A Code Size Microbenchmark for C

Yang Chen, John Regehr
School of Computing, University of Utah
{chenyang, regehr}@cs.utah.edu
http://embed.cs.utah.edu/embarrassing

Test Input

```c
int pmat (int m, int n, double *y) {
    int i, j, k;
    k = 0;
    i = 0;
    while (i < m) {
        j = 0;
        while (j < n) {
            k++;
            j++;
        }
        i++;
    }
    return (0);
}
```

LLVM-GCC Output

```assembly
Disassembly of section .text:
00000000 <pmat>:
  0: 8b 44 24 04 mov 0x4(%esp),%eax
  4: 85 c0 test %eax,%eax
  6: 7e 0f jle 17 <pmat+0x17>
  8: 8b 4c 24 08 mov 0x8(%esp),%ecx
  c: 85 c9 test %ecx,%ecx
  e: 7f 07 jg 17 <pmat+0x17>
  10: 31 c9 xor %ecx,%ecx
  12: 41 inc %ecx
  13: 39 c1 cmp %eax,%ecx
  15: 7c fb jl 12 <pmat+0x12>
  17: 31 c0 xor %eax,%eax
  19: c3 ret
```

GCC Output

```assembly
Disassembly of section .text:
00000000 <pmat>:
  0: 31 c0 xor %eax,%eax
  2: c3 ret
```

Compilation Options

```
llvm-gcc -Os -mfpmath=sse -msse3 -march=core2 -mtune=core2 -fomit-frame-pointer -fno-stack-protector -w -c pmat.c
```

```
gcc -Os -mfpmath=sse -msse3 -march=core2 -mtune=core2 -fno-stack-protector -fomit-frame-pointer -w -c pmat.c
```

Motivation

- No compilers could always generate smaller code than others
- Hand-optimized code is often hard to understand
- Programmers tend to write readable code and trust compilers generate fast and compact code for them
- There is a plenty of room for improving compiler optimizations

Our goal: help compiler developers improve their products by giving them actionable test cases that pinpoint missed optimizations

Reality to Compilers

- Compilers need tradeoff between the degree of optimizations and the amount of time spent in optimizing code
- Compilers suffer phase-ordering problems where the quality of generated code depends on the order of executed optimizations
- Compilers have to balance the machine-dependent and machine-independent features

Microbenchmark

- billions of lines of C code
  - linux kernel, FreeBSD, CPU 2000, etc
  - written in CIL
  - extracts small functions from open source projects written in C
  - performs our customized inlininer and cleaner passes
  - harvests small functions with respect to their AST sizes, and put them into separate files with the minimal sets of declarations required for it to compile
  - computes a checksum over the AST of each harvested function, and only keep one from each group of functions with the same checksum

Harvester

Test on a variety of compilers

```c
gcc, llvm-gcc, clang, etc
```

Experimental Results

- Fig 1: median code size for each compiler. The median_min and median_max bar shows the median size of the smallest and largest code produced by any compiler, respectively.
- Fig 2: how much of the time each compiler produces the smallest object code