Symbolic Computations on a Personal Computer

S.N. Baranoff
Leningrad Institute for Informatics and Automation
USSR Academy of Sciences
14 liniya 39, Leningrad, 199178, U.S.S.R.

Abstract

This paper describes an approach that allowed a rather developed system for symbolic computations, SAC-2 [1], to be moved to an IBM PC XT, with the addition of natural dialog features. The size of the compiled system diminished from 700 to 64K in size while the run-time was only doubled.

Introduction

The source language of the SAC-2 system is ALDES (ALgorithmic DEScription); it is an ALGOL-like, typeless language. A program unit in ALDES is named an algorithm and corresponds to a procedure or a function. An algorithm consists of declarations, assignments to variables, conditional statements, loops, and calls to other algorithms. The SAC-2 system contains over 500 algorithms as source text in ALDES and a compiler that translates ALDES source code into standard FORTRAN. This gives wide transportability of the SAC-2 system among computers with appropriate FORTRAN compilers and core storage. I used this system on an IBM/360 compatible with the OS/360 operating system and found that the object code resulting from the two-step compilation was too large. The ALDES compiler, which itself was written in ALDES and then bootstrapped via FORTRAN, compiles to about 400K of machine code and requires about 500K of memory for its run.

In order to cut down the memory requirements so as to run on an IBM PC XT we used Forth instead of FORTRAN as the intermediate language. When the ALDES compiler was rewritten in Forth its size diminished to 20K (the size of the compiler plus the supporting Forth nucleus was 35K). This substantial decrease was due both to the Forth language itself and to quite another technique of compilation (70 screens of Forth code) as well. Source texts of the SAC-2 algorithms could be submitted unchanged to the new compiler.

A Forth source-code text is produced as the output of the compiler, the ALDES constructs being transformed into special patterns of Forth words. In order to receive executable binary code, this text must be processed by a Forth interpreter in the context of definitions of those Forth words. Access to this intermediate representation as ordinary Forth source-text allows us to have specialized generating passes of the compiler; those pay special attention to run-time optimizing, debugging facilities, gathering statistical information, etc.

A standard version of the generating pass (the loader) consists of a set of Forth word definitions (about 50 screens of Forth text) and produces the threaded code that is generally accepted for internal representation of Forth programs. In order to raise the run-time speed and enlarge the available address space it uses direct token-threaded code, the run-time address interpreter having been designed accordingly. The token representing a byte-address consists of that address divided by two; this enlarges the address space to 128K (the SAC-2 system occupies

64K in this representation instead of the 700K of machine code in the original FORTRAN version).

A dialog user interface is supported by the basic Forth system with additional words for compiling, loading, and running ALDES programs. The memory requirements of the system may be minimized by using a simple overlay technique (the standard size being 64K). When started, the system allocates additional memory for the precompiled SAC-2 algorithms (maximum 128K, actually 64 to 66K) and loads that binary code. The corresponding tables, which contain check-up information about loaded algorithms, are loaded into the Forth address space (within its 64K). After that the user may compile his own program in the given context. In contrast to the original ALDES compiler the system supports strong parameter checking for calls to algorithms (this let one error and several inaccuracies be found in the original texts).

It is possible to save memory for binary code. That increases compile time and is an implementation of an AUTOLINK option of a linkage editor. The user's program is compiled in an empty context and then all of the external references are resolved by loading (i.e. Forth interpreting) the appropriate SAC-2 precompiled algorithms.

In order to run a successfully compiled program, memory should be allocated for its internal data. A stacking mechanism is used for allocating local variables when a call to an algorithm occurs. This gives a considerable gain compared with the static memory allocation of FORTRAN. For global data a heap is used with a simple garbage collection scheme. If the heap is too large to reside in main storage a variant with virtual memory supported by mass storage may be used, but that will inevitably slow down the run time. The program may communicate with its environment through supporting modules that form a specialized extension of the basic Forth system. After debugging, a new algorithm may be added to the SAC-2 system as its extension.

Run-time debugging of the original SAC-2 system was one of its main faults, as it depended greatly on the particular FORTRAN compiler and operating system. Thus, the built-in debugging processes of the ALDES compiler required special planning at compile-time. The corresponding method for the approach described here forms a separate subsystem, also written in Forth. It uses knowledge of the threaded code structure and requires no special compiler options. In response to the dialog user's requests, it may switch on and off the tracing of algorithm calls and can display data snaps in terms of the source program.

This system, named MINISAC, was implemented on an ES Ryad computer (an IBM/360 compatible) using the FORTH-ES system, [2] which I also developed. This allowed a detailed comparison of this "interpretive" approach with the original SAC-2 version on the same ES-1052 computer under OS/360 6.1 with the FORTRAN H compiler. As an example, evaluating and factoring of a variant of the van der Mond determinant for a 6×6 matrix, with elements being polynomials in three variables, gave the following numbers:

System	Memory used	CPU time consumed
SAC-2	480K	3 min 17 sec
MINISAC	258K	5 min 56 sec

Now the MINISAC system has been moved to an IBM PC XT running under MS-DOS and an SM-4 (a PDP-11 compatible) running under Unix using Forth systems also developed at LIIAN. Experiments with these systems are in progress.

References

[1] Collins, G. "The SAC-2 Computer Algebra System." *Lecture Notes in Computer Science*. 1986. v. 204, pp. 34-35.

[2] Baranoff, S. N. *Sistema Forth-ES*. LIIAN, Leningrad. 1986. 46 pp. In Russian.