Conclusion

Vcode: DCG system for building systems

- fast + portable + expressive + real
- interface = assembly language for idealized RISC
- implementation = set of macros to emit code
- Used to build real systems
- Can give 10x performance improvement

Vcode lives at:

http://www.pdos.lcs.mit.edu/~engler/

Experience

In use for ~1.5 years (e.g., in Aegis OS)

Low-level interface good for building systems:

- **Allows optimizations not possible otherwise**
- **Can be implemented simply (bugs have been rare)**

Drawbacks:

- **Interface challenging for normal apps (use ‘C!)**

Compilation target for ‘C

‘C (Tick-C) [POPL 96, WCSSS ‘96]:

- Extends ANSI C to support DCG
- Provides easy to use DCG

Implementation: extended lcc to generate vcode

Result: on 10 benchmarks, 1.2 - 10 x speedup

ASH network system

Problem:

- **Network protocols touch messages** (e.g., checksum)
- **Compose protocols: message touched many times**

  (Costly!)

Solution [SIGCOMM '96]:

- **Dynamically compile into single pass over memory**
- **Exploit low-level vcode interface**

Result: Up to 3 x higher throughput than hand integrated

DPF: dynamic packet filter compiler

Packet filter is a kernel extension:

- Boolean expression applied to each network packet
- Determines who a packet is for

Problem: interpreted (expensive!)

DPF system [SIGCOMM ‘96]:

- Compile filters at runtime using vcode

Result: 10-20 x faster than best numbers in literature

Good code without global information

Problems:

- **Is the function a leaf?**

  (Need to perform good register allocation)

- **How many locals were allocated?**

  (Need to determine stack size, AR layout

- **How many callee saved registers used?**

  (Need to generate prologue/epilogue code)

General solution: reserve space + fix up later

Portable DCG as fast as non-portable

Two reasons transliteration can be done with vcode:

1. Vcode designed to map to modern hardware
2. Clients perform register allocation

Key effect:

- Doesn't need to build and consume IR at runtime

Key challenge:

- Code generation with only local information

Code generation at preprocess time

Given a macro to emit MIPS addu instruction:

- #define addu(rd, rs1, rs2) \

  (rs1 << 21 | rs2 << 16 | dst << 11 | 0x21)

We want to map the following vcode insn to it:

v_addu(rd, rs1, rs2)

This requires a trivial relabeling:

#define v_addi(rd, rs1, rs2) addu(rd, rs1, rs2)

Result: portable DCG is as fast as non-portable DCG

Problem: translation without taxation

Non-portable (fast) DCG systems:

- Use machine-specific macros to generate code

Portable (slow) DCG systems:

- Build and consume IR at runtime

Vcode:

- Replace IR translation with string substitution

Problem: translation without taxation

```
V_iptr mk_plus1(void) {
    v_reg_t x;              // input register

    // start dynamic code gen
    v_lambda("plus1", "%x", &x);
        v_addi(x, x, 1); // Add Int Imm
        v_reti(x);          // Return Int
    // Return pointer to generated code
    return v_end();
}
```

Recipe for Bits

Operational overview:

1. Client calls vcode to begin code generation
2. Client uses vcode macros to generate code
3. Client calls vcode to end code generation
4. vcode performs cleanup (link, cache flush, etc.)
5. Returns pointer to generated code to client
6. Client can then call the new function

Interface: assembly for a virtual RISC

Example instructions:

- **memory operations:** load, store
- **ALU operations:** add, xor
- **control flow:** jump,branch not equal

Composed with vcode types:

- i (int), u (unsigned), d (double), v (void), etc

“Add Integer” = v_addi(rd, rs1, rs2)

System overview

vcode:

- interface = assembly language for idealized RISC
- implementation = macros to emit code
- ~5 - 10 instruction per generated instruction
- runs on MIPS, SPARC, Alpha, .5 x86

Main technical challenge:

- portability without code generation cost

Vcode: DCG for building systems

- **Fast + Portable:**

  Generates code as fast as non-portable methods
  Clients can get 10x speedup

- **Expressive:**

  Clients can write code not expressible in HLL

- **Real:**

  Publicly available, used by real systems

- **No special compiler support**

  Can be used with different dialects/languages

The Problem

Many systems can profitably exploit DCG:

- **Interpreters** (Self, Smalltalk, Java)
- **Operating Systems** (Synthesis, Spin, exokernel)
- **Compilers** (‘C, C-Mix, Tempo)

But: no good DCG system exists for building systems

Need: Fast + Portable + Expressive + Real

VCODE: Fast, Simple Dynamic Code Generation

Dawson Engler
MIT Lab for Computer Science

http://www.pdos.lcs.mit.edu/~engler