The IBM family of APL systems

by A. D. Falkoff

The developmental history of IBM subfamilies of APL systems is traced in this paper, focusing on the inter-relationships among them and the methods of implementation used by the various groups involved. The language itself, and the way its evolution was managed, are also considered as factors influencing the development process. A chart is included that illustrates the evolution of mainframe and small machine programming products supporting APL, beginning in 1964 up to the present time.

In the 25 years since the first viable APL system was introduced outside of IBM, offerings of APL systems spanning most of the significant hardware families have been produced at a rate of more than one per year. These systems have been produced by small groups of designers and developers; at no time have there been more than about 20 people, company-wide, working on APL implementations at the same time. It is worth asking how this high productivity came about: the methods of implementation, the language itself, and the management of its evolution must have all been factors. In this paper, each of these factors is discussed as the history of the various subfamilies of APL systems is traced.

Figure 1[1] is provided to visually aid the reader in following this history. In this chart, shown later, the entries shaded in blue are systems that achieved some form of product status; the others are developmental or experimental systems, which in many cases had significant IBM internal usage. The vertical coordinate is a time line, starting with 1964 at the top. On the horizontal axis there are six columns. In general, each column is devoted to a major subfamily of APL systems, or to the work of a particular implementation group. The fourth column does not fit this description; it shows work performed by different groups on two different subfamilies of systems, but they are connected in an interesting way that is described later. The directed lines on the chart indicate significant design influences or transport of code. Of course, they do not tell the whole story, as the actual transactions were usually more complex than can be so simply diagrammed.

Mainframe systems

The earliest work on APL and its forerunners, PAT and another called IVSYS, was done in IBM's Research Division. As has been reported elsewhere,[2] PAT (for Personalized Array Translator) was an interactive interpretive system using a limited set of array operations, coded for the IBM 1620 processor. It made clear that such a system could successfully be built, and it helped to motivate the design of the APL type element for the IBM Selectric* typewriter mechanism. IVSYS (for Iverson system) was the first attempt at a mainframe system.[3] It was an interpreter written in FORTRAN to run in batch mode on the IBM 7090 series of machines, and was rendered interactive by running it under an experimental time-sharing monitor[4] (TSM) on an IBM 7093 processor.

APL\360. No sooner did the original APL group have IVSYS running in late 1965, but they were told that the TSM project, which was not under their control, would be dismantled. If they were to continue experimenting with Iverson's ideas,[5] the only recourse was to undertake the development of a time-sharing system of their own, along with an interpreter, for the recently announced IBM System/360* line of machines. This work went remarkably well, resulting in an integrated system, APL\360,[6] with excellent performance characteristics.[7] The system was operational about three months after work was started, and the three implementers who did the bulk of the programming were later to receive an industry award for their work.[8] It is worth looking at the factors that contributed to this success.

First, although this was a new system, there were some important design decisions regarding the language, as well as some coding experience, carried over from the IVSYS project. Second, the design and development group was small and enthusiastic. This attracted help, both in the form of direct contributions to the coding and thoughtful feedback from early users. Third, the group did not try to do it all themselves. Mathematical functions were borrowed from the FORTRAN IV subroutine library, and ideas from other sources were adopted if considered useful. Fourth, the systematic nature of the language lent itself to a clean internal design of the interpreter. Fifth, the system was designed to be independent of the host operating system. The handling of input and output, management of user storage, and time-sharing functions were all built into the supervisor, which was tailored to the specific needs of the language processor, thus avoiding some of the complexities of more general systems. And last, even at that early stage, APL itself was used as a design tool. The supervisor, for example, was modeled in APL, and as the interpreter code progressed, the model was run on it for validation.

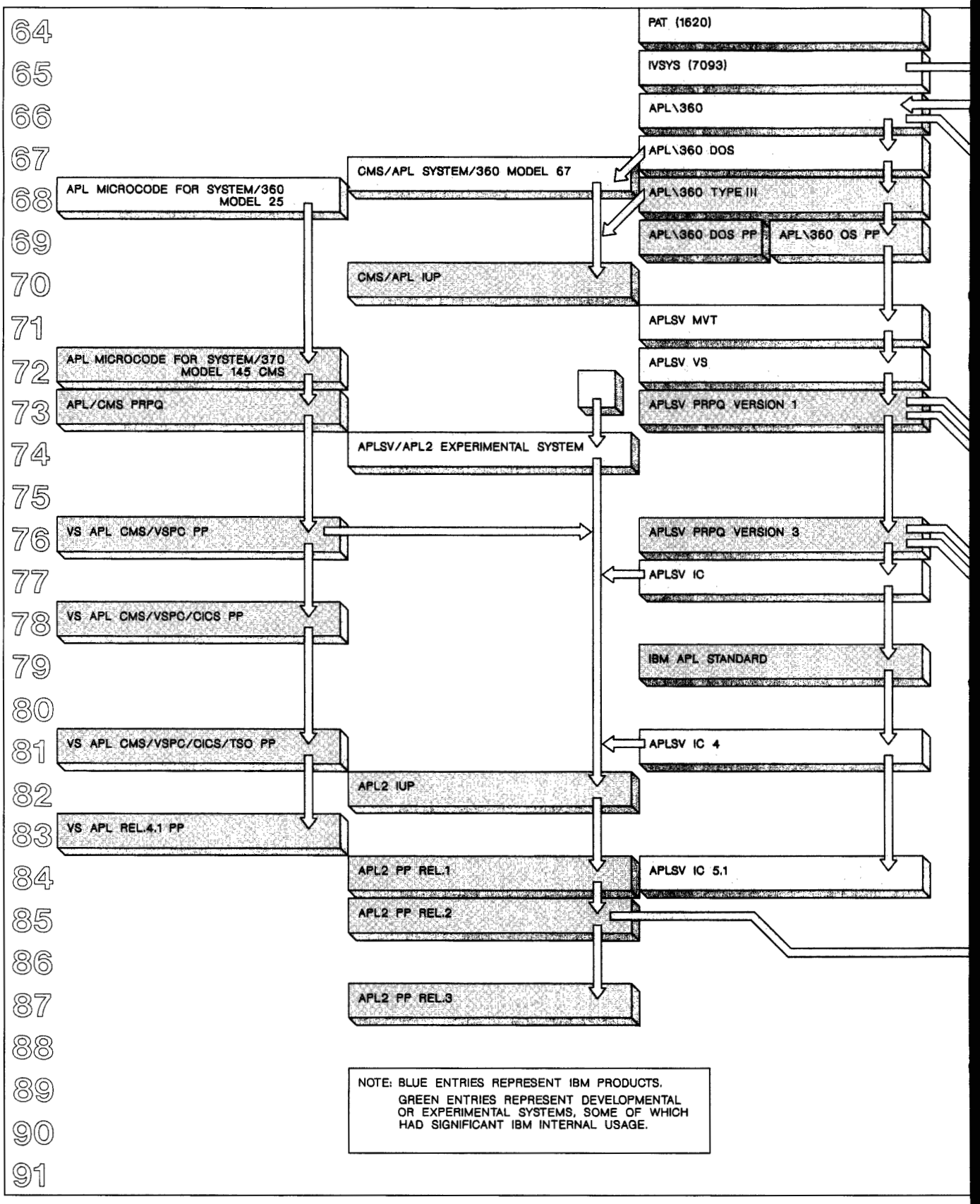
Starting in November of 1966 an APL\360 system operating on an IBM System/360 Model 50 was providing regularly scheduled service to users in the IBM Research Division in Yorktown Heights, New York. Soon thereafter copies were started up in other IBM locations, notably Endicott and Poughkeepsie, New York. The next evolutionary step was the development of systems to run under the two extant operating systems, DOS/360 and OS/360, and this was accomplished with help contributed by knowledgeable users in Poughkeepsie.

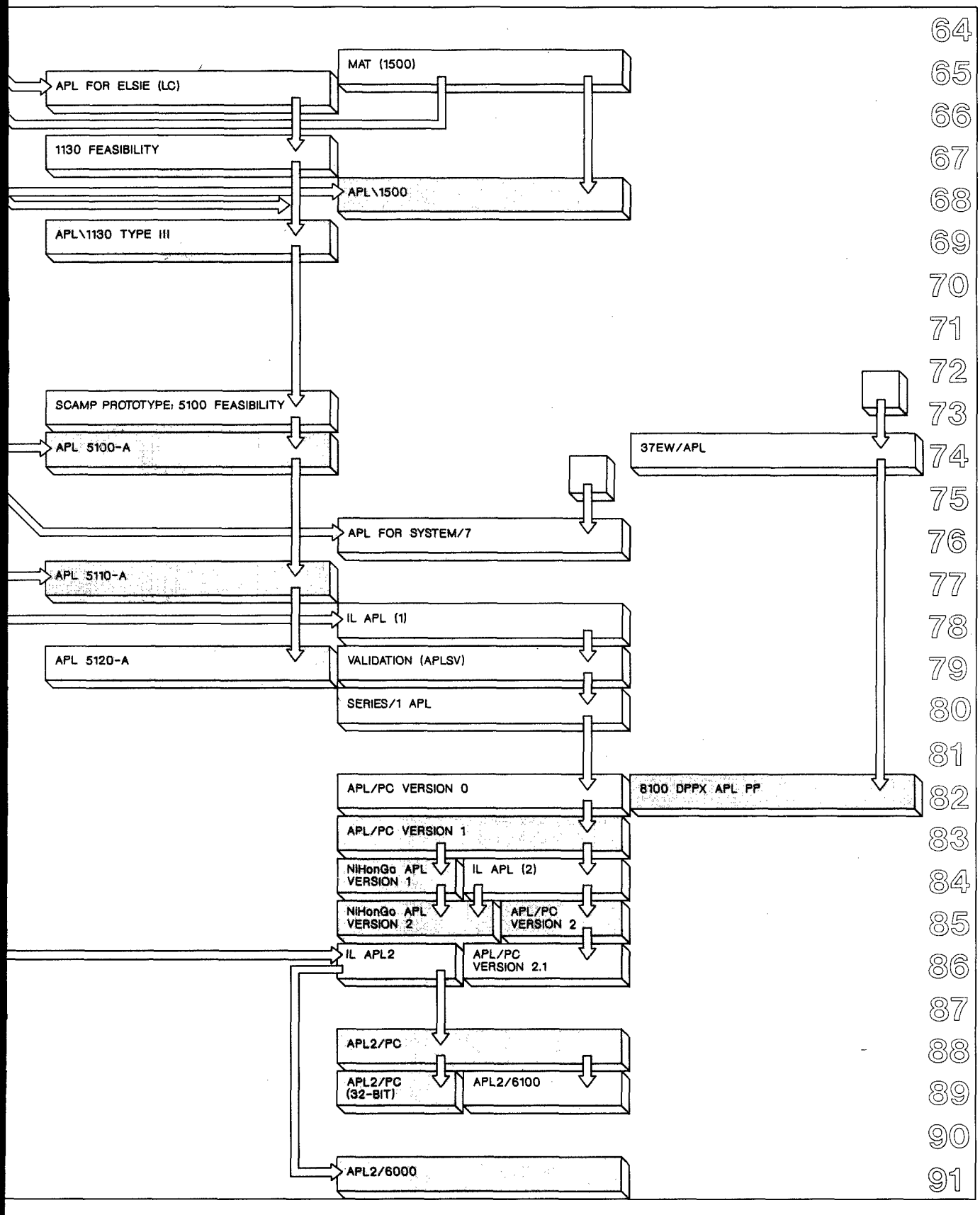
The first publicly available APL system was the cost-free "Type III" program (available without formal support) released in 1968. It was followed in 1969 by the two program products (PPs) shown in the chart. These were among the very first programs offered when IBM unbundled programs and hardware. An important decision taken then, which would influence the progress of APL in ways that even now are not completely understood, was to hold back the source code and release only object code to customers. This was done deliberately, to discourage proliferation of language variants and to give the original designers a better chance of directing the further evolution of APL along a coherent and consistent path. A positive effect of this policy was to facilitate formal standardization of APL later on, and the ad hoc standardization that resulted from having a single control point simplified the development of other APL products along the way. A possible negative effect was the discouragement of interest in APL as a subject of university research.

CMS/APL. An early variant of APL\360 was produced in IBM's Cambridge Scientific Center, where pioneering work on virtual systems was in progress. A small team there[9] adapted the APL\360 DOS code to make use of virtual storage under the Conversational Monitor System (CMS), running in the specialized hardware of the IBM System/360 Model 67. This CMS/APL system, which was made available as IBM's first installed user program (IUP), was also the first to explore two significant variations in the design of APL systems.

One such variation had to do with workspace size, which, in APL\360, was fixed at a constant value (of 32K bytes) for all workspaces in the system. By means of a relatively small modification to the internal structure of the workspace, CMS/APL enabled operation in the memory paging environment of the control program of CMS (CP/CMS) and enabled the use of variable-sized workspaces up to the capacity of the virtual storage available. An issue here, which was to be argued at length for a long time after, was the difference between swapping complete workspaces (in effect, paging logical units), and the paging of fixed segments of memory having no necessary relationship to the computational process occurring. It is probably fair to say that with the state of the art then, and for some time thereafter, swapping was more efficient, although it did require a uniform, fixed workspace size in the system. With modern hardware and programming

Figure 1 IBM APL systems




64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
MAT (1500)
APL FOR ELSIE (LC)
1130 FEASIBILITY
APL\1500
APL\1130 TYPE III
SCAMP PROTOTYPE; 5100 FEASIBILITY
APL 5100-A
37EW/APL
APL FOR SYSTEM/7
APL 5110-A
IL APL (1)
APL 5120-A
VALIDATION (APLSV)
SERIES/1 APL
APL/PC VERSION 0
8100 DPPX APL PP
APL/PC VERSION 1
NiHonGo APL VERSION 1
IL APL (2)
NiHonGo APL VERSION 2
APL/PC VERSION 2
IL APL2
APL/PC VERSION 2.1
APL2/PC
APL2/PC (32-BIT)
APL2/6100
APL2/6000

techniques, paging problems such as thrashing (inefficient paging into and out of real memory) have been reduced or eliminated. Present-day mainframe APL systems all use paging, and workspaces do not have to have a fixed size.

The other variation, which is not unrelated technically to the first, but which had greater significance for the marketing of APL, was that CMS/APL separated the APL interpreter from the rest of the APL\360 system and used it as a language processor in a different supervisory environment. APL\360 was a complete subsystem having minimal dependency on the host operating system. Its supervisor and user interface management were tailored and refined to optimize the use of APL and were never applied directly to other processors, whereas CMS, Time-Sharing Option (TSO), and the Customer Information Control System (CICS) were built to be hosts to many different processors. In its time, CMS/APL did not make a strong impression in the marketplace, but in the longer run the more general type of system that it represented turned out to have greater market acceptance, and nowadays APL products are marketed as language processors rather than as subsystems like APL\360 or APLSV (discussed below). However, with the powerful means of access to other host facilities provided by modern APL2 systems, this distinction has become less compelling.

Shared variables work well for communicating with any facility outside of the APL workspace.

APLSV. Although APL\360 was complete, in the sense that it implemented the entire APL language as it was then defined and it could be used for significant applications, it nevertheless lacked certain practical facilities. There was no way for a user to import or export information except through a typewriter terminal, and there was no means of file access. Work to rectify this situation was started in 1969 when the original APL group moved from IBM's Research Division to IBM's New York Scientific Center, and continued when the group subsequently moved to IBM's Philadelphia Scientific Center.

There was a vigorous debate within the APL group on the choice of a direction for providing the necessary communication facilities,[10] and ultimately it was decided to use *shared variables* with a formalized protocol.[11] The consensus was that this approach was the one least likely to compromise the integrity and generality of the language, as it avoided the introduction of special functions just for manipulating files. It was considered that the APL array functions already encompassed the usual file operations—for example, appending a record to a file is an instance of catenation—and elaboration of them just for files was not desirable.[12]

Under the shared-variable paradigm, access to an external file system would be provided by means of relatively simple auxiliary processors (APs) having an interface to a shared-variable processor (SVP) on one side and an interface to the host file system on the other. The APL processor would, of course, also have an interface to the SVP. Thus, any of the operating system's file operations could be specified by an appropriate character string that was generated in APL as a character vector and passed as a shared variable to the AP, which then put it into a form understood by the host file system.

This paradigm of shared variables was shown to work as well for communicating with any facility outside of the APL workspace, including the APL interpreter itself. The same facility that was introduced to provide file access thus turned out to be a rational basis for the solution of the problem of how to incorporate into the language dynamic control of primitive-function parameters such as index origin and print precision. This took the form of *system variables*, which were formally a subclass of shared variables having distinguished names, and *system functions*, which in principle implicitly utilized system variables.[13,14] The shared-variable interface to APL is itself represented by a set of such system functions and system variables.

The shared-variable facility was completely modeled in APL, including the system functions that were intended to manage it. Other enhancements to the APL interpreter were also modeled; the new primitive format function, for example, was based upon format functions written in APL that had been provided in the APL\360 product. In general this method of programming, starting with APL models,

was a multistage process. A functionally correct model was first written without regard to machine considerations, and when this was deemed to be correct, another version was produced using only APL primitives that could be easily mapped to machine code. Since both versions were executable, it was not too difficult to validate their functional equivalence, after which the second version could be used as a model for the final machine language program.

Experimental APLSV systems were produced for the then current System/360 operating systems in 1971 and 1972, as shown in Figure 1. Again, the job was accomplished in a relatively short time by a small, highly motivated team. An internal IBM announcement and a technical seminar on APLSV and shared variables was held in 1971, after which the Philadelphia Scientific Center made available on-line APLSV service to other IBM locations. This service was well received, and the high rate of usage constituted very effective testing for the product offering, which was made publicly available in 1973 in the form of a specially priced and contracted product, or programming request for price quotation (PRPQ).

The APL standard. Although questions were raised at the time, particularly in response to the seminar in 1971, regarding the wisdom of the shared-variable approach—as contrasted, for example, with building specific file and input/output facilities into the language—it does appear in retrospect that it was the proper direction. At the very least, by establishing a clear boundary between the language and the system facilities, it ultimately made it easier for the industry to agree on an APL standard. And by the same token, it has made it easier to build new APL systems, and to port APL systems between machines with dissimilar architectures.

The first official IBM standard for APL, put in place as an interim document in 1974, was the language as defined by the APLSV implementation.[15] Work on a formally written standard had already been started in the Philadelphia Scientific Center, but was still a long way from completion and adoption. Over the course of several years and many iterations, the work product and the responsibility was transferred to IBM's Santa Teresa Laboratory in California. Finally, after undergoing the formal ratification process in IBM, this formal document became the IBM APL standard in late 1977.[16] In 1979 the technical portion of this standard was published in its entirety as an appendix to a paper describing its evolution.[17] This appendix was later adopted as the first draft APL standard by a committee of the International Organization for Standards (ISO). It was not accepted as wholeheartedly by the American National Standards Institute (ANSI) committee, which insisted on rewriting the document in a different style altogether. Nonetheless, the APL language definition finally embodied in the standard adopted by all parties in 1987 is essentially that of APLSV.

Internal APLSV systems. By the time that the Philadelphia Scientific Center closed in mid-1974, IBM in general, and certain key sites in particular, had developed a strong dependency upon the APLSV service for running daily business. By this time also, the product direction had taken a turn, as discussed later, and there was not yet a fully supported APL product that could sustain the necessary maintenance and service level required. The affected sites therefore banded together to form an internal APL support group for the purpose of maintaining the APLSV program while they waited for a product to which they could satisfactorily migrate.

Some language development was included in the work of the support group, but their major activity was more in the nature of systems work—keeping up with evolving operating systems, and developing new or enhanced auxiliary processors for file management and other purposes. Notable among the latter was a processor, AP19, that enabled one active user to activate another user account under program control from inside the first user's workspace.[18] The first version of this worked only in a single machine, but a later version worked between machines not even necessarily in the same location. The primary motivation for this facility was the practical need to run long jobs in batch mode unattended, but it also made it possible to easily model and simulate general forms of cooperative and parallel processing.

APL/CMS and VS APL. While the original APL group was working on the design and development of APLSV in Philadelphia, a rather different line of inquiry was going on in IBM's Palo Alto Scientific Center in California. Here, the interest was in performance and the possibilities inherent in building a hardware APL machine. As shown in the first column of Figure 1, this work first resulted in a microcoded APL system for the System/360 Model 25. This was a single-user dedicated APL system in

which the control code that emulated the System/360 was replaced with code that emulated APL.[19]

APL\360 was used as the model of how an APL machine should appear to a user, and some pieces of code were used from existing systems, but overall the implementation was basically new. It introduced the use of arithmetic progression vectors (APV), which conserved both time and storage in many common situations, and facilitated more efficient evaluation of certain array transformations;[20] it made use of a very fast syntax analyzer that required a new internal representation of APL statements; and it used a different storage allocation method. Not all of APL was implemented at the microcode level, but this being an APL machine, the part not so implemented was necessarily written in the subset of APL that was microcoded. The supervisor program was also written in APL and executed that way without further translation.

The next step along this line of development was APL microcode for the System/370* Model 145. By this time (1972) APLSV had seen heavy use internally, and the shared variable concept had been generally accepted as the proper direction for managing system-related operations in APL systems. This technology was transferred, and other aspects of the work planned for the Model 145 were discussed, at a week-long workshop set up by the teams from Palo Alto and Philadelphia.

Also by this time, CMS as a time-sharing host was gaining in market acceptance, and a decision was taken by the Palo Alto group not to make a dedicated APL machine, as was done for the Model 25. Instead, they concentrated on an APL interpreter that would run under CMS and optionally use microcode to enhance its performance.[21] Two product offerings came directly out of this work: the interpreter with microcode assist, which could run only on the System/370 Model 145, and an independent interpreter named APL/CMS, which could run on any machine running CMS.

The microcode assist did indeed provide customers with a significantly more powerful APL processor than the Model 145 could provide without it, but its marketing was hampered by the fact that there was no similar upgrade available for the more powerful machines in the System/370 family. Although the design of the APL assist was quite general, the code itself could not be ported to other machines because they had a different underlying processor or did not use microcode at all.

While this work was going on in the Scientific Centers, plans were being made in the IBM Programming Center in Palo Alto for a new interactive time-sharing system to be called Virtual Systems Personal Computing (VSPC), and a principal language processor under that system was to be APL. Because of the marketing considerations noted previously in the discussions of CMS/APL and APL/CMS, this type of general time-sharing system, with independent language processors, was preferred over integrated systems like APL\360 or APLSV. As a consequence, when APLSV was made available as a product in 1973, it was given the more tentative status of a PRPQ, rather than full program product status, and the stand-alone interpreter developed in Palo Alto to run under CMS was chosen as the base for VS APL, the processor planned for VSPC. However, as an interim product of the type anticipated, the APL/CMS interpreter produced in the Palo Alto Scientific Center was also released then as a PRPQ.

In its original form and before it was actually put on the market, the APL/CMS interpreter had incorporated some language changes in addition to the changes in the internal design. Several of these were considered to cause problems in the language definition, and were opposed by the APL group in the Philadelphia Scientific Center where, as described earlier, work on an APL standard was already under way. The disagreement was escalated and resolved expeditiously under pressure of the need to get on with product plans. In addition to settling the issues of the moment, this resolution of the problem had the beneficial effect of accelerating the adoption of an APL standard within IBM, which, as noted earlier, has been an important factor in the continuing high productivity of APL development groups.

Eventually, the VS APL interpreter was produced by the APL product development group in the General Products Division of IBM as their first major product. They had previously (while still part of the Systems Development Division) taken over maintenance of APLSV when the Philadelphia Scientific Center closed in mid-1974. Over the course of the next several years, as shown in Figure 1, successive releases of VS APL added support for additional IBM mainframe time-sharing environments until all four—CMS, VSPC, CICS, and TSO—were included. A still extant final release was made in 1983.

An ongoing use of VS APL is the hands-on network environment (HONE) system, where APL has long been the vehicle for delivering configurators and financial analysis programs to the IBM marketing and support teams. This use posed two system problems that were not addressed by the APL product systems until the most recent release of APL2, described below. These problems arise in a situation in which large numbers of people must use identical programs but also maintain individual workspaces to hold their own data. First, if each person copies the programs into an individual workspace, and then saves it, the file storage system will be flooded with redundant material. Second, the common programs change over time as new products and new plans evolve. This information, which comes from centralized responsible sources, would somehow have to be propagated to all the copies in the individual workspaces.

The HONE solution to these problems was to develop a system facility where the individual users are given only *use access* to the common programs, which are held in a privileged storage area. The parties responsible for maintaining the programs can then upgrade as necessary the single copy held in common.

APL2. The evolution of APL2 is an interesting illustration of how a small group of people with a shared vision can maintain the continuity of their technical work and bring it to a successful conclusion, even over a time span of more than 15 years. During this time, people were transferred between three or four divisions and made several cross-country moves, all while producing other results of value to the company.

Thus, the desirability of breaking out of the constraints of rectangular arrays was recognized very early in the course of the work on APL, and some background work on the subject was steadily maintained in the Research Division while APL\360 was being developed. The group was then transferred to the Philadelphia Scientific Center, where definitive work, leading to an implementation of some form of generalized arrays, was started after the APLSV program was well along. When the center was closed in 1974, most of the APL group was transferred, as a group, to the West Coast, where they became part of the APL development organization. The work on a new APL interpreter—dubbed "APL2" at this point—was kept going there for a while, along with maintenance of APLSV, but the pressures of producing the VS APL products eventually reduced this to a crawl. However, language studies had been continued by the small contingent of the Philadelphia group that had remained on the East Coast, and the design of a new interpreter was resumed in earnest in 1978 after they and others were reassigned to the Research Division in Yorktown Heights, New York. The transfer of APL2 technology was completed later (1982), when the people directly working on the interpreter were again transferred to the APL development group in California.

The evolution of APL2 illustrates how a small group with a shared vision can be successful.

In keeping with the usual method of doing things in the APL development milieu, the initial work on APL2 did not start as a blank slate, but as a variation of the working APLSV interpreter. Actual coding started in Philadelphia in 1971, a comprehensive paper on the principal ideas was published in 1973,[22] and by 1974 an interpreter with general array operations was available for experimentation, first running under APLSV in the Philadelphia system, later running in Palo Alto, and later still in IBM Kingston, New York, as an alternative interpreter on their APLSV service system. As this evolved, new functions unrelated to general arrays were picked up from the APLSV internal releases.

The first APL2 product was an interpreter running under CMS, which was announced as being somewhat experimental and was marketed as an installed user program (IUP). In addition to the functions necessary for the accommodation of general arrays, it incorporated numerous language enhancements. These ranged from simply making the primitive mathematical functions work with complex numbers, through several new and extended primitive functions such as eigenvalues, picture format, and replication, to simple-sounding but far-reaching changes in APL operators, which were now able to accept defined functions as operands, and could themselves be user-defined.[23,24]

The APL2 IUP included an important new system function, ⎕TF, which either generated a transfer form—a system-independent representation—of an APL object, or established an object in a workspace from the transfer form. It also included two new system commands, *)OUT* and *)IN*, which generated and accepted host system files composed of collections of objects in transfer form. Although the primary motivation for these operations was to facilitate migration between different APL systems, in time these collections of APL objects in transfer form have come to be regarded as another form of saved workspace with its own useful characteristics, even where migration is not an issue.

A full-fledged APL2 program product, which emphasized system facilities for integration with other IBM programs as much as new language features, was released in 1984. The code was a further development of the IUP, with some emphasis on speeding up execution, some language changes, and a full complement of auxiliary processors. Many of these were inherited from VS APL, with or without enhancements. This use of existing code was facilitated by resolving some differences between APLSV and VS APL in the internal design of the shared variable processor to ensure portability of existing auxiliary processors. Notable among these were a full-screen session manager and a processor for access to database products such as DATABASE 2* (DB2*) and System Query Language/Data System (SQL/DS*). Communication with APL2 from the Interactive System Productivity Facility (ISPF) products was provided by an auxiliary processor distributed with ISPF. Other system facilities included national language support for system commands and messages, a new internal character type of four bytes per character for supporting large character sets such as Kanji, and various utilities to facilitate migration from older APL systems.

Carried over from the APL IUP was the use of *primitive defined functions*—functions written in APL rather than machine language that are nonetheless part of the language processor and are invoked by the use of primitive function symbols or system commands. First used to facilitate experimentation with language changes, primitive defined functions have been retained in the later releases of APL2, where they are used for a variety of system operations and primitive functions, or portions of primitive functions, for which high performance is not a requirement. There is also a complementary facility in APL2 that uses ordinary user-type names to invoke machine coded functions. This is a device that goes back to the first version of APL\360, where it was used to provide useful functions, variously called keyword functions or workspace functions, for which special-character names were not available. In the case of APL2 it was used for the eigenvalue and polynomial functions that were included as primitives in the IUP but were felt to be somewhat premature for inclusion as such in the program product.

The second release of APL2, which followed the first by little more than a year, continued the trend toward closer integration of APL with its environment. There were improvements in the support for database products and graphic display devices, and direct access was provided to system editors outside of APL. Of possibly greater significance, however, was the introduction of a new facility known as *name association*, where routines written in FORTRAN, assembler, or Restructured Extended Executor (REXX) could be called from APL applications.[25] This facility works by providing dynamic linking between the active workspace and other namespaces, allowing different parts of a process to be sequentially executed by different processors, as may be appropriate. Although inspired in part by a shared variable auxiliary processor developed many years earlier at the IBM Heidelberg Scientific Center in Germany,[26] it differs from the use of the shared variable facility in that the parts of the process are never executed in parallel or asynchronously, the associated names may refer to external objects of any kind (not just variables), and the name association is preserved across working sessions.

The third release of APL2, in late 1987, included two major extensions to APL2 system capabilities. One was the automatic utilization of hardware vector processing when available, an obvious exploitation of the natural array properties of APL. The other was the inclusion of an encapsulation mechanism for APL workspaces, which transformed them into load modules, known as *packages*, which could then be accessed by a name association processor. Among other applications, packages have the potential to solve the problems addressed by *use access* on the HONE APL system previously mentioned. The existing primitive defined function facility, which already depended upon isolation of namespaces for its operation, was used as an integral part of the implementation of the package facility. The associated processor was also extended

to support FORTRAN function calls in addition to calls to subroutines; and a complementary facility was provided to allow routines written in other languages to request execution of APL expressions.

In recognition of the greater availability of personal computers and workstations with versatile displays, and their use as terminals and for running native APL systems, this release of APL2 allowed the use of lowercase alphabetics as an alternative to underscored alphabetics, and provided a system command for setting the mode.

In earlier times of APL design and development there was a strong effort made to reach consensus on new ideas, and an equally strong emphasis on the importance of testing by users. As the development center shifted about and the development process itself became more formalized this was not lost sight of, although some aspects of it have been hard to maintain. Since about 1982, however, with the popularization of electronic conferencing, the IBM internal computer network has been used quite effectively to gather together user experience with developmental systems, and publicize opinions on new ideas. User testing of new systems has been formalized at the same time, with selected sites within IBM undertaking responsibilities as virtual extensions of the regular development test group.

Small machines

The first implementation of an APL-like system on a small machine was the PAT system on the IBM Model 1620, done in 1964. APL has had a presence of small machines ever since. In fact, as is detailed below, the first portable desktop personal computer marketed by IBM was designed as an APL machine.

APL\1130. In 1965–1966 the IBM Los Gatos Laboratory in California was working on the design of a very small, low-cost (hence LC or "Elsie") machine. It was to have a relatively simple instruction set and an internal memory of only 1024 words, supplemented by an external magnetic disk, about eight inches in diameter, which used grooves on one side for mechanically indexing to the magnetic tracks. Science Research Associates, then a subsidiary of IBM, was interested in the educational potential of such a machine, and commissioned a study to produce an APL system for it. Two of the three people who conducted the study had previously worked on IVSYS.[27] Drawing on this experience, the group proposed a modified architecture for Elsie, better suited to implementing APL. An emulator for this machine design, and an assembler for programming it, were written for the IBM Model 7090, and design of the APL system proceeded from there. The result was then successfully transferred to a real Elsie prototype, so that in due course an APL system was running in Los Gatos.

Unfortunately, business considerations kept Elsie from ever becoming a product, but the work on it was not wasted. By 1967 APL\360 was becoming widely known within IBM, and the Research APL group was approached by an IBM branch office interested in the possibility of having an APL system available for the IBM Model 1130, a midsize "scientific" machine. To quickly produce a prototype and show feasibility, an Elsie emulator was written for the Model 1130 and the APL system was installed on it. It ran successfully. To improve performance, one additional instruction was added to the Elsie emulator, an escape to the native 1130 architecture, which was used as the path to more efficient coding of successive parts of the interpreter. As shown in Figure 1, an upgraded APL\1130 was later produced as an IBM Type III program.

Not shown in the figure is a more formal APL\1130 product that had a very short life. It was a time-sharing upgrade of the Type III program, produced by the APL development group in Palo Alto, which was then still part of the Systems Development Division. It was shipped to one or two customers before being withdrawn from the market. But it, too, was not wasted. Indeed, it figured importantly in the early development of the modern personal computer.

APL 5100. In late 1972 the Palo Alto Scientific Center was asked by IBM's General Systems Division headquarters in Atlanta, Georgia, to suggest an APL product suitable for production by their division. In response, the Scientific Center proposed an entry-level machine that could fit on a desk. This suggestion was accepted, and they proceeded to assemble a team composed of people with hardware knowledge from Los Gatos and people with software knowledge from the Scientific Center to work on the design. The team selected a processor engine known internally as "Palm" for the machine's central processing unit, in preference to another, called UC.5, that was also available at the time.

Once again, the quickest way to show feasibility and produce a prototype was to emulate an existing ma-

chine that already had APL programmed for it. In this case, the Model 1130 was chosen. Thus, APL\1130, a system that had its origins in Elsie, the earlier Los Gatos machine, and that had been ported by emulation to the Model 1130, where it was eventually converted to native 1130 architecture code, was now ported to a new machine in which Los Gatos was also involved in the hardware design. The functioning prototype, know as SCAMP (Special Computer APL Machine Portable), was produced in the short time of six months, and was successful in persuading the General Systems Division to proceed with a production machine.[28]

At present the SCAMP prototype, an APL machine that was the unique forerunner of the first production personal computer, resides in the collection of the Smithsonian Institution in Washington, D.C.[29]

The production machine was designed at IBM's General Systems Division laboratory at Rochester, Minnesota, and was made available as a product, the IBM 5100 machine, in 1974—less than a year and a half from the start. This remarkably short development cycle for such a complex new product can be attributed in large part to the fact that emulation was used again, even in the final product. This time, however, although the same Palm internal engine was used, System/360 architecture was emulated rather than 1130 architecture, so that the up-to-date APLSV product system could be used as the APL facility with virtually no modification. There were some changes, however, that anticipated later developments in personal computers. For example, the primary input/output device was a cathode ray tube with an associated keyboard that included an extra shift, named "CMD," and a number pad; there was a software switch to enter a communication mode to enable the machine to act as a terminal on a host system; and another switch to automatically copy input and output to an attached printer.

The later models, the IBM 5110 and 5120, which had a different internal processing engine and also used a later version of APLSV, carried these forward-looking changes considerably further. Where the IBM 5100 had only a tape cartridge for nonvolatile storage of files and workspaces, the later machines included an eight-inch diskette facility, separately available in the IBM 5110 and integral in the IBM 5120. Whereas the CMD key in the IBM 5100 was used very modestly to generate APL system commands from six keys in the top row, the IBM 5110/5120 CMD key was also used to produce the APL overstrike characters, as well as the distinguished names of system variables and system functions, with a single shifted keystroke. The CMD key was also used to switch the entire keyboard from an APL character mode to a standard lowercase and uppercase character mode in which the single APL characters were still available as a third shift. All the models had a shared variable facility for communicating with the tape drive and the printer, and in the later models this was extended to include the diskette drives, the display screen, and the serial input/output port.

The SCAMP prototype, an APL machine, resides in the collection of the Smithsonian Institution.

There is considerable family resemblance between these early APL machines and the personal computer (PC) line of machines IBM produced a few years later. The IBM Portable Personal Computer, in particular, with its built-in small screen looks a lot like the IBM 5110, and its part number of 5155 is clearly in the sequence of the earlier machines. (The early PC itself is model number 5150, and the PC/XT* and PC/AT* have model numbers 5160 and 5170.) This is not really surprising, since the IBM Rochester development group that produced the 5100 and 5120 machines was later transferred to the IBM laboratory at Boca Raton, Florida, where they constituted the beginning of the Entry Systems Division of IBM, which developed the IBM PC.

APL\1500. Returning for a moment to the 1960s, a variant of the IBM 1130 machine was the IBM 1500, a system intended for the educational market. This system used a faster version of the 1130 processor, known as the 1800. The IBM 1500 was an early example of a multimedia machine, featuring a cathode ray tube display and a film projection unit in addition to the usual typewriter input and output. In 1965 the Service Bureau Corporation wrote a program called MAT/1500 for the IBM 1500, whose primary software was a computer-aided instruction program called "Coursewriter." MAT/1500 was in-

tended to augment this mostly verbal system with a mathematical capability, including elementary functions and some array operations.

Some three years later, Science Research Associates undertook to write a full APL system for the IBM 1500. They modeled their system after APL\360, which had by that time been developed and seen substantial use inside of IBM, using code borrowed from MAT/1500 where possible. It is interesting to note that in their documentation they acknowledge their gratitude to "a number of high school students for their compulsion to bomb the system."[30] This was an early example of a kind of sportive, but very effective, debugging that was often repeated in the evolution of APL systems.

DPPX APL. At about the same time that the Palo Alto Scientific Center was working on SCAMP, another APL system design was under way at IBM in Poughkeepsie, New York, using the UC.5 engine that had been considered as an alternative to Palm when Palo Alto selected its processor engine. When nearly completed, the project was moved to Kingston and the target machine became the IBM 8100, which had the UC1 as its internal engine, an upgrade of the UC.5. This was to have been a complete APL system, including its own supervisor, but work on it was halted before it reached product level. The project was subsequently moved again, this time to the Lidingo laboratory of IBM Sweden. The technology transfer was effected in part by the temporary assignment of one, and then another, of the original developers. It was brought to product status running under the Distributed Processing Program Executive (DPPX) operating system of the IBM 8100, rather than its own supervisor.

DPPX APL was a multiuser time-sharing system that made innovative use of the shared variable processor in its internal operations. (Work on its design also led to suggestions for broadening the functionality of shared variables, which, though not implemented at the time, are still worth considering.[31]) Motivated by an absolute limit of 64K bytes for the workspace size, the designers consigned as much function as possible to the shared variable processor, so as to free up space in the workspace that would otherwise be taken up with the interfaces to other parts of the system. Thus, for example, communication to the keyboard and display input and output was mediated by the same shared variable processor as was available at the user level. Also, to facilitate the use of shared variables between workspaces—a means of overcoming the workspace size limitation as well as a way of functionally segmenting programs—the system provided support functions to start and control secondary sessions from inside an active workspace, much in the manner of the AP19 processor on the internal APLSV systems described earlier.

The system emphasized the utilization of DPPX facilities from inside APL programs. Sets of support functions, which had the same appearance as the workspace functions mentioned previously in the discussion of APL2, were provided, for example, to facilitate the use of the DPPX Presentation Services (PS). Alternatively, these operations, and others, could be effected by means of explicit shared variables using an auxiliary processor connecting directly to DPPX input/output and command programs. This gave the APL programmer willing to work at that level access to the operating system commands and macros.

Another innovation, at the APL language level (which was otherwise essentially that of VS APL), was the introduction of a system variable, □CMD, to which a character string depicting an APL system command could be assigned. Thus, it was possible to imbed in a running program an order to save the workspace at that point, while the program continued to run. Though sometimes controversial, this feature of dynamic execution of system commands was well thought out, as were the other innovations in DPPX APL. It is unfortunate that the system did not see enough real use for a body of opinion to build upon the value of these innovations. Still, there is little doubt that with its emphasis on communication and integration with the environment, DPPX APL was a step in the right direction, as evidenced by subsequent developments in the two major current APL systems, APL2 and the derivatives of IL APL, discussed next.

IL APL. In 1974 the Computer Science Department of the IBM Madrid Scientific Center started an APL system for the IBM System/7, a small sensor-based machine intended for use in applications such as process control and laboratory automation. The APL system was modeled after APLSV in the expectation that the use of shared variables would simplify both the design and the subsequent operation of the sensor input/output, but the APLSV code itself was not used. In order to accommodate an APL time-sharing system to a machine that had as little as 16K of two-byte words in its main memory, the

interpreter was modularized so that its parts could be swapped into memory much the same way as the workspaces. The system was coded in assembly language.[32]

System/7 APL was never made into an IBM product, but it saw some use in several research laboratories both inside and outside of IBM, and was used by the Madrid Scientific Center itself to control the environment in an experimental agricultural project. Its major significance, perhaps, was that it was the first implementation of APL by a team that went on to develop a portable APL system that has been the basis for the IBM implementations of APL on personal computers and workstations.

In 1976 the Scientific Center was asked to write an APL system for the IBM Series/1*, the successor to the IBM System/7. Reluctant to simply repeat the same work in another low-level language, the team conceived the idea of writing a portable APL system in a systems programming language intermediate between assembler and a high-level language such as APL. The language they designed, known as IL (for Intermediate Language), has a simple syntax, somewhat resembling APL, and a semantics closely related to that of assembly languages, but tailored to the requirements of an APL system.[33] An APL system written in this language can be ported to different machines by writing compilers from IL to each. Since each compilation is essentially a one-time affair, the execution speed of the compilers is not an issue, but the time to produce one is, and therefore they have been written in APL.[34]

The IL approach was first tested by writing an interpreter only, and compiling it to System/370, where it could be compared to APLSV and debugged. Once this was successful, the IL implementation was expanded to include an APL system command handler, an input editor and scanner, and a shared variable processor.[35] Nearly all of the coding for IL APL was new, taking only a few algorithms from APLSV and VS APL. Others were based on publications, some of which were also the source for APLSV and other mainframe APL systems.[36,37]

Series/1 APL. After the validation of IL APL on the IBM System/370, the first download porting was to the Series/1. It was still necessary to code machine-dependent parts of the system, such as the APL time-sharing supervisor and library management operations, by other means. The IL interpreter was also modified for the Series/1. The architecture of this machine placed severe limitations on the size of the APL workspace, and to mitigate this problem the IL APL designers developed the idea of a two-part workspace: a *main workspace* of the maximum size, where APL objects were created and modified, and an *elastic workspace*, which used a secondary memory to swap out APL objects not currently referenced, when more execution space was needed.

A choice had to be made between two operating systems on Series/1: Realtime Programming System (RPS), which was the official IBM offering, and Event-Driven Executive (EDX), which was then being developed informally by interested groups in the company. The Madrid Scientific Center did not have resources to do both machine-dependent subsystems. RPS was selected, on the basis that it was the mainline offering, while internal interest in an APL system on EDX was probably strong enough to generate its own separate support. In fact, this proved to be the case, and a support group for an EDX version was formed under the aegis of the APL Design Group in Research. A viable EDX system was produced,[38] which was used in about 40 internal IBM sites. Neither version was ever offered as an IBM product.

APL/PC. The second download porting of IL APL was to the IBM Personal Computer (PC), in 1982.[39] One requirement placed on the design was that it should be usable in a PC with only 128K of random access memory, a configuration that was considered generous at that point in the evolution of the personal computer market. But even with larger memories, in order to achieve acceptable performance it was necessary that the workspace size stay within the 64K primary addressing capability of the machine. To reduce the severity of this limitation, the elastic workspace concept was carried forward from the Series/1 design.

The language level of APL/PC was essentially that of the APLSV internal system, which included picture format, ambivalent defined functions, and the execute alternative system function. All of these were also in the APL2 IUP, which became available at about the same time as the zero-level of APL/PC, but not in VS APL, the principal mainframe product at the time. APL/PC also included ⎕TF and *)IN* and *)OUT*, as found in the APL2 IUP. In addition to facilitating communication and migration between different APL systems, especially between mainframes and PCs, the use of the transfer form also

served to overcome the absence of the APL copy command in APL/PC.

An important aspect of the design of APL for the PC was the deliberate effort made to bring as much of the underlying machine as possible under control of

APL2 supports 32-bit addressing for the PS/2 and runs on the AIX platform for the IBM RISC System/6000.

the APL programmer. This took two forms. First, a new system function, □PK, was introduced to allow access to any part of the machine memory for both reading and writing, and to execute machine-code subroutines. Second, auxiliary processors were provided to interface with the Basic Input/Output System (BIOS) and Disk Operating System (DOS) interrupts, with the DOS file system, and with peripheral devices, including the display.

The development versions of APL/PC were tested by the APL Design Group in Research, using scripts and programs first constructed in connection with work on APL2. A preproduct-level program was then made available for testing by interested parties in many different parts of the company, before the first product offering was released in 1983. This was the beginning of an iterative process—upgrading or changing the IL APL, subjecting the resulting PC program to widespread internal use and testing, and product release—a process that is still going on, through several versions of APL/PC, APL2/PC, and APL2 for workstations.

The next use of IL APL was the porting to the IBM 5550, the personal computer available in Japan, done in collaboration with the IBM Tokyo Scientific Center. This resulted in a product known as NiHonGo APL. For this version the internal data types were expanded to include two-byte characters, and the keyboard and display operations were elaborated, so as to accommodate the much larger Kanji character set. Otherwise, NiHonGo APL and APL/PC were the same.

In the period from 1984 to 1986, a second IL APL interpreter was developed and also ported to the IBM 5550 machine. The main changes affected memory management, and many of the implementation limits of the first version were markedly increased. There were also some performance improvements, and a substantial increase in the number and scope of the auxiliary processors. Most significant among the latter was AP2, an interface to non-APL programs, which made it possible to dynamically load and run DOS programs or programs written in FORTRAN or assembly language. This processor was under development at about the same time as the name association facility in APL2, and represents an alternative approach to solving the same problems.

There was one more refinement of APL/PC, a version intended for internal use only, which included support for IBM Personal System/2* (PS/2*) Model 80, and a workspace packaging program. Although the same term is used, the resulting APL/PC package is quite different from that of the mainframe APL2. In this case, a separate program, running directly in DOS, uses the name of an APL workspace and the list of auxiliary processors it uses, and produces a DOS (.EXE) program that contains the workspace and the necessary parts of the APL system and can therefore run independently.

APL2/PC. Over the period from 1986 to 1990, an IL implementation of APL2 was produced, and successively enhanced, by the Madrid Scientific Center in collaboration with the IBM United Kingdom Scientific Centre in Winchester.[40] There have been two releases of this system and several field upgrades. The first release, in 1988, was a 16-bit version that can run on any of the IBM PC or PS/2 machines, and requires only 256K of real memory. It retains most of the implementation limits of APL/PC version 2, which derive from the 16-bit addressing structure of the underlying structure, but the workspace size can be as large as 440K bytes. Except for complex number arithmetic and some minor language refinements, it is a full-function APL2 system with a comprehensive set of auxiliary processors, a full screen manager modeled after the mainframe APL2 version, and direct invocation of DOS operations by means of a *)HOST* system command.

The 32-bit version, released in 1989, was generated, downloaded, tested, and debugged in 13 man-weeks, an impressive confirmation of the effectiveness of the IL approach. In this version there is no

practical limit on workspace size, which can be as large as 15 megabytes, for example, on a 16-megabyte PS/2, and there are no separate limits on the size of APL variables. It has all of the language and system features of the 16-bit version, and both may be used to produce running packages of APL applications, as described previously.

APL2/6000. The most objective test of the IL APL approach was the most recent one, the porting to an Advanced Interactive Executive* (AIX*) platform on the IBM RISC System/6000*. In this case, one person with no prior knowledge of either IL APL or the RISC System/6000—working alone except for a few days of help at the end—was able to produce the necessary back end of the IL compiler, which translates the IL code to the language of the object machine, and bring up a viable APL workspace on the machine in less than 10 weeks. With a second person writing the machine-dependent parts of the program in C, the system was brought to the point of being publicly demonstrated less than six months from the start. An internal IBM release was reached in 10 months and a product announcement was made two months after that.

Other APL processors

All of the APL machine implementations described so far (and shown in Figure 1) are interpreters, as befits the language processor in a highly interactive system. However, there has been a steady pressure in the marketplace to improve the performance of production applications in APL. As a result, in addition to the microcode assist described above, acceleration techniques ranging from adaptive interpretation, to translation to intermediate languages, to direct compilation to machine language have been worked on and used experimentally.

An adaptive interpreter for APL was designed in the IBM Israel Scientific Center in the mid-'70s. The program analysis was implemented in APL, and it compiled code to an intermediate language conceived of as a virtual APL machine.[41] The implementation was completed far enough to estimate its performance, which was promising as far as it went, but no production use was made of it. However, the techniques were further evaluated in the APL Design Group in the Research Division when one of the investigators took an assignment there, and they provided background for the APL compiler work that followed.

This compiler work branched into two principal directions, both of which used APL itself as the principal programming tool. One direction emphasized the exploitation of APL array operations to directly generate very fast machine code and take advantage of the potential for automatic parallelization of APL programs at the basic block level.[42] At first relatively narrow in the range of APL expressions it could compile, this program has been improved and enhanced to the point where other internal IBM sites are experimenting with it for production applications while the investigation continues in the Research Division. Consideration is currently being given to translating into another high-level language, rather than directly into machine code.

The other branch of the Research Division work in APL compilers started out with the intent to translate into a high-level language, namely FORTRAN, in order to take advantage of the optimizing compilers already extant for that language and the portability implied by the widespread availability of FORTRAN compilers.[43] The general scheme of this compiler is to work within the APL2 system, compiling those functions in an application that are most resource consuming, and invoking the compiled functions at run time by means of the name association facility in APL2. An important objective of the work on this compiler was to translate all of APL, up to its chosen language level, without compromising on the nuances of end conditions or other detailed aspects of the language definition. The work was transferred to the numerically intensive computing (NIC) group at IBM in Kingston, New York, around 1987, where it underwent enhancement of its user interface and was migrated from CMS to Multiple Virtual Storage. Finally, under the aegis of that group, the program, now known as AOC (APL2 Optimizing Compiler), was turned over to an IBM Business Partner for marketing and further enhancement. It was announced as a product in early 1991.[44]

Another instance of translating APL to a high-level language is the work done in the IBM Federal Sector Division using Ada as the target language. The translator was written in APL2, and had the limited goal of allowing an algorithm designer to prototype rapidly in APL and, after debugging there, translate the program to Ada for compilation and running in that environment.[45] The APL acceptable to this translator had to be highly stylized, but it nevertheless turned out to be useful in an important prototyping application.

Concluding remarks

It is perhaps fitting to make note of some of the things not discussed in this paper. Foremost among these is all the work on APL implementation done outside of IBM. The actual number of implementations of APL is in the dozens, most of which have, or have had, an economic life. Virtually every major manufacturer of computers has had its own implementations, starting very early in the history of the language, and many of these, like the systems produced or modified by APL time-sharing vendors, have contributed to the evolution of the language itself.

As noted in the text, APL has figured prominently in the evolution of small machines. Its very interactive nature, combined with the simplicity and power of its array operations, has been a magnet for designers of small machines. Thus, even before the IBM 5100 was developed, a small Canadian company, Micro Computer Machines, had built several APL machines small enough to fit in an attache case. At the present time, there are implementations for all the major families of small computers, as well as for several workstations and lesser-known small and intermediate machines.

Another large area untouched by this paper is that of applications written in APL, except for one. That one, of course, is the design and implementation of APL systems. As the APL compilers come into their own, this field of application may well broaden significantly.

Finally, it should be mentioned that there has been an unbroken series of international APL conferences since 1969, and numerous implementers' workshops and standards committee meetings, at which language, implementation, and standardization issues have been refined to the benefit of all concerned. Thus, IBM's family of APL systems has evolved in an active and stimulating environment that continues to attract the kind of highly talented people who made it happen in the first place.

Acknowledgments

While I have tried to achieve a dispassionate and even-handed tone in describing the developments in APL products, the actual events often took place in a far more emotional, and sometimes adversarial, atmosphere where points of view were advanced with fervor and fiercely defended. In the course of writing this paper I consulted with many of the people involved in these events, most of whom are mentioned in the references. Without exception, the responses were not only helpful, but warm with the remembrance of past associations. I hesitate to list their names, for fear I may inadvertently leave some out, but I want to thank them all for their present help, and for their earlier contributions to the evolution of APL systems. I also want to express my gratitude to John C. McPherson, whose name does not appear elsewhere herein, but whose influence was pervasive. Now a retired IBM vice president, John recognized the value of APL very early, and shared his technical insights and gave support and encouragement to everyone involved in APL development throughout the course of the work.

Cited references and notes

1. Figure 1 is an extension and elaboration of one produced by R. H. Lathwell in 1982.
2. H. Hellerman, "Experimental Personalized Array Translator System," *Communications of the ACM* **7**, 433 (July 1964).
3. A. D. Falkoff and K. E. Iverson, "The Evolution of APL," in *History of Programming Languages*, H. L. Wexelblat, Editor, Academic Press, New York (1981), p. 666.
4. H. A. Kinslow, "The Time-Sharing Monitor System," *Proceedings AFIPS 1964*, FJCC **26**, Spartan Books, Washington DC (1964), pp. 443–454.
5. K. E. Iverson, *A Programming Language*, John Wiley & Sons, Inc., New York (1962).
6. A. D. Falkoff and K. E. Iverson, "The APL\360 Terminal System" in *Interactive Systems for Experimental Applied Mathematics*, Academic Press, New York (1968).
7. L. M. Breed and R. H. Lathwell, "The Implementation of APL\360," in *Interactive Systems for Experimental Applied Mathematics*, Academic Press, New York (1968).
8. The Grace Murray Hopper Award of the ACM, presented to L. M. Breed, R. H. Lathwell, and R. E. Moore in 1973. L. J. Woodrum of the IBM Poughkeepsie Laboratory contributed code for sorting and other operations, and provided continuing assistance in the development of APL\360.
9. W. Barrett and M. F. C. Crick.
10. A. D. Falkoff, "A Survey of Experimental APL File and I/O Systems in IBM," *Colloque APL*, Institut de Recherche d'Informatique, Rocquencourt, France (1972).
11. R. H. Lathwell, "System Formulation and APL Shared Variables," *IBM Journal of Research and Development* **17**, No. 4, 353–359 (1973).
12. The technology and the concept have now come together, 20 years later. As this paper was going to press, Release 1 of APL2 version 2 was announced, one of its principal new features being the ability to directly apply primitive APL functions to host system files.
13. A. D. Falkoff and K. E. Iverson, *Communication in APL Systems*, Technical Report 320-3022, IBM Philadelphia Scientific Center, PA (1973).
14. A. D. Falkoff and K. E. Iverson, "The Design of APL," *IBM Journal of Research and Development* **17**, No. 4, 324–334 (1973).

15. IBM Corporate Bulletin C-B 3-9045-001 (October, 1974).
16. IBM Corporate Standard C-S 3-9045-001 (December, 1977).
17. A. D. Falkoff and D. L. Orth, "Development of an APL Standard," *APL79 Conference Proceedings, APL Quote Quad* **9**, No. 4, Part 2, 409–453, ACM, New York (1979).
18. B. J. Hartigan, "AP19—A Shared Variable Terminal Interface for APL Systems," *APL81 Conference Proceedings, APL Quote Quad* **12**, No. 1, 137–141, ACM, New York (1981).
19. A. Hassitt, J. W. Lageschulte, and L. E. Lyon, "Implementation of a High Level Language Machine," *Communications of the ACM* **16**, No. 4, 199–212 (1973).
20. A. Hassitt and L. E. Lyon, "Efficient Evaluation of Array Subscripts of Arrays," *IBM Journal of Research and Development* **16**, No. 1, 45–47 (1972).
21. A. Hassitt and L. E. Lyon, "An APL Emulator on System/370," *IBM Systems Journal* **15**, No. 4, 358–378 (1976).
22. Z. Ghandour and J. Mezei, "General Arrays, Operators and Functions," *IBM Journal of Research and Development* **17**, No. 4, 335–352 (1973).
23. D. A. Rabenhorst, "APL2 Language Manual," SB21-3015, IBM Corporation (1982); available through IBM branch offices.
24. J. A. Brown, *The Principles of APL2*, Technical Report 03.247, IBM Santa Teresa Laboratory, CA (1984).
25. J. A. Brown, J. Gerth, and M. Wheatley, *Communication Method Between an Interactive Language Processor and External Processes*, U.S. Patent No. 4,736,321 (1988).
26. H. Eberle and H. Schmutz, *Calling PL/I or FORTRAN Subroutines Dynamically from VS APL*, Technical Report 77.11.007, IBM Heidelberg Scientific Center, Germany (1977).
27. L. M. Breed and P. S. Abrams; the third person was W. S. Worley, Jr.
28. P. J. Friedl, "SCAMP: The Missing Link in the PC's Past?," *PC* **2**, No. 6, 190–197 (November 1983).
29. J. Littman, "The First Portable Computer," *PC World* **1**, No. 10, 294–300 (October 1983).
30. S. E. Krueger and T. D. McMurchie, *A Programming Language\1500*, Science Research Associates, Chicago, IL (1968).
31. K. Soop and R. A. Davis II, "Extended Shared-Variable Sessions," *APL85 Conference Proceedings, APL Quote Quad* **15**, No. 4, 148–150, ACM, New York (1985).
32. M. Alfonseca, M. L. Tavera, and R. Casajuana, "An APL Interpreter and System for a Small Computer," *IBM Systems Journal* **16**, No. 1, 18–40 (1977).
33. M. L. Tavera and M. Alfonseca, *IL. An Intermediate Systems Programming Language*, Technical Report 01-78, IBM Madrid Scientific Center, Spain (1978).
34. M. Alfonseca and M. L. Tavera, "A Machine-Independent APL Interpreter," *IBM Journal of Research and Development* **22**, No. 4, 413–421 (1978).
35. M. L. Tavera and M. Alfonseca, *The IAPL Machine-Independent APL Processor*, Technical Report 03-80, IBM Madrid Scientific Center, Spain (1980).
36. R. H. Lathwell and J. E. Mezei, *A Formal Description of APL*, Technical Report 320-3008, IBM Philadelphia Scientific Center, PA (1971).
37. A. D. Falkoff, "A Pictorial Format Function for Patterning Decorated Numeric Arrays," *APL81 Conference Proceedings, APL Quote Quad* **12**, No. 1, 101–106, ACM, New York (1981).
38. P. A. McCharen, *The Series 1 APL-EDX System Installation and User's Guide*, Technical Report 19.0552, IBM Burlington, VT (1981).
39. M. L. Tavera, M. Alfonseca, and J. Rojas, "An APL System for the IBM Personal Computer," *IBM Systems Journal* **24**, No. 1, 61–70 (1985).
40. M. Alfonseca and D. A. Selby, "APL2 and PS/2: The Language, the Systems, the Peripherals," *APL89 Conference Proceedings, APL Quote Quad* **19**, No. 4, 1–5, ACM, New York (1989).
41. H. J. Saal and Z. Weiss, "A Software High Performance APL Interpreter," *APL79 Conference Proceedings, APL Quote Quad* **8**, No. 4, 74–81, ACM, New York (1979).
42. W.-M. Ching, "Automatic Parallelization of APL Programs," *APL90 Conference Proceedings, APL Quote Quad* **20**, No. 4, 76–80, ACM, New York (1990).
43. G. C. Driscoll, Jr. and D. L. Orth, "Compiling APL: The Yorktown APL Translator," *IBM Journal of Research and Development* **30**, No. 6, 583–593 (1986).
44. "1991: The Year of the APL2 Optimizing Compiler," *Interlink, January 1991*, Interprocess Systems, Inc., Atlanta, GA (1991).
45. J. G. Rudd and E. M. Klementis, "APL-to-Ada Translator," *APL87 Conference Proceedings, APL Quote Quad* **17**, No. 4, 269–283, ACM, New York (1987).

Accepted for publication June 27, 1991.

Adin D. Falkoff *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.*
Mr. Falkoff is currently a research staff member in the Computer Science Department at the Thomas J. Watson Research Center. He joined IBM in 1955, and since 1960 has worked on various aspects of computer science, including APL. He was a member of the visiting faculty at the IBM Systems Research Institute for several years, and a visiting lecturer in computer science at Yale University. From 1970 to 1974, Mr. Falkoff established and managed the IBM Philadelphia Scientific Center, and from 1977 to 1987 was the manager of the APL design group at the Thomas J. Watson Research Center. He received a B.Ch.E. from the City College of New York in 1941 and an M.A. in mathematics from Yale University in 1963, the latter under the IBM Resident Scholarship Program. He has received IBM Outstanding Contribution Awards for the development of APL and the development of APL\360.

Reprint Order No. G321-5443.