# Porting DPPX from the IBM 8100 to the IBM ES/9370: Feasibility and overview

by R. Abraham
B. F. Goodrich

***The DPPX/SP operating system was converted from its original implementation on the IBM 8100 Information System architecture to a new implementation—DPPX/370—on the System/370 architecture of the ES/9370 Information System processors. Portability was not an original design objective for DPPX, and yet the conversion of the operating system was straightforward and successful. This paper investigates the design fundamentals and technical approaches that led to the successful porting of DPPX/SP to the ES/9370.***

In 1978 IBM announced the IBM 8100 Information System and the Distributed Processing Programming Executive (DPPX) operating system. These products were designed to provide distributed processing capabilities in a centrally controlled and managed network. The 8100 processors were relatively inexpensive, occupied little space, and were capable of being run in an operatorless environment. The control in an operatorless environment was done by providing such features as autoIPL after power failure, software controlled reIPL, and extensive error notification prior to hard failure. DPPX was similarly designed to operate in a centrally controlled network. No programmer or operator was required at the remote sites. This capability was unique within the IBM product line and generally throughout the industry. These hardware and software products were improved and expanded up through the announcements of the 8150 multiprocessor in 1983 and the Distributed Processing Programming Executive System Product (DPPX/SP), Release 4, in 1987. DPPX/SP had by then grown to over one million lines of code. At this time, there were many hundreds of customers worldwide running a total of many thousands of DPPX systems, some with individual networks of hundreds of DPPX nodes. The smooth operation of their businesses was dependent on DPPX/SP and the 8100.

A typical network is shown in Figure 1. In this example, the host system is an MVS (Multiple Virtual Storage) complex. There is one or more central 8100 system for application development, initial service installation, and pilot testing. The central system is used to create a master copy that is replicated to the remote systems. The remote network consists of many 8100s. Corporate data can be kept in the remote processors or may be kept in master files at the host. Transactions are run in the 8100, and they in turn can interact with host transactions on either the Information Management System (IMS) or the Customer Information Control System (CICS). Additionally, terminal users on a remote system can directly access applications on either the host or other remote systems. Network management programs on both the host and the remote systems are coordinated with one another to keep the central network operator informed of any problems. A more detailed overview can be found in Reference 1.

During this same time period (1983 to 1986), IBM was struggling with having too many hardware and software systems with similar processing power and function. In addition to the 8100 there were the System/36, the System/38, the Series/1, the 4300 family, and even the personal computer, with a total of ten operating systems among them! Other products had in fact added many of the features that had made DPPX and the 8100 unique.

Beginning in 1986, IBM made four significant announcements: (1) In March 1986 IBM announced that the 8100 hardware line would not be expanded. IBM would investigate moving DPPX/SP to System/370 processors. (2) In October 1986 the IBM ES/9370 processors were announced. These provided many of the same features that made the 8100 attractive for distributed processing but within the framework of System/370 architecture. (3) In March 1987 IBM announced that the investigation of moving DPPX/SP to System/370 was successful and IBM intended to provide a version of DPPX that ran on the ES/9370 processors in 1989. (4) In March 1988 IBM made a formal product announcement of DPPX/370, the version of DPPX that executes on the ES/9370 family of processors. DPPX/370[2] was made available to customers in December 1988.

Moving DPPX/SP to the ES/9370 family offered customers the advantages of preserving their investment in software and training while at the same time allowing them to use the System/370 family of processors, a family with large growth potential and capable of supporting several different operating systems.

Portability was not an objective of the original DPPX design. The decision to do the port was done only after extensive examination of the alternatives and thorough planning of how the port would be accomplished.

### The alternatives

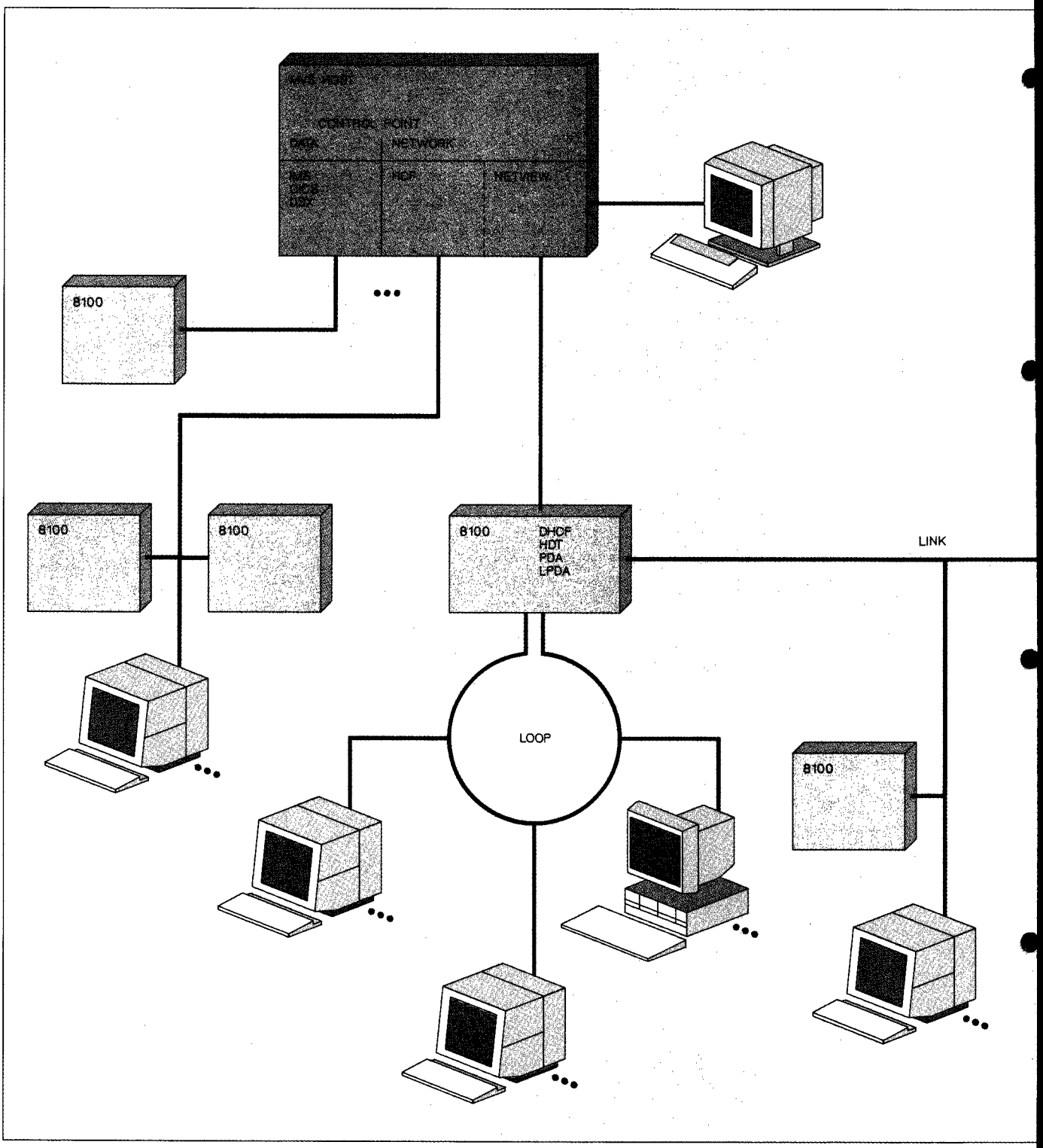
Before deciding that DPPX/SP must be ported to System/370, extensive analysis was done on the feasibility of providing migration tools that would allow customers to migrate to other IBM systems. DPPX/SP customers have large investments in their application programs. Some customers have many millions of lines of application code. Any alternative to DPPX/SP would have to provide a clear and clean migration from DPPX/SP. A number of alternative IBM operating systems were investigated.

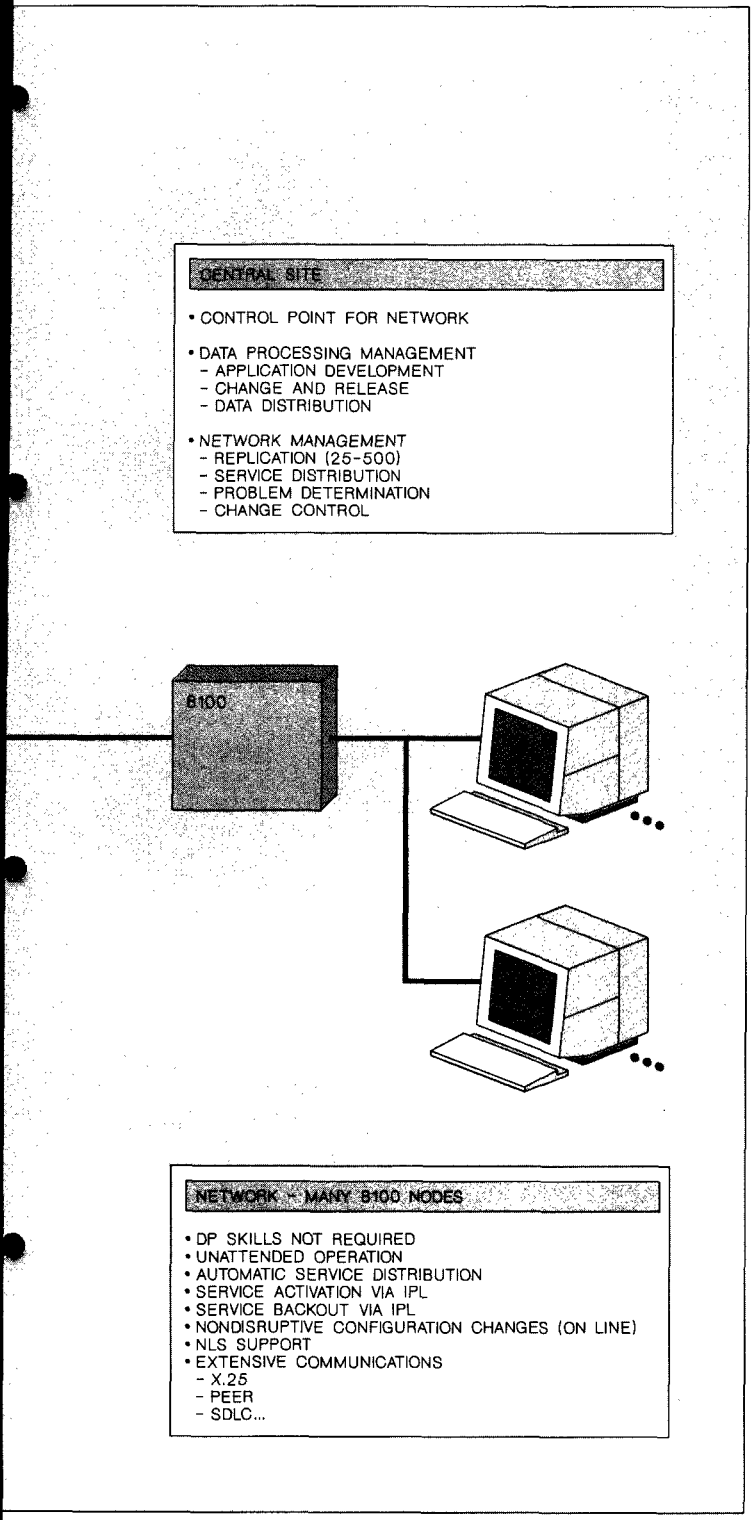
There are many attributes of operating systems that make each one unique. During the study several major areas stood out that prevented us from providing a general migration path from DPPX to another operating system. Typical customer applications were structured in ways that depended on the following facilities.

**DPPX Transaction Processing Manager (TPM) allows a large number of users to share relatively limited storage resources.**

- *Data Base Manager.* DPPX Data Base Manager (DBM) provides indexed, sequential, and random access to data. It allows up to eight indexes and does not require any to have unique keys. When index tree structures become unbalanced or when room to insert new records is no longer available, the index or data structure is reorganized automatically, while still on line.
- *Transaction Processing Manager.* DPPX Transaction Processing Manager (TPM) allows a large number of users to share relatively limited storage resources. It provides recovery scope management, which allows all actions taken by a transaction to be viewed by all other users as a single, atomic operation. That is, all database updates are made visible together when the transaction completes successfully, and none is made visible if the transaction signals an error or if the system should fail. It provides the ability to keep a transaction's recovery scope open across several terminal interactions. It provides the ability to create work for another user or terminal (or printer), all under the control of the originating user's recovery scope.
- *Display Manager.* DPPX Display Manager allows programmers to interactively describe the layout for variable data and static descriptive text. It saves this information in data sets known as *maps.* When the application program runs, the display manager merges maps with the variable data pro-

**Figure 1 A typical DPPX network**






vided by the application program to write panels on display terminals and output to printers. For input operations it does the reverse, separating data entered at the terminal from the static text and delivering the data to the application in an easy-to-use format.

- *Unattended operation.* There are many functions in DPPX/SP that allow unattended operation. These include components that interact with the DSX,[3] HCF,[4] and NetView™[5] products of the host. DPPX/SP service is provided in a format that allows simple distribution, installation, and, if necessary, backout on remote systems. Service can be installed while the system is on line; it is activated by a remotely requested, software-initiated IPL (Initial Program Load). If the newly provided programs fail, a reIPL can be automatically triggered to reload the old version of the system. Disk shadow volumes can be provided to protect against disk failures.
- *Embedded commands.* DPPX/SP commands can be embedded in application programs. As an example, a transaction program can contain a DEFINE.DATASET command, then proceed to write into the data set, and then issue a SUBMIT.BATCH command to submit that data set as a batch job.
- *The router.* The router allows a user at a single IBM 3270 terminal to be logged on concurrently to several different applications on different processors in the SNA (Systems Network Architecture) network and to switch among them at any time. It is in many ways similar to the multiple-session capabilities of the IBM 3270 PC or the IBM 3194 display.

These facilities and other user requirements were matched against other operating systems, and no suitable match could be found. The other operating systems did not have the centrally managed, unattended environment of DPPX, and their application program interfaces were significantly different. The effort for IBM to alter a potential target operating system to provide equivalent support, or for customers to recode around the missing function, was deemed too expensive or disruptive to the customer to be a viable alternative to actually moving DPPX/SP to the ES/9370.

### What makes DPPX/SP portable?

There are many success stories for operating systems that did have portability as an objective, either originally or during some intermediate reimplementation. Perhaps most notable of these is UNIX®.[6] DPPX

instead was designed to be extendable and reliable and was implemented in a manner to provide high productivity. It is these attributes that have made it portable in practice, though not in intent!

The extendability objective led to a highly structured design that provides I/O services via layers of components. The lowest layers deal with physical I/O concerns, whereas the upper layers deal with ever-increasing generality. The layers have highly architected interfaces that force any given layer to be implemented independently of its surrounding layers. The layers for a particular connection between a program and the I/O device are chosen and dynamically bound together at "open time," using information provided by the program and, optionally, the user (Figure 2). A more detailed description can be found in Reference 7.

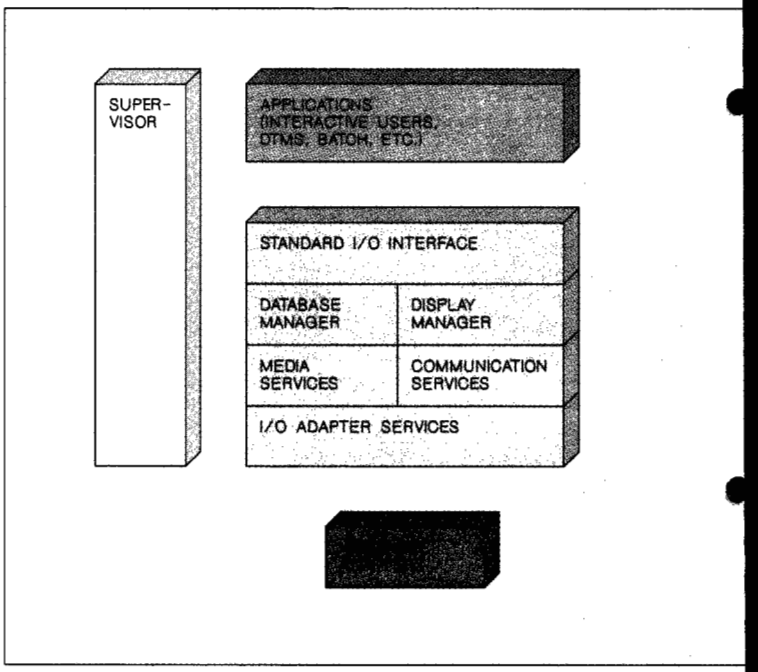
Furthermore, the reliability objective led to a highly structured design. Components tended to encapsulate data that they controlled, not externalizing their internal structure to other components. Interfaces between components were strictly monitored during the development of DPPX/SP to prevent numerous private protocols from being invented.

Last, the high productivity objective led to the exclusive use of a single high-level language for implementation. The language used was PL/DS, available externally as part of the Distributed Processing Development System.[8] This, plus extensive rules for standardized module prologs, copious comments, and structured programming led to a standardized programming style across a large programming organization.

DPPX benefitted from these objectives long before the port to the ES/9370 was undertaken. Major functions and complex hardware support were continually added to DPPX/SP for a relatively small amount of programming cost. Some of the functions include the following.

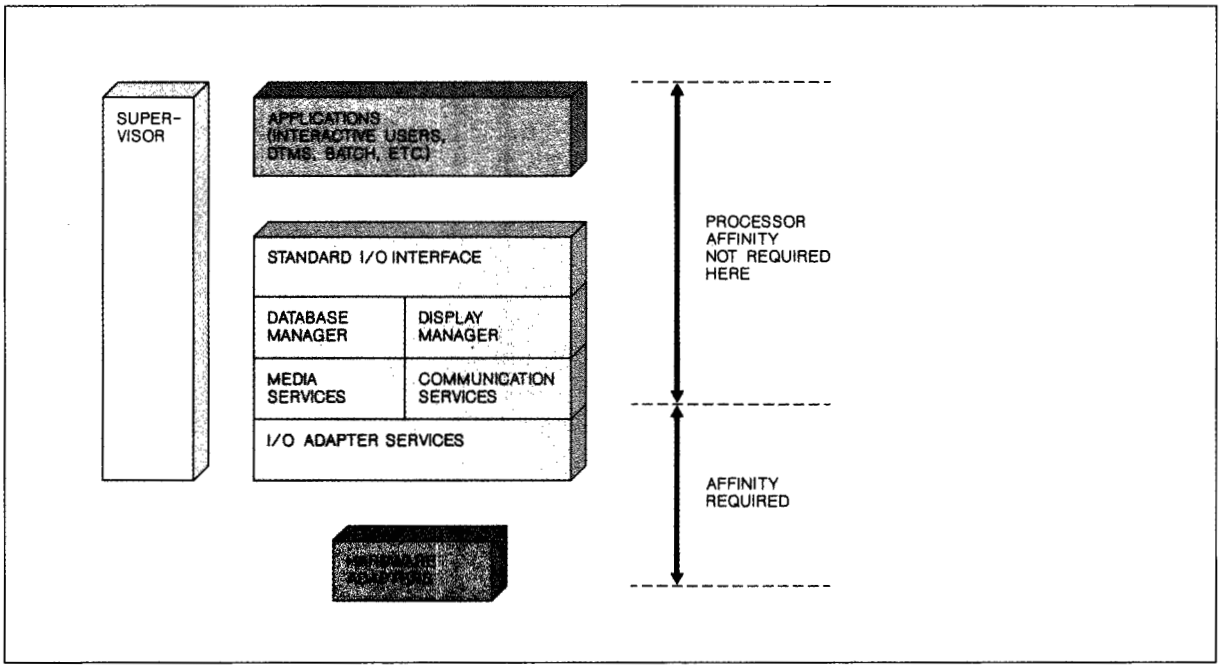
- *Added support for attached processor and multiple processor architectures.* The strict interfaces requirements isolated components that managed the processors and the actual I/O devices. The changes required to assure that I/O requests were executed on the processor side that attached the desired device were limited to just these two areas (see Figure 3). There were no changes required in application programs.

**Figure 2 I/O layer structure**




- *Change in address space structure.* The 8140 Model C processor quadrupled the amount of logical storage available, and the 8150 processors changed the storage protection scheme from a nested format to a keys-and-locks format. Both of these changes were accommodated for relatively minor amounts of programming because of the strict adherence to component boundaries. Again, there were no changes required in application programs.
- *The insertion of new layers to provide additional function.* The disk I/O paths have had two additional layers added without requiring any change to user programs and minimal changes elsewhere (see Figure 4). One new layer provides the DASD cache function, increasing performance by using otherwise unused main storage to hold an image of most recently used disk records. The other new layer provides the DASD shadow function, which provides increased reliability by duplicating one volume on a backup (shadow) volume. All writes are done to both volumes; reads are done from either. In the event of a disk failure, the remaining volume is automatically used until the broken one can be repaired. Once again, there were no changes required in application programs in order to take advantage of these additions.

**Figure 3 Minimal changes required for multiprocessing**




- *A new communications protocol.* The OSI (open systems interconnection) X.25 communications protocols were inserted without the need to change major portions of the other communication I/O layers.
- *Tolerance toward personnel turnover and changes of responsibility.* The programmers and sometimes even development locations for nearly every module of DPPX have changed since DPPX was first implemented. The single high-level language and strict adherence to programming standards have made those changes much less disruptive than they would otherwise have been.

The extendability and reliability objectives had the same effect for the port as they did for the 8100 product. Although the System/370 architecture of the ES/9370 processors is radically different from the 8100, the components needing major changes were limited to a relative few (Figure 5). These were:

- *Supervisor.* The major areas needing change involved interrupt handlers and serialization, storage mapping, and timer formats.
- *I/O Adapter Services (IOAS).* Changes were needed for different status reporting mechanisms from devices, different low-level command formats and functions, and use of real addresses for I/O rather than logical addresses.
- *Data management.* The fixed block architecture (FBA) for System/370 disks did not support the 256-byte block size that was standard for 8100 disks. As a result, the lower layers of data management were changed to simulate the 256-byte interface to upper layers when necessary.

Similar changes were required in the DPPX/SP standalone utilities. The extensive use of a Programming Language for Distributed Systems (PL/DS) allowed most components to be recompiled using a new PL/DS compiler targeted for System/370. Since the PL/DS language allowed programmers to "drop down" to machine instructions (within a macro) for functions not supported by the language, some programs had to have these few lines recoded.

In the course of porting, other changes were made to the operating system. Most of these were done to remove previous implementation constraints. For instance, the size of an important systemwide identifier (the environment id) was changed from 8 to 16 bits. Although not strictly required for the port, this was done in recognition of the full range of processing power available in the System/370 product line.

This change, although bothersome at the time, would be a major change to introduce if needed at a later time. It was quite simple to do when the entire system was being recompiled and retested anyhow. Another such change was for the DATE function to return four digits of the year rather than just two. The year 2000 looms near, and it is no longer acceptable to assume the year falls within the 1900s!

The remainder of this paper describes the feasibility and results of porting this large operating system.

## Feasibility of porting

**Code classification.** An early feasibility study of porting DPPX examined each DPPX/SP base system component (and licensed programs) and identified areas with the following requirements:

- *Redesign* (high-level design necessary for hardware, RAS [Reliability, Availability, and Serviceability], etc.; inspection of component structure required because of changed module-to-module interfaces)
- *Rewrite* (only low-level design necessary; some changes necessary for hardware, RAS, etc., but no change to module-to-module interfaces)
- *Recompile* (little or no change, affected only by items in a checklist)

Estimates were made of thousands of lines of code (KLOC) in each category per each system component. These were continually refined throughout the implementation cycle. An early estimate for the DPPX/370 base system (excluding licensed programs, such as the COBOL compiler) is shown in Table 1.

**Programmer productivity.** An estimate of programmer productivity rates is shown in Table 2. These productivity rates translate to an overall productivity rate of 6204 shipped source instructions (SSI) per programmer year, or 1440 changed source instructions (CSI) per programmer year. This includes development activity for high-level design, low-level design, code, functional verification (FV) test, independent component test,[9] and support of system test, based on the following assumptions:

- Redesign is the same as creating anew and could occur at the prevailing rate for DPPX/SP 8100.
- Rewrite could occur at a faster rate due to a well-defined list of changes to be made to well-structured and well-commented code by an experienced team familiar with each component. Also, only the changed code flows would be FV tested.

**Figure 4 Insertion of new layers**

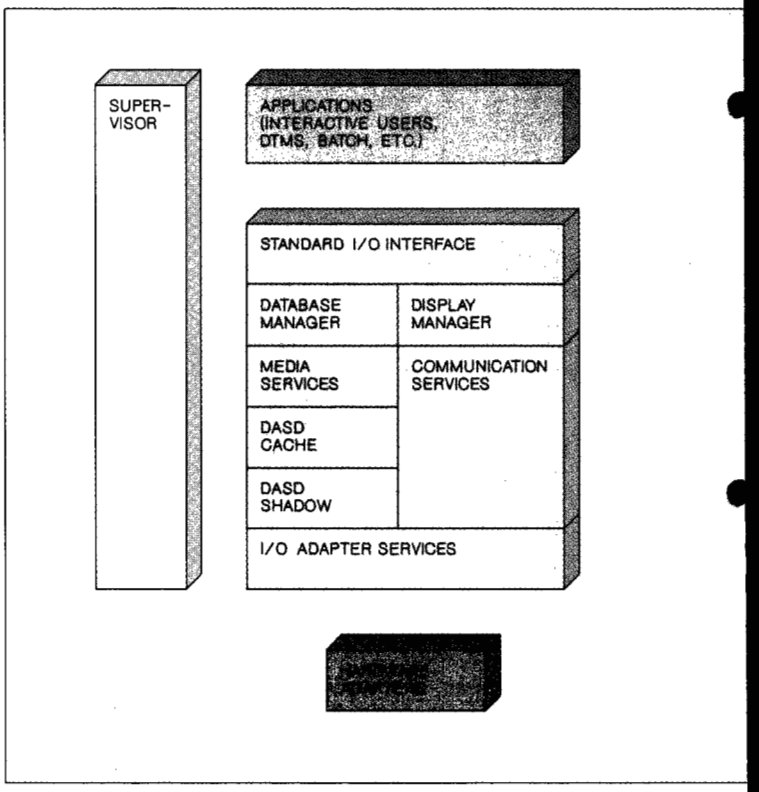



**Table 1 An early estimate for the DPPX/370**

| KLOC Recompile | KLOC Rewrite | KLOC Redesign |
|---|---|---|
| 796 | 55.2 | 124 |

**Table 2 Estimate of programming productivity**

| Activity | KLOC per Programmer Year |
|---|---|
| Redesign | 1.3 |
| Rewrite | 1.9 |
| Recompilation | 24.0 |

- Recompilation could occur at a very fast rate. This was based on the following:
  - A reliable System/370 compiler would be available.

- No code changes would be necessary to the majority of DPPX modules for a successful compilation.
- No specific FV test would be required for these recompiled modules.
- No additional system interface changes would be needed other than the few originally defined.

### Implementation compiler

**Compiler choices.** Various technical approaches had to be considered. One of the most important to the success of the project was the choice of the implementation compiler.

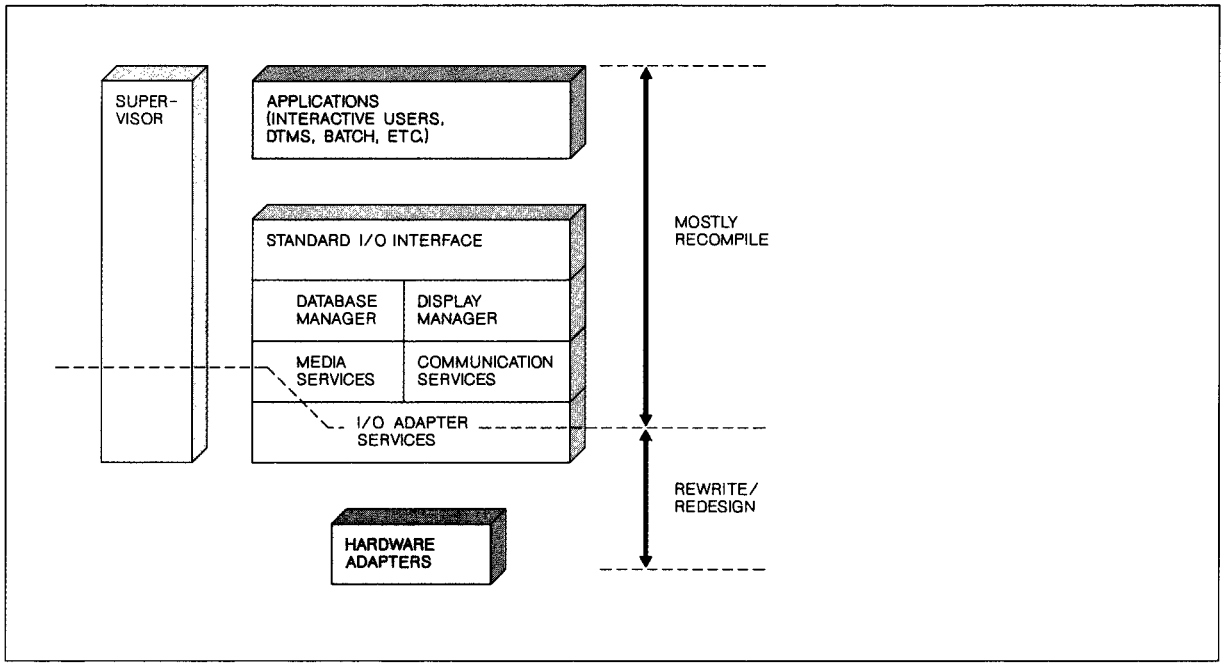
Over 90 percent of DPPX/SP 8100 had been coded using the Programming Language for Distributed Systems (PL/DS), which was also available to DPPX/SP customers as Programming RPQ P88016.[8] The PL/DS language provides PL/I-like coding power and is very similar to the Programming Language/Systems (PL/S) and Programming Language/Advanced Systems (PL/AS) languages used internally by IBM to produce other System/370 operating systems and control programs. It is a high-level programming language with preprocessor-like macro support, and provides simplified intramodular linkage for programs that will be executed under DPPX/SP. PL/DS provides an optimization option that generates highly optimized code, allowing a skilled PL/DS programmer to produce very efficient code.

A task force consisting of DPPX/SP designers and developers, a PL/DS expert, and a PL/AS expert defined the recompilation requirements and considered three compiler alternatives.

The recompilation requirements were:

1. Generate optimal System/370 object code.
2. Support structured declarations of registers of length 1, 2, or 4 bytes.
3. Map 8100 registers to System/370 registers and generate System/370 code sequences that simulate the register operations of the 8100.
4. Support assignment and comparison statements of all bits (that is, not just 1 bit and multiple of 8 bits).
5. Support a small set of 8100 built-in instructions (BIIs), based on frequency of use in existing DPPX/SP modules considered as recompilation, and generate System/370 code sequences that simulate their operation. A built-in instruction is a PL/DS compiler statement that corresponds to a

**Figure 5 Changes required for conversion to ES/8370**



machine instruction. The operands of the BII are PL/DS expressions.
6. Support DPPX/SP linkage conventions (that is, CALL/END/LINK/EXIT and DPPX PROCEDURE statement options) and generate System/370 code sequences that simulate the action of the conventions of DPPX/SP.
7. Support 8100 DECLARE defaults—that is, default declare precision of BIT(15).

The three compiler alternatives considered were:

1. Get the PL/AS compiler to support all DPPX/370 requirements.
2. Get minimum PL/AS compiler support and develop macros and tools to convert DPPX/SP source from PL/DS to PL/AS.
3. Get the PL/DS compiler to support all DPPX/370 requirements. This meant retargeting the PL/DS compiler to generate System/370 Assembler (the PL/DS compiler was in the process of being restructured). This is the alternative that was chosen. The second version of the PL/DS compiler is called PL/DS2.

The decision to develop and use the PL/DS2 compiler was made on the basis of the following objectives:

- Minimize development cost for recompile effort (that is, the number of DPPX/SP modules requiring manual rewrite must be kept to a minimum).
- Minimize the risk to quality, schedules, and cost.
- Maximize the overall DPPX/370 development resource (that is, minimize cost of compiler, macros, and conversion tools).
- Maximize the performance of DPPX/370 on the ES/9370 hardware.

Following the completion of the task force, a compiler development group was formed and a group consisting of DPPX designers and developers, DPPX tool support personnel, and PL/DS compiler personnel developed a detailed set of requirements for the PL/DS System/370 compiler (called PL/DS2). These requirements addressed the following areas:

- Support for key 8100 DPPX/SP conventions, including mapping of 8100 registers on System/370 hardware and linkage conventions
- PL/DS language compatibility
- New language requirements
- New linkage convention requirements
- Support for System/370 operations
- Support for built-in instructions (BIIs), both 8100 and System/370
- Compiler optimization

**Compiler development.** Compiler development proceeded in parallel with the development of DPPX/370.

**Compiler development proceeded in parallel with the development of DPPX/370.**

Incremental versions of the PL/DS2 compiler were made available, as follows:

1. Most language function supported
2. Some BII support, some optimization
3. All language function and BIIs supported, additional optimization
4. Significant optimization
5. Some incremental optimization

DPPX/370 development personnel knowledgeable with DPPX linkage conventions developed the compiler function for the linkage conventions for DPPX/370 (6000 lines of compiler code).

In addition to the incremental versions, a "throw-away" version of a PL/DS2 compiler was used to identify parts of DPPX impacted by the port. The "throw-away" version of the compiler generated unique messages when a specified condition was encountered—for example, multiplication of FIXED(15). The entire DPPX/SP system (over 5000 modules) was compiled with the "throw-away" PL/DS2 compiler, and the compiler-produced message files were scanned for the condition under study.

Also, as incremental versions of the PL/DS2 compiler became available, the entire DPPX/SP system was "quick compiled" and scanning of compiler-generated messages occurred. This was done to assess the overall extent of the DPPX recompilation effort and to identify which DPPX components were significantly affected. The first occurrence of the "quick

compile" showed that 2123 DPPX/SP modules, out of 5158 total, successfully compiled and assembled using the PL/DS2 compiler (using only 32 macros modified for DPPX/370). The third (and last) occurrence of the "quick compile" showed that 3632 DPPX/SP modules successfully compiled and assembled (using 100 macros and 75 control blocks modified for DPPX/370). This activity also allowed assessment of incremental PL/DS2 compiler improvements. Note that all "quick compiles" and the use of a throw-away compiler were auxiliary to the real process of porting code to DPPX/370, since they were performed with minimal development programmer involvement and only the PL/DS2 compiler and System 370 Assembler message files were saved (not the listing and assembler output files).

**Compiler optimization.** When the first version of the compiler became available, there were over 600 DPPX/SP modules, all of which had to execute as re-entrant with no automatic storage, that failed to compile. Re-entrant modules provide the ability to have a single copy of a module be invoked concurrently by two or more tasks. Re-entrant modules with no automatic storage are usually invoked very frequently and can not afford the performance overhead of allocating and deallocating automatic storage each time the module is invoked, or they are a part of the supervisor core that does not have storage management services available to it. Re-entrant modules with no automatic storage must have all module variables assigned to hardware registers by the compiler. (A register can have more than one variable assigned, depending upon where in the module each variable is used.)

The compiler was unable to assign all of a module's variables to registers because the level of compiler optimization available used too many registers for intermediate calculations. Also, there is one less register available for compiler use on System/370, as compared to 8100, because one of the 16 registers has to be used as a base register.

If a frequently invoked module cannot be compiled as re-entrant with no automatic storage, then one alternative is to change the module to be nonre-entrant and serialize the module's invocation via system disable (the module runs to completion uninterrupted). But this would have serialized the execution of key portions of the DPPX/370 operating system and would not be practical in many cases (some modules invoke system services which have to run enabled).

An analysis of the compiler-generated object code resulted in a set of additional compiler optimization requirements, which were scheduled for implementation in subsequent versions of the compiler, in time to have almost all of these 600 modules compile successfully.

DPPX/370 designers and developers, DPPX/370 tool support personnel, and PL/DS2 compiler personnel met periodically (usually monthly) to discuss additional requirements, problems, compiler usage, etc. Plans for incremental improvements to the PL/DS2 compiler were jointly arrived at and periodically reviewed. Compiler problems found by DPPX/370 development were formally reported and tracked via the same problem-reporting tool used for tracking DPPX problems.

**The performance of the object code generated by the PL/DS2 compiler was closely monitored.**

The compiler group was provided a set of 40 DPPX/370 modules to use for testing of the compiler. Also provided was a particularly large DPPX/370 module that was re-entrant and required zero automatic storage to compile successfully.

The performance of the object code generated by the PL/DS2 compiler was closely monitored. One person kept a single program, an efficiency test case written in PL/DS2, containing numerous situations where the System/370 object code produced by the PL/DS2 compiler needed to be improved to shorten execution time. Whenever an incremental version of the PL/DS2 compiler became available, the program was compiled and the object code analyzed. As developers discovered other instances of object code that could be improved, the efficiency test case was expanded. Requirements were prioritized and added to the PL/DS2 compiler implementation plan to improve the performance of situations in this program.

Throughout the development of DPPX/370, PL/DS2 compiler upgrades were received as significant compiler fixes became available. The upgraded version of the PL/DS2 compiler would be established as the version to be used for all subsequent development of DPPX/370.

Also, many instances of compromise occurred between PL/DS2 compiler development and DPPX/370 development. If a problem prevented a DPPX/370 module from compiling or executing correctly, and a PL/DS2 compiler fix would be difficult, then an analysis would determine whether the PL/DS2 compiler or DPPX/370 should be changed. The impact to DPPX/370 development of how many modules and lines of code change are required (sometimes requiring extensive scanning of DPPX/370 modules)—plus testing, etc.—would be compared with the impact on PL/DS2 compiler externals or scheduled availability of optimization improvements. A joint decision would be arrived at to change either the PL/DS2 compiler or DPPX/370 modules, or sometimes both.

When the first version of the PL/DS2 compiler became available, it was realized that significant optimization improvements would be needed to achieve reasonable performance. This meant that the DPPX/370 system would have to be recompiled when compiler optimization improvements became available.

Two major system-wide recompiles of DPPX/370 occurred during its development. The end result was 25 percent reduction in both DPPX/370 instruction path and module size, and a good quality compiler used for all DPPX/370 modules.

### Checklist

A checklist was developed to provide a list of items a developer must consider when porting an existing DPPX/SP component or subcomponent to DPPX/370, for both recompilation and partial rewrite. This checklist was updated periodically as new items appeared. It listed items for each of the following areas:

- Operating System Architecture changes (for example, the size of an environment ID expanded from 1 byte to 2 bytes, usage of 24-bit addressing converted to 32-bit addressing, page size changed from 2Kb to 4Kb, maximum logical record length increased from 4Kb to 16Kb, etc.)
- DPPX command changes (for example, deleted commands for specific 8100 hardware and devices not supported by System/370, etc.)
- DPPX macro changes (for example, deleted or changed macros due to 8100 System/370 hardware differences, etc.)
- System control block changes (for example, changes to supervisor services control blocks, etc.)
- Error processing (for example, recovery from process checks)
- 8100 built-in instructions (BIIs) and assembly code (for example, only eight 8100 BIIs [out of more than 150] are supported by PL/DS2, changed fullword division routines, etc.)
- PL/DS2 source language differences (for example, higher-precision arithmetic, implicit truncation, compiler-created temporaries for CALL parameters, different result for some signed-unsigned comparisons, etc.)
- PL/DS2 linkage convention differences (for example, base register [called CODEREG] per each 4Kb of module code storage, changed format of module save area, labels on secondary entry points, etc.)

A list of DPPX modules impacted was developed for the majority of checklist items via automated scanning of DPPX module source code, scanning DPPX module compilation listings produced by both the PL/DS compiler and early versions of the PL/DS2 compiler.

### Choice of COBOL compiler

The DPPX COBOL compiler supports a dialect of COBOL that is unique to DPPX. There are two reasons for this. First, the compiler was written to compile the COBOL language as specified by the then-proposed 1978 standard. The standard was later adopted, but not before some very subtle but important changes were made to the proposal, mainly dealing with such scope-ending phrases as END-READ.

In addition, the compiler supports extensions to the language that are unique to DPPX. These extensions were added to integrate the Transaction Processing Manager and Data Base Manager control statements closer into the application programs.

Two approaches were examined to handle these COBOL problems. The first was to recode the DPPX COBOL compiler to produce System/370 object code and then to compile it using PL/DS2 like the rest of the system. The other alternative was to convert the existing System/370 COBOL compiler, VS COBOL II, to run on DPPX/370. The latter approach was chosen so as to provide closer affinity to the COBOL compiler available to our customers on their MVS host. This

approach also provides a cross-compilation capability, allowing DPPX/370 programs to be compiled on System/370 VM and MVS hosts.

The COBOL II compiler and run-time library were modified to request system services using the DPPX macros rather than the MVS macros in the existing product. The language incompatibility problems were solved by providing a preprocessor that can convert the DPPX incompatibilities and extensions into a form that can be successfully compiled and run by the modified VS COBOL II compiler and run-time library.

**DPPX/370 is designed to run on today's ES/9370 class machines, including the IBM ES/9370 Model 90 processor.**

### System performance

The main impact on the performance of DPPX/370, versus 8100 DPPX/SP, comes from the different hardware instruction set. Specifically, if a DPPX/SP function took $n$ 8100 instructions, then how many System/370 instructions would be required for the same function? An early study examined five modules that existed in both a DPPX 8100 licensed program and a System/370 licensed program. The programs were written in a common subset of the PL/DS and PL/S languages. All five were compiled with PL/DS (which generated 8100 object code) and PL/S (which generated System/370 object code) and a comparison showed:

- System/370 object code size, in bytes, was approximately 25 percent larger than 8100 (due to predominance of 4-byte instructions on System/370 compared with 2-byte instructions on 8100).
- System/370 object code took approximately 18 percent fewer System/370 instructions than the corresponding 8100 object code (due to the more powerful System/370 instruction set).

These numbers assumed a 10 percent penalty on System/370 for byte and halfword register manipulation. Also, it was assumed that the PL/DS2 compiler would be as efficient as that of PL/S.

A separate comparison of 8100 and System/370 ES/9370 hardware instruction timings concluded that the midrange ES/9370 models were about as powerful as the high-end 8150B processor, in terms of instruction processing capability. Thus, no drastic redesign of DPPX was necessary to achieve an increased system throughput with DPPX/370 executing on the most powerful models of the ES/9370. The supervisor functions GETMAIN/FREEMAIN and LOAD/DELETE are the only exceptions to this.

DPPX/370 is designed to run on today's ES/9370 class machines, including the largest model, the IBM ES/9370 Model 90 processor. Performance analysis of existing DPPX/SP systems showed that in some configurations a large amount of system time was spent in GETMAIN/FREEMAIN processing. The Model 90 provides significantly more throughput than the IBM 8150B. It was feared that "large system effects" might start appearing when running on this model. Since storage management needed redesigning anyhow to account for the differences in the logical storage mapping mechanism between the IBM 8100 and the IBM System/370 architectures, performance enhancements were included. DPPX/SP had a simple linked list structure for maintaining free storage blocks in ascending address order. The DPPX/370 implementation uses a similar structure but maintains multiple pointers into the list to locate the first entry greater than or equal to specific sizes. These sizes were determined by analysis of DPPX/SP trace data.

It was also discovered that DPPX/SP systems running large Cross System Product[10] applications tended to spend a large amount of time in the supervisor LOAD/DELETE functions, even if the modules were already resident or were not actually deleted. Changes were made to these functions to execute much faster paths, both in the event that a LOADed module was already resident and when a DELETEd module would not be deleted (either because of other users or because it had been preloaded during IPL). Essentially, the appropriate control block updates are done while executing disabled rather than going to the expense of obtaining software locks to control concurrency.

Table 3 Total lines of code to develop the base DPPX/370 system

| Item | KLOC Recompilation | KLOC Rewrite | KLOC Redesign |
|---|---|---|---|
| Original estimate | 796 | 55 | 124 |
| Actual | 782 | 80 | 299 |

Table 4 Planned and actual productivity rates

| Item | CSI KLOC per Programmer Year | SSI KLOC per Programmer Year |
|---|---|---|
| Original estimate | 1.440 | 6.204 |
| Actual | 2.362 | 8.468 |

## Migration

Functions to aid migration were added to the DPPX/370 development effort. This was done to ensure that a customer moving from DPPX/SP 8100 to DPPX/370 would be able to migrate the system and not have to undertake a costly conversion. Migration of customer applications occurs as follows: The DPPX/SP COBOL application programs are preprocessed and recompiled. The Cross System Product applications on DPPX/370 are object-code-compatible with DPPX/SP 8100 (no regeneration is required). The command lists (CLISTs) are copied unchanged to DPPX/370. The Display Manager maps are copied to DPPX/370 (no regeneration is required). The user data and most profiles[11] are copied to DPPX/370.

However, it was necessary to provide assistance to enable easy movement of user applications and data.

Three methods of transmitting customer parts (including customized parts) from DPPX/SP 8100 to DPPX/370 are provided for different customer environments:

1. Via tape, using stand-alone DASD dump/restore (SADDR), which allows dumping of DASD volumes on the 8100 to tape and a subsequent restoration from tape to DASD on ES/9370
2. Via host transmission, using Distributed Systems Executive (DSX) or NetView Distribution Manager (NetView DM), which allows data to be transmitted from DPPX/SP 8100 to DPPX/370 via a System/370 host
3. Via peer-to-peer transmission, using Peer Data Transfer (PDT), new for DPPX/370, which allows data transfer between a DPPX/SP system and DPPX/370 system or between two DPPX/370 peer systems

Refer to "Porting DPPX from the IBM 8100 to the IBM ES/9370: Migration," a companion article in this issue, for additional migration information.[12]

## Results

All of DPPX/370 (including the first- and second-level interrupt handlers) is written in the PL/DS language. Components that had been changed because of items in the checklist did not experience the usual high rate of errors per KLOC associated with changed code. This was attributed to the availability of the checklist with its detailed descriptions of changes required. The result is a System/370 operating system that was developed by reusing more than 800 KLOC of code originally implemented for the 8100.

### Total effort expended

Table 3 illustrates total thousand lines of code (KLOC) required to develop the base DPPX/370 system.

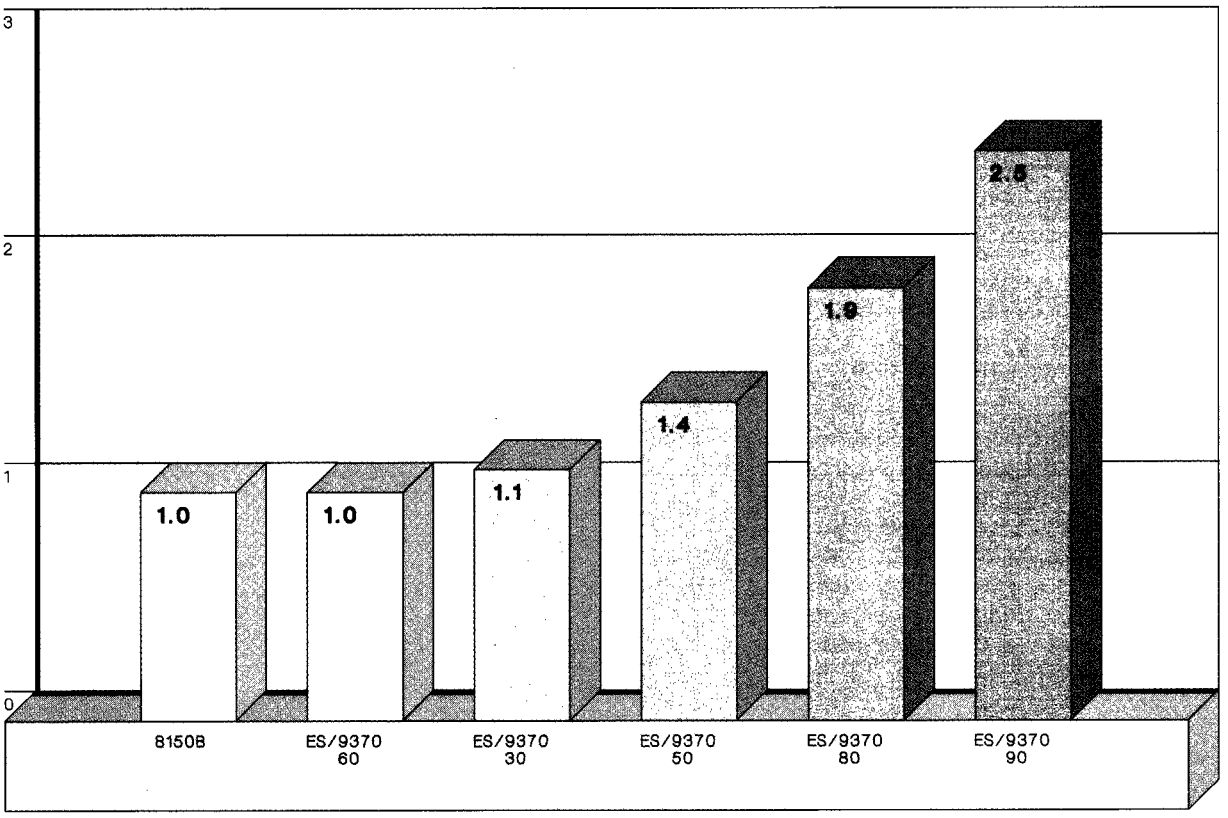
The increase in the KLOC for redesign is due to several new functions added to DPPX/370. Table 4 shows the planned and actual productivity rates, where *CSI* is changed source instructions and *SSI* is shipped source instructions.

The SSI includes the recompiled lines of code. The actual rates include development activity for high-level design, low-level design, code and FV test. In both cases the productivity rate exceeded the original estimate. Some recompilation components were ported at a rate in excess of 30 KLOC per programmer year (versus the original estimate of 24 KLOC per programmer year). Some new components were developed for DPPX/370 at a rate as high as 8 KLOC per programmer year (versus the original estimate of 1.9 KLOC per programmer year).

### Performance

The original comparison of 8100 and ES/9370 hardware instruction timings, which predicted that the midrange ES/9370 processors were about as powerful as the high-end 8150B processor, turned out to be correct. Figure 6 shows this, comparing internal throughput rates ratio (ITRR) measured for DPPX/SP

Figure 6 DPPX/SP 8100—DPPX/370 ES/9370 relative processor performance




on the 8150B processor versus those measured for DPPX/370 on the various ES/9370 models. ITRR is the number of completed jobs or transactions per processor busy second.

### Compatibility

With few exceptions, the functions provided by DPPX/370 are essentially the same as DPPX/SP 8100. These functions have the same user externals on DPPX/370 as on DPPX/SP 8100, so no retraining of end users or operators is necessary. Also, DPPX/SP and DPPX/370 have a similar appearance and can coexist in the same network.

### New functions

A number of new functions were incorporated into DPPX/370:

- IBM token-ring network support, to take advantage of the ES/9370 token-ring adapter
- System Configuration Manager to simplify hardware configuration
- Maintenance via tape, not diskette
- Enhancements to SADDR to improve performance
- Peer Data Transfer (PDT), to aid migrating user data, plus customer maintenance and development within a mixed network. PDT is a batch transfer facility to send data sets, catalogs, and CLISTs between two DPPX systems using communications facilities.
- Enhancements to IPL to improve usability
- Use of display panels by stand-alone I/O functions to improve usability
- Performance functions such as multivolume user catalogs (also known as data striping)
- New printer office support for printing Document Content Architecture (DCA) documents on Intelligent Printer Data Stream (IPDS) printers
- New data management functions—for example, on-line compression of catalogs and logical disk volume

- Migrator tool to assist the customer moving application programs and data from the DPPX/SP system to the DPPX/370 system

A detailed description of these and other functions incorporated into DPPX/370 can be found in Reference 2. The lines of code necessary and the resource needed to implement these additional functions are shown in Tables 3 and 4.

## Conclusion

The port of DPPX demonstrates the unplanned value of structured design beyond the expected benefits of extendability, reliability, and high productivitv.

The port of the DPPX/SP operating system from the 8100 to the ES/9370 processors was made possible due to a number of factors. Originally DPPX/SP had been designed to be extendable and reliable, with highly structured design and data encapsulation, and had been coded in a single high-level language. The PL/DS implementation compiler was retargeted to System/370. There was close cooperation between the development of DPPX/370 and the PL/DS2 compiler, both of which were developed in parallel. An experienced synergistic team with detailed knowledge of DPPX performed the development and test of DPPX/370. The functions used to migrate user applications and data were incorporated into DPPX/370.

As a result, DPPX/370 provides new function as well as DPPX/SP functions with the same user externals, and can coexist with DPPX/SP.

## Acknowledgment

It is not possible to list all the people who contributed to the original design of DPPX, to the analysis of alternatives to the 8100, and to the actual porting. Therefore, we wish to acknowledge the work of all the skilled professionals who have contributed to DPPX since its inception.



## Cited references and notes

1. *DPPX/SP General Information*, GC23-0600, IBM Corporation; available through IBM branch offices.
2. *DPPX/370 General Information*, GC23-0640, IBM Corporation; available through IBM branch offices.
3. DSX is the Distributed Systems Executive product. It manages transmission of data sets between a host system (MVS or VSE) and remote nodes of the network. When used with DPPX, it can also execute command lists (CLISTs) on those remote systems. This product has been superceded by NetView Distribution Manager (NetView DM).
4. HCF is the Host Command Facility product. It allows a user at a host-attached terminal (which itself can be anywhere in the network) to log on to a remote system and appear to that system as a local user.
5. NetView provides centralized error management capability. It provides displays useful to a central network operator to highlight trouble spots throughout the network. It also provides programmed operator capability and allows access to HCF and other applications from NetView DM terminals.
6. D. E. Bodenstab, T. F. Houghton, K. A. Kelleman, G. Ronkin, and E. P. Schan, "UNIX Operating System Porting Experiences," *AT&T Bell Lab Technical Journal* **63,** Part 2, No. 8, 1769–1790 (October 1984).
7. H. R. Albrecht and L. C. Thomason, "I/O Facilities of the Distributed Processing Programming Executive (DPPX)," *IBM Systems Journal* **18,** No. 4, 526–546 (1979).
8. *Distributed Processing Development System (DPDS) General Information*, GC27-0505, IBM Corporation; available through IBM branch offices.
9. Independent component test is described in the companion article by G. E. Boehm, A. M. Palmiotti, and D. P. Zingaretti, "Porting DPPX from the IBM 8100 to the IBM ES/9370: Installation and Testing," *IBM Systems Journal* **29,** No. 1, 124–140 (1990, this issue).
10. Cross System Product is IBM's primary fourth-generation language application generator for data processing professionals. Cross System Product/Application Development (CSP/AD) is a licensed program that guides users in writing application programs by requesting responses to prompts at display terminals. Cross System Product/Application Execution (CSP/AE) is a licensed program that executes application programs developed with CSP/AD.
11. A profile is data that describes the significant characteristics of one or more computer resources. Examples of these resources are data sets, programs, catalogs, commands, and users.
12. C. Goodrich and M. B. Loughlin, "Porting DPPX from the IBM 8100 to the IBM ES/9370: Migration," *IBM Systems Journal* **29,** No. 1, 106–123 (1990, this issue).

**Robert Abraham** *IBM Industrial Sector Division, Atlanta, Georgia.* Mr. Abraham is a senior programmer in the Engineering Management System group at IBM's Atlanta laboratory. He joined IBM in 1971 at the Advanced Systems Development Division (ASDD) Mohansic laboratory. In 1976 he started work on the CPU management component of DPPX in Kingston, New York. He continued to work on DPPX in Kingston in various development and design activities until June 1987. These activities included work on the attached processor and multiple processor designs and the DPPX follow-on activity. Mr. Abraham received his B.A. in applied mathematics from Brown University in 1971. He is a member of the Association for Computing Machinery.

**Brian F. Goodrich** *IBM Data Systems Division, Neighborhood Road, Kingston, New York 12401.* Mr. Goodrich is a senior programmer in the Distributed Systems Programming Develop-

ment group at IBM's Kingston laboratory. He joined the Kingston Programming Center in 1968 as a junior associate programmer and worked on DOS and OS BTAM support for the IBM 3270 terminal. From 1972 to 1976 he participated in the design of VTAM. Since early 1976 he has been active in the design and development of several components of DPPX, including data management (for which he received a Division Award), Host Transaction Facility, and DASD cache. Since 1981 he has been active in the development and coordination of DPPX follow-on plans. Mr. Goodrich received his M.S. in mathematics from the State University of New York at Albany in 1964.

Reprint Order No. G321-5388.