR) = A*S                    divide pivotal row of A and rhs Q by pivotal element
      Q(LV) = Q*S

      A(.NOT.LR) = A - MATC(A(LC))*MATR(A(LR))     select pivotal column and expand
10    Q(.NOT.LV) = Q - A(LC)*B(LV)                 columnwise, multiply by pivotal
                                                   row spread row-wise and subtract
                                                   from A. Do likewise for Q which
                                                   at the end contains the solution
```

Fig. 1 DAP Fortran program for Gauss-Jordan method

The functions ROW, COL, EL set the *i*th row, column or element of a logical matrix or (for EL) vector to .TRUE. The functions MATC and MATR create matrices of columns and rows, respectively, from a vector. The paper by Gostick[3] gives more details of DAP Fortran.

Pivoting, either partial or full, may be performed by choosing *LR* and *LC* by some other criterion such as the position of the largest element in a row. The operations are then identical to those shown above, although there are a few extra masking operations to mark and order the elements selected as pivots.

	P.E. (1, 2)			P.E. (1,64)
P.E. (1,1)	$a_{1,1}$ $a_{1,2}$ $a_{2,1}$ $a_{2,2}$	$a_{1,3}$ $a_{1,4}$ $a_{2,3}$ $a_{2,4}$	...	$a_{1,127}$ $a_{1,128}$ $a_{2,127}$ $a_{2,128}$
P.E. (2,1)	$a_{3,1}$ $a_{3,2}$ $a_{4,1}$ $a_{4,2}$			
	. . . .			
P.E. (64,1)	$a_{127,1}$ $a_{128,1}$			

Fig. 2*a* Crinkled storage mode for 128 x 128 matrix

If the number of equations $N > 64$, we can use standard matrix partitioning or a technique known as 'crinkling'. Partitioning splits the problem into portions which will fit on to the DAP, whilst crinkling maps the larger problem on to the DAP by 'folding' until it fits. For example, a 128 x 128 matrix may be mapped on to a 64 x 64 DAP by storing more than one element per processing element, as in Fig. 2*a*. Partitioning is illustrated in Fig. 2*b*.

	P.E.(1,2)		P.E. (1,64)	
P.E. (1,1)	$a_{1,1}$ $a_{1,65}$ $a_{65,1}$ $a_{65,65}$	$a_{1,2}$ $a_{1,66}$ $a_{65,2}$ $a_{65.66}$	...	$a_{1,64}$ $a_{1,128}$ $a_{65,64}$ $a_{65,128}$
P.E. (2,1)	$a_{2,1}$ $a_{2,65}$ $a_{66,1}$ $a_{66,65}$			
P.E. (64,1)	$a_{64,1}$ $a_{64,65}$ $a_{128,1}$ $a_{128,65}$			

Fig. 2*b* Partitioned storage mode for 128 x 128 matrix
The original matrix is regarded as partitioned into four 64 x 64 matrices and the elements stored in the P.E.s as shown

2.2 *Tridiagonal system*

This is an important form which arises from the discretisation of Poisson's Equation in one dimension and which plays an important role in some iterative methods to be discussed later. It is characterised by being very sparse – i.e. by having only a few nonzero elements in each row or column – and with the nonzero elements forming a regular pattern, only the leading diagonal and one element on either side of this in each row being nonzero. This is illustrated in Fig. 3.

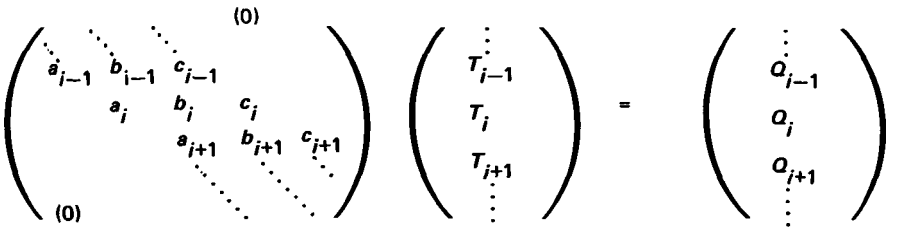


Fig. 3 Tridiagonal system

For this set of equations the Gaussian algorithm takes a particularly simple form.[5] The first step is to eliminate all nonzero elements to the left of the leading diagonal and the second is to back-substitute to obtain the solution; no new nonzero elements are generated outside the diagonal so the matrix retains its sparsity pattern and no extra storage is required.

On a serial computer the method gives the solution for a set of n equations in $2n$ steps; on the DAP we can solve such a set in $\log_2 n$ parallel steps, as follows. Consider the ith line

$$a_i T_{i-1} + b_i T_i + c_i T_{i+1} = Q_i \qquad (3)$$

We can rewrite this with T_i on the left side:

$$T_i = (Q_i/b_i) - (a_i/b_i)\, T_{i-1} - (c_i/b_i)\, T_{i+1} \tag{4}$$

Substituting this and the corresponding equations for T_{i-1} and T_{i+1} into eqn. 3 and writing $Q_i' = Q_i/b_i$ etc. we get

$$-a_i' a_{i-1}'\, T_{i-2} - (-1 + a_i' c_{i-1}' + c_i' a_{i+1}')\, T_i - c_i' c_{i+1}'\, T_{i+2}$$
$$= Q_i' - a_i' Q_{i-1}' - c_i' Q_{i+1}' \tag{5}$$

and have eliminated T_{i-1} and T_{i+1}. We now repeat this, substituting for T_{i-2} and T_{i+2} in terms of T_i, T_{i-4} and T_{i+4}, and so on. After $\log_2 n$ substitutions we have an explicit expression for each T_i.

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 \\ a_2 & b_2 & c_2 & 0 \\ 0 & a_3 & b_3 & c_3 \\ 0 & 0 & a_4 & b_4 \end{pmatrix} \longrightarrow \begin{pmatrix} b_1' & 0 & c_1' & 0 \\ 0 & b_2' & 0 & c_2' \\ a_3' & 0 & b_3' & 0 \\ 0 & a_4' & 0 & b_4' \end{pmatrix} \longrightarrow \begin{pmatrix} b_1'' & 0 & 0 & 0 \\ 0 & b_2'' & 0 & 0 \\ 0 & 0 & b_3'' & 0 \\ 0 & 0 & 0 & b_4'' \end{pmatrix}$$

Fig. 4 Cyclic reduction method for 4 x 4 tridiagonal system

For four equations the process is shown diagrammatically in Fig. 4. The two diagonals adjacent to the leading diagonal 'skip out' progressively towards the outside of the matrix, leaving eventually only the leading diagonal.. The DAP Fortran program for this cyclic reduction process* is easily derived from eqn. 5 and for a set of 64 equations is given in Fig. 5.

```
      REAL A( ), B( ), C( ), T( ), Q( )        Declare non-zero terms of matrix A
                                               as vectors A,B,C. Also declare solu-
                                               tion T and rhs Q
      K = 1
      DO 10 I = 1,6
      A = A/B                    normalise to the diagonal element
      C = C/B
      Q = Q/B

      B = 1 - A*SHRP(C,K) - C*SHLP(A,K)        form the new terms A, B, C, Q as in
      Q = Q - A*SHRP(Q,K) - C*SHLP(Q,K)        eqn. 5.
      A = - A*SHRP(A,K)                        A*SHRP(C,K) = a_i c_{i-k}, since the
      C = - C*SHLP(C,K)                        function SHRP shifts the vector C k
                                               places right.
10    K = K + K
      T = Q/B                    complete the solution
```

Fig. 5 DAP Fortran program for solution of tridiagonal set by cyclic reduction

*I am indebted to Professor Dennis Parkinson, ICL and Queen Mary College, University of London, for this information.

The program works equally well if A,B,C,T,Q are long vectors and we run the DO loop from 1 to 12; thus we can solve a set of 4096 equations.

2.3 Limitations of direct methods

These methods are less useful for problems in more than one dimension. This is because the sparsity pattern of the original matrix is not preserved in either the elimination or the back-substitution process, as it is in the one-dimensional case. Areas which consisted entirely of zeros become filled with nonzero elements and in the two-dimensional case, for example, the storage requirements are proportional to n^2 as opposed to n for the one-dimensional case; and on a serial machine the execution time is proportional to n^3 compared to n again. So, as stated in the Introduction, iterative methods are used for problems in two or more dimensions.

3 Iterative methods

3.1 Point Jacobi and Gauss-Seidel methods

Consider the problem of solving Poisson's Equation in two dimensions:

$$\partial^2 T/\partial x^2 + \partial^2 T/\partial y^2 = Q$$

with the values of T given on a closed boundary in the $x - y$ plane. This could describe, for example, the steady-state temperature distribution in some conducting material, when the function $Q(x,y)$ on the right side would describe the heat source or sink. Making the standard replacement of derivatives by finite differences and for convenience scaling the independent variables x,y so that the spacings Δx, Δy are both unity, we have

$$T_{i,j-1} + T_{i-1,j} - 4T_{i,j} + T_{i,j+1} + T_{i+1,j} = Q_{i,j} \tag{7}$$

If we assume a 64 x 64 grid this gives a set of 4096 equations for the $T_{i,j}$. The simplest iterative method is the *Point Jacobi* which from a given approximate solution $T_{i,j}^{(p)}$ forms a new approximation $T_{i,j}^{(p+1)}$ by replacing each value by the mean of its four nearest neighbours:

$$T_{i,j}^{(p+1)} = 0{\cdot}25\ [\ T_{i,j-1}^{(p)} + T_{i-1,j}^{(p)} + T_{i,j+1}^{(p)} + T_{i+1,j}^{(p)} - Q_{i,j}] \tag{8}$$

If one starts with an initial (guessed) solution $T^{(0)}$ and carries out the process successively this is analogous to allowing the initial solution to 'diffuse' to the correct solution.

Eqn. 8 may be performed at each point simultaneously and the DAP Fortran follows easily:

$$T = 0{\cdot}25 * (T(,-) + T(-,) + T(,+) + T(+,) - Q)$$

The notation is explained in the paper by Gostick.[3]

The mean of the four nearest neighbours to each point can be found with only two additions rather than three, by writing the process:

$T = T(,-) + T(-,)$
$T = 0{\cdot}25^{*}\ (T + T(+,+) - Q)$

However, whilst this method is the simplest it is also the slowest to converge. Convergence can be improved by using the slightly more complex *Gauss-Seidel* scheme, which is a natural extension of Point Jacobi. In the latter we have to store all the values at both the old and the new iteration levels; if instead we overwrite the old value with the new immediately that has been calculated we halve the storage requirements and also improve the convergence, because at every step we are using more recent values. This algorithm is

$$T_{i,j}^{(p+1)} = 0{\cdot}25\ [\ T_{i,j-1}^{(p+1)} + T_{i-1,j}^{(p+1)} + T_{i,j+1}^{(p)} + T_{i+1,j}^{(p)} - Q_{i,j}]$$

We cannot immediately construct DAP Fortran to compute all the values at the new iteration level simultaneously because at each point the new value depends upon two old and two new values. Consider however scanning through the mesh as in Fig. 6, calculating each new value by the Gauss-Seidel algorithm.

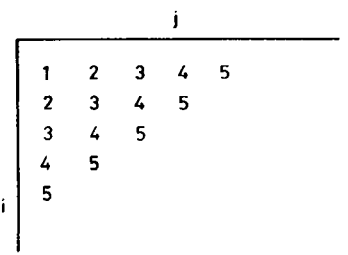


Fig. 6 Step sequence for Gauss-Seidel process on DAP

The first step is to up-date the corner point (Step 1). This then allows us to up-date the next points in the i and j directions, which can be done simultaneously in Step 2. Then (Step 3) the next three points can be calculated simultaneously and at the same time the first point can be up-dated for the second time because this does not directly affect the points 3. Similarly at Step 4 the points 4 are up-dated and simultaneously points 2 for the second time, and so on. Thus as the first iteration sweeps across the mesh the second follows behind it and behind that the subsequent iterations.

This process is made possible because the difference scheme only couples a point to its nearest neighbours. Thus at some stage in the iteration we are up-dating one set of points, the black cells in Fig. 7, at say, odd numbered iterations and the white cells at even iterations. We therefore redefine the iterative scheme slightly to define one iteration level at the black points and another at the white, rather than have the mesh points at a number of different iteration levels. The two levels are defined first by calculating new values at the black points using the old values at the white and then using these new values at the black points to form new values at the white. In

this scheme all the black points are at the same iteration level and all the white at an adjacent level. Note however that on our 64 x 64 mesh it will take 32 steps for information about the boundary to reach the centre and therefore at least this number for the process to converge.

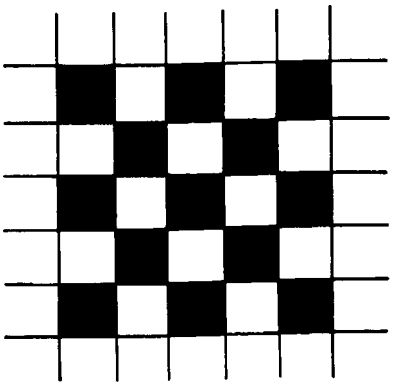


Fig. 7 Two-level scheme for Gauss-Seidel process on DAP

The DAP Fortran for this algorithm is now easy to define:

```
    BLACK = .NOT.WHITE
T (BLACK) = 0·25 * (T(-,) + T(,-) + T(+,) + T(,+) - Q)
T (WHITE) = 0·25 * (T(-,) + T(,-) + T(+,) + T(,+) - Q)
```

As before a faster version is possible using only two additions in each line. All that is needed is to define the WHITE mask:

```
    WHITE = ALTC(1).LEQ.ALTR(1)
```

which gives

```
F T F . .
T F T . .
F T F . .
. . . . .
```

ALTC and ALTR are DAP Fortran functions which set alternate rows and columns, respectively, TRUE.

3.2 Successive overrelaxation

This is a variation of the Gauss-Seidel method which was first found empirically to improve the rate of convergence and was later proved analytically to do so.[6] The essence of the method is to find the amount by which a Gauss-Seidel iteration would alter the value at a point and then to make a greater change; the aim is to try to push the solution closer to the correct solution than the Gauss-Seidel iteration would take it. The scheme is now

$$T^{(p+1)} = T^{(p)} + \omega \Delta T \tag{10}$$

where ΔT is the amount by which the Gauss-Seidel process would change the value and ω is the overrelaxation parameter.

Since $\Delta T = T_{GS} - T^{(p)}$ this is equivalent to

$$T^{(p+1)} = (1 - \omega)\, T^{(p)} + \omega T_{GS} \tag{11}$$

It has been shown[6] that this process will converge if ω is less than 2. However, the number of iterations required for convergence does not vary smoothly with ω but has a sharp minimum at an optimum value, in the neighbourhood of which it varies very quickly: Fig. 8 shows this behaviour qualitatively

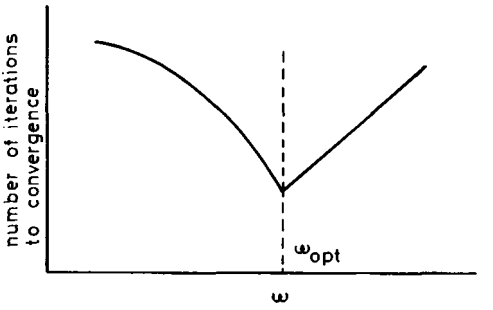


Fig. 8 Variation of convergence rate with overrelaxation parameter

This is the only major drawback of what is a powerful iterative method. The Figure shows that it is better to overestimate ω than to underestimate. The usual method of finding ω for any particular case is by trial and error; exact methods are available but these can be as costly in computation as the solution process itself.

The DAP Fortran follows easily from the Gauss-Seidel coding.

```
    BLACK = .NOT. WHITE
T (BLACK) = (1 - ω)*T + 0·25*ω *(T(-,) + T(,-) + T(+,) + T(,+) - Q)
T (WHITE) = (1 - ω)*T + 0·25*ω *(T(-,) + T(,-) + T( +,) + T(,+) - Q)
```

3.3 Combined direct and iterative methods

Powerful methods can be constructed by suitably combining the direct and iterative methods already described. If we consider a two-dimensional problem as a series of weakly-coupled one-dimensional problems we can use the tridiagonal method of para. 2.2 to solve the latter and then introduce the coupling iteratively. For example, let us consider the columns (j) of the two-dimensional mesh as one-dimensional meshes coupled by the rows (i). For a column j the one-dimensional problem is formulated

$$T_{i-1}^{(p+1)} - 2T_i^{(p+1)} + T_{i+1}^{(p+1)} = Q_i$$

which we can solve using the cyclic reduction method. If we now put in explicitly the coupling of these one-dimensional meshes the complete problem becomes, for the point (i,j)

$$T_{i-1,j}^{(p+1)} - 4T_{i+1,j}^{(p+1)} + T_{i,j-1}^{(p+1)} = Q_{i,j} - [T_{i,j-1}^{(p)} + T_{i,j+1}^{(p)}] \qquad (12)$$

This is the *Line Jacobi method*; as with the point methods we can define a black/white ordering of the columns and so extend to *Line Gauss-Seidel* and to the more frequently used *Line Successive Overrelaxation (LSOR)* method.[6] The code for LSOR follows readily from Eqn. 12 and that for the tridiagonal algorithm of para. 2.2. It is given in Appendix 1.

The difficulty with schemes of this class is that we must choose which mesh direction to treat implicitly and which explicitly. In some cases where solutions exhibit preferred directions and these coincide with a mesh direction the choice is simple: we solve implicitly in this direction. In this type of problem LSOR works particularly well. In most cases however this is not so and convergence is retarded by the explicit coupling of the columns.

One way to resolve this new problem is to alternate the direction in which the mesh is solved implicitly; this is known as the *Alternating Direction Implicit* (ADI) method[7] and is an extension of the Line Jacobi method. The algorithm can be written

$$T_{i-1,j}^{(p+\frac{1}{2})} - 2\,T_{i,j}^{(p+\frac{1}{2})} + T_{i+1,j}^{(p+\frac{1}{2})} + \rho\, T_{i,j}^{(p+\frac{1}{2})}$$

$$= Q_{i,j} - [T_{i,j-1}^{(p)} - 2T_{i,j}^{(p)} + T_{i,j+1}^{(p)} - \rho\, T_{i,j}^{(p)}] \qquad (13a)$$

$$T_{i,j-1}^{(p+1)} - 2\,T_{i,j}^{(p+1)} + T_{i,j+1}^{(p+1)} + \rho\, T_{i,j}^{(p+1)}$$

$$= Q_{i,j} - [T_{i-1,j}^{(p+\frac{1}{2})} - 2\,T_{i,j}^{(p+\frac{1}{2})} + T_{i+1,j}^{(p+\frac{1}{2})} - \rho\, T_{i,j}^{(p+\frac{1}{2})}] \qquad (13b)$$

Here ρ is in iteration parameter. The method first solves implicitly in the i-direction, giving $T^{(p+\frac{1}{2})}$, and then in the j-direction using the $T^{(p+\frac{1}{2})}$ which has just been found. Both the implicit solutions are found by using the method for the tridiagonal set given in para. 2.2. Whilst with a single iteration parameter ρ the convergence rate of ADI is similar to that of SOR, it can be accelerated considerably by using several parameters in cyclic order.

Once again the DAP Fortran code follows easily from eqn. 13 in conjunction with that for the tridiagonal set given in Fig. 5, para. 2.2. It is given in full in Appendix 2.

4. Some DAP timings

The ADI, LSOR and Point SOR methods as described in this paper have all been used to solve a Poisson problem on DAP with various mesh sizes.[8] The execution times for these runs are given in Table 1.

Table 1. Times per iteration, seconds

Method	Mesh size 64 x 64	Mesh size 256 x 256
SOR	0·00854	0·0532
LSOR	0·0589	0·231
ADI	0·0497	0·420

It is worth noting that during the tridiagonal part of the LSOR algorithm on the 64 x 64 problem (that is, the DO loop to Label 15 in Appendix 1) the DAP is working at only 50% efficiency. This is because at any step through this loop only the results from the alternate lines are used. The efficiency cannot be increased for this size of mesh (unless processors are made to co-operate on some instructions) but a 64 x 128 problem could be solved with the same processing time. If we examine the processing time per mesh point for the cases of Table 1 we see the result of this effect, shown in Table 2.

Table 2. Times per mesh point per iteration, microseconds

Method	Mesh size 64 x 64	Mesh size 256 x 256
SOR	2·1	0·81
LSOR	14·4	3·5
ADI	12·1	6·4

We see that, comparing the times per mesh point for LSOR on a 64^2 and a 256^2 problem, the solution for 16 times as many points is being obtained in only 4 times as long. A factor 2 arises because of the inefficiency of the black/white ordering of the lines and the other factor of 2 from the tridiagonal algorithm.

In order to solve for lines 256 long we first perform two steps substituting for alternate points to reduce to 128 and then to 64 points. We then solve for these 64 points and back-substitute to obtain the solution at all 256 points in each line, having saved the coefficients needed to enable us to do so. These two steps keep all the processors busy all the time and are 100% efficient, whereas during the tridiagonal algorithm the efficiency decreases as zeros are shifted in from the edges. The forward elimination and back-substitution increase the ratio of efficient usage and this accounts for the extra factor of 2 in the move from a 64^2 to a 256^2 problem.

5 Concluding discussion

It has been shown in this paper that a parallel processor such as DAP can be used to solve large finite difference problems effectively and efficiently. It has been shown also that the balance of cost per iteration in the case of the iterative methods is

rather different from what it is on a serial machine. For instance, simple Point SOR appears rather more attractive on DAP than the more widely used LSOR. This is due to an inherent inefficiency within the tridiagonal algorithm implemented on DAP: whilst it is effective, it is only during the first step that all the processors are doing useful work and therefore it is inefficient.

It must however be pointed out that the example chosen to illustrate the solution by SOR and LSOR was a particularly simple isotropic problem. On a stiffer anisotropic problem SOR could well have difficulty and might not converge at all, but LSOR could be expected to do particularly well. Also on such a problem ADI could well run into trouble.

An area which has not been touched on in this paper but which has become of considerable interest recently is the solving of linear equations by preconditioned conjugate gradient methods. The conjugate gradient algorithm is inherently parallel and has been coded in DAP Fortran.[9] With a suitable choice of preconditioning matrix this could give a very powerful parallel algorithm which would avoid the basic problem of the overrelaxation methods, that of choosing the overrelaxation parameter. Experience with this method of attack on the problem will be reported in a subsequent paper in this Journal.

Appendix 1
DAP Fortran for line successive overrelaxation (LSOR) method

```
      SUBROUTINE LSOR (A,B,D,E,G,T,Q,ITN, TOL, OMEGA)
      REAL A(,), B(,), D(,), E(,), G(,), T(,), Q(,), RHS(,), RES(,)
      REAL AT(,), ET(,), GT(,)
      LOGICAL BLACK(,)

      BLACK = ALTC(1)                    set up mask BLACK with alternate
                                         columns .TRUE.
      ITN = 0
      OMEGAM = 1.- OMEGA
      RHS = 0
      AT = A
      ET = E
      GT = G
      TRACE 1  (OMEGA, TOL)              trace variables OMEGA and TOL
5     RES = Q -  (A*T(-,) + B*T(,-) + D*T(,+) + E*T(+,) + G*T)
                                         calculate current residual
      RM = MAXV(ABS(RES))                find maximum absolute residual
      RN = SQRT(SUM(RES**2)/4096         calculate rms residual
      TRACE 2  (ITN, RM, RN)
      IF(RN.LT.TOL.AND.RM.LT.TOL).OR.ITN.GT.100)  GO TO 2
                                         check RM,RN against convergence
                                         criteria
      ITN = ITN + 1
      DO 15 IODD = 1,2
```

```
      RHS (BLACK) = Q - D*T(,+) - B*T(,-)  form RHS on alternate columns
                                           using solution from east and west
                                           neighbours
      K = 1
                                           begin tri-diagonal loop
10    A = A/G                              a_i T_{i-1} + g_i T_i + e_i T_{i+1} = rhs_i
      B= B/G                               a_i → a_i/g_i  etc.
      RHS = RHS/G
      RHS = RHS - A*SHSP(RHS,K) - E*SHNP(RHS,K)
                                           rhs_i → rhs_i - a_i rhs_{i-k} - e_i rhs_{i+k}
      G = 1. - A*SHSP(E,K) - E*SHNP(A,K)   g_i → 1 - a_i e_{i-k} - e_i a_{i+k}
      IF (K.EQ.32) GO TO 11                finish if this is the 6th step
      A = -A*SHSP(A,K)                     a_i → -a_i a_{i-k}
      E = - E*SHNP(E,K)                    e_i → -e_i e_{i+k}
      K = K*2                              double k
      GO TO 10
11    RHS = RHS/G                          complete solution
      A = AT
      E = ET                               restore a,e,g
      G = GT
      T(BLACK) = OMEGAM*T +OMEGA*RHS overrelax solution on BLACK lines
15    BLACK = .NOT.BLACK                   change BLACK to select other set
                                           of lines
      GO TO 5
2     TOL = RM
      RETURN
      END
```

Appendix 2

DAP Fortran for alternating direction implicit (ADI) method*

```
      SUBROUTINE ADI (A,B,D,E,G,T,Q,ITN,TOL,OMEGA)
      REAL A(,),B(,),D(,),E(,),G(,),T(,),Q(,)
      REAL RES(,), RHOLIST( )
      REAL EF(,),WF(,),NF(,),SF(,)
      REAL AT(,),VT(,),ET(,),BT(,),HT(,),DT(,)
      EQUIVALENCE (AT,BT), (VT,HT), (ET,DT)
      EQUIVALENCE (EF,NF), (WF,SF)

      RHOLIST = 0
      RHOLIST(1) = 0·5248616
      RHOLIST(2) = 1·698467                set up Wachpress parameters
      RHOLIST(3) = 7·883137
      RHOLIST(4) = 40·47514

      ITN = 0
100   DO 900  IRHO = 1,4                   perform 4 iterations with different
                                           parameters, using parameters
      RHO = RHOLIST(IRHO)                  cyclically
```

*I am indebted to David Hunt, ICL, for details of this program.

```
      RES = Q - A*T(-,) - B*T(,-) - D*T(,+) - E*T(+,) - G*T
      RM=MAXV (ABS(RES))              calculate maximum residual
                                      first solve implicitly along rows
      BT = B
      DT = D                          calculate coefficients
      HT = - (B + D + RHO)

      N = 1                           tridiagonal loop
200   EF = - BT/SHEC(HT,N)
      WF = -DT/SHWC(HT,N)
      RES = RES + EF*SHEP(RES,N) + WF*SHWP(RES,N)
      HT = HT + EF*SHEP(DT,N) + WF*SHWP(BT,N)
      IF(N.EQ.32) GO TO 210
      BT = EF*SHEP(BT,N)
      DT = WF*SHWP(DT,N)
      N = N + N
      GO TO 200
210   T = T + RES/HT                  update solution
      RES = Q - A*T(-,) - B*T(,-) - D*T(,+) - E*T(+,) - G*T
      RMT = MAXV(ABS(RES))            calculate new residual
      IF(RMT.GT.RM) RM=RMT
                                      now solve implicitly along columns
      AT = A
      ET = E                          calculate coefficients
      VT = - (A + E + RHO)
      N = 1                           tridiagonal loop
400   SF = - AT/SHSC(VT,N)
      NF = -ET/SHNC(VT,N)
      RES = RES + SF*SHSP(RES,N) + NF*SHNP(RES,N)
      VT = VT + SF*SHSP(ET,N) + NF*SHNP(AT,N)
      IF (N.EQ.32) GO TO 410
      AT = SF*SHSP(AT,N)
      ET = NF*SHNP(ET,N)
      GO TO 400
410   T = T + RES/VT                  update solution

900   ITN = ITN + 1                   loop for next iteration
      IF (RM.LT.TOL.OR.ITN.GE.100) RETURN   exit if convergence
                                            is achieved
      GO TO 100
      END
```

References

1 FLANDERS, P.M., HUNT, D.J., REDDAWAY, S.F., PARKINSON, D.: 'Efficient high-speed computing with the Distributed Array Processor' Symposium on High Speed Computer and Algorithm Organisation, Illinois, 13-15 April 1977.

2 SCARROTT, G.G.: *ICL Tech. J.*, 1978, 1 (1), 35.

3 GOSTICK, R.W.: *ICL Tech. J.*, 1979, 1 (2), 116.

4 FOX, L.: *'Numerical linear algebra'* Chapter 3, 7, Oxford University Press, Oxford, 1964.

5 AMES, W.F.: *'Numerical methods for partial differential equations'*, Chapter 2, Nelsons, Sunbury, 1978

6 VARGA, R.S.: *'Matrix iterative analysis'*, Prentice Hall, Hemel Hempstead, 1962

7 PEACEMAN, D.W., and RACHFORD, H.H.: *J. Soc. Ind. Appl. Math.*, 1955, 3, 28.

8 HUNT, D.J., WEBB, S.J., WILSON, A., Conference on Elliptic Problem Solvers, Los Alamos Scientific Laboratory, July 1980. Proceedings to be published by Academic Press, New York.

9 PARKINSON, D.: Private communication.

Data routing and transpositions in processor arrays

C.R.Jesshope
Department of Computer Science, University of Reading, Whiteknights Park, Reading RG6 2AX

Abstract

Processor arrays require the alignment of data with target processors. This creates overheads to some algorithms in the form of routing through some form of network connecting the processors. Routing is often associated with the transposition of sets of data in the processor array's memory. This paper defines and evaluates the use of a k-dimensional cyclic network and proceeds to establish timings for a general class of mapping on this network. Some multidimensional mappings on the two-dimensional DAP array are used to illustrate the results.

1 Introduction

1.1 Processor arrays

Processor arrays are a new breed of computer which make use of parallelism to increase machine performance. Processor arrays normally operate in lock step, with one instruction stream controlling a number of identical processors. Designs for processor arrays vary enormously. At one end of the range are machines such as the Burroughs BSP,[1] which has a small number (16) of relatively complex, floating point processors. This contrasts with machines such as ICL's DAP,[2,3] which has a large number (4096) of very simple processors. If they can be effectively utilised such large arrays of processors hold great promise for the future of computing. Obviously cost must limit the complexity of the individual processors if many are to be put together. However, with the rapid escalation of the so called 'microprocessor revolution', designs like DAP with its bit serial processors may soon be superseded by similar sized arrays of 16 or even 32 bit microprocessors. Alternatively VLSI may make much larger DAP designs feasible, for example 16, 64 or even 256K processor arrays. These large arrays have the potential for very rapid computation. However, to capitalise on this, there are two major problems which must be overcome.

1.2 Parallelism

The most obvious problem associated with large arrays of processors can be summed up by the word 'parallelism'. It is essential in any algorithm to use as many

processors as possible. The array processor is useless if only a few of the processors are effectively being used during a computation. Thus algorithms must be found which have enough parallelism to utilise the full array of processors. There is a growing body of literature (cf[4-6]) which testify to the fact that most algorithms contain inherent parallelism (or if not, alternative algorithms can be found). However what most of these papers fail to consider is one of the fundamental problems associated with processor arrays, that of data organisation and its associated overheads in data routing.

1.3 Data routing

In a processor array the total memory available is generally partitioned between the processors so that each has its own slice of memory. Thus each processor will have access to data in its own memory plus data available from a neighbouring processor. These neighbours will be defined by some connectivity pattern or routing network. In the Burroughs BSP a full crossbar switch is implemented which connects every processor to every partition of memory, thus any processor can access any memory location in one pass through this switch. However the cost of such a switch increases as the product of inputs and outputs and becomes prohibitive for even modest arrays of processors. In the DAP the processors are connected in a two-dimensional grid pattern with cyclic connections possible at the boundaries of the grid.[7] Thus routing can be considered as an operation which transfers data from one processor to its neighbour, or which corresponds to one pass through the switching network. Many such operations or passes may be required to forward data from its source to its destination. Thus data routing can be a large overhead to the parallel implementation of an algorithm. In fact it has been put forward that some algorithms will be limited by data routing.[8] For example with N^3 processors $N \times N$ matrix multiplication can be performed using one multiplication and $\log_2 N$ adds.[9] However $2N$ data-routing operations are required to perform this algorithm on a two-dimensional mesh-connected array. Thus if the routing operation executes at the same rate as arithmetic, data routing would dominate the calculation. This situation is somewhat ameliorated on most processors as routing, being a simple operation, usually executes faster than arithmetic. For example on the DAP the ratio between execution rates of data routing and floating point arithmetic, r, is 30. However data routing can still generate a significant overhead on many algorithms. Furthermore the overhead is likely to increase with the size of the array of processors.

1.4 Data mappings

In more complex algorithms the overheads from data routing often occur in the mapping of one data structure to another. Data structures are varied dynamically to obtain the maximum amount of parallelism at each algorithmic step.[10,11] The mappings usually take the form of transpositions of blocks of data about diagonals in processor-memory space. A simple example of this is shown in Fig. 1 for a four processor array. In Fig. 1*a* a 4 x 4 matrix is stored by rows. The processors therefore have parallel access to rows and sequential access to columns. At another stage in the computation, the algorithm may require parallel access to columns and sequential access to rows. In this case the data structure must be physically trans-

posed as in Fig. 1*b*. These mappings are encountered when partitioning a multi-dimensional computational space onto the processor array. In this example a two-dimensional data array maps neatly onto a single dimensional processor array.

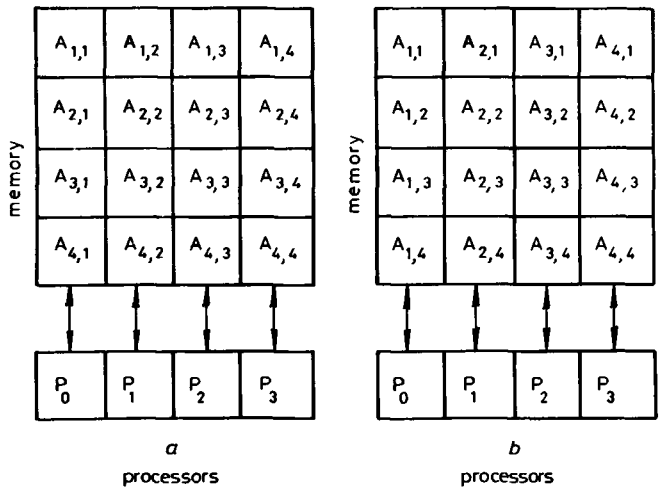


Fig. 1 4x4 matrix stored in a four-processor array
a by rows
b by columns

An alternative data structure can be used to avoid the dynamic mapping in certain circumstances. For example, if each processor can index independently any location within its own memory, then diagonals of the processor-memory space may also be accessed in parallel (e.g. Illiac IV[12]). Fig. 2 shows the skewed matrix storage scheme that allows parallel access to both rows and columns.[13] However it must be noted that to align sequentially accessed rows or columns data must be routed between processors. To access row or column *i* the data must be routed *i*-1 processors to the left. If each processor is connected cyclically to its left and right neighbour, then to access all rows requires four routing operations. Similarly to access all columns requires four routing operations. Compare this with the routing required for the transposition in Fig. 1, where each circulant diagonal must be routed by the number of places shown in Fig. 3*a*. The sum of these distances is 4, so all rows and all columns can be accessed in parallel at the expense of only four routing operations in this case. If rows and columns of a data structure are accessed more than once then the advantages of dynamic data mapping become obvious. This situation occurs in complex algorithms which can be separated into independent operations on more than one dimension of a data structure. One such algorithm is the fast Fourier transform.[11]

Like skewed data storage the transposition of a data structure requires parallel access to diagonals in processor-memory space. This in turn requires independent processor indexing. Where this facility is not available, as in the DAP, a further transformation[14] must take place before and after the data routing. This trans-

formation, illustrated in Fig. 3 in two stages, transforms circulant diagonals to rows and *vice versa*. The general algorithm is described in the Appendix; it requires $N \log_2 N$ memory accesses for N diagonals.

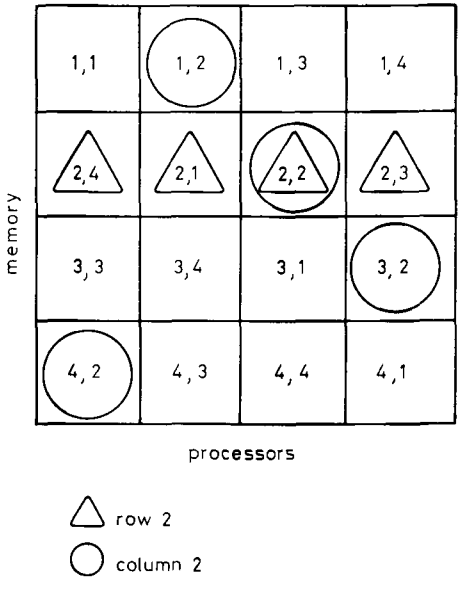


Fig. 2 Skewed matrix storage of a 4x4 matrix in a four-processor array

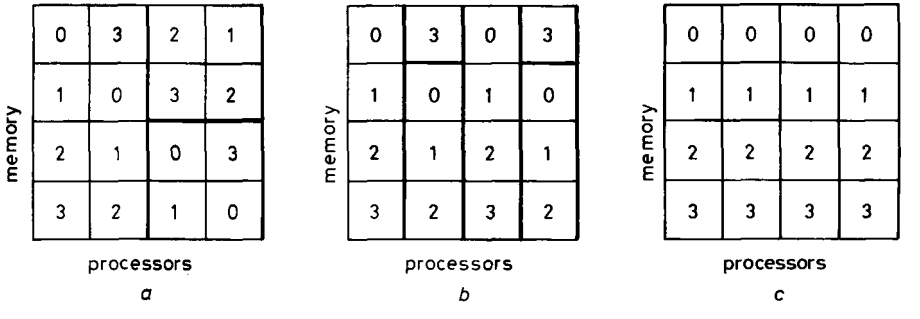


Fig. 3 Example of the exchange algorithm to yield rows from cyclic diagonals, in a 4x4 matrix

2 Some properties of connectivity networks

In the example considered in Section 1 a simple network of connections was postulated, where each processor was connected only to its left and right neighbours. In the DAP a two-dimensional network of connections has been used with cyclic connections in each dimension. In order to keep the results produced in this paper as general as possible, a k-dimensional network of cyclic connections is assumed, where each processor has a 'left' and 'right' neighbour in each dimension.

This network is defined more formally below. However this is not the only possible connection strategy[15] which can be used; Stone[16] and Grosch[17] look at the possibilities of computing with the perfect shuffle network.

2.1 *k-dimensional cyclic networks*

The model used in this paper is of an array of P processors with SIMD organisation. Each processor has its own memory and in addition can communicate with other processors (or their memory) in the array by means of a connectivity network. The connectivity network has P nodes, one associated with each processing element with unique integer identifiers $\{i\}$, where $\{i\} = \{0 \leqslant i \leqslant P-1\}$ and where $P = Q^k$ and $Q = 2^q$. The connections in the network can then be defined as follows. Each node in the set $\{i\}$ can be connected to one of the nodes in the set $\{j\}$, where $\{j\} = \{(i \pm Q^{\ell-1}) \text{ modulo } Q^{\ell}, 1 \leqslant \ell \leqslant k\}$. It follows from the SIMD organisation that each node in the set $\{i\}$ must make corresponding connections in the set $\{j\}$.

Some properties of this network are now considered; some of these properties were derived by Jesshope.[18]

2.1.1 Property 1: If D_k is the maximum number of routing operations required to pass data between any two nodes in the network then

$$D_k = k\,Q/2 \tag{1}$$

This property may be used as a measure of the amount of routing required for problems which involve long range communication; data transpositions in processor-memory space belong to this class of problem. It is one measure of the effectiveness of a given configuration of P nodes. The ideal, that found in scalar computers, is for D_k to have the value of 1.

2.1.2 Property 2: If F_k is the number of routing operations required to pass one item of data from one node to all others in the network then

$$F_k = k\,(Q-1) \tag{2}$$

This property may be used as a measure of the amount of routing required for problems involving fan-out and fan-in. Examples are broadcasting and in any problem which involves the summation of data arrays mapped over the processor array. Again the ideal for F_k is a value of 1. It should also be pointed out that some processor arrays have additional facilities for broadcasting data; however this measure is still a valid one for fan-in problems.

2.1.3 Property 3: If S_k is the sum of all routing operations required to pass P items of data from one node, one to each node in the network, then

$$S_k = k\,Q^{k+1}/4 \tag{3}$$

This property is more concerned with data transposition. In fact it is a measure of the amount of routing required to transpose a $P \times P$ array in processor-memory space. The example in Section 1 illustrates this (see Fig. 3*a*). The next property considered is also concerned with data transposition, it is an extension of property 3 and gives a measure of the routing required to transpose rectangular sets of data.

2.1.4 Property 4: If $S_k\ (B)$ is the sum of all routing operations required to pass P/B items of data from one node i, to all other nodes j where $(i-j)$ is divisible by B (B is a power of 2), then

$$S_k\ (B) = (k - [\log_Q B]_F{}^*)\ Q^{k+1}/4B$$

2.2 Cost and effectiveness

One simple measure of cost which can be applied to switching networks of this type is the number of possible connections which can be established. For example in the cross-bar switch which connects every input to every output, the cost is proportional to the products of inputs and outputs (i.e., P^2 for P inputs and outputs). This configuration is thus only economic for small arrays of processors, the BSP with 16 processors being a good example.

For the k-dimensional cyclic network the cost is proportional to $2kP$, the number of possible connections, with a limit of $P \log_2 P$ for the network which forms a binary hypercube.[19] One- and three-dimensional connection patterns are illustrated in Fig. 4 for an eight-processor array. The crossbar switch would of course have switches everywhere in the matrix.

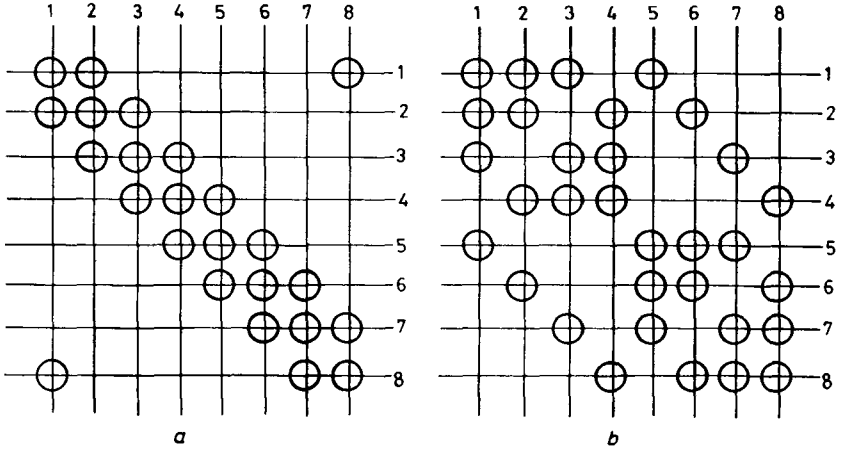


Fig. 4 Connectivity matrices for an eight-processor array
a one-dimensional cyclic *b* three-dimensional cyclic

Table 1 lists some of the measures defined above for different configurations of a 4096 processor array. The cost function used gives the number of connections per

*$[\ ]_F$ and $[\ ]_C$ denote the integer floor and ceiling functions, respectively.

node. It can be seen that all measures are minimised for $k = \log_2 P$, where all long range routing becomes a $\log_2 P$ process. However, for a large array of this type the most cost-effective configuration is given for $k = 2$, as this gives the greatest improvement in the networks effectiveness for a given incremental cost. An order of magnitude change is made going from 1 to 2 dimensions, while the benefits diminish going to higher dimensions.

Table 1 Configuration and properties of a 4096 processor array

k	Cost*	Q	D_k	F_k	S_k
1	2	4096	2048	4095	$4{\cdot}19\times10^6$
2	4	64	46	126	$1{\cdot}31\times10^5$
3	6	16	24	45	$4{\cdot}91\times10^4$
4	8	8	16	28	$3{\cdot}27\times10^4$
6	12	4	12	18	$2{\cdot}46\times10^4$
12	12†	2	12	12	$2{\cdot}46\times10^4$

*Connections per node
†For $Q=2$, $(i+Q^{\ell-1})$ modulo $Q^{\ell} = (i-Q^{\ell-1})$ modulo Q^{ℓ}

3 Data transpositions

The routing requirements for the transposition of data in processor-memory space are now examined. A generalised mapping is defined and the properties given in the previous section are used to derive timings for this mapping.

3.1 The general mapping

The general mapping is defined as a transposition of a rectangular subset of array memory, consisting of L processors by L/B store locations. Obviously if L divides P then P/L such transpositions may be performed simultaneously for the same cost. Also, as the set to be transposed is rectangular, the elements or blocks of memory which are actually transposed must also be rectangular and in the same ratio. Thus diagonals consisting of blocks of B processors by one store location are transposed about the major diagonal, which consists of L/B such blocks (see Fig. 5*a*). It is assumed that B and L are powers of 2.

To perform the transposition, diagonals of the structure must be accessed by the processors, routed by the appropriate distance and restored. Fig. 5 gives routing distances over the linear set for each diagonal. However to access diagonals of processor memory requires an independent processor indexing facility (in a SIMD organisation each processor receives the same address). If this facility is not available on a particular architecture, then the algorithm described in Section 1 and defined in the Appendix must be used. This algorithm will map L/B circulant diagonals onto L/B store rows and to perform this it will require $L/B \log_2 L/B$ exchanges in memory. Once this has been performed the data routing can be performed in parallel and the transposed structure can be recreated using the converse of the exchange algorithm.

Three special cases of the transposition algorithm are considered below. Each gives different results depending on the number of dimensions in the network that L and B span. The routing operations in each case are given by the function $R_k\ (L, B)$ for the k-dimensional network.

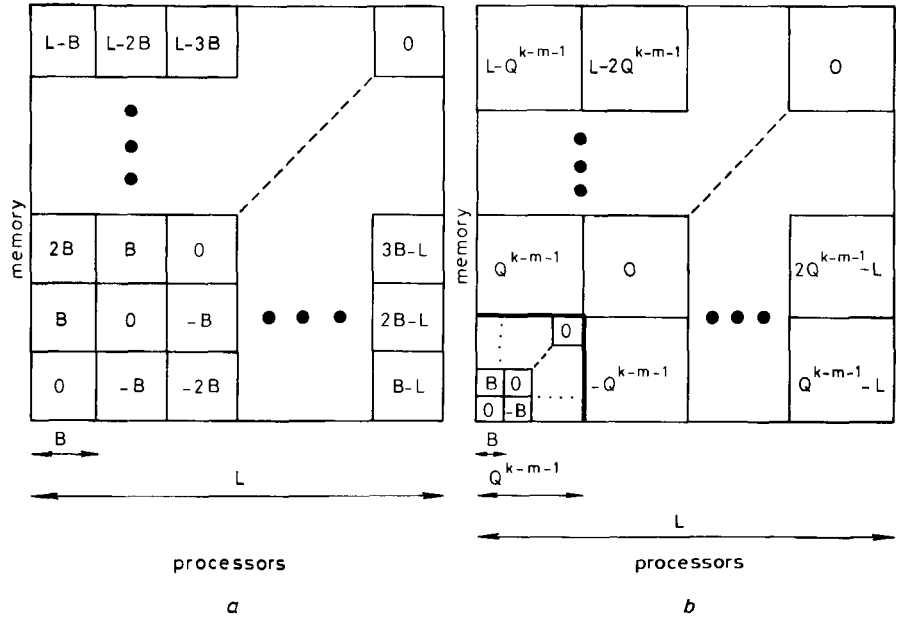


Fig. 5 Transposition of data in a processor array with shift distances indicated for each diagonal
a Case 1 and 2
b Case 3

3.1.1 Case 1 (see Fig. 5a): In this the simplest of the cases it is assumed that L completely spans a subspace of the network of $k-m$ dimensions.

$$\text{i.e. } L = Q^{k-m} \text{ where } 0 \leqslant m < k \tag{4}$$

As each of the diagonals must be routed by multiples of B processors and as all possible multiples in a network of $k-m$ dimensions are used then the number of routing operations are given by property 4 for a network of $k-m$ dimensions.

$$\text{i.e. } R_k\ (L, B) = (k - m - [\log_Q B]_F)\ Q^{k-m+1}/4B$$

Rearranging this and substituting from eqn. 4 gives

$$R_k\ (L, B) = [\log_Q L/B]_C \frac{L}{B}\frac{Q}{4} \tag{5}$$

3.1.2 Case 2 (see Fig. 5a): In this case L does not span a complete subspace in the

network and B and L, the first block and the whole set, are both in the same dimension of the network.

$$\text{i.e. } Q^{k-m-1} < L \leqslant Q^{k-m}/2, \quad 0 \leqslant m < k \tag{6}$$

$$\text{and } B \geqslant Q^{k-m-1}$$

In this case the routing can be considered as if in a one-dimensional network, as all routing operations will be by multiples Q^{k-m-1}. Thus

$$R_k\,(L, B) = \frac{2B}{Q^{k-m-1}} \sum_{i=1}^{L/B} (i-1)$$

which gives

$$R_k\,(L,B) = \frac{L}{Q^{k-m-1}}\,(\frac{L}{B} - 1) \tag{7}$$

It should be noted that the factor 2 in the sum arises from the fact that in the absence of cyclic connections at the set boundary, the upper and lower diagonals must be routed independently in the opposite directions. Thus on average only one half of the processors are active for the shifting operations. It is thus possible to make optimisations in some cases. For example in Fig. 6*a* ($Q = 8, L = 4, B = 1, k = 1, m = 0$) it can be seen that the -3 route can be performed using the combination of the 3 and 2 routes. Also in Fig. 6*b* ($Q = 16, L = 8, B = 1, k = 1, m = 0$) the $-7, -6$ and -5 routes can be saved using combinations of the 3, 4, 5 and 7 routes.

3.1.3 Case 3 (see Fig. 5b): This last case can be considered as a combination of the above two cases. Here the set lies between two dimensions in the network but the first block does not.

$$\text{i.e. } Q^{k-m-1} < L \leqslant Q^{k-m}/2 \quad 0 \leqslant m < k \tag{8}$$

$$B < Q^{k-m-1}$$

In order to make full use of the cyclic connections modulo Q^{k-m-1}, the transposition can be considered as the product of two mappings.

$$\text{i.e. } R_k\,(L, B) = \frac{L}{Q^{k-m-1}}\, R_k\,(Q^{k-m-1}, B)\;|_{\text{case 1}} + R_k\,(L, Q^{k-m-1})\;|_{\text{case 2}}$$

which gives

$$R_k\,(L, B) = [\log_Q Q^{k-m-1}/B]_C \frac{L}{B}\frac{Q}{4} + \frac{L}{Q^{k-m-1}}\,(\frac{L}{Q^{k-m-1}} - 1) \tag{9}$$

Again it may be possible to make some optimisations to the second term in the above expression, as described above for case 2.

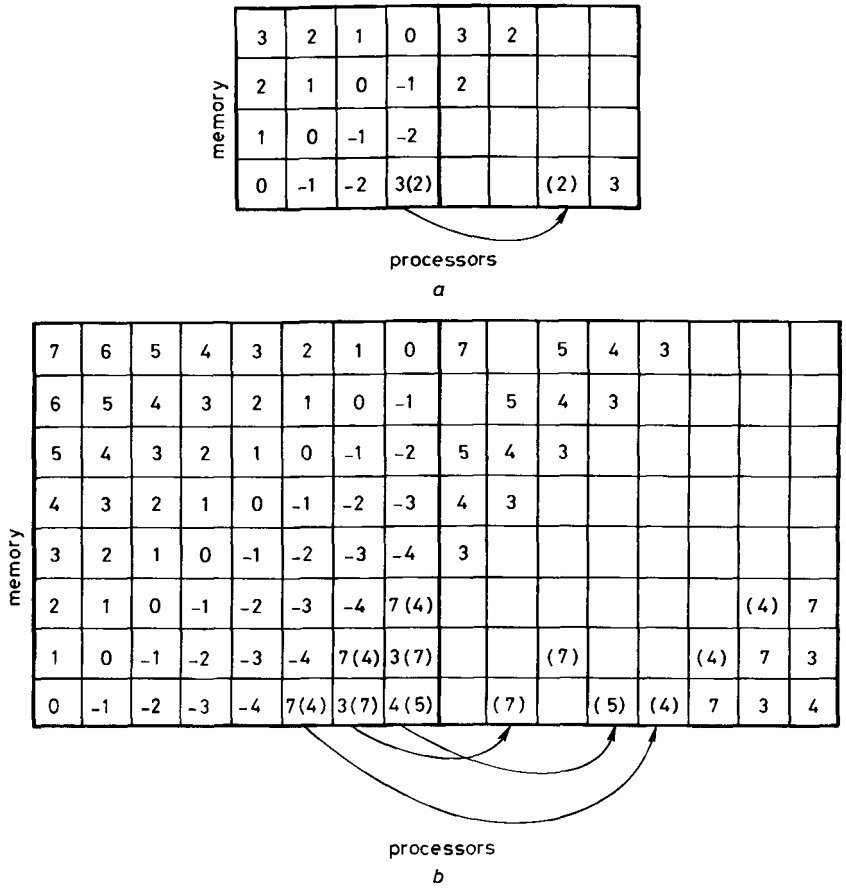


Fig. 6 Examples of optimisations made in case 2
a $Q=8$ $L=4$ $B=1$ $k=1$
−3 route performed using a combination of 3 and 2 routes
b $Q=16$ $L=8$ $B=1$ $k=1$
−7, −6 and −5 routes performed using combinations of 3, 4, 5 and 7 routes

4 Applications

To understand the importance of the results derived above, it is necessary to understand when these transpositions are likely to occur and how they affect the computation in which they occur. The most important point is how much overhead data manipulation adds to a given algorithm and how this may be minimised.

Remember that from Section 2 the measures D_k and F_k gave an indication of the effectiveness of a network. In the ideal case these measures would both be equal to

one and then for any parallel operation operands could be fetched from the memory in one step with no overhead associated with the alignment of the correct data. However for the class of networks considered here both D_k and F_k are greater than one and unless processors only access data in their own partitions of memory, data alignment will generate an overhead to the computation. This overhead will take the form of routing operations through the connectivity network. Truly independent computations on parallel data can be found in some data processing applications: for example in a payroll calculation employee A's hours, rate and salary are independent of employee B's. This is an ideal computation for processor arrays, provided there are sufficient employees to share out between the processing elements. However this situation does not occur often in applications in the physical sciences, although many mesh problems can be made independent by the factorisation of a problem or algorithm. For example relaxation is a factorisation applied to matrix inversion over a set of mesh-defined values. This factorisation reduces long range dependencies to an iteration over short range dependencies. Other mesh-related techniques require the factorisation of an algorithm over the component dimensions of a mesh. For example on certain problems a Fourier analysis in one dimension of a mesh will decouple the problem giving independent problems in the others.[20]

4.1 Factorisation and parallelism

Algorithms which can be factored often give a choice of implementation, as many have sequential and parallel counterparts. However it is often the case that a parallel implementation of an algorithm loses some efficiency in the process of obtaining parallelism. This may mean an additive factor, a multiplicative factor or even an order of magnitude change in the complexity of the algorithm. A good example is the tridiagonal solver; a sequential algorithm requires $O(N)$ operations for an order N system but the parallel implementation of the same algorithm requires $O(N \log_2 N)$ operations. However because N operations may be performed in parallel the algorithm requires only $O(\log_2 N)$ steps and gives a speedup over the sequential algorithm. Now if the problem can be factored and N such system must be solved it is obvious that most benefits will be obtained from applying the sequential algorithm using the parallelism available over the N independent systems to be solved.

In some situations the factorisation is symmetrical and the same algorithm must be performed in each of the dimensions of the factorisation. In this case at different stages in the algorithm the access changes from parallel to sequential and *vice versa.* this implies a transposition in processor-memory space. A good example of this is the fast Fourier transform, where even a one-dimensional transform may be factored down to the prime components in its length.[11] However the most common factorisation is over a mesh of defined values,[10] where for example one-dimensional transforms are performed in each dimension in turn. Although the parallel implementation of the fast Fourier transform is only marginally less efficient than the sequential implementation, it is still more efficient to perform independent transforms on each dimension in turn, transposing the data in processor-memory space.[11] The reason for this is that the parallel implementation also contains long range routing.

4.2 *Multi-dimensional mappings*

To illustrate the overheads associated with data routing in the situations described above, two examples will be considered, which show how the size of the mesh and the extent of parallelism alter the mapping and overheads.

4.2.1 Example 1: Consider first the ideal situation, where a 64^3 mesh is to be mapped onto the 64^2 DAP processor array. Obviously two dimensions map exactly onto the parallelism available, leaving the third dimension as a sequentially indexable set. To transpose dimensions in this structure the routing requirements are given by Case 1 in Section 3 with $L = Q = 64$ and $B = 1$. This gives an operation count proportional to the square of one dimension of mesh array:

$$R_k\,(Q, 1) = Q^2/4 = 1024$$

In comparison, the arithmetic operation count for the fast Fourier transform is proportional to $Q \log_2 Q$ for one dimension in the mesh:

$$A\,(Q) = \tfrac{5}{2}\, Q \log_2 Q = 960$$

Thus the number of routing operations dominates the overall operation count. However on the DAP the relative execution rate between routing the floating point arithmetic, r, is 30 and therefore the overhead generated by the transpositions is not too significant. This situation may easily be reversed if the balance between arithmetic and routing changes. On the DAP this does not require a major hardware change, as the following examples illustrate.

(i) Using low-precision fixed-point representation of data, e.g. picture-processing pattern recognition ($r = 2$).
(ii) Using number theoretic algorithms and integer arithmetic, e.g. number theoretic transforms[21] where 32-bit integer addition and shifting replace floating point arithmetic ($r = 3$).
(iii) Using block floating point representation, where one exponent is used to scale all mantissas within a block of floating point numbers ($r = 15$).

Data structures do not always map neatly onto the array size and when this occurs the interleaving of other dimensions over the processor array creates a greater variety of more complex mappings.

4.2.2. Example 2: Consider now the mapping of a 32^3 mesh over the same 64^2 DAP processor array. The easiest and least efficient approach is to map two of the three dimensions over the processor array using only 25% of the processing power. Other mappings interleave the third dimension across the processor array. For example two symmetrical mappings which interleave the third dimension equally between the other two are shown in Fig. 7. The so-called low-order interleaving is often best for mesh problems as it maintains local routing between adjacent data in all three coordinate directions. What then are the relative costs for symmetrically transposing these two structures?

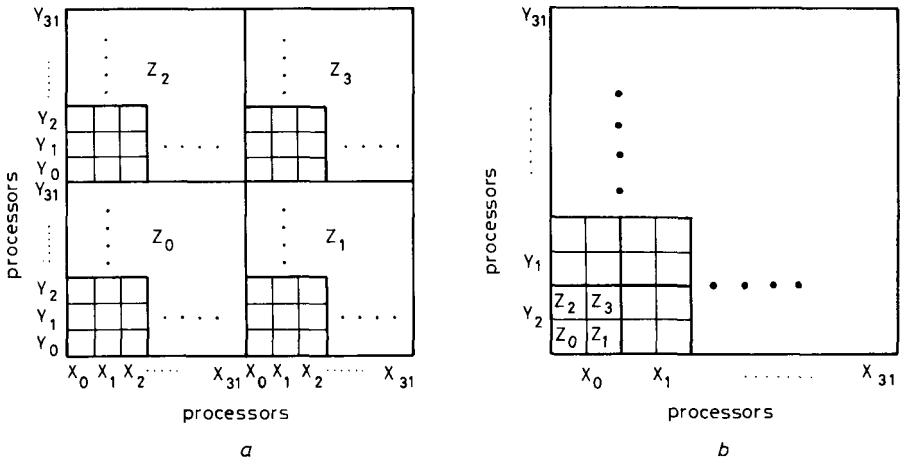


Fig. 7 Symmetrical mappings of a part of a 32^3 mesh over the 64^2 processor array
a High-order interleaving of *Z* in *X* and *Y*
b Low-order interleaving of *Z* in *X* and *Y*

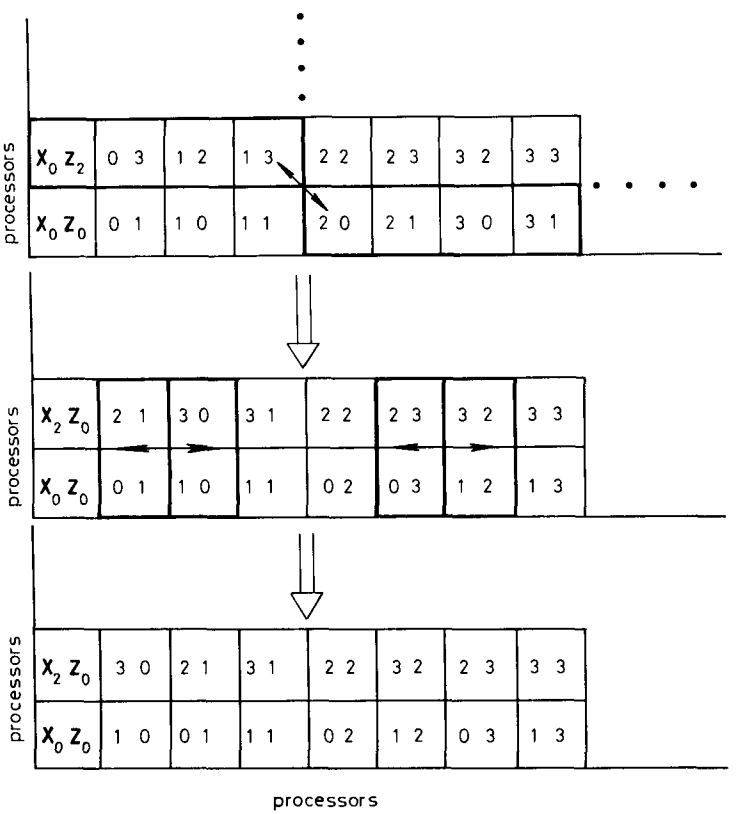


Fig. 8 Exchange pattern required to transpose *X* and *Z* in processor–processor space, for low-order interleaving

Consider the XZ case. Because data in the Z coordinate is interleaved across the processor array some rearrangement is required before a transposition can be made in processor-memory space. This rearrangement transposes the Z and X coordinates in processor–processor space as shown in Fig. 8. The same exchange pattern is performed over all of the array for each level in the indexable set. Having done this the transposition is completed using Case 1 from Section 3 with $L = Q = 64$ and $B = 8$. The total number of routing operations is given by

$$\frac{Q}{8}(10 + \frac{Q}{4}) = 208$$

For the high-order interleaving the exchange pattern in processor–processor space is given in Fig. 9. It can be seen that here the routing required is long range. In addition the transposition in processor-memory space uses the less efficient Case 2 from section 3 with $L = Q/2 = 32$ and $B = 4$.

Thus the total number of routing operations required for this case is given by

$$\frac{Q}{8}(Q + 1) + \frac{Q}{2}(\frac{Q}{8} - 1) = 744$$

which is four times slower than the alternative.

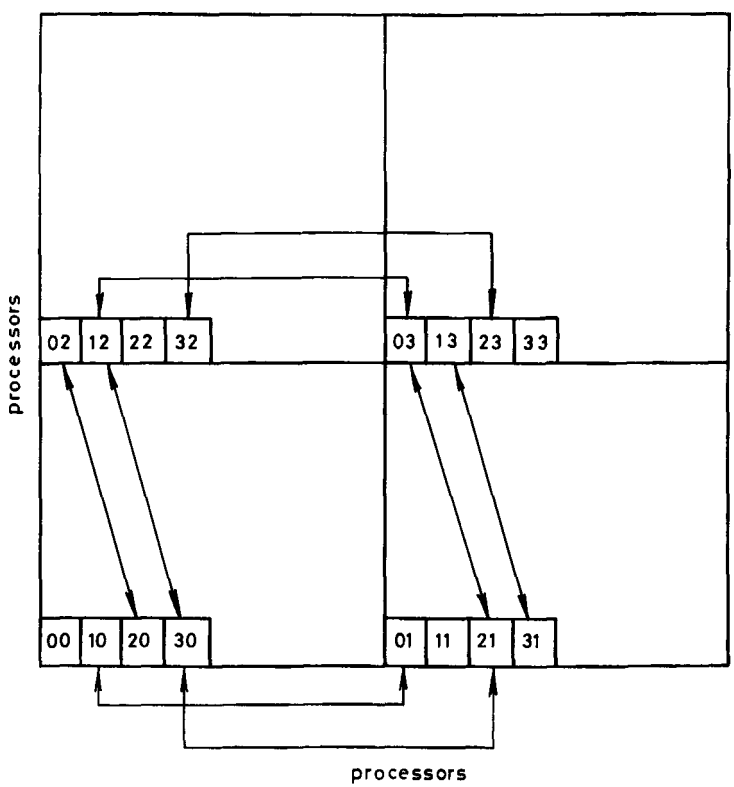


Fig. 9 Exchange pattern required to transpose X and Z in processor–processor space, for high-order interleaving

5 Conclusions

Processor arrays have the capacity for virtually limitless processing power, only if they can be used efficiently and without too many overheads. This paper has investigated one of these overheads, that of data routing, as applied to the transposition of data sets in processor-memory space.

These transpositions are used in many algorithms and the routing required can often dominate the operation count. For example, consider any algorithm which is factored between memory and processors. The algorithm is likely to have complexity of $O(N)$, e.g. tridiagonal solver, or $O(N \log_2 N)$, e.g. fast Fourier transform. However the data routing for the transposition of data is $O(N\ D_k)$, where D_k is given in Property 1 and lies between $O(P)$ and $O(\log P)$ depending on the connectivity structure used. In practice however and actual overhead to calculation will depend on the relative execution rate between routing and arithmetic, r.

Therefore a measure M can be defined as the ratio of the average routing distance per operation and the ratio r. This will give a feeling for the overhead to be expected in algorithms which require communication other than nearest neighbour. If it is assumed that the routing distances are evenly distributed between 1 and D_k, the maximum routing distance in a given network, then

$$M = \tfrac{1}{2}(D_k + 1)/r$$

For a well balanced processor array this measure M should be near unity.

Consider the DAP and BSP, the two contrasting processor array designs mentioned in the Introduction. For floating point arithmetic the measure for the DAP is given by

$$M_{\mathrm{DAP}} = \tfrac{1}{2}(k\ Q/2)/30 \approx 1$$

On the BSP all operands require only one pass through the crossbar switching networks so that $D_k \equiv 1$. However for triadic operations on the BSP $r = 1$, so

$$M_{\mathrm{BSP}} = 1$$

Although both machines are well balanced in the above situation consider the effects of scaling up these two processor arrays. The crossbar switch on the BSP is not suitable for scaling, as the cost of such a switch will soon become prohibitive. Any other switching network would not give $D_k \equiv 1$. On the DAP the situation is different, the array size can be increased easily, but D_k increases as the square root of the number of processors if a two-dimensional geometry is maintained. Routing will therefore become more important on such arrays.

Appendix

An algorithm is given for rearranging sets of data so that cyclic diagonals become rows and *vice versa*. This technique is required when rotating sets of data, to obtain

parallelism in the routing steps, if the processors do not have individual indexing facilities.

For a set of L processors by L/B store rows, with diagonal blocks of B processors by one store row the algorithms are as follows:

For $j=1,2,\ldots,\log_2 L/B$ *do*
disable even groups of $L/2^j$ processors
For $i=1,2,\ldots,L/B$ *do*
move row (i) to row $(i \pm \frac{L}{B2^j})$ modulo L/B
End
End

The minus sign creates rows and the plus sign recreates diagonals. Fig. 3 gives a simple example on a 4 x 4 set.

References

1 AUSTIN, J.H.: 'The Burroughs Scientific Processor' *Infotech s.o.a.r. Supercomputers*, 1979, Vol. II, pp.3–31.
2 PARKINSON, D.: 'An introduction to array processors', *Systems Int.*, 1977, pp.311–329
3 REDDAWAY, S.F.: 'The DAP approach' *Infotech s.o.a.r. Supercomputers*, 1979, Vol.II, pp.311–329.
4 HELLER, D.: 'A survey of parallel algorithms in numerical linear algebra', *SIAM Rev.*, 1978, **20**, 740–777.
5 MIRANKER, W.L.: 'A survey of parallelism in numerical analysis', *SIAM Rev.*, 1971, **13**, 524–547.
6 POOLE, W.G.Jr., and VOIGHT, R.G.: 'Numerical algorithms for parallel and vector computers: an annotated bibliography', *Comput. Rev.*, 1974, **15**, 379–388.
7 FLANDERS, P.M. *et. al.*: 'Efficient high-speed computing with the distributed array processor', *High-speed computer and algorithm organisation*, London, Academic Press, 1977, pp.113–128.
8 GENTLEMAN, W.M.: 'Some complexity results for matrix computations on parallel processors', JACM, 1978, **25**, 112-115
9 JESSHOPE, C.R.: 'Small is O.K. too (another matrix algorithm for the DAP)' *DAP Newsletter*, 1980, **4**.
10 JESSHOPE, C.R. and CRAIGIE, J.A.I.: 'Some principles of parallelism in particle and mesh modelling', In *Infotech s.o.a.r. Supercomputers*, 1979, Vol.II, pp.222-235.
11 JESSHOPE, C.R.: 'The implementation of fast radix 2 transforms on array processors', *IEEE Trans. Comput.*, 1979, C-29 20-27
12 BOUKNIGHT, W.J. *et. al.*: 'The Illiac IV system', *Proc. IEEE*, 1972, **60**, 369-388.
13 KUCK, D.J.: 'Illiac IV software and application programming', *IEEE Trans. Comput.*, 1968, C-17, 758-770.
14 REDDAWAY, S.F.: Private communication, 1977.
15 SIEGEL, H.J.: 'Intercommunication networks for SIMD machines', *Computer*, 1979, **12**, 6, 57-65.
16 STONE, H.S.: 'Parallel processing with the perfect shuffle', *IEEE Trans. Comput.*, 1974, C-20, 153-161.
17 GROSCH, C.E.: 'Performance analysis of Poisson solvers on array processors', In *Infotech s.o.a.r. Supercomputers*, 1979, Vol. II, pp.148-181.
18 JESSHOPE, C.R.: 'Some results concerning data routing in array processors', *IEEE Trans. Comput.*, to be published, 1980.
19 PEASE, M.C.: 'The indirect binary n-cube microprocessor array', *IEEE Trans. Comput.*, 1977, C-26, 458-473.
20 HOCKNEY, R.W.: 'Rapid elliptic solvers', Reading University Computer Science Report RCS 94, 1978.
21 EASTWOOD, J.W. and JESSHOPE, C.R.: 'The solution of elliptic partial differential equations using number theoretic transforms with application to narrow or limited hardware', *Comput. Phys. Comm.*, 1977, **13**, 233-239.

A Bayesian approach to test modelling*

M.Small and C.W.Bartlett
ICL Product Development Group, Northern Development Division, West Gorton, Manchester

Abstract

The problems of designing test systems for medium to large machines at the same time as their development are described. It is shown that there is a consequent need to model the performance of handwritten tests, possibly before they are written and before the hardware design is complete. Use of a Bayesian model is proposed. Definitions, in a Bayesian context, are offered for: test context; test characteristics; test performance; coverage; resolution; and accuracy. The construction of a Bayesian model is described and the experimental verification of its correctness discussed. The paper ends with a description of the utility of the technique during the development of test systems for a small system and for a processor of about 6000 dilics. An indication is given of the utility of the results of a model in the design of field maintenance strategies.

1 Introduction

The exigencies of the development of medium to large processors present severe constraints on the provision of their testing systems. Commercial pressures force a condensed time scale for the design and development phase. In this short time scale there is often not adequate time to provision a generated (i.e. derived from the detailed logic design by algorithmic methods) test system before production starts or even before first customer delivery. It therefore becomes necessary to consider the provision of handwritten tests to cover early production and customer use until such time as sufficient product stability exists for the supply of generated tests to be practical.

However, at the time at which the design of a hand-written test system must commence there is only scant information, regarding the hardware design available. A means is therefore required of integrating the development of the hardware and test systems in order to minimise the risks of ending up with an inadequate means of diagnosing failures.

Further problems arise from the scale of the effort required for handwritten tests. Depending on the ratio of 'random' to 'regular' logic, between 10 and 50 bytes of test code are required for each dilic. For a medium sized processor of c.6000 dilics,

*This paper was presented at the Second International Conference on Reliability and Maintainability, September 8-12, 1980/Perros Guirec, France. It is reprinted here with the permission of the conference chairman.

writing of some 200 kbytes is indicated. With this scale of effort it will be advantageous if it is possible to investigate the applicability of tests previously provisioned for similar designs.

It thus becomes clear that a means of modelling the performance of test systems is required. A model using Bayesian statistics is preferred for a number of reasons.

(*a*) Experimental verification is possible without a need to choose sets of faults according to a presumed distribution of actual faults.
(*b*) The inferences drawn from a particular test result are clearly a function of one's prior knowledge of the state of the system. For example a test, run at random, giving a pass result would be believed. The same test, giving the same result, would not be believed if the device under test was emitting smoke.
(*c*) As evidence emerges from the field about actual fault distributions, the recommended actions on each test result can easily be re-evaluated.

2 A Bayesian model of a test system

A test system may be represented by two pieces of data, one describing the input to the test and the other the test itself. From these data it is possible to derive further data representing the inference which may be drawn when particular test results are obtained.

In the model presented here, the input to the test is described by the set of probabilities, before the test is applied, that the device under test (d.u.t.) is in each of a specified set of working or faulty states. This data is called the test context since it describes the circumstances under which the test is run.

The characteristics of the test are represented by another set of probabilities. For each of the specified test inputs these give the probabilities that each test result would be obtained if that input were present. This data is called the test characteristics.

From these two pieces of data it is possible to calculate using Bayes Theorem a third piece of data giving for each test result the probabilities that this result was obtained because the d.u.t. was in each of its specified states. Hence this data represents the inference which may be drawn and can be called the inference data.

This inference data can be used to compile and evaluate various decision rules which represent the actions to be taken when various test results are obtained in the specified test context. From the decision rule, using the inference data, performance figures for fault coverage, fault resolution and accuracy can be determined.

3 Test context

This is a specification of the classes of input to the test, together with their rates of occurrence. The classes of input may be both types and locations of faults in the d.u.t. The rates of occurrence may be estimated from experience (reliability data) or obtained from a previous test result.

It may conveniently take the form of a table (Table 1).

Table 1: Probabilities of inputs

Test input	A priori probabilities
a_1	$Pr(a_1)$
a_2	$Pr(a_2)$
a_3	$Pr(a_3)$
a_r	$Pr(a_r)$

Note $\sum_{i=1}^{r} Pr(a_i) = 1$

3.1 Classes of faults

Any set of classifications for faults which is useful may be used providing the fault classes do not overlap. In our use of this model we were interested in predicting the performance of the test system for solid failures in terms of resolution to exchangeable spares item. Therefore we considered all solid failures on each PCB to be a class. Each test input was defined to be any fault from that class being present.

A test input representing the absence of the specified fault classes may be included if this is relevant (e.g. when running tests at random or when evaluating system software).

3.2 Estimation of rates of occurrence

The probable rates of occurrence of each test input can be estimated from reliability data; this being either reliability predictions or field returns. An understanding of the variations in these data must be applied. For example reliability can be expected to change with the age of the individual system. Differing values of test input probabilities can be calculated to suit the circumstances and more than one test context may be relevant.

As an example consider the case where a single solid failure has occurred in the d.u.t. The probability that this failure lies in any given area of the d.u.t. is pro-

portional to the failure rate (λ) for that area. (The failure rate being the reciprocal of the mean time between failures (m.t.b.f.).)

$$\lambda(a_i) = 1/\text{m.t.b.f.}(a_i)$$

$$Pr\,(a_i) \propto \lambda\,(a_i)$$

Since a failure has occurred somewhere within the d.u.t. the constant associated with this proportionality (K) can be determined as follows:

$$Pr\,(a_i) = K.\lambda\,(a_i)$$

since $\sum_{i=1}^{r} Pr\,(a_i) = 1$

then $K = 1/\sum_{i=1}^{r} \lambda\,(a_i)$

4 Test characteristics

This is a specification of a test covering the specified classes of inputs to the test and the results which may be obtained. It gives for each element of the test context the probabilities, given that the test was applied in the presence of that input, of obtaining each of the test results. This specification takes slightly differing forms depending upon the nature of the test. For a fixed test sequence a table is appropriate. Where tests are conditionally applied the tables may be assembled into a tree.

4.1 Fixed test sequences

Where a test can be considered to be a series of subtests which are applied in strict sequence until the first positive (failure) result is obtained this is called a fixed test sequence. This is usually the case for test programs and may be usefully applied to selected sequences of system software (e.g. initial program load). The results of such sequences can be called Test (1) Fail to Test (s) Fail, where there are s sub-tests. In addition there is a negative result (all tests pass). Thus a fixed sequence of s sub-tests has $s + 1$ results. The characteristics of a fixed test sequence can be usefully represented as a table (Table 2).

4.2 Estimating test characteristics

Test characteristics in the form shown in Table 2 can be estimated with increasing accuracy as information concerning the d.u.t. and the test itself becomes known in more detail. In the early stages of a project only the approximate structure and partitioning of the d.u.t. may be known. At the same time the structure of the test

Table 2 Test characteristics

$Pr(b_j|a_i)$ = probability of obtaining result b_j if the test input is a_i

	Test result		
Test input	Fail (1) (b_1)	Fail (s) (b_s)	All pass ($b_s + 1$)
a_1	$Pr(b_1\|a_1)$	$Pr(b_s\|a_1)$	$Pr(b_{s+1}\|a_1)$
a_2			
a_3			
a_r	$Pr(b_1\|a_r)$	$Pr(b_s\|a_r)$	$Pr(b_{s+1}\|a_r)$

itself may only be an outline proposal. At this stage very approximate estimates, based on experience, may be made. These will be useful to estimate the general testability of the system and to evaluate proposals for tests. Later on when schematic diagrams for prototypes become available and the draft code for the test software is written a much more accurate set of figures can be constructed. This can be done by considering possible failure modes and their consequences. An example of this is given in.[1] This information can be built up incrementally and estimates of varying accuracy combined.

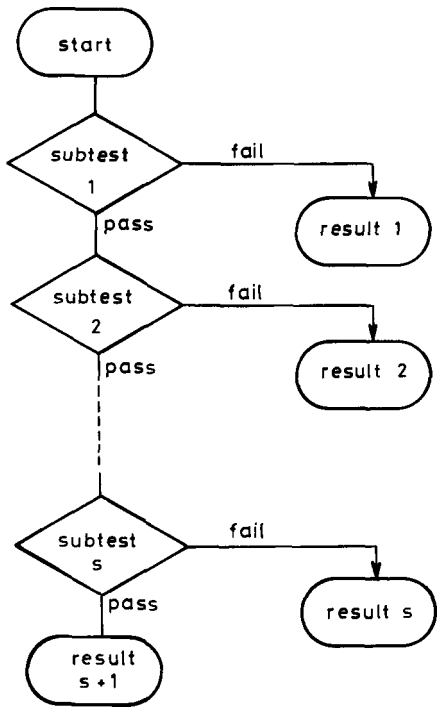


Fig. 1 A fixed test sequence

This process of estimation is of course intensely subjective. However a few simple techniques can be used to guard against the dangers of this subjectivity. First, the estimates should always be jointly agreed between the hardware designer and the test designer. The hardware designer will tend to overestimate, the test designer to underestimate, so a reasonable balance will be obtained. Secondly, once a set of estimates has been derived which are consistent with achievement of the overall system objectives, then these estimates should be adopted as targets and progress towards achieving these targets should be monitored by experimental insertion of faults as the hardware is commissioned. Any effects of under achievement can then be evaluated by insertion in the model and appropriate management actions or engineering trade-offs determined.

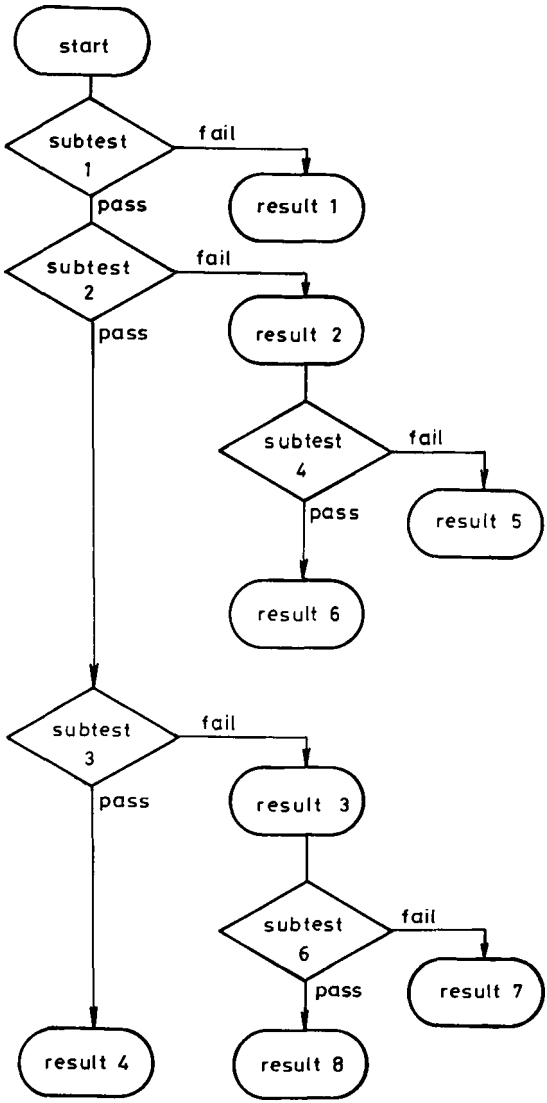


Fig. 2 A conditional test sequence

Nevertheless one may still be left with doubts. However our experience shows that it is better to have subjective information than no information at all. In addition subjectivity may have its own benefits in that experience and expert knowledge can be incorporated. This is not usually the case for automatically generated data.

4.3 Conditional test sequences

Where the next test to be applied at any point in a sequence of tests depends upon the outcome of one or more previous tests this is known as a conditional test sequence. Such sequences are typically used to narrow down the area containing a fault. Such sequences may be represented as a tree of fixed test sequences (Fig. 2).

Each test or fixed test sequence can be represented by test characteristics data as previously described. It should be noted that this data will depend upon the place of the test in the sequence unless the tests are completely independent.

5 Inference data

This data can be calculated from the test context and test characteristics using Bayesian statistics. It gives the probability of obtaining each result and the probabilities for each result that this arose as a consequence of each of the test inputs being present.

5.1 Result probabilities

Let the probability of obtaining the jth result (b_j) be $Pr(b_r)$

Then $$Pr(b_j) = \sum_{i=1}^{r} Pr(a_i).Pr(b_j|a_i)$$

5.2 Backward probabilities (Bayes Theorem)

Let the probability that the jth result (b_j) arose because the ith test input (a_i) was present be $Pr(a_i|b_j)$

Then $Pr(a_i|b_j) = Pr(b_j|a_i).Pr(a_i)/Pr(b_j)$

5.3 Test performance

The actual performance of the test system when the actions following the test are defined by some specified decision rule, can be evaluated using the inference data. The usual dimensions of test performance are fault coverage, fault resolution and accuracy of resolution.

5.4 *Fault coverage*

This is the proportion of the specified set of fault classes (i.e. test inputs) detected by the test in the specified context.

Let the fault cover be C. Since all the test results except the last (s + 1th) represent the detection of a fault the implicit decision rule is that a fail result is to be taken as a fault detected.

Then $C = \sum_{j=1}^{s} Pr(b_j)$

5.5 *Fault resolution and accuracy*

The fault resolution is the number of items implied as being faulty by the test results used with a particular decision rule. The accuracy of this resolution is the proportion of times that this will be correct (i.e. the fault lies within the listed items).

To determine resolution and accuracy a decision rule must be specified and the number of items associated with each test input must be known. Two common decision rules which can be automatically derived from the inference data are:

Best m items for each result;

at least y accuracy for each result.

5.6 *Best m items for each result*

Let the accuracy with which m items can be determined to be faulty from the jth result (b_j) be $A_m(b_j)$.

Let the number of exchangeable items associated with the ith test input (a_i) be $S(a_i)$.

Let the nth most likely test input to contain the fault be a_n.

Then $A_m(b_j) = \sum_{n=1}^{k} Pr(a_n | b_j)$

where k is such that

$$\sum_{n=1}^{k-1} S(a_n) < m \leqslant \sum_{n=1}^{k} S(a_n)$$

Let the average accuracy of the test sequence be A_m.

$$\text{Then } A_m = \sum_{j=1}^{s} Pr(b_j).A_m(b_j) \Big/ \sum_{j=1}^{s} Pr(b_j)$$

$$\text{giving } A_m = \sum_{j=1}^{s} Pr(b_j).A_m(b_j)/C$$

5.7 *Given accuracy for each result*

Let the number of exchangeable items within which the fault lies with an accuracy of at least y for the jth result be $\text{Ry}(b_j)$.

$$\text{Then Ry}(b_j) = \sum_{n=1}^{k} S(a_n)$$

where k is such that:

$$\sum_{n=1}^{k-1} Pr(a_n|b_j) < y \leqslant \sum_{n=1}^{k} Pr(a_n|b_j)$$

Let the average resolution of the sequence be Ry.

$$\text{Then Ry} = \sum_{j=1}^{s} Pr(b_j).\text{Ry}(b_j)/C$$

Note that the average accuracy of the sequence will be at least y and will usually be greater than y. Its actual value can be calculated as above. The automatically derived lists of items to be changed in order of preference against each test result can be used in the field.

The average resolution can be expressed as a histogram or frequency table for numbers of items. Curves of average resolution against accuracy may be constructed. It can be seen that resolution may be increased at the expense of accuracy and *vice versa*.

6 Measurement

The above techniques have been applied to the construction of the test software, and to a more limited extent the system software, for the ICL ME29. The results

obtained were confirmed by fault application experiments. These used the technique known as sequential sampling to reduce the numbers of faults which had to be applied to obtain a sufficiently accurate result with a high confidence. This technique was used to compare the observed fault coverage, resolution and accuracy of the tests with the modelled figures. These experiments showed the tests to be slightly better than as modelled. The results are summarised in Table 3. Similar experiments were conducted for the system software initial program load.[1]

7 Utility of the technique

The modelling of the test system for the ICL ME29 was performed using this technique from an early stage of development. In attempting to construct the models and from the results obtained it was found that relevant issues concerning the testability were exposed. Information was obtained from the exercise which made

Table 3 Measured results

Quantity	Predicted value	Measured value (0·95 confidence)	
		Lower limit	Upper limit
Coverage	0·95	0·89	0·98
Resolution	0·8 to 2 items or less	0·89	0·98
Accuracy	0·95	0·92	0·99

it possible to design changes both to the hardware and test software to overcome the problems raised. The information provided by this modelling was also of use in planning the logistics of the maintenance service. Since the technique is not limited only to test software a useful improvement to system diagnosability was obtained by applying it to the initial program load.

Following the success with the ME29 project the technique has been employed for the design of the testing system for a medium-sized high-speed processor. The design of the processor was evolved from the processors used on ICL 2950 and 2960 systems, so there was an existing body of tests of potential applicability. The modelling technique was eminently successful in giving an early indication of those tests which were still of use and those which would need replacing. Whilst not yet complete, the results obtained to date indicate that an average resolution of 4·13 PCBs with an accuracy of 95% should be achievable. Since the prime requirement of this test system is the location of the failure to a small spares kit for transportation to the site, this result is regarded as satisfactory. (On arrival at the site, further resolution within the indicated area is achieved by interactive use of in-built engineer's facilities and tests.)

8 Conclusion

An argument for a Bayesian test modelling has been presented. The way in which

such models may be constructed has been described. Experimental results have demonstrated the effectiveness of the technique.

Future work in this area will be concerned with the incorporation of results from the Bayesian model into a model of the maintenance activity. The objective of this work will be to optimise the trade-off between accuracy and resolution with respect to staff and spares holding requirements.

Acknowledgments

J.B.Bell for his valuable encouragement, J.D.Bilham, R.Lakin, J.J.Stewart and W.Terrington for their use of the system, J.Aspden and M.Schwarzer for the fault application and experimental verification, D.Massaux for translating the abstract.

References

1 LAKIN, R.: 'ME29 Initial program load – an exercise in defensive programming' *ICL Tech. J.*, 1980, 2(1), 29-46.

2 SHIELDS, S.: 'A review of fault detection methods for large systems *Radio Electronic Eng.*, 1976, 46(6), 276-280

3 SHIELDS, S.: 'Engineering diagnosis the state of the art/science' in 'Technology – does it work in the process industries?' Proc.I.Mech.E. Conf., Sept., 1975.

4 ASPDEN, J.: 'Fault application exercise on ME29' Unpublished work, ICL Kidsgrove, 7th March 1980.

Notes for authors

1 Content

The *ICL Technical Journal* publishes papers of a high technical standard intended for those with a keen interest in and a good working knowledge of computers and computing, but who nevertheless may not be informed on the aspect covered by a given paper.

The content will have some relevance to ICL's business and will be aimed at the technical community and ICL's users and customers. It follows that to be acceptable, papers on more specialised aspects of designs or applications must include some suitable introductory material or references.

The Journal will usually not reprint papers already published, though this does not necessarily exclude papers presented at conferences. It is not necessary for the material to be completely new or original (but see 10, 12 and 13 below). Papers will not reveal matter related to unannounced ICL Products.

2 Authors

Anyone may submit a paper whether employed by ICL or not. The Editor will judge papers on their merits irrespective of origin.

3 Length

Full papers may be of up to 10 000 words, but shorter papers are likely to be more readily accepted. Letters to the Editor and reviews may also be published.

4 Typescript

Papers submitted should be typed in double spacing on one side of A4 paper with full left-hand margin. Mathematical expressions are best written in by hand. Care should be taken to form Greek letters or other unusual symbols clearly. Equations referred to in the text should be numbered. Detailed mathematical treatments should be placed in an Appendix, the results being referred to in the text.

At least two copies should be submitted, both carrying the author's name, title and date of submission.

5 Diagrams and tables

Line diagrams supplied will if necessary be redrawn before publication. Be especially careful to label both axes of any graphs, and mark off the axes with values of the variables where relevant.

All diagrams should be numbered and supplied with a caption. The captions should be typed on a separate sheet forming part of the manuscript. Since diagrams may have to be separated from their manuscript every diagram should have its number, author's name and brief title on the back.

All diagrams and Tables should be referred to in and explained by the text. Tables as well as diagrams should be numbered and appear in the typed MS at the approximate place, at which they are intended to be printed. Captions for Tables are optional. Be careful to ensure the headings of all columns in Tables are clearly labelled and that the units are quoted explicitly in all cases.

6 Abstract

All papers should have an abstract of not more than 200 words. This ought to be suitable for the various abstracting journals to use without alterations.

7 Submission

Before submission authors are strongly urged to have their MSS proof read carefully by a colleague, to detect minor errors or omissions; experience shows that these can be very hard for an author to detect. Two copies of the MS should be sent to the Editor.

8 Referees

The Editor may refer papers to independent referees for comment. If the referee recommends revisions to the draft, the author will be called upon to make those revisions. Minor editorial corrections, e.g. to conform to a house style of spelling or notation, will be made by the Editor. Referees are anonymous.

9 Proofs

Authors will receive printed proofs for correction before publication date.

10 References

Prior work on the subject of any paper should be acknowledged, quoting selected early references. It is an author's reponsibility to ensure references are quoted; it will be unusual for a paper to be complete without any references at all.

11 Style

Papers are often seen written in poor or obscure English. The following guidelines may be of help in avoiding the commoner difficulties.

- Be brief.
- Short sentences are better than long ones but on the other hand do not write telegrams.
- Avoid nested relative clauses; preferably start new sentences.
- Define the meaning of ordinary words used in special senses. Define acronyms or sets of initials by quoting the full meaning the first time the initials are mentioned.
- Include a glossary of terms if necessary.
- Avoid words in brackets as much as possible.
- Avoid the frequent use of the type of construction known as a 'buzzword'. This often takes the form of a noun followed by a present or past participle followed by another noun e.g. 'system controlling parameters'.
- Take care in using the word 'it' that the reader will easily understand what 'it' refers to. An unambiguous rule, that cannot always be applied, is that 'it' refers to the nearest preceding noun in the singular.
- Several 'its' in one sentence each used in a different sense can cause considerable confusion. Similar remarks apply to 'this', 'that' and other prepositions.

12 Copyright

Copyright in papers published by the ICL Technical Journal rests with ICL unless specifically agreed otherwise before publication. Publications may be reproduced with permission and with due acknowledgement.

13 Acknowledgements

It is customary to acknowledge the help or advice of others at the end of papers when this is appropriate. If the work described is not that of the author alone it will usually be appropriate to mention this also.