Planned ESA/370™ Exploitation by the SAS® System under MVS

Keefe Hayes, SAS Institute Inc., Cary, NC

ABSTRACT

IBM's® powerful Enterprise Systems Architecture/370™ provides many new capabilities for system and application software. The exploitation of these capabilities in future releases of Version 6 of the SAS® System under MVS will offer enhanced function and performance.

INTRODUCTION

It is through MVS/ESA™ that the new capabilities of the ESA/370™ architecture are available to applications. Therefore, this paper focuses on the use of these new capabilities by the SAS System within the MVS/ESA environment. The paper presents discussions of

- MVS/ESA facilities that use the architectural enhancements of ESA/370
- areas in which these facilities can enhance the SAS System under MVS
- issues that may affect the implementation strategy.

Version 3 Release 1.3 of MVS/SP™ (MVS/ESA) provides the foundation for current research into the utilization of ESA/370 by the SAS System. As research progresses, plans for implementing enhanced function within the SAS System are unfolding. However, the plans for implementation are somewhat volatile, since the many capabilities of MVS/ESA may suggest alternative approaches and various factors may influence priorities. Plans may also evolve as new releases of MVS/ESA with additional capabilities become available.

MVS/ESA FACILITIES

The MVS/ESA operating system offers a number of new facilities. Three principal facilities that the SAS System plans to utilize for improved function and performance are

- very large in-memory storage addressability
- the use of Program Call (PC) linkage for system services
- enhancements to the PC linkage mechanism.

In-Memory Storage Addressability

Very large in-memory storage addressability offers the greatest performance benefit of the new ESA/370 architecture. It enables applications to move more data from external storage devices into processor memory, eliminating subsequent I/O to manage the data. MVS/ESA supports very large in-memory storage through four new facilities:

- data spaces
- hiperspaces (High PERformance SPACES - HIPERSPACE™)
- data windowing services
- the Virtual Lookaside Facility (VLF).

Data spaces Data spaces are similar to address spaces. The data within them are directly addressable using standard System/370™ instructions. To provide access to data spaces, the ESA/370 architecture defines a set of 16 access registers for use in implementing a new address translation process. Each access register contains a token that identifies an address space or data space to which a program has addressability. The architecture pairs each access register with a general register. A general register contains a data address within a space in the range from 0 to 2 gigabytes. Together, an access register/general register pair positions a data reference along two dimensions. The access register positions data reference along the horizontal dimension, and the general register locates the data along the vertical dimension (see **Figure 1**).

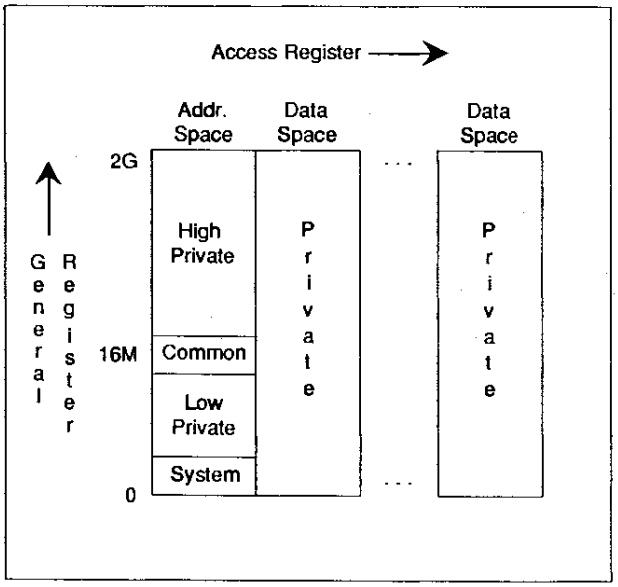

Figure 1 *Two-Dimensional Addressing Structure in ESA/370.*

By default, an MVS/ESA program executes in primary address space control mode (primary mode). In this mode, address translation ignores the access registers and all data references resolve to the primary address space. To reference data in other address/data spaces, ESA/370 defines the new access register address space control mode (AR mode) that an MVS/ESA program can establish to get addressability to that data.

Data spaces differ from address spaces in two important ways. First, data spaces do not share common storage system control blocks and code with other address spaces (see **Figure 1**). All the storage in a data space is private, so the entire addressing range of a data space is available for storing data. Second, a data space cannot be the target of an instruction fetch operation, so programs can reside in a data space, but they cannot execute there. Data spaces are especially appropriate for holding data for which the pattern of access is random, such as for the elements of a large table.

To create a data space, a program issues a DSPSERV macro that returns a unique system-wide space identifier known as an STOKEN. To gain access to the data space, a program passes the STOKEN to the ALESERV macro, which returns an access token (ALET). This ALET serves as a local data space identifier for a program to use in an access register. This two step process allows the creator of a data space to share access with other programs.

Hiperspaces Hiperspaces are similar to data spaces in that to create one, a program issues a DSPSERV macro. However, hiperspaces differ in that the system only uses expanded storage for their data. Data in expanded storage are not directly addressable from programs. Therefore, to access the data in a hiperspace, a program issues the macro HSPSERV. This macro moves the data in 4-kilobyte blocks between the hiperspace and buffer areas that the program allocates in its address space. Accessing data in a hiperspace is conceptually like I/O, so for existing applications that process blocked records, converting their I/O routines to take advantage of data-in-memory may be relatively easy.

An advantage of hiperspaces over data spaces is that expanded storage is a more economical place to put data than the central storage that data spaces use. The system only moves hiperspace data into central storage in response to an HSPSERV request. Therefore, a program can better manage its use of the central storage resource. Hiperspace processing does introduce some operating system overhead associated with moving the data. However, the new move-page facility available on 3090J processors provides hardware support for HSPSERV requests, alleviating this overhead.

Data windowing services Data windowing services are a set of callable routines that couple the new hiperspace facility with data-in-virtual. MVS provides data-in-virtual for mapping large data objects into memory, and with hiperspaces the size of an object that can reside in memory is quite large. Data windowing services enable a high-level language application to map a large data object directly into a hiperspace. The application can then manipulate the object without doing any I/O to a work data set. By using data-in-virtual, data windowing services also can automatically copy hiperspace data to and from permanent storage by using a special type of data set known as a VSAM linear data set.

A VSAM linear data set is a new form of data set. Unlike other data set types, it provides no access method. Instead, to process a linear data set, a program creates a window area and uses a data-in-virtual function to map the window to a location in the data set. Data-in-virtual positions the data set to correspond to the desired window view. A linear data set can range in size from 0 to 4 gigabytes. It can have any organization, so applications can format data in them to best meet their needs.

The Virtual Lookaside Facility (VLF) The Virtual Lookaside Facility (VLF) is an MVS/ESA subsystem that manages data objects in memory and provides access to them by name. To use VLF, a program reads in a group of objects, such as the members of a load module library, and gives them to VLF to manage. VLF defines a group of objects as a class. Other programs can then request the object without incurring I/O overhead. For VLF to manage a class of objects, the installation must describe the class to VLF through parameters that the operating system reads at start-up. VLF services are only available when its subsystem is up. VLF is well suited to applications that access data by named object.

Program Call Linkage for System Services

The use of Program Call (PC) linkage for system services provides both functionality and performance benefits. By using the PC instruction, MVS/ESA system services can take advantage of hardware support to pass control to routines that execute at a higher authority and in a different address space than a calling program. This support offers two benefits. First, it allows programs to execute authorized system services without causing a supervisor call (SVC) interrupt. Therefore, system services that use PC linkage incur no SVC interrupt or dispatcher overhead. A positive side effect is that system throughput increases since fewer interrupts occur. Second, PC linkage supports inter-address space communication without scheduling service requests across address spaces. For services that communicate with system address spaces, use of PC linkage can eliminate the complex and expensive protocol of scheduling service requests to transfer data.

The MVS/ESA operating system now supports PC linkage for several key system services:

STORAGE	allocates and frees memory
WAIT POST	synchronize the operation of interrelated tasks
ESTAEX	establishes routines for handling unexpected errors.

Since applications frequently request these services, their PC implementation will improve overall system performance.

Enhancements to PC linkage

Enhancements to the PC linkage mechanism greatly expand the potential for using PC routines in application software. The useful new feature is that PC routines can now save the attributes of the caller's environment on a linkage stack that ESA/370 defines. This linkage stack is a secure place on which to save these attributes. Therefore, PC routines no longer must save the caller's environment by executing the PCLINK service that requires a supervisor state caller. Also, because unauthorized programs cannot modify the linkage stack, the new PC linkage supports calls to programs that execute in a state and key of lesser authority, as well as greater authority, than that of their caller.

This new PC linkage, when coupled with the enhanced system services described above that use it, can greatly reduce the amount of authorized code that runs on the system. For example, cross memory routines cannot issue SVC instructions. By using the system services that implement the new PC linkage, more cross memory routines can execute in problem state and task key. Formerly, these routines had to use the branch entry forms of the GETMAIN and FREEMAIN macros to allocate and free memory and the branch entry forms of the WAIT and POST macros to synchronize tasks. Branch entry requires a supervisor state, key zero caller.

Therefore, authorized programs can localize code that executes in supervisor state or in a system key to a handful of small routines. Virtually all code that executes on behalf of a user request to such programs can run in the user's state and key. Potentially, a user can even provide an exit routine that an authorized program can invoke through program call without an integrity exposure. Such a linkage avoids the overhead of the prevailing mechanism of issuing a SYNCH SVC.

The enhanced PC linkage also simplifies the process of establishing authority for communication between a user and a cross memory application. First, the enhanced linkage uses the new program return (PR) instruction to return control to the user by means of the linkage stack. Since unauthorized code cannot alter the contents of the stack, issuing PR requires no authority. The old PC linkage uses the program transfer (PT) instruction, which returns control by means of register parameters. Since unauthorized code can modify the registers, issuing PT requires authority. Second, the user's home address space is now accessable to a cross memory PC routine through the MVS/ESA defined ALET value of two. Therefore, the cross memory application no longer must access it as a secondary address space, and authority to issue the set secondary address space (SSAR) instruction to reference data in the home address space is no longer necessary.

The PC linkage offers some additional performance benefits over SVC linkage. First, a subsystem can selectively grant users access to its PC routines. Therefore, it need only perform authority checking when the user first requests access to its services. This reduces

the overhead associated with repeated checking. Also, each PC routine can now define an associated recovery routine (ARR) to provide an error recovery environment. Therefore, the PC routine does not need to reinitialize its recovery environment on each invocation.

SAS SYSTEM ENHANCEMENTS UNDER MVS/ESA

The three principal enhancements outlined in **MVS/ESA FACILITIES** form the basis of the exploitation strategy for the SAS System under MVS. Current research is focusing on

- using the very large in-memory storage addressability to reduce I/O
- using data-in-memory to extend function
- using the new Program Call (PC) versions of system services to improve performance
- using the enhanced PC linkage mechanism.

Using In-Memory Storage to Reduce I/O

Using the very large in-memory storage addressability to reduce I/O offers the primary performance benefits for the SAS System in the MVS/ESA environment. These categories of data benefit:

- nonshared user data
- shared read-only data
- data shared for concurrent update.

Data in the first category include the SAS WORK file and a user's temporary and permanent files. Data in the second category include SAS System data sets such as the message file and various catalogs. Data in the third category include SAS data sets that several users share concurrently for update access. For each category, the SAS System can reduce its demand for I/O by caching members of data sets into memory where it can reuse them many times. For temporary data sets, all of the processing can be done in memory, eliminating I/O completely.

Nonshared user data Nonshared user data can use MVS/ESA hiperspaces as a foundation on which to implement a SAS data set access method for in-memory data sets. The hiperspace structure closely parallels the existing I/O model for blocked data sets, using a 4-kilobyte blocksize. The WORK file and other temporary files are good initial candidates for hiperspace support. Such support can completely replace the existing access method and eliminate its I/O overhead.

For a user's permanent SAS data sets, using hiperspaces may require a two-phase I/O engine, one phase for accessing the data sets on external storage and another for managing them in memory. A better alternative may be to build SAS data sets in such a way that the SAS System can load them directly and transparently into hiperspaces. This would allow the SAS System to process the data sets similarly to the method described earlier for temporary data sets. As outlined in **MVS/ESA FACILITIES**, data windowing services enable a program to map data automatically from a VSAM linear data set into a hiperspace. Therefore, it may be desirable to implement a new SAS data set engine that formats data to a VSAM linear data set to capitalize on this feature of MVS/ESA.

Another enhancement to the SAS System that may depend on using VSAM linear data sets is for external file support. Many products that run under MVS are migrating towards the use of VSAM linear data sets. New and existing user applications can also take advantage of their capabilities. Therefore, data that a user may want to access in a SAS program may reside on such data sets.

Shared read-only data Shared read-only data include a number of SAS System objects that many users access but never update. Examples of such objects are messages, help windows, SAS/ASSIST® screens, and maps. If access to these objects on external storage is frequent, moving the data into memory can provide a significant performance benefit.

In order to share the data stored in memory, a new SAS System component is necessary to provide access to it. This component acts as the owner of the data. It reads the data from external storage and places them into a data space or hiperspace or sends the data to VLF. The component provides a routine that users call to get authority to access the data. If the routine successfully validates a user's authority, it grants access to a set of other routines that the user calls to read the data.

Any of the four facilities for supporting very large in-memory storage addressability can serve to provide repositories for shared data. Each has advantages and disadvantages. Data spaces, because they offer byte addressability to data, provide a good location to store small and irregularly sized objects. But reading the data requires that a program run in access register mode, which is relatively cumbersome and implies the use of assembler programs. Hiperspace access works well with large objects, but it is best suited for processing objects organized into blocks of equal size. A data windowing services implementation supports the automatic mapping of data from external storage into a hiperspace, but only if the data reside in a VSAM linear data set. Any of these facilities requires implementing accounting mechanisms for identifying what to keep in memory if storage is constrained. Using VLF eliminates the need to account for an object's location and usage, but VLF services are only available if its subsystem is running. As research continues, it may show that it is necessary to utilize several of these facilities to best meet the different requirements of various applications for sharing data within the SAS System.

Data shared for concurrent update Data shared for concurrent update include an installation's SAS data sets to which multiple users both read and write observations. The SAS/SHARE® server provides support for access to this category of data. The server actually owns the data and acts as a single user of them. Therefore, using the data-in-memory facilities for nonshared user data also offers support for this category.

Using Data-in-Memory to Extend Function

Using the new MVS/ESA capabilities to extend function includes the utilization of the very large in-memory support for other benefits. Data spaces in particular are quite versatile. For example, SAS procedures that use large structures or tables could store them in data spaces. SAS tasks such as the program editor and the log could isolate their data in data spaces to enhance system reliability. Data sets that are too large to reside in memory could still keep their observation indices in data spaces.

However, the nature of referencing data spaces through the use of access registers makes implementation of functions to use them somewhat complex. An important area of research is in enhancing the SAS/C® compiler to support data space addressability. Compiler support for access register mode execution would accelerate the conversion of existing functions to support data spaces by reducing the need to rewrite them in assembler.

Using the New Program Call (PC) Versions of System Services to Improve Performance

Using the new Program Call (PC) versions of system services to improve performance involves minor enhancements to the SAS System under MVS. The SAS System can issue a PC where it currently issues a supervisor call (SVC) to request those system services.

Using the new versions is relatively easy because the SAS System localizes most calls to system services in one module. This enhancement may reduce the SVC interrupt overhead for running SAS.

Using the Enhanced PC Linkage

Using the enhanced PC linkage forms the basis for the planned implementation of much of the ESA/370 support by the SAS System under MVS. For example, current research into developing a new SAS System component for sharing read-only data suggests its implementation as an MVS subsystem. This subsystem would provide its services through PC routines. The PC routines would perform the functions of user authorization checking, data access, and resource accounting. While some of the code in such a subsystem would have to execute authorized, very little of that code would run on behalf of the end user. The subsystem would have to be started in order for its services to be available. Operation of the subsystem would be optional; if it is unavailable, the existing mechanism for accessing the data, namely through I/O to external storage, would still function.

Even new SAS System capabilities that do not require the support of a subsystem may still use one. For example, in-memory support for the WORK file could use a hiperspace that each user allocates and owns. However, this approach does not allow for any regulation of user requests for system storage. Therefore, it may be appropriate to use a subsystem that manages resource utilization and governs the use of hiperspaces by SAS users. Criteria for limiting access to prevent overutilization could be to restrict the capability to certain users or to respond to hiperspace allocations on a first come first served basis.

A possible long-range objective for a SAS subsystem would be for it to operate as a multiuser server. The server would provide access to all of the functionality of the SAS System to non-TSO requestors, such as workstations. It would localize all mainframe SAS processing to one address space and would segregate user data into data spaces assigned to each user. Potentially, the server could off-load to workstation requestors such SAS functions as those for preprocessing data requests and for formatting display windows.

IMPLEMENTATION ISSUES

Several issues may influence the utilization of ESA/370 by the SAS System under MVS. The primary one is the availability at your site of an MVS/ESA system. None of the new capabilities is available without this version of MVS.

A second issue is the requirement for a subsystem. A subsystem requires that your data center operations staff start it and monitor it to ensure its availability. Also, some portions of the code must run in an authorized state. You will have to determine if your site can support such a subsystem.

Another issue involves priorities. This paper describes a number of possible new enhancements to the SAS System. Each enhancement will take time to implement. You may want to decide which features are the most beneficial.

An important concern is the potential for excessive memory resource utilization. For the case of user data that formerly went to external storage, its migration to processor storage to reduce CPU and channel overhead may increase the demand on processor memory and system paging. For shared data, the actual cost of additional memory may be minimal, because storage for the shared objects may reduce the requirements for individual user copies of the objects.

CONCLUSION

The ESA/370 architecture offers many potential benefits for improving the performance and function of the SAS System under MVS. The implementation of ESA/370 capabilities will proceed over the next several releases of Version 6 of the SAS System. As IBM adds additional capability to MVS/ESA, and as users suggest requirements, the implementation strategy will evolve. Version 6 of the SAS System provides an excellent foundation for delivering ESA/370 capabilities to your users. A key objective of this paper is to outline our current research to give you an opportunity to review it and perhaps suggest your priorities for delivering the most important capabilities to your users first.

REFERENCES

The following references describe the capabilities of ESA/370 and MVS/ESA:

IBM Corp. (1988), *IBM Enterprise Systems Architecture/370 Principles of Operation* SA22-7200, Poughkeepsie, NY: IBM Corporation.

IBM Corp. (1989), *MVS/ESA Application Development Guide* SC28-1821, Poughkeepsie, NY: IBM Corporation.

IBM Corp. (1989), *MVS/ESA Application Development Macro Reference* SC28-1822, Poughkeepsie, NY: IBM Corporation.

IBM Corp. (1989), *MVS/ESA System Programming Library: Application Development Guide* SC28-1852, Poughkeepsie, NY: IBM Corporation.

IBM Corp. (1989), *MVS/ESA System Programming Library: Extended Addressability* SC28-1854, Poughkeepsie, NY: IBM Corporation.

IBM Corp. (1989), *MVS/ESA System Programming Library: Application Development Macro Reference* SC28-1857, Poughkeepsie, NY: IBM Corporation.

IBM Corp. (1989), *IBM Systems Journal*, Vol. 28, No. 1, G321-0093, Armonk, NY: IBM Corporation.

Mullen, J. William (1989), "MVS/ESA Architecture, Modeling and Performance,"*Mainframe Journal*, Volume IV, Number 10, Dallas, TX.

Mullen, J. William (1990), "MVS/ESA Dataspaces and Hiperspaces," *Mainframe Journal*, Volume V, Number 1, Dallas, TX.

National Systems Programmer Association Inc. (1989), *Technical Support*, Vol. 3, No. 11, Milwaukee, WI.

ACKNOWLEDGMENTS

Special thanks to Gary Burchett and Dan Berry of SAS Institute for assistance in developing this paper.