ICL Technical Journal

Volume 1 Issue 2 May 1979

ICL Technical Journal

Contents

Volume 1 Issue 2

ICL Technical Journal

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

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

Annual subscription rate: £5 (Details from: The 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

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

Computers in support of agriculture in developing countries

G.P. Tottle
ICL Product Development Group, Technology Division, Manchester

Abstract

This paper describes the facilities of a generalised system, SCAPA (system for computer-aided agricultural planning and action) which explores how far small computer systems can support the planning and production activities of large numbers of small farmers in developing countries. The facilities are discussed in relation to the real requirements, which were encountered in joint field studies during 1978, for coffee farmers with the Kenyan Ministry of Agriculture, and for rubber farmers with the Rubber Industry Smallholders Development Authority in Malaysia.

1 Introduction

'Give a man a fish and he'll eat for a day; teach him to fish and he'll thrive for a lifetime.' The proverb has become a part of the conventional wisdom of development economics–how can the relevant results of research, which have immense potential to increase for example agricultural production, be selected and communicated to small farmers in developing countries? And once the technology has been communicated and accepted, how can the small farmers' activities be supported, in terms of agricultural inputs, credit, advisory services and marketing?

The agricultural output that is possible, and the pitfalls towards its realisation, can be illustrated from a joint ICL study with the Rubber Industry Smallholders Development Authority (RISDA) in Malaysia. For a quarter of its 25-year life cycle the main crop, rubber, underexploits its environment–commonly one sees young trees at 4m intervals lined up on bare, moist, sun-baked earth which seems to pulsate with productive potential. There is an obvious opportunity for intercropping with other plants, but technology, environment, crops, markets and logistical support all have to be evaluated and established before this potential can be realised. In one carefully planned and monitored scheme in Pokok Sena, 20 acres (8 ha) of such immature rubber on poor soil were being intercropped to maximise output. This involved heavy use of fertilisers, production credit, a complicated crop rotation and assured market outlets, and, the primary requirement, the enthusiasm, commitment and understanding of the individual smallholders who ran it. The result was three to four varied crops per year, with a net return to farmers of roughly £800 per year per acre (£2000 per year per hectare), from land which would otherwise have lain fallow.

This scheme illustrates at its best the spectrum of opportunities, the range of choice and challenge open to smallholders; if it can be repeated elsewhere, as RISDA in its group replanting schemes is attempting, and if further opportunities for propagating allied processes in rural areas can be created, the wealth and attractiveness of farming can be increased and the drift of the younger farmers to the towns may be reversed. However, if this illustrates the potential, the problems in achieving it for large numbers of smallholdings–100,000 coffee farmers in Kenya, or 150,000 rubber farmers in Sri Lanka for instance–are formidable. Farmers are at the centre of networks of interdependent activities, and problems in any one of the dependencies at Pokok Sena, for instance, could seriously set back the introduction of new techniques. The setback here would however, be less serious than in the less resilient but more straightforward monocropping pattern–a pattern that is commonly followed in agricultural-development projects because it is more easily managed.

This illustrates the main requirements, to meet which the SCAPA system was framed: to explore how far computer facilities might assist in this situation, where on the one hand the complexity of choice and agricultural system interactions, and the large volume of transactions make computer use a natural conclusion, while on the other hand computer technology may seem unintelligible, costly, and inappropriate. The opportunity cost of one 2903 in terms of 'borehole equivalents' for example is clearly very high; this cost is unacceptable if the end result is to give an illiterate farmer a sheaf of PERT output, or worse to override the farmer's well informed judgement by a set of instructions that are less effective, or inappropriate to his needs.

This paper outlines, against this background, the facilities of the SCAPA system (system for computer-aided agricultural planning and action) in the light of feasibility studies carried out during 1978 in Kenya with the Ministry of Agriculture and in Malaysia with the Rubber Industry Smallholders' Development Authority. ICL designed the basic system in collaboration with Professor Black and members of the Department of Computation at the University of Manchester Institute of Science & Technology, with great benefit from discussions with overseas development groups at the universities of Nottingham, Reading and East Anglia, and with the essential participation in particular of Dr. C.L.A. Leakey and Major T.F. Ellis. Their practical experience, and the views and ideas from field staff in Kenya and Malaysia, were indispensable. The basic concepts, specification and potential of the system have been established, and the next stage, a pilot implementation under the Ministry of Overseas Development, is currently being planned.

2 Field background to feasibility studies

Two 1-month studies were carried out. The first took place in two coffee growing districts in Kenya: Murang'a, where a highly successful small farm production system is well established; and Kisii, an area of high potential where, however, less than one-third of the productivity is being achieved. In these areas about one-third of the cultivated land is assigned to arabica coffee, a crop which produces high returns but demands careful attention and costly agricultural inputs at critical time intervals. The Ministry of Agriculture provides a specialist advisory service, and production and marketing are organised by small growers co-operatives, each covering up to three factories. Factories are normally less than 3 km from the farms. They tend to serve about 100 growers with plots of 1 to 6 acres (0•4-2•4 ha) in size, providing agricultural supplies and facilities for processing and marketing the produce and accounting for farm output. Our study involved visits to a small but reasonably representative number of farms to discuss production and problems, and a follow-up through the infrastructure to assess how far SCAPA facilities might assist in solving problems or exploiting opportunities.

The second study, in Malaysia, concentrated on the state of Kedah, a designated poverty area. RISDA, The Rubber Industry Smallholders Development Authority, has a very wide remit to promote and support the interests of the 'man or woman behind the rubber tree'–280,000 smallholder families with holdings averaging 5•5 acres (2•2 ha). Thus, in addition to supporting the mainstream technical activity of replanting, new planting, production and processing, the Authority runs a variety of projects with farmers, locally controlled, to explore opportunities for new crops, livestock, new markets, rural industries or trade, and tries to pull some of the industrialised activities back from the towns to the villages. This involves an imaginative, project-oriented approach, with RISDA's extension officers (advisers) acting as catalysts in a less integrated system than in Kenya–marketing, for example, is organised by RISDA in fortnightly auctions of unsmoked rubber sheet, transacted directly between growers and buyers, with RISDA staff stepping in to bid and purchase only if the price offered seems unreasonably low.

The following Sections summarise the SCAPA facilities, and the conclusions reached during the field studies, under the following heads:

(a) farm profile recording and analysis
(b) plan development
(c) progress recording and monitoring
(d) advisory/extension services
(e) agricultural input supply
(f) credit assessment and control
(g) marketing
(h) management and project monitoring.

These functions are modular with the intention that users can select, and tailor if needed, those functions appropriate to their needs. Before embarking on the detailed section a brief study of Table 1, and Figs. 1 and 2 showing the system aims and the overall application phases and structure may be useful.

3 Farm profile recording and analysis

The system provides for a profile of the capability of each individual farm to be built up gradually, extending from basic identification information to more detailed data defining the farm's production capability, the labour availability, soil type and depth, gradient, aspect, exposure etc. Once captured, this data can be analysed and matched with a variety of possible cropping plans to indicate their relative merits and eventually to vet the farmer's proposed plan and report any potential problems.

Table 1 Aims of SCAPA

The system's prime target is to support individual growers and extension staff. To do so, it provides a back-up system to:

match farm potential to production plans
ensure key activities are recorded in plans
monitor and support their achievement
raise problems for solution
free experts to concentrate on opportunities
provide credit relative to farmers' achievements
schedule and check the provision of inputs
support and record marketing of produce
summmarise progress for management

Capturing this data in sufficient detail for valid technical conclusions to be drawn is expensive—a similar system in Bulgaria,[1] for instance, records 17 data items on soil capability—taking about 3 man days per farm. On the other hand it provides a semipermanent record against which the continuing stream of technical possibilities can be evaluated in the future.

In Kenya, profile recording and analysis offered considerable potential for exploration. Excellent research recommendations from the Coffee Research Foundation, for instance, identified, among others, optimum phosphate fertilisers for 12 possible combinations of calcium, phosphorus and acidity.[2] Digesting and applying this recommendation to a particular situation involved a complicated set of conditional decisions—a competent extension officer might take 40 minutes study to come up with the right answer for a particular farm. In the field, however, basic soil data, even for acidity levels, and even on an area basis was not yet available, and in its absence there was a single standard field recommendation that all farms apply two fertilisers, CAN and ASN, in fixed quantities, in alternate years. If one bears in mind that input costs for coffee are high—roughly 40% of total production costs—and that a high return, over £500 per acre (£1250 per hectare), is common, the process of data trapping and analysis seems attractive. In place of the above standard field recommendation, a precise recommendation as to quantity and type of fertiliser can be made for each farm, using the computer system. As an extension, the effects of cropping, run-off etc. on soil quality might be monitored dynamically to maintain a continuing awareness of residual fertility levels. In Malaysia, profile data for the farms was already being recorded, under IDMS on a 2903 computer, though the depth of coverage needed to be extended for valid technical conclusions on soil capability to be drawn. Since rubber is less demanding in its input requirements and offers a lower return, the benefits of detailed recording of soil data are debatable unless intercropping young rubber with high-value cash crops is envisaged.

4 Plan development

The system requires time-barred 'action lists' as the plan for each farm, showing each major activity, its expected completion date, its dependencies, and its phased

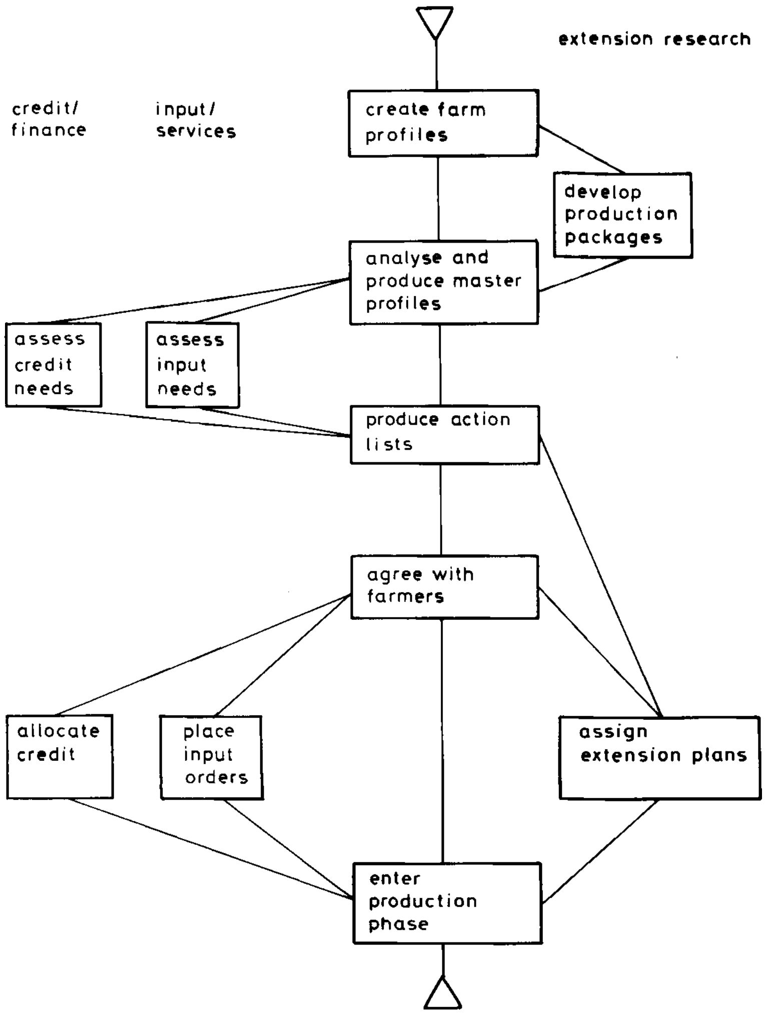


Fig. 1 SCAPA planning phase

resource requirements in man days or quantities of inputs, services or credit. These are drawn up by agricultural advisers–extension staff–as a set of models for equally sized plots to cater for a wide range of farm capability and farmer intentions: a young 'emergent commercial' farmer is likely to set different goals and different crop mixes and to accept different levels of risk from a traditional farmer who is seeking primarily varied subsistence crops for a large family, but with some cash crops as a side line.

The end result of the planning activity is a variety of action lists, which are discussed in detail with each farmer, who then selects that which meets his own requirements or drafts a special list if needed. Table 2 gives part of a schematic 84-stage action list for rubber and tapioca over 5 years. In contrast to the glossies

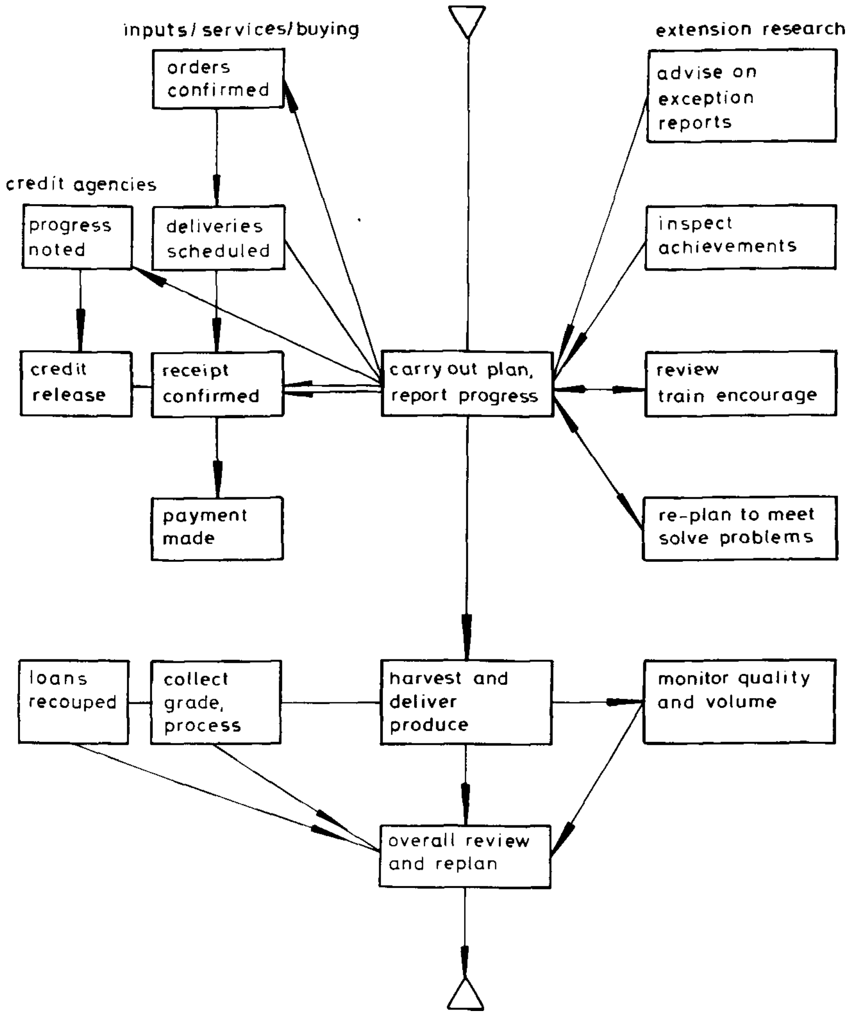


Fig. 2 SCAPA production phase

and pamphlets often distributed, these lists are precise as regards timescale and the inputs and returns for each individual farm–it is expected that farmers will keep them pinned up for ready reference to the current window of activities, much as dairy farmers do currently in the UK for cattle feeding and milk recording.

Research and extension advisers are responsible for the action lists, for evaluating costs, labour requirements, returns, markets etc. This is a tough assignment but it is one that provides a useful discipline, particularly in view of the common discrepancies which intervene between the results of experimental research undertakings and real field practice. As their familiarity with the computer system develops, they can exploit linear programming and simulation packages, a large range of which are available; initially however all this will probably be done 'by hand' relative to local conditions–road networks, weather patterns, markets, transport costs etc. Tables 3 and 4, and Fig. 3 show the range of choice open typically to farmers, and the calculations that are desirable in developing appropriate action lists. As indicated later, the aggregation of resource requirements and outputs from the action lists is a vital base for production, extension services, credit and marketing.

Farm planning with this level of precision was the rare exception during our field studies, confined generally to one or two outstanding 'progressive farmers' in Kenya, or to the best run 'group replanting schemes' in Malaysia. Farm planning teams in the districts worked very hard but necessarily covered a very small fraction of the population, commonly those who were making large loan applications. Some extensionists might argue that the farmers nevertheless had a comparable programme of work 'in their heads', but even so the cost benefits were not calculated–some farmers chose between tea and coffee as cash crops according to which was pleasanter to tend; tea plucking is more congenial than the tedious process of selecting and picking ripe coffee cherry. More importantly, if the farmer's plan, however good, is 'in his head', the possibility of reconciling it with input requirements and market opportunities is limited. This applied particularly where developing technology demands close adherence to a complex chain of activities. High-yield varieties of wheat, maize and rice, for instance, developed during the seventies, have achieved a threefold increase and more in output, but only if critical requirements as to planting dates and fertiliser applications are met.

5 Progress recording and monitoring

The output per hectare currently achieved by small farmers is often more than twice that of large farms (in Brazil a factor of eight, in Colombia a factor of 14).[3] The potential return per unit of land (but not labour) is even higher, provided that the relevant results of research can be communicated, and their field implementation can be supported and monitored. The SCAPA facilities aim at the following approach.

At the completion of the planning phase, the intention is that each farmer has his own agreed, committed plan. Thereafter, at monthly intervals he receives a 'monthly review', a printed 'window' of his action list which:

(a) records the report he made last month
(b) reminds him of his planned activities over the coming two months
(c) requires him to report his success in carrying out his production plan for this month
(d) requests order confirmation of any planned input deliveries for the next month
(e) gives the delivery details (time, place, quantity, cost) for inputs this month.

At the year end this is supplemented by a summary of income and expenditure. The farmer receives his monthly review form and completes and returns the tear-off, OMR readable, report section, which is in due course read by a document reader, the reports being aggregated and analysed for management action.

Table 2 Action list (in addition to normal tapping) for a replanting programme for 3 years before and 7 years after actual planting

Year 1	Year month	Description	Action Number	Must be preceded by	Labour man days	Latest completion date	Credit action	Value of grant or credit and value of goods or sources in kind	Budget cash flow
	End March	*Fell* and cut and load contractors lorries with wood	19	18					+ $1100 (less transport cost at $0•5 per tonne km (say $900)
	April	*Poison* tree stumps as felled	20	11	2				
		Apply for contractor ploughing and make contract	21		-				
	Apr/May	*Collect* and burn small wood in tree rows	22		2				
		Apply for hole inspection for end of May	23		-				
		Contractor carries out 2 discings + harrow rotovate in the interrow spaces	24	20	-	15th May		*	
		Contractor prepares holes (by postholer) between standing tree stumps, at least 60 cm diam. by 45 cm deep	25					*	
		Receive CIRP and GML (10 bags)	26	16	-	1st May		$80 (grant) (in fertiliser)	−$120*
		Apply CIRP and GML broadcast in interrow before rotovation	27	26	8				
		Receive tapioca sticks and cut to 23 cm lengths	28	14					
		Plant tapioca 5 rows between rubber rows using 23 cm cuttings	29	28	8				
	May	*Inspection* of clearance and rubber holes	30	25	-	31st May		$20 (Credit) (in planting materials) $180 (Grant) balance* (in cash)	+$180
		Receive fertilizer for tapioca (8 @ 40 kg bags CCM44, 10 bags @ 40kg bags CCM77)	31	15	-			$786 (Credit) (in fertiliser)	
	Mid May	*Apply* fertilizer to tapioca 8 bags CCM44 week after planting in 15-23 cm rings around plants	32	31	1				
	End May	*Weed* tapioca	33	-	5				

*NB: Some of the $180 may be paid out by *RISDA* to contractors for action 24 and 25. If so then cash balance will be reduced accordingly to $60

Table 3 **Rubber wood and nonrubber enterprises: labour and returns net of labour**

CROP	Labour, man days/ha	Maturity	Direct costs $/ha	Quantity harvested	Unit value	Gross returns, $/ha	Net return, $/ha without labour
Rubber wood	20 (with chain saw)	30 years	100	120-150 t	$10/ton less t'port at 50c per tonne Km	1200-1500	1100-1400
Tapioca	68	12 months	307	18-22 t	$70/ton less t'port at 50c per tonne km	1260-1540	953-1233
Soyabeans	~90	100-101 days	570	800-2000 kg	$1·0/kg	800-2000	230-1430
Groundnuts	~105	110-120 days	1330	ca 3000 kg (in shell)	$0·67 (in shell)	2010	680
Maize	68	65-100 days	840	ca 3100 kg	$0·4	1240	400
Chillies Fresh	70+	5 months	1201	up to 13 t/ha	1·6-2·0$/kg	20,800-26,000	more than 8000
Dry				up to 2·5 t/ha	4-6S kg.	10,000-15,000	
Sweet potato	38	100 days	200	100-140 pik	$7 per pik	700-980	500-780
Ginger	58	7-8 months	1585	6800-8500 kg	$1100 per tonne	7480-9350	5885-7765
Cabbages	101	80 days	1059-1440	150-(300) -520 pik	$35-42 per pik	5250-21, 840	3816-20,781
Chickens (broilers)	~10	60 days	feed $2310-2400	500 broilers	$4·50-5·20	2400-2600	$0-290

Table 4 Outline suitability of possible intercrops according to soil and conditions

Crop	Slope	Drainage	Flooding	Soil depth*	Soil texture	Depth of surface peat	Acid sulphate layer	Soil workability	Acidity†
bananas	0-15°	not critical	can stand temporary flooding	>60 cm	avoid very light soils	unimportant	avoid	<30% stoney	5•5
tapioca	0-6°‡	not critical	none	>60 cm	not sands or heavy clays	<50 cm	not less than 70 cm	<25% stoney	5•0-7•0
soyabeans	0-6°	good	none	>50 cm	not critical	<50 cm		good <5% stoney	(5•0) 5•5+
groundnuts	0-6°	moderate - good	none	>50 cm	not heavy clays	<50 cm	not less than 75 cm	good <5% stoney	5•5-7•0
maize	0-6°‡	good	none	>50 cm	loams to light clays	<50 cm		<25% stoney	(5•5) 6•0-7•0
hill padi	0-6°	less the better	can stand flooding	>50 cm	silts (heavy) - clays	unimportant	unimportant	<25% stoney	5•5-6•5
chillies (fresh) & dry)	0-6°‡	not critical	none	>50 cm	loams to light clays	<50cm	not less than 75 cm	<25% stoney	5•0-6•0
Sweet potato	0-6°	good	none	>60 cm	sandy loam - clay loam (not heavy clay)	<50 cm	not less than 75 cm	<10% stoney	5•5-6•5
Ginger	0-6°‡	good	none	ridges 30 cm High	peats - clays	ridging	avoid	good crumb structure and friability	6•0-6•5
Cabbages	0-6°	good	none	on beds with drains betweem	river alluvium		avoid	very good	5•4-5•9
Tobacco	0-6°	moderate - good	none	>50 cm	clay loam avoid very light soils and heavy clays	<50 cm	avoid	<25% stoney	5•5-6•5
Pineapple	0-6°‡	good	can stand temporary flooding	>25 cm	all except coarse sands and heavy clays	deep peat suitable if well humified and drained	>50 cm	no problem	5•0-6•0

*Soil depth required can be obtained by ridging if natural soil depth is less than stated.
†Lime can be used to bring soil into required range if necessary.
‡Up to 12° with conservation

promotion to headquarters. In fact the field officers we talked to usually enjoyed the challenge and contact in their jobs, but nevertheless felt isolated and faced with a nearly impossible task in trying to communicate to upwards of 1000 farmers, particularly when routine administration – crop returns, various reports, or the paperwork surrounding the 18 stages of each replanting grant for rubber – often took 50% of their time.

The SCAPA facilities are intended to have three main effects. First to structure extensionists' work so that some less demanding functions – random inspection to ensure that progress is being correctly recorded, or that disease control is carried out – can be delegated to less trained staff (the scarcity of trained staff is an endemic problem). For budgetary reasons, the numbers of such untrained staff were sometimes being reduced drastically, even though, as men with practical farming experience, they were often very good at their jobs. If the effectiveness of their participation were to be demonstrable in tangible terms – e.g. by farmers' action lists correctly being followed, or rising crop returns – this phasing out of a potentially valuable service could be re-evaluated. Secondly, the exception reports extracted from the farmers' monthly review forms could point the extensionists at problem areas – this was generally welcomed. Studies by the Institute of Development Studies in Nairobi for instance have shown that over 50% of the wealthy progressive farmers are visited and advised, against 3% of the poorer smallholders, the men with the problems who needed assistance. A similar pattern sometimes occurred in Malaysia. Again understandably, because the extensionist found closer rapport and interest among the progressive farmers. However, the progressives tend to be individualists, whose example is frequently rejected by the average smallholder for a variety of socio-economic reasons. Thus the effectiveness of extension visits in improving average standards was greatly diminished.

Thirdly, the administrative workload should be reduced, since farmers and the computer system that analyses their returns can take on some of the routine reporting functions. This was attractive – both in Kenya and Malaysia successful extension is increasingly engulfed in necessary paperwork, to the extent that RISDA in Malaysia required one clerical worker per extensionist even at local levels.

7 Agricultural input supplies and services

Returns in developing countries from the use of agricultural inputs tend to be much higher than from the use of equivalent quantities on the relatively jaded soils of developed countries, soils which are, for example, usually further up the slope of diminishing returns per unit of fertiliser. However, the logistics of input supply – forecasting, distribution and payment – are extremely difficult for large numbers of small farms.[5]

The relevant SCAPA facilities provide for:

(a) long-term forecasts, based on the aggregation of the precise requirements embedded in the action lists for each farm

(b) short-term confirmation by the farmers that the embedded orders still apply
(c) scheduling (from manually input vehicle schedules) of delivery arrangements
(d) recording and accounting for receipt.

In Kenya and Malaysia, satisfaction of farmers' input requirements was given high priority and was closely monitored. Nevertheless, problems occurred. Most notably, arabica coffee requires spraying at 3-weekly intervals during the rains with a very costly fungicide, benlate, to prevent coffee-berry disease. Over 50% of farmers had been unable, through shortages, to apply the full series of sprays, usually achieving less than half the cycle. The resultant disease cost was, for a single district, a loss of at least £1 million in potential production. The problem stems from the original forecasts, which are derived from subjective judgements for each factory (factories store and supply inputs to the farmers) of the area under cultivation, and the historical pattern of demand by the farmers in the area.

The key variables – the farmers' intentions for the future – were necessarily unknown, while the historical demand pattern was suspect, since in previous years local rumours of shortage had caused spurts in buying and hoarding and hence artificial shortages. It seemed therefore that the precise knowledge of each farmer's intention as recorded in his action list, and monitoring its satisfaction, could be highly effective. So also could the normal procedures of computerised stock control – monitoring stock-outs and controlling re-order levels and quantities. These last need to be varied by season and area; here the manual system, which prescribed standard levels, fell down and was not used.

8 Credit assessment and control

Developing agricultural technology frequently demands credit for seeds, fertilisers, equipment and buildings, but the cost of administering large numbers of small loans to the small farm sector is high, defaults are often excessive, and it has been estimated that less than 50% of the funds provided are spent on the purpose intended.[6,7]

The SCAPA proposals involve assessment of a farmer's creditworthiness and requirements by reference to his farm profile; the credit is subsequently meted out in relation to his completion (reported and inspected) of specific production activities in his action list.

In Kenya, the general problems described above had led to a tight and effective system, whereby farmers could acquire credit for input purchases up to two-thirds of their output, averaged over the preceding three years. There was in fact no bar to credit for the successful farmers, but equally no route in for the poor farmer who wanted to enter production, but had no previous record of production achievement to justify credit. SCAPA proposals would assist in the latter case, because the door could be opened to poorer farmers with reduced risk to the lender, since expenditure and achievement are monitored.

RISDA in Malaysia is funded from a levy on rubber sales, and allocates part of this levy as credit for replanting or new-planting grants. A farmer receives his grant in stage payments, each stage in the planting cycle being inspected. Here we concluded that the SCAPA procedures could be used in place of the existing effective but lengthy manual procedures—in another country, a backlog of two to three years is common—and also to control other loans, where formal control is less well defined. This summary sounds somewhat clinical; the problem of credit release and control is central to development, as is the release of funds, which can be penned up behind barriers of information and decision.

9 Marketing

SCAPA facilities provide for each farm the recording on OMR forms and subsequent accounting for produce sales in terms of quantity and quality.

Manual procedures were followed in Kenya. Complicated calculations were required to deduct credits, apportion a variety of charges and quality bonuses averaged for each factory, for large numbers of transactions—reflecting the activities of 40,000 growers over three months. Carried out manually, the system imposed a clerical nightmare similar to a prolonged year-end bank audit, with teams of clerks, drawn in from the factories, computing and checking alternately, working unpaid overtime when reconciliations failed, or when mechanical hand calculators went wrong. (The chief accountant strongly advocated computerisation). The best organised districts achieved this reconciliation and disbursement to farmers in about a fortnight, a remarkable achievement, but in other cases the delay in payment reached three months or more. Significantly, all the data needed to produce farm accounts, enabling a valuable analysis of profitability for each farm, are recorded. However, under a manual system, restructuring this data to provide such analyses (a key to effective farm management)[8] is in effect impossible because of the volume involved. The relevance of such analyses in feedback to management, research, extension and market forecasting for processing, storage and transport planning has already been mentioned.

In Malaysia marketing is not under the integrated control which is essential in Kenya for coffee, but most sales of rubber occur at RISDA's group processing centres, and are recorded on an individual-farm basis, so that similar analyses and feedback are feasible. The broader planning aspects of marketing, as opposed to straight sales recording, were also relevant because RISDA places great emphasis on mixed cropping and diversification, which can only succeed if production output and market demand are in reasonable balance. Regular feedback on production activities during the growing season can help to achieve such a balance. Without such reports the market-development staff were in a very exposed situation.

Imaginative attempts to introduce new and potentially valuable crops could be hamstrung because either shortfalls or surpluses could deflate confidence among both producers and buyers.

10 Project management

'The way ahead will also depend on bold innovative projects which will depend on creating and quickly responding to suitable opportunities as and when they appear'.[10] Local initiatives on these lines are in particular a strong feature of RISDA's policy, to exploit the potential challenge and attrractiveness of rural life.

A wide range of group projects cover, for example, crop diversification, animal husbandry, fish farming, irrigation, land development, local shops, rural industries and community welfare. There is, however, an inevitable risk that well meant but inadequately researched or managed local schemes could cause severe hardship. SCAPA facilities might provide a framework under which such schemes are evaluated, supported and monitored.

11 Computing aspects

The main requirements are for high-volume/low-cost throughput, ease of operation, a straightforward and easily understood user image, capacity to handle different natural languages, and data protection to ensure privacy for sensitive personal data. In addition, database management, a variety of applications software, mathematical and statistical packages and high-level language compilers are needed—for example, to support the planning activity, where a certain amount of simulation and modelling are involved. Remote job entry is an immediate requirement, with full communications use as an extension. The target machine is a 2903, operating primarily in batch mode, located at District or State headquarters.

Details of the procedures and field implementation costs are given in the project working papers. The cost-benefit ratio is about 1:7, based on assumptions which seem typical but require evaluation in the light of particular proposed applications.

12 Management aspects and conclusion

Computational facilities have been widely exploited for agricultural planning for developing countries[9] covering systems simulation, mathematical modelling and econometric models. However, the facilities have only been invoked in a partial, piecemeal fashion in the areas of production, supply, marketing and management. This is probably partly because computers are not seen as 'intermediate' technology, but as tools of affluence, and are therefore held to be inappropriate. Much as tractors are often disfavoured because each tractor may supplant 40 labourers, computers are feared to displace clerical staff. However, if the very convincing argument is accepted that the small-farm sector holds the key to increased world agricultural production—an incremental rate of 5% per year is quite feasible[3]—then the logistical and management support of the small farmer involves, as indicated earlier, an information-handling requirement which clerical methods cannot meet. Viable advice has to be propagated and timely supplies assured, and, in particular, feedback on performance and problems is needed on a large scale. Managing these activities on a large farm or estate basis is difficult, but doing so for a large population of independent smallholders is far more complex.

The SCAPA facilities provide some answers in a variety of contexts. Specific areas like the planning and supply of inputs, the control of credit and sales recording and analysis clearly present an immediate requirement. These are areas where current manual procedures are a constraint. Over a million transactions for instance are already recorded manually for one district, Murang'a, per year; if held on computer files this data could be readily abstracted and analysed to give control and feedback from farm level upwards. In other areas the system facilities have greater potential benefit, notably in the communication and acceptance of new techniques and in monitoring their use, involving direct interaction between farmers and the systems. These facilities require evaluation for acceptability and cost benefit in pilot studies during the next phase of implementation.

References

1 GARBOUCHEV, I.P , and SADOVSKI, N.: 'An application of a soil information system'. Poushkarov Institute of Soil Science, Sofia, 1975

2 GATITH, G.M. KABAARA, A.M., and MICHORI, K.P.: 'Standard recommendations for coffee fertilisers, 1976. Coffee Research Foundation, Ruiru, Kenya

3 LEAN, G.: 'Rich world poor world' (Allen & Unwin, 1978), p. 43

4 CHAMBERS, R., and BELLSHAW, D.: 'Managing rural development'. IDS Discussion Paper 15, Section 3.13, University of Sussex, 1973

5 ELLIS, T.F.: 'Report for western region coffee sector, 1972'. Ministry of Agriculture, Uganda.

6 BESSELL, J.E., and ILES, J.M.: 'Farmer operating efficiency and credit worthiness'. Nottingham University, 1976

7 HUNT, D.M.: 'Credit for agricultural development' (Third World Publications, 1974) p. 135

8 UPTON, M.: 'Farm management in Africa' (Oxford University Press, 1973), p.8

9 BIGGS, S.D., SWONG CHONG YUAN and LANGHAM, M.R.: 'Agricultural sector analysis' (Singapore University Press, 1977)

10 MINISTRY OF OVERSEAS DEVELOPMENT:'More help for the poorest', Cmnd. 6270, HMSO, p.20

11 MOHAMMED NOR and CHONG KWONG YUAN: 'Proceedings of RRIM Planters Conference 1973'. RISDA, Kuala Lumpur, p. 377

12 BENOR, D. and HARRISON, J.Q.: 'Agricultural extension'. World Bank, 1977

13 UMA LELE: 'Design for rural development' (John's Hopkins University Press. 1977), pp. 70 and 102

Software and algorithms for the Distributed-Array Processors

R.W. Gostick
Senior Consultant, DAP Marketing Unit, ICL, London

Abstract

The ICL distributed-array processor, (DAP) is a radical departure from conventional serial computer architectures, providing new opportunites for system and language design. This paper describes briefly the interaction of the DAP with a standard 2900 series computer, and indicates the ways in which Fortran has been enhanced to cater for parallel-processing capabilities. Examples of algorithms using the new system in widely differing applications are given.

1 Introduction

The technology that has made the microprocessor and pocket calculator possible also makes it feasible to consider building powerful computers out of many identical elementary processors. The distributed-array processor[1,2] (DAP) is made by embedding a matrix of elementary processing elements within the store of a standard computer and is the most advanced of these new computer systems. As a direct result of their nature processor arrays have the capability of simultaneously producing many identical results and sometimes, when compared with single processor systems, require different techniques for their exploitation. A new generation of programming tools needs to be developed to allow processor arrays to be programmed and this paper discusses DAP-Fortran, the language proposed for scientific and technical programming for the DAP.[3,4] The paper deals with only a few of the novel features of DAP-Fortran. Its aim is to give the general flavour of the language, and an indication of methods of 'parallel thinking'.

There are many high-level languages and equally many fiercely held opinions as to their relative merits. It is only a very brave and foolhardy worker who dares to suggest a new language and the suggestion will only survive if it contains multiple benefits for the end users. Languages like Fortran and Algol were invented in the days when computers were only capable of obeying a single stream of elementary operations (arithmetic, boolean or control) and so not surprisingly contain very few defined operations of greater complexity than these simple operations.

As processor arrays by their very nature perform many repetitions of the same operations it is desirable that a programming language for processor arrays contain facilities for specifying easily such automatic repetitions. The fundamental question is whether to try to specify a completely new language or try to modify and extend an existing language. A quick investigation of the potential users of processor arrays shows that the community is 99•9% of the opinion that 'Fortran is the only natural language for programming computers'. It should be emphasised that this is the view of those workers who actually use computers for very large scale computing. The view of the language theoreticians – who apparently never seem to do very large computations – is 99•9% 'Fortan is awful, language X is much better'. Unfortunately, there is no agreement as to X, the actual name changing about as rapidly as the fashionable height of a hem line.

Putting the users' opinions first, a dialect of Fortran has been proposed which it is believed contains sufficient new facilities to allow greater ease of programming while leaving freedom for the compiler writer to alter the implementation to suit the characteristics of processor arrays.

2 Basic concepts

2.1 *Hardware structure*

Although the DAP shares the name 'array processor' with many other current computers, there are many uses of the term. In the general sense an array processor may be considered simply as a machine that processes arrays. In this sense, of course, any modern computer such as the 2980 may be considered to be an array-processing machine. A more widely accepted definition is that the machine should have special hardware facilities for array processing, such as a vector instruction set or vector registers. Indeed the terms *vector* and *array* are freely interchanged. The DAP is in many ways a 'true' array processor, since not only does it process arrays but it is actually a physical array of processors.

The basic hardware of the DAP consists of an array of identical processors, known as processing elements (PEs). A pilot machine with 32 x 32 (ie. 1024) processors has been operating in ICL's Research and Advanced Development Centre since 1976. The first production model with 64 x 64 (4096) processors will be delivered to Queen Mary College, London University, during 1979. Larger arrays, particularly 128 x 128 (16384 processors), are already being considered. In this paper N refers to the basic dimension of an N x N DAP. Each processing element is in fact much simpler than either the chip in a pocket calculator or a typical modern microprocessor. As with most microprocessors, the PE has a store (a single 4kbit store chip) and a processing unit, but the processing unit has very few hardware facilities. It may be considered as having merely three registers and a set of simple instructions which operate on the three registers and the store. (Fig. 1) Of particular importance is the A-register, known as the Activity register. Certain instructions may be made conditional on the setting of this register in each processor, hence providing a measure of local autonomy to the processors in the array. By using this facility, through the use of the DAP-Fortran language, most of the conditional operations common in many physical processes may be handled naturally and efficiently, with no expensive tests and jumps within the code. There is no equivalent to the normal functions of instruction fetching and decoding, address calculations, interrupt handling etc. All these are handled by a central unit called the master control unit (MCU).

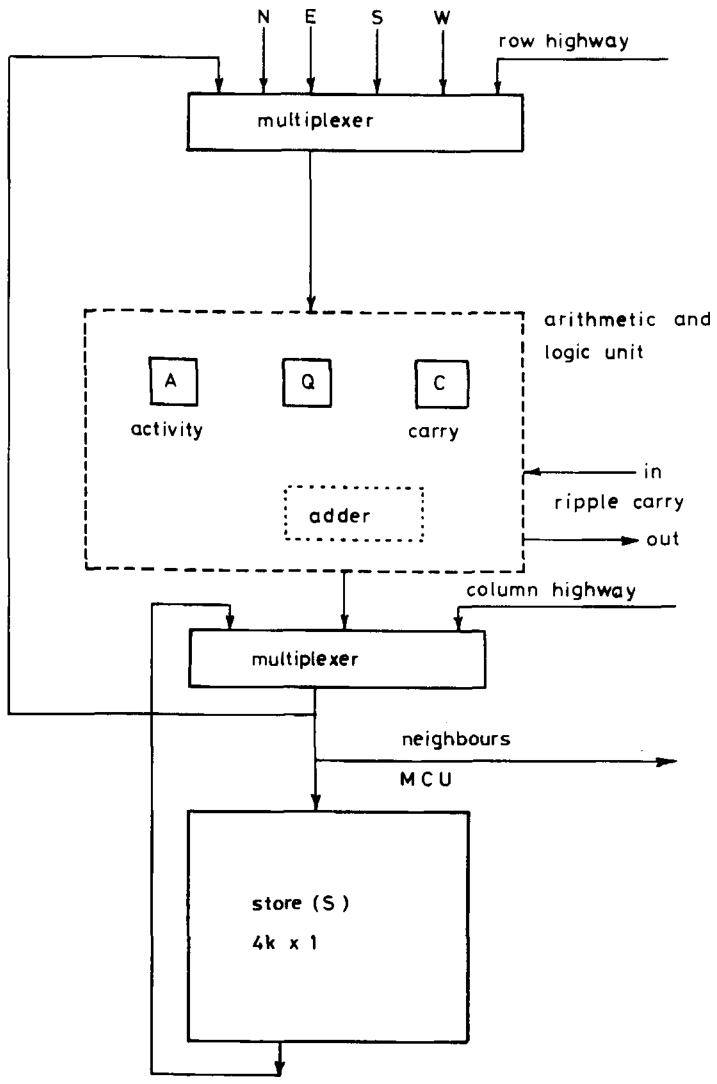


Fig. 1 Simplified processing element

The master control unit has two significant roles, one as a co-ordinator and one as a processor performing certain centralised functions. In its primary role as a co-ordinator, the MCU has the tasks of instruction fetch, decoding and modifying via MCU registers. In this respect it acts much like a normal central processor, but in one sense the similarity ends there, since most of the instructions decoded by the MCU are not in fact 'obeyed' by the MCU itself. The MCU simply sends, or broadcasts, the decoded instructions to the array of processors which then obey the instruction simultaneously, acting on their locally held data. Thus the DAP may be regarded as a single-instruction-stream multiple-data-stream machine

(SIMD), since one instruction is obeyed in parallel by many independant processors.

The secondary, but also very important, function of the MCU is as a true centralised processor. While the array of processors works efficiently on large sets of data in arrays, there are still some simple functions that may only be performed in a scalar fashion. These functions, such as the control of DO loop variables in Fortran, are performed within the MCU itself. The MCU also has powerful facilities for global actions on the data held in the PE store, such as finding a maximum value.

The term 'distributed' in distributed-array processor refers to the unique way in which the DAP interfaces to a supporting, or host, 2900 computer. Most fast-array or vector processors are attached to a host machine as a back-end processor As such they are connected to the host by a channel, in a similar way to a disc unit. The DAP is built into a system by distributing its processing elements within a store module of the 2900, allocating one processor to each semiconductor store chip in the module. This means that once the data has been loaded into the store, using all the conventional 2900 facilities, it may be processed either by the 2900 or by the DAP. There is no overhead associated with 'loading' the DAP as there is with a back-end processor. Fig. 2*a* shows the DAP as part of a typical 2900 configuration.

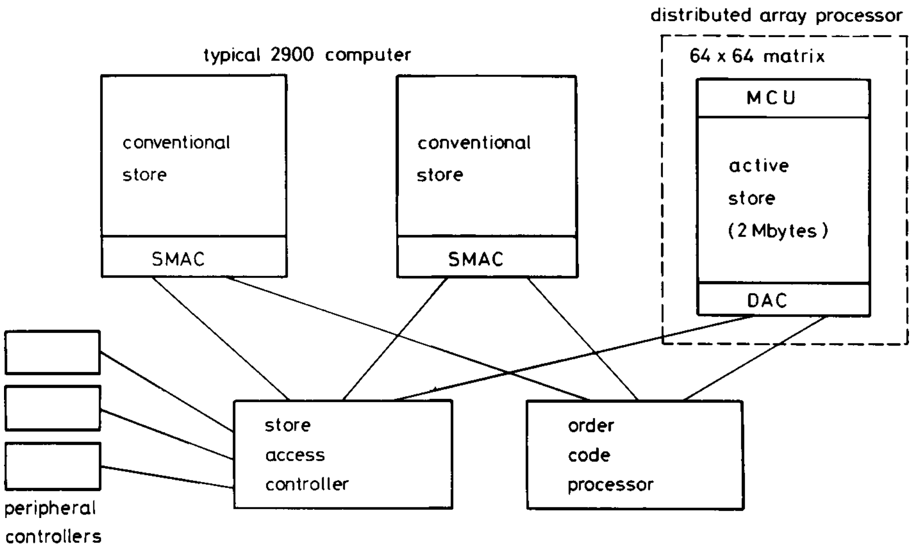


Fig. 2*a* DAP as part of an ICL 2900 'host' computer

2.2 *Software structure*

Since the operating system can regard the DAP as normal main store, any application which runs on the 2900, using any standard facilities, will run on the 2900/DAP configuration. Such an application would not, however, notice the existence of the DAP and hence would not benefit. To make use of the DAP suitable parts of the selection in that dimension. Thus A(3,,2) selects the third row of the second matrix of the set, and A(,3,2) selects the third column, while A(,,2) selects the entire second matrix. Note that a subscript of the form B(,3) is syntactically correct either to select a column from a matrix or a vector from a vector set—a very desirable consequence, since both items are logically the same in the programmer's view. The previous examples were, of course, simple cases where each array element only interacted with other array elements at the same position (ie. with the same Fortran subscript). There are many cases where this simple relationship will not be true, and DAP-Fortran has new indexing techniques to handle them.

Consider, for example, a simple physics problem, which is to solve Laplace's equation on an N x N element grid. One standard technique uses an iterative procedure, where at each iteration the value at any grid point is replaced by the average of the four neighbouring values. This could be expressed in Fortran by:

```
      DO 1 I = 1,N
      DO 1 J = 1,N
    1 X(I,J) = (Y(I + 1, J) + Y(I - 1,J) + Y(I,J + 1) + Y(I,J - 1))/4·0
```

Note, however, that this would fail when I or J became 1 or N, at the edges of the array, since the index calculations would try to access over the edges. The user has to be careful about the range of the index variables in this situation, and will normally have to write special code to handle the edge effects.

In DAP-Fortran the problem is expressed using a concise notation where the explicit index variables are omitted giving:

```
    X = (Y(+,) + Y(-,) + Y(,+) + Y(,-))/4·0
```

The form Y(+,) is termed shift indexing and is used to access the neighbours of each point in the array. Longer shifts, corresponding to the Fortran expressions of the form Y(I + 4, J + 3), are performed by explicit functions which are considered below.

As with the Fortran solution the DAP-Fortran code for the mesh problem has accessed the complete array, but unlike the Fortran case the conditions at the edges are defined explicitly by the programmer. The programmer is able to tell the DAP, by means of a GEOMETRY statement, how he wishes the edges to be handled. In the most usual circumstances one of two edge conditions will hold. The first condition, known as plane boundaries, assumes that outside the bounds of the array is a world of zero values, so that accesses outside the array produce zero results. The second condition, known as cyclic boundaries, assumes that an edge of the array is connected directly to the opposite edge. By combining these two geometries in the north-south and east-west directions the DAP can be made to resemble a plane, a cylinder in either of two orientations, or a torus (doughnut). Other boundary conditions, such as continuity of the first derivative or reflective boundaries can be handled using some of the masking techniques shown below.

The longer-distance shifts are handled by function calls, with a simple naming convention. As an example, the function SHWC performs a SHift to the West with Cyclic geometry. For shifting in two dimensions the functions may be combined, and so, in plane geometry,

```
    M1 = SHNP(SHEP(M,3),4)
```

performs first a shift of array M by three positions east followed by four positions north, with zero fill-in at the edges. This is the equivalent of the Fortran code:

```
    DO 1 I = 1,N -4
    DO 1 J = 4, N
1   M1(I,J) = M(I + 4, J -3)
```

Note that the Fortran equivalent would be more complex if a cyclic shift were required.

Many common algorithms, such as the fast Fourier transform, make considerable use of these shift routines, although it is usually found that only a small proportion of the overall time is actually taken performing the shifts on the DAP.

3.3 Conditional operations: masks, logical arrays

All the examples above have assumed that the same activity is being carried out at every point in the array. For many practical applications this will not be the case, and Fortran users have to make use of the ubiquitous IF statement. Consider again the quadratic equation given previously,

$$ax^2 + bx + c = 0$$

which has to be solved for a set of values of the coefficients *a, b* and *c*. If only the largest real root, if any, is required at each set of coefficients, the programmer first has to test whether the discriminant $b^2 - 4ac$ is positive before performing the square root operation. This will typically be written in Fortran as:

```
    DO 1 I = 1,N
    DO 1 J = 1,N
    X(I,J) = -999
    DISC = B(I,J)**2 - 4·0*A(I,J)*C(I,J)
    IF(DISC.GE.0.0)X(I,J) = (-B(I,J) + SQRT(DISC))/2·0*A(I,J))
    1 CONTINUE
```

The line X(I,J) = -999 serves as an indicator for unset, or imaginary, results. Although this is a simple piece of code it will work only slowly on a conventional or vector computer, since each time the IF statement is reached the computer has to test and branch to alternate sets of code. This is particularly inefficient on a pipelined machine, which works best when there are no tests in the code.

On the DAP the hardware works in parallel on all elements of the array, except where it is turned off by the programmer effectively setting the activity register, thus permitting simple implementation of conditional problems using a technique known as mask indexing. We can write the above problem in DAP-Fortran as:

```
X = -999
DISC = B**2 - 4·0*A*C
X(DISC.GE.0.0) = (-B + SQRT(DISC))/(2·0*A)
```

where all variables are real matrices.

The expression DISC.GE.0.0. is a logical or boolean expression which returns a matrix of values which are TRUE only where the corresponding value of the discriminant is nonnegative. This matrix of boolean values is then used as a mask in the subsequent assignment, so that only those values of X where the appropriate value it TRUE will be changed. This provides a very natural way of writing complex conditional programs while using the parallel nature of the DAP.

One extension of the use of the masking technique is the ability to create and manipulate the logical masks. On the DAP logical variables may be stored as one bit per variable, which is of course the minimum possible storage for a logical item. We can therefore create and store logical arrays using a small percentage of the DAP store. The DAP hardware can also manipulate logical items efficiently, for instance a logical AND of two logical N x N matrices can be performed in a time of the order of 1 μs.

In the above example we can store the positions of the real roots by declaring a logical matrix REALROOTS and using the basic Fortran type logical expressions, giving:

```
  REALROOTS = DISC.GE.0.0
X(REALROOTS) =(-B + SQRT(DISC))/(2·0*A)
```

The use of logical variables is further extended by many of the basic functions provided in DAP-Fortran. As well as the standard mathematical functions such as SQRT, which have been extended to use array arguments, DAP-Fortran provides a wide range of functions for creating arrays from scalars, extracting scalars from arrays and performing global tests on arrays.

The MAXV function finds the maximum value from a vector or matrix, using an efficient parallel algorithm which will be shown later. This function may take either one array argument, containing the values to be compared, or a second argument which may be used as a mask. Thus to find the maximum root of the earlier equation, we can simply write:

```
MAXIMUMROOT = MAXV(X,REALROOTS)
```

Another function which is used with logical variables is the ANY function. This produces a single TRUE value if and only if at least one of the values of its array argument is TRUE, corresponding to the logical OR of all the values. This could be used in the example to avoid performing the square root calculation if there are in fact no real roots. We can write:

```
IF(ANY(REALROOTS)) X(REALROOTS) = (-B + SQRT(DISC))/(2·0*A)
```

Note that this IF statement is used on a global basis, whereas in the Fortran case it is used element-by-element.

The standard functions may also be used to set up logical arrays. The functions ROWS and COLS return logical matrices with a set of rows or columns set TRUE. Thus to declare a matrix indicating just the interior of a N x N array we can write:

```
    INTERIOR = ROWS(2,N-1). AND . COLS(2,N-1)
```

This type of construction is used in many processes where different conditions apply inside and outside the boundary of a region. Such a problem can be expressed as:

```
      X(INTERIOR) = interior function
X(.NOT.INTERIOR) = exterior function
```

This has naturally been only a brief overview of DAP-Fortran, since as with any language the full power can only be appreciated by actual use. In the final section of this paper we shall consider some actual examples of DAP-Fortran, but before that there is one important aspect to be considered, that of diagnostics.

3.4 Diagnostics

The 2900 series uses a language-independant diagnostic system known as the object program error handler (OPEH),[5] which provides diagnostic reports using the names and formats actually used by the programmer in his high-level language program. A typical report is shown in Fig. 3, where the user has asked for local scalar values to be printed out for each active routine in the calling sequence. Various options exist for the level of information provided and the types of errors giving rise to reports. The important point for the user is that there is no run time overhead when there are no errors, since the diagnostic information merely resides in the virtual store until needed. It is a relatively simple task to interface DAP-Fortran to this system, requiring only that the compiler produces diagnostic records along with the DAP code.

With an array processor, such as the DAP, new diagnostic facilities are often required to allow for errors occurring in several array elements at one time, or indeed in array elements where the results would not be required, for instance in conditional expressions. In DAP-Fortran the programmer may allocate two vectors and two matrices for use in diagnostic control. One array of each mode controls the reporting of errors, while the other is used to indicate where errors have occurred. By using these arrays in conjunction with error-classification specifications the user can either control all diagnosis and recovery within the program or leave the total control to the system. A typical case where the user might require total control is where the nature of his problem is such that at some stage overflow will occur. When this happens the values in the problem are rescaled before continuing. Using the system, the user may check for overflow at any stage by a statement of the form,

```
    IF(ANY(OVERFLOW))CALL RESCALE
```

where the logical matrix OVERFLOW will have been set if errors have occurred in any array element.

```
OPEH REPORT IDENTIFY 1
OPEH MK 203 ON 1977/10/20 AT 23:34:55

INTERRUPT ERROR: -500
DESCRIPTION: ZERO DIVIDE

PROGRAM AT LINE:  533(OFFSET:416)
IN PROCEDURE:FOURTL
OF MODULE:FOURTL


SUMMARY OF ROUTE LEADING TO THE ERROR (REVERSE ORDER)

FORTRAN SUBPROGRAM FOURTL(MODULE FOURTL) AT LINE   533
FORTRAN SUBPROGRAM FOR12S(MODULE FOR12S) AT LINE   421
FORTRAN MAIN PROGRAM FFT(MODULE FFT) AT LINE        45

END OF ROUTE SUMMARY

REPORT OF CURRENT STATE OF PROGRAM

FORTRAN SUBPROGRAM FOURTL(MODULE FOURTL) AT LINE   533
ICENT   =  0                          IDIM   =  1
ISIGN   =  1                          ISYM   =  0
N       =  0                          NCURR  =  0
NFACT   =  0                          NFCNT  =  0
NPREV   =  0                          NREM   =  1
NWORK   =  97
FORTRAN SUBPROGRAM FOR12S(MODULE FOR12S) AT LINE   421
I       =  1                          ISGN   =  1
K       =  0                          N      =  32
NSTRNS  =  1                          NTYPE  =  0
FORTRAN MAIN PROGRAM FFT(MODULE FFT) AT LINE        45
A       =  0.0                        AA     =  3.535534

IFORM   =  0
JDIM    =  1
NDIM    =  1
NFSYM   =  0
NTOT    =  0

J       =  1
NCLBT   =  0
N1      =  33

CN      =  (0.0, 0.0)
```

Fig. 3 Typical 2900 high level diagnostic report

4 Programming Examples

The four examples below show differing aspects of the DAP in use: a typical arithmetic operation, a 'low-level' operation, a nonumeric application and an algorithm for sorting. The Appendix gives the methods used in the algorithms for computing standard numerical functions such as square root and logarithm.

4.1 Matrix multiplication

This is an important and time-consuming routine in many applications.

To multiply two N x N matrices, most programmers use a simple translation of the mathematical formula,

$$C_{ij} = \Sigma_k A_{ik} B_{kj}$$

which becomes in Fortran:

```
      DO 1 I = 1,N
      DO 1 J = 1,N
      DO 1 K = 1,N
1     C(I,J) = C(I,J) + A(I,K)*B(K,J)
```

This gives the standard 'inner-product' technique, where at each step a row of A and a column of B are multiplied together element by element and summed into the appropriate element of C. We could perform this process in DAP-Fortran by using the SUM function, giving:

```
      DO 1 I = 1,N
      DO 1 J = 1,N
1     C(I,J) =  SUM(A(I,)*B(,J))
```

This is still not a highly parallel solution, using at most an N element vector multiplication. A small amount of lateral thinking is required. If we look at the original Fortran code we can see that it will work with the DO loops in any order. Let us move the inside K loop to the outside. Now we have:

```
      DO 1 K = 1,N
      DO 1 I = 1,N
      DO 1 J = 1,N
1     (C(I,J) = C(I,J) + A(I,K)*B(K,J)
```

We can see that in the inner loops we are keeping K constant for all I and J. Consider the term A(I,K)*B(K,J) for all I and J. A(I,K) for all I gives the Kth column of A, and similarly B(K,J) gives the Kth row of B. We are then multiplying *every* element in a column of A by *every* element in a row of B. This is not a simple vector multiplication, since we are not just multiplying corresponding elements, as we did for the original matrix multiplication. It is, in fact, an outer product of two vectors, and is identical to an element-by-element multiplication of two matrices formed from the row and column, respectively. In DAP-Fortran such matrices can be formed using the standard MATR and MATC functions. The complete problem can now be coded as:

```
        C = 0·0/
        DO 1      K = 1,N/
1       C = C + MATC(A(,K))*MATR(B(K,))
```

which uses the full N x N parallelism of the DAP.

4.2 *Finding the maximum element of an array*

As noted earlier, logical variables in DAP-Fortran are one bit in length, and thus provide a powerful technique for bit manipulation of numbers. This ability is used here when finding maximum elements of vectors and matrices. This is of particular importance in matrix algebra for determining pivot elements of matrices.

The traditional way to find the maximum number in a list is to take the first value, compare it to successive elements in the list until a larger element is found. A simple analogy for the DAP algorithm is that of wanting to find out who in a room full of people has the most money in his pocket. Assume two limitations - we can only ask simple yes/no questions, and nobody has as much as say £20. We could start by asking all those people with more than £10 to indicate. If one or more people do indicate, tell the rest to leave the room. Now ask who has more than £15, and repeat the process. If at any stage nobody has more than the currently asked value, say £15, ignore that value and proceed to the next lower level, ie. £12.50. Using this 'binary-chop' method we shall eventually end up with the largest value (held by one or more people).

In computers we are dealing with binary words typically of 32 bits, and an equivalent of the binary-chop procedure is simply to move down the bits of a word from the most significant bit.

In DAP-Fortran we can equivalence a set of 32 logical matrices onto successive bits of a matrix of 32-bit real or integer words (see Section 3.2). For both real and integer formats the first bit is effectively the sign bit, with successive bits in descending order of significance. The DAP-Fortran code for finding the maximum is shown below. For simplicity only the case where some or all of the numbers are positive is considered. SIGNS, MAXP and MAXIMUM POSITION are all logical matrices; BITS is a set of 32 logical matrices; NUMBERS is a real or integer matrix; all other variables are scalars.

```
      EQUIVALENCE (NUMBERS, BITS (,,1))
C     Successive logical matrices of the set BITS map onto successive bits of the
C     word array NUMBERS
      IF (ALL(SIGNS)) ERROR 100
C     Only deal with one or more positive numbers, ie. not all negative numbers
      MAXP = .TRUE.
C     MAXP will hold the position of the current maximum value candidates
      DO 1 I = 2,32
C     At each stage only alter MAXP if one of the current candidates has the next
C     significant bit set
1     IF (ANY(BITS(,,I).AND.MAXP))MAXP = MAXP.AND.BITS(,,I)
C     Extract the maximum value (there may be more than one identical value)
      MAXIMUM POSITION = MAXP
      MAXIMUM VALUE = NUMBERS(FRST(MAXP))
```

The ERROR statement is used by the programmer to signal error codes to the high-level diagnostic system or to a specially written diagnostic routine.

The code can be used within a function to return either the maximum value, which is a scalar, or the maximum element position which is a logical matrix. Note that this code is given by way of illustration only; DAP-Fortran provides this facility in an intrinsic function.

4.3 Route-finding algorithm

In such tasks as network analysis, traffic planning and even the travelling-salesman problem, one prime requirement is to find the shortest route, if any, between two points in a network. Consider the network shown in Fig. 4. Conventionally this may be represented by giving lists showing each node that may be reached directly from a given node. On the DAP such a network may be represented more simply and directly using a logical matrix. This has the advantages on the DAP both of compact storage and rapid manipulation. Fig. 5*a* shows one corner of the connectivity matrix representing the network of Fig. 4, where a 1 indicates that a connection is possible between the nodes in the row and column. The algorithm to trace through the network is basically very simple. At each step we will be at a certain node in the network. The nodes that may be reached on the next step are found in the column of the matrix for the current node. This new set of nodes may be used to repeat the cycle by taking all columns representing each new node. Since there will in general be several such columns, the algorithm considers merely the resultant nodes by combining all columns (a logical OR function). This results in one new column, and the cycle is repeated.

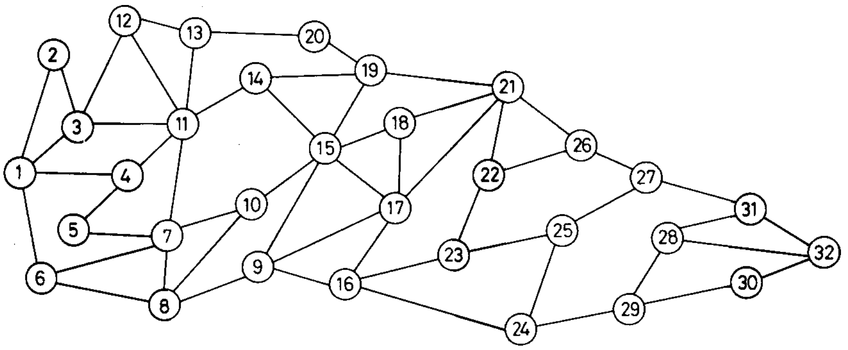


Fig. 4

At each stage there are three further administrative tasks. First, of course, the nodes already encountered must be eliminated to avoid looping. Secondly, we must store the nodes that have been reached. Finally we must check whether we have indeed reached our destination.

The algorithm is coded very simply in DAP Fortran for a network of up to N nodes; for more than N the network must be partitioned.

The logical matrix NETWORK contains the logical connectivity matrix at the start of the test. STEP is a logical vector which contains the nodes reached at the next step. STORE is a set of logical vectors which is used to save the path currently traced. STARTNODE and ENDNODE are integers which specify the starting and ending nodes.

	1	2	3	4	5	6	. . . . . .
1		1	1	1		1	
2	1		1				
3	1						
4	1				1		
5				1			
6	1						
•							
•							

Fig. 5a Section of matrix representation of Fig. 4

The DAP-Fortran code is thus:

```
      I = 1
      NETWORK(STARTNODE,) = .FALSE.
      STEP = NETWORK( , STARTNODE)
C     Extract the nodes that can be reached from the starting node
1     IF(STEP(ENDNODE)) GO TO 10
C     Check whether we have reached the end
      NETWORK = NETWORK.AND..NOT.MATC(STEP)
C     Eliminate all nodes that have been selected by switching off in those rows
C     corresponding to nodes already reached.
C     MATC forms a matrix with equal columns
      STORE( ,I) = STEP
C     Store current step
      I = I + 1
      STEP = ORCOLS(NETWORK.AND.MATR(STEP))
C     Create a set of nodes that can be reached in the next step by combining the
C     vector of current nodes with the network and then combine the resulting
C     columns with the logical OR function
```

```
      GO TO 1
C     Continue on next step
10    CONTINUE
```

Although this sounds complex, an example based on the network of Fig. 4 indicates the working of the algorithm. Fig. 5*b* shows the values of the logical vector STEP at each iteration, from node 11 to node 23. At each iteration STEP contains all those nodes that can be reached from any of the currently reached nodes. Thus the first contains all nodes that can be reached directly from the start (ie. 3, 4, 7, 12, 13, 14), the second contains all nodes not already reached which can be reached from any of the nodes in the first vector etc. The fifth vector has the end node (23) set, and the process ends.

It is obvious that the tree constructed in Fig. 5*b* has not provided a unique route. If the process is repeated in the opposite direction, a similar tree will be produced. These two trees are then intersected (a logical AND process) to eliminate all nodes not on a route. If more than one node exists in any row, a further sweep through selecting the first node in a row will produce a unique route.

	Element Number																											
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	...	32
Step 1			1	1			1					1	1	1														
2	1	1			1	1		1		1					1				1	1								
3									1								1	1										
4																1					1							
5																						1	1			1		

Fig. 5b STEP vector showing possible nodes along route

Notice that this algorithm uses no arithmetic and no indirect addressing. The majority of the operations are simple logical expressions, which are handled efficiently by the low-level DAP hardware.

One further point of interest is that during the operation on a typical network the majority of the processing elements will be inactive. This should not, however, be taken to mean that the DAP is consequently working inefficiently, since it is the execution time that is of prime importance, not absolute hardware utilisation.

4.4 Sorting

Sorting on DAP may be performed by many techniques, and the one shown below is one of the simplest, based on a Bubble sort technique. As with a Bubble sort on a conventional computer, the technique relies on comparing neighbouring values in an array. For *n* items, where *n* is less than N^2 on an N x N DAP this technique will take at most *n* steps, compared with n^2 steps on a conventional machine. The algorithm treats the DAP as a linear string of N^2 items, using the appropriate DAP geometry, rather than an N x N array. Faster algorithms, based for instance on the Batcher sort,[6] may easily be implemented, making use of the 2-dimensional connectivity of the DAP.

It should be realised that, for many applications on DAP, sorting is unnecessary or inappropriate, since the associative nature of the DAP permits rapid selection of particular items in a data set.

The program is as follows:

```
     LOGICAL  MASK( , ), CHANGE( , )
     REAL     TEMP( , ), VALUE( , )

     MASK = ALTC(1).LNEQ.ALTR(1)

     MASK = ALTR(1)

C    Set up mask for alternate odd-even pairs of values
C    ALTR produces alternate rows TRUE

1    CHANGE = VALUE.LT.VALUE(+)

C    Start main iteration. Compare each number with its neighbour in descending
     order

     IF(.NOT.ANY(CHANGE)) RETURN

C    Terminate if no further interchanges required

     CHANGE = CHANGE.AND.MASK
     CHANGE = CHANGE.OR.CHANGE(-)

C    Only swap on successive odd-even pairs on alternate steps

     VALUE(CHANGE) = MERGE(VALUE(+), VALUE(-), MASK)

C    Merge original with swapped pairs as defined by CHANGE, using high-level
C    MERGE function to perform the swapping

     MASK = .NOT.MASK
     GO TO 1

C    Start next iteration
```

5 Conclusion

The DAP, initially conceived as a machine for weather forecasting, provides powerful processing capability for a much wider range of applications than the traditional fast floating-point (vector) processors. The content addressable nature of the processors allows the full performance of the DAP to be achieved using the high-level DAP-Fortran language, without recourse to expensive assembly-language programming. Integration of the processors into the store of an advanced general-purpose computer reduces the required design effort and production costs while providing full user facilities.

Acknowledgments

I should like to thank Dr Dennis Parkinson for his valuable help during the production of this paper. The design and development of the DAP-Fortran language is largely the work of Peter Flanders of RADC, who also provided invaluable help during the testing of the algorithms. David Hunt of RADC provided the information on the numerical algorithms given in the Appendix.

References

1 PARKINSON, D.: 'An Introduction to Array Processors' *Syst.Int.*, 1977
2 SCARROTT, G.G.: 'Wind of Change', *ICL Tech.J.*, 1978, 1, pp. 35–49
3 FLANDERS, P.M.: 'DAP-Fortran Language'. CM 39, RADC, ICL, 1976
4 GOSTICK, R.W.: 'Introduction to DAP-Fortran'. AP 20, ICL, 1978
5 GOSTICK, R.W.: 'High Level User Diagnostics with the ICL DAP'. LASL Workshop on Vector and Array Processors, September 1978
6 STONE, H.S.: 'Parallel processing with the perfect shuffle', *IEEE Trans.* 1971, C-20, pp. 153–161

Appendix
DAP Algorithms for standard numerical functions

1 Introduction
Any system which may have to deal with even the most modest amount of scientific computation must provide standard functions such as square root, logarithm, exponential and sine. Many algorithms are available for these based either on orthogonal polynomials or on iterative processes such as Newton-Raphson; these can be very powerful and efficient on a serial machine but are not ideal on the DAP as a consequence of the extreme simplicity and bit-level operation of the PEs. The algorithms which we have used, far from being penalised by these properties, actually take advantage of them; and as they produce their results one bit at a time they have the additional advantage of allowing one to specify and control the precision of the calculation easily and exactly. The methods are well known in the sense that accounts of them have been published in various places; useful references are Chen[1] and Egbert[2]. The following notes give brief descriptions of those used on the DAP for square root, logarithm and sine/cosine; they are intended simply to show the principles of the algorithms and give neither programming details nor indications of refinements which are included in the actual programs to improve the efficiency. A full acount will be written later.

2 Square Root
Given a number N we want to find x such that $x^2 = N$ to the precision to which we have decided to work. Suppose we have found the first n bits of x, giving us an n-bit number x_n which approximates x to this number of bits and satisfied $x_n^2 < N$. We want to find the next approximation x_{n+1}, which will have the same first n bits

and the (n+1)th bit b_{n+1} such that $(x_n + b_{n+1})^2 < N$. If r_n is the error at stage n then

$$r_n = N - x_n^2$$

so $\quad r_{n+1} = N - (x_n + b_{n+1})^2 = r_n - (2x_n + b_{n+1})\ b_{n+1}$

Then $b_{n+1} = 1$ if this gives $r_{n+1} > 0$, otherwise $b_{n+1} = 0$.

Thus we can start with a first approximation $x_0 = 0$ and build up the root digit by digit. If at any stage we find $r_n = 0$ the corresponding x_n is the exact root and we terminate there.

No arithmetic is needed in forming the error terms. $2x_n$ is x_n shifted one place to the left, producing a 0 at the right; addition of b_{n+1} is then done by putting a 1 at the right of this zero and the final multiplication by b_{n+1} is a shift of n+1 places to the right. When the program is worked out in detail it is found that the number of operations required is about half that for a division to the same number of bits.

3 Logarithm

Given x we want to find $\ln x$ to some stated number of bits, where as usual ln means the logarithm to base e. If we can find a set of constants $a_1, a_2, \ldots . a_i, \ldots$, such that

$$x\, a_1\, a_2 \; \ldots .\; a_n = 1 \text{ to the stated precision, then}$$
$$\ln x = -\ln a_1 - \ln a_2 - \;\ldots\; - \ln a_n$$

We can achieve this by taking $a_i = 1 + 2^{-i}$ and storing a table of $\ln a_i$. Multiplication by any a_i then requires only a shift and an add. Suppose we have got to the nth stage, the last a_i used being a_n. To go to the next stage we multiply the current product by a_{n+1} and add $-\ln a_{n+1}$ (looked up in the stored table) to the current sum if and only if the new product remains less than 1; otherwise we ignore a_{n+1} and continue with a_{n+2}.

As all the a_is are greater than 1 we must first scale x if necessary to bring it below 1; the standard arrangement is to scale (by shifting) so that we start the process with a value between ½ and 1. The final stages can be truncated: if at stage n we have

$x a_1\, a_2 \;\ldots .\; a_n = 1 - \delta$ where δ is small enough for δ^2 to be neglected to the precision to which we are working; then since $\ln(1 - \delta) = -\delta - \tfrac{1}{2}\delta^2 \ldots$ we have

$$\ln x = -\ln a_1 - \ln a_2 - \;\ldots\; - \ln a_n - \delta$$

Exp *(x)* is found by an analogous process in which numbers from the sequence $a_i = \ln(1 + 2^{-i})$ are added to produce x.

4 Trigonometric Functions

We have to compute both sine and cosine and for this we use the standard addition formulae written as

$$\sin(A \pm B) = \cos B\ [\sin A \pm \cos A\ \tan B]$$
$$\cos(A \pm B) = \cos B\ [\cos A \mp \sin A\ \tan B]$$

together with a precomputed table of numbers θ_n for which

$$\tan\theta_n = 2^{-n}\ (0 < \theta_n < \tfrac{1}{2}\pi), n = 0, 1, 2 \ldots.$$

To find the sine or cosine of a given x we first reduce x to the first quadrant $(0 < x < \tfrac{1}{2}\pi)$ and then, starting with $\sin 0 = 0$, $\cos 0 = 1$, add or subtract the successive members of the sequence (θ_n) until we have built up x. Thus if we have used $\theta_0, \theta_1, \theta_2, \ldots \theta_n$ giving us an approximation x_n, then the next approximation is

$$x_{n+1} = x_n \pm \theta_{n+1} \text{ according as } x_n \lessgtr x$$

The corresponding approximations t_0 sin x and cos x are

$$\begin{aligned} \sin x_{n+1} &= \cos\theta_{n+1}(\sin x_n \pm \cos x_n \tan\theta_{n+1}) \\ &= \cos\theta_{n+1}(\sin x_n \pm 2^{-n-1}\cos x_n) \\ \cos x_{n+1} &= \cos\theta_{n+1}(\cos x_n \mp 2^{-n-1}\sin x_n) \end{aligned}$$

and the evaluation of the terms in the brackets involves only shifting and adding. If we write the relation in matrix form

$$\begin{pmatrix} \sin x_{n+1} \\ \cos x_{n+1} \end{pmatrix} = \cos\theta_{n+1} \begin{pmatrix} 1 & \pm 2^{-n-1} \\ \mp 2^{-n-1} & 1 \end{pmatrix} \begin{pmatrix} \sin x_n \\ \cos x_n \end{pmatrix}$$

we see that the $\cos\theta_i$ terms enter into the final result only as the product $\cos\theta_0 \cos\theta_1 \ldots \cos\theta_n$; this can be precomputed for $n = 0, 1, 2, \ldots$ and stored, and the required product looked up and applied at the end of the process.

5 Execution times

Codes have been written for these algorithms and have been run on the 32 x 32 pilot machine in the Stevenage laboratory. More highly optimised versions are being developed on the 64 x 64 machine now at Bracknell but as this development is still going on it is not appropriate to quote actual times at this moment. However, reasonably firm values relative to the basic multiplication time can be given and these are as follows. All refer to 32-bit floating-point numbers with 24-bit mantissae.

square root	0•65 multiplication times
logarithm (base *e*)	1•15
sine or cosine	2•5
sine and cosine	3
and for comparison	
add or subtract	0•7

References

1 CHEN T.C.: 'Automatic computation of exponentials, logarithms, ratios and square roots', *IBM J. Res. & Dev.*, 1972, pp. 380-388

2 EGBERT W.E. 'Personal calculator algorithms IV: logarithmic functions', *Hewlett-Packard J.*, 1978, 29, (8), pp. 29-32

Hardware monitoring on the 2900 range

A.J. Boswell and M.W. Brogan
ICL Product Development Group, Systems Control Division, Bracknell

Abstract

The main purpose and practice of hardware monitoring as carried out in ICL are outlined. After a brief summary of the relevant features of the 2900 architecture the monitoring equipment is described, followed by an account of some of the measurement techniques that have been developed and of the ways in which these can be applied to machines of the 2900 range. There are comments on the possibilities of statistical and other errors in the measurements and indications of how these can be dealt with.

1 Objectives

The work described was carried out in the Product Development Group of ICL. We have a small team which is devoted to performance measurement. The work we do has two main objectives: to quantify the performance, and then determine the product enhancements that are necessary to improve performance.

First we need to find out how well our systems perform. We do that in two ways. We run benchmarks as standard tests of system performance, and measure throughput. We also measure low-level critical functions in the system, I/O pathlengths for example. In this way we can control performance and monitor the progress of our systems towards their performance goals. We also use the detailed measurements as input for sizing guides which are then used to size customer benchmarks.

The other aspect of our work is investigating problems that have been thrown up either by our own work or by problems that other users have experienced. The objective is to identify the software or hardware enhancement required to get us out of the problem and back on the path towards the most effective performance of the system.

2 2900 Architecture

The structure of 2900 systems is based on the concept of virtual-machine environment (VME), which means that each batch job or terminal session runs in its own virtual machine containing both the user application program and the supervisor. Each virtual machine holds its instructions and data in a virtual store which is logically split into two halves: the local virtual store which contains data private to the particular virtual machine, and the public virtual store which is common to all virtual machines and contains the supervisor.

The format of a virtual address is shown in Fig. 1. The virtual store is organised into segments of variable size, each made up of a set of 1 kbyte pages. Each segment has a set of segment properties associated with it, including the read and write access keys which are used in conjunction with the access control register (ACR) to protect the data, and the execute permission bit which prevents data being executed as code by mistake owing to program error.

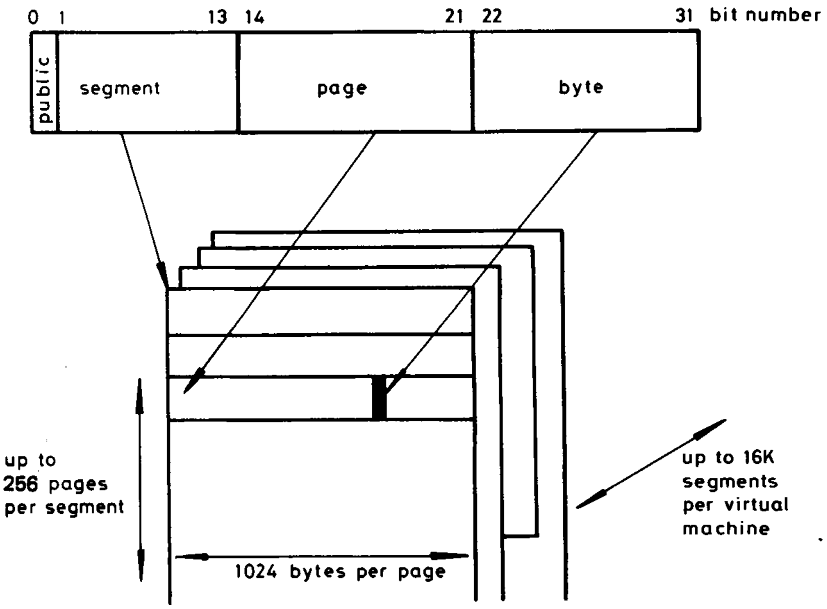

0
1
13
14
21
22
31
bit number
public
segment
page
byte
up to 256 pages per segment
1024 bytes per page
up to 16K segments per virtual machine

Fig. 1 Format of a virtual address

Each virtual machine has one specified segment used as a last-in first-out stack. Each procedure has a 'local name space' on the stack, and the instruction set and hardware design are optimised for stack access.

Fig. 2 shows some of the most important CPU registers. There is one general-purpose arithmetic register called ACC, which can be set to the required size for 32, 64 or 128 bit working. Data can be accessed via the descriptor register, which describes the type and length of data as well as its virtual address. The descriptor can be modified by the B register. Alternatively, data can also be accessed relative to base registers such as local name base (LNB) and extra name base (XNB).

CPU REGISTERS

* ACC	accumulator
* DESCRIPTOR	
* B-REGISTER	modifier
* LNB	local name base
* XNB	extra name base
* ACR	access control register
* PC	program counter (instruction address)

Fig. 2 CPU registers

The most important register from the point of view of hardware monitoring is the ACR, which is used in conjunction with the read and write access keys to limit the amount of data accessible to a program. Whenever a data access is attempted, the hardware checks the current value of the ACR against the access keys of the segment containing the data. On VME/B, user programs run with ACR 10-15 and ACR levels between two and nine are allocated to the various supervisor levels.

If a segment has a read access key set to a value of eight, then, if a user program running at ACR10 attempts to read from it, a program error will result. Access is only allowed if the ACR is not greater than the access key.

The values of ACR allocated on VME/B are as follows:

ACR 10 - 15	user program
ACR 8, 9, 10	system control language (SCL) interpretation
ACR 7	record access
ACR 5	loader and catalog handler
ACR 4	block access
ACR 2	kernel
ACR 1	idle

By attaching a hardware monitor to the four ACR bits, we can determine which of the above levels of software is active at any point in time. In fact, the idle loop was put at ACR 1 just so that the hardware monitor could pick it up easily.

3 Hardware monitor*

3.1 Description

The hardware monitor* used by the Performance Monitoring Team at Bracknell is shown in Fig. 3. The monitor is attached to the computer system being monitored, usually referred to as the host computer, by up to 144 probes, each of which is capable of detecting changes in voltage at the host and transmitting the resulting signal to the hardware monitor. Within the monitor, each signal is routed through a patch panel to one or more of eight registers known as collectors. The patch panel contains a number of logic elements such as AND gates, OR gates and flip-flops through which the signals can be routed before entering the collectors. This enables two or more signals to be combined together. For example, if it was necessary to count the number of input/output (I/O) transfers that occurred when the CPU was busy then this could be achieved by combining the signals 'I/O transfer' and 'CPU busy' through an AND gate and then counting the result.

The collectors are general purpose registers which, according to the mode of use, can be configured to perform a number of different functions on the incoming signals. After being processed by the collectors, data is transferred by the controller module into buffers in the minicomputer memory and from there to files on floppy disc or magnetic tape. Subsequent to a measurement session, these files can be analysed by standard analysis packages on the minicomputer system.

*The hardware monitor described here is the TESDATA 1187. However, most types of hardware monitor employ similar techniques.

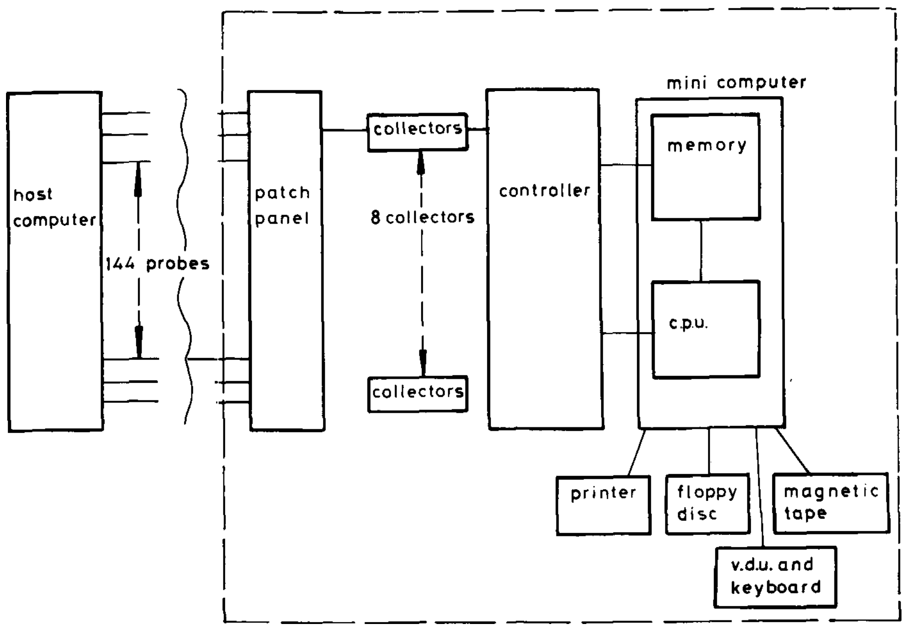


Fig. 3 Hardware monitor

3.2 Modes of operation

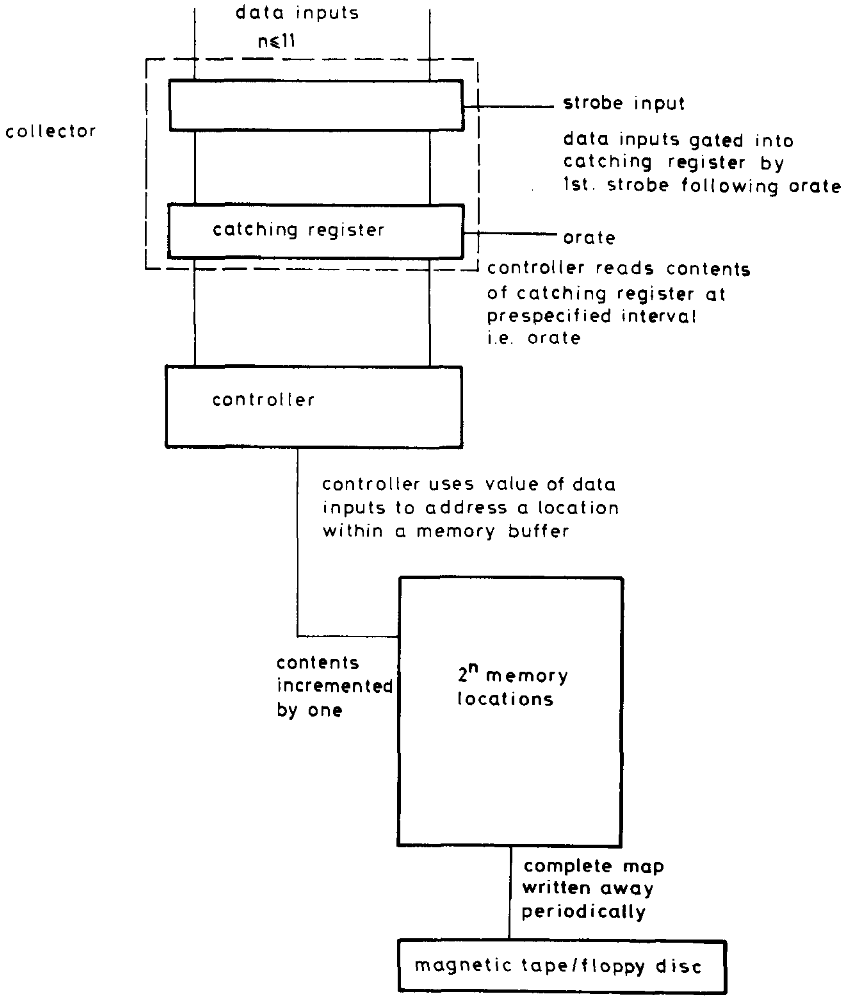
Each collector can be configured to perform in one of three different modes:

(a) count/time mode
(b) map mode
(c) store mode.

(a) Count/time mode: In this mode of operation, each collector operates as four separate modules, each of which can either count the number of times a signal changes from FALSE to TRUE or count the number of microseconds that a signal is found to be in a TRUE state. The contents of these counters are then written to tape or floppy disc at predefined intervals.

(b) Map mode: Map mode is illustrated by Fig. 4. Up to 11 signals are connected into data inputs on the collector. In addition, a futher signal is connected to the strobe input. The collector functions as follows. The first time the strobe signal goes from a FALSE to a TRUE state, the values of the data inputs are stored in the catching register. At a prespecified rate, known as the ORATE, the contents of the catching register are read by the controller and used as an address to a buffer in memory. The memory location thus addressed is then incremented by one. More data is then gated into the catching register the next time the strobe signal goes TRUE, and at the next ORATE it is read and the appropriate memory location incremented by one, and so on. Eventually

a profile or map of data input values is built up in the memory buffer and, periodically, the buffer is written away to backing store.



data inputs
n≤11
collector
strobe input
data inputs gated into catching register by 1st. strobe following orate
catching register
orate
controller reads contents of catching register at prespecified interval i.e. orate
controller
controller uses value of data inputs to address a location within a memory buffer
contents incremented by one
2^n memory locations
complete map written away periodically
magnetic tape/floppy disc

Fig. 4 Map mode

(c) Store mode: In this case up to 16 data signals can be connected to the collector together with a strobe signal. The collector functions in exactly the same way as when it operates in map mode; however, instead of using the values read from the catching register as an address the 16 bits are simply stored in the next free location in the memory buffer. The next 16 bits of data that are read at the next ORATE are stored in the next location and so on. When the buffer is full it is written away to backing store. When this data is subsequently analysed, either counts or maps can be generated by analysis software.

4 Measurement Techniques

4.1 ACR-level monitoring

Measurements of ACR levels made during runs of standard benchmark tests can be used to identify the contribution of each major software component. The following are normally measured:

(a) CPU time used at each ACR level
(b) number of instructions executed at each ACR level
(c) instruction execution rate (MIPS) at each ACR level.

The instruction count is used to check sizing techniques based on pathlengths; the variations in MIPS allow the interactions between the software and hardware to be studied.

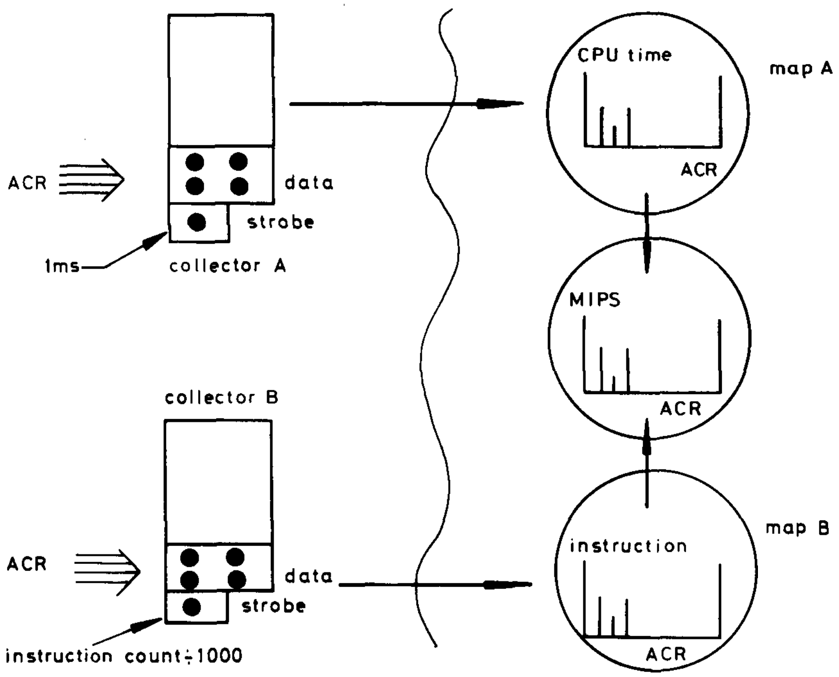


Fig. 5 ACR-level monitoring technique

Fig. 5 illustrates the hardware-monitoring technique used to achieve this. Two collectors are used, each producing a map of the four ACR bits. Collector A measures CPU time by ACR. The four ACR bits are strobed periodically, say every 1 ms; thus for every millisecond of CPU time a sample is obtained at the current ACR level. When the maps are subsequently analysed each element contains a count of milliseconds of CPU time at the corresponding ACR level. Collector B measures the instruction count by ACR level. In this case the ACR bits are strobed every 1000th instruction, the strobe signal being formed by merely dividing the instruction count signal by 1000 on the monitor patch panel. Thus for every 1000th instruction a sample is obtained at the current ACR level. When the maps are analysed each element will contain a count of the number of thousands of instructions executed at the corresponding ACR level. To obtain the instruction rate, each element of map B (thousands of instructions) is divided by the corresponding element of map A (milliseconds), giving MIPS by ACR level.

4.2 Address mapping

ACR-level monitoring gives a fairly coarse measure of the contribution to overall performance by the various system components. It is often necessary to identify the contributions of different software modules within one particular ACR level. This can be achieved by mapping the segment number of the virtual address or in some cases, where even finer resolution is required, mapping addresses within certain segments.

(a) Segments mapping: The technique used here is the same as for ACR mapping, using two collectors, one strobed periodically every millisecond and the other strobed every 1000th instruction; but in this case the segment bits of the virtual address register are used as data. One problem arises here. The hardware monitor has a maximum map length of 2048 elements, which corresponds to 11 bits of data. The segment field of the virtual address is, however, 14 bits. Usually this causes no difficulty because segments in VME/B are seldom so big; this means that the segment address can be reduced to 11 bits for mapping.

(b) Addresses within segments: To enable addresses within selected segments to be mapped, patchboard logic is used on the segment bits of the virtual address to construct a signal that is true whenever the virtual address is within the segment of interest. This indicator is then wired into the most significant bit of the collector and the page and byte address bits wired into the rest of the collector. To obtain the maximum resolution, store mode is used and the maps assembled later by analysis software as described in Section 3. Thus for one segment it is possible to map 15 bits of the page and byte addresses, which gives a resolution of 8 bytes. If it is required to map addresses within more than one segment, then a number of indicator bits may be constructed using patchboard logic and wired into the collector with a corresponding reduction in the resolution to which the addresses within those segments can be mapped. The technique to obtain distributions of instructions and CPU time at different addresses within specified segments is the same as that used for ACRs or segment numbers. Two collectors are used in store mode, in this case one being strobed every 10 ms and the other strobed every 10000th instruction.

(c) Mapping the complete address: A technique has been developed by the Performance Monitoring Team which enables up to 30 bits of the address to be monitored. Special analysis software has been written to enable this to be done. The monitor setup is shown in Fig.6. Two collectors are used – one for each half of the address – with the same strobe. To detect any errors when the two collectors get out of step, part of one collector is copied into the other (Fig. 6) and a consistency check done by the analysis software. This reduces the

number of bits that can be monitored from 32 to 30, but since it is rarely necessary to monitor down to single-byte resolution this is acceptable.

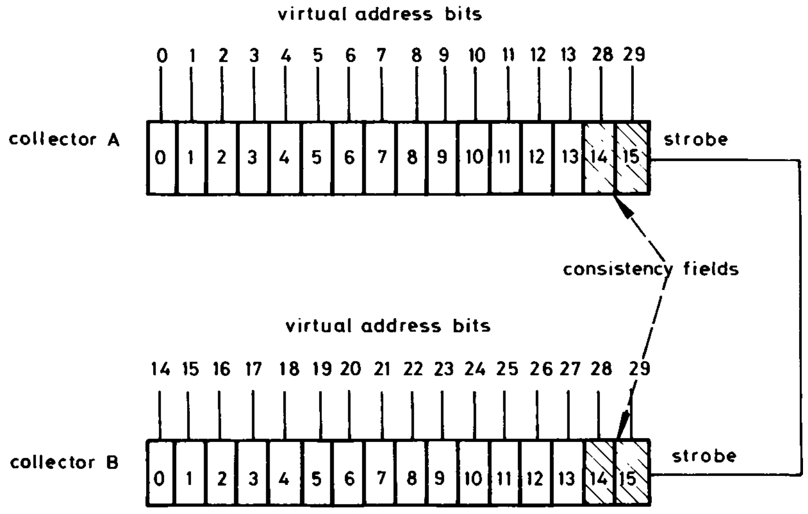


Fig. 6 30 bit address mapping

4.3 Instruction and CPU time distributions using a single collector

It is often found that the number of parameters to be measured exceeds the capacity of the monitor. In earlier sections it was described how the number of instructions and the amount of CPU time used at each ACR level or each address could be monitored. This technique required the use of two collectors, one for instructions and the other for CPU time. It is possible, however, to map both instructions and CPU time with a single collector.

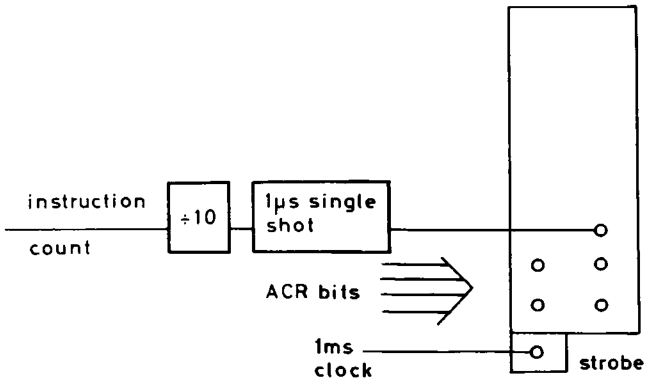


Fig. 7 Instruction and CPU time distributions using a single collector

Fig.7 illustrates this technique as used on a 2980 for ACR mapping; it applies equally well to virtual-address mapping. The instruction count signal is first converted to a signal that is TRUE for an amount of time proportional to the number of times that it (the instruction count) goes TRUE. This is done by dividing the signal by ten on the monitor patchboard and then single-shotting the result to 1 μs, which results in a 1 μs pulse every ten instructions. This signal is then wired into the collector as shown in Fig.7.

The collector is then used in the same way as the collector that mapped CPU time in Section 4.1, i.e. it is strobed with a 1 ms clock pulse. The result of this is a map with 32 elements from which the amount of CPU time and the number of instructions executed at any given ACR can be derived. The actual relations are, for ACR level n, where $n < 15$,

CPU time (ms) = element n + element $(n + 16)$
Number of instructions = element $(n + 16)$
(units of 10,000)

4.4 *Starting and stopping the monitor*

One problem that often arises when using hardware monitors is that of being able to identify the periods over which results are to be collected. It is possible to do this by specifying the required periods to the analysis software; but since the hardware monitor and mainframe clocks are set manually they are not necessarily in step.

To resolve this difficulty, a method has been devised by which the mainframe can signal to the hardware monitor when it is required to start or stop monitoring. This has been done by writing a user program which executes an IDLE instruction at the user level ACR (ACR 10) immediately after reading the real time clock. An IDLE instruction executed on VME/B is a unique event, since VME/B idles on an instruction loop so it is possible to set the hardware monitor to switch on all its collectors when an IDLE is seen at ACR10 and switch them off again when the next one is seen.

4.5 *Accuracy of results*

(a) Sampling error: Suppose it is required to monitor the execution of a program, and the total number of instructions executed is N and it is required to know what proportion p of these instructions was executed at a particular ACR level, say ACR = 2.

Because the hardware monitor is not fast enough to record the ACR level of every instruction executed, it is necessary to resort to sampling, and look at the ACR level of only n of the instructions executed.

The use of sampling techniques introduces the possibility of sampling errors. If the experiment were repeated a number of times it would be found that the number of instructions at ACR = 2 would be slightly different each time; we need to make some estimate of the spread we should expect, and therefore of the confidence we can place in whatever conclusions we draw from the experiment.

(b) Statistical treatment: Suppose our experiment has recorded a instructions at ACR = 2, giving a proportion $f = a/n$ in the sample. We want to know how good an approximation this is to the proportion p in the complete program of N instructions.

Standard statistical theory can be applied here. We are sampling from a large population of instructions, each of which is either at ACR=2 or is not, with probabilities p and q, respectively, where $q = 1 - p$. The possible numbers at ACR=2 recorded in a sample of n are given by the binomial distribution, that is the successive terms in the expansion of $(p + q)^n$. The mean number will be np and the standard deviation $\sqrt{npq}$. However, because we are taking a large sample, with n typically of the order of several thousands, this binomial distribution will approximate very closely to a normal distribution with the same mean and standard deviation; and therefore we can say that there is a 95% probability that any experiment will give a result within two standard deviations of the mean: i.e. that any sample of size n will with 95% probability give a result a in the range

$$np \pm 2\sqrt{[np(1-p)]}$$

and that therefore, if we divide by n, the proportion observed in a sample of n will with 95% probability lie in the range

$$p \pm \frac{2}{\sqrt{n}}\sqrt{[p(1-p)]}$$

We do not know the value of p—this is what we are trying to find—but we can again draw on standard theory, which tells us that for a large sample the observed proportion $f = a/n$ is an unbiassed estimator for p to an accuracy of the order of $1/\sqrt{n}$; and that, if we say that the proportion p in the complete program lies in the range

$$f \pm \frac{2}{\sqrt{n}}\sqrt{[f(1-f)]},$$

we shall be right in 95% of cases.

Example: If a program executes 100 million instructions altogether and 10,000 are sampled and it is found that 2000 of this sample are at ACR=2, then, for this case, the sample proportion is $f = 0{\cdot}2$ and the formula gives $p = 0{\cdot}2 \pm 0{\cdot}008$. We can therefore be confident that the total number executed at ACR=2 is in the range 20 million ± 0•8 million, which is equivalent to saying that the estimate of 20 million derived from the sample is correct to within 4%.

The formula can obviously be applied to other situations. For example, instead of ACR = 2 it could be the execution of code at a particular virtual address; or it could be sampling events other than instruction executions, such as virtual-store interrupts or units of mill time.

(c) Other sources of error: The above has dealt with the error introduced by random sampling; the possibility of errors from other sources needs to be

considered. For example, most computer systems have some periodic functions and if these get into synchronisation with the sampling frequency then distorted results will be obtained. It is useful when setting up a measurement to measure the same thing in different ways to check this. Thus the ACR = 10 signal can be timed and then compared with the ACR 10 element of the ACR map.

Section 4.4. described the method that has been devised for eliminating errors which could arise from the setting of the clocks. These could be much more serious than the potential errors just discussed. For example, suppose a program which takes 100 s to run is being monitored and written to the monitor file every 10 s. The recording resolution means that the monitor cannot measure precisely the period during which the program is running, so the result is only accurate to within 10 s, which is a large uncertainty. The possibility of the mainframe and monitor clocks not being accurately in step could also lead to errors of this order. This was the reason for developing that particular piece of technique.

5 Validation

This paper has described a few of the hardware monitoring techniques that have been used in product development. Extremely valuable measurements have been obtained by developing new techniques but when these are implemented it is essential to validate the correctness and accuracy of the results. This can be done by monitoring programs for which the values of the measured quantities are already known (e.g. executing exactly one million instructions at ACR 10). A new technique should only be used in earnest when sufficient confidence in it has been established by validation. Even when this has been done, it is still advisable to run validation tests periodically to check for malfunctions in the hardware monitor or its connections to the host machine.

Network models of system performance

C.M. Berners-Lee*
ICL Product Development Group, Bracknell, Berkshire

Abstract

The paper describes the derivation and validation of the FAST model of system performance. A general theory of a wider class of such models is described. The FAST model and closed multiclass models are then derived as particular cases. An Appendix gives a short report on a meeting at the University of Maryland at which models of this type were discussed. The author attended and took part in this meeting shortly after completing the main paper.

1 Introduction

Since 1972 ICL has made extensive use of models using networks of queues to represent the behaviour of batch-processing and multiple-access-computing (MAC) workloads on ICL and other systems. These models have been used to predict the effect of changes of system resources on the throughput of work. The model used has been known in ICL as FAST – football analogy of system throughput. The analogy is that of a group of players kicking several footballs around between them. The success of the model for use by systems staff in the field has been due in no small measure to the fact that the theory can be thus simply explained.

The analogy represents the components of a real system by players in the game and the programs or processes operating concurrently by the footballs. If, for example, a program in the central processing unit (CPU) initiates a peripheral transfer, the initiative may pass from the CPU to a disc controller and thence to the disc drive, finally returning to the CPU on completion of the transfer. This is represented by the ball being kicked along the route CPU – controller – disc – CPU. When the multiprocessing level is increased from 1 to some number N there will be an increase in throughput by some factor depending on N, say $f(N)$. The model enables this factor to be calculated, given the usages of the different system components (CPU, controllers, discs, tapes, slow peripherals, communications controllers etc.) measured in appropriate time units. For the CPU the usage is the total mill time and for other devices it is the number of device services multiplied by the service time.

FAST was first used in 1972 to study the effects of alternative means of improving the effectiveness of the 1900 disc controllers. It had been derived by generalising the Konigsberg model[1] of a cyclic queueing system, such as arises with a set of buses travelling around a circular route. This generalisation was first noted by J.R. Jackson[2] in 1963 and has been applied to computer systems by Buzen[3] and others

*Now with Corporate Communication Division, Putney in the United States. Since then the basic theory has been extended in the USA to include less restrictive assumptions and the resulting literature is substantial. The purpose of this paper is to present the core of the thinking as simply as possible.

In ICL, FAST was used extensively for pricing studies for the 2900 series. The requirement here was for a systematic method of assessing the economic value to the user of particular system components in terms of their effect on the throughput of 'typical' user work. The technique has also been used within the Product Development Group for a number of purposes, including the evaluation of dual-processor systems and of alternative strategies for DME systems, and for the study of pipeline performance. Another topic is the evaluation of different store hierarchies using different technologies. All this product-evaluation work has depended on establishing profiles of typical or average customer workloads in terms that are applicable to the model. Such profiles comprise the mill utilisations together with the utilisations of the peripheral devices.

In its original form the FAST model made very crude assumptions about the statistical behaviour of the processes. It was assumed that each system component behaves as a server whose service time is a random negative-exponentially distributed variable, serving processes which move around the system in a random manner. The actual situation is that the CPU service times in a large system are fairly close to negative exponential but that the peripheral service times are very different in shape–for example, line-printer times are cyclic. However, these peripherals are seldom heavily loaded and the effect of the shape is only felt at the higher loadings: it may be neglected to a first approximation. The assumptions with regard to randomness provide, in theory, a much more difficult problem. However, in practice, it is well known that quite small injections of random behaviour are sufficient to make a system behave in an effectively random manner. Some light has been thrown on this point by Buzen and Denning[4], who point out that theories of this type only require that the average service rates at given queue lengths should have the same values as they would have had had the assumptions regarding randomness been strictly fulfilled. Nevertheless there are cases where nonrandom behaviour is significant, notably in small systems operating at a low level of multiprogramming, where the system may be locked in repetitive cycles of operations for an appreciable time.

Another respect in which real systems differ from the theoretical models is that it sometimes happens that the processes are not conserved in the way implied by the football model. For example, a process may continue to execute in the CPU after having initiated a peripheral transfer. In this case an additional process is in effect in operation until the CPU execution of the original process ceases. Thus double buffering may result in an effective increase in the value of the multiprogramming level N.

In practice these departures of the model from reality can be regarded as perturbations of the effective value of N. Thus if we run a workload or benchmark on a real system and use the resulting component usages to calculate the value of N that would explain a given throughput, this value will give an effective concurrency which may be used to predict the effects of changes in system resources. This was the method used in the Trondheim validation of the model, which will be referred to in Section 2.

In the years since 1972 previously started and parallel developments in the USA have led to a considerable proliferation of models of the same general class as FAST. These enable more complex situations to be treated, in particular the processes, instead of being assumed to be statistically identical, may have different statistical characteristics, provided that certain mathematically tractable assumptions can be made about the way these different processes are scheduled for service by the system components. These later developments enable the effects of the simple assumptions of the original FAST model to be evaluated, to see if further elaboration of the model would provide worthwhile improvements in accuracy. So far our conclusion has been that this more recent work reinforces the case for using FAST in its original form for throughput calculations. For response times, however, the more detailed analysis is sometimes valuable.

In the literature generally, network queueing models are classified as either open or closed. An open model is typified by a single server serving an infinite stream of processes as in simple queueing theory. An open network replaces the single server by a network of servers and the number of processes (customers) in the network will vary with time. In the closed case there is no infinite stream and the number of footballs in the game is conserved, remaining constant at a given value N. The open game can be regarded as the limiting case of the closed game, in which N tends to infinity and the most heavily loaded server becomes the source and sink for the infinite stream. Only the closed game, in which the number of balls N is conserved, is considered in this paper.

The paper as a whole develops the theory of closed models using the football analogy in its most general form: here the transition rates are general functions of the midflight queue state, i.e. the state of the queues of balls at the feet of the players when the ball is in flight from one player to another and not at the feet of any player and therefore not part of any queue. The treatment is then progressively specialised until the particular form known in ICL as FAST is reached, which is characterised by the kicking rates being constant. The paper concludes with an introduction to the American theory of multiclass models, deriving these from the general theory by means of a different mapping of players and footballs on to the system to be represented.

To be a little more specific, after a consideration of validation of the model in Section 2 a top-down approach to the development of the general theory is given in Sections 3–5, in which the taxonomy of the models is shown. Sections 6 and 7 discuss the particular case of FAST in more detail. Sections 8 and 9 give an introduction to the theory of multiclass models and Section 10 deals with the use of such models for calculating queue lengths. Section 11 discusses the problem of implementation in the light of the queueing results.

2 The Trondheim validation

It was realised from the outset that model validation must go hand in hand with construction and use. The process was greatly aided by two developments, both of which sprang from the conference on system performance held at the University of Surrey in 1972 at which FAST was first described.[5] Another paper[6] at this conference gave a description of a technique devised by the Royal Radar Establishment for displaying system-monitoring information, which clearly pointed the way to a means of obtaining a detailed minute-by-minute validation of model predictions of throughput at different multiprogramming levels over a period of time.

The conference was attended also by Hughes and Moe of the University of Trondheim, who went away and undertook a quite independent validation study, unknown to us at the time. This was published later as an AFIPS paper[7] and showed excellent agreement between measurements and predictions of changes in CPU and device utilisations resulting from major changes made to the system, such as increasing the core store and relocating files on peripheral devices. Confidence in the general validity of the approach was increased by the knowledge that the installation was from a different manufacturer, Univac in this case.

Similar studies have been repeated wherever possible, using benchmark data or measurements of normal work and obtained with the aid of either hardware or software monitoring.

The technique of using FAST was documented by the ICL Systems and Technical Support organisation and was introduced into the Beaumont training school where it proved valuable in providing insight into system behaviour. It is now used regularly as one of a number of aids to sizing.

The model enables predictions to be made about the proportions of time for which channels are jointly busy and also about the lengths of disc queues. For jointly busy channels, time comparisons were available from hardware-monitoring studies and were found to be satisfactory[5]. For disc queues, there was very satisfactory agreement with measurements made by the Road Research Laboratory, which indicated that the model was significantly more accurate than a simple queueing formula. We can say in fact that such validation studies as we have been able to make have proved satisfactory.

3 State-change models

The models we are concerned with work by considering the number of times a system changes state, equating the number of times it enters a given state to the number of times it leaves that state. Markov models are a class of such state-change models. In general, suppose we have a system which can be in any one of a number L of possible states and that the average rate at which it changes from a state Q to a state R is $F(Q, R)$ changes per second. Then, if we equate the total number of changes from the state Q into all other possible states R, to the total number of changes from other states into Q, we have

$$\sum_{R} F(Q,R) = \sum_{R} F(R,Q) \tag{1}$$

where the summation is over all possible states R.

Suppose now that the system spends a fraction $P(Q)$ of its time in state Q; if we denote by $A(Q,R)$ the average rate of change from Q to R *per unit of time spent in* Q then

$$\begin{aligned} A(Q,R) &= F(Q,R)/P(Q) \quad \text{if } P(Q) \neq 0 \\ &= 0 \qquad\qquad\quad P(Q) = 0 \end{aligned} \tag{2}$$

and hence $$\sum_R A(Q,R)\,P(Q) = \sum_R A(R,Q)\,P(R) \tag{3}$$

If we have some information from which we can deduce the values of the rates A, we can solve these equations for the proportions of time spent in the different states; for example, we may be able to determine the idle time for a system component. Eqns. 1 and hence their equivalent eqns. 3 are not all independent, for if they are summed for all Q an identity results; the further relation necessary to compensate for this is that the proportions $P(Q)$ for all Q must sum to unity.

However, there are two practical difficulties: the first is that we cannot estimate the rates A without some further assumptions about the state changes and the second is that the number of equations in any realistic case is unmanageably, indeed astronomically, large. Fortunately, however, eqns. 3—known as the global balance equations—can in practice be given a special structure which enable them to be solved in a trivially easy way. We shall use the term football models to refer to systems that exhibit this structure in a way to be described. The problem of the assumptions made about the relationship between the values of A and the observables in the real world will be considered after the football model in its most general form has been described and analysed.

As mentioned earlier, the model was originally thought of as one representing computer-system components and processes by players and footballs, respectively. However, in view of the later American developments, it will be convenient to develop a general football model which can be identified with real systems by means of different mappings of model entities on to system entities. The approach in what follows is to begin with a very general form of model which is gradually specialised until we eventually arrive at those used in practice, such as FAST and the American multiclass models.

One of the ways in which a state-change model can arise is by means of a continuous-time Markov process. This involves much more specific assumptions about the fine structure of the behaviour of the system over time. Specifically, it is assumed that the average transition rates $A(Q,R)$ arise from constant probabilities proportional to $A(Q,R)\,dt$ that a transition will occur in the next interval dt. Thus the probability that a transition from Q to some other state will occur in dt is equal to

$$\sum_R A(Q,R)\,dt$$

and the probability that it will be to a given state R is

$$A(Q,R)\,dt/\sum_R A(Q,R).$$

The Markov assumption implies that the successive transitions are statistically independent, and the consequence is that the durations in state Q have a negative-exponential distribution with mean value

$$1/\sum_R A(Q,R).$$

In other words, there is a direct relationship between the average transition rates and the average durations of states. Further, the theory of Markov models leads to the result that, provided there exists between any pair of states a path through some chain of intermediate states along which the transition probabilities are all nonzero, then the frequencies with which the system occupies states Q will converge in a statistical sense to stable proportions of time $P(Q)$.

In what follows, care will be taken to indicate the precise point at which a continuous Markov system is assumed.

4 Football models

Suppose we have a game in which M players are kicking N footballs around. The state of the game will be regarded as being defined by the state of the queues of balls waiting to be kicked by the players; we denote by Q_i the length of the queue (number of balls at the feet of the ith player) the sum of these numbers being constant and equal to N. We regard the Q_i as components of a vector $\boldsymbol{Q}$ which defines the state of the system, and which plays the role of the state identifier Q of the previous section. Any vector $\boldsymbol{Q}$ that has nonnegative components summing to N is said to be a *feasible* vector.

If now the ith player kicks the first ball in his queue–i.e. the first to have arrived–to the jth player, the state of the game changes from

$$Q = (Q_1, Q_2, \ldots Q_i, \ldots Q_j, \ldots Q_M)$$
to
$$R = (Q_1, Q_2, \ldots Q_i{-}1, \ldots Q_j{+}1, \ldots Q_M)$$

Let us denote by $\boldsymbol{I}$ the unit vector for which the ith component is 1 and all the others are zero; and similarly for $\boldsymbol{J}$. Then $\boldsymbol{R} = \boldsymbol{Q} - \boldsymbol{I} + \boldsymbol{J}$.

As in Section 3 we denote the mean rate of occurrence of changes from $\boldsymbol{Q}$ to $\boldsymbol{R}$ by the flow function $F(\boldsymbol{Q},\boldsymbol{R})$. We see that in the football game this can be nonzero only if $\boldsymbol{Q}$ and $\boldsymbol{R}$ are identical in all but two components and that these must differ by unity. Such a pair of feasible vectors are said to be *neighbours*. The relationship between neighbours is thus symmetrical, and, if a kick from $\boldsymbol{Q}$ to $\boldsymbol{R}$ is possible, one is possible also from $\boldsymbol{R}$ to $\boldsymbol{Q}$ by player j kicking to player i. This changes the state from $\boldsymbol{R}$ into $\boldsymbol{R} - \boldsymbol{J} + \boldsymbol{I} = (\boldsymbol{Q} - \boldsymbol{I} + \boldsymbol{J}) - \boldsymbol{J} + \boldsymbol{I} = \boldsymbol{Q}$. In either direction, during the kick itself the game passes through what may be called the midflight state in which the remaining balls form a set of queues described by the vector $\boldsymbol{Q} - \boldsymbol{I}$, which is the same as $\boldsymbol{R} - \boldsymbol{J}$.

For $\boldsymbol{Q} - \boldsymbol{I} + \boldsymbol{J}$ to be a neighbour of $\boldsymbol{Q}$ it must be feasible, so no component may be negative; since its ith component is $Q_i - 1$ this means that $Q_i > 0$, which is obvious, since the player cannot kick if his queue is empty. Thus the neighbours of $\boldsymbol{Q}$ are those vectors $\boldsymbol{Q} - \boldsymbol{I} + \boldsymbol{J}$ for all i for which $Q_i > 0$ and for all j. Hence eqns. 1 become

$$\sum_i{}' \sum_j F(Q, Q-I+J) = \sum_i{}' \sum_j F(Q-I+J, Q) \tag{4}$$

where $\sum_i{}'$ indicated summation over all values of i for which $Q_i > 0$.

These are the *global balance equations.*

The simplicity of structure already referred to results when the equality of the two sides holds for each value of i; i.e. when for all i for which $Q_i > 0$

$$\sum_j F(Q, Q - I + J) = \sum_j F(Q - I + J, Q) \tag{5}$$

These are called the *local balance equations*. If they are satisfied, then by summation over i it follows that eqns. 4 are also satisfied.

Local balance models come in families. If we define a vector function $S(\boldsymbol{Q},\boldsymbol{R})$ to be equal to the midflight vector $\boldsymbol{Q} - \boldsymbol{I} = \boldsymbol{R} - \boldsymbol{J}$ if $\boldsymbol{Q},\boldsymbol{R}$ are neighbours and zero otherwise, then $S(\boldsymbol{Q},\boldsymbol{R}) = S(\boldsymbol{R},\boldsymbol{Q})$. If now $h(\boldsymbol{S})$ is any scalar function of the vector $\boldsymbol{S}$, it follows that $h[S(\boldsymbol{Q},\boldsymbol{R})] = h[S(\boldsymbol{R},\boldsymbol{Q})]$ and each is $h(\boldsymbol{Q} - \boldsymbol{I}) = h(\boldsymbol{R} - \boldsymbol{J})$. Then if we define a new flow function $F'(\boldsymbol{Q},\boldsymbol{R})$ as $F(\boldsymbol{Q},\boldsymbol{R}).\, h[S(\boldsymbol{Q},\boldsymbol{R})]$ we see, by multiplying both sides of eqn. 5 by h, that this also satisfies the local balance equations.

We can use this result to derive a very general class of football models. Consider the simple case of a constant flow function

$$F(Q, Q - I + J) = f(i,j) \tag{6}$$

which means that the rate of flow between neighbouring states depends only on the pair of players involved. This will satisfy the local balance eqns. 5 if, for each i, eqns. 5 if, for each i,

$$\sum_j f(i,j) = \sum_j f(j,i) \tag{7}$$

Hence a set of general flow functions satisfying local and therefore global balance is given by multiplying eqn. 7 by $h(S) = h(Q - I)$:

$$F(Q, Q - I + J) = f(i,j) . h\,(Q - I) \tag{8}$$

If now we divide the flow by the proportion of time spent in the state we get, from eqn. 2,

$$A(Q, Q - I + J) = f(i,j) . h(Q - I)/P(Q) \tag{9}$$

This relationship gives the average kicking rate A, which would explain the existence of a given probability distribution of the states Q.

The transformation just described gives us considerable scope for fitting models of this type to real systems, because the function h is now at our disposal.

5 Team games

Suppose we have a scalar function $X(Q)$ of the vector Q which is positive for all Q whose components are nonnegative, and suppose that G is the sum of $X(Q)$ taken over all feasible Q – i.e. over all Q with nonegative components summing to N. We can define a game by equating both $P(Q)$ and $h(Q)$ to $X(Q)/G$. With this, the kicking rates from eqn. 9 are

$$A(Q, Q - I + J) = f(i,j) . X(Q - I) \;/\; X(Q) \qquad (10$$

Now suppose that the players are divided into a number T of sets which we call teams; and that $X(Q)$ has the form

$$X(Q) = X_1(Q)\, X_2(Q) \;\; \ldots\ldots \; X_T(Q) \tag{11}$$

where $X_t(Q)$ is a function of the lengths of the queues of balls at the feet of the players from team t only. For example, if team 1 was made up of players 1,2,3, we should have $X_1\,(Q) = X_1\,(Q_1, Q_2, Q_3)$.

If player i belongs to team t then from eqns 10 and 11 we find

$$A(Q, Q - I + J) = f\,(i,j)\, X_t\,(Q - I)/X_t(Q) \tag{12}$$

since all the other X-factors cancel.

Thus we have a model in which the kicking rates are functions of the queue lengths of the team to which the kicking player belongs and no others. If we carry this to the extreme where each team has only a single member we have a model in which the kicking rates depend only on the length of queue at the feet of the kicker. The equation is now

$$A(Q, Q - I + J) = f\,(i,j)\, X_i\,(Q_i - 1)/\, X_i(Q_i) \tag{13}$$

with

$$P(Q) = \left\{ X_1\,(Q_1)\, X_2\,(Q_2) \;\ldots . X_M(Q_M) \right\} /G \tag{14}$$

This is the Jackson model, in which the probability of being in a given state is expressed as the product of functions of the queue lengths.

A particularly interesting case arises if the ratio of the Xs in eqn. 13 is constant. This will happen if $X_i(Q_i)$ has the form $x_i^{Q_i}$, where the x_is are as yet undefined. We have now a model with constant kicking rates for which

$$A(Q, Q - I + J) = f(i,j)/x_i = k(i,j) \text{ say} \tag{15}$$

and

$$P(Q) = \left\{ x_1^{Q_1} \; x_2^{Q_2} \; \ldots . \; x_M^{Q_M} \right\} /G \tag{16}$$

where

$$G = \sum x_1^{Q_1} \; x_2^{Q_2}2 \ldots . \; x_M^{Q_M} \quad \text{summed over all } Q_i{>}0$$

$$\text{such that} \quad \sum Q_i = N$$

A model of this last type arises if we have a Markov football game in which there is a constant probability $k(i,j)\, dt$ that if the player i has the ball–that is, $Q_i{>}0$–he will kick it to player j in the next interval dt. Since from eqn. 15 $f(i,j) = k(i,j)x_i$, it follows from eqn. 8 that

$$\sum_j k(i,j)\, x_i = \sum_j k(j,i)\, x_j \tag{17}$$

If we consider a game in which there is only one ball in play (only one process in the system) then $N = 1$ and eqns. 3 reduce to

$$\sum_j k(i,j)\, P(I) = \sum_j k(j,i)\, P(J)$$

from which it is evident that the quantities x_i are proportional to the times the ball spends with players i, i.e. to $P(I)$.

6 Throughput and component utilisations

We are usually interested in the throughput of a batch-processing/MAC system or subsystem and for the purpose of analysing the behaviour of such a system it is almost always sufficient to use a simple Jackson model with constant kicking rates. This has the great advantage that the model parameters x_i bear a simple relationship to the utilisations of the components of the real system. We therefore assume a Markov system with negative-exponential distributions of service times and random movement of processes around a system of M components (players) at a multiprocessing level of N (number of balls in play). The results (eqns. 15-17) are now available.

The total throughput of processes from player i to player j is given by summing eqn. 8 over all feasible Q such that $Q_i > 0$; with $h(Q) = X(Q)/G$ this gives for the throughput $\theta(i,j)$

$$\theta(i,j) = {\sum}' F(Q, Q - I + J) = f(i,j) {\sum}' X(Q - I)/G \tag{18}$$

Each value of $Q - I$ such that $Q_i > 0$ is a member of the set of vectors with nonnegative components summing to $N-1$. In general we denote the sume of values of $X(Q)$ over all Q with nonnegative components adding to n as $g(n)$, so that

$$G = g(N) \text{ and } {\sum}' X(Q - I) = g(N - 1) \tag{19}$$

Hence the throughput $\theta(i,j)$ from i to j is

$$\theta(i,j) = f(i,j)\, g(N - 1)/g(N) \tag{20}$$

Using eqn. 15 and summing over all j we get for the total throughput from i to all other players

$$\sum_j{}' \theta(i,j) = \frac{g(N-1)}{g(N)} \sum_j{}' k(i,j)\, x_i \tag{21}$$

This total throughput occurs at times when the system component i is busy. If its utilisation–the fraction of time for which it is busy–is U_i then there will be U_i seconds in every elapsed second when kicks from it could occur at a rate $k(i,j)$. Hence this throughput can be written also as

$$U_i \sum_j{}' k(i,j)$$

and we have from eqn. 21

$$U_i = x_i\, g(N - 1)/g(N) \tag{22}$$

Thus the xs are proportional to the utilisations of the system components. Eqn. 22 shows also that the values of the Us as given by this relationship are unaffected by the absolute values of the xs and depend only on their relative sizes. For $g(n)$ is a homogeneous polynomial in the xs of degree n and therefore the right side of eqn. 22 is the quotient of two homogeneous polynomials of degree N and is unaffected if the same scaling factor is applied to every x. Thus we can choose the scale for the xs as we wish.

We can measure the work done by a system component during any period by the length of time during which it is busy in that period; the utilisation, as has been said, is the ratio of these two times. Thus if the utilisation is U_i we are getting U_i units of work from the component in every clock second. If we sum the U_i over all i, that is over all the components of the system, we get a measure of the total amount of work done by the complete system per second of clock or elapsed time. If this is greater than 1, as we hope it will be in a multiprocessing system, we have gained by overlapping activities and it is reasonable to use this sum as defining the *overlap factor*; as it will depend on the number of processes N that are available for overlapping, we can denote it by $f(N)$. Thus from eqn. 22 we have for the overlap factor

$$f(N) = \frac{g(N-1)}{g(N)} \sum_i x_i \tag{23}$$

Suppose now that we have a system running a given workload involving a number of actions for each component, and–for example in a sizing study–measurements of the numbers of actions and the service times for each. We can then calculate the total busy times for each component, to which the utilisations and therefore the xs will be proportional. Because of the independence of scale we can take these times as the actual values of the xs. The sum of these times will give what the total elapsed time would be if there were no overlapping and therefore the elapsed time corresponding to multiprocessing at level N will be this divided by the overlap factor $f(N)$. Thus from eqn. 23 we have, if T_N is the elapsed time for a given workload when the multiprogramming level is N, so that T_1 is the time when there is no overlapping of actions,

$$T_1/T_N = \sum_i U_i = f(N) = \left\{ g(N-1)/G(N) \right\} \sum_i x_i = g(N-1)/g(N)\, T_1$$

and therefore

$$T_N = g(N)/g(N-1) \tag{24}$$

This important relationship enables us to calculate elapsed times from a knowledge of the relative loadings of the system components. Since $g(n)$ is a homogeneous polynomial of degree n in the xs, T_N is of degree 1 in these variables and therefore if the xs, which meausre the loadings, are given in units of time, say seconds, then T_N will be in these same units. The possibility of actually performing such a calculation depends on the possibility of evaluating the polynomials $g(n)$, which at first sight would seem to be a considerable task for all but small systems. A technique can however be developed which leads to a quite manageable process and this is described in Section 7, where an example is given.

Given that the polynomials $g(n)$ can be evaluated without unreasonable labour, standard tables or graphs can be provided giving the overlap factor $f(N)$ as a function for a range of sets of relative loadings, that is, of the xs. Fig. 1 gives a set of curves corresponding to different loadings on 10 components. The method of using curves or tables such as these is as follows. From monitoring information we have, for a given system running a given workload, values of the elapsed time and for the utilisations or busy times for all the components; thus we can calculate the elapsed time T_1 corresponding to no multiprocessing (simply the sum of the separate busy times) and the overlap factor $f(N)$ as T_1 divided by the observed elapsed time, and finally from the curves, because we know the relative loadings, the actual level of multiprocessing N. We can then estimate the change in throughput that would result from altering a component service time or the CPU utilisation, using this value of N and appropriately adjusted xs; or the effect of a change in N resulting from an increase in main store, which increases the space available for programs.

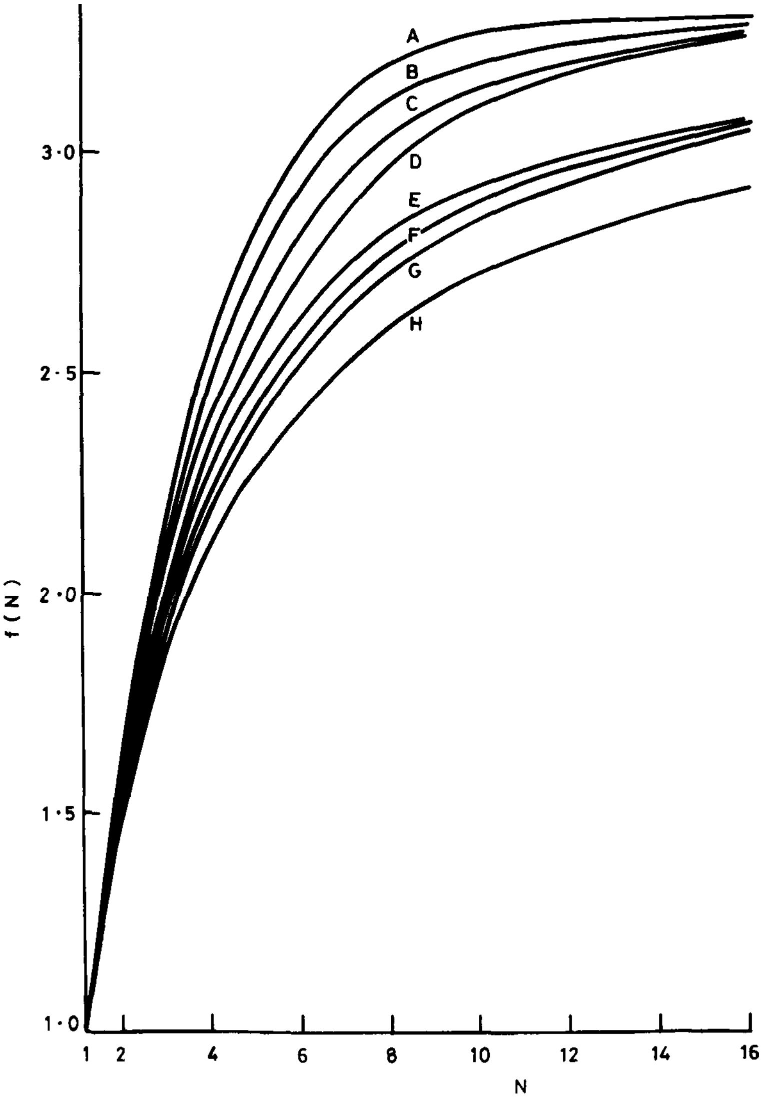


Fig. 1 Overlap factors f(N) giving increases in throughput for different loading patterns (x_1, x_2, . . ., x_{10}) (e.g. pattern H corresponds to $x_1 = x_2 = x_3 = 3$, $x_4 = 1$, $x_5 = x_6 = \ldots = x_{10} = 0$

A – (3, 1, 1, 1, 1, 1, 1, 1)	E – (3, 3, 1, 1, 1, 1)
B – (3, 2, 1, 1, 1, 1, 1)	F – (3, 3, 2, 2)
C – (3, 2, 2, 1, 1, 1)	G – (3, 3, 2, 2)
D – (3, 2, 2, 2, 1)	H – (3, 3, 3, 1)

This method of using the Jackson model appears to give very accurate extrapolations in practice, in spite of the inevitable inaccuracies there will be in the data, as for example in the assumptions about disc seek times. It seeks likely that the consequential error in the value estimated for N just about compensates for the effects of these inaccuracies. The method has the great advantage of giving a very high ratio of Enlightenment to Complexity. Curves such as Fig. 1 are simple to use and give a good idea of the range of variation of the overlap factor $f(N)$ for different mixes of loadings on the system components. If more accuracy is required this can be obtained by using a small BASIC program which has been written, or a programmable hand calculator such as the TI 59; or hand calculation of a simple recurrence relationship as will be described in Section 7.

7 Computational technique and network synthesis

The sums involved in $g(N-1)$ and $g(N)$ are unmanageable in general but with team games the team principle can be exploited to overcome this difficulty. The key concept is that of the *generating function.* If Q ranges over all vectors with nonnegative components without restriction and if the sum of these components for any given Q is $N(Q)$ then we define the generating function for the game to be

$$\gamma(z) = \sum_{Q} X(Q)\, z^{N(Q)} = \sum_{n=0}^{\infty} g(n)\, z^n \qquad (25)$$

The sum on the left is taken over all Q with nonnegative components and that on the right is obtained by grouping together all the terms for which $N(Q) = n$ for successive values of n.

We now define a *team generating function* $\gamma_t(z)$ for each team t; consider the notation introduced in Section 5: if for example team 1 consists of players 1,2,3 then its generating function is

$$\gamma_t(z) = X_1\,(Q_1, Q_2, Q_3)\, z^{Q_1 + Q_2 + Q_3}$$

Then the generating function for the game as a whole is the product of the generating functions for the several teams.

With the Jackson model there is only one player in each team and therefore there is one generating function for each player; hence for player i (representing component i in the system) the function is

$$\gamma_i(z) = \sum_{n=0}^{\infty} X_i\,(n) z^n \qquad (26)$$

and when the kicking rates are constant as in eqns. 15 and 16 this reduces to

$$\gamma_i(z) = \sum_{n=0}^{\infty} x_i^n z^n = (1 - x_i z)^{-1} \tag{27}$$

The procedure in practice is to build up the power series for the complete system by successive multiplication of the separate series for each component; in this particular case, where the kicking rates are constant, the process can be carried out very conveniently with the aid of a simple recurrence relation, as follows.

From eqn. 27 the generating function $\gamma(z)$ for the complete system is the product

$$\gamma(z) = \prod_{i=1}^{M} (1 - x_i z)^{-1} \tag{27a}$$

If we denote by $\gamma_m(z)$ the product of the first m terms (i.e. for the first m components) then

$$\gamma_m(z) = \gamma_{m-1}(z)(1 - x_m z)^{-1}$$

so

$$\gamma_m(z) = \gamma_{m-1}(z) + x_m z\, \gamma_m(z) \tag{28}$$

Thus if $g(m, n)$ is the coefficient of z^n in the series for $\gamma_m(z)$

$$g(m, n) = g(m-1, n) + x_m\, g(m, n-1) \tag{29}$$

We can start the recurrence with the known values $g(1, n) = x_1{}^n$ and $g(m, 0) = 1$, and proceed stepwise until we reach $g(M, n)$, which is the value $g(n)$ we require. The following example shows the process applied to a very simple case.

Example: Consider a system with three components: CPU, controller and disc. We wish to find the elapsed time for a workload made up of 3 units mill time, 1 unit controller time and 2 units disc time, allowing multiprocessing levels up to five. The actual size of the time units is irrelevant so long as it is the same for all components. We take the values 3, 1, 2 of the component usages as the values of x_1, x_2, x_3, respectively, and use the recurrence relationship to calculate successive values of $g(m, n)$ for $m = 1$ to 3 and $n = 1$ to 5. Here $g(N)$ is $g(3, N)$ and the elapsed time corresponding to $N = 5$ is $g(5)/g(4)$.

The starting values are $g(1, n) = x_1{}^n = 3^n$, $g(m, 0) = 1$ for all m. These give the first column and row, respectively, in Table 1, from which the table can be completed either by rows or by columns using the recurrence relationship given by eqn. 29: each new entry is obtained by multiplying the entry above it by the appropriate value x_m and adding the entry on the left.

Table 1

	$m=1$ $x_1=3$	$m=2$ $x_2=1$	$m=3$ $x_3=2$	T_n	$f(n)$
$n=0$	1	1	1		
1	3	4	6	6	1
2	9	13	25	4.167	1.440
3	27	40	90	3.600	1.667
4	82	121	301	3.344	1.794
5	243	364	966	3.209	1.870

The column $T_n = g(n)/g(n-1)$ is the elapsed time at a multiprocessing level n, and $f(n) = T_1/T_n$ is the overlap factor. T_1, the elapsed time with no multiprocessing, is the sum of the separate component times, Six in this case.

It is worth noting that the radius of convergence of the power series in eqn. 25 is the limit of the ratio $g(n-1)/g(n)$ as $n \rightarrow \infty$, which ratio is, from eqn. 24, the reciprocal of the throughput time T_n when the xs are the busy times for the separate components. However, since the series is the expansion of the product (eqn. 27*a*) this must be the value of z corresponding to the singularity of the product nearest to the origin, which is the reciprocal of the largest of the xs. This means that as the level of multiprocessing increases the elapsed time is increasingly dominated by the most heavily loaded component. For this reason the result has been known in ICL as the bottleneck theorem. The football analogy is that as the number of balls in the game is increased, the total activity is increasingly dominated by the slowest player.

The power-series or generating-function technique applies equally to the analysis of networks, such as communication networks. The basic principle, as here, is that if two or more networks are joined the behaviour of the network formed by their join is represented by the power series formed by multiplying together the series representing the separate networks.

8 Multiclass models: an introduction

The model described in Section 7 has treated all processes as statistically identical. This is clearly a vulnerable assumption and it would be useful to be able to investigate the extent to which the results are invalidated if the assumption is wrong. Multiclass models provide a way of doing this. Such models also allow the effect of giving different priorities to processes to be estimated in some cases.

It is possible however to go some way in this direction by means of a very simple argument. Consider a football game, such as in previous Section 7, in which there are N_1 black footballs and N_2 white. Suppose that the players always give absolute priority to the black balls over the white; then if we consider the black balls alone it is evident that they are quite unaffected by the white. Hence as before there will be an overlap factor $f(N_1)$ and the throughput will be increased by this factor compared with what it would be with only one ball in play. Suppose we now consider the whole game with both black and white balls treated equally. Since all processes are now identical it clearly makes no difference in which order the players kick the balls, so far as throughput is concerned. Hence the throughput of the $N_1 + N_2$ balls taken together is increased by the overlap factor $f(N_1 + N_2)$ over the singleball game and therefore the factor for the white balls alone is $f(N_1 + N_2) - f(N_1)$. Thus from graphs such as those in Fig. 1 we can easily see to what extent such universal overriding priority will have. For batch-processing systems generally, there will be differences in the priorities enjoyed by different processes on different components of the system.

Apart from the question of priorities in a first-come-first-served queue there is also the question of the service time distribution for real system components, which will not in general be negative exponential. Both these problems are dealt with in a multiclass model by using an idealised mathematical scheduling process to represent the way the server schedules the service of the customers. The simplest of these processes is the so-called process sharing (PS) algorithm. This assumes that the server allocates a slice of time to each customer in turn in a fixed sequence, but with different sized slices to customers in different priority classes, such as black and white in the previous Section. The sizes of the slices are then made to tend to zero while their relative sizes are kept constant. Thus a slice sp_c is given to each customer in class c where $\sum_c p_c = 1$, and then s is made to tend to zero. Such an algorithm has the remarkable property that if it is used to schedule an infinite random stream of requests arriving for service, with an arbitrary distribution of service times, then the emerging stream of processes is statistically identical to that which would have arisen from a negative-exponential distribution of service times. Thus the effect of slicing up the time by means of this limiting form of round-robin scheduler is to make an arbitrary distribution of service times appear negative-exponential.[8]

The sorts of algorithm used in practice by real CPU schedulers are only very roughly approximated by this idealised scheduler. Nevertheless it is of considerable interest that the idealised process produces perfect negative-exponential distributions. It argues that it is very likely that real schedulers have the effect of making effective service time distributions more nearly negative-exponential than they were before; and as CPU service times are known to be quite near to negative-exponential already this adds to our confidence in the reasonableness of the assumption with regard to the CPU.

Suppose now that there are two classes of customer corresponding to $c = 1$ and 2, with Q_c members in class c, and a server whose total service rate k is apportioned by the PS algorithm between the two classes in amounts k_c, where $k_1 + k_2 = k$. The probability that the next slice of time will be for class c will be $Q_c/(Q_1 + Q_2)$ and that service is given in the next interval dt is $k_c\, dt$. Hence the service rate for a member of class c is

$$k_c Q_c / (Q_1 + Q_2) \qquad (30)$$

We can represent this situation by a football model in which the server is represented by a team of two players corresponding to the two queues for the single PS server. These two players share their kicking power between them. We give them a team function as follows. If the team is numbered t then, again with the notation of Section 5, we have the form

$$X_t(Q_1, Q_2) = \frac{(Q_1 + Q_2)!}{Q_1!\, Q_2!} x_1^{Q_1} x_2^{Q_2} \tag{31}$$

from which we see that eqn. 12 gives for the kicking rate for class c

$$\{f(c, f)/x_c\} Q_c/(Q_1 + Q_2) \tag{32}$$

as follows for $c = 1$ for example by putting $X_t(Q - I) = X_t(Q_1 - 1, Q_2)$.

Thus if we identify $f(c, f)/x_c$ with k_c we have a model with the required service rates as functions of queue lengths.

In general, for more than two classes of customer we shall have a team function that is a multinomial in the xs instead of the binomial in eqn. 31. Such an arrangement will suffice to represent a server serving several classes of customer, each of which requires service to an extent which is statistically distributed with a distribution depending on the class.

A model may contain two types of servers: multiclass PS servers as above with general service-time distributions, and servers serving a single class of customer for which the service-time distribution is negative-exponential but to which a different, so far unspecified, queue discipline applies. For example, consider a batch/MAC system comprising a CPU which handles both batch and MAC processes, a peripheral A which handles only batch and a peripheral B which handles only MAC. Assume that the CPU is PS-scheduled with arbitrary service-time distributions for batch and MAC, and that both peripherals have negative-exponential service times and first-in-first-out (FIFO) queue disciplines.

Suppose further that it is possible for a process to move from any valid combination of server and class to any other through some chain of such combinations. This is possible if batch processes leaving the CPU can occasionally be brought online as MAC jobs and correspondingly if MAC jobs can be put in the background. Thus processes can change class at the CPU provided that they go to the peripheral appropriate to the new class and similarly processes arriving at the CPU can change class before being scheduled. This proviso is necessary to ensure that there is always a path from any state Q of the system to any other through intermediate states joined by nonzero flows. The complications that result if this condition is not fulfilled are discussed in Section 9.

Given all these conditions, the system can be mapped on to a football game as follows:

CPU team (2 players)	player 1	PS-scheduled batch processes
	player 2	PS-scheduled MAC processes

Peripheral A team	player 3	batch processes, negative exponential first-in-first-out
Peripheral B team	player 4	MAC processes, negative exponential first-in-first-out

Then eqn. 11 takes the form

$$X(Q_1, Q_2, Q_3, Q_4) = \frac{(Q_1 + Q_2)!}{Q_1!\ Q_2!} x_1^{Q_1} x_2^{Q_2} x_3^{Q_3} x_4^{Q_4} \tag{33}$$

As with the single-class model of the previous Section the xs can be taken as being equal to the work done by the players. Thus x_1 can be taken as the CPU utilisation time for batch processes and x_2 as that for MAC.

Eqn. 33 enables the probability distributions of queue lengths to be determined, along lines similar to those to be given in Section 10 below. The relevant generating function, as in Section 6, is

$$\begin{aligned}\gamma(z) &= \sum X(Q_1, Q_2, Q_3, Q_4)\, z^{Q_1 + Q_2 + Q_3 + Q_4} \\ &= \{1 - (x_1 + x_2) z\}^{-1} (1 - x_3 z)^{-1} (1 - x_4 z)^{-1}\end{aligned} \tag{34}$$

This generating function is exactly that which would be obtained with a single-class model as described in Section 7, with loadings $x_1 + x_2$, x_3, x_4 for the three components. The analysis shows therefore that it is possible to derive the same generating function, and hence the same throughput, with the rather different assumptions of the multiclass model.

The analysis extends straightforwardly to systems with any number of components and any number of classes at the PS scheduled components. There is however the proviso mentioned earlier, that there must be a route between any allowable combination of class and component and any other; thus we could not apply this model as it stands to a case where MAC processes must remain forever MAC and batch remain forever batch. An extended version is required for this which is discussed in the next Section.

9 Multi-class models: the degenerate case

We deal here with the situation known as the degenerate case, in which it can no longer be assumed that there is a path between any pair of states of the system without restriction. Only the general lines of the analysis are given, and the algebraic detail omitted in the interests of brevity.

Consider first the Jackson model (Section 5, eqn. 14) with constant kicking rates; the time spent by a single ball at the feet of the players is given by the solution of eqns. 17:

$$\sum_j k(i, j)\, x_i = \sum_j k(j, i)\, x_j$$

Suppose now that the game comprises two groups of players who never kick to each other, so that if there is only one ball in play it will remain always in one group or the other. In this case the equations will have two linearly independent solutions.

Next suppose that we give these two groups, which we shall call *subgames*, N_1 and N_2 balls. Then a particular state of the queues is feasible only if the sums of the lengths of the queues in the two subgames are N_1 and N_2, respectively.

With this revised definition of feasibility the analysis of Sections 4 and 5 remains valid. The normalising factor G is now calculated by summing $X(Q)$ over all feasible Q in the new sense; and the concurrency of the system is described by the pair (N_1, N_2) which can be conveniently denoted as a vector N.

As in Section 6, throughput can be calculated with the aid of a generating function. Now however we need a bivariate function of the form

$$\gamma(z_1, z_2) = \sum X(Q)\, z_1^{N_1(Q)}\, z_2^{N_2(Q)}$$

$$= \sum_{n_1=0}^{\infty} \sum_{n_2=0}^{\infty} g(n_1, n_2)\, z_1^{n_1}\, z_2^{n_2} \qquad (35)$$

When we come to calculate the throughput we find that $X(Q - I)$ in eqn. 19 is equal either to $g(N_1 - 1, N_2)$ or to $g(N_1, N_2 - 1)$ according to whether player i is a member of subgame 1 or 2. We then find that in place of the single overlap factor $g(N - 1)/g(N)$ we have two such factors $g(N_1 - 1, N_2)/g(N)$ and $g(N_1, N_2 - 1)/g(N)$ which give the throughputs in the two subgames compared with what these would be if there were only one ball in play in each.

For example, if for the system described on p. 163 there was no possibility of processes changing class at the CPU we should have a system with concurrencies constrained as follows:

Batch concurrency in sub-game 1	$Q_1 + Q_3 = N_1$
MAC concurrency in sub-game 2	$Q_2 + Q_4 = N_2$

The generating function is obtained by summing the eqn. 35, using eqn. 33 to give X. The result is

$$\gamma(z_1, z_2) = 1 \dagger (x_1 z_1 + x_2 z_2)^{-1} (1 - x_3 z_1)^{-1} (1 - x_4 z_2)^{-1} \qquad (36)$$

which may be compared with eqn. 34.

In general, a degenerate multiclass model has a generating function in as many variables as there are subgames. Such functions can be built up as described in Section 6 by multiplying together the appropriate multivariate power series. This principle of summarising all the relevant information about a system or network of

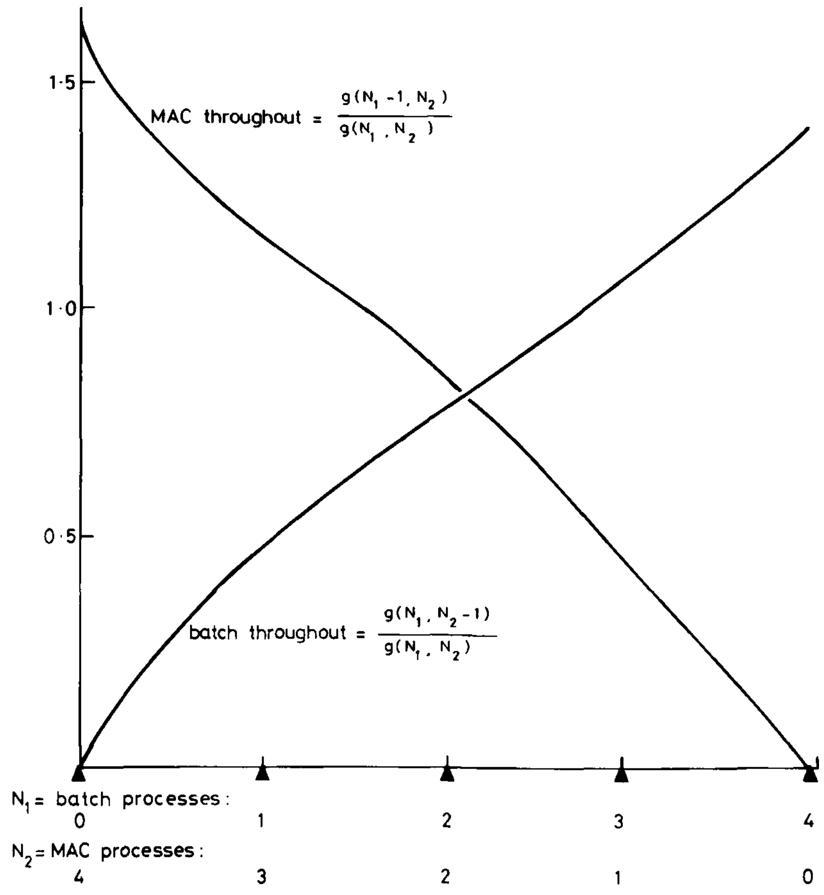


Fig. 2 Throughputs of MAC and batch processing (relative to throughput when run single-processing in isolation) when a total concurrency of 4 is variously split between batch and MAC
Assumed loadings (time units)
Batch x_1 = CPU = 0°7, x_3 = peripheral 'A' = 0•3
MAC x_3 = CPU = 0◦5, x_4 = peripheral 'B' = 0◦5

Degenerate multicalss model as in eqn. 36

components in the form of a power series or array of coefficients is often referred to in the literature as Norton's collapse, by analogy with the process of reducing an electrical network to an equivalent circuit using Norton's theorem.[9]

As an illustration of the use of this form of the model, Fig. 2 shows the results of the calculation of the relative throughputs of batch and MAC work for the system whose generating function is given by eqn. 36. It is assumed that if the batch load is run on its own on a single-programming basis the CPU utilisation is 70%, so that on average it is busy for 0•7 s of every second of elapsed time; and the peripheral A is busy for 0•3 s. Thus x_1 = 0•7, x_3 = 0•3. Similarly if for the MAC load we assume

50% loading of the CPU when this load is run on its own with single programming, $x_2 = x_4 = 0{\cdot}5$.

Given these values we can calculate the relative throughput factors

$$\frac{g(N_1 - 1, N_2)}{g(N)} \quad \text{and} \quad \frac{g(N_1, N_2 - 1)}{g(N)}$$

for any pair N_1, N_2. Fig. 2 shows the results for a total multiprogramming level $N_1 + N_2 = 4$.

It will be noticed that the curves are very nearly straight lines, and that therefore we should not have been seriously in error had we calculated the results by simple linear interpolation between extremes of pure batch and pure MAC. This is a valuable conclusion because the throughputs for these extremes can be calculated using the simpler single-class model.

For a fuller treatment of the theory the key papers are those listed in References 9 and 10.

10 Queue lengths

So far all results have concerned throughput; but there are many situations in which we are interested in response times and therefore in queue lengths. The models that have been described involve representing the behaviour of system components by power series which are multiplied together to give a representation (generating function) of the complete system. It turns out that the mean queue lengths can be obtained quite simply from these same power series.

With the Jackson model the probability of the system being in a state Q is given by eqn. 16. The mean length of queue for the ith player in the game is the sum of $Q_i\, P(Q)$ over all feasible Q; and it will be seen from eqn. 16 that this is the result of applying the operator $x_i \partial/\partial x_i$ to the right side of the equation and summing over Q. We therefore get the result we are seeking by applying the operator

$$\frac{x_i}{G} \frac{\partial}{\partial x_i}$$

to the generating function

$$\gamma(z) = \prod_i (1 - x_i z)^{-1}$$

and finding the coefficient of z^N.

Similarly, for the degenerate multiclass model described in Section 9 we find that the mean queue length in subgame 1 is the coefficient of

$$z_1^{N_1} \; z_2^{N_2}$$

in the expansion of

$$(z_1 \, \partial/x_i)\, \gamma\,(z_1, z_2)/g\,(N_1, N_2)$$

11 Implementation and conclusions

The practical application of the models described involves a simple algebraic technique, the multiplication of power series. This can be done with the aid of a computer program which is not at all difficult to provide, and which comprises a small package of Algol procedures for addition, multiplication, division and scaling of power series. The extensions required in going from single-class to multiclass models are simply the replacement of single DO loops by multiple loops to match the number of subgames involved.

For throughput calculations, single-class models are generally quite adequate and very many such calculations have been carried out routinely in ICL for sizing purposes. Moreover, this type of model gives a great deal of insight into system behaviour for very modest complexity.

For the analysis of response time with multiple classes of work, a multiclass model may be used, but the increase in complexity is not a negligible factor. A very considerable benefit from such multiclass models, however, is the light they throw on the limitations of simpler models. The techniques have been used in the USA to examine the effects of giving different resource priorities to different classes of message; and for this purpose some very extensive software systems have been created for synthesising and analysing large networks. It could well be that, with Viewdata services for example, such tools will prove to be needed for analysis of response-time problems when the service eventually runs out of resources and it becomes necessary to study the consequences of different possible methods for resource allocation.

Acknowledgments

The author wishes to acknowledge the help and encouragement of Dr. J.M.M. Pinkerton and Mr. D.H. Carpenter of ICL in the work reported in this paper, and also helpful discussions with Mr. T. Giammo of the US Health, Education & Welfare Department, Baltimore.

References

1 KONIGSBERG, E.: *Oper. Res.*, 1958, 9, (1)
2 JACKSON, J.R.: *Manage. Sci.*, 1963, 1
3 BUZEN, J.P.: 'Computational algorithms for closed queuing networks with exponential servers', *Commun. ACM*, 1973, 16, pp. 527–531
4 DENNING, P.J., and BUZEN, J.P.: 'Operational analysis of queuing networks. Modelling and performance evaluation of computer systems'. Proceedings of the 3rd International Symposium, IRIA–Laboria IFIP Working Group 7.3. North-Holland
5 BERNERS-LEE, C.M.: British Computer Society Conference on Performance Evaluation, University of Surrey, 1972
6 SENNET, C.: British Computer Society Conference on Performance Evaluation, University of Surrey, 1972
7 HUGHES, P.H., and MOE, G.: 1973 National Computer Conference AFIPS Proceedings, 42, p. 109
8 KLEINROCK: *Queuing systems*, Vol. 2, Chap. 4

9 CHEUDY, K.M., HERZOG, U., and WOO, L.: 'Parametric analysis of queuing network models', *IBM J. Res. & Dev.*, 1975, **19**, (1)

10 BASKETT, F., CHEUDY, K.M., MUNTZ, R.R., and PALACIOS, F.G.: 'Open, closed and mixed networks of queues with different classes of customer', *J. Assoc. Comput. Mach.* 1975, **22**, (2)

Appendix
ACM Sigmetrics Workshop on the Theory and Application of Analytic Models to ADP System Performance Prediction
University of Maryland, 12-13 March 1979

This 2-day meeting was a sequel to one held two years ago under the auspices of the US Federal Computer Performance Evaluation and Simulation Center (FEDSIM). It was a small meeting intended for specialists, with an attendance of about 150. The highlights were an overview by Professor James Browne of the University of Texas, a debate on the validity of a fresh approach to queuing theory known as operational analysis, a very good paper on the application of what we know as FAST to a large Univac system at Bell Labs and an excellent paper by one of the implementers of IBM's packages for the solution of queuing problems in networks.

1 Theory overview

Browne's group at Texas has been closely connected with the development of queuing-network models from the start. The group also had close links with a similar group at IBM Yorktown Heights. In the period 1965-1972 the performance models in use were comparatively primitive, being restricted to central server models with exponential service times and a single job class. During the period 1972-1976 extensions to include queuing networks and multiple job classes took place. Hierarchical methods were introduced, as were methods for dealing with state-dependent routings, extended servers and mixed networks. Since then, however, progress has been slow. In particular, the vexed problem of blocked networks – where the customer cannot move on to the next server because there is no room – has proved much more difficult than had been expected. Approximate methods are, however, making progress, as are hybrid analytic-simulation methods.

An important new area of work is in the optimisation of networks: the problem here is not how to evaluate the network but how to establish that the performance surface is convex with a unique optimum.

All the American work reflects the prevalence of large communication networks and is rather less concerned than we are with system performance.

2 Operational analysis

This is an ingenious but controversial idea for simplifying the derivation of queuing network theory, due to Denning and Buzen (Buzen was using FAST – type models before 1972: see References 3 and 4 in the main paper). The idea may be explained as follows. Consider a simple negative-exponential server serving a stream of customers – the simplest queuing situation. The usual way of treating this is to assume negative-exponential service times and random arrivals, from which it can be deduced that the average rate of service when the queue length is n, i.e.

$$\frac{\text{number of customers served when queue length is } n}{\text{time during which queue length is } n}$$

is constant, independent of n; and the usual results for the probability distribution of queue lengths follow. The snag is in the assumption of exponential service times, which do not always happen.

Buzen gets around the difficulty by assuming directly that this service rate is constant, in which case the rate is said to be homogeneous. It is theoretically possible to have a situation in which service rates are homogeneous even though the distribution is not negative-exponential. For example, if we take a negative-exponential system and distort the time scale in some arbitrary manner, the resulting distortion will upset the negative-exponential form but not necessarily the homogeneity. The assumption of homogeneity makes it possible to explain the derivation of queue-length distribution very simply; but the snag here is that there is no more evidence for homogeneity than there is for the negative-exponential form, although in principle it would be easier to gather such evidence. There is also another objection, raised by Professor Sevcik of Toronto, that, although homogeniety is sufficient for the derivation of the throughput of a system, it is not sufficient for the derivation of the response times, for which a service-time distribution must be assumed.

It should be stressed that in practice the operational analysis approach yields the same results as the older Markov approach in all cases for which it is appropriate. Section 3 of the main paper refers to this.

3 *Bell Laboratories*

There were two papers from Bell of which the second, by Sheldon Becker, gave a convincing account of the benefits of using a simple FAST model for predicting the performance of a very large real-time inventory system running on a Univac system. As others have found, much the most difficult part of the job is getting the model data right. Becker concluded that 'if we give the models the right input we do extremely well'.

They also used event simulation but found that the run times (15 min) were too long to permit an adequate exploration of the performance space.

4 *Queuing-network solution packages*

Both IBM and the University of Texas have implemented large packages for executing the algorithms for single-class and multiclass models. Rudi Downs of IBM had helped Martin Reiser (whose name appears in the literature in connection with computational methods) to implement the QNET4 and RESQ packages. These provide an interactive means of setting up the data in a user-orientated language. Furthermore, RESQ provides the means for including results of simulation of particular components or subsystems in the analytical model.

An interesting feature of the numerous applications illustrated was the occurrence of studies of queuing for access to code in software systems. Similar studies have been carried out in ICL in relation to queuing for semaphore-protected code in software.

5 *Hand-held calculators*

Interest was expressed in the ICL TI 59 programs for hand calculators which offer an alternative to the use of a large package. This alternative is quite suitable in many cases.

Advanced technology in printing: the laser printer

A.J. Keen
ICL Product Development Group, Northern Development Division, Kidsgrove, Staffs.

Abstract

The electro-photographic process used by some recently produced high-performance printers is described. The basis of the process is the use of the modulated beam from a laser to write the information that is to be printed as a pattern of discharged areas on the charged photoconductive surface of a drum; the visible record is then produced by using this pattern to control the attraction of a coloured powder (toner) to the paper. Very high quality of printing and an unlimited range of character fonts can be achieved, with speeds of over 10,000 lines per minute. Some technical and other details given in the paper relate to the ICL LPS-14 laser printing system which uses this process.

1 Nonimpact printers

For many years now the familiar impact printer, in which the impression is formed by causing a hammer to strike a metal relief of the appropriate character, has met the needs for printing of computer output, offering reliability, good quality of print and adequate speed. However, demands already being made, and certain to arise more frequently in the future, exceed the capabilities of impact printers in speed, quality of print and versatility of character repertoire. Nonimpact printers, often referred to as NIPs, are freed from the restrictions inherent in mechanical printers and can meet these demands at an economic cost.

As the name implies, nonimpact printers do not use precast type to form the impression on the paper; several technologies are available of which the one to be described here is the electrophotographic process. One of its advantages is that it allows the use of standard fanfold computer paper, whether preprinted or plain; this compatibility with conventional printers makes it easy for the user to change from one type to another. As with all nonimpact printers, only single-part paper can be used, but the very high speed makes it practical to run off as many copies of any document as are required, with the extra advantage that all are equally legible; the elimination of multipart paper is itself an advantage. The process also makes it simple to produce any design of standard form by printing directly on to the plain paper and thus to reduce the need to hold stocks of pre-printed stationery. How this is done will be described below, as will the means by which high resolution (on which the quality of the print depends) and the great flexibility of character repertoire are achieved.

Several printers using the electrophotographic process are now available, including the Siemens 2500, the IBM 3800 and the Xerox 9700. ICL has introduced an off-line system LPS-14, which prints from magnetic tape at speeds up to 21,000 lines per minute. All use lasers in the process and offer similar performance and facilities.

2 The electrophotographic process

The essence of the process is shown in Fig. 1, which describes the LPS-14 system. The functions of the various components will be explained as we go through the process. In summary the process is as follows.

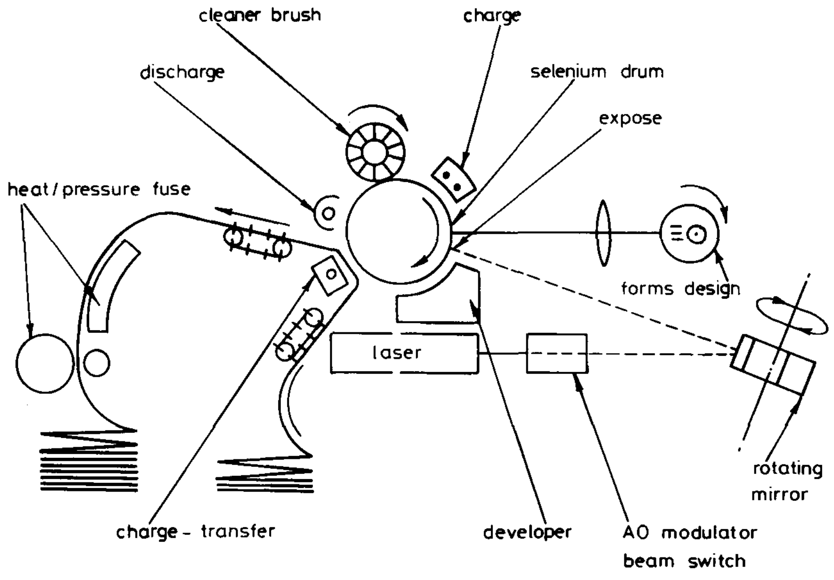


Fig. 1 LPS-14 system

The surface of the drum is coated with a photoconducting material and is charged to a potential of several hundred volts. The beam from the laser passes through the transparent modulator to the surface of the rotating mirror from where it is reflected to strike the drum surface. The mirror is multifaceted and as it rotates the beam sweeps across the drum and returns to its starting point for the next sweep; the photoconductive property of the drum surface is that charge leaks away where light falls on to it, so if the beam intensity were steady during the sweep a line would be 'written' across the drum in the form of an area of discharge, and a series of parallel lines as the drum rotates. The modulator effectively switches the beam on and off; it is a block of high-refractive-index glass with an acoustic transducer on one face; high-frequency pulses sent to the transducer generate acoustic waves in the glass which change the refractive index as they travel and thus deflect the beam. It is arranged that only the deflected beam, not the undeflected, strikes the drum, so a sequence of pulses fed into the modulator is reproduced as a pattern of discharged spots along a line on the drum surface. Finally for this part, information to be printed is read into the printer's buffer – either directly from the computer or from tape or disc – and from there controls the modulator; and so produces a representation of itself as a pattern of discharged spots written along lines on the charged surface of the drum.

The remaining stage is the printing from this electrostatic image, and, in principle, this is done by using it to cause what is called toner powder to adhere to the paper; the toner is a thermoplastic material with a carbon-black pigment and might be called a dry ink. It is attracted to the discharged areas of the drum surface, thus making visible the electrostatic pattern, and as the drum rotates it comes into contact with the paper, to which it is transferred. The paper then goes through heated rollers to make the impression permanent, any remaining toner on the drum is removed by the cleaner brush shown in the diagram, the electrostatic pattern removed, and the process repeated for the next piece of information in the buffer.

The process is continuous and is of constant velocity: i.e. printing continues without interruption so long as there is information in the buffer and the time to print a page is independent of what is printed. A typical rate of paper feed is 145 ft/min (45 m/min). With a line spacing of 6 per inch this gives a printing speed a little over 10,000 lines per minute, and over 20,000 lines per minute at 12 lines per inch spacing. Maximum efficiency is obviously attained by putting as much information on a single page as possible and it is perhaps better to regard the device as a page printer rather than as a line printer.

It would be reasonable to ask, why use a laser rather than any other light source? The answer is that the properties of laser light allow very high precision to be attained in the optics of the system and high energy input at the drum surface; and that both these are needed to get quality and performance required.

3 Some details

3.1 Photoconductive process

The photoconductive material used in the drum coating is selenium (Se). In its pure amorphous (i.e. noncrystalline) form it will retain an electrostatic charge for a considerable time in darkness and when it is exposed to light the charge is conducted away at a rate depending on the light intensity. Both purity of the material and the amorphous form are essential to the electrophotographic process, because otherwise there will be irregularities in the surface charge and losses by conduction which will degrade the photoconductive function.

Crystallisation of amorphous selenium commences at temperatures above 50°C and with photon energies above 2·2 eV. Peak photosensitivity is near to 0·4 μm wavelength, but here photon energies are above 2·2 eV. Addition of arsenic (As) extends the photosensitivity towards the orange–red band, where photon energies are below the crystallisation threshold, and also raises the temperature threshold above the 50°C. Arsenic also increases the hardness of the photoconductive layer and its resistance to the abrasion that is inherent in the complete process. The general practice is to use arsenic selenide ($As_2 Se_3$) and a helium-neon laser, which gives light of wavelength 0·633 μm for illumination. The selenide is deposited in vacuum to a thickness of 50 μm under controlled conditions on the surface of an aluminium cylinder on which a film of aluminium oxide has previously been deposited to act as an electron barrier.

Further difficulties arise because of the semiconducting properties of selenium. In the printer the initial charge on the drum will decay even in darkness because of injection of electrons from the base conductor, despite the electron barrier; and after a rapid charge-discharge cycle this decay will increase, owing to the release of conduction carriers previously trapped at impurity sites during discharge. This latter effect also prevents the complete discharge of the desired areas during exposure. Consequently a high charging voltage is necessary, to ensure that with these limitations a great enough difference in potential between charged and discharged areas may be obtained. It is found that the initial charge must be to at least 700 V. This allows a difference of at least 300 V between the two areas, which provides adequately strong fields for the attraction of the toner particles. It is found that a 50 μm selenium layer can be positively charged to about 1000 V before rapid breakdown in its resistivity occurs, and this sets the limit in the practical case.

Charging and discharging: The charge mechanism makes use of the corona emission from a thin taut platinum wire running axially through an approximately cylindrical hollow conductor with a high voltage (over 4 kV) applied between the wire and the conductor. As indicated in Fig. 1 the actual arrangement is that a longitudinal strip along the drum surface forms part of the conducting shell, and charge is deposited on this as it passes under the wire. A first approximation to the rate of charge deposition, and hence to the potential reached, can be calculated by treating the shell as a cylinder, which is good enough to guide the design of the device. With a drum surface speed of 75 cm/s* a 12 mm section will pass under the wire in approximately 16 ms and will reach a potential of around 100 V. Thus, in order to reach the 700 V or over that is required the charging device has to have a number of parallel wires and cover a wider section of the drum.

Again as a first approximation, the rate of discharge of an area of the drum surface exposed to light can be taken as proportional to the energy intensity of the incident light. This is not however a good approximation and is considerably modified by effects due to the semiconductor properties of the selenium; a full treatment is given in Reference 3. With the values already quoted it is found that the incident light energy must be around 4·5 W/cm^2.

3.2 Optical system

The optical system is shown in Fig. 2. As was explained in Section 2 the beam path can be deflected as the beam goes through the modulator and the geometry is arranged so that only the deflected beam strikes the drum. The light emerging

* This corresponds to the paper speed of 145 ft/min given above. The mixture of metric and imperial units is regrettable but difficult to avoid, because, while the design of the machine is based on metric dimensions, printing standards such as line and character spacing are defined in imperial units, such as lines per inch.

from the laser is polarised and the beam is about 1 mm in diameter and is closely collimated. For efficient operation of the modulator the beam diameter must be reduced, which is the reason for the 'beam converger' in Fig. 2, and the system must be set up so that the beam has a defined angle of incidence and a defined direction of the plane of polarisation. With proper setting an efficiency of 80% can be achieved. The transducer generates acoustic waves in the modulator of 50 MHz, which deflect the beam by about 6 mrad, which gives a linear deflection of about 0•15 in at the drum surface. After passing through the modulator the beam is expanded before being re-focussed at the drum surface to give the required size of dot.

The mirror will typically have 12 facets; in this case it must rotate at approximately 22,000 rev/min to give the required scanning rate.

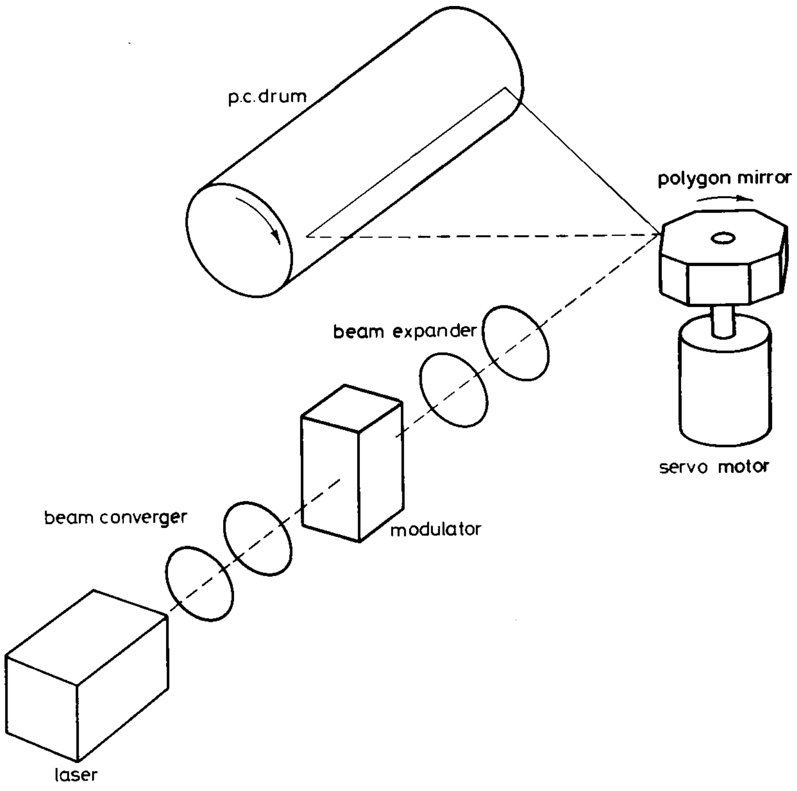


Fig. 2 Detailed structure of image written on drum

The detailed structure of the image written on the drum is shown in Fig. 3. Each switching of the laser beam writes a dot, as a discharged area, on the charged surface. The dot diameter is 0•010 in and the spacing along the swept line is 0•0055 in; thus adjacent dots overlap and give the effect of continuous line. The spacing between successive lines is 0•00694 in, again giving overlap. These dimensions are equivalent to resolutions of 180 per inch horizontally and 144 per inch vertically, which give excellent character definition meeting the requirements of the ECMA-11 standard.[4] The scanning rate is determined by the line spacing and the speed of rotation of the drum; here it is approximately 4,200 scans per second.

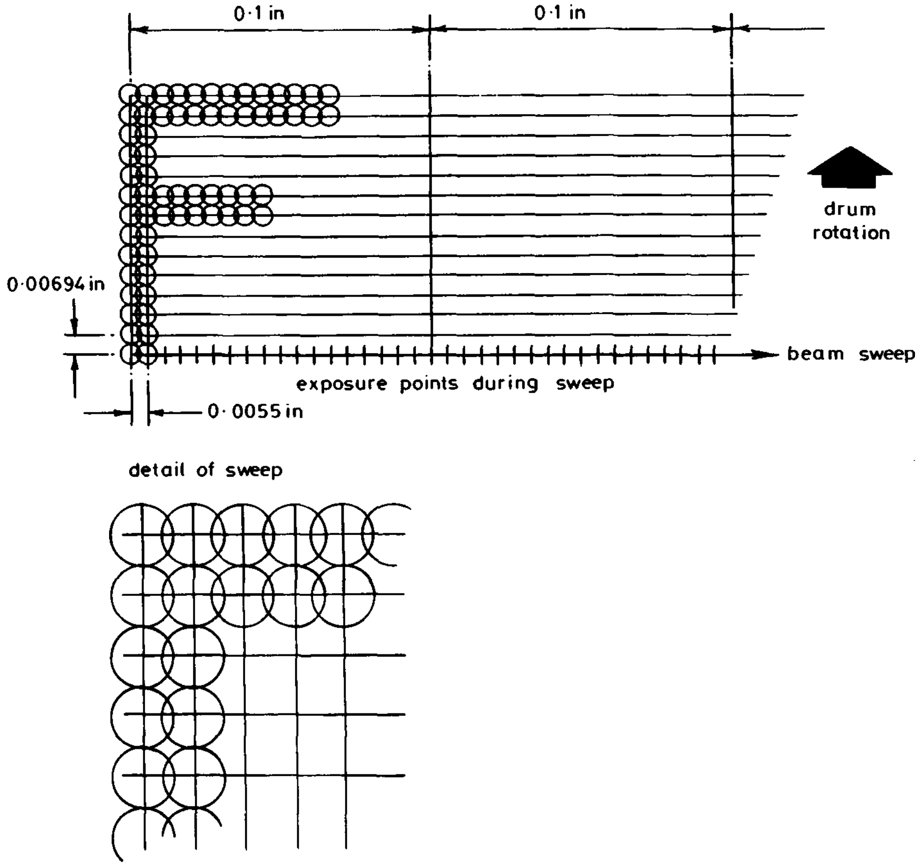


Fig. 3 Optical system

The modulator, which is controlled by the information held in the printer's buffer, can be turned on every 80 ns, which is therefore the rate at which the dots can be written on the drum. If we allow for the rise and fall times in the modulator switching, each dot is exposed to the full intensity of the beam for about 35 ns. The incident light energy necessary for effective writing was given in Section 3.1 as about 4·5 W/cm^2, which, for a dot of diameter 0·010 in (0·0254 cm) implies a total input over the dot area of about 2·5 mW; this has to be increased to about 4·0 mW to allow for losses in the optical system. Thus the laser beam has to be of at least 4 mW.

3.3. The printing process

As was explained in Section 2 the visible image is produced by causing particles of a coloured powder, the toner, to adhere to the discharged areas of the drum. There is an electrostatic field between these relatively negative areas and the unexposed areas, which still carry the full positive charge. The field is confined effectively to within a very short distance (about 30 μm) of the surface. This implies that the toner particles must be brought into intimate contact with the drum surface and also that to be attracted by the field they must carry a positive charge.

In the development chamber (Fig. 1) the toner is mixed with carrier material which consists of resin-coated magnetic beads of about 300 μm in diameter. The mixing action causes a charge transfer between the toner and the carrier beads, the toner becoming positively charged and being held to the carriers by mutual attraction. The mixture is conveyed to the drum by rollers rotating in a stationary magnetic field and on which the carriers, acted on by this field, assume a brush-fibre stance and are wiped against the drum. Toner particles will be transferred to the drum provided that the field force attracting them exceeds that holding them to the carriers. The latter depends on the proportion of toner in the mix, which is therefore a critical parameter and is controlled automatically by a monitoring process which is described later in this Section.

The next stage is that the drum surface, now carrying toner powder held to the discharged areas, comes into contact with the paper on which the information is to be printed. Transfer of the toner from the drum to the paper is brought about by applying a negative charge to the paper, sufficient to overcome the attraction to the drum. This is done by using a single-wire corotron operated so as to give peak corona current. The toner is only loosely held to the paper at this stage and the image is made permanent with the application of heat and pressure to the paper, by passing it over a heated platen and then through heated rollers. The fusing temperature of the thermoplastic material of the toner is around 130°C. To ensure proper fusing in the short times available at the speeds of printing being achieved, the rollers are heated to about 195°C and have a coating of a polymer which deforms under pressure, thus increasing the area and consequentially the time of contact with the paper.

For the monitoring referred to above, a small all-black square is printed at the edge of each sheet and its density measured. If this falls outside preset limits the comparison is used to adjust the proportion of toner in the toner-carrier. The carriers are returned to the developing chamber after each printing and reused; in time they become permanently coated with fine particles of toner and lose their effectiveness, when they have to be replaced; this occurs after about a million sheets have been printed. It will be clear from the description given in the preceding paragraphs that the complete system works to very tight tolerances, and that therefore a comprehensive range of testing and diagnostic facilities must be provided.

Forms-design printing: This term refers to 'overlay printing' of a standard form onto each plain sheet on which the information is printed; an invoice form is an example. There are two ways of doing this. One is to use a subsidiary drum around which a transparency is wrapped and from which an image of the form is cast on the photoconductive drum just before the region where the laser beam is to write the data. This is shown in Fig. 1 as the 'forms design' component. The other is to code the description of the form and to read this into the printer's buffer so that it is written on to the drum along with the relevant data. The second method has the advantages of not requiring any manual intervention and of allowing many different designs to be stored and selected as required.

4 Summary and Conclusion

The electrophotographic process has been developed around photoconductive materials previously proven in document copiers. A multiwire corona charging device is used to charge the photoconductor uniformly to a sufficiently high voltage such that laser exposure of computer-generated information in dot-matrix patterns will create consistent electrostatic images on the photoconductor. These images are developed by magnetic-brush application of toner powder. After transfer to paper the toner images are made permanent by hot-roll fixing.

The combined techniques used in the process enable print images to be generated at over 20,000 lines per minute and transferred to standard computer paper. The formation of characters in dot-matrix patterns allows multiple sets of character shapes to be stored and selected for use. It also enables character modification by which user-designated characters may be printed. Online forms generation using either transparancy projection or laser imaging of forms-design data can replace the need for preprinted stationery.

To the user the nonimpact printer introduces new standards of speed, quality and flexibility in computer printing.

References

1 DESSAUER, J.H., and CLARK, H.E. (Eds.) *Xerography and related processes* (Focal Press, 1965)
2 SCHAFFERT, R.M. : *Electro-photography* (Focal Press, 1965)
3 LI, H.T., and REGENSBURGER, P.J. : *J. Appl. Phys.*, 1963, 34, p.1730
4 ECMA (European Computer Manufacturers Association). Standard 11 for the alphanumeric character set OCR-B for optical character recognition. 1965

The new frontier: three essays on job control

D.W. Barron
Professor of Computer Studies, Faculty of Mathematical Studies,
University of Southampton

'Man is born free but is everywhere in chains', said Rousseau. A recent candidate in a computer-science degree examination put it more succinctly: 'The function of the software is to prevent people using the hardware'. On anything bigger than a microcomputer the user interacts not with the hardware but with the operating system, and the operating system interface that most concerns him is the job-control language. The horrors of job-control language have been extensively documented, and there is no doubt that this is the least satisfactory aspect of operating-system technology. In these essays we look at some current developments in this field and make some suggestions as to how the situation can be alleviated. We first explore the topic of standardisation.

1 Is a common job-control language both possible and desirable?

In many ways the current situation in job control resembles the state of programming before the advent of high-level languages. Just as each computer had its own assembly language, making program transfer very difficult, so each operating system has its own job-control language (JCL). Like the assembly languages, these are incompatible; like the assembly languages they have, in general, a minimal structure and an impoverished syntax, and they are, if that is possible, even more obscure and error prone than assembly language ever was. (The consequences of inserting a single blank in a line of JCL for IBM's operating system are truly awful to contemplate.)

In the programming field great advances have been made by the introduction of machine-independent (on the whole) standardised (more-or-less) high-level languages, and it is tempting to suppose that similar benefits might accrue in the field of job control. Many national and international organisations are pursuing this line: an incomplete list includes the British Computer Society, the Dutch JCL Committee, IFIP (. . . and now, from the people who brought you Algol 68 . . .), ANSI (first FORTRAN 77, now . . .), and CODASYL. (This last organisation claims that it plans to do for JCL what COBOL did for programming languages: a terrifying prospect.)

The problems confronting anyone attempting to design a machine-independent job-control language are formidable. It is not the syntax that causes the trouble, it is the semantics. It is a cardinal principle of language design that you should decide what you want to say before you think about how you are going to say it, and in job control it is not entirely clear (and certainly not universally agreed) what is the fundamental universe of discourse. Apart from the superficial differences of external form, JCLs for existing operating systems show clearly that they are predicated on very different models of the process of using a computer. To give but one example, ICL systems recognise clearly the concept of a 'user', who has proprietorial rights over certain resources that have been allocated to him, and can in consequence own files, initiate jobs etc. There is no analagous concept in IBM operating systems. In so far as the concept appears at all, the user is the person who pays the rental for the computer system—the idea of a community of users sharing the machine is not really there at all. It is therefore difficult to conceive of a nontrivial job-control language that could be common to ICL and IBM machines.

If this argument is taken to its logical conclusion it would appear that a universal job-control language implies a common operating system. Despite the confident prediction by Gene Amdahl that IBM's MVS will become the *de facto* standard operating system for the industry, we may reasonably doubt the emergence of a common operating system at the present time. The technology of operating systems is nowhere near mature enough to justify standardisation at present, and we may be sure that companies who have invested millions of pounds in the development of an operating system will not readily abandon that investment.

It thus appears that a common job-control language is not possible. Yet there is an evident need for it. Or is there? When one reaches a dead end in an argument it is always worthwhile reassessing the premises on which the argument is based. The need for a common JCL is by now a proposition in the 'motherhood and apple-pie' category, and, like many such propositions, it does not stand up to close scrutiny. Computer users, like the jobs they submit, come in all sorts of shapes and sizes, and it is no more reasonable to assume that a single JCL can be devised to suit them all than it is to assume that a single programming language is suitable for all classes of jobs from sorting to simulation. Even within a single machine range the range-standard JCL is found to be overcomplicated, and subsystems are developed for certain well defined and common tasks, e.g. running a simple Fortran program. It is reasonable to argue that the user should not be concerned with any aspects of machine organisation that are not self-evidently part of his job. Such relegation of irrelevant detail is very difficult in a general-purpose language, by virtue of its very generality. However, once we stop trying to be all things to all men, and tailor a subsystem to a particular class of users and/or jobs, the situation is very different.

It is at this level of subsystems that we require commonality. Given that Fortran is (or should be) the same on all machines, the user could reasonably expect that the protocol for using Fortran, including such things as linking external files to input/output channels, specifying libraries, choosing peripheral devices, should be similarly standardised. Indeed, the specification of a Fortran subsystem could reasonably form part of the international Fortran standard, and other languages should be treated similarly.

If such standard subsystems were available, much of the demand for a 'universal JCL' would vanish. Each machine range would still have its job-control language *(system-control language) for the benefit of the specialised systems programmer: it is reasonable that these should differ between machines, since they reflect the underlying operating-system structure in the same way that assembly languages reflect the machine architecture. These subsystems can, with some grief and pain, be grafted on to existing operating systems. A prime requirement in the design of future operating systems must be to facilitate the construction of subsystems. The third essay in this group explores some of the implications for operating system design, but first we look at a typical subsystem from the user's point of view.*

2 Simple interfaces for simple users

In the preceding essay we have argued the case for subsystems designed for simple, well defined classes of job. This essay explores the design of such a subsystem. I have chosen to describe a system for running Pascal programs, because that is the aspect of current operating systems with which I most frequently come into contact. However, the subsystem here described would apply with only trivial modifications to Fortran, Algol or any other language.

The fundamental principle underlying the design of the interface is that the user should never be required to specify things that are not directly relevant to his job as he sees it. Thus, while it is reasonable for the user to say where his source program is, and what is to be done with the results, it is not reasonable for him to be expected to specify the work files to be used by the compiler.

For a simple job, this can be achieved if the job-control system allows macros (procedures). Thus, with a suitably designed macro a simple Pascal job using default input-output is very easy to run on VME/B:

```
JOB(username, etc.)
PASCAL
- - - -
<program>
+ + + +
- - - -
<data>
+ + + +
ENDJOB
* * * *
```

However, as soon as we introduce any variation from this very simple basic job, complication sets in. For example, suppose I want to edit the contents of a file and then compile the edited program, also saving the source; I need something like the following

```
JOB (username, etc.)
NEWFILE (NAME = NEWPROG)
EDIT (OLDFILE = PROG, NEWFILE = NEWPROG)
- - -
<editing commands>
+ + + +
    PASCAL (NEWPROG)
    - - - -
    <data>
    + + + +
    SAVEFILE (NEWPROG)
    ENDJOB
    * * * *
```

If I want to pass the edited file through a preprocessor (written in Pascal) I am likely to finish up with something like the following sequence.

```
    JOB (username, etc.)
    NEWFILE (NAME = NEWPROG)
    EDIT (OLDFILE = PROG, NEWFILE = NEWPROG)
    - - -
    <editing commands>
    + + + +
    ASSIGNFILE (NAME = NEWPROG, LNAME = ICL9CE3)
    NEWFILE (NAME = MODPROG)
    ASSIGNFILE (NAME = MODPROG, LNAME = ICL9CE4)
    PREPROCESS
    - - - -
    <commands to preprocessor>
    + + + +
    PASCAL (MODPROG)
    - - - -
    <data>
    + + + +
    SAVE (NEWPROG)
    ENDJOB
    * * * *
```

This cannot be regarded as a simple or 'friendly' interface. In part this is due to the jargon, but even more so it is due to the stilted form of communication: each line consists of an imperative verb followed by some parameters. It is surprising how even a flavour of natural language improves the situation.

The sequence given above resembles a sequence of commands to a robot (as indeed it is). If we compare it with a piece of English Prose we notice one particular difference: it contains none of the common prepositions, demonstratives and connectives like 'it', 'this', 'with' etc. The clue to a friendly interface is the use of these neglected parts of speech. Compare the following version of our job:

```
    JOB (Username etc.)
    EDIT (PROG) WITH THIS
    :<editing commands>:
    CALL IT NEWPROG
    PREPROCESS IT WITH THIS
    :<preprocessing commands>:
```

```
COMPILE IT
RUN IT WITH THIS
:<data>:
ENDJOB
* * * *
```

We have immediately obtained a much more 'friendly' interface, by removing extraneous material such as invented names for temporary files.

We achieve this new interface in a very simple way. Each processor (editor, compiler, preprocessor) is regarded as a file-transforming device which takes a file and produces another file. It may also require steering information. The file created by the most recent such process is referred to as IT; steering information is introduced by WITH; and THIS introduces an immediate data stream ('alien data' in ICL terminology, a 'here' document in the sense of Barron and Jackson[1]).

The idea can be readily extended, e.g. concatenation is denoted by 'AND', thus:

```
COMPILE PROG 1 AND PROG 2
```

Input of a named file is, naturally, achieved by

```
READ THIS
:<data>:
CALL IT filename
```

A typical subsystem might be defined as follows.

(a) A command consists of an imperative verb followed by a principal file designator and optionally by a subsidiary file designator. Unless otherwise noted, a command produces as its result an anonymous file called the *current file*, and may also send output to a system-output device.

(b) A file designator may take the following forms

<filename>	meaning the named file
IT	meaning the current file
THIS	meaning the data that immediately follows in the command stream, suitably delimited
<file designator> AND <file designator>	indicating concatenation of the specified files

(c) The subsidiary file designator takes the form
WITH file designator

(d) Commands are
READ
PRINT (has no result)
EDIT
COMPILE
RUN

It is important that completely general composition of these simple concepts be allowed, so as to permit combinations like

```
RUN  IT  WITH  fileA  AND  file B
EDIT  fileA  AND  fileB  WITH  fileC  AND  file D
COMPILE  fileA  AND  THIS
:<immediate data>:
WITH THIS
:<immediate data>:
etc.
```

We thus see that a simple concept of commands as file transformers (which owes much to the UNIX concept of 'filters'[2]) coupled with some simple linguistic devices allows us to have subsystems that combine a friendly informality of interface with a rigorous definition.

Finally, we look at the operating system structure needed to support a sane approach to job control. It turns out to have a strong resemblance to MUSS, and to be not unlike what VME/B might have been (and might still yet become).

3 Sketches of an operating system

An operating system can usefully be regarded as divided into a kernal and a superstructure. The kernel is responsible for process management, providing an environment in which concurrent processes can exist; it will administer virtual memory, communication between processes, and the lowest level of peripheral control. The superstructure comprises data management and job-management functions, and it is this latter area that we concentrate on in this essay.

3.1 Job-management functions

It is a cardinal principle of software engineering design to keep the architecture and the implementation clearly separated. We therefore consider first the user interface of the operating system as far as job control is concerned. We consider a job that consists of an arbitrary number of job steps (including one step as a semitrivial case). Then we can distinguish two job-control functions immediately:

(a) sequencing of job steps, the actual sequencing depending on the success or otherwise of individual job steps
(b) creating an environment for a job step—providing work files, binding symbolic names to actual files, peripherals etc.

A further function, really an extension of sequencing is:
(c) dealing with error situations.

Finally, there is one more major activity:
(d) budgeting and accounting.

In conventional systems these activities are all contained within the operating system, and are therefore immutable so far as the user is concerned. Conceptually we can imagine that there resides within the operating system a job-control program which accepts the job, notes its requirements and initiates the job steps; whenever a job step terminates (in the normal way or prematurely) control goes back to the job-control program, which may initiate another job step or terminate the job as appropriate. This structure is shown in Fig. 1. Of course, most systems do not exhibit this clear division in actual fact: job-control functions are accomplished by code scattered throughout the system. However, if we define the actions of a (hypothetical) job-control program as equivalent to the actions specified by the JCL we have a useful model. (We note in passing that a scientific model does not have to be a simplified version of the real thing: it is a mechanism that would produce the same observed results as the real thing. When he developed his electromagnetic theory, Maxwell used a model that envisaged space as being filled with intermeshing rollers: the unreality of his model did not prevent him making a major advance in physics.)

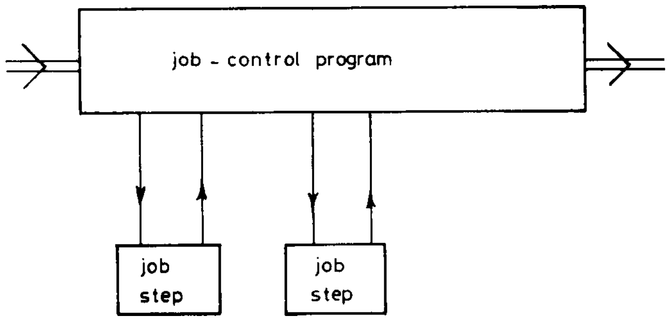


Fig. 1 Model of job control function

A scientific model is useful if the results obtained by studying the model lead to a better understanding of phenomena in the real world. (Maxwell's electromagnetic equations were one of the major factors in the discovery of the possibilities of radio transmission.) On this basis, our model of the job-control function is a useful one. We first observe that it clearly separates application programs and utilities (compilers, editors etc.) from that which is conventionally regarded as firmly rooted in the operating system (Fig. 2). We next remark that the sequencing of job steps requires communication between the job steps and the sequencing mechanism (e.g. to report success or failure). In actual systems this may be done by placing values in registers according to some convention, or by providing a shared communication area. In terms of our model we can abstract the communication between job steps and describe it in terms of a set of variables belonging to the job-control program, and hence global to the job steps.

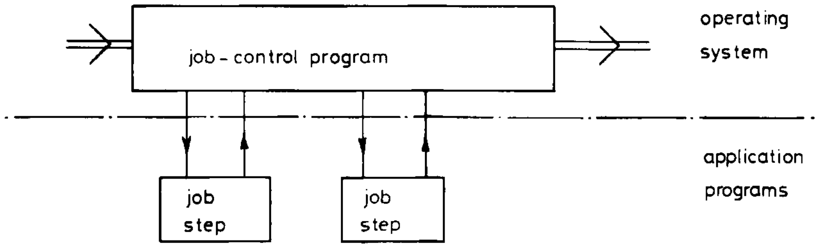


Fig. 2 Conventional division of responsibility in job control

Using this model we can distinguish various categories of systems. We have already observed that two important functions of the job-control program are setting up an environment for a job step sequencing between job steps. If the setup for each job step immediately precedes initiation of that job step, we have an OS/360-like system (Fig. 3). At the other extreme, if all the initialisation is done at the start we have an Atlas-type job description, i.e. a static description of the environment (Fig. 4).

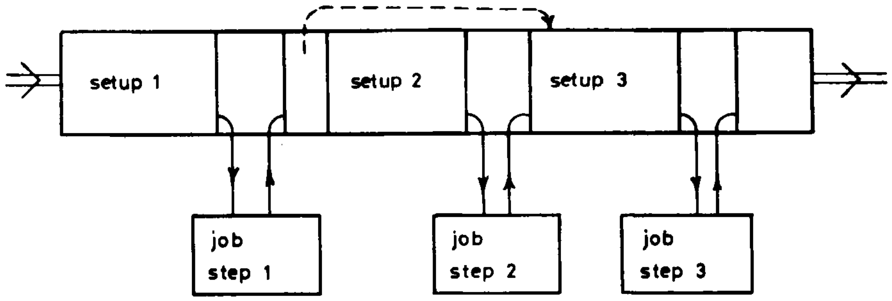


Fig. 3 Model of OS/360 job control
Note the possibility of optional omission of a job step, shown by the broken line

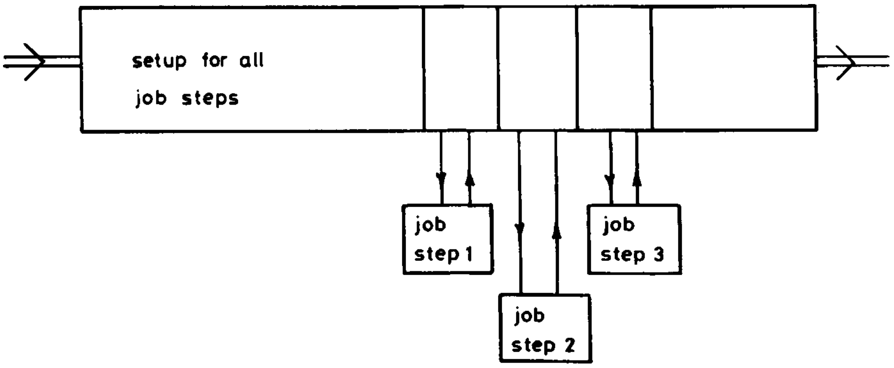


Fig. 4 Model of Atlas job control
Note that this does not permit optional execution of job steps

We have so far omitted the budgeting and accounting role of the job-control system, so to complete our model we must prefix an authorisation check to the job-control program, and add at the end an accounts updating procedure as in Fig. 5.

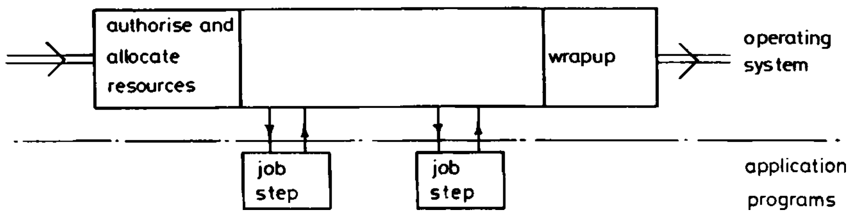


Fig. 5 Elaboration of job-control program

3.2 Extension of the model

We now postulate a specific operating-system structure. We suppose that the primitive-level job-control interface consists of a large body of procedures each of which, when called, performs an elementary job-control function. The job-control program will now reduce to a (possibly large) number of calls on these primitive procedures. If this is the case we can make life imitate art by making the actual system more like our model, by having an operating-system architecture that isolates the job-control program as a clearly defined unit. Now we have in principle a very flexible system. There exists in programming a body of knowledge and technique whereby sequences of procedure calls can be composed into calls of higher level procedures, and we can therefore provide job-control interfaces at various levels of abstraction from the basic interface.

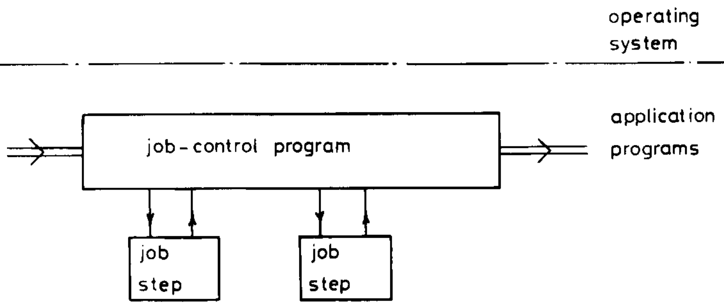


Fig. 6 Hypothetical job-control system outside the operating system

Since the job-control program now consists of a sequence of calls of primitive procedures, it is tempting to argue that it can be expressed in any language that can be compiled into, or interpreted in terms of, the basic primitives. In other words, the job-control program comes below the operating-system line (Fig. 6). A moment's thought shows that this cannot be entirely so. Validation of a user, checking budgets and setting limits etc. must be done within the operating system in a protected manner, and we are therefore led to a '3-tier' model (Fig. 7). Now the top layer is clearly operating system, the bottom layer is 'applications', and the question remaining is how do we program the middle layer.

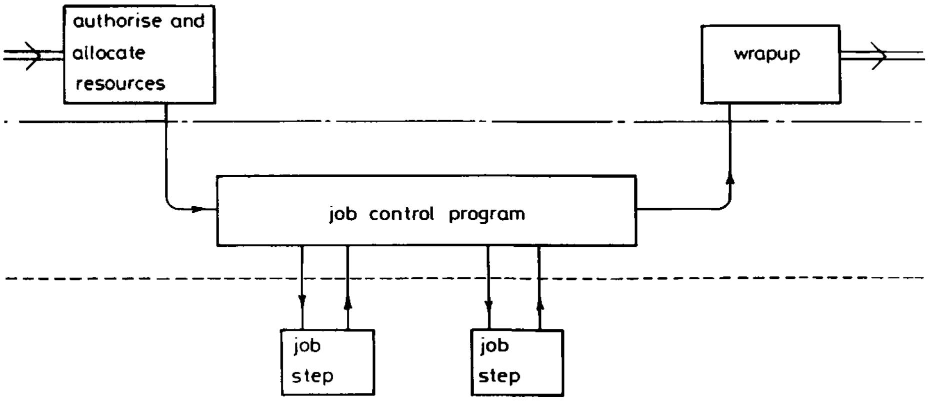


Fig. 7 Three-layer model of job control

3.3 Job-control programs

The first important point to note is that once the job-control program has come 'below the line', it is no longer a sacred cow, and does not have to be the same for all users. There are three broad possibilities for the job-control program in such a context:

(a) use a specialised language–a job-control language
(b) provide 'end-user' subsystems for well defined classes of job
(c) use a 'conventional' high-level language.
(Note that this may or may not be the same high-level language that is used for any or all of the job steps.)

Possibility *(a)* is appropriate to professional programmers who wish to exploit the more esoteric facilities of the system; we do not consider it further in this essay. Possibility *(b)* is appropriate when there is a community of users whose jobs conform broadly to a standard (and simple) pattern: students in an educational institution are an obvious example. We have outlined a typical subsystem in the preceding essay. Possibility *(c)* is likely to commend itself to writers of packages, since it enables them to conceal much of the job control from the user. It is important that the system design should not make these options mutually exclusive. A combination of *(b)* and *(c)* in which much of the job control is absorbed into the program, while what remains is dealt with in a simple user-oriented language, has great promise as an attractive and 'friendly' interface. Thus, for example, the transition from compiler to link editor (collector) to running the object module can conveniently be subsumed into the compiling system.

The MUSS operating system[3] employs the approach of making all job-control functions available as procedures. The designers of VME/B started well by providing a procedural interface, but took a reactionary step by concealing this procedural interface from the ordinary user (and, indeed from the compiler writer.) It would not be too late to change, but one cannot be sanguine about the chances. Perhaps the real moral to be drawn from the sorry history of job control is that operating-system designers should limit their efforts to providing the tools for job management. They should not build into the operating system a preconceived idea of how it is going to be used, least of all if (as is most common) they have never been users themselves.

References

1 BARRON, D.W., and JACKSON, I.R.: 'The Evolution of Job Control Languages', *Software Pract. & Exper.*, 1972, 2, pp. 143-164.
2 KERNIGHAN, B.W., and PLAUGER, P.J., *Software tools* (Addison Wesley, 1976)
3 FRANK, G.R., and THEAKER, C.J. 'MUSS - the user interface' *Software Pract. & Exper.*, 1979 (in press)

ICL Worldwide

The head office of International Computers Limited is at
ICL House, Putney, London SW15 1SW, England

Issue 1 of the *ICL Technical Journal* gave a list of the principal address of ICL subsidiary companies, branches, dealers and agents. Since that list was published a new subsidiary has been established:

MALAWI	**ICL Malawi Ltd** Henderson Street, Blantyre, PO Box 5070, Limbe

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.