file - Multics design proposals

Saltzer

copy Corby
[illegible]

HONEYWELL INTEROFFICE CORRESPONDENCE

~~FJC~~
Corbató

DATE: June 7, 1972

TO: R. A. Freiburghouse

FROM: M. G. Smith, D. Bricklin

DIVISION: CISL/PCO

SUBJECT: Proposed Enhancements to Multics APL

Summary.

The present Multics APL (apl 052572) is, on a lightly-loaded Multics system, slower than APL/360 under CP-67 by a factor of 20:1 (indicated virtual times; ours includes system overhead; theirs does not) when running a typical test script. We estimate that the improvements summarized below will change the factor to 3:1, at a cost of one man-year of effort spend enhancing APL. Actual test results on a subset APL have already demonstrated a 5:1 ratio. We recommend that this modest price be paid to secure such an improved product.

Proposed Improvements.

Two major changes to Multics APL are proposed which will each result in a great performance improvement. These are a new temporary management scheme and a new fast call mechanism.

The new temporary management scheme involves using a stack discipline to keep track of temporary results during expression evaluation, instead of individual allocations and deallocations in a free-storage pool. Savings result not only from the elimination of allocations and deallocations, but also from eliminating the bookkeeping necessary to properly deallocate storage upon errors.

The new fast call mechanism involves binding all names used in a function at function call time, rather than each time the names are encountered during function execution. Hence, much more than the call alone is speeded up. In combination with the new temporary management scheme, function returns will be simply a release of the stack, eliminating some costly deallocations which are now necessary.

Each of the above changes significantly alters the data formats processed by the APL interpreter, and hence impacts upon most modules of the interpreter to a greater or lesser extent. All things considered, if those changes are to be implemented, it will probably be easier to recode most modules than to edit and patch the old ones. This will result in a beneficial general clean-up and permit us to throw in a couple of new features and other performance improvements at almost no additional cost.

To be more specific, an outline of the proposed changes is given below:

I. New Temporary Management Scheme.

A. New parser, handles all on-conditions for operator-control.
B. New operator-controls (soplop, dopmop, sopmop).
C. New operators, including improved algorithms and general clean-up.
D. New error recovery.
E. Save/load new workspace format, including real pointers throughout.

II.Fast Call Mechanism.

A. New function caller, which binds all names at call time in a local symbol table placed in stack.
B. New function return, which merely releases the stack.
C. New procedure-bead builder.
D. New lex which builds local symbol table skeleton, shorter tokens, special tokens for APL system commands.
E. Parser to implement fast step from line-to-line.
F. Save/load new workspace format.

III. While We're At It (A grab-bag of inexpensive items).

A. Function editing will be able to use any Multics editor.
B. Provision of a real I/O dim in place of internal routines, support 33 & 35 ttys & ARPA network, redo internal read/write routines conformably.
C. Execute operator.
D. Stop/trace control.
E. Profile option.
F. Limited-service or subsystem version.

Three-Month Plan.

Assuming that one man-year of effort cannot be invested in APL, what can be done in three months? Unfortunately, the major speed-ups proposed above are the new temporary management scheme and the fast call mechanism, each of which spreads across nearly all of APL because the underlying data structures are considerably altered. To do either one would cost almost one man-year by itself, as so many modules would have to be changed. Therefore, it seems that the only worthwhile expenditure of three months in terms of performance could be had be redoing the individual operator routines, improving their algorithms where possible, and a general cleaning-up and belt-tightening of all the modules. This might result in a performance improvement of 50 per-cent, giving us a 10:1 ratio to APL/360.

Conclusion and Recommendation.

The present Multics APL is not in any sense competitive with or even a reasonable alternative to APL/360. The modest investment of one man-year would completely change this; Multics APL would then be within approximately the machine differences of APL/360 (on a lightly-loaded system). The follow-on processor would make us actually faster in some cases. In view of the low cost of achieving this, the availability of willing and able personnel, and the rising importance of APL in the marketplace, we definitely recommend that Multics APL be so enhanced. Also, since a three-month plan would not permit any fundamental changes in workspace organization so as to allow either the new temporary management scheme or the fast call mechanism to be implemented, the three-month plan is hardly worth considering as an alternative.