The MPE VFX Forth code generator

Stephen Pelc
MicroProcessor Engineering
133 Hill Lane
Southampton SO15 5AF
England

Tel: +44 1703 631441
Fax: +44 1703 339691
Net: sfp@mpeltd.demon.co.uk
Web: http://www.mpeltd.demon.co.uk

A new code generating system has been developed by MPE for Forth which produces code similar in quality to that produced by many C compilers. This paper discusses the reasons for accepting the additional complexity of native code generation and optimisation, the portability of the code generator, and the results achieved. Comparisons are made with a recently released commercial Forth that claims good performance.

Introduction

Over the years, I have increasingly been told that interpreted languages are slow, and Java has done nothing to reduce this impression. Forth is held to be an interpreted language, but it would be truer to say that it is an interactive language. Rather than have to justify an interpreted solution, MPE decided to produce a Forth system which can compile good quality code, while retaining speed of compilation and full interactivity.

The results justify our investment. The new VFX compiler system is being incorporated into ProForth/VF (the next version of ProForth for Windows) and MPE's new VFX range of Forth cross compilers.

Code generators have been produced or are in development for the following processors, 80386/486/Pentium, Hitachi H8/300H, Motorola 68xxx/Coldfire, and ARM. Other targets are planned.

Several other code generators exist for Forth, but some suffer from being extremely CPU specific, some are not yet finished, others have copyright or other commercial restrictions, and others are not maintainable.

Objectives

The objectives are:
1) good code quality
2) low increase in code size over threaded code
3) portability of the code generator
4) maintainability of the code generator

Good code quality is necessary not only as a sales feature, but also because it reduces the proportion of the Forth kernel that has to be written in assembler, so reducing porting costs.

The requirement for only a small increase in code size is created by customers with previous versions of the MPE Forth cross compilers. They will not be pleased by significant increases in code size. In addition, good code size is important with embedded RISC processors that tend to have small caches.

Good portability of the code generator to new target processors reduces implementation cost, both of the code generator itself, and in the reduction of target code that must be rewritten. In addition it should be easy to update the code generator easily when new features are added.

Maintainability of the code generator affects code reliability, lifetime costs, and ongoing development costs.

Complexity

Compared with the classical threaded Forth compiler, a code generating compiler is completely different in scale. However, every time a new target is created, the conventional code primitives have to be written, tested, and debugged. The tradeoffs are simply whether the resulting additional performace will attract a good price, and will significant cost reductions be available when writing a new target for a cross compiler.

After writing a number of targets, we can answer yes to both questions above.

The figures below show an order of magnitude improvement in PC applications, and about a fourfold improvement over simple subroutine threading and inlining. In addition, the code generation quality is very much better than is achievable with simple methods.

From the customer's point of view, the code quality reduces the amount of assembler code that is needed, and for the embedded system developer, the code quality is perfectly adequate for all but the heaviest interrupt loads.

Portability

Compared to writing a naïve subroutine threaded cross compiler with simple inlining, the MPE VFX code generator takes considerably longer to write, but this is balanced by the portability of the target code.

The VFX code generator is largely CPU independent, but variations in CPU architecture do affect it. Overall, the portability of the code generator is heavily dependent on how aggressive the optimisations are.

Technology

This being a commercial product, the amount of hard information that will be released is necessarily limited.

The code generator is designed as one of number of phases from source to binary code. This topic is covered adequately in the compiler literature.

Although the results below show a considerable improvement over an equivalent commercial offering, there is plenty of scope for further improvement.

Results

The code generation quality of ProForth/VF for Windows was compared with that of a recently released commercial Forth for Windows.

Source:

```
: n>$      \ n -- char ; convert n to ASCII
  dup 9 >
  if  7 +  endif
  $030 +
;
```

Threaded system – 8 cells + 3 lits + EXIT, 48 bytes

MPE ProForth/VFW – 4 instructions plus RET, 16 bytes

```
dasm n>$
( 0043B8CD    83FB09 )                CMP     EBX, 09
( 0043B8D0    0F8E03000000 )          JLE/NG  0043B8D9
( 0043B8D6    83C307 )                ADD     EBX, 07
( 0043B8D9    83C330 )                ADD     EBX, 30
( 0043B8DC    C3 )                    NEXT,
```

FI SwiftForth – 25 instructions + RET, 83 bytes

```
see n>$
BB51EB   4 # EBP SUB                     83ED04
BB51EE   EBX 0 [EBP] MOV                 895D00
BB51F1   4 # EBP SUB                     83ED04
BB51F4   EBX 0 [EBP] MOV                 895D00
BB51F7   9 # EBX MOV                     BB09000000
BB51FC   EBX 0 [EBP] CMP                 395D00
BB51FF   BB5208 JLE                      7E07
BB5201   -1 # EBX MOV                    BBFFFFFFFF
BB5206   BB520A JMP                      EB02
BB5208   EBX EBX SUB                     29DB
BB520A   4 # EBP ADD                     83C504
BB520D   EBX EBX OR                      09DB
BB520F   0 [EBP] EBX MOV                 8B5D00
BB5212   4 [EBP] EBP LEA                 8D6D04
BB5215   BB522C JZ                       0F8411000000
BB521B   4 # EBP SUB                     83ED04
BB521E   EBX 0 [EBP] MOV                 895D00
BB5221   7 # EBX MOV                     BB07000000
BB5226   0 [EBP] EBX ADD                 035D00
BB5229   4 # EBP ADD                     83C504
BB522C   4 # EBP SUB                     83ED04
BB522F   EBX 0 [EBP] MOV                 895D00
BB5232   30 # EBX MOV                    BB30000000
BB5237   0 [EBP] EBX ADD                 035D00
BB523A   4 # EBP ADD                     83C504
BB523D   RET                             C3
```

In order to provide a better basis for comparison, the SwiftForth benchmark suite was compiled for ProForth VFW, and following results were obtained. Note that the SwiftForth compiler is a production compiler, and the VFX compiler is the alpha test release.

Test	Swift 1	Swift 2	Swift 3	OptPFW3 1	OptPFW3 2	OptPFW3 3
Primitives						
Do Loop	0.071	0.07	0.065	0.028	0.027	0.028
*	0.156	0.155	0.16	0.16	0.12	0.136
/	0.405	0.404	0.406	0.544	0.684	0.664
+	0.106	0.1	0.1	0.046	0.054	0.053
M*	0.15	0.15	0.155	0.173	0.255	0.235
M/	0.65	0.65	0.65	0.644	0.574	0.59
M+	0.405	0.415	0.405	0.299	0.229	0.223
/MOD	0.41	0.405	0.405	0.378	0.3	0.42
*/	0.485	0.486	0.485	0.499	0.494	0.492
Benchmarks						
Sieve	3.052	2.442	2.442	0.732	0.854	0.732
Fib Recurse	1.941	1.834	1.963	0.215	0.215	0.194
QSort	24	25	24	10	9	6

The results show that the effect of optimisation is to be found in the test of application code, and not in the raw speed of primitives. The reduced improvement in the QuickSort test is caused by inlining being turned off in the PFW code generator.

Conclusions

The MPE VFX code generator produces code that is between four and ten times faster than the nearest equivalent commercial offering.

The code generator is portable and will be introduced in MPE's ProForth/VF for Windows and in the MPE VFX Forth cross compilers.

There is an increase in development costs which is balanced by the performance results and the increased maintainability and portability of target code.

Acknowledgements

Some of this work has been done under ESPRIT project 25344 PRACTICAL

Papers by and conversations with the following people have been useful:
Anton Ertl, Peter Knaggs, Paul Curtis, Thomas Worthington

Steven ("Harry") Coul conducted the benchmarking and drove us to make this code generator better than we had anticipated.