CALIFORNIA INSTITUTE OF TECHNOLOGY

MEMORANDUM

VLB ARRAY MEMO No. 358

To: VLBA Memo Series 6/21/84
From: Martin Ewing
Subject: A Question of Language: C vs Fortran, etc.

At the request of some interested NRAO parties, I have culled and slightly edited a dialog that occured through the Caltech VOX system.

- - - - - - - - - - -

From: DLM 1-MAY-1984 11:11
To: VOX
Subj: C, FORTRAN, VMS and U***

I have had occasion to run essentially the same program on three machines (PHOBOS, LOGOS, and the UNIX HEP VAX) in two different languages (C and FORTRAN). The program is a one dimensional hydrodynamic calculation of a shock wave traveling through a tube; the computations include setting up the problem and running the evolution 200 time steps. Each code was compiled with optimization (/OPTIMIZE or -O). In the case of FORTRAN, the code was compiled on PHOBOS and run on both PHOBOS and LOGOS and then conversely compiled on LOGOS and run on both with identical results. (As UNIX FORTRAN is not as structured as VMS FORTRAN, DO/END DO structures and other items had to be translated in order to run on HEP. The translation from VMS FORTRAN to C, however, was nearly trivial as they are both structured in similar ways.)

Each program was timed using ^T or the UNIX TIME command and extracting the user time only. The results are shown below in CPU seconds (numbers in parentheses show the evolution time only -- the most CPU-intensive part of the calculation -- and are not available for the UNIX VAX):

LANGUAGE	LOGOS(VMS 750)	PHOBOS(VMS 780)	HEP(UNIX 780)
FORTRAN	12.5 (11.6)	3.8 (3.2)	6.0
C	33.3 (32.5)	7.5 (6.7)	6.7

It is no surprise that the VMS FORTRAN compiler is very good. This is common knowledge. What is most surprising, however, is that it is still a factor of 2 better than any C compiler, even the UNIX one. Furthermore, I would strongly suggest that no production runs of C programs be done on LOGOS as that machine is a factor of 4 slower in this case and a factor or 9 slower than FORTRAN on PHOBOS.

welcome any suggestions from C pundits on ways to optimize the C version of the program which cannot be done in VMS FORTRAN. The code resides in JPL:[DLM.CCP]LAXWEN.FOR and LAXWEN.C. Further investigations of this sort are needed, I think, in order to determine how we are to use LOGOS, PHOBOS (or the new JPL machine), and the concurrent processors we will acquire, and which language will become dominant. Personally I look forward to the installation of VMS FORTRAN on the CPs someday soon.

From: BCB 1-MAY-1984 15:06
To: VOX

I have examined your C code for LAXWEN, and have the following comments.

1. As they stand, your comparisons are not entirely fair. Two factors affect the speed of any program:
   a. Design of the language, i.e. access to efficient methods
   b. Quality of the compiler

The primary speed advantage of C over FORTRAN as a language is that one may avoid index calculations in tight loops. With regard to this, LAXWEN.C is really crypto-FORTRAN with array indexing going on in the innermost loop. I made a first pass over the code to convert it to a more efficient form (ten minutes in EDT); the result is jpl:[bcb]LAXBCB.C. I suggest you compare its runtime with the others (I will be happy to help you get it compiled if there are bugs in it -- No guarantees as it stands since it is super quick-and-dirty). I shall be highly surprised if it is not a lot better (at least 50%) on VMS and MUCH FASTER THAN FORTRAN on UNIX.

2. The VAX/VMS FORTRAN compiler is very good. The C compiler is relatively new, and, as I understand it, it is merely a reworking of the VMS PL/1 compiler. It is not shocking that the 'bugs' are not out of it yet. DEC's new, and, as I understand it, it is merely a reworking of the VMS PL/1 compiler. It is not shocking that the 'bugs' are not out of it yet. DEC's compilers tend to be highly non-straightforward and very complex, and this one is probably not optimized for the language. The UNIX compilers, on the other hand, are very simple in design (made to be portable) and not nearly so sophisticated, and tend to produce mediocre code. The UNIX compilers themselves take a disgustingly long time to run.

If you need convincing that C is faster than FORTRAN, just take a look at back issues of the Bell Labs Technical Journal. It certainly is under UNIX.

From: DEIMOS::KS 1-MAY-1984 15:09
To: PHOBOS::DLM,VOX
Subj: C and Fortran

An initial guess would be connected with the fact that C does all its floating point arithmetic in double precision, and the code generated by C is full of CVTFD and CVTDF instructions. I raced the C and Fortran compilers some time ago on integer arithmetic - the Eratosthenes sieve - and saw almost no difference.

Keith

From: SL 1-MAY-1984 21:29
To: BCB,VOX

Why should C be faster than FORTRAN in loops when it is possible to have DO WHILE(.TRUE.) loops in VAX FORTRAN? Or is there still some sort of indexing going on in FORTRAN still which slows it down? The VAX manual states that DO WHILE loops are faster than conventional indexed do loops.

From: DLM 1-MAY-1984 22:53
To: SL,VOX
Subj: C

Brian was referring to the ability to address array elements without having to compute array indices each time. This is done with pointers and works best with linear arrays, although it can be done for multi-dimensional arrays. It may be fast, but it appears tedious to me, even in such a simple program as LAXWEN. It would be a bear, I think, in a much more complicated hydro-code. I think KS's suggestion is the "correct" one. C has to work harder to multiply two numbers. The language will probably display its efficiency if double precision arithmetic is needed. I will try compiling LAXWEN.FOR with double precision to see if this assumption is correct. Let me reiterate that more studies of this type are needed to fully assess VMS FORTRAN and C.

From: DLM 1-MAY-1984 23:24
To: KS,BCB,VOX
Subj: C

It looks as though KS is correct. The revised figures with double precision FORTRAN are shown below:

LANGUAGE	LOGOS(VMS 750)	PHOBOS(VMS 780)	HEP(UNIX 780)
FORTRAN (SINGLE)	12.5 (11.6)	3.8 (3.2)	6.0
FORTRAN (DOUBLE)	37.5 (36.6)	6.7 (6.0)	12.6
C	33.3 (32.5)	7.5 (6.7)	6.7

However, if I don't want all that precision, VAX FORTRAN still wins. It is a shame C doesn't have that choice.

From: BCB 2-MAY-1984 14:09
To: VOX,DLM,SL,DEIMOS::KS

A very large speedup can usually be realized by avoiding indexing in C (not possible to avoid it in FORTRAN when using arrays). The trick is illustrated by the two following code fragments:

```
LOW WAY (CRYPTO-FORTRAN)                FAST WAY (REAL C)

ouble  array[ SIZE ];                   double array[ SIZE ], *pa;
ouble  result[ SIZE ];                  double result[ SIZE ], *pr;
egister i;                              register i;

..                                      ...

for(i=0; i<SIZE; i++) {                 for(i=0, pa=array, pr=result;
      result[i]= munge( array[i] );     i<SIZE; i++, pa++, pr++) {
or(i=0; i<SIZE; i++) {                  for(i=0, pa=array, pr=result;
      result[i]= munge( array[i] );     i<SIZE; i++, pa++, pr++) {
      /* munge does something */                *pr = munge( *pa );
      /* time-consuming */                      }
      }
...                                     ...
```

he slow way gets array i by a multiply, add, and indirect chain. The fast way does only an increment and indirect. Obviously, if the loop ontains repeated references to an array, the savings is increased since he increment is done only once (LAXWEN.C contained about ten references to each array in the tightest loop -- every reference entails unnecessary computation and expense).

really smart optimizer might be able to create temporary variables for you, but I doubt the VAX/FORTRAN is that smart; does anyone out there know?

```
rom:   DEIMOS::KS              2-MAY-1984 15:27
To:    PHOBOS::BCB,VOX
ubj:   C and speed
```

I hate to spoil a good theory by actually testing it, but I coded up those examples and looked at the code generated (I really ecommend FOR/LIST/MAC and CC/LIST/MAC to people who worry about his sort of thing) and the code generated by

```
      INTEGER SIZE
      PARAMETER (SIZE=1024)
      DOUBLE PRECISION ARRAY(SIZE)
      DOUBLE PRECISION RESULT(SIZE)
      DO I=1,SIZE
         RESULT(I)=MUNGE(ARRAY(I))
      END DO
      END
      END DO
      END
```

nd

```
#define size 1024
main()

   double array[size];
   double result[size];
   int i;
   for (i=0; i<size; i++) {
       result[i]=munge(array[i]);
       }
```

are almost identical, and do not involve multiplication at all. In fact, he code looks rather nice.

On the other hand

```
#define size 1024
main()

   double array[size], *pa;
   double result[size], *pr;
   int i;
   for (i=0, pa=array, pr=result;
        i<size; i++, pa++, pr++) {
       *pr = munge(*pa) ;
       }
```

enerates pretty horrible code, and deserves to as well, if I might say o. (I have to admit that I've not actually raced them, since I didn't code up a time-consuming MUNGE, so I suppose I could be wrong about the code...)

I think the point is that if you can make it clear enough what you want to be able to do, a GOOD compiler will be able to do it efficiently. If ou make it hard for the compiler to see what's going on, it will not e able to optimise your code.

Keith

From: TJP 2-MAY-1984 17:43
To: VOX
ubj: Optimization

As Keith has pointed out: the VAX Fortran compiler optimizes subscript alculations for one-dimensional arrays very well. It is pretty bad at wo-dimensional arrays though. The VAX C compiler is also an optimizing compiler and will probably do a similar job. I don't know whether the nix compiler optimizes, but I suspect that it will not as it is intended o be more "portable" and less machine-dependent. The VAX architecture includes instructions specifically designed for one-dimensional array to be more "portable" and less machine-dependent. The VAX architecture ncludes instructions specifically designed for one-dimensional array ubscripting; most machines do not.

rom: JLV 2-MAY-1984 17:58
o: VOX,DLM,BCB,
Subj: C vs FORTRAN, and all that jazz

he comparisons of C vs. FORTRAN on the different machines are interesting but a couple of caveat's should be added....

.) A comparison of 780 vs. 750 should be done with the same "process parameters" in each case. Such things as paging do indeed (though theoretically they shouldn't) increase one's CPU usage.

.) Same thing for (TMHW)+ and VMS. >Though this is impossible to achieve. I don't know enough about UNIX internals to know how much a user gets nailed for "system" tasks.

3.) The VMS FORTRAN compiler stores away intermediate values such as ARRAY[I], and does some index calculation at compile time (I believe) when constants are used. This tends to make a "munge" routine with many references to the same location look the same whether pointers or constants are used. This tends to make a "munge" routine with many references to the same location look the same whether pointers or indices are used.

.) The (TMHW)+ FORTRAN-77 compiler produces (or so I am told) notoriously bad code (but it does work...).

In conclusion small differences should probably be ignored, but anything over a actor of 2 is probably significant. Also, why is the 750 so slow? It is eputed to have 60% the speed of a 780. Maybe it's FPA is not working?

rom: MSE 2-MAY-1984 20:06
o: VOX
Subj: C

ow universal is this use of double precision floating point in C? This looks ike a fatal flaw if it can't be undone. We might like to adopt C for the VLBA correlator, but I can't believe we would accept a factor of 2+ degradation n single precision real work.

Of course, the VMS MTH$ routines can be called in any VMS supported precision. aybe that's good enough.

From: TJP 2-MAY-1984 21:29
To: MSE,VOX
ubj: C double precision

It is part of "standard" C that all integer arithemetic is done on ong integers (32 bits on VAX) and all floating-point arithmetic is done n double precision (64 bits on VAX). I know of no compiler which does not follow these rules. Thus using "short int" or "float" variables is only n advantage if you have an awful lot of them and want to save storage space. ne can pass short ints and floats to routines written in other languages, but not to C routines. One wouldn't want to call a subroutine, though, to add two floating point numbers. A typical C program will be speeded up y changing all "short int" declarations to "long int" and all "float" o "double".

From: SL 3-MAY-1984 10:27
To: VOX
Subj: C

Given that the double precision tests (reported to VOX so far) show about equal speed between FORTRAN and C, and the single precision tests showed a clear advantage for FORTRAN, what is the motivation for us to learn and program in C, especially when most people (at least around here) do not know C?

Apparently C has some advantages in character manipulations. However I recently had to modify a whole library of source code to run on a VAX (it was written for a UNIVAC), and this required many character changes -- all of which were pretty easy to do, I found, in an automated way using the VAX run time library calls for character string procedures.

I guess an important factor is that many people are using rainbow computers now, and apparently these are being acquired with C (but not FORTRAN) compilers. I don't think that UNIX is relevant to any of these discussions, because our system management has stated that UNIX will never be the operating system for the Caltech-JPL production programs (i.e. the ones that use a lot of cpu). Although I am teaching myself C, it is hard to be motivated when one sees that after spending a lot of time and effort, the new language is slower than FORTRAN, except when everything is double precision (in which case it is about equal).

From: ZAR 3-MAY-1984 11:04
To: VOX
Subj: Language Comparisons

Has anyone one tested PASCAL vs. FORTRAN or C?

From: TEL 3-MAY-1984 11:43
To: VOX
Subj: C arithmetic conversions

According to Kernighan & Ritchie, the "usual arithmetic conversions" for binary operations are:

First, any operands of type char or short are converted to int, and any of type float are converted to double.
Then, if either operand is double, the other is converted to double and that is the type of the result.
Otherwise, if either operand is long, the other is converted to long and that is the type of the result.

Otherwise, if either operand is unsigned, the other is converted to unsigned and that is the type of the result.
- Otherwise, both operands must be int, and that is the type of the result.
is the type of the result.
Otherwise, if either operand is unsigned, the other is converted to unsigned and that is the type of the result.
Otherwise, both operands must be int, and that is the type of the result.

So, as has been accurately stated earlier in Vox, all floating-point numbers are converted to double precision. However, it is not true that all integer types are converted to long int's, only to int's. While it is true that int's and long int's are both 32-bit quantities on the VAX, there are many machines (most micros, for example) on which these two types are different sizes.

And, by the way, I use C not because of any efficiency consideration, but because I prefer the language - in spite of having used Fortran exclusively for several years before ever hearing of C (or maybe because of this).

- Todd Litwin

From: DLM 3-MAY-1984 13:26
To: TJP,VOX
Subj: UNIX C OPTIMIZATION

Unix provides a -O option which is supposed to produce optimized code. I do not know whether the optimization is VAX-specific or not. On the one hand, UNIX is supposed to be portable. On the other hand, I always thought that it is the C code that is portable and it is in the C compile (read "compiler") where the machine-specific interface takes place. (The UNIX operating system is written in C.) I must ask the HEP people.
(read "compiler") where the machine-specific interface takes place.
The UNIX operating system is written in C.) I must ask the HEP people.

Does anyone know who is the final authority on C? ANSI? Bell Labs? Kernighan & Ritchie? If ANSI, maybe someday we can lobby for single precision arithmetic in C88 or C99. Won't help the VLBA much though.

From: DEIMOS::WALTON 3-MAY-1984 16:44
To: PHOBOS::VOX
Subj: Efficiency etc.

According to Norm Wilson, the HEP UNIX VAX manager, Berkeley claims to have written a Fortran compiler for Berkeley Unix which is as good as the VMS one. This was done by answer analysis; i.e., by looking at the code the VMS compiler generated and rewriting the Berkeley one to produce the same code. HEP does NOT, I believe, have this latest compiler. The VMS Fortran compiler uses tricks to get that last factor of 2 in speed which are highly dependent on the peculiarities of the hardware, and are very likely to contain hard-to-find bugs.

As far as C vs. Fortran vs. Pascal - if you don't need double precision, then Fortran is the clear winner. Before you decide you don't, though, consider carefully the problem at hand. For example, in my thesis precision, then Fortran is the clear winner. Before you decide you don't, though, consider carefully the problem at hand. For example, in my thesis research I was doing radiative transfer calculations using the usual forward-backward sweep method for solving a tridiagonal matrix (Mihalas, Stellar Atmospheres, 2nd edition, chap. 6). This method fails due to single precision floating-point roundoff errors when the minimum optical depth is less than 1.e-5 on the Vax. This value is not atypical for a stellar atmosphere calculation. Moreover, the good structured programming constructs like while loops are part of standard C and Pascal, but not of Fortran-77. Given the itinerant nature of many astronomers nowadays, non-standard Fortran-77 constructions are probably to be avoided.

One very useful tool for efficiency tests is a profiler, a program which (ideally) produces a table of what percentage of your program is spent in which statements. Unix has one; does anyone know of a way to do such a thing in VMS?

From: BCB 3-MAY-1984 16:47
To: VOX,SL
Subj: C vs. FORTRAN

One reason C is preferable to FORTRAN is that it provides data abstraction in the form of recursively defined data structures. This paradigm enables many elegant solutions to difficult problems that are not possible in FORTRAN. As a result, coding time goes down, debugging time goes down, and the code is more readable (because you don't have to do headstands to get simple things done). Any language which provides linguistic constructs that more closely match the structures of the problems at hand is preferable. Linguistically, C is much more "complete" and consistently designed than FORTRAN. It is not the last word on languages, but many agree it is the most usable language currently in existence.

The C Monster

From: DLM 4-MAY-1984 20:10
To: TJP,VOX
Subj: The UNIX C Compiler

The following was sent to me by norman on CITHEP when I asked him whether their UNIX compiler was VAX-specific or portable. His answer helps (somewhat) in explaining why the UNIX C code is a bit faster than VMS C (it does SINGLE precision arithmetic) but not that much faster (it does not know a lot about the VAX architecture). At least that is the way I read it -- DLM:

Obviously any compiler is machine-specific to an extent; the goal of portability is to isolate the dependencies. The C compiler used by almost all UNIX systems except the original PDP-11 version is the so-called `portable C compiler,' which is designed to be easily modified to produce code for different machines. The VAX one has been tuned to some extent, but doesn't know a great deal about the VAX architecture. There's a separate optimizer which knows some quite VAX-specific things, including putting in some of the fancy loop instructions; however, since it's a separate program, and has access only to the assembly languange output of the compiler, there are a lot of optimizations it can't make because it doesn't know if they are safe. (For example, it can't assume that a variable in memory has the same value from one instruction to the next, since it's easy to have many names for the same address from assembly language.)

So both sides are right; the compiler is a generic one, but has been customized to produce code for VAXes (and the optimizer makes no pretense of machine-independence).

The compiler at HEP is slightly divergent, in that it does single-precision floating point ops in single precision (which is a slight violation ot the definition of C, but one the compiler occasionally violated anyway), and allows float and double variables to be in registers. These should be second-order effects.

Dramatis Personae:

DLM David Meier
BCB Brian Beckman
KS Keith Shortridge
SL Steve Lichten
TJP Tim Pearson
JLV Jon Vavrus
MSE Martin Ewing
ZAR Dan Zirin
TEL Todd Litwin
WALTON Steve Walton