October 14, 1969

TO: Distribution

SUBJECT: System performance effects of the new PL/I compiler

FROM: F. J. Corbató

To determine the impact of the new PL/I compiler, a study has been made with the procedure "get_calendar". In particular, a sequence of evolutionary steps were evaluated starting from:

Case 1.epl: A circa 1967 version of get_calendar (1-1/2 scp source-coded-pages)) which along with its companion, calendar_output, was coded in a casual EPL style with strong use of the substr function. Two entry points were implemented, brief and full, both performing functions with rather baroque specs. External calls were made to: calendar_output (2 scp) and bin_dec (1/2 scp).

Case 1.pl1: get_calendar, calendar_output and bin_dec were all given minimum modifications to become pl1 compilable. Changes were:

get_calendar: 1 - insertion of "returns( )" around a return argument declaration.

1 - change of the "/" operator to the "divide" function.

	1 -	Correction of a latent bug present in the EPL coded version which caused the procedure results to be stored outside the area of the return argument. This bug went undetected previously because the EPL run-time string-operator subroutine has a compensating side-effect. Use of PL/I revealed the problem and forced a correction.
calendar_output:	13 -	changes of the "/" operator to the "divide" function
	1 -	insertion of explicit "fixed" and "float" functions to avoid a temporary compiler bug.
bin_dec:	1 -	change of the "/" operator to the "divide" function.

The clerical changes could all have been made with two compilations since the pℓ1 compiler gave thorough diagnostics. (Several compiler bugs and temporary limitations were also encountered along the way. This class of bugs, however, is not considered significant since not only is the number of such bugs finite but they should be quickly discovered and eliminated with normal use.)

Case 2.epl: The specifications of get_calendar$brief were changed and streamlined to support most previous uses of the entry and yet allow simpler coding of the procedure. (The "full" entry point was abandoned.) The new coding style was "1969 restricted EPL". External calls were avoided and especially the use of the string operator subroutines.

Case 2.pℓ1: The version of get_calendar prepared for Case 2.epl was minimally modified. Changes were:

6 - changes of the "/" operator to the "divide" function.

4 - additions of explicit aligned/unaligned attributes to keep the same array packing rules as EPL.

Case 3.pℓ1: A new version of get_calendar was coded taking advantage of PL/I features and strengths of the new compiler. The specs were the same as for Case 2. The principal changes were use of "initial" declaration to initialize internal static and straightforward use of the "substr" function.

The running time of the above programs was determined by repeatedly calling them in a tight loop and noting the minimum execution time. These results along with the procedure sizes are given below in Table 1. (Size figures are decimal; the first portion is the text, the second the linkage.)

Table 1

case	epl	pℓ1
1		
(get_calendar)	59.3 ms., 1329 + 88 wds.	17.6 ms., 718 + 52 wds.
(calendar_output)	676 + 36	432 + 36
(bin_dec)	260 + 20	153 + 20
Total:	59.3 ms., 2265 + 144 wds.	17.6 ms.,1303 + 108 wds.
2	6.2 ms., 666 + 54 wds.	6.6 ms., 453 + 50 wds.
3		3.0 ms., 474 + 52 wds.

Notes:
1. The execution time of case 2.pℓ1 is expected to improve somewhat as further improvements are made in the object code generated by the PL/I compiler.
2. The execution time of case 2.pℓ1 was increased by .6 ms., when explicit aligned and unaligned attributes were not added in converting from case 2.epl

Conclusions

Several observations can be drawn from the above results:

1. For a given procedure the PL/I compiler (compared to the EPL compiler) will reduce the amount of text produced to about 2/3; linkage will essentially remain the same size unless specifications or organization of the procedure are changed.
2. Changes in execution time are more variable: Case 1.epl (1967 EPL) to Case 1.pℓ1 improvement was a factor of 3; Case 2.epl (REPL coded) to Case 2.pℓ1 produced negligible changes

in execution time. A reasonable conclusion is that the PL/I version of a procedure will never be significantly slower than and will usually be faster than the EPL version.

3. Although straightforward brute-force recompilation of the Multics system is conceivable, there is careful editing and review required to check for subtle issues such as equivalence of array packing, the effective use of label arrays in computed-go-to statements, etc. Also at first, there will be residual compiler bugs and a lack of familiarity among programmers with the new compiler properties. One can conclude that the system recompilation should be done area-by-area until confidence is built up.
4. Because the effect of changes in coding tactics or general organization can have larger impact than changing compilers, it would appear adviseable to accompany general system recompilation with an editorial review of each module as it is converted.
5. Finally from the examples which have been given it is obvious that the pℓ1 compiler has the potential for space and time improvements which could easily lead to a factor of 2 in system performance.