Design of a fast 68000-based subroutine threaded FORTH with inline code & an optimiser

Anthony Rose
A.R.Technology
68a Sandown Road
Rondebosch 7700
Cape Town South Africa
Ph. (021) 652184

An examination of several 68000-based FORTH implementations (direct, indirect, token, subroutine threading) showed that up to half the CPU time is spent executing NEXT, DOCOL or other chaining words. Processors such as the 68000 have complex instruction sets which allow the coding of many FORTH primitives in few bytes, sometimes fewer than the alternative JSR to them. ART-FORTH is a full 32-bit subroutine-threaded FORTH which, by compiling words shorter than a dynamically alterable size inline, increases execution speed, as shown below.

Machine	FORTH	Sieve time	#NEXTs	NEXT-time	Fraction of time threading
Macintosh	MacFORTH	3480ms	203682	1430ms	41%
HP9816	MultiFORTH	2330ms	203682	1020ms	44%
HP9816	ART-FORTH				
	unoptimised	1050ms	2	10µs	<1%
	optimised	820ms	2	10µs	<1%

#NEXTs is the number of times NEXT was called during execution of the Sieve. NEXT-time is the time spent executing NEXT and the calls to it.

Since the Sieve uses only primitives, no calls to DOCOL are performed. As more colon definitions and DOCOLs are encountered, the proportion of the time spent threading increases and hence the efficiency decreases. ART-FORTH attempts to reduce this threading overhead by coding short words inline, bypassing NEXT.

A summary of some of the ideas implemented in ART-FORTH follows.

Object code structure

The object code consists of machine code followed by a RTS. Since there are no identifying CFAs, it is essential that any status information be kept with the names, which are stored in a separate vocabulary structure.

The first 64k of object is relocatable. All addresses are referenced relative to a base pointer, BP. Apart from making the kernel relocatable, this facilitates 4 byte address referencing (including the opcode) rather than 6 bytes using 32-bit absolute addressing. Since the 68000 provides only 16-bit signed offsets, BP is set 32k into the object structure, allowing a full 64k of relocatable code. When this limit has been reached, the word-building routines revert to 32-bit absolute addressing. In practice, this appears to increase code size by about 15%, since most references are still to kernel primitives, which, being in the first 64k, take short-form addressing.

Inline Compilation

A subroutine-threaded system forms high level words by compiling a list of JSRs to primitives or to other high-level words. In ART-FORTH, the compiler continually compares the length of the word being compiled with the variable CRITICAL.SIZE. If the word length is less than the critical size, the body of code comprising it, minus the terminating RTS, is

added to the current definition, rather than a JSR. A critical size value of 13 has proven a good compromise between speed and code size. The Sieve benchmark times vary from 1500ms using a CRITICAL.SIZE value of zero (giving a straight subroutine-threaded system), to 1050ms using a CRITICAL.SIZE value of 13 (giving completely inline code). In the case of the Sieve, changing from subroutine calls to inline code had little effect on code size: 166 bytes unoptimised, 144 bytes optimised (with CRITICAL.SIZE=13), versus 154 bytes on MultiFORTH and 82 bytes on MacFORTH.

CONSTANTs, like many other other words, cannot be modified once compiled, as the data appears within the code and is typically coded inline. Q-VARs have been implemented to cater for variable constants.

There are some words which cannot be compiled inline, and others which must be. Unlike an address-threaded system where nesting occurs only on DOCOL, subroutine threading results in calls to primitives, as well as to other high-level words, leaving return addresses on the return stack. All words which reference return stack parameters left by other words (such as >R, R>, RP) must be coded inline. To facilitate this, the name field contains two bits which, when set, cause the compiler to opt for either subroutine or inline code. The words SUBR and INLINE are used like IMMEDIATE, and set the appropriate names-field bits of the latest definition. Words which must be called as subroutines are those with multiple exit points or those which must leave an indication of their data address, such as words created using BUILDS-DOES> or those followed by data, such as dot-quote. Words are therefore compiled as subroutine calls unless: ((word.size<CRITICAL.SIZE)AND(word does not have SUBR bit set)) OR(word has INLINE bit set).

Vocabulary Structure

Since a word consists solely of machine code with no identifying CFA, all information about the word must be kept in the separate names field. In ART-FORTH the names field contains several status bits in addition to the standard precedence and smudge entries.

2 bytes:	relative offset to next nfa in hashed chain
1 byte:	length of object code in 16-bit words, max 255
3 bytes:	address of object code relative to BP
1 byte:	D7-D6 = SUBR & INLINE compiler directives
	D0-D4 = word type: code/colon/variable/constant, etc
1 byte:	D7 = precedence bit
	D3-D5 = # stack words required on entry
	D0-D2 = # stack words left on exit
1 byte:	D5-D7 = vocabulary# (0-7), 0=FORTH, 1=TRANSIENT, etc
	D0-D4 = name count (0-31)
n bytes:	variable length name, terminating on an even address

Two sets of information are maintained for user convenience. VLIST uses the word type field, assigned by a defining word, to display the word type. CREATE resets the word type field to zero, which is displayed as type DATA. Any defining word can then alter the word type bits. For example, CONSTANT sets the field to 3 and VARIABLE to 4. New defining words add their types to the display list with ADD.TYPE (eg: " Q-VAR" ADD.TYPE).

The stack entry and exit fields contain the number of stack items which are taken by the word on entry and left on exit. These are displayed by VLIST. INTERPRET also uses this information to flag an error if there are insufficient items on the stack prior to executing a word.

The definition of left-bracket has been changed to scan comments of the form (a/b/c -- x/y/z | remarks) and examines the stack usage. The default stack usage is set to (0--1) by CREATE,

and this applies to most word types, such as CONSTANT, VARIABLE, etc. Colon (:) sets it to (?--?) so that any words without stack usage comments have question marks in their usage field when VLISTed.

In ART-FORTH, multiple vocabularies exist within a single vocabulary structure. The advantage of this is that only one set of vocabulary headers is needed (names entries are accessed via a 16-bit offset from each of 256 hashed links, so this header would require 512 bytes per vocabulary). In addition, there are now only two dynamic structures - object and names, which simplifies heap management over approaches where each vocabulary is a separate heap item. To facilitate this while providing multiple vocabularies at the user level, a vocabulary number forms part of each name entry. New definitions simply have the contents of CURRENT placed in this field, and vocabularies store their own number into CONTEXT (the definition of FORTH is 0 CONTEXT !). The function of CONTEXT and CURRENT has therefore shifted from holding a pointer to the first item in a vocabulary to holding the vocabulary number.

The vocabulary number is stored in the same byte as the name count, on an even address boundary. Before FIND is called, the contents of CONTEXT are similarly added to the search string's count byte. FIND can then rapidly traverse a hash chain, matching only entries with the same contents & vocabulary number as the search string. Hashing is performed by summing all bytes of the search string mod 256, including the count byte and associated vocab number. This gives an even distribution of entries per hash link, independent of the vocabulary or name length. The ART-FORTH kernel with extensions contains about 1000 definitions; the longest hash chain contains 13 entries.

The Optimiser

At the machine code level, stack usage produces inefficient code. For instance the sequence OVER @ + produces the following code:

Before optimising		After optimising
MOVE.L 4(SP),-(SP)	(OVER)	MOVE.L 4(SP),A0
MOVE.L (SP)+,A0	(@)	
MOVE.L (A0),-(SP)		MOVE.L (A0),D0
MOVE.L (SP)+,D0	(+)	
ADD.L D0,(SP)		ADD.L D0,SP

Note the superfluous push and pop connecting OVER to @, and @ to +. This inefficiency is normally disguised, as the colon definitions contain calls to the primitives rather than inline code. In ART-FORTH a peephole optimiser scans previously compiled colon definitions (at the rate of over one kbyte/second) and removes this and other interface redundancy, reducing code size by about 10-15% and execution time by 20-30%.

The resulting machine code is a factor of two to three slower than the equivalent code written in assembler, due mainly to the use of the stack rather than registers. Typically, 70-80% of the opcodes are MOVE instructions, most of which operate on the stack. An extension of the optimiser is planned that will defer stack operations, keeping operands in registers where possible and restoring them to the stack before a branch or JSR instruction. This optimisation would take place transparently either during or after compilation, as is done at present.

The compiler

Compiler operation is similar to that in a standard FORTH implementation, except that instead of appending the address of the word being compiled onto the dictionary, either a JSR to the word or the code comprising the word is added. Despite the overhead incurred by the conditional inline code, compilation is at the rate of over twenty screens per second (from

blocks in RAM). This rate is achieved by the compiler having itself been compiled to machine code, as well as by tuning of time-critical words in the compile chain. The use of hash links, restructuring of WORD and ENCLOSE, and a BLOCK which returns a value fast (without searching the block buffers) if the block number has not changed since the last BLOCK, reduce the text and vocabulary search time.

Metacompilation

Subroutine threading obviates the need for run-time code such as DOCOL, DOCONST and the ubiquitous NEXT. Consequently, if the kernel can be structured such that few or no forward references exist, then metacompilation is simplified. In ART-FORTH the kernel has been ordered so that only one forward reference, QUIT, exists (apart from I/O, which is vectored anyway). Regeneration (metacompiling using a similar kernel as a host) of a kernel then consists of creating a new vocabulary heap area followed by successive loads of source screens. Since the new kernel is relocatable, it can be loaded on top of the old kernel with appropriate adjustments to BP.

A standard FORTH hosted ART-FORTH during development. The large differences in vocabulary and object code format prevented the simple loading of the new kernel on top the the host possible when both are similar (as in kernel regeneration). The loading of selected source code screens into the host system provides the compatibility bridge: definitions then exist which execute on the host but perform the functions required by the new kernel. This is must be done with care to ensure that new defining words are not inadvertently invoked (by for instance creating a new CONSTANT after CONSTANT has been redefined).

After the new defining words have been loaded and can execute on the host, a new vocabulary area is created and initialised and the primitives for the new kernel are assembled, followed by compilation of colon definitions up to QUIT, at which point the new kernel is invoked and allowed to extend itself.

Conclusion

This necessarily limited article highlights some of the features incorporated in a FORTH implementation which trades code size for speed. High-level compatibility is retained despite the low-level differences between this and standard address-threaded versions. Applications are easily ported between ART-FORTH and other implementations.

ART-FORTH has been developed for a parallel-processing machine, using multiple 68000/68020 processors, currently being developed by A.R.Technology. The ART-1 is to be a loosely-coupled multiprocessor machine with an initial sixteen CPU's linked in a grid array. Each processor board contains four high-speed parallel links (12mbytes/sec on 68000 boards, 20mbytes/sec on 68020 boards) to its north/south/east/west neighbours. Resources such as video displays and mass storage are distributed throughout the system, with each resource typically being connected to one CPU board and accessed by other boards via the data links. A processor-controlled memory access system (analogous to a DMA system) has been designed to allow simultaneous data transfers to any combinations of data links and memory. For instance, a board could receive a data packet from a neighbour to the south while simultaneously broadcasting it to the north, east and west neighbours and writing it to on-board RAM.

The high volume of data being transferred requires fast machine-code routines; the large amount of code comprising these routines warrants a high-level implementation. ART-FORTH satisfies these requirements by facilitating the writing of code in high level with performance within a factor of 2-3 of the equivalent code written in assembler. Multitasking under interrupt control rather than the typical PAUSE round-robin system is also then feasible.