SACK: a Semantic Automated Compiler Kit

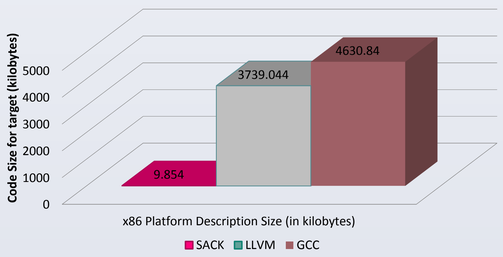
Colby T. Skeggs

Applications

- Rapid architecture testing
- Automatically-generated architectures:
  - For security
  - For optimization
- Dynamic binary translation
- Automated decompilation
- Hobbyist/esoteric architectures

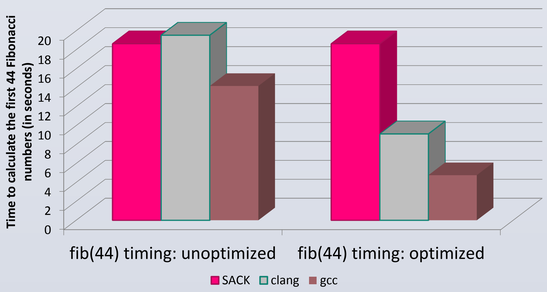
Code Size Comparison (x86 target)

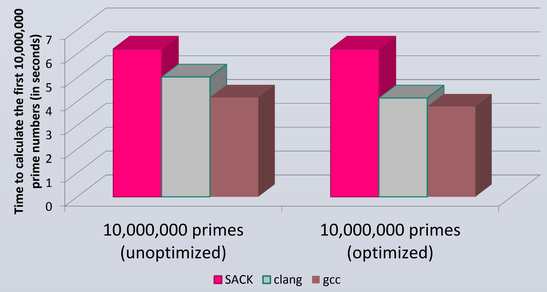
SACK supports targets with a tiny fraction of the target-specific code needed by other systems.



Efficiency Comparison (x86 target)

SACK compares reasonably well with invocations of other major compilers if they have optimizations turned off.





The Problem

- New architectures are important for innovation, but adoption requires working compilers, which are difficult to write:
  - significant time
  - high cost
  - lots of effort
  - lots of code
- Other retargetable compilers are used, but
  - are insufficient
  - require lots of code and effort
  - tend to require *human* work

The Goal

- Prototype a system to make retargeting easy
  - less than a day to retarget
  - low cost
  - minimal effort
  - short code
- Produce reasonably-fast code
  - does not have to be competitively fast
- Support standard imperative features
  - similarity to the C programming language

The Solution

SACK is a compiler toolkit that compiles code for a platform based on a description of the platform.

- SACK starts with a platform description that
  - is written in a custom domain-specific language
  - describes semantics/behavior of instructions
  - is used to convert from source code to platform assembly language
- SACK uses the platform description to generate tree rewriting rules that
  - are derived from instruction behavior
  - determine where the instruction should be used
- SACK has been tested with x86, JVM, and a custom architecture:
  - the JVM target was implemented in 85 minutes
  - the x86 target is similar in speed to unoptimized clang/gcc

Sample instruction descriptions from the x86 platform description file

```
[instruction      ; The x86 'add' instruction with two registers.
  (x86/add/dd (dest any?) (source any?))              ; The identifier and arguments
  ("  add " dest ", " source)                        ; The output assembly template
  (set-reg dest (+ (get-reg dest) (get-reg source)))]  ; The instruction behavior

[instruction      ; The x86 'mov' instruction with a relative memory reference.
  (x86/movfm/d+c (dest any?) (source any?) (offset const?))
  ("  mov " dest ", [" source "+" offset "]")
  (set-reg dest (get-memory (+ (get-reg source) offset)))]
```

Sample rule conversions for the above instructions

```
; For x86 add instruction.
(set-reg dest (+ (get-reg dest) (get-reg source)))  ; instruction behavior, from above

(+ dest source)                                     ; generated pattern to match
     => (x86/add/dd dest source)                    ; generated replacement

; For x86 mov from memory with offset instruction.
(set-reg dest (get-memory (+ (get-reg source) offset)))

(get-memory (+ source offset))                      ; demonstrates multiple operations
     => (x86/movfm/d+c dest source offset)          ; being matched at once
```

Dataflow Pipeline