Abstracts of the Tenth Asilomar FORML Conference

November 25 – 27, 1988
Asilomar Conference Center
Pacific Grove, California

Music Reader: A Program Converting Music Score to Machine Readable/Playable Code

Pei-Tao Ting
Beacon Christian School, San Carlos, CA
C. H. Ting
Offete Enterprises, Inc., San Mateo, CA

It is rather tedious to program a computer organ to play Bach's organ music because every note has to be entered manually. The process is very time-consuming and error-prone.

A hand-held scanner HS3000, made by DFI, was installed in our XT computer and used to scan music scores. The scanned image is stored in the Dr. Halo's '.CUT' format on a hard disk. The scanned image is then analyzed and valid music notes are extracted from the binary image. The pitches of the notes are determined from their locations in the scanned image. Groups of notes are assigned to proper voice channels and are written out to an output file as a sequence of chords.

The chords written to the output file are in a form suitable for playing on the computer organ through a six channel tone generator. The conversion process is quite efficient. The conversion rate currently is about two chords per second, more than an order of magnitude faster than manual entry.

The conversion program is not yet perfect. Notes are frequently mis-assigned to neighboring pitch. Channel assignments depend upon the position of notes, and can be confused when many notes ocurred on the same staff and when voices cross each other.

This music reading program, even at its present primitive state, greatly enhanced our productivity in coding Bach's organ music. Most of the pieces the second author played at the Real Time Programming Convention were first entered using the scanner.

Techniques, results, problems, and possible improvements in reading music scores automatically will be discussed.

Tree Structure

Glen B. Haydon
Haydon Enterprises, Box 429 Route 2, La Honda, CA 94020

Tree structures are common in a variety of studies: Functional programming, natural languages, artificial languages, brain function, hypertext, Forth and the WISC CPU/32 computer architecture. Conventional computer architectures provide a poor platform for the execution of tree structures.

MENU: A Forth Menu Compiler

Kenneth B. Butterfield
Los Alamos National Laboratory, P.O. Box 1663, MS J562, Los Alamos, NM 87545

Menu-driven command interpreters are an effective means of controlling portable instrumentation where input may be limited to a hex keypad and output to a few lines on a liquid crystal display. The MENU compiler uses the name fields from the words comprising the menu options as prompt strings. It produces compact program code, thus conserving memory that is often limited in ROM-based systems. Input for menu function selection is vectored to allow switching between input sources. The host system consists of the eight-bit Motorola M68HC11 processor and Max-FORTH (FORTH-83).

Incremental Recompiling through Secure Forward References

Mike Elola

By making the physical layouts of data structures alike, we make reuse of operations more of a possibility. See the first installment of "Designing Data Structures" (ELO88) for a detailed explanation.

A General List Processing INTERPRET for F83

Jeff Elliot

EVEREX, 245 N. Main Sebastopol, CA 95472

This paper discusses an alternative definition of INTERPRET and describes a set of generalized list processing definitions. Advantages to this method include reduced dependence on instream syntax and stack arguments, improved readability, and better error handling. This technique could be useful in building any general object-oriented FORTH application.

Compatability for Floating Forth or Any Forth is Good Enough

Robert L. Smith

Many people have complained about various incompatibilities among different Forth systems. I am unaware of any significant software packages which can load and run on a variety of Forth systems without significant alterations. Partly as an experiment, I have implemented a high level Floating Point package, called FLOAT4TH.BLK, which can load and run on a variety of 16 bit Forth systems, including polyFORTH, FIG-Forth, 79-Standard, and 83-Standard systems. For the systems I have been able to try, no changes in the source code have been necessary.

Multitasking: The Right Way

John R. Hayes

John Hopkins University/Applied Physics Laboratory

Traditional Forth multitasking requires each process (task) to voluntarily release control of the processor at reasonable intervals. In contrast, a preemptive multitasking system can switch between processes whenever necessary. The system is responsible for deciding which process runs. The disadvantage of this approach is that the occurrence of real-world (real-time) events such as the arrival of interrupts effects the order in which processes are run and suspended. Consequently, the programmer must use interprocess synchronization operations when the execution order matters. This paper describes the use of `wait` and `signal` synchronization primitives in a Forth preemptive multitasking environment. An extended example showing how `wait` and `signal` perform mutual exclusion, event notification, and resource allocation is presented.

Dynamic Memory Allocation

Klaus Schleisiek-Kern

DELTA t GmBH, Roter Hahn 42, D-2000 Hamburg 72, FRGermany

Implementing a time-sliced multitasker in Forth reveals the inadequacy of the BLOCK concept – the validity of a block address cannot be guaranteed any longer. The words `ALLOCATE` and `FREE` are defined to manage main memory which can be explicitly used to store mass storage buffers (records), data- and return-stack, string-stack and string-variables. As it turns out, an optimal algorithm for dynamic memory allocation is more compact than a clean implementation of an LRU-scheduled block buffer scheme.

Foolish Old Man Moving a Mountain—Documenting F-PC System

C. H. Ting

Offete Enterprises, Inc., San Mateo, CA

F-PC is a version of Forth for IBM-PC/XT/AT computers, derived from F83/8086. It is implemented by Tom Zimmer and Robert Smith with substantial extensions, modifications and enhancements so that very large application can be programmed and run at high speed in these readily available and economical platforms.

Major features include:

- Use handles instead of file control blocks
- Use sequential text files instead of block files
- Integral WordStar-like text editor with on-line help
- 20,000 lines per minute compiling speed with 10 Mhz AT
- Separated heads

- 64 Kbyte code segment
- 128 Kbyte or more for colon defintions
- Access to 1 Mbyte of memory space
- Automatic configuration to CGA, EGA, VGA, Monochrome cards
- Turnkey and metacompilation of applications
- Multitasker
- Enhanced debugger
- Hardware and software floating point packages
- Tons of user contributed applications

Documenting such a huge system is a challenge. So far we have generated two documents to support this system: *F-PC Users Manual* and *F-PC Technical Reference Manual.* The users manual provides top level description of features and functions included in the system, and some tutorial material to help new users. The reference manual documents the internal mechanics and kernel words for advanced users and system programmers. More advanced features like 8088 assembler, multitasker, floating point packages, graphics, etc., are left for future volumes.

This paper discusses styles of presentation and problems in documenting systems of this magnitude.

F128, Experiments On An 128 Bit Forth

C. H. Ting

Offete Enterprises, Inc., San Mateo, CA

We are at the threshold of migrating from 16 bit Forth to 32 bit Forth. A few years later, we may have to consider 64 bit Forth. Is there an end? This paper anticipates that we will have to deal with even larger word width.

The 8088 family of CPU's provides us a very convenient platform to explore Forth systems which will use 128 bit words, if we insist on using the segment pointers to address memory. The memory is thus organised neatly in 16 byte pages or 128 bit words. We shall consider various interesting consequences in building a Forth system in which data must be represented in 128 bit chunks.

An 128 bit word can be used very effectively to store the following information:

- Executable code, up to 16 bytes or multiples of 16 byte pages
- Headers
- List of pointers to executable code (colon definitions)
- Data structures for integers, up to 16 bytes
- Data structures for floating point numbers, up to 16 bytes
- String data, multiples of 16 byte pages

Initial experimental results on hosting a Forth kernel on this type of architecture will be presented in this paper. Several interesting ideas on implementing a Virtual Forth Machine will also be discussed.

A Self Hosted Embedded Microprocessor

Martin E. Fraeman

John R. Hayes

John Hopkins University/Applied Physic Laboratory

Lee W. Nearhoof

NASA Goddard Space Flight Center

A prototype embedded computer based on the SC32 32 bit Forth microprocessor is being developed. It is electrically identical to a potential flight computer and will be used to demonstrate the advantages of developing flight software directly on target hardware. The prototype includes a magnetic bubble mass memory and provisions for power cycling to conserve battery power. System software supports interactive program development and testing on the target flight computer as well as traditional real-time control. The combination of a powerful target processor, mass storage, and suitable system software will lower the costs of implementating computer based instrument control systems while increasing the flexibilty of those instruments.

Stack Caching in the SC32 Forth Processor

John R. Hayes
Susan C. Lee
John Hopkins University/Applied Physics Laboratory

Support for Forth's parameter and return stacks is critical in the design of a Forth processor. We have implemented two stack caches in the SC32 Forth processor. This paper reviews the stack caching algorithm used in the SC32 and the programmer's view of the stack cache. Performance measurements of the cache show that typically less than 1% of the processor's time is spent managing the cache.

Fast Double Unsigned Multiply and Divide

Wil Baden
339 Princeton Drive, Costa Mesa, CA 92626

These high level implementations of double unsigned multiply and divide are twice as fast because they do half the work. They do half the work because they assume that the magnitude of the factors are such that the result will be valid. Multiplication uses * and `UM*` once each to get three partial sums. Division is long division as taught in the forth grade.

Lean & Mean Single Pass Adaptive Data Compression

Wil Baden
339 Princeton Drive, Costa Mesa, CA 92626

The August 1988 issue of *Communications of the A.C.M.* had a paper, "Application of Splay Trees to Data Compression," by Douglas W. Jones, that may have important impact on data management. The paper gives an adaptive method of data compression that is so fast, so simple, and uses so little storage, that it could be used in program I/O. It is feasible for a program to keep data in compressed form, expand it, modify it, and compress it back again.

Co-routines

Wil Baden
339 Princeton Drive, Costa Mesa, CA 92626

Input and output of single bits are good applications for co-routines.

On each call of the co-routines we want to pack or unpack one bit. The overhead to determine which bit must be fast, as this has to be done for each bit of every byte.

Zen and Forth

C. H. Ting
Offete Enterprises, Inc., San Mateo, CA

This paper is based on the introduction to a GEnie Real-time Conference, October 26, 1988.

The historical developments of Forth are compared with that of Zen in China. There are striking similarities between Forth and Zen in their struggles against mainstream doctrines and establishments. The central messages borne by them are also very similar in stressing simplicity and personal freedom/understanding/involvement.

The question of the religious experience (enlightenment) of Forth programmers is also raised. Is this experience required of a successful programmer? Is it a hindrance to the acceptance of Forth to the general computing public?

Forth is the right programming language because it is not a programming language. Rather, it is a tool that allows the user to build a computer optimized for an application. The application is expressed in terms of a specialized instruction set, built upon the Forth instruction set. Forth user is designing special computers to solve specific problems. Other programmers are writing programs to cheat the computer. In most cases, the computer cheats back.

Design of a Data Base for Go Game

C. H. Ting
Offete Enterprises, Inc., San Mateo, CA

Go game is one of the oldest and most complicated board games. It is still very popular in Japan and China. The rules are rather simple. It is complicated becauuse of the large number of possible locations where stones can be placed. Documenting the moves is a big headache, not to mention programming such a game.

One approach to computerize Go game is to consider it as a data base problem. If all possible moves can be stored systematically in a data base, which allows easy and fast access, a large fraction of the problem in playing Go can be handled automatically.

This paper describes a data base system using tree structures to store Go games efficiently. The traversing mechanism can be used to build and extend the data base, and to retrieve data from the data base to guide actually playing.

The fundamental data unit in this system is a variable length record, which contains a list of stone locations and linking information. The list can be extended at run-time if necessary. Each element in the list has a pointer pointing to a sublist. The lists thus form a gigantic tree structure.

A tree can be traversed, searched, and extended by the following commands: **LEFT**, **RIGHT**, **UP**, and **DOWN**. These commands allows a user to navigate through the tree structure to search for playable sequences if the structure already exists. They can also be used in compiling the tree structure. When the user reaches the end of a list, he can extend the list with new data if necessary.

This tree structured data base will greatly facilitate the documentation of Go games. It reduces published game data and playing sequences to a machine readable form. It will thus improve the intelligence of any Go game playing program.

Trainable Neural Nets on a Novix Based Personal Forth Environment

John D. Carpenter

The performance of an advanced trainable neural net scheme will be demonstrated on a stand alone Novix based Personal Forth Environment. An attempt will be made to identify Forth words of general use in developing trainable neural nets with fuzzy data handling. Feasibility for ongoing training of neural nets under real time conditions with the Novix with projections to the Harris family of Forth engines will be studied.

An Improved Interpreter

Michael Perry

Why an improved interpreter? The goal is to provide more generality without adding much complexity or sacrificing much efficiency. A little history is in order. The FIG-Forth interpreter, **INTERPRET**, was a single loop which processed a chunk of source coming either from a disk block or console input. It was controlled by the system variable **STATE**, compiling or interpreting as needed. It had the advantage of simplicity, and a few drawbacks.

ANS X3/J14 Public Field Trial for Locals and Globals

The ANS X3/J14 Committee is hard at work developing an ANSI Standard for the Forth computer language and programming environment. As a part of that work, the committee is currently evaluating implementation strategies for a Local and Global variable extension word set. Attached are a related proposal and a comment. The proposal was synthesized by members of the Technical SubCommittee of X3/J14 as a composite of the proposals before the committee, taking account of "common practice" in this area. The comment, while not in the form of a proposal to the committee, is also included as an example of an alternate approach with more usage and performance restrictions, but perhaps a simpler syntax.

GOTO: A Proposal

C. H. Ting

Offete Enterprises, Inc., San Mateo, CA

There have been many proposals presented in previous FORML Conferences and recently to the ANS-Forth Standards Committee. It is quite obvious that people hate the strait jackets of the classical control structures (**DO-LOOP**, **IF-ELSE-THEN**, and **BEGIN-UNTIL**) which do not let them leave the structures conveniently.

Most of the new proposals develop various ways to exit loop structures, which tend to mess up the existing control structure word set. They also tend to make the control structures more complicated and less useful.