File - Multics APL

Date

To: Prof. Corbató, Prof. Saltzer

From: Daniel Bricklin

Subject: Proposed Improved APL for Multics

APL is a powerful computer language for manipulating arrays of data. It is used by several popular courses in the EE department at MIT, as well as by research groups. There are useful subsystems written in APL, such as the MARTHA system written at MIT. APL has proven easy to learn, and is a very potent tool for engineers, and other people who work with arrays of data. It is also becoming increasingly popular as a programming language for business applications. There are many books available to help teach the beginner -- both from computer manufacturers and normal publishing houses.

APL is an interactive language. Fast response time is important, since it is often used interactively during design sessions. A defacto standard of the APL language has arisen with the APL/360 implementation. Most of the texts on APL assume that one is using this version of APL. APL/360 not only consists of a language, it also defines an editor, and command environment. To be able to fully exploit the texts on the market, and for other obvious reasons, any APL should at least contain APL/360 as a subset.

The Multics computer system provides a very hospitable base for an APL system. Within Multics it is easy to provide encapsulated subsystems, such as for use by a course for

providing computer time to students. Unlike other systems, Multics makes security rather simple. A restricted APL system can be provided, or APL could be incorporated into a larger system, such as a data base inquiry system. Multics provides to subsystems the use of many peripherals, so APL could use tapes, line printers, card readers and many other useful devices. The Multics virtual memory allows APL workspaces of very large size, so that one can work with arrays many times larger than those allowed by standard APL/360. Use of the Multics I/O system will allow using terminals such as the Tektronix APL display, normal ASCII terminals, as well as the ARPA network. By allowing calls to functions external to the APL environment, routines written in other languages, such as PL/I, FORTRAN and LISP, could be used by APL programs. Also, there are ways to allow routines written in these other languages to use functions written in APL through a suitable interface. These enhancements to a strict APL/360 type system should increase the usefulness of APL, and provide an APL superior in several aspects to most others available on other machines.

The APL system currently on Multics does not meet all of the requirements outlined above. While it allows use of facilities external to APL, such as routines written in other languages, it is inadequate for use in many applications. The editor grossly differs from the one provided by APL/360, and requires many pages of supporting documentation to make it usable by the novice. This negates much of the usefulness of the texts about APL on the market. In addition to the obvious difference with the editor,

there are small differences, and deficiencies -- mainly extensions to the original APL/360 that haven't been added to APL/Multics as yet. The glaring deficiency, though, is in speed. APL/Multics averages about 10 times slower than APL/360 in cpu time (taking in account differences in the raw speed of the base machine).

What we are proposing to do is re-write APL/Multics to remove the deficiencies mentioned above. We would write a new APL that would appear as close to APL/360 as the Multics system allows. There would be no gross deficiencies, as compared to APL/360, and there would be several valuable extensions. The list of differences, and how to get around them (e.g. new login procedure) should fit on one side of a sheet of paper. This will allow existing texts to be used by anyone. Execution should be sped up by at least a factor of 10 from the current version. Also, use of Multics facilities, such as tape and line printers, should be made more readily available.

By examining the cost, effort, and final performance of similar projects on Multics (the current APL, LISP, Seal), approximate estimates of cost (both in money and development time) and performance for the new APL can be determined. If the proper people are available, the entire re-write could be done in 3-4 months, by 5-6 people. A tentative budget would look as follows:

20 man-months @ $1250/mo. (inc. O/H)	$25,000
6180 time @ $2000/man-month	40,000
Space, terminal rental, etc. @ 850/mo.	3,400

Estimated total cost: 68,400

If the go-ahead were given now, the project could begin in June, 1973, and be completed by September, 1973.

Several good people are available to do the re-write, and are interested if they can be obtained before they make other commitments. They include:

Dan Bricklin - worked on the original APL/Multics, and the Multics LISP.

Paul Green - worked on Multics PL/I compiler, and the Seal compiler.

Rick Gumpertz - worked on the Multics assembler (ALM) and some interactive languages.

Rich Lamson - worked on Multics Dartmouth simulation, and knows APL/360 very well.

Dave Moon - worked on Multics LISP.

Dave Reed - led the Multics LISP project.

If not all of the above people can be obtained, there are others available, including members of the old Planner project. All of the people are enthusiastic about working on a new APL for Multics. Most have experience that is needed to produce a result that meets the specifications. Several of them are leaving after the summer, or will be committed to other projects. Therefore, if a re-write of APL will be desirable anytime in the next year or two, it would be advisable to do it now, while this talent with interactive subsystems is available. Dan Bricklin is willing to lead the group.

Now that a need for a re-write of APL/Multics has been

established, an estimate of its cost, and a list of who is available to write it, all that remains is to justify the claims that it will meet the specifications.

The first problem, that of the unusual editor, and other differences from APL/360, can be overcome by just writing a new editor for APL/Multics. The only difficult problem is with using the "ATTN" button for editing, as is done with APL/360. At present the Multics TTY DIM prevents us from simulating the APL/360 function, but that should be changed by the end of August, 1973. This change will be made by the Multics group at CISL, and is necessary for our planned improvements. Also, we plan to use character conversion done outside of the APL subsystem (as an IOSIM) to simplify use of APL from terminals other than IBM 2741's with the special typeball. The use of an IOSIM (such as the code converter developed by Ken Pogran and others) will simplify the construction of the APL interpreter, by removing the problem of writing another special piece of code (as was done with the current APL).

The current APL/Multics has a few deeply ingrained design flaws which necessitate a total re-write rather than a mere tightening up of code. Some of the control data that it uses is stored in a special format. Subroutine calls are often necessary to decode the data for use. Unfortunately, these calls are often in the middle of deep loops. The full PL/I call on Multics is expensive (about 100 usec. on the 6180), and we find that a large part of the APL/Multics cpu time is spent in the call operator. Redesigning the data formats and other simple stratagies should

replace most of the calls with simple, inline code (about 10 usec.). Restructuring of the data, though, means changing all the modules that refer to it -- i.e. most of the interpreter.

The cost of using APL/Multics will be further reduced by other techniques. By changing the internal format of functions we will reduce the size of workspaces, thereby reducing paging and online storage requirements. The current APL is very wasteful in storing functions, and we should be able to reduce that by a factor of 2.

The speed of execution will be increased by making several changes to the execution environment, and to the interpretation algorithm. Currently, APL/Multics looks up reduction rules in a table as it interprets each line. This was metered, and was shown to be very expensive. By "writing out" the rules as explicit PL/I statements, and other simple techniques, we should remove 20-30 percent overhead on interpretation. Currently, APL/Multics allocates and frees (using subroutine calls) all temporary storage. Most of this could be replaced by a push down stack for temporary values, which can be manipulated quickly and without subroutine calls. This technique was tried in an experimental version of APL/Multics, and seems to have significant results. It should remove another 40 percent. By using methods used by the Multics LISP interpreter (better use of symbol table, faster function call/return) we should be able to easily get the rest of the speed up. Multics LISP is currently ten times faster than Multics APL, for similar tasks, and APL seems intrinsically easier to interpret.

The final speed up should come with the recode of the operators. We plan to use a simpler data format, which can take advantage of the EIS instructions on the 6180. It has already been shown that if operators are coded carefully they can be just as fast on APL/Multics as APL/360. Only a few of the operators have been recoded with this in mind. The re-write of APL will involve recoding the rest.

To achieve maximum speed, we are planning to use the EIS. The language for the entire implementation will be PL/I. Therefore, it is important that the PL/I compiler be able to produce inline EIS code. We plan to design all the interfaces so that critical modules can be recoded in ALM at some later date to get the last few percent of performance. Due to the number of extensive changes that must be made, it looks like a total re-write of APL/Multics would have to be done. Many of the algorithms used in the current APL can be used, though, which will save a significant amount of time in development.

We are rather sure that the new APL/Multics will be a significant improvement over the current version. As noted above, we base our optimism on the current speed of Multics LISP which is one of the faster versions of LISP available. Similar techniques will be used, and implemented by the same people. We also tried out some of the new design features in a special subset of APL, and had dramatic speed ups. Our goal is an overall speed up of ten over the current APL, which should make us competitive with other APL's. The chances of us achieving that goal are very good, we feel -- about 80-90%. The chances of

as coming within a factor of 2 of our goal are better than 95%. Some of this is dependent on the speed of EIS and the code produced by the compiler. There could, of course, be a time over-run of a month or two, but this would probably only be because of lost time due to vacations, etc.

One final question is the transferability of the APL system to other H6000 machines. Given that a similar PL/I compiler is available to run on those machines, there should not be many problems in performing the transfer. The design of the new APL does not make many assumptions about the operating system. Most terminal I/O and file manipulation is restricted to a few modules. If the transfer of the APL to another member of the H6000 line is desired, we would like to have one additional person on the project who is familiar with GCOS. This would be a part-time job, mainly to sit in on design decisions that may affect transferability, and to write the modules that have to be different on GCOS. If the H6000 GCOS PL/I compiler were available on Multics GCOS the entire operation should be rather simple, and should be completed within a few months of the Multics effort. Again, performance would depend on the quality of the code produced by PL/I, and the competence of the people involved.