Product Review News

Dick Bowman
2 Dean Gardens
London E17 3QP
England
01-634-7639

Review Policy

The review policy of APL Quote Quad is an inclusive policy—we want to review as many APL and APL-relevant products as we possibly can. The invitation is a permanent one: if you are a vendor wanting your product reviewed then please contact me. If you are not a vendor but use a product which you feel is interesting and valuable for SIGAPL members, then we're just as pleased to hear from you.

An interesting dilemma presents itself as part of this policy: what are we going to do about flawed products? At one extreme we could just tough it out; if a review shows a product in a bad light then, having checked that the reviewer's findings are accurate, we publish as is. The vendor is unhappy at best, and out of business at worst. Taking the other extreme, we could decide to publish only favourable reviews. Then the members get mad because they pay good money for products that don't work.

So, this is what we're going to do:

We review the product, and if we find flaws we work with the vendor to determine whether they're flaws or features. In the published review we tell you that this has happened. If we don't get any improvement as a result of this process then we'll tell you that as well. The reason for this policy is that SIGAPL's primary purpose is "promoting the development and application of [...] APL"; a purpose which will not be achieved by unjustified denigration of sound products. But we will not be untruthful either—if our review experience makes us feel that a product will impede the promotion of APL, you'll know that too, and in no uncertain terms.

...Any reactions?

Review Plans

Response to my call for products for review has started well; we have many interesting items coming up in future issues of APL Quote Quad. Some of these, to be fair, predate me but I will be progressing them through to publication. Reviews being prepared include:

- **NewFase and WordPerfect:** a way of including APL code seamlessly into your conference papers and Quote Quad articles.
- **APL★PLUS/PC Version 10:** updates on an old favourite.
- **Tool of Thought VII:** the latest of NY/SIGAPL's annual seminars.
- **Dyalog APL/X:** APL on the X Window System.
- **Dyalog APL for DOS/386 Version 6.1:** updates on a new favourite.
- **IAPL/MAC:** an inexpensive first generation APL for the MAC.

Vendors,-get your products onto this list; write or call and we will organise a review. If you're not a vendor but think there's a product which APL Quote Quad ought to review, then let's hear from you as well. And if you fancy trying your hand at being a reviewer, remember we're always pleased to hear from volunteers. ■

APL.68000 Level II for the Amiga

Harry C. Bertuccelli
The Aerospace Corporation
Computer Systems Division
P.O. Box 92957
Los Angeles, CA 90000-2957 USA
213-336-6319

In the early 80's when personal desktop computers began to leave the private preserves of computer hobbyists I knew that I too had more than a mild interest in acquiring one. I had had the good fortune in the 70's to use the IBM 5100 in my office at work. As most of you undoubtedly know, one version of this desktop machine (the one I had) had a built-in APL interpreter. The company I was working for paid $12,000 for that computer. It was slow (by today's standards) and I was restricted to a 32K workspace size and a tape cartridge storage and retrieval system. Yet despite these limitations, my productivity as an engineering analyst grew substantially through the use of that 5100. And when I had to leave it (moving on to greener pastures), it was with a strong promise to myself that someday I would acquire such a machine in my home office.

Little did I realize at the time just how rapidly the technology would burgeon. When IBM introduced what became known as *the* PC (as though alternatives were not personal computers) I was not quick to jump, having already developed a preference for either Motorola or National SemiConduc-

tor technology. The commercial success of NSC's chips was in doubt, but Motorola's 68000 chip appeared to be on its way, even though it was being used at that time only in very expensive desktops, more workstations than personal computers.

As I looked for alternatives to the PC, I was accepting the likelihood of spending a substantial amount of money, as well as the apparent need to become familiar with Unix. Many times during this searching period I was tempted to forsake my dream system and go with the crowd. I was also tempted by Apple's offerings, especially the MacIntosh. Towards the end of 1985 I encountered the Amiga 1000. For me it was a stunning discovery. Almost everything I had been seeking was there—Motorola technology, multitasking, fantastic graphics, WIMP interface— and at a price I wouldn't have expected in my wildest moments.

The one fly in the ointment was that at that time an APL interpreter for the Amiga was not available. And APL at my fingertips was central to my needs. So for a time I closely tracked the trials and tribulations of early offerings to supply Amiga users with a means to run IBM PC programs, my interest being a way to use STSC's APL★PLUS. My concern happily proved to be short-lived; within a year, MicroAPL had come to the rescue, releasing versions of APL.68000 for the Mac, the Amiga, and the Atari. And what a rescue that was: the Amiga version allowed substantial control of Amiga's special features from APL!

However—time marches on. Even at the time of this initial version of APL for the Amiga, I had been using APL2 on a mainframe at work, becoming increasingly entranced by its advantages over "classical" APL. Still, users of Intel-based PC's were in the same boat, so I had no reason to feel slighted. Certainly the APL I was using on the Amiga was in my eyes superior to the APL being used on PC's. The feeling of deprivation returned, however, with the release of APL2/PC by IBM. At first the concern was minor, since Richard Nabavi of MicroAPL had assured APL.68000 users at APL87 of his commitment to implement a full nested-array interpreter which would include the features of APL2. At that time he spoke of a three-year time frame, indicating also that the first such implementation would probably be for the Mac because he saw Amiga systems as typically using too little memory.

When I inquired at APL89 regarding the status of that effort, Richard's response led me to believe that an enhanced interpreter was *still at least* three years away, since MicroAPL's major thrust had been to improve the Mac version of APL.68000, effectively bringing it up to the level of the Amiga version which had been the best of the three (Mac, Amiga, Atari) at the time of the original release. I certainly could understand the greater emphasis on the Mac, which had (and still does) a larger share of the market than does the Amiga. Nevertheless, from a purely personal point of view, I was very disappointed. I began again to think about a way to use PC software, perhaps via the mode currently available: the so-called BridgeBoard, for which a 386 version is expected "momentarily."

So I was jolted by the news of MicroAPL's release of Level II, and for all three of the Motorola-based desktop computers at that. That surprise was augmented by joy when I discovered how far MicroAPL had gone in matching the APL2 system I've been using on the mainframe. Once again I had been rescued from any dependence on the pedestrian technology of the PC-world (or should I say the PS-world, where PS signifies "personal system," another presumption—this time, that IBM's offering constitutes the only system so describable).

Any one who feels that the last slur regarding PS-technology is simply the ravings of an Amiga bigot should spend a little time contemplating the import of an article appearing in the January 1991 issue of *Byte* Magazine (pp329-334) entitled *"The Object-Oriented Amiga Exec."* The author, Tim Holloway, is president of MTS Associates, a system software development firm in Jackonsonville, Florida. He does not explicitly summarize his description this way, but I believe the following sentence is a fair rephrasing of his remarks. The Amiga operating system is elegant, largely transparent, and sparing in its memory requirements; in contrast, the recent operating systems of both the IBM and the MacIntosh personal computers (which claim comparable capabilities) are ponderous, largely opaque, and memory guzzlers.

Overview

Strictly speaking, of course, APL2 is simply one variety of enhanced APL. Practically speaking, however, only two versions have been serious contenders for widespread use: that offered by IBM (to which STSC's version has conceded dominance, promising to modify their initial version to conform with IBM's), and that offered by Sharp (which no longer exists as a distinct entity, having been absorbed by a company whose primary business interests lie elsewhere).

It is interesting to note that Level II is prepared to keep up with the evolution of APL by incorporating into its APL character-set not only all of the characters of IBM's APL2, but also all of the characters employed by Sharp's enhanced APL as well. In particular, left and right tack, the dieresis-jot and dieresis-circle combinations, and bar-comma (for leading-axis catenate) are included. Furthermore, Level II retains the diamond (statement separator) and the four quad-enclosed symbols for its APL-oriented filing system.

MicroAPL has most agreeably chosen not to make separate products, one requiring, the other not requiring a floating-point coprocessor. Both versions, APL2FPU and APL2, are found on the diskette provided.

A very welcome feature of APL.68000 Level II is the ease provided in moving workspaces between it, mainframe APL2, and APL2/PC: the *)IN* and *)OUT* system commands are so implemented as to obviate any character translation requirements. I did test one facet of this generality: I used the *)IN* command on several transfer files created by APL2 on the IBM mainframe at work. The only hitch encountered was my initial failure to remove the carriage-return/line-feed pairs that had been tacked onto the end of each 80-character line during the transfer from the IBM mainframe to a VAX, a transfer enabling me to use the KERMIT protocol to move the file to my Amiga (since we didn't have a working KERMIT on the IBM system).

Specifics

My place of work was selected as a testing site for APL2/370 before the initial release. Consequently, I've had extensive experience with this marvelous tool. I've also faced the problem of teaching APL2 to technical coworkers (engineers, physicists, and mathematicians). This experience has led me to value certain features of the language as essential to its utility for technical applications. I would be unfavorably disposed towards any enhanced APL for a personal computer that did not have at least the following features:

- nested and/or mixed arrays plus a modicum of supporting primitives to construct, rearrange, and exploit the nesting—the all-important "each" operator, and the functions "enclose," "partition," "disclose," "pick," "depth," and "match";
- enlargement of the class of acceptable function operands beyond primitive scalar functions to include not only user-written and/or non-scalar functions, but also operator-derived functions;
- enlargement of the class of operators to allow the user to design his own operators, which are as unrestricted with regard to operands as are primitive operators;
- strict adherence to the binding hierarchy of APL2 syntax, a clean scheme for rationalizing symbol grouping in APL expressions.

I'm happy to report that (since release 1.16) APL.68000 Level II has met these primary requirements with flying colors. In particular, let me note that I exercised many of the APL2 tools I had developed on the mainframe, as a way of testing Level II's fidelity to the binding hierarchy rules. I am now confident that using my APL2 coding habits with Level II will not bring me to grief as a consequence of some essential syntactical discrepancy.

But MicroAPL has gone well beyond this minimal set of features, incorporating in Level II almost all of the primitives which distinguish APL2 from VS APL:

- multiple (vector) specification
- without (dyadic ~), enlist (monadic ∈), find (dyadic ⍷), first (monadic ↑), grade with collating sequence (dyadic ⍋ and ⍒)
- axis qualification with take (dyadic ↑), drop (dyadic ↓), and ravel (monadic ,)
- almost all of the system variables and functions that are as appropriate to a desktop as to a mainframe, including (in particular) the important error control functions *⎕EA*, *⎕EC*, and *⎕ES*

The initial version of Level II provides a number of features which were not provided by Version 1 of APL2/PC. I assign high importance especially to:

- selective specification, although what is acceptable in the specification (left-hand) portion of such a statement is narrower than in APL2/370;
- index function (⌷, including the option of axis qualification), which merges better than bracket indexing with the nested-array orientation of APL2 (allowing, e.g., indexing to be an operand of the each operator);
- partition function (dyadic ⊂), although currently lacking an axis qualification option.

To be sure, there *are* some discrepancies between Level II and APL2/370, but for a first version I find it surprising that the differences are as small as they are. Furthermore, for some of the differences I find that I favor the Level II version over that of APL2/370.

Some of the discrepancies have already been noted—such as no axis qualification for partition, and greater restrictions on selective specification. One of the more interesting differences is the use of negative integers in an operand for replication. With APL2/370 such negative integers are "extra," in the sense that they play no role in conforming the operand with the argument, and serve simply to place fillers into the result. Thus,

```
2 ¯1 3/'AB'  ⟷  'AA BBB'
```

In Level II, such an expression would lead to a *LENGTH ERROR*, since negative integers do play a role in the conformance requirement—i.e., the argu-

ment '*AB*' calls for a two-element operand: a negative integer signals a count of fillers *in place of* the corresponding item. Thus,

```
2 ¯4/'AB'   ⇔   'AA    '
¯2 4/'AB'   ⇔   '  BBBB'
```

On the other hand, to get '*AA BBB*' from '*AB*' with Level II, use expand:

```
2 ¯1 3\'AB'
```

an illegal expression in APL2/370, which accepts only a Boolean operand for expand. Frankly I prefer the Level II scheme: it better retains the intuitive distinction of compression (/) versus expansion (\). In the APL2/370 scheme that distinction has become muddled.

Some features of APL2/370 are incompletely implemented in Level II:

- Fillers (as in overtake, for example) are properly handled only for the last axis. Where fillers are required for a different axis, the prototype for the entire array is used.
- Display wrapping is by line rather than by plane.
- Numeric grade up or down accepts arguments only of rank < 2.
- Character grade up or down accepts collating arrays (left argument) only of rank < 2.
- Format does not include the "by-example" option.
- Although ⎕*FX* is ambi-valent, the sole control possible with the optional left argument is whether or not the resulting function is locked.
- As with Level I, name sorting for ⎕*NL*,)*FNS*, etc. is on the first letter only.

There are other "shortages," some of which have already been noted. As for the rest, I regard them as too trivial to explicitly list. Some of them, in fact, are irrelevant in a non-mainframe environment. On the other hand, Level II has retained all Level I features that do not conflict with eventual acceptance of all APL2/370 code which is not mainframe specific.

- The diamond separator is still legal.
- Groups have been retained. (This may prove temporary, until indirect naming is implemented.)
- The very useful APL-oriented overlay and file systems have not only been retained, but enhanced to handle nested arrays.

Moreover, Level II has some useful system commands not in APL2/370:

-)*XLOAD* to ignore ⎕*LX* when loading;
- silent versions of commands which suppress the usual displayed messages — viz.,)*SCOPY*,)*SDROP*,)*SLOAD*,)*SPCOPY*,)*SSAVE*, and)*SWSID*.

Looking Ahead

MicroAPL has announced their intention to eventually close the gap, but in the meantime there are a number of features of APL2/370 currently missing from Level II. I will not mention them all; they've been listed elsewhere (see, e.g., Issue 13, August 1990 of *MicroAPL News*). Instead I will focus on a few I regard as especially important; hopefully, these are also high priority items to MicroAPL in their schedule of future upgrades.

- Axis qualification for scalar functions provides a very convenient way to specify what otherwise requires a fairly clumsy expression.
- Dyadic (*n*-wise) reduction also is extremely useful, particularly the case where the left argument is 2.
- Inclusion of complex numbers as a primitive number type is very important to me. I have a large collection of APL2 tools designed on the mainframe that depend on this data type; all of these would have to be recast if used with Level II before complex numbers are made available.
- It is difficult to exaggerate the power given to an APL2/370 user via the Name Association (⎕*NA*) system function. Not only does it then become easy to take advantage of other languages for those computing subtasks where they are more efficient than APL, but it also provides a mechanism for reducing name clutter in a workspace: multiple namescopes (also known as "packaged" workspaces). Implementation of all aspects of ⎕*NA* in Level II will probably take some time, since it requires ties to linkage conventions appropriate to the variety of platforms to which Level II is ported.
- Level II is currently without a cluster of features which I have found very helpful in debugging. The most useful of these is the ability to resume execution at the point of suspension (which might have been in the midst of a line) using "→⍳0". Associated with this facility are the system variables ⎕*L* and ⎕*R* holding the values of current left and right arguments, respectively. APL.68000 has the minor problem of name conflict, since it uses the names ⎕*L* and ⎕*R* for the linefeed and carriage return characters, respectively—but I personally would have little trouble accepting new names either

for the special characters or for the current left and right arguments. Clearly MicroAPL didn't cross this bridge yet since they didn't incorporate immediate-calculation errors into the state indicator stack as has IBM with APL2/370, another feature I find useful. The final item of this cluster I miss is the *)SIS* command, which displays all suspended statements.

- I find the indirect *)COPY* and *)ERASE* features of APL2/370 preferable to *groups*, primarily because the names being grouped for indirect reference are more accessible (more directly controlled, and easily formatted in documentation utilities).
- Level II continues to rely on *)SYMBOLS* to control the size of the symbol table; the automatic expansion of this table, whenever necessary, is a very welcome feature of APL2/370.

Extra Features in Amiga Version of Level II

None of these are specific to Level II; they had already been provided in Level I for the Amiga. To begin with, there are a host of tools included which permit APL use of AmigaDOS environmental facilities:

- set up menus;
- use full-screen dialog, including design and control of windows and requestors;
- create sounds: noise, music, and speech;
- enliven user interface via low-level graphics: palette control, straight line drawing, polygons, arcs, rectangles, ovals, round-cornered rectangles; shapes in outline or filled; text-support (incl multiple text styles and fonts).
- assign functions keys;
- under program control you can reconfigure the keyboard, specify keyboard buffering, and use clipboard transfers;

Besides the typical complement for APL.68000 of

- an excellent APL-oriented file system which includes "overlays" (grouping any collection of APL objects into a single file item),
- multiple editing-windows with clipboard support to exchange text between these windows,
- terminal emulation (VT100),

there are tools to exploit the multitasking capabilities of the Amiga: in APL programs you can

- detect and respond to keyboard, mouse, disk drive, menu, and window events;
- initiate new APL sessions with specifiable workspace size, and with an optional initial character string (up to 40 characters) to specify a startup process;
- share data between separate APL sessions, wherein each APL task can temporarily restrict access to maintain the integrity of an update.

I applaud all of these extras. However, there are a few characteristics of APL.68000 for the Amiga (as true of Level I as of Level II) about which I will register some small protests. There are two major areas where I hope that MicroAPL will revamp the current modus operandi.

Currently Workbench activation of APL.68000 (regardless of level) leads to a workspace size which grabs almost all available memory. This is a defensible design decision for systems with only a small amount of available memory. But for Amiga systems with substantial RAM (mine has 18 Megs: 2M chip, 16M fast), this default size is totally unacceptable, "discourteous" to the next program which might be activated (as is likely when multitasking is an option). It is possible to counter this "greed" by invoking APL via the CLI (Command Line Interpreter) wherein you can specify an initial workspace size (but not, unfortunately, a character string to specify a startup process—as you can with a second APL session once an initial session has been started). Since I can't accept the "greedy" default, I always use the CLI. It ought to be possible, however, for the user to establish his own default for workspace size in the absence of direct specification via the CLI, or when activation of the initial session is via double-clicking the icon. And that brings up a related defect.

The Amiga-specific APL.68000 manual indicates that you can start an APL session by double-clicking a workspace icon. Certainly the Amiga operating system facilitates that option, and many other Amiga programs allow activation by double-clicking a project icon. However, APL.68000 has not properly coupled to this feature. Icons on the Amiga are associated with so-called INFO files, and the INFO item of the Workbench menu allows a user to edit certain fields of an INFO file. Such editing is a necessity to exploit this feature of activating a program by double-clicking on a project icon. Thus, it is easy to accommodate a file organization wherein the "projects" are found in a different directory than the "tool" (the program which constructs and uses the projects). It is only necessary to place in the Default Tool field of the INFO file for the project a character string which tells the operating system where to find the appropriate tool. In the case of APL.68000, the "projects" are saved workspaces and the "tool" is the interpreter (APL.68000). Unfortunately, this simple mechanism doesn't work with APL.68000 except when the workspace is in

the same directory. If you place the correct information into the Default Tool field of the INFO file for a workspace placed in a different directory (quite likely in a hard-disk system wherein, e.g., certain libraries might be kept on diskettes), and you double-click on the workspace icon, you certainly succeed in activating the interpreter, *but* the interpreter reports it is unable to find the workspace. Other programs don't suffer this befuddlement. MicroAPL has a correctable problem here.

For me these are nuisance problems. My very first activity upon invoking APL is to load a workspace called *SETUP* which establishes my function key assignments and defines my libraries. To establish a sensible workspace size I use a CLI command (a small nuisance); I then load *SETUP* using the menu)*LOAD* option (a second small nuisance). What I'd prefer is simply to double-click on the icon for *SETUP*, and thereby get both a workspace size I favor and the setup I've chosen.

A second area of discontent for me is the limited size (about two screens) of what I'll call the "log," although that term is not really appropriate for the current implementation. A log, however, is what I want—i.e., an accurate record of activity: inputs and system responses. Accurate it is not, as of now, since if you modify an earlier line as a way of speeding up input, it is not refreshed when you offer it to the interpreter. Instead the modified line appears in two places: where the original line was, and at the bottom of the "log." I'd like the original line reinstated, the modified line appearing only at the bottom. Furthermore, I would urge additional features found in the Session Manager associated with APL2/370: an adjustable buffer size (determining how soon lines get lost at the top), automatic saving of the log at the end of a session, and automatic reinstatement at the beginning of the next session (to facilitate continuity of development work across sessions).

Performance

Personally, performance (efficiency) is an aspect of APL-implemented applications that does not concern me to the degree that it does many users, *provided* the interpreter is "reasonably" fast. In that respect I find APL.68000 Level II more than satisfactory. In choosing to use APL for many of my computing tasks, I do not do so under any illusions as to its efficiency relative to other languages. The convenience and speed afforded by APL to move from concept to working program more than compensate for any loss in execution efficiency. However, I would be remiss if I failed to provide some summary of the performance of Level II relative to comparable interpreters on other systems.

Obviously performance depends upon the platform. There were two distinct Amiga systems used to test performance. The weaker system was an Amiga 1000 (the original model) which uses a 68000 CPU operating at 7.14 Mhz and no floating-point coprocessor. This was my only system until last July, and was equipped with 2.5 Megs of fast RAM, the usual 0.5 Meg of chip RAM, no hard disk, and 2 diskette drives. On this system (let me refer to it simply as A1000) version APL2 was the exercised version of Level II.

The stronger system was a fully loaded (16 Megs fast RAM, 2 Megs chip RAM) Amiga 3000 using a 25 Mhz 68030 as CPU and a 25 Mhz 68882 as floating-point coprocessor, and equipped with a 170 Mbyte hard disk and two diskette drives. APL2FPU was the version of Level II exercised on the A3000 (the abbreviated designation I'll use for this system).

The September 1990 issue of *APL Quote Quad* (Vol. 21, No. 1) contained a review of Dyalog APL's port to PS/2. That review used some benchmarks to compare the performance of this port with the performances of APL★PLUS II and APL2/PC. Although I would prefer to use different benchmarks, I've decided to go along with those used in the afore-mentioned review as a simple way of comparing Level II performance on A1000 and A3000 with that of Dyalog APL, APL★PLUS II, and APL2/PC on a 25 Mhz PS/2 Mod 70. That comparison will require, of course, joint perusal of both this and the earlier review.

The table below is puzzling at first glance, because some items have two execution times (in milliseconds). These are tests which make use of either *BV* or *BM* (Boolean vector and matrix, respectively). The Dyalog APL review contained the listing of a test program *TEST*0. Line 5 reads:

```
BM←50 100⍴BV←?1000⍴2
```

Since ⎕*IO* has been set to zero, both *BV* and *BM* consist only of ones and zeros. But with Level II (or with APL2/370, or quite possibly with some of the interpreters noted in that other review), this manner of construction leads to *integer* arrays—i.e., the zeros and ones are two-byte integers (or two-byte integers on APL2/370). APL.68000, however, does have a Boolean type wherein but one bit is allocated per item; and that type for *BM* and *BV* can be constructed using

```
BM←50 100⍴BV←1=?1000⍴2
```

Doing so has a substantial impact on the execution time of most (though not all) of the items making use of *BM* or *BV*. In the tabulation below, wherever there are two numbers side by side, the first refers to the integer arrays and the second to Boolean arrays. Although the Boolean scan and Boolean compare tests get massive speed gains via Boolean arrays, the compression tests favor the integer arrays a bit (dropping execution time by

about 30%). Presumably, the latter tilt towards integer operands for "/" is the natural consequence of generalizing compression to replication in APL2, a generalization which allows an integer operand having no Boolean counterpart.

	A1000	A3000
INT ADD	381.0	67.5
FP ADD	8717.0	187.3
INT MULT	7134.3	238.0
FP MULT	13829.1	199.4
INDEX	126.4, 180.8	28.4, 42.4
CHAR COMPR	125.1, 178.0	29.5, 43.3
INT COMPR	119.0, 173.2	42.0, 27.9
INT +RED	25.0	4.0
INT ⌈RED	24.1	4.0
BOOL SCAN	67794.3, 100.0	17358.4, 18.0
MAT ROTAT	576.2	144.1
CHAR TRANS	753.8	171.5
INT TRANS	804.1	178.0
VEC OF VECS	1019.3	224.2
PARTITION	456.9	105.6
RHO EACH	630.1	157.8
VEC COMPAR	21.1	3.0
INT SRT	710.2	151.4
BOOL COMPAR	437.0, 185.1	108.6, 38.0
IOTA	1552.3	76.2

Although the numbers for A3000 are generally in the same ball park as those in the tabulation found in the Dyalog APL review, substantially better than the PS/2 Mod 70 in a few instances (the *BOOL SCAN* test with Boolean *BM*, and the dyadic *IOTA* test), but substantially poorer in many instances (e.g., *MAT ROTAT*, *CHAR* and *INT TRANS*, *INT SRT*, and *BOOL COMPAR*), its relative performance seems a bit weak to me, considering the potential strength of a 25 Mhz 68030/68882 combination. I can only speculate about the reasons —one contributing source possibly being the multitasking operating system of the Amiga, which makes it impossible for the system to devote its exclusive attention to an APL computing task.

Overall, I'm not really unhappy with these numbers. In a practical sense more power would simply be a luxury. I did find some evidence that the algorithm employed by APL.68000 for the interpretation of ⌹ may need some fine-tuning. The following comparison between A3000 with APL.68000 Level II and a PS/2 (20 Mhz 80386/80387) with APL2/PC uses two test expressions:

```
Q*0.5   ←→where Q←?40000⍴999999
⌹A      ←→where A←(¯50000+?40 40⍴99999)÷25000
```

the first to test the performance of the 25 Mhz 68882 relative to the 20 MHz 80387 (since both have a single instruction for square root), the second to compare execution times in inverting a large matrix. In all cases, in order to remove any dependence upon data, ⎕*RL* is initialized to 16807 before the use of "*?*".

```
     ∇ ID TEST X;C;Z
[1]    C←⊂(Z←'TEST',⍕ID),';⍙T;⍙S'
[2]    C←C,'⍙T←⎕AI[2]' ('⍙S←',X) '⎕AI[2]-⍙T'
[3]    ⍞←(⎕FX C),': '
[4]    ⍎Z
[5]    C←⎕EX Z
     ∇
```

Expression	25 Mhz A3000 Level II	20 Mhz PS/2 APL2/PC
1 *TEST* '*Q*0.5'	680	4510
2 *TEST* '⌹*A*'	13720	5220

Conclusions

Notwithstanding the few complaints(?) I've made in this review, APL.68000 Level II for the Amiga is an excellent product worth every penny of its very reasonable price. It provides a surprisingly large portion of APL2/370 features for an initial release. MicroAPL's proven track record lends credence to my belief that it won't be long before as much of APL2/370 as is appropriate to a personal desktop computing system will be fully incorporated into Level II. ■

CPCUG APL Lessons Now Available for Several Interpreters

Dick Holt
HRH Systems
Box 4496
Silver Spring, MD 20914
202-586-4449

A series of twenty-six interactive self-teaching on-disk lessons are now available for TryAPL2, IBM APL2, STSC APL★PLUS and Pocket APL, Sharp, and I-APL. Based on the work of Z. V. Jizba, these lessons were transferred to multiple APL formats by the APLSIG of the Capital PC User Group (CPCUG) for use in their 1991 APL classes. Lessons were edited to make them more generic, and to incorporate classroom experience.

CPCUG lessons are downloadable free from the BBS\APL: 301-384-3672, 300/1200/2400 baud, N-8-1, 24 hours a day. See File menu Y.

Lessons may be ordered by mail from HRH Systems at the above address, for US$19, postage-paid worldwide. Mail orders should specify disk size and APL version (.TRY, .ATF, .AWS, .SAW, or .IWS). Checks accepted in any national currency.

Lessons will be sent free upon request to Eastern Europe and the Soviet Union. CPCUG lessons were also a part of the APL91 Software Exchange. ■