A Look at Gforth Performance

M. Anton Ertl
TU Wien
New performance features since Gforth 0.5.0 (2000)

- primitive-centric direct threaded code (EuroForth 2001)
- dynamic superinstructions with replication (PLDI 2003)
- static superinstructions (EuroForth 2003)
- automatic build tuning (explicit register allocation)
- workarounds for GCC performance bugs
- branch target alignment
- ...

What is the big picture?
How well do the performance features work relative to others?
How well does it work across machines and GCC versions?
Portability space

- 7 architectures, 12 architecture/CPU combinations
- up to 9 GCC versions per architecture

How well do the performance features do in all variations?

Measurements

- 4 Gforth versions, 7 with options
- 5 application benchmarks (geometric mean reported)
- 3 runs (median reported)
- logarithmic graphs
Overall performance

- Typical speedup factor: 3
- Biggest contribution:
  Dynamic superinstructions in 0.6.2 for Alpha IA32 PPC in 0.7.0 for others
- Automatic tuning (0.7.0) helps IA32
- Multi-state stack caching vs. static superinstructions
- Branch target alignment helps Alpha (0.7.0)
- best performance per cycle: IA32, AMD64
  Reason: indirect branch predictors
Dynamic Superinstructions

Forth                Threaded code                Native code

: foo 5 ;             lit                       mov [esi], ecx
                        mov ecx, [ebx]               mov ecx, [ebx]
5                       add ebx, #4                add ebx, #4
;s                      add esi, #-4              add ebx, #4
                        add ebx, #4                mov ebx, [edi]
                        add ebx, #4                add edi, #4
                        jmp -4[ebx]               add ebx, #4

Engines

- Benchmarking: gforth-fast
- Debugging: gforth
- Error detection and reporting
- Typical difference: factor 2
- Debugging engine:
  - dynamic superinstructions
  - no static superinstructions
  - no multi-state stack caching
  - no automatic tuning
• Gcc ≥ 3.4 disables dynamic superinstructions in gforth 0.6.x
• Gforth 0.7.0 works around that
• gcc-2.95 works well
GCC versions (2)

- PR15242 lowers branch prediction accuracy (gcc-3.4, gcc-4.4.0)
- Gforth 0.7.0 works around that
- Gforth 0.7.0 affected by bad register allocation (4.1, 4.2)
- NEXT expansion (4.4.0)
- gcc-2.95 works well
GCC performance bug: PR15242

Code 1+
( $804B6D8 ) add ebx, #4
( $804B6DB ) inc ebp
( $804B6DC ) mov esi, -4[ebx]
( $804B6DF ) mov eax, esi
( $804B6E1 ) jmp 804AE8C
...
( $804AE8C ) jmp eax

instead of

Code 1+
( $804B6D8 ) add ebx, #4
( $804B6DB ) inc ebp
( $804B6DC ) jmp -4[ebx]
GCC performance bug: NEXT expansion

before_goto:
    goto *real_ca;

is compiled to: instead of:
    mov edx, 58 [esp]    jmp esi
    mov eax, esi
    mov 68 [esp], edx
    mov 6C [esp], edx
    mov 70 [esp], edx
    mov 74 [esp], edx
    jmp eax
Other Forth systems

Benchmark results for IA32 Opteron 270:

- **benchgc4**: Speed 0.7
- **brainless**: Speed 1.0
- **brew**: Speed 0.9
- **cd16sim**: Speed 1.4
- **fcp**: Speed 2.8
- **lexex**: Speed 5.6

Legend:
- iforth
- bigforth
- gforth
- vfxlin
- spf4
Conclusion

• Typical speedup factor: 3

• Most important optimization: dynamic superinstructions
  New gcc versions often disable it ⇒ workarounds

• Important on IA32: Explicit register allocation
  Automatic enabling and testing to get it into Linux distributions

• Other optimizations have small or architecture-specific effect
  But their combination is still significant

• Gforth is very portable
  0.5.0 runs on architectures that were not available on release

• Future work
  Inlining
  Compilation through C (independence from GCC)
  Native code generation