‘C: A Language for Fast Dynamic Code Generation

Dawson Engler
Frans Kaashoek
MIT Lab for Computer Science

Wilson Hsieh
University of Washington

What & Why

Old insight:

- More information => better optimization
- Runtime = information bonanza
- Generate code at runtime!

  “dynamic code generation” (DCG)

Result: Order of magnitude performance improvement

DCG For Speed and Simplicity

Interpreters compile code “on-the-fly”:

- Smalltalk-80 [Deutsch et al. 1984]
- Self [Chambers et al. 1989]

Specialization:

- Synthesis [Massalin et al. 1988]
- Dynamic partial evaluation [Leone et al. 1994]

Simplicity:

- Constant folding [Thompson]

Problems With Available Methods

Want: portability + usability + efficiency

Methods:

- Generate C code, farm out to compiler and link (slow)
- Lisp-like eval (slow)
- Generate bits “by hand” (very non-portable)
- Portable DCG system (low-level)

Our solution: the ‘C Language

Extend ANSI C to support dynamic code generation

Portability:

- Machine-independent language

Usability:

- Familiar, widely-used language

Efficient code generation:

- Static typing moves most DCG costs to compile time

Keys:Language + RTS + Static types

Language support for DCG:

- DCG code written in same language
- Can support profiling and debugging

Runtime system (RTS):

- Minimize language changes needed

Static typing:

- Minimize effect on language (ANSI C)
- Allows DCG support in widely used languages

Operational Overview

Edit time:

1. Programmer uses ‘C constructs to specify code

Compile time:

1. Emit code to generate specified code at runtime

Runtime:

1. Application builds & composes code specs
2. ‘C runtime system generates dynamic code
3. Dynamic code can be called as a function

Language Overview

Two type constructors:

- cspec's (code spec): specify exprs or stmts
- vspec's (variable spec): specify lvalues

Three operators:

- Tick (‘): applied to exprs and stmts to create cspec's
- At (@): compose cspec's and vspec's
- Dollar ($): bind runtime constants in cspec

Tick (‘) Operator: Create cspec’s

Apply ‘ to expression or statement to create cspec’s:

- cspec for integer 4:

  ‘4 /* type is “int cspec” */

- cspec for greeting:

  ‘printf(“hello world\n”) /* type is “int cspec” */

- cspec for compound statement:

  ‘ { int i; i = 3; foo(i); } /* type is “void cspec” */

At (@) Operator: Composition

Used within tick-expression to compose vspec/cspec's:

- **Compose cspec's c1 and c2, evaluation yields "9":**

```
int cspec c1, cspec c2;

c1 = ‘4;
c2 = ‘5;
/* compose, compile, and run: will return “9” */
compile(‘{ return @c1 + @c2;})();
```

Dollar ($) Operator: Binding

Used to resolve ambiguity: bind at spec time or runtime?

Applied to expressions within a ‘ to bind runtime const:

```
int x, cspec c1;

x = 9;
c1 = ‘$x; /* bind value of x at this point */
x = 13;
/* will print “c1 = 9” */
compile(‘{ printf(“c1 = %d\n”, @c1)})();
```

Static Typing: Simple, Efficient DCG

All types known at compile time:

- Can most intra-cspec code generation statically

  (Instruction selection, peephole optimization, etc.)

What's left at runtime:

- Specialization based on runtime constants
- Inter-cspec optimizations
- Emit the actual bits

Experiment: “Compiling” Interpreter

Small C-like language for interactive use

Interpreter written in ‘C:

- Code is entered, compiled on-the-fly, then run
- Example, fibonacci:

    Within factor of 2 of “GCC -O2”
    50 x faster than simple interpreter

Speed from elimination of interpretation overhead

Simplicity through DCG: no need to write evaluator

tcc: a Sophisticated ‘C Compiler

Added ‘C support to the lcc compiler:

- Emits templates that generate code quickly

  (10 x faster than compiler described in paper)

- Based on vcode virtual machine [Engler, PLDI 96]
- Beta-release in next few months

Future work:

- Add debugging and profiling support

Experience

Main challenge:

- **Efficiently handling functions whose number & type of arguments/locals are unknown at static compile time**

  (Needed for compiling interpreters)

Lessons learned:

- **Language has more power than designed (good)**
- **Perhaps a too-heavy reliance on RTS than language**

Related Work

Eval: (Lisp, perl)

- Easy to use, high-level
- Expensive

Dynamic partial evaluation (FABIUS [Leone & Lee]):

- Easy to use, automatic
- Less powerful than direct access to DCG primitives

Conclusions

Use of DCG can give 10x performance improvement

‘C: five extensions to ANSI C to support DCG

‘C is fast, portable, high-level:

- Static typing = fast
- C = portable + easy + widely used

Goal: move DCG from curiosity to much-used technique