;login:

The USENIX Association Newsletter

Volume 7 Number 3 June 1982

CONTENTS

USENIX Association Notices

Temporary Change of the Offices of the Association

Beginning July 1, 1982, the offices of the Association will be temporarily located at:

USENIX Association
321 Mystic Street
Arlington, Mass 02174

Since we will not have a phone at this temporary location, all communications with USENIX should be via US Mail or electronic (ucbvax!g:usenix).

The USENIX Association will be soliciting proposals from responsible organizations for the provision of office and administrative services. The specifications of services required to be used in formulating a proposal can be obtained from the above address after the Boston Meeting.

Elections

Ballots have been mailed to all institutional members of the Association for the election of officers and directors. The deadline for return of the ballots was June 15, 1982. The results of this election will be announced at the Boston USENIX Conference and in the next issue of *;login:*.

NOTICE

;login: is the official newsletter of the USENIX Association, and is sent free of charge to individual and institutional members of the Association.

The USENIX Association is an organization of AT&T licensees and sub-licensees formed for the purpose of exchanging information and ideas about the UNIX* operating system and the C programming language. It is a not-for-profit corporation incorporated under the laws of the State of Delaware. The officers of the Association are:

President	Lou Katz	Directors	John Donnelly
Vice-President	Peter Weiner		Ira Fuchs
Secretary	Lew Law		Debbie Scherrer
Treasurer	Mel Ferentz		Wally Wedel

Membership information can be obtained from the office of the Association:

USENIX Association
Rockefeller University
Box 8
1230 York Avenue
New York, NY 10021
212-570-8934

Members of the UNIX community are heartily encouraged to contribute articles and suggestions for *;login:*. Your contributions may be sent to the editor electronically:

ucbvax!g:usenix

or through the US mail:

Lou Katz
541 Evans Hall
EECS Department
University of California
Berkeley, CA 94720

The USENIX Association reserves the right to edit submitted material.

;login: is produced on UNIX using *troff* and a variation of the −me macros. We appreciate receiving your contributions in *n/troff* input format, using any macro package. If you contribute hardcopy articles please leave left and right margins of 1" and a top margin of 1½" and a bottom margin of 1¼". Hardcopy output from a line printer or most dot-matrix printers is not reproducible.

;login: Editor	Lou Katz
;login: Publisher	Strong Consulting
	5706 Van Fleet Ave
	Richmond, CA 94804

---

*UNIX is a trademark of Bell Laboratories

Boston Conference

USENIX and /usr/group (Tentative) Meeting Agenda

Tentative Schedule for Monday, July 5, 1982

7:00pm - 10:00pm	Registration at the Copley Plaza Hotel

Tentative Schedule for Tuesday, July 6, 1982

8:30am - 5:00pm	Registration at the Copley Plaza Hotel
9:00am - 5:00pm	Introduction to UNIX Tutorial
8:45am - 8:50am	Welcome - Alan Nemeth, Program Chairman
8:50am - 8:55am	Welcome - Lou Katz, USENIX president
8:55am - 9:00am	Welcome - Bob Marsh, /usr/group president
	Session Chair - Unassigned
9:00am - 9:20am	William R. Northlich, Jr., Zehntel, Inc. Embedding UNIX in a Product (or, is "Real Time" Real?)
9:20am - 9:40am	A.V. Hays, Jr., National Eye Institute, B.J. Richmond, National Institute of Mental Health, and L.M. Optican, National Institute of Health REX: A UNIX-Based Multiple Process System for Real-Time Data Acquisition and Control
9:40am - 10:00am	Robert Schwartz, Mark Williams Company Real-time and UNIX
10:00am - 10:20am	Jim Isaak, Charles River Data Systems Real Time Systems
10:20am - 10:40am	Coffee Break
	Session Chair - Joseph Yao, Science Applications, Inc.
10:40am - 11:00am	Dennis F. Meyer, UNIQ Computer Corporation Optimizing Database Queries in SQL
11:00am - 11:20am	Michael E. Duffy, Software and Consulting Interfacing UNIX to Backend Database Machines
11:20am - 11:40am	Fred M. Katz, Logical Software, Inc. Time and Tuples: Concurrency Control in LOGIX
11:40am - 1:20pm	Lunch
	Session Chair - Bill Joy, U.C. Berkeley
1:20pm - 1:50pm	Bill Reeves, Lucasfilm, Ltd. UNIX at Lucasfilm Ltd. or Does Darth Vader Code in C?
1:50pm - 2:10pm	Bill Joy, U.C. Berkeley 4.2BSD Overview
2:10pm - 2:50pm	Sam Leffler, U.C. Berkeley 4.2BSD Network Communications
2:50pm - 3:20pm	Coffee Break
3:20pm - 4:00pm	Bill Joy, U.C. Berkeley 4.2BSD Interprocess Communication Primer
4:00pm - 4:40pm	Kirk McKusick, U.C. Berkeley 4.2BSD File System
4:40pm - 4:50pm	David Mosher, U.C. Berkeley 4.2BSD Licensing
4:50pm - 5:15pm	Joy, Leffler, Mosher and McKusick 4.2BSD Questions and Answers
evening	Boston Pops Orchestra Concert

Tentative Schedule for Wednesday, July 7, 1982

8:30am - 5:00pm	Registration at the Copley Plaza Hotel
	Session Chair - Unassigned
8:40am - 9:00am	Mario Ruggiero, Univ. of Toronto Computing Services Ped - A Portable Editor
9:00am - 9:20am	Chaim E. Schaap, Delft Consulting Corporation Portability of C Language Programs
9:20am - 9:40am	Dr. Brad J. Cox, ITT Programming Technology Center The Object Oriented Pre-Compiler Programming Smalltalk 80 Methods in C Language
9:40am - 10:00am	Bill Tuthill, Computing Services, Univ. of California Teaching AWK as a First Programming Language
10:00am - 10:20am	Eric S. Rosenthal, IMI Systems Spelling Checkers, Compound Words and Variant Spellings
10:20am - 10:40am	Coffee Break
	Session Chair - Mike O'Dell, Lawrence Berkeley Laboratory
10:40am - 11:00am	Richard Fortier and Anthony Lake, Bolt Beranek and Newman, Inc. Design of an Intelligent Bitmap Terminal
11:00am - 11:20am	Andreas Bechtolsheim, Sun Micro Systems, Inc. The Sun Workstation
11:20am - 11:40am	Rob Pike, Bell Telephone Laboratories, Murray Hill Merging Bitmap Graphics with UNIX
11:40am - 1:20pm	Lunch
	Session Chair - Unassigned
1:20pm - 1:40pm	Douglas P. Kingston III and Michael J. Muuss, Ballistics Research Lab., Aberdeen Proving Grounds The Multiple Device Queueing System
1:40pm - 2:00pm	Mark Kampe, INTERACTIVE Systems Everything you wanted to know about System III but Bell was afraid to tell you
2:00pm - 2:20pm	Steve Dyer, Bolt Beranek and Newman, Inc. Bad Block Handling
2:20pm - 2:40pm	Mark Horton, Bell Laboratories The New Curses and Terminfo Package
2:40pm - 3:00pm	Douglas I. Kalish, Logical Software, Inc. Programdb: Maintaining Symbol Use Data for Source Code Control
3:00pm - 3:20pm	Fred Katz, Logical Software, Inc. Logical Shell
3:20pm - 3:40pm	Coffee Break
	Session Chair - Mike O'Brien, Rand Corporation
3:40pm - 4:00pm	David Tilbrook and Ken Jackson, Systems Designers Limited Mascot and UNIX: Their Combination and Applications
4:00pm - 4:10pm	Michael D. Tilson, Human Computing Resources Corp. How to use lots of memory
4:10pm - 4:30pm	Robert Ragan-Kelley, Pyramid Technology Corp. The Ultimate UNIX System
4:30pm - 4:50pm	Charles Minter, INTERACTIVE Systems Corp. A High-Performance Computer System Suited to UNIX
4:50pm - 5:10pm	Joel Carter, The Wollongong Group, Inc. Perkin-Elmer's Hardware I/O system: Flexibility that matches UNIX
evening	Cocktail Reception at the New England Aquarium

Tentative Schedule for Thursday, July 8, 1982

8:30am - 5:00pm	Registration at the Copley Plaza Hotel
	Session Chair - Unassigned
8:30am - 8:45am	Lou Katz News from USENIX
8:45am - 9:00am	Bob Marsh News from /usr/group
9:00am - 9:20am	Larry Isley, AT&T News from AT&T
9:20am - 9:40am	Armando Stettner, DEC News from DEC
9:40am - 9:50am	Carolyn Autrey-Hunley and Jack McCredie Introduction of EDUCOM-UNIX Task Force
9:50am - 10:00am	George Goble What's new at Purdue EE Department
10:00am - 10:20am	Michael C. Toy and Kenneth C. R. C. Arnold Rogue: Where It Has Been, Why it Was There, and Why It Shouldn't Have Been There in the First Place
10:20am - 10:40am	Coffee Break
10:40am - 11:00am	Jeffrey L. Kodosky, National Instruments UNIX etc. at National Instruments
11:00am - 11:20am	Dave Preston, Perkin-Elmer Corporation News from Perkin-Elmer
11:20am - 11:40am	John Z. Kornatowski and Ivor Ladd, Rhodnius Incorporated Current Database Research at the Computer Systems Research Group, University of Toronto
11:40am - 1:20pm	Lunch
	Session Chair - Lou Katz, U.C. Berkeley
1:20pm - 1:40pm	Philip J. Mercurio, Cognitive Science Lab, UCSD The UCSD MSG System: Iterative Design in the UNIX Environment
1:40pm - 2:00pm	Mark T. Horbal, UNIQ Computer Corp. ATLAS test language
2:00pm - 2:20pm	Jack Dixon, UNIQ Computer Corp. UNIX and manufacturing testing
2:20pm - 2:40pm	Curtis Sanford and David Walden, BBN Computer Corporation Development of a Large Applications System under UNIX
2:40pm - 3:00pm	Masatoshi Kurihara and Yukio Ikadai, Software Research Associates, Inc. Application Programming Environment on UNIX
3:00pm - 3:20pm	Benjamin J. Woznick, Bolt Beranek and Newman, Inc. Managing a Roomful of UNIX Systems
3:20pm - 3:40pm	Coffee Break
	Session Chair - Marleen Martin, 3COM Corporation
3:40pm - 4:00pm	Howard Salwen, Proteon Associates, Inc. On Ring Architected Local Networks
4:00pm - 4:20pm	Steve Zucker, INTERACTIVE Systems Corp. A Family of Portable Systems Based on System III
4:20pm - 4:40pm	John Z. Kornatowski and Ivor Ladd, Rhodnius Incorporated Current Status of Mistress (Version 2) and Future Plans
4:40pm - 5:00pm	Mike Meissner and Robert Weisman, Data General Corp. A C Compiler for AOS/VS
5:00pm - 5:20pm	Elwyn Wareham, Systems Designers Limited Systems Designer Limited Vendor Presentation on Angus
5:20pm - 5:40pm	Daniel Walsh, Amdahl Corporation UTS: UNIX on the Amdahl 470
evening	USENIX Open Board Meeting

Tentative Schedule for Friday, July 9, 1982

8:45am - 11:45am	Software Tools Technical Sessions
8:45am - 11:45am	Birds of a Feather Sessions
9:00am - 5:00pm	4.2BSD Tutorial from Berkeley
	Session Chair - Unassigned
8:40am - 9:00am	Gary Perlman, Department of Psychology, U.C. La Jolla Compact Data Analysis Programs for UNIX
9:00am - 9:20am	Gary Perlman, Dept. of Psychology, U.C. San Diego MENUNIX: An Interface to UNIX Programs and Files
9:20am - 9:40am	Michael J. Heffler, Delft Consulting Corp. Description of a Menu Creation & Interpretation System
9:40am - 10:00am	Eugene F. Dronek, Aim Technology Benchmarking to Eliminate the Benchwarmers
10:00am - 10:20am	Martin Tuori, D.C.I.E.M. A UNIX Benchmarking Tool, and Results from the PDP 11/44, VAX 11/780 and Perkin-Elmer 3242
10:20am - 10:40am	Coffee Break
	Session Chair - Unassigned
10:40am - 11:00am	Daniel Walsh, Amdahl Corporation UTS: UNIX on the Amdahl 470
11:00am - 11:20am	Gregory J. O'Brien, Digital Equipment Corp. Porting UNIX to a Personal Computer
11:20am - 11:40am	James L. Weiner and Brian L. Johnson, Computer Science Department, University of New Hampshire UNIX/Prime: Porting the UNIX operating system to Prime machines
11:40am - 1:20pm	Lunch
1:00pm - 5:30pm	Software Tools Technical Sessions
1:30pm - 5:00pm	Birds of a Feather Sessions
	Session Chair - Unassigned
1:20pm - 1:40pm	Sanand Patel and Richard Sniderman, Human Computing Resources Corp. UNIX Emulation, Again
1:40pm - 2:00pm	Michael Caplinger, Dept. of Math Sciences, Rice University A UNIX Emulator for VAX/VMS
2:00pm - 2:20pm	Mike Bender, ZILOG Selecting a DBMS for a Super Micro
2:20pm - 2:40pm	Gary Williams, Durango Systems, Inc. A Business-Oriented File Manager under UNIX, with Contention Control and ISAM
2:40pm - 3:00pm	T. Scott Pyne, Science Applications, Inc. IAFORM, An On-Screen Definition Package for Data Retrieval Forms
3:00pm - 3:20pm	Gordon W. Waidhofer, The Wollongong Group, Inc. TABSTAR - Information Data Base Management
3:20pm - 3:40pm	Coffee Break
	Session Chair - Unassigned
3:40pm - 4:00pm	Bob Greenberg, Independent Consultant Is UNIX as a Standard Doomed?
4:00pm - 4:20pm	James R. Hanley and Jeffry A. Scott, Laboratory for Information Science in Agriculture, Colorado State University A Survey of UNIX Usage in Scientific and Business Applications
4:20pm - 4:40pm	Roger McKee, The Wollongong Group The Coming UNIX Crash
4:40pm - 5:00pm	Dr. Rebecca Thomas and Jean Yates, Gnostic Concepts, Inc. The Commercialization of UNIX

Software Tools Users Group (Tentative) Meeting Agenda

Friday, July 9, 1982
Copley Plaza Hotel

8:45	Welcome David Stoffel, Users Group Co-ordinator
9:00	Software Tools Bulletin Board CompuServe
9:30	Tools on the DG Nova Jon Hanshew, CompuCode
10:00	Tools on the DEC-20 Tektronix
10:30	Tool Portability Issues Bob Upshaw, Lawrence Berkeley Laboratory
11:00	Proposed Extensions to the Primitives and Library Panel Discussion
11:45	Lunch
1:00	The NBS "Software Tool Database" NBS
1:30	Spelling Checker Algorithms Massachusetts Institute of Technology
2:00	Graphic Output Tools Rosenbaum, IMI
2:30	Software Testing and Tracking Neil Groundwater, Analytic Disciplines Inc.
3:15	Rocky Mountain Area Implementors Group Ben Domenico, NCAR
3:30	Commercialization of the Tools Panel Discussion
4:00	Implementor and SIG group meetings
5:00	Future Directions Discussion

Boston Meeting 4.2BSD Tutorial

By Sam Leffler of the Computer Science Research Group at U.C. Berkeley

The tutorial will cover the design and use of the networking and interprocess communication facilities to be distributed in the fall 4.2BSD release of UNIX for the VAX. In particular, the following topics will be discussed:

- writing client/server style programs
- providing new protocol support
- writing network device drivers

Discussions will cover existing examples of servers, protocols, and device drivers. Ongoing work at Berkeley and other sites will also be covered. Some familiarity with basic networking concepts and the UNIX operating system internals is expected. All attendees are expected to have experience with the C programming language.

The tutorial will consist of two half-day sessions. The morning session will concentrate on topics related to use of the networking and interprocess communication facilities, while the afternoon session will deal with system internals.

USENIX Distribution Tape

The first 1982 USENIX distribution tape is completed and currently being distributed. There were four organizations who made submissions, primarily at the January meeting in Santa Monica.

Computer Corporation of America submitted a relatively complete EMACS. This is well-documented with help files, Read_Me's, and manual entries. It includes directories for setup and addresses of persons to contact if trouble arises. According to CCA, this EMACS should be relatively bug free as it has been in operation at CCA for some time. It is intended to run only on a VAX (it's big).

The submission by General Instrument Corporation contains a device driver and library to run the Genisco GCT300 color graphics system on a VAX. The code has no license restrictions, although one must have a source license for the kernel to install the driver. GI has also submitted some sources to system commands that require UNIX 32V and Berkeley 4.1 BSD licenses. Many of the updates include those made necessary by the Berkeley increase in block size from 512 to 1024 bytes. There is also a set of commands to implement functions that act on group id's much like "su" acts on user id's. Finally, there are some line printer programs that are enhanced versions of 4.1 code, and include graphics support for the LSY-11 (Printronix 300).

Geotronics Corporation has added some useful utilities to its Austin submission. There are Seventh Edition commands for Sixth Edition systems, a program that turns terminal ports on and off, an interactive programmable form filler, development tools for multiplexed binary data files, a general purpose simple-minded output spooling system, a "lock" call and real time support for Sixth Edition kernel, and a graphics driver for the H.I. CPS-15/6 plotter. There is plenty of documentation and lots of support (library and include files) for all the programs.

The final contribution is from Lincoln-Sudbury Regional High School. All programs have been tested on the school's PDP 11/70, but the submitter warns that some editing may be necessary to remove some installation-specific features he may have missed. The programs submitted are as diverse in function as the Geotronics submission, and no less practical. There is an implementation of the Logo language interpreter, programs to assist in the creation and typing of form letters, some modifications to V7 system programs (requires V7 license) including improvements to *nroff*, automatic system down/up, better administration of *dump/restor*, etc., an EMACS-like editor called TORES, an EMACS-like editor patterned after TORES but faster, a directory editor, and quite a few games. Since these programs were written by high school students, it will be interesting to hear the response from members on their quality and practicality.

In all submissions, the Read_Me files were generally very helpful and most included names and addresses of people to contact for help or further information.

USENIX would like to encourage response to these submissions, both in the form of newsletter letters or articles discussing their merits and in future tape submissions built on or inspired by these packages.

Submissions for the second 1982 distribution tape will be accepted at the Boston meeting or through the mail to the USENIX office.

Jeff Hardy
Phil Cohen
USENIX Tape Distribution Committee

Interesting Developments

Bill Joy of UCB Moving to Sun Microsystems

Bill Joy has decided to become involved with a new startup company and will be phasing out of the Computer Systems Research Group at Berkeley over the next few months. He will be joining Sun Microsystems, Inc., a company whose founders include Andy Bechtolsheim, the designer of the Sun workstation. SMI is one of a number of companies which plan to offer microprocessor-based networked workstations running 4.2BSD software.

Bill plans to continue full time until July 1 when an early version of the 4.2BSD distribution should be complete and running in-house. He will then continue half time through its polishing, tuning, beta testing and documentation phases. Bill expects to finish writing his PhD thesis by December.

Bill will continue as a contributor and advisor to CSRG, although it will be a secondary activity for him. While SMI may need to develop proprietary software in certain specialized areas, Bill expects fixes to the shared base of 4.2BSD programs which are made at SMI can be distributed by Berkeley. The current cooperative efforts between CSRG and various industrial groups are seen as a model for the relationship.

Bill has been a valued colleague and friend during his years at Berkeley and he will be very much missed. I hope you will join me in wishing him well as he makes this transition.

Prof. Bob Fabry
University of California, Berkeley

Daisywheel Printer with Extended Character Set

Diablo Systems Inc. of Hayward, CA, has announced a new daisywheel printer that can reportedly print about 340 characters from one printwheel. The new model 630 ECS uses new printwheels that have 2 rows of characters on each finger for 192 characters per wheel. It can also can use existing metal and plastic printwheels. One of the two new printwheels announced with the 630 ECS combines an ASCII alphanumeric character set with Greek, math and other specialized symbols. Diablo says that additional characters can be constructed by printing part of the character from the outer wheel and part from the inner wheel. This printer sounds very interesting for UNIX users who need to produce high quality scientific and technical documents with *nroff*.

The 630 ECS is currently available only with a parallel interface; a serial interface model should be available in the fourth quarter of this year. It will be compatible with existing model 630s. A package to update existing 630s to the ECS capabilities will be available early next year. The printer will cost around $3500 and both it and the upgrade kits will be available from Diablo distributers.

DEC C Compiler for VMS

DEC has announced a C compiler package called VAX−11 C for their VMS operating system. It supports all of the language features in Kernighan & Ritchie's book. The compiler produces optimized, sharable, and position-independent code. Runtime support includes all of the non-UNIX-specific routines offered by Bell and some UNIX-specific routines (but not *fork* or *pipe*). There is also an new "byte string" file type. Their C manual is reported to have a section on how to transport C programs developed on UNIX to VMS.

The package is integrated into the VMS common language environment so all VMS services are available to programs written in C. The package is available now for about $8000 for binaries, documentation, and support services, or about $5000 for binaries only.

F77 Performance

David A. Mosher
Robert P. Corbett

Computer Systems Research Group
University of California
Berkeley, California 94720

April 1, 1982

ABSTRACT

The Fortran compiler under UNIX† has been the object of many heated battles. This paper describes work done to improve the performance of the Fortran compiler to a level comparable with the Fortran compiler under VMS‡ on a VAX‡.

Introduction

Fortran, though not a dominant language in our work, is still heavily used in other research efforts and in the commercial world. Many companies have invested millions of dollars in programming efforts using Fortran. In addition, there are a number of applications oriented tools (such as SPICE) used by research organizations which are written in Fortran for reasons of portability. Converting these programs to other languages would be costly and time consuming. Thus, having a Fortran compiler on a system is imperative.

The original UNIX Fortran 77 compiler was written to be portable. *F77* has been brought up on new machines in as little as two days. The performance of the compiler was considered a secondary factor in the design of the original compiler. The only real criterion was that it should produce correct code for Fortran 77 programs on a wide variety of machines.

For serious Fortran applications, the efficiency of the object programs is almost as important as correctness. A large percentage of Fortran programs are compute bound, and so the relative efficiency of codes produced by different compilers is easily noticeable. Many VAX users have expressed concern about the efficiency of the codes produced by the original *f77* compiler. Therefore, an effort was organized at Berkeley to add an optimizer to *f77*.

The remainder of this paper discusses our improvements to the *f77* compiler and our analyses of the problems yet to be solved.

Addition of a Simple Optimizer

Our initial goal was to add a basic optimizer to the compiler. To provide a framework for the addition of these optimizations, a scheme was coded to buffer incoming statements into an internal buffer of a fixed size. The fixed size was necessary to maintain compatibility with machines which have smaller address spaces. Since there was no guarantee that enough information could be buffered to take advantage of more than a simple block of statements, the optimizer only works over blocks of statements which are known to have a single entry point.

The two initial optimizations were common subexpression elimination for basic blocks and invariant code motion for DO loops. A simple example where these optimizations take place is:

† UNIX is a trademark of Bell Laboratories

‡ VAX and VMS are trademarks of Digital Equipment Corporation

```
dimension integer a(2800,1100)
do 10 i = 1, 100
        do 20 j = 1, 200
                n = i * 5 + j * 3
                m = n * 2 + j * 3
                a(m,n) = 1
                a(0,n) = 2
                a(i,1) = 3
        20 continue
10 continue
```

This program segment includes both the implicit and explicit occurrences of the types of codes we expected to optimize. The two instances of the subexpression 'j * 3' are examples of explicit common subexpressions. The common subexpressions generated by the array references 'a(m,n)' and 'a(0,n)' are implicit. The address calculations for those references include the implicit subexpression '2800*n'. In both cases, the optimizer will produce code to reuse the value obtained from the first instance of each of those subexpressions in place of the code to reevaluate them. Similarly, the explicit expression 'i * 5' and the implicit address calculation for 'a(i,1)' are invariant over the inner loop and both will be moved out of the loop.

Optimizing array references has proven particularly troublesome. Special consideration is needed to utilize the VAX addressing modes which support array indexing. Applying optimizations without checking for special cases results in a substantial performance degradation.

Adding those optimizations proved to be of significant benefit for most of our benchmark programs; however, the benchmark *fft* was still much slower than the version compiled using the VMS compiler. Close analysis of *fft* showed that the optimizations performed by our basic optimizer had already been carried out through careful hand optimization of the source code.

A significant deficiency of the compiler was its lack of global register allocation. Adding a simple register allocator promised a large payoff.

Allocating registers for simple variables and common subexpressions had a minimal impact on the speed of *fft*. However, after the compiler was changed so that registers could be used as base registers for arrays, its speed improved dramatically.

The basic optimizer was completed but did not achieve expected results. Analysis of the optimized code led us to believe that the differences in the execution times of the codes produced by *f77* and VMS Fortran could not be traced solely to the lack of global optimizations.

The Real Problems

By comparing the codes produced by the VMS Fortran compiler and the second pass of the portable C compiler, we discovered two major problem areas.

One problem was that double precision calculations and functions were often generated where single precision was sufficient. An error was found in the backend code generator which forced calculations following a unary operator to be done in double precision. The use of double precision math functions was viewed as a problem but we could not accurately assess the cost because of the complexity of the benchmark programs. Further analysis revealed that the conversion from single to double precision took 5 times longer than instructions which would have accomplished the same result. An integer to floating point conversion took 10 times as long as a simple move instruction. Conversion of numerical types turns out to be very expensive and should be avoided.

During our investigation of the aforementioned problem, we noticed that the VMS Fortran compiler used integer move instructions to move floating point numbers instead of the corresponding floating point instructions. At first glance, this seemed to be an ugly abuse of the instruction set. Later, we concluded that the integer move instructions could be used without ill-effect as long as the next instruction was not a conditional branch. But the question remained "why use the integer instructions?" Timing of these instructions showed that the floating point move instruction took 3 times as long to execute.

With the addition of a global register allocator, we noted that the portable C compiler required two registers be assigned to every floating point value. Further, those registers must be paired and aligned on an even register boundary. Since the VAX needs only one register to hold a single precision value, and makes no alignment requirements, those requirements are extremely wasteful. Because all floating point operations generated by the C compiler are double precision, that pattern of register allocation has been ingrained in the code generator. We are continuing to work on this problem.

The second problem was that the compiler did not generate indexing modes. In timing tests, we found that an arithmetic shift was extremely expensive and that the equivalent elementary code to replace indexing modes was significantly more expensive than the use of indexing modes. Allocating registers to serve as base registers for arrays caused some index mode instructions to be produced. However, there were still many instances in which index mode instructions should have been generated but were not. In an attempt to understand the problem, a patch was added to the code generator to produce index mode instructions where applicable. Still the problems persisted. After a careful study of the code generator, we determined that the common subexpression optimization was inhibiting the use of index mode. We also found that the trees produced by the original *f77* compiler were not of the form the portable C compiler recognized as candidates for index mode. To resolve this problem, special cases had to be introduced into the optimizer for producing subscript codes for the VAX. While implementing those special cases, we uncovered a coding error in the original compiler which was responsible for the expression trees not being in the proper form to generate indexing modes.

With relatively few changes but a much better understanding of the code generated, the code ran significantly faster. In all, one VAX dependency and a few errors accounted for much of the inefficiency of the code produced by *f77*.

Closer, But Still More to Do

At this point, the timings for some of the benchmarks were within 10% of the reported timings of VMS Fortran generated code. The major exceptions were *fft* which was a full 100% slower, and *bench2* which was 85% slower. Because the code produced for *fft* by our compiler was similar to that produced by the VMS compiler, we began to have serious doubts about the reported times given for *fft*. Our first task was to reproduce the timing results. After a few tries and a midnight session, the reported times were reproduced to within 10%.

Before our timing experiments, we did not know if the timing results produced by VMS even measured times in the same units as UNIX. Also, the methods used to obtain timing data were very different. A function was developed to return a timing value with the measurement characteristics described in the VMS manuals. This function showed that little of the time reported by timing commands on UNIX was due to startup or cleanup of a program. With this tool in hand, the task was to prove that 1 unit of time on VMS was equivalent to 1 unit of time on UNIX.

A Close Analysis of FFT

Because the execution time discrepancies were greatest for *fft*, it was used to verify the VMS timings. Solving the riddles of this program would shed a light on the remaining problems.

The *fft* program has three major sections: initialization, a fast Fourier transform, and an inverse fast Fourier transform. In measuring each section, we found that the initialization took 2.6 times longer than the measured times under VMS. Since the initialization code was small and relatively simple, running the assembly code generated by the VMS Fortran compiler under UNIX would provide a simple comparison of the timings units of VMS and UNIX.

Our approach was to start with the assembly language produced by *f77* and transform it into the assembly language produced by the VMS compiler. The first glaring problem with the code generated by *f77* was the poor handling of complex values. By recoding the complex assignment to a generatable code sequence, the initialization took 2.5 times longer than under VMS.

The effects of using double precision functions for single precision calculations had to be assessed. The exponential function in the initialization code was replaced with a handwritten assembly language routine for computing a single precision exponential function with the argument and result passed as double precision values. This change made some difference but did not prove significant. The

exponential function and the calls to it were changed to single precision. The initialization then took only 1.2 times longer than the code generated by the VMS Fortran compiler. Once again, unnecessary type conversions had proven to be the major bottleneck. Additional work is needed in this area to provide both single and double precision library routines and the interface to these routines.

The remaining differences in the initialization code were that the C backend code generator used the *acbl* instruction in a case where the *aobleq* instruction could have been used, a few register variables were allocated differently, and a few extra move instructions between registers and variables were being generated. When the *acbl* instructions were changed, the initialization code took 1.1 times longer. This result was a surprise since the change was so minor. Later timings showed that an *acbl* instruction takes twice as long to execute as an *aobleq*.

With the removal of extra assignments of register variables to their memory locations and adding a few more register variables, the initialization code ran as fast as the VMS Fortran generated code. We proved that the timing units were indeed the same and learned a fair amount about the VAX.

In summary, the major problem in the initialization code was the use of double precision functions where single precision routines could have been used.

The Final Analysis of FFT

At this point, *fft* took 1.9 times longer on UNIX than its VMS counterpart. There were four main classes of differences between codes produced by the VMS compiler and those produced by *f77*.

We found that VMS was able to make better use of registers. Variables in inner loops which our optimizer kept in memory were moved to registers. With this change, *fft* took 1.85 times longer than its VMS counterpart.

In our investigation, we found two interesting results. Special casing of powers of 2 in the code produced for the *power of* operation had little effect. We also found that the compiler generated elementary code for what could have been an *acbl* instruction with an increment of 2. After replacing the elementary code with an *acbl* instruction, we found performance suffered. This result was confirmed by comparisons of the *acbl* instruction with other equivalent codes.

The next problem area was the use of double precision math function. We had hoped that changing the math function calls from double to single precision would account for most of the slowness of the fft calculation since this problem was quite significant in the initialization code. But it turned out that using single precision math functions only improved the performance ratio to 1.7 times longer. We obtained slightly better performance by reducing the number of coefficients used in the *cos* and *sin* functions to be adequate for single precision use. We were able to gain a small improvement by putting a few more variables into registers. The execution time for *fft* dropped to 1.6 times slower when single precision values were no longer spilled unnecessarily.

By eliminating six move integer from register to memory instructions, we improved performance to 1.5 times slower.

We found that the DO loop code was quite different. We adopted the VMS DO loop code and found little difference in performance. Our study of DO loop codes showed that the code *f77* produces for floating point DO loops does not conform to the Fortran standard. The standard defines the number of iterations of a DO loop to be given by an expression which is evaluated at the start of the loop. We found that the number of iterations of loops generated by *f77* need not be close to the number indicated by the value of that expression. Tests of other Fortran 77 compilers have shown that they too fail in this area. Presumably, floating point DO loops are used so infrequently, that this failing has not been noticed. The code produced by the VMS Fortran compiler is technically correct but has a number of surprising properties. In particular, the final value of the loop control variable can be far greater than the limit value.

In summary, global register allocation was a major problem area in the fft calculations. The cost of double precision math function was significant but not great.

The Remaining 50%

At this point, the assembly code running under UNIX looked almost identical to the code produced by the VMS compiler. The only remaining difference of any significance was that *f77* produced absolute addresses for static variables, while VMS used base register/displacement addresses. Because our investigations used assembly language programs, we were slow to recognize the importance of the different choices of addressing modes. Absolute mode addresses on the VAX require five bytes per address. Displacement addresses typically require only two or three bytes. Therefore, object programs produced by the VMS compiler could be as much as 50% smaller than those produced by *f77*. Because of the VAX instruction buffer, the size of the object code can have a large effect on its execution time.

F77 has now been modified to produced the same addressing modes as the VMS compiler. With this change alone, the execution time of *ffi* is reduced from 24 seconds to 17 seconds. When our basic optimizer is used together with the new addressing scheme, its execution time drops to 12 seconds. The VMS compiler is still somewhat faster, but the difference is no longer as pronounced. Over our set of benchmark programs, the average difference in executions times is now between 10 and 15%.

Conclusions

In conclusion, the lack of a global optimizer was a key problem but not the only problem. *F77* uses double precision library functions for both single and double precision computations because the C libraries do not include double precision routines. But as shown above, the conversions added to the object code to support such a scheme are very painful on the VAX. Also, some of the instances in which bad code was produced could be traced to a few simple errors in the compiler which had major ramifications. But in the final analysis, the extra bytes needed for absolute addresses were the most significant factor in the performance.

Significantly, the speed of Fortran object programs under UNIX was found to be related to the "maturity" of the compiler. In no way was the UNIX system itself found to adversely affect program execution speed.

Future Work

We are now in the process of producing a more elaborate global optimizer. New optimizations specifically keyed to the VAX are to be added. Also, the old buffering scheme is being replaced with a more flexible scheme which will permit more global data flow computations to be performed. We hope that the work underway will resolve the problems cited above and produce a compiler which generates code equal in execution speed to the code generated by the VMS Fortran compiler.

In addition, we are now making a major effort to improve the reliability of the compiler. We also hope to improve the facilities provided by the compiler to aid in debugging.

Acknowledgements

We would like to express our appreciation for gift funds from TRW which made this work possible. We would like to acknowledge the contributions of the other members of the Fortran group: Channing Brown and Alastair Fyfe. We would like to thank Tom Ferrin at the University of California at San Francisco and Mike O'Dell at Lawrence Berkeley Laboratory for access to VMS systems. We would like to acknowledge the work done by Professor Kahan for the new math library functions and Dave Wasley for the I/O library.

References

For previous discussion on Fortran performance, see *UNIX and VMS — Some Performance Comparisons* by David Kashtan, *Comments on the performance of UNIX on the VAX* by William Joy, and *Performance Issues of VMUNIX Revisited* by Thomas E. Ferrin.