SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 16(6), 531–539 (JUNE 1986)

Providing Multi-User Access to Distributed Array Processors

P. D. STEPHENS AND J. K. YARWOOD
Edinburgh Regional Computing Centre, University of Edinburgh, Edinburgh EH9 3JZ, U.K.

SUMMARY

This paper describes the integration of two ICL Distributed Array Processors (DAPs) into a dual ICL 2976 configuration running the Edinburgh Multi-Access System (EMAS). The principles underlying the general multi-access service are not compromised; special scheduling arrangements provide effective control and use of the DAP resource while allowing multi-user access to the DAPs for DAP program development.

KEY WORDS EMAS Multi-access 2900 Parallel processing Array processor DAP

HARDWARE OVERVIEW

The basic dual ICL 2976 configuration is shown in Figure 1. Accesses to each unit of main store are resolved by the store multiple access controllers (SMACs). Data paths from peripheral controllers to SMACs are via store access controllers (SACs). The SMACs offer four ports to the SACs and central processor units (known by ICL as 'order code processors' or OCPs. The more usual term CPU will be used hereafter).

The hardware supports 32-bit virtual addressing and 28-bit real addressing. The SMAC number occupies four bit positions at the high-order end of a real address.

Segment tables used for virtual address translation can specify that a given segment be paged or unpaged. In the former case, the real address of a page frame is obtained from a page table pointed to by a segment table entry. In the unpaged case, the segment table entry contains the real address of the base of a segment of contiguous real storage.

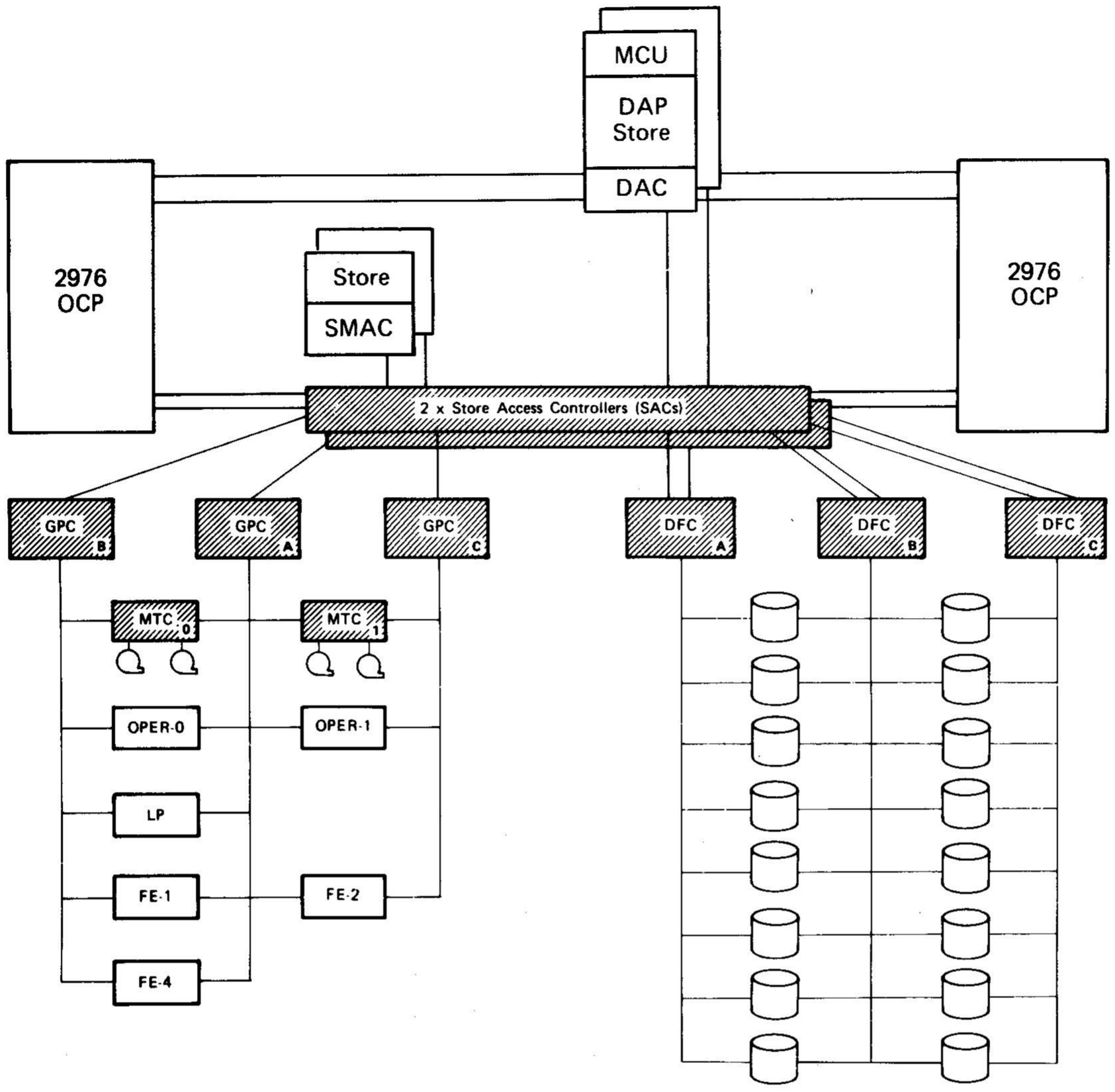
The ICL Distributed Array Processor (DAP) comprises

(i) a number of simple processors, called processing elements (PEs), each with a local store
(ii) a master control unit (MCU).

There are two models, engineered respectively with 4K and 16K chips; in each case the PEs are presented in 64 by 64 two-dimensional arrays. In the 4K-chip model each PE local store is 4096 bits and in the 16K-chip model each store is 16384 bits. Total storage (DAP storage) for the two models is therefore 2 Mbytes and 8 Mbytes, respectively.

The operations performed by the PEs are logical and arithmetical, with inputs from and outputs to the stores of adjoining elements.

0038–0644/86/060531–09$05.00 *Received 19 July 1984*



Key:

OCP	– Order Code Processor	GPC	– General Peripheral Controller
DAP	– Distributed Array Processor	MTC	– Magnetic Tape Controller
DAC	– DAP Access Controller	OPER	– Operator Console
MCU	– Master Control Unit	LP	– Line Printer
SAC	– Store Access Controller	FE	– Front End (Communications Processor)
SMAC	– Store Multiple Access Controller		
DFC	– Disc File Controller	Mbyte	– 1,048,576 bytes

(Shaded boxes represent control units)

Figure 1. ERCC dual 2976 configuration

The processing elements operate under the control of a program (the DAP program) which itself resides in DAP store. The PEs containing the program do not take part in the DAP processing, other than by supplying program instructions to the MCU, which controls all the PEs in the data area; all PEs can execute each MCU primitive command simultaneously.

The DAP has no I/O capacity; it is attached to the host 2900 system as main storage. That is, it connects to SACs and CPUs exactly as a unit of main storage. The DAP access controller (DAC) performs the same functions as a SMAC, for storage accesses initiated by CPUs or device controllers. In addition, control and interrupt paths exist between DAP and CPU, analogous to those between device controllers and CPUs.

When the MCU is not executing, the DAP behaves exactly as main store; when the MCU is executing the contents of the store change without any of the normal read and write cycles initiated by the CPU or SAC.

A *DAP program block* is a contiguous area of DAP storage described by a base and limit register in the DAP. A program block consists of a number of sub-areas:

(a) workspace (read/write), for use by software
(b) control (read only), for hardware/software communication
(c) program (read only) data
(d) code (execute only)
(e) program (read/write) data.

(The access shown relates to accesses by the DAP program. Certain alignment requirements must also be satisfied).

DAP (MCU) execution is initiated by control signal from the host Supervisor after the following DAP registers have been loaded:

(i) program block datum
(ii) program block limit
(iii) code base
(iv) code limit
(v) program counter (code entry point)
(vi) interval timer
(vii) instruction counter.

MCU execution continues until one of the following occurs:

(a) a DAP STOP instruction is executed
(b) the host forces a STOP (by control signal)
(c) DAP program error
(d) expiry of the interval timer or instruction counter
(e) hardware error.

The host is notified by interrupt when any of these occurs.

The installation of the DAP at Edinburgh Regional Computing Centre is described elsewhere.[1]

EMAS OVERVIEW

The Edinburgh Multi-Access System is described elsewhere.[2–4] Briefly, it is a general-purpose time-sharing system providing rapid response to the terminal user for all interactions having modest resource requirements (CPU-time, virtual storage). Processes are scheduled according to their current behaviour, independently of the global resource available, and the strategy is to attempt always to have sufficient CPU-bound

processes in main storage to keep CPU utilization high, while keeping sufficient spare page frames for rapid servicing of interactive processes with small *working sets*.[5,6] A comprehensive spooling and background job scheduling system complements the interactive service, and full NIFTP-B(80) file transfer and JNT MAIL facilities are provided.

A hundred and twenty simultaneous user processes are supported on the configuration shown in the diagram. (Each 2976 processor executes approximately 1·3 Mips.) Supervisor overhead under the worst conditions does not exceed 40 per cent of total CPU time, and 100 per cent CPU utilization is normal for periods of many hours.

Each user process is provided with a large virtual memory (currently 40 Mbytes), a virtual processor capable of executing the non-privileged instructions of the real processor, and a set of services in the form of procedure calls to more privileged system software for file and terminal access.

No I/O devices are directly available to an EMAS user process; listings and file transfer are performed by communication with a Spooler process. EMAS provides a virtual filestore; files are accessed by being *connected* (not copied) into virtual memory; virtual addresses are *referenced* by machine instructions within the process; the Supervisor satisfies resulting *page-faults* by moving the necessary *pages* to main store from the disc storage which is the longer term site for the user's file data. Programs, data, system service code and file index data all use the same mechanism, in order to arrive in real main store for referencing or execution.

All EMAS programs are re-entrant, and the access permission mechanisms result in a high level of sharing (typically up to 50 per cent) of real store pages, both for system and user virtual memory pages.

The privileged procedures providing file system, timing, contingency-handling and terminal services are collectively known as the EMAS Director.[7] Director is implemented as paged, shared, in-process code.

EMAS was originally written for an ICL (formerly English Electric) System 4–75 machine. After 1976 it was re-implemented for ICL 2900 Series systems and is currently known, therefore, as EMAS 2900.

DAP PROGRAMMING OVERVIEW

The high-level aspects of DAP programming are described in a further paper.[8] In summary, a DAP program comprises a host part, programmed in a standard dialect of FORTRAN, and a DAP part, programmed in DAP FORTRAN (which embodies extensions to FORTRAN reflecting the three-dimensional nature of the storage model). Communication between the host and DAP parts is by means of COMMON areas which physically reside in the DAP during DAP execution, and by subroutine call of the DAP part from the host part.

Execution commences in the host part, which is responsible for the presentation of data, in a suitable form, to the DAP COMMON area. When a DAP-part subroutine is called, supervisory software intervenes to load the DAP registers and start the DAP. Execution normally continues until a STOP instruction is executed in the DAP part, when the DAP interrupts the host system to resume the host-part computation for unloading or perhaps re-loading the DAP data areas.

INCORPORATING THE DAPS INTO EMAS

Configuration control and provisions in Supervisor

As described in 'Hardware overview', the DAP is capable of acting simply as a main storage unit. At the busiest times of day one of the DAPs is made available to the general interactive service as store; during off-peak times the normal service operates using 8 Mbytes of main store, while two DAP job streams execute in the DAPs.

This capability to alternate between using the DAP as store and as 'DAP proper' was a requirement from the outset. It is a generally accepted judgement that the DAP provides maximum communal benefit by being used as main store at peak periods. This was in any case a basis of the original funding arrangement.

EMAS was designed to operate a continuous service, and is equipped to deconfigure CPUs, store modules and filestores and to re-incorporate them into the service configuration. These changes may be made automatically after unit failure, or by operator command, to produce a parallel test system while maintaining a reduced level of service.

When EMAS is loaded, the DAPs are identified, tested, and incorporated as store. The operator can later deconfigure them as store units and re-incorporate them as DAPs. When they are available as DAPs, Supervisor will allocate them to Director on a block basis (a block is 128 Kbytes or 256 PEs). Supervisor also provides facilities to start a DAP and to multi-program between blocks allocated to different programs (but see the next subsection).

If a DAP is unused and the system is 'busy' (more than ten users per megabyte of available main store) the DAP is used as main storage. In this state it is still available as a DAP but a Director request for blocks will be subject to a delay (five to ten seconds) as the pages in the DAP are relocated.

The current status of the DAPs is visible to all Directors.

Interface for application-support software: provisions in Director

At the higher levels, the application-support software is able to request the following services by procedure call to Director (see 'EMAS overview' and elsewhere[7])

(i) allocate *n* Kbytes of DAP storage for the DAP program block
(ii) initiate DAP (MCU) execution at a given offset into the code area, with a nominated limit on total MCU instructions to be executed
(iii) de-allocate the storage for the DAP program block.

In practice the DAP program block for any significant problem is found to require at least half of the available real DAP storage; indeed the 2 Mbyte DAP is in reality much too small for many of the problems for which it is otherwise ideally suited. Consequently the decision was taken always to allocate the whole DAP.

The Director procedures provide mechanisms

(a) to validate the requests from the higher-level software
(b) to translate the requests into corresponding real DAP program block allocation/execution requests on the resident Supervisor
(c) to locate the DAP program block (simply a contiguous block of *virtual* storage from the point of view of the application-support software) in real DAP storage before execution commences.

The Director procedures are for the most part unconcerned with real main store addresses, since in-process references are normally exclusively virtual. One of Director's primary functions is to manage a process's virtual storage and segment tables. A fundamental service which it provides is the *connection* of files into the virtual memory and the converse operation of disconnection to ensure that updated files are safely on the disc (see 'EMAS overview' and References 2, 4 and 7). Director is thus well placed to manage DAP storage on behalf of the user process, and two schemes have been devised.

In the first, and simpler, scheme Director responds to the user's request for DAP allocation by claiming a real DAP allocation from Supervisor and filling in the user's segment table so that a set of virtual addresses corresponds to the real DAP. Thus while the program executes in host mode, setting up its data areas, the data move directly into the DAP. When the program switches into DAP mode, the DAP start request is forwarded directly to Supervisor. When the DAP part of the program stops, control is passed directly back to the host part of the program. The DAP cannot be freed until the host part has extracted any useful results and concluded.

The following considerations led to the adoption of a more complicated scheme as an alternative to the above. Typical runs of some production DAP programs take several hours of DAP time. For the above straightforward loading approach DAP MCU-time and job elapsed time are closely comparable, since no multi-programming is done in the DAP. (In many DAP programs the DAP is called once only from the host part, and DAP loading/unloading times form a small part of the total time). One long run thus occupies the DAP to the exclusion of any short development runs for long periods; nor is it possible to reserve one of the DAPs for short or development runs, because it would lead to poor use of that valuable resource and also because the second DAP is in any case normally used as main store during the prime shift, when a facility for interactive DAP program development is most needed.

Further, the DAP is up to 100 times faster than a 2976 CPU. The host portion of the program executes in a multi-access environment with around 100 other users. Thus programs under development often claimed the DAP for a much longer period than its DAP MCU time.

In the second scheme the Director responds to the user's request for DAP allocation by providing a virtual DAP. Having checked that the current limit on the number of virtual DAPs has not been exceeded, Director creates a temporary file exactly the same size as the DAP and *connects* it into the virtual memory as in the first scheme. The host part of the program executes as before; indeed it perceives no difference as it sees only virtual addresses. When the program switches to/from host DAP mode, Director has to

(a) disconnect the virtual DAP, forcing any updated pages to backing store
(b) claim a real DAP
(c) roll in the virtual DAP, now on backing store, into the real DAP
(d) forward a DAP start to Supervisor.

When the program completes its DAP mode, Director again intervenes to

(i) roll out the real DAP to the disc site of the virtual DAP (read-only areas are not rolled out)
(ii) re-connect the virtual DAP into the same position in the virtual memory
(iii) pass back control to the program.

The virtual DAP is moved into real store by page-fault interrupt as the host program extracts the results from the COMMON areas. Thus the DAP is occupied for the minimum time.

Each time the DAP is started, Supervisor loads the DAP's interval timer (IT) register. The value always expires some eight seconds after DAP execution commences, and the host Supervisor is thus alerted by interrupt. If no allocation request is queued, the DAP is immediately restarted with negligible elapsed time overhead and with a reloaded IT register. However, if another process is awaiting the DAP, a special 'DAP stopped' reply may be given to the waiting Director which then executes, rolls out the DAP into the file created when the original allocation request was received, alters the DAP program block segment table entries to reflect the move and calls Supervisor to de-allocate the real DAP storage. Supervisor now responds to the queued DAP allocate request, the Director of the waiting process loads the allocated DAP area and DAP execution continues.

Jobs are normally allowed a minimum of two minutes MCU time before being forced out. The mechanism provides a round-robin servicing of waiting DAP jobs on a roughly two-minute interval for modest degradation of DAP utilization, and allows DAP program development concurrently with long production runs.

It should be restated[2] that in EMAS there is no distinction, from the point of view of in-process code, between a *file* which is *connected* and the area of *virtual store* which it *occupies*. It is clear, however, that Director needs to create a distinction in order to achieve the move of data from physical DAP to virtual store. This is not difficult, using the EMAS 2900 feature of having all *real* main store mapped permanently into every *virtual memory* (as well as having virtual memory selectively mapped into real storage as necessary).

It will be noted that the scheme adopted pre-empts the use of the DAP interval timer register by higher-level software. All high-level scheduling and accounting is done on the basis of DAP instructions executed, using the instruction-counter register and a factor chosen empirically to relate instructions executed to DAP processing time.

VIRTUAL DAPS LARGER THAN THE REAL DAPS

A modest extension of the notion of a virtual DAP enables DAP programs having COMMON areas much larger than those in the real DAP to be handled. With the co-operation of the DAP loader[8] and other application-support software, the DAP programmer is able to label one or more COMMON areas as being eligible for emptying and refilling with the contents of other nominated COMMON areas during DAP program execution. This technique is called DAP data expansion, described more fully in the paper by Brown.[8] For a class of problem in which computations are performed progressively through large quantities of data, a special subroutine call in the DAP part of the program is provided to cause entry to supervisory software specifying a swap of given pairs of COMMON areas. This gives the programmer the ability to conflate a problem which would involve many DAP runs, because of the small size of the DAP, into a single run with internal data transfer. The gains, both in avoiding complete DAP loads (and indeed in scheduling a single longer job rather than many shorter ones) and in automating the passing of data between successive separate DAP jobs, are considerable.

The scheme is implemented by providing

(i) special DAP stop codes for use by the DAP part of the DAP program. These

stop codes are generated as a result of 'swap COMMON area' subroutine calls in the DAP part, and are recognized by Director when the 'DAP stops' interrupt has been received. The stop code, planted by the higher-level software, identifies the source and destination areas for the transfers.

(ii) a further procedure call in Director allowing the application-support software to nominate offsets and lengths of COMMON areas within the DAP program block which are to form source and destination areas for movements of data.

The communication between Director and support-software is effected by requiring a procedure parameter to be supplied when the start-DAP request is made on Director. When a 'swap-data type' STOP is noted by Director, it calls the supplied procedure, which recursively calls Director to initiate one or more transfers between DAP-resident COMMON areas and COMMON areas in the larger virtual DAP program block. The primitives at the Director interface can request

(a) a uni-directional transfer (rather than a swap). An identifier is returned to the caller

(b) a WAIT, specifying one or more transfer identifiers for which completion is to be awaited.

The low level of these primitive Director calls favours two important classes of problem:

1. If the computation progresses serially through large data areas, it is possible to request transfers in advance of the requirement of new data by the program, and by double or multiple-buffering use of the resident DAP COMMON areas it is possible to overlap DAP processing with data transfer, resulting in significant elapsed-time savings.
2. If the computation uses the contents of COMMON areas as read-only data, outward transfers from the DAP can be avoided altogether.

Some of the most important problems[9–12] being analysed in Edinburgh do indeed fall into these categories.

CONCLUSIONS

Although the scheduling of such an unusual resource (for a multi-access environment) presented a variety of apparently unwieldy problems, suitable facilities and system services have been designed and implemented to the satisfaction of an important group of users. Apart from an unfortunate spate of DAP-related hardware problems, the effect on the general user population is limited to the fluctuation of the amount of real main storage available on the system; at peak service times the additional storage has been of very real direct benefit.

REFERENCES

1. J. G. Burns and A. McKendrick, 'The acquisition and installation of a DAP (the ICL Distributed Array Processor)', *University Computing,* **6**, (1984).
2. H. Whitfield and A. S. Wight, 'The Edinburgh Multi-Access System', *The Computer Journal,* **16**, (4), 331–346 (1974).
3. P. D. Stephens, J. K. Yarwood, D. J. Rees and N. H. Shelness, 'The evolution of the operating system EMAS 2900', *Software—Practice and Experience,* **10**, 993–1008 (1980).
4. D. J. Rees and P. D. Stephens, 'The kernel of the EMAS 2900 operating system', *Software—Practice and Experience,* **12**, 655–667 (1982).

stop codes are generated as a result of 'swap COMMON area' subroutine calls in the DAP part, and are recognized by Director when the 'DAP stops' interrupt has been received. The stop code, planted by the higher-level software, identifies the source and destination areas for the transfers.

(ii) a further procedure call in Director allowing the application-support software to nominate offsets and lengths of COMMON areas within the DAP program block which are to form source and destination areas for movements of data.

The communication between Director and support-software is effected by requiring a procedure parameter to be supplied when the start-DAP request is made on Director. When a 'swap-data type' STOP is noted by Director, it calls the supplied procedure, which recursively calls Director to initiate one or more transfers between DAP-resident COMMON areas and COMMON areas in the larger virtual DAP program block. The primitives at the Director interface can request

(a) a uni-directional transfer (rather than a swap). An identifier is returned to the caller

(b) a WAIT, specifying one or more transfer identifiers for which completion is to be awaited.

The low level of these primitive Director calls favours two important classes of problem:

1. If the computation progresses serially through large data areas, it is possible to request transfers in advance of the requirement of new data by the program, and by double or multiple-buffering use of the resident DAP COMMON areas it is possible to overlap DAP processing with data transfer, resulting in significant elapsed-time savings.
2. If the computation uses the contents of COMMON areas as read-only data, outward transfers from the DAP can be avoided altogether.

Some of the most important problems[9–12] being analysed in Edinburgh do indeed fall into these categories.

CONCLUSIONS

Although the scheduling of such an unusual resource (for a multi-access environment) presented a variety of apparently unwieldy problems, suitable facilities and system services have been designed and implemented to the satisfaction of an important group of users. Apart from an unfortunate spate of DAP-related hardware problems, the effect on the general user population is limited to the fluctuation of the amount of real main storage available on the system; at peak service times the additional storage has been of very real direct benefit.

REFERENCES

1. J. G. Burns and A. McKendrick, 'The acquisition and installation of a DAP (the ICL Distributed Array Processor)', *University Computing,* **6**, (1984).
2. H. Whitfield and A. S. Wight, 'The Edinburgh Multi-Access System', *The Computer Journal,* **16**, (4), 331–346 (1974).
3. P. D. Stephens, J. K. Yarwood, D. J. Rees and N. H. Shelness, 'The evolution of the operating system EMAS 2900', *Software—Practice and Experience,* **10**, 993–1008 (1980).
4. D. J. Rees and P. D. Stephens, 'The kernel of the EMAS 2900 operating system', *Software—Practice and Experience,* **12**, 655–667 (1982).