WIS-CS-72-160
COMPUTER SCIENCES DEPARTMENT
The University of Wisconsin
1210 West Dayton Street
Madison, Wisconsin 53706

THE EVOLUTION OF A COMPILER: PUFFT 1964-1972

by

E. J. Desautels and M. D. Shapiro*

Technical Report #160

October 1972

Received September 8, 1972

*Purdue University Computing Center, Lafayette, Indiana
Presently with National Cash Register, San Diego, California

THE EVOLUTION OF A COMPILER:

PUFFT 1964-1972

ABSTRACT

The evolution of a compiler: PUFFT 1964-1972. The Purdue University Fast Fortran Translator PUFFT went into operation on a 7090 in 1964. It has since then been used in many environments and subject to many extensions and improvements. These changes and improvements are described, along with the reasons for making them. The use of PUFFT demonstrates that second-generation equipment may find its useful life prolonged, particularly at educational institutions seeking low cost processing of large numbers of jobs.

THE EVOLUTION OF A COMPILER: PUFFT 1964-1972

E. J. Desautels and M. D. Shapiro

Introduction

The PUFFT compiler [14] is a core-resident, one-pass, compile-to-core Fortran compiler. It was developed for an IBM 7090 tape system when it became apparent that the system's large-job oriented Fortran compiler [10] was not the appropriate tool to cope with the ever increasing number of small jobs being submitted. The same Fortran compiler was characterized (as are many other currently used compilers) by a lack of error detecting facilities and other aids for the non-professional programmer.

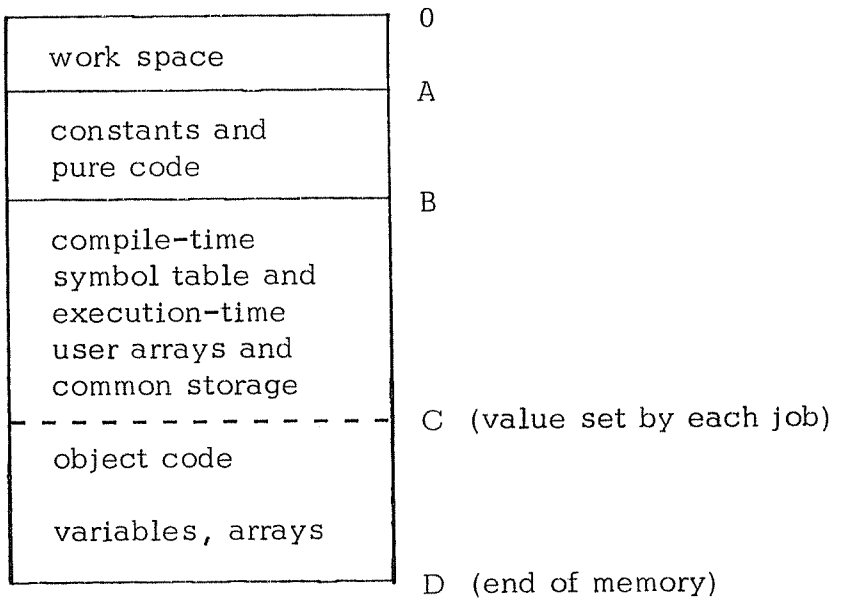
Thus construction of a new compiler was undertaken with the following goals:

1. fast compiling
2. full Fortran IV language capability

This suggested the following initial implementation guidelines

1. minimize the job to job transition time
2. eliminate the use of an assembler
3. eliminate the need for a relocating loader
4. minimize the use of magnetic tape while compiling

This led to having the compiler core-resident, with its own capability for handling a batch stream. This also led to adapting the compile-to-core approach, coupled with a single pass over the source code. No relocatable code is generated, nor is any accepted. The 2^{15} word memory was allocated as follows:



The workspace accounts for approximately 1700 words; constants and pure (invariant) code use another 18,800 words and the balance of some 12,300 words can be allocated with 2^{n+1} words for the space which at compile-time is used for the symbol table, and for user arrays and common storage at execution-time. The value of n may vary from 1 to 6 and is chosen by each user job (default $n = 5$).

The pure code includes the standard set of real, complex and double-precision routines. Since the entire program (written in FAP

assembly language) is a single program unit, all constants may be shared.

Extensions and Improvements

The PUFFT compiler has also been used on 7094's, on direct-coupled 7044-7094 systems, as a stand-alone system, and as a component of the IBSYS 7094 operating system. The PUFFT compiler has been used as the major component of a remote access Fortran system with interactive capabilities [4]. This paper describes the extensions of and improvements to the original (1964) PUFFT, as implemented at Purdue and the reasons for making these changes. Other relevant modifications are discussed.

The primary changes which have been made which are described in this paper come under the following categories:

1) source language library
2) subscript checking
3) compile time format checking
4) execution in spite of compile time errors
5) execution time parameter checking

The above features are intended to provide the maximum useful service and debugging information on each run; none of the above enhance either compilation speed or execution speed of user programs.

Source Language Library

PUFFT maintains a core-resident library of standard routines such as SQRT or EXP. In order to provide a larger number of routines in a library without dedicating any more main storage for library routines, one is forced to use auxiliary storage. When the decision to provide a non-resident library was made, several possibilities were considered:

(i) fully relocatable code, as produced by standard assemblers and compilers

(ii) simply-relocatable code

(iii) Fortran source decks.

Possibility (i) was attractive in that the entire IBM 7090 Fortran library could be used. However, it would require use of a "sophisticated" loader of the IBLDR [9] type with the attendant excessive overhead. The use of simply-relocated code in (ii) would reduce the loader overhead but would still require additional facilities and time to perform the relocation and linking.

The remaining possibility (iii) was implemented. The library is generated by reading source decks and compressing them into a library file, while generating an entry point directory. When programs are compiled, this directory is searched if any user-referenced routines remain undefined when the user-provided source input is exhausted.

Regardless of the representation of the non-resident library routines, a directory-search routine is required. When the auxiliary library is stored in source form, the only extra code needed is a statement scan mode test for compressed or standard input.

Library Organization

The non-resident library was first implemented when the transition was made from a tape-7090 to a 1301 disk-and-tape 7094, with the library and its directory kept on the disk. It was organized to permit direct (random) access to the needed routine. The disk was subsequently retired and replaced by a CDC disk pack drive shared with a CDC 6500; the same library organization was used on the disk pack, with the library maintenance handled by the 6500. At the present time, a 7094 is used as a dedicated PUFFT machine. It uses 4 magnetic tapes (card reading, output printing performed off-line) and the non-resident library now finds itself on tape. The library tape contains the library directory as its first record, and the compressed Fortran statements are stored following the directory. Since Fortran disallows mutual recursion, all external references to the library can be satisfied in at most a single pass over the library tape.

Library Use

No special action is required to access a non-resident library

routine. Routines are carefully screened before inclusion in the library, as these routines are privileged and the 7094 is vulnerable due to the lack of memory protection hardware and the lack of privileged instructions. Besides the usual mathematical and statistical and line-printer plotting routines, several special packages (sets of related routines) are included. For instance, the MORON system [13], which simulates a simple hypothetical computer with its assembler, and the MIMIC system [15], which essentially simulates an analogue computer for use in solving differential equations are provided as they are used extensively in a number of courses. Special packages of general interest are included, such as the PUSHUP package [17] which provides the Fortran user with a string handling capability similar to those of PL/1 and SNOBOL.

As special needs arise, it sometimes appears that a change in the compiler might be required. For instance there was a requirement for a very simple output capability for beginning calculus students who were learning a minimal Fortran subset. It was felt that FORMAT's and NAMELIST output were too involved. Rather than modify the compiler to provide a format-free list directed capability, a new routine was placed in the non-resident library, and the students were taught that output of the form ABC = 1.5 could be obtained by using a statement CALL OUT ('ABC=', ABC). The OUT routine determines dynamically whether an I, E, or F format is appropriate.

All the non-resident library routines are standard Fortran. They may however invoke a routine which user code cannot invoke. This routine, called APPLY$ is given the address of an array, and a list of n arguments. It gives control to the instructions held in this array, setting an index register to produce the effect of a subroutine call with an n-argument parameter list. This allows carefully screened library routines to directly execute machine language code in the interest of efficiency. Should a user reference a library routine which in turn calls on APPLY$ (or on any other routine whose name ends in $), the name APPLY$ (or any name ending with $) will not be printed in the entry point table. This makes it possible to provide packages of many routines, with most of the routine names being suppressed from the external reference printout to avoid intimidating the novice.

Subscript Checking

Subscript checking is provided as a default option. Each array reference (fetch or store) involving non-constant subscripts leads to generation of a subscript checking instruction word containing a call on the subscript checking routine and a maximum displacement from the array origin value as a parameter. This can be handled with the 7094 store and trap (STR) instruction. When a program goes into execution, the effective displacement for an array reference is placed in an index register.

The subscript checking routine compares the actual to the maximum permissible displacement. An out-of-range condition leads to a message which identifies the statement involved and terminates the job with the incorrect value in question displayed.

Such a technique does not check that each subscript is within range, as is the case for instance in DITRAN [11]. This was a compromise chosen to provide a good tradeoff for system protection and user diagnostics without excessive cost in code for checking, subscript parameter storage and execution time spent performing the check.

Compile Time Input/Output Format Checking

One of the most frequent errors, especially for beginning Fortran programmers, occurs in writing FORMAT statements [11]. When the format is stored (as it must be) for interpretation at execution time, a syntax check is performed. The frequent errors such as improper parenthesization and inaccurate counts for Hollerith fields are detected. In the event of a format error, a warning message is generated. A job is not aborted during compilation when a bad format is detected. Such a format causes a job to be terminated only if an attempt is made to use the erroneous format at execution time.

Execution in Spite of Compile Time Errors

Wherever feasible, each fatal error message (intended to inhibit

execution of a job) has been replaced by a diagnostic message of lesser severity. Thus when a program has been compiled and it is established that the user has called subroutine ABCD, that the user has not provided a definition for ABCD, and that the libraries do not contain such a routine, a message is printed saying ABCD is undefined. Should ABCD be called during execution, then its absence will lead to the job being terminated with an appropriate message.

Execution Time Parameter Checking

According to the Fortran specification [1], the programmer is responsible for ensuring that the proper number and type of parameters are passed to subroutines. Consequently one finds programs that execute data and constants, that change the value of constants, that modify object code and so on. It is sometimes difficult for users to discover these types of errors.

One standard technique for execution time parameter checking involves passing descriptors along with the parameter addresses. The structure of PUFFT lends itself to a more compact approach using address ranges to provide similar information. Consider the following figure:

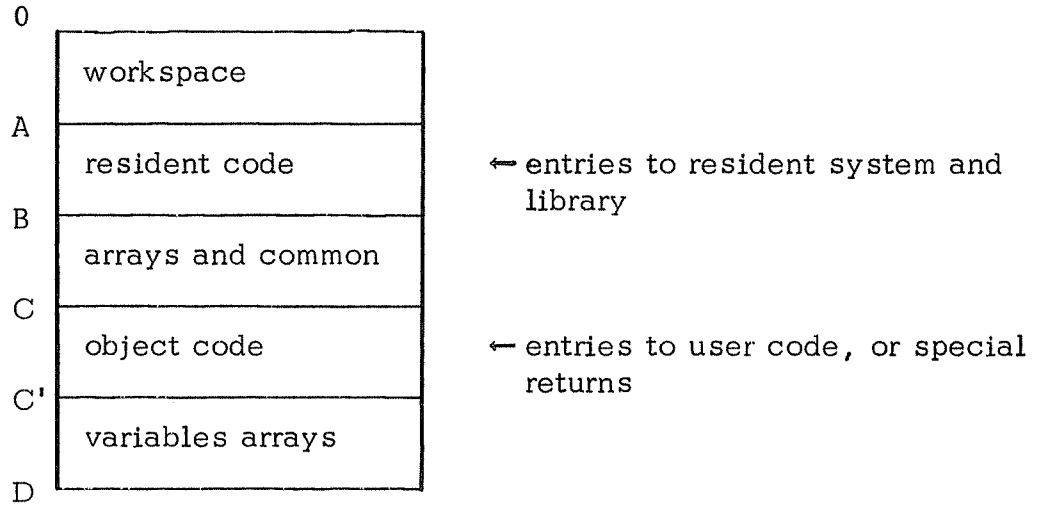


Figure 1: Using addresses for execution time parameter checking

A, B, fixed, with the resident compiler and system occupying memory locations A to B

C, set at run time, indicating the beginning of the object code for a user's program. The space from A to B is used for array and COMMON variables.

C', set at run time, indicating the end of the object code.

D, fixed, indicating the end of useable memory. The space from C' to D is used for variables and arrays.

If X is an argument, as in CALL R(X) then X < B implies X must be the entry to a core-resident library of system routine. If C < X < C', then X must be an entry or a return point defined within the user's source code. In routine R, a statement such as Q = X(7) generates a

call on a system routine which verifies that the parameter address lies in one of the two acceptable ranges. This form of checking is necessary but it is not sufficient, in that a call such as CALL R($1) meets the test. However, such a call does not have a destructive effect; it merely causes skipping of code.

Special returns can be checked in two ways. A special return address X must meet the condition $C < X < C'$ and the condition that the instruction at address $X - 1$ must be of the special form "LOAD INDEX REG 5, line number" which is only generated as the first instruction of every executable statement. Only the first check is applied in the batch PUFFT system; both are applied in the time-shared PTTS version [4].

Advantage is taken of the special form of the parameter list in the calling sequence to determine that the proper number of arguments have been supplied. If too many are provided, then the compiled return address points to an instruction (i.e. an item in the parameter list) which is trapped by the system at the next timer interrupt (i.e. within one second).

If too few arguments are provided, checking in the called routine traps the error by looking at the form of parameters.

Special Adaptation 1: BTL-PUFFT

Although PUFFT was designed primarily for use in the "student job" situation of many short jobs, it also found its way into industry. One variation was implemented for the IBM 7094s at Bell Telephone Laboratories for use primarily as a debugging compiler. BTL-PUFFT was designed to run under the BESYS7 operating system, a core-resident monitor using 8,256 of the machine's 32,768 words. In order to run in this space, the system was split into three modules: common routines, compile-time routines, and execute-time routines. The execute-time routines and blank common overlayed the compile-time routines at execution time. This technique provides more space for program and array storage at the expense of requiring an overlay and reloading portions of the system for each job.

While the entire system is no longer core-resident, the object program is still written in core and a loading-and-linking phase is not needed. When installed, BTL-PUFFT was the fastest Fortran available on the machines. When Bell Labs converted to IBM 360/65s with 7094 emulation capability, PUFFT was made part of the system and was intially the fastest Fortran on it, until 360 WATFOR was installed.

Special Adaptation 2: PUFFT Time-Sharing System PTSS

In the period September 1968 to August 1970, a multiple-access system based upon a special version of PUFFT was in operation at Purdue. This system described in [4] supported a maximum of 28 teletype users in the preparation, debugging and execution of Fortran jobs. A uer's Fortran job could be swapped out if it exhausted its time slice or if it was requesting "immediate" input for an interactive job.

The organization of the system is as shown previously in figure 1, with the addition of some 900 words of resident code to handle the terminal interrupts, and a block of 450 words used as a terminal command-language code overlay area. Swapping of a job involves copying all but the resident and command-language code onto the disk; this copying of approximately 12,000 words is accomplished in 12/15 seconds. The resident compiler code is not swapped out because it is pure (read-only). Its integrity is checked at the beginning of each compilation by use of a checksum routine, to compensate for the lack of memory hardware protect.

This on-line system was retired in September 1970 when the Purdue 7094 was coupled to the CDC 6500 to form a new on-line system called PROCSY [12].

PTSS supported user private libraries by maintaining directories and the user text on the disk, with the usual forms of password protection. That is, a user could designate an item as being in one of four classes: public (unrestricted), public read-only, public execute-only or private.

In modifying PUFFT for on-line use in PTSS, the transition from card-reader input and line-printer output to low-speed, narrower line width, one-character-at-a-time Teletype terminals especially influenced modifications to the diagnostic message handling. The compiler was modified so that all error messages would include a source program line number designation, based on the line numbers provided by the system when the program is initially keyed in, or those maintained in the user's private on-line library.

Each executable statement is compiled into a sequence of instructions beginning with the instruction "LOAD INDEX REG 5, line number"; index register 5 always contains the line number of the executable statement currently being executed. The compiler generates instructions for subroutine entry which cause the symbolic name of the calling routine to be saved while the symbolic name of the called routine is placed in a special location indicating which routine currently has control. Complementary instructions are compiled for subroutine returns to restore the name of the calling routine. Consequently

the system can provide the symbolic routine name and line number in the event of an error. This differs slightly from error walkback facilities provided by compilers such as the UNIVAC Fortran V [19] in that these do not obtain symbolic subprogram names until after an error occurs, and in the event of an unanticipated error, not at all.

A relatively simple change was made to the basic input routine used by the compiler and by all user-referenced input routines which was very significant for on-line users. The basic system input routine was modified to recognize commands of the form "$LINK file-name deck-name". This caused the on-line user library directory to be searched for a "file-name" match, and for a "deck-name" match within the specified file-name directory. If both searches were successful, then the text which is stored under the given name is processed by the basic input routine. The text can be part of, or all of a Fortran job; it may consist exclusively of data. It may also contain more "$LINK file-name deck-name" commands. It thus becomes possible for the on-line user to easily combine text as the occasion arises, and to treat it as input to the compiler, or as data for one of his jobs, or as a combination of both.

Related Developments

Shortly after PUFFT went into operation, the WATFOR compiler for 7040-44 Fortran was announced [16]. A similar version has been

released for the IBM 360. The WATFOR compiler places emphasis on runtime diagnostics and achieves this by retaining the symbol table at run time.

The CORC compiler [8] and more recently the CUPL compiler [3] were developed at Cornell to provide a student oriented language and a very forgiving system.

The design of the IITRAN programming language [6] takes into consideration many of the error diagnostic capabilities discussed here. Not being constrained to compatability with Fortran, many of the error-inducing Fortran restrictions are eliminated, and more powerful error-avoiding constructs such as array operations are included.

The DITRAN compiler for Basic Fortran developed at the University of Wisconsin [11] originally for the CDC 3600 and now for the UNIVAC 1108 [7] is intended to provide the utmost in diagnostic information.

To a certain extent, PUFFT was a partial response to the inadequacy of the software on the IBM 7090/94. Fortran IV, version i ($i \leq 12$) compiled optimised assembly code, assembled and loaded it with a minimum overhead of 20 seconds. Version $i = 13$ was released soon after PUFFT was in use and it was an improvement over $i = 12$ as it avoided use of the assembler. Yet it still could not compete with PUFFT on the smaller jobs (those with 1000 or fewer statements).

Furthermore it provided no additional features, such as subscript checking.

A New Look with Old Gear

As years passed and the Purdue University Computing Center acquired newer equipment, the 7094 was made a part of a multi-terminal system. While still available for occasional runs, batch PUFFT service became relatively infrequent. In late 1969, in order to improve overall service, Purdue was able to obtain a used 7094 which has become a dedicated PUFFT machine [17].

The major software effort in bringing up the system was conversion of the source program library from disk units to tape, as the new machine has no disk available. Despite complaints from some that Purdue would become known for being a dumping ground for old computers, prior experience with the PUFFT system software allowed the new service to be brought into operation two days ahead of its published schedule date (a radical departure).

The system is now run on a two-hour cycle seven days a week, with a total weekly PUFFT load in excess of 9000 jobs. While such operation would not be economically feasible at commercial rental rates, the availability of used equipment at relatively low prices makes it reasonable. PUFFT has been given another lease on life.

As an indication of student response, the ACM student chapter suggestion box, which usually receives complaints, started getting a few thank-you notes.

Changes in Operations and Maintenance

In order to keep up with the volume of PUFFT jobs without complicating the procedures for handling the CDC 6500 jobs, all jobs submitted at Purdue enter the same queue. At a site serviced by a 1401, user jobs are separated (by the 1401) onto two magnetic tapes for subsequent processing by the 6500 or the 7094 as indicated. At a site serviced by a 6500 batch card reader, the PUFFT jobs are queued onto disk, to be dumped onto tape at intervals by an operator command. Output tapes from PUFFT can be listed by either 1401's or the 6500.

In an attempt to keep the dedicated 7094 dedicated to PUFFT, support programs which run on the 6500 have been developed. They are used to update the PUFFT library tape, to assemble the PUFFT compiler (in its original 7094 FAP source code) and generate self-loading system tapes for running PUFFT on the 7094.

In the dedicated PUFFT system operation, operating procedures for the 7094 have been simplified to the extent that all the necessary instructions (including error recovery after a system crash) are provided on a single sheet. The pertinent console switches and lights

are clearly labelled with regard to their role in the operation of PUFFT. The most difficult training task in initiating a new operator in the running of PUFFT jobs involves the tape mounting procedures.

Work in Progress

Investigatory work is underway at Purdue to examine the effectiveness of using the 7094 running PUFFT as a directly coupled peripheral machine to a larger CDC 6500 system. In the same vein, the 1401's will be coupled to the 6500. These couplings are in addition to that of the current 7094-6500 coupling already in operation.

A completely different effort was undertaken at the University of Wisconsin, where the feasibility of converting the PUFFT compiler for use on an 1108 as part of a study of reprogramming techniques was examined. [5].

It should be noted that as of Sept.,1972 dedicated PUFFT service on the 7094 was still being provided, as it was more cost-effective than any other core-resident Fortran compiler in the CDC 6500. Over 29,000 PUFFT jobs were processed in May, 1972. [20]

Conclusions

At this time, PUFFT has gone full circle in terms of operating systems in which it has been embedded. It began with a tape system (IBSYS, IBM 1301), which in turn continued with the 1301 disk replaced

by a CDC disk pack drive, and now we have returned to a tape system completely dedicated to PUFFT. It would appear that a batch processing or stream processing facility can be well supported by second generation hardware with sequential access devices, for certain job classes.

With the advent of third generation systems, the cost/performance ratio of second generation equipment has decreased dramatically. To the extent that an organization is capable of doing its own hardware maintenance, it should seriously consider using second-generation equipment for certain classes of service.

Acknowledgements

The authors are indebted to the originators of PUFFT, Dr. Saul Rosen, Joel Donnelly and Robert Spurgeon. The subscript checking was originated by Donnelly. The source language disk library was implemented by L. Symes; the compile-time format checking by R. Paddock. The design of the 7094 terminal multiplexor, originally used with PTSS and now used with PROCSY, and the design of the 7094-6500 interface, are due to John Steele, who also contributed in many ways to the software development. The other features were developed principally by the present authors, with the BTL-PUFFT and tape library systems done entirely by M. Shapiro. The continuing interest and support of Professor Rosen has been most helpful.

Addendum

The recently released FORGO compiler, which was developed for the Datacraft computer by the University of Wisconsin Engineering Computing Laboratory, is an excellent recent example of a diagnostic Fortran compiler [21]. It can produce such a variety of English language diagnostics at compile or run time, that four pages are needed to list all of them.

REFERENCES

1. American Standards Association, American Standard Fortran. American Standards Association, New York, 1966.

2. Conway, R. W., and Maxwell, W. L., CORC: The Cornell Computing Language. Comm. ACM 6, 6 (June 1963) 317-321.

3. Conway, R. W., and Morgan, H. L., Tele-CUPL: A telephone time sharing system, Comm. ACM 10, 9 (Sept. 1967) 538-542.

4. Desautels, E. J., The PUFFT time sharing system - design, implementation and performance. Ph.D. thesis, Purdue, May 1969. University Microfilms, Ann Arbor, Mich.

5. Desautels, E. J., Analysis of the PUFFT compiler, in preparation.

6. Dewar, R. B., Hochsprung, R. R., and Worley, W. S., The IITRAN programming language. Comm. ACM 12, 10 (Oct. 1969) 569-575.

7. Doig, B. DITRAN Implementation. University of Wisconsin Computing Center, Technical Report No. 17, August, 1970.

8. Freeman, D. N., Error correction in CORC. Proc. AFIPS 1964 FJCC Vol. 26, 15-34.

9. IBM Corp., 7090/7094 IBSYS operating system, IBJOB processor, form C28-6275-3, Poughkeepsie, New York, 1965.

10. IBM Corp., IBM 7090/7094 IBSYS Operating System, Version 13 Fortran IV Language. IBM Systems Reference Library form C28-6390-0.

11. Moulton, P. G. and Muller, M. E., DITRAN - A compiler emphasizing diagnostics. Comm ACM 10, 1 (Jan. 1967) 45-52.

12. Purdue University Computing Center, PROCSY, Purdue remote operation computing system, September 1969.

13. Rice, J. K. and Rice, J. R., Introduction to Computer Science, Holt, Rinehart and Winston, New York, 1969.

14. Rosen, S., Spurgeon, R. A., and Donnelly, J. K., PUFFT - The Purdue University fast FORTRAN translator. Comm. ACM 8, 11 (Nov. 1965) 661-666.

15. Sanson, F. J. and Peterson, M. E., MIMIC Programming Manual, Technical Report SEG-TR-67-31, Wright Patterson AFB, Ohio, July 1967.

16. Shantz et. al. WATFOR - The University of Waterloo Fortran IV Compiler. Comm. ACM 10, 1 (Jan. 1967) 41-44.

17. Shapiro, M. D., PUFFT User's Guide. Purdue University Computing Center, February 1971.

18. Shapiro, M. D., An introduction to character string operations using Fortran IV and the Purdue University String Handling Utility Package (PUSHUP). Purdue University Computing Center, December 1970.

19. Sperry Rand Corp., UNIVAC 1108 multi-processor system Fortran V, UP-4060 Rev. 1, 1969.

20. Purdue, Purdue University Computing Center Newsletter, Vol. VI, No. 5, June-July 1972.

21. Datacraft Corporation, FORGO compiler general specification and user's manual, Datacraft Corp., Fort Lauderdale, Fla., July, 1972.

BIBLIOGRAPHIC DATA SHEET	1. Report No. WIS-CS-72-160	2.	3. Recipient's Accession No.
4. Title and Subtitle The Evolution of a Compiler: PUFFT 1964-1972			5. Report Date October 1972 6.
7. Author(s) E. J. Desautels and M. D. Shapiro			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Computer Sciences Department The University of Wisconsin 1210 West Dayton Street Madison, Wisconsin 53706			10. Project/Task/Work Unit No. 11. Contract/Grant No.
12. Sponsoring Organization Name and Address			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

The evoluation of a compiler: PUFFT 1964-1972. The Purdue University Fast Fortran Translator PUFFT went into operation on a 7090 in 1964. It has since then been used in many environments and subject to many extensions and improvements are described, along with the reasons for making them. The use of PUFFT demonstrates that second-generation equipment may find its useful life prolonged, particularly at educational institutions seeking low cost processing of large numbers of jobs.

17. Key Words and Document Analysis. 17a. Descriptors

Compilers, second-generation, diagnostics, Fortran, time-sharing, operating systems

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Available to public.	UNCLASSIFIED	24
	20. Security Class (This Page) UNCLASSIFIED	22. Price