Transcript of a GEnie Forth RoundTable Conference with Mike Haas, network specialist and author of JForth, the JSR threaded Forth for Amiga. Mike's topic, "JSR Threaded Forth".

Date: 12/12/91 Time:21:45 EST

Attendees:

<[Mike-H] FIGGUEST> <-- Guest, Mike Haas, author JForth
<[Host SysOp] GARY-S> <-- Moderator SysOp
<[Elliott] ELLIOTT.C>
<[wheels] S.WHEELER>
<[Perry] P.MITCHELL9>
<[Len] NMORGENSTERN>
<[IRV] I.MONTANEZ>
<[Dennis] D.RUFFER>
<[Rob] R.ANDRE>
<[Prez of Vice] JAX>
<[Phil] PLBURK>
<[side] M.CHRISTOPH2>

Minutes:

<[Host SysOp] GARY-S> The GEnie Forth RoundTable is pleased to welcome as tonight's guest, Mike Haas, employed as a Mac networking developer for Starnine and better known to Forthers as author of JSR threaded JForth for Amiga. Mike originally built his own computer (6800-based, 64k, 4 disk dirves, etc..) needed software to run on it, bought the fig-forth listing for $15, and typed it in. Eventually that became his full multi-tasking OS, including cross-compiler, assembler all written in forth.

As the writer of Jforth, Mike has tried to implement many modern concepts (files, creation of small standalone executables (small as 3K!), wanting Forth to become competitiveg in the software development arena.

Please welcome our special guest, Mike Haas.

<[Mike-H] FIGGUEST> hey guys.
first, I'd like to say that I'm not supposed to make this a commercial for JForth.
So, even though I'm going to talk about it, don't anyone BUY IT! :-)
JForth has been around about 5 years, and is currently the only one on the Amiga. It employees (primarily) JSR threading. It also caches the top-of-stack in a 68xxx data register. These two concepts put together some pretty fast code.
Using assembly for the execution code (vs.indirect-threaded stuff) lends itself to some pretty interesting things, such as being able to write optimizers for the compiler.
One of the most interesting & unique features of a JSR threaded forth is that it is a true compiler.It interprets source and ALWAYS builds assembly directly executable by the hardware. The 'inner interpreter' or NEXT-loop is the actual MICRO-CODE of the cpu!

When conventional forths execute their 'programmed steps', they spend some amount of processing just finding the next 'instruction'. (sorry if this is too basic for some, but I'm not real clear on the experience level here). anyway, an indirect-threaded forth (which the great majority are) use several CPU instruction in sequencing through their programs. This is because they are all SIMULATING an execution environment one that is independent of the hardware they are actually running on.
By contrast, a JSR-threaded forth takes advantage of the fact that there is a "sequencer" built-into the CPU itself this 'strips' away that extra level of indirection between the high-level forth code and the actual execution of the CPU instructions (which, after all, is all you ultimately have).

Lets look at a simple short word such as

```
: foo   dup @ ;
```

When this word is entered by a conventional inner interpreter, it has to push on the return stack all the info about the word that called it.
This includes standard forth registers like IP (the place in the calling word we will return to), W (the 'beginning of that particular word) and others.
Then FOO takes over if it doesn't call other high level words (and how many forth words don't do that?), things proceed relatively quick. in between the dup and the @ the process has to 'unwind' all this overhead takes time and doesn't directly contribute to the function of FOO to save a parameter (dup) and then read it (@).
JSR threading comes into play by actually compiling CPU instructions to do the DUP followed by more to do the @, instead of just references to dup & @.
The result is a much faster 'thread' of the program all CPU instructions executed contribute directly to "getting the job done".
Actually, pure JSR threading would put together code something like...

```
JSR   dup
JSR   @
RTS
```

and even the hardware doing all those JSR's and RTS's (which take time since the CPU also has to save it's return addresses) can be improved. This is done in JForth by compiling such short functions completely 'inline'.

As I said at the beginning, JForth caches the top-of-stack in a cpu register (d7) and, like other forths, keeps the remaining stack parameters in memory.
So to DUP an item means simply "write d7 to memory, pre-decremented". For this reason, a DUP in JForth is one cpu instruction...

```
MOVE.L  D7,-(A6)
```

(of course, A6 points to the rest of the 'stack').
Similarly, the @ is also one instruction put the two together...

```
MOVE.L  D7,-(a6)
MOVE.L  0(A4,D7),D7
```

The A4 reference is because JForth keeps it's addresses relative to itself it's base address is in A4. So A4 and D7 are added together, and the CPU reads the data at that location into D7. Add an RTS to the above, and you have the entire code that the JForth compiler will put together

for the above FOO word. No wasted instructions.
To make this even faster, JForth will allow you to compile those twoinstructions INLINE (minus if course the RTS) when a reference to FOO is made.

To illustrate...assume another word...

```
: FOO2  FOO +  ;
```

The JForth compiler will generate...

```
MOVE.L  D7,-(a6)
move.l  0(a4,d7),d7
add.l   (a6)+,d7
```

Notice that items are 'dropped' efficiently via the post-increment mode of the 68xxx family.
Any questions?

<[Host SysOp] GARY-S> Phil Burk has joined us. Phil is co-author of JForth, but wishes to participate only as a spectator at this time.
<[IRV] I.MONTANEZ> Is version 2.0 of JForth the most current?
<[Mike-H] FIGGUEST> Yes, soon 3.0 will be released with many AmigaDOS 2.0 features supported.
<[side] M.CHRISTOPH2> This J forth should be competitibve with C for speed?
<[Mike-H] FIGGUEST> Yes, it is VERY competitive with C for speed...the sieve of eros-u-know-what on a 7.16mhz 6800 finishes in about 8 seconds.
<[IRV] I.MONTANEZ> Have you added any enhancements to the JForth Target Compiler
<[Mike-H] FIGGUEST> CLONE (the target compiler) now creates code about twice as fast as 2.0. It also support interrupt code, allowing to to force BSR's (cause A4 ain't set up) other enhancements.
I meant that CLONE actually runs TWICE as fast itself the code it creates runs about the same speed.
<[wheels] S.WHEELER> A few short (I hope) questions ...
Have you found any Forth constructs which will effectively compile into a DBcc loop? Do you do peephole optimization once code is compiled? How many levels to the optimizer?
<[Mike-H] FIGGUEST> no DBcc is limited to 16-bit loop iterations of course anyone could create their own special DO LOOPs for this purpose, and there's always assembly.
What do you mean by peepholing (not done yet). The optimizer acts to cache MORE than one stack parameter... up to 5 if memory serves me right.
<[Host SysOp] GARY-S> Steve - any follow on peepholing?
<[wheels] S.WHEELER> By peepholing, I mean do you optimize a sequence where one word's code may end in a "drop" and the next's starts with a "dup", and if you don't optimize to "move.l (a6), tos" you have "move.l (a6)+,tos" followed by... "move.l tos,-(a6)".
<[Mike-H] FIGGUEST> yes, all of the standard kernal words do this we call it HEAD-TAIL optimization.
<[Host SysOp] GARY-S> Phil has a comment to add regarding optimization.
<[Phil] PLBURK> Mike already mentioned the HEAD-TAIL, there is a second optimizer that is optional that does global optimization using as many registers as it can get. Once parameters are in registers then words like SWAP and ROT may only have an effect at compile and not actually generate any code. The compiler just shuffles its table to where it keeps the stack items.
<[Mike-H] FIGGUEST> Anoother level of optimization that is in effact all the time is with conditionals. When in assembly, the compiler is not used unless you intentionally invoke it so no optimization is done.
Normally, you are free to use A0 & A1, D0-D4.
<[side] M.CHRISTOPH2> Does this imply that there are no free registers while doing code words? do all regs need to be pusshed?

Not Clear, if the compiler can let the registers dance around you would never know which ones are free. Does the compiler expect the registers to stay correct across other words (code words)?

<[Mike-H] FIGGUEST> No. When a call is made to any other word, the regs are flushed. This is mainly good for long series of short words. Note that the compiler doesn't look for specific sequences of words it will optimize ANY series as long as the words being referenced are known "optimizable".

<[Len] NMORGENSTERN> Do you have a version for other 68xxx machines such as the Mac?

<[Mike-H] FIGGUEST> Not currently. There are other packages available for the Mac, some even are JSR threaded. Phil himself has written one that he includes in th Mac version of HMSL. But no other forth I know of includes the level of optimization and features like CLONE.

<[Host SysOp] GARY-S> Mike, you mentioned several packages to me, why don't you mention a few now, and give folks your phone/address/ e-mail.

<[Mike-H] FIGGUEST> Well, MACH2 comes to mind...Phil has used that extensively. MacForth is not (i believe) jsr threaded, but it's another Mac 4th package.
One PD forth that include object orientation is YERK (mac only, formerly known as NEON).
My e-mail address is mikeh@starnine.com
Delta Research's address (makers of JForth...that's us) is PO Box 1051, San Rafael CA, 94915

<[Host SysOp] GARY-S> No - I meant goodies from Delta besides JForth - don't you have some things that are turn key ?

<[Mike-H] FIGGUEST> I have released several PD products...moist notably TEXTRA, a user-friendly editor that includes an AREXX interface, and LCD CALCULATOR all written in JForth and CLONEd.

<[Phil] PLBURK> Mike mentioned HMSL which is a music programming language based on Forth. The Forth is similar to JForth except that I don't cache Top of Stack in D7 and I use absolute addresses instead of addresses relaitive to the base of Forth. Caching TOS is better and I regret not caching but I prefer absolute addresses, I was just experimenting to see what the differences were like.
If anyone wants other HMSL info, I'm at phil@mills.edu

<[Host SysOp] GARY-S> Mike , give us your closing remarks please

<[Mike-H] FIGGUEST> In JForth, we tried to implement state-of-the-art programming techniques more familiar to users languages such as C. I firmly believe that Forth needs to progress more along these lines...creation of standalone programs, file & memory interfaces, etc. Hopefully, we will move in that direction.
Keeping Forth the BEST environment there is.

<[Host SysOp] GARY-S> Mike (and Phil) thanks for an informative meeting. Many of us are versed primarily in Indirect and some Direct Threaded code, so this was a treat.

Thanks for coming.

<[Host SysOp] GARY-S> This meeting is officially closed.
All may stay and Chat.

=== END ===