FDO:
Magic "Make My Program Faster" compilation option?

ARM

Paweł Moll

Embedded Linux Conference Europe, Berlin, October 2016

Agenda

- FDO Basics
- Instrumentation based FDO
- Sample based (“Auto”) FDO
- Deployments

TLAs

- FDO: Feedback Directed Optimisation
- FDO: Feedback Driven Optimisation
- PGO: Profile Guided Optimisation
- PDF: Profile Directed Feedback
- PFO: Profile Feedback Optimisation

Decisions to be made

- Compiler has to make number of decisions
  - Is "`then`" more probable than "`else`"?
  - Is a function worth inlining here?
  - Should I unroll this loop?
- Questions get down to branch probability assessment
  - Usually estimated by a number of heuristics
- The decision making process can be influenced by the programmer
  - Fortran's FREQUENCY hints for basic blocks Monte Carlo simulation
  - GCC's `__builtin_expect()` function, used by `likely()` and `unlikely()` macros in the Linux kernel
  - *"(…) programmers are notoriously bad at predicting how their programs actually perform."*
- An obvious idea is to capture such data automatically
  - Measuring frequency of branches (not)taken during real workload execution

Programmer's
Reference Manual

October 15, 1956

THE FORTRAN AUTOMATIC CODING SYSTEM FOR THE IBM 704 EDPM®

This manual supersedes all earlier information about the FORTRAN system. It describes the system which will be made available during late 1956, and is intended to permit planning and FORTRAN coding in advance of that time. An Introductory Programmer's Manual and an Operator's Manual will also be issued.

APPLIED SCIENCE DIVISION
AND PROGRAMMING RESEARCH DEPT.
International Business Machines Corporation
590 Madison Ave., New York 22, N. Y.

WORKING COMMITTEE

J. W. BACKUS
R. J. BEEBER
S. BEST
R. GOLDBERG
H. L. HERRICK
R. A. HUGHES
University of California
Radiation Laboratory,
Livermore, Calif.
L. B. MITCHELL
R. A. NELSON
R. NUTT
United Aircraft Corp.,
East Hartford, Conn.
D. SAYRE
P. B. SHERIDAN
H. STERN
I. ZILLER

3. Execution of a DO will in general store a new value of the index. (It will not always do so, however; see the section on Further Details about DO Statements in Chapter 7.)

4. Execution of a READ, READ INPUT TAPE, READ TAPE, or READ DRUM stores new values of the variables listed.

FREQUENCY

GENERAL FORM	EXAMPLES
"FREQUENCY n(i, j, ...), m(k, l, ...), ..." where n, m, ... are statement numbers and i, j, k, l, ... are unsigned fixed point constants.	FREQUENCY 30(1, 2, 1), 40(11), 50(1, 7, 1, 1)

The FREQUENCY statement permits the programmer to give his estimate, for each branch-point of control, of the frequencies with which the several branches will actually be executed in the object program. This information is used to optimise the use of index registers in the object program.

A FREQUENCY statement may be placed anywhere in the source program, and may be used to give the frequency information about any number of branch-points. For each branch-point the information consists of the statement number of the statement causing the branch, followed by parenthesis enclosing the estimated frequencies separated by commas.

Consider the example. This might be a FREQUENCY statement in a program in which statement 30 is an IF, 40 is a DO, and 50 is a computed GO TO. The programmer estimates that the argument of the IF is as likely to be zero as non-zero, and when it is non-zero it is as likely to be negative as positive. The DO statement at 40 is presumably one for which at least one of the indexing parameters (m's) is not a constant but a variable, so that the number of times the loop must be executed to make a normal exit is not known in advance; the programmer here estimates that 11 is a good average for that number. The computed GO TO at 50 is estimated to transfer to its four branches with frequencies 1, 7, 1, 1.

All frequency estimates, except those about DOs, are *relative;* thus they can be multiplied by any constant. The example statement, for instance, could equally well be given as FREQUENCY 30(2,4,2), 40(11), 50(3,21,3,3). A frequency may be estimated as 0; this will be taken to mean that the frequency is very small.

The following table lists the 8 types of statement about which frequency information may be given.

Example code

```
#define ARRAY_SIZE(_a) (sizeof(_a) / sizeof((_a)[0]))
#include "bubble.h" /* array of 30000 integers in random order */
int main(void) {
    int done, i;
    do {
        done = 1;
        for (i = 0; i < ARRAY_SIZE(a) - 1; i++) {
            if (a[i] > a[i + 1]) {
                int t = a[i];
                a[i] = a[i + 1];
                a[i + 1] = t;
                done = 0;
            }
        }
    } while (!done);
    return 0;
}
```

Instrumentation based FDO

- Classic approach, available both in gcc and LLVM
- Compile a program with additional, profiling code injected by the compiler

```
$ gcc bubble.c -g -O3 -fprofile-generate \
                                    -o bubble-O3-profile-generate
```

- Run the instrumented program, generating profile

```
$ ./bubble-O3-profile-generate
$ ls *.gcda
bubble.gcda
```

- Compile the program again, using the profile

```
$ gcc bubble.c -g -O3 -fprofile-use -o bubble-O3-profile-use
```

gcc 4.8 -O3

```
        mov     w0, #0x0
        mov     w6, #29998
        cmp     w0, w6
        adrp    x2, _G_O_T+0x28
        add     w1, w0, #0x1
        mov     w7, #0x1
        add     x2, x2, #0x30
        sbfiz   x4, x0, #2, #32
        sbfiz   x3, x1, #2, #32
        b.hi    while
if:     ldr     w0, [x2,x4]
        ldr     w5, [x2,x3]
        cmp     w0, w5
        b.le    lesseq
        str     w5, [x2,x4]
        str     w0, [x2,x3]
        mov     w7, #0x0
```

```
lesseq: mov     w0, w1
for:    cmp     w0, w6
        add     w1, w0, #0x1
        sbfiz   x4, x0, #2, #32
        sbfiz   x3, x1, #2, #32
        b.ls    if
while:  mov     w1, w7
        cbnz    w7, return
        mov     w7, #0x1
        mov     w0, w1
        b       for
return: mov     w0, #0x0
        ret
```

```
if (a[i] > a[i + 1]) {
    int t = a[i];
    a[i] = a[i + 1];
    a[i + 1] = t;
    done = 0;
}
```

gcc 4.8 -O3 -fprofile-generate

```
stp     x29, x30, [sp,#-32]!
adrp    x2, __gcov_i_c_c
mov     x29, sp
str     x19, [sp,#16]
mrs     x19, tpidr_el0
add     x19, x19, #0x0, lsl #12
add     x19, x19, #0x10
mov     x1, #0x0
add     x2, x2, #0xd60
ldr     x0, [x19]
ldr     x3, [x19,#8]
bl      __gcov_i_c_p
adrp    x11, a+0x1cf00
add     x0, x11, #0x670
mov     w7, #29998
str     xzr, [x19,#8]
ldr     x6, [x0,#8]
ldr     x10, [x0,#24]
mov     w0, #0x0
cmp     w0, w7
```

```
adrp    x2, _G_O_T_+0x48
add     w1, w0, #0x1
ldr     x8, [x11,#1648]
mov     w9, #0x1
add     x2, x2, #0x100
sbfiz   x4, x0, #2, #32
sbfiz   x3, x1, #2, #32
b.hi    main+0xac
ldr     w0, [x2,x4]
ldr     w5, [x2,x3]
add     x6, x6, #0x1
cmp     w0, w5
b.le    main+0x94
str     w5, [x2,x4]
str     w0, [x2,x3]
add     x8, x8, #0x1
mov     w9, #0x0
mov     w0, w1
cmp     w0, w7
add     w1, w0, #0x1
```

```
sbfiz   x4, x0, #2, #32
sbfiz   x3, x1, #2, #32
b.ls    400dd0
cbnz    w9, 400e24
mov     w1, w9
add     x10, x10, #0x1
mov     w9, #0x1
mov     w0, w1
b       400df8 <main+0x98>
add     x1, x11, #0x670
mov     w0, #0x0
ldr     x19, [sp,#16]
ldr     x2, [x1,#16]
str     x6, [x1,#8]
add     x2, x2, #0x1
str     x10, [x1,#24]
str     x2, [x1,#16]
str     x8, [x11,#1648]
ldp     x29, x30, [sp],#32
ret
```

gcc 4.8 -O3 -fprofile-use

```
        mov     w9, #0x0
        mov     w6, #29998
        cmp     w9, w6
        adrp    x2, _G_O_T+0x28
        add     w1, w9, #0x1
        mov     w7, #0x1
        add     x8, x2, #0x30
        sbfiz   x4, x9, #2, #32
        sbfiz   x3, x1, #2, #32
        b.hi    while
if:     ldr     w0, [x8,x4]
        ldr     w5, [x8,x3]
        cmp     w0, w5
        b.gt    then
        mov     w9, w1
for:    cmp     w9, w6
        add     w1, w9, #0x1
        sbfiz   x4, x9, #2, #32
        sbfiz   x3, x1, #2, #32
        b.ls    if
while:  cbnz    w7, return
        mov     w1, w7
        mov     w9, w1
        mov     w7, #0x1
        b       for
then:   str     w5, [x8,x4]
        str     w0, [x8,x3]
        mov     w7, #0x0
        mov     w9, w1
        b       for
return: mov     w0, #0x0
        ret
```

gcc-4.8 results

metric	-O3	-O3 -fprofile-generate	-O3 -fprofile-use
time elapsed	**3.306690054 s**	**3.382299600 s (+2.3% vs -O3)**	**3.422646478 s (+3.5% vs -O3)**
cycles	**6,612,612,325**	**6,763,814,485 (+2.3% vs -O3)**	**6,844,522,764 (+3.5% vs -O3)**
instructions	9,599,581,077	10,716,296,612 (+11.1% vs -O3)	9,823,874,803 (+2.3% vs -O3)
IPC	1.45	1.58	1.44

Cortex-A57

gcc 6.1 -O3

```
        adrp    x5, __F_E__+0xfa10
do:     add     x0, x5, #0x830
        mov     w4, #0x1
        add     x3, x0, #0x1d, lsl #12
        add     x3, x3, #0x4bc
for:    ldp     w1, w2, [x0]
        cmp     w1, w2
        b.le    lesseq
        mov     w4, #0x0
        stp     w2, w1, [x0]
lesseq: add     x0, x0, #0x4
        cmp     x0, x3
        b.ne    for
        cbz     w4, do
        mov     w0, #0x0
        ret
```

gcc 6.1 -O3 -fprofile-generate

```
stp     x29, x30, [sp,#-32]!
adrp    x1, __gcov_i_c_c+0x3ffff8
add     x1, x1, #0xd60
mov     x0, #29419
mov     x29, sp
movk    x0, #0x670, lsl #16
stp     x19, x20, [sp,#16]
adrp    x19, a+0x1c810
bl      __gcov_i_c_p_v2
add     x20, x19, #0xd90
adrp    x1, __F_E+0xfb60
ldr     x1, [x1,#1704]
mrs     x2, tpidr_el0
mov     x0, x20
str     xzr, [x2,x1]
bl      __gcov_t_p
ldp     x4, x8, [x20,#16]
mov     w12, #0x0
ldr     x7, [x20,#40]
mov     w6, #0x0
adrp    x10, __F_E+0xfb60
mov     x9, #29999
add     x0, x10, #0x7f0
mov     x11, x7
add     x3, x0, #0x1d, lsl #12
mov     w5, #0x1
add     x3, x3, #0x4bc
ldp     w1, w2, [x0]
cmp     w1, w2
b.le    main+0x88
add     x4, x4, #0x1
mov     w6, #0x1
mov     w5, #0x0
stp     w2, w1, [x0]
add     x0, x0, #0x4
cmp     x0, x3
b.ne    main+0x6c
add     x8, x8, x9  // #29999
add     x7, x7, #0x1
cbnz    w5, main+0xa8
mov     w12, #0x1
b       main+0x58
add     x0, x19, #0xd90
str     x8, [x0,#24]
cbnz    w6, main+0xd8
cbnz    w12, main+0xe0
add     x1, x19, #0xd90
mov     w0, #0x0
ldp     x19, x20, [sp,#16]
ldr     x2, [x1,#32]
add     x2, x2, #0x1
str     x2, [x1,#32]
ldp     x29, x30, [sp],#32
ret
str     x4, [x0,#16]
b       main+0xb4
add     x0, x19, #0xd90
str     x11, [x0,#40]
b       main+0xb8
```

gcc 6.1 -O3 -fprofile-use

```
adrp    x6, __F_E+0xf8f8
add     x0, x6, #0x950
ldr     w1, [x6,#2384]
add     x5, x0, #0x1d,\
              lsl #12
mov     w4, #0x1
add     x5, x5, #0x4bc
ldr     w2, [x0,#4]
cmp     w1, w2
b.le    main+0x30
str     w2, [x6,#2384]
mov     w4, #0x0
str     w1, [x0,#4]
add     x7, x0, #0x4
ldr     w8, [x0,#4]
ldr     w3, [x7,#4]
cmp     w8, w3
b.le    main+0x50
str     w3, [x0,#4]
mov     w4, #0x0
str     w8, [x7,#4]
add     x14, x7, #0x4
b       main+0xcc
```

```
add     x11, x14, #0x4
ldr     w13, [x14,#4]
ldr     w12, [x11,#4]
cmp     w13, w12
b.gt    main+0x138
ldp     w14, w15, [x11,#4]
cmp     w14, w15
b.gt    main+0x12c
ldp     w16, w17, [x11,#8]
cmp     w16, w17
b.gt    main+0x120
ldp     w18, w0, [x11,#12]
cmp     w18, w0
b.gt    main+0x114
ldp     w1, w2, [x11,#16]
cmp     w1, w2
b.gt    main+0x108
ldp     w7, w8, [x11,#20]
cmp     w7, w8
b.gt    main+0xfc
ldp     w9, w3, [x11,#24]
cmp     w9, w3
b.gt    main+0xf0
```

```
ldp     w10, w12, [x11,#28]
cmp     w10, w12
b.gt    main+0xe4
add     x14, x11, #0x20
cmp     x14, x5
b.eq    main+0x148
ldp     w9, w10, [x14]
cmp     w9, w10
b.le    main+0x58
mov     w4, #0x0
stp     w10, w9, [x14]
b       main+0x58
mov     w4, #0x0
stp     w12, w10, [x11,#28]
b       main+0xc0
mov     w4, #0x0
stp     w3, w9, [x11,#24]
b       main+0xb4
mov     w4, #0x0
stp     w8, w7, [x11,#20]
b       main+0xa8
mov     w4, #0x0
stp     w2, w1, [x11,#16]
```

```
b       main+0x9c
mov     w4, #0x0
stp     w0, w18, [x11,#12]
b       main+0x90
mov     w4, #0x0
stp     w17, w16, [x11,#8]
b       main+0x84
mov     w4, #0x0
stp     w15, w14, [x11,#4]
b       main+0x78
str     w12, [x14,#4]
mov     w4, #0x0
str     w13, [x11,#4]
b       main+0x6c
cbz     w4, main+0x4
mov     w0, #0x0
ret
```

gcc-6.1 results

metric	-O3	-O3 -fprofile-generate	-O3 -fprofile-use
time elapsed	**3.268757833 s** *(-1.1% vs 4.8)*	**3.372646410 s (+3.1% vs -O3)**	**2.504173270 s (-23.4% vs -O3)**
cycles	**6,536,735,848** *(-1.1% vs 4.8)*	**6,744,497,117 (+3.1% vs -O3)**	**5,007,557,329 (-23.4% vs -O3)**
instructions	5,806,220,662 *(-39.5% vs 4.8)*	6,254,942,732 (+7.7% vs -O3)	3,873,453,819 (-33.3% vs -O3)
IPC	0.89	0.93	0.77

Cortex-A57

Challenges with instrumentation based FDO

- Training data generation
  - SPEC2006 benchmark suite ships with carefully researched dataset
  - "Evaluating whether the training data provided for profile feedback is a realistic control flow for the real workload" paper
- Substantial profile generation overhead
  - 16% on average for SPECint2006 quoted
  - But observed up to 100 times slowdown on particular workloads
- Requires two-stage build, interleaved with a training run

Sample based AutoFDO

- Introduced in "Feedback-Directed Optimizations in GCC with Estimated Edge Profiles from Hardware Event Sampling" paper from 2008, available upstream in gcc since 5.1 and LLVM since 3.5
- Compile a program as normal

```
$ gcc bubble.c -g -O3 -o bubble-O3
```

- Run the program as normal, capturing profile using standard Linux `perf` tool

```
$ perf record -b bubble-O3
```

- Convert `perf.data` into a profile using the `autofdo` tool (available on github)

```
$ create_gcov --binary=bubble-O3 --profile=perf.data \
                                 --gcov=bubble-O3.gcov --gcov-version=1
```

- Compile the program again (perhaps for the next release), using the profile

```
$ gcc bubble.c -g -O3 -fauto-profile=bubble-O3.gcov \
                                                  -o bubble-O3-profile-use
```

AutoFDO advantages

- Lower runtime overhead
  - Profile generation can be performed off-line
- No need to generate special training data
  - Profiles can be generated based on real (even end user) program execution
  - And can be aggregated from a number of runs
- Source-oriented profile
  - Applicable even after (reasonable) source code changes
- Easier to integrate with build systems
  - New release can use profiles generated with older release

gcc 6.1 -O3 -fauto-profile

```
adrp    x6, __F_E+0xf920
add     x0, x6, #0x920
ldr     w1, [x6,#2336]
add     x5, x0, #0x1d, lsl #12
mov     w2, #0x1
add     x5, x5, #0x4bc
ldr     w3, [x0,#4]
cmp     w1, w3
b.le    main+0x30
str     w3, [x6,#2336]
mov     w2, #0x0
str     w1, [x0,#4]
add     x7, x0, #0x4
ldr     w8, [x0,#4]
ldr     w4, [x7,#4]
cmp     w8, w4
b.le    main+0x50
str     w4, [x0,#4]
mov     w2, #0x0
str     w8, [x7,#4]
add     x14, x7, #0x4
ldp     w9, w10, [x14]
cmp     w9, w10
b.le    main+0x68
mov     w2, #0x0
```

```
stp     w10, w9, [x14]
add     x11, x14, #0x4
ldr     w13, [x14,#4]
ldr     w12, [x11,#4]
cmp     w13, w12
b.le    main+0x88
str     w12, [x14,#4]
mov     w2, #0x0
str     w13, [x11,#4]
ldp     w14, w15, [x11,#4]
cmp     w14, w15
b.le    main+0x9c
mov     w2, #0x0
stp     w15, w14, [x11,#4]
ldp     w16, w17, [x11,#8]
cmp     w16, w17
b.le    main+0xb0
mov     w2, #0x0
stp     w17, w16, [x11,#8]
ldp     w18, w0, [x11,#12]
cmp     w18, w0
b.le    main+0xc4
mov     w2, #0x0
stp     w0, w18, [x11,#12]
ldp     w1, w3, [x11,#16]
```

```
cmp     w1, w3
b.le    main+0xd8
mov     w2, #0x0
stp     w3, w1, [x11,#16]
ldp     w7, w8, [x11,#20]
cmp     w7, w8
b.le    main+0xec
mov     w2, #0x0
stp     w8, w7, [x11,#20]
ldp     w9, w4, [x11,#24]
cmp     w9, w4
b.le    main+0x100
mov     w2, #0x0
stp     w4, w9, [x11,#24]
ldp     w10, w12, [x11,#28]
cmp     w10, w12
b.le    main+0x114
mov     w2, #0x0
stp     w12, w10, [x11,#28]
add     x14, x11, #0x20
cmp     x14, x5
b.ne    main+0x54
cbz     w2, main+0x4
mov     w0, #0x0
ret
```

gcc-6.1 results

metric	-O3	-O3 -fprofile-use	-O3 -fauto-profile
time elapsed	**3.268757833 s** *(-1.1% vs 4.8)*	**2.504173270 s (-23.4% vs -O3)**	**2.806803990 s (-14.1% vs -O3)**
cycles	**6,536,735,848** *(-1.1% vs 4.8)*	**5,007,557,329 (-23.4% vs -O3)**	**5,612,823,771 (-14.1% vs -O3)**
instructions	5,806,220,662 *(-39.5% vs 4.8)*	3,873,453,819 (-33.3% vs -O3)	3,649,604,577 (-37.1% vs -O3)
IPC	0.89	0.77	0.65

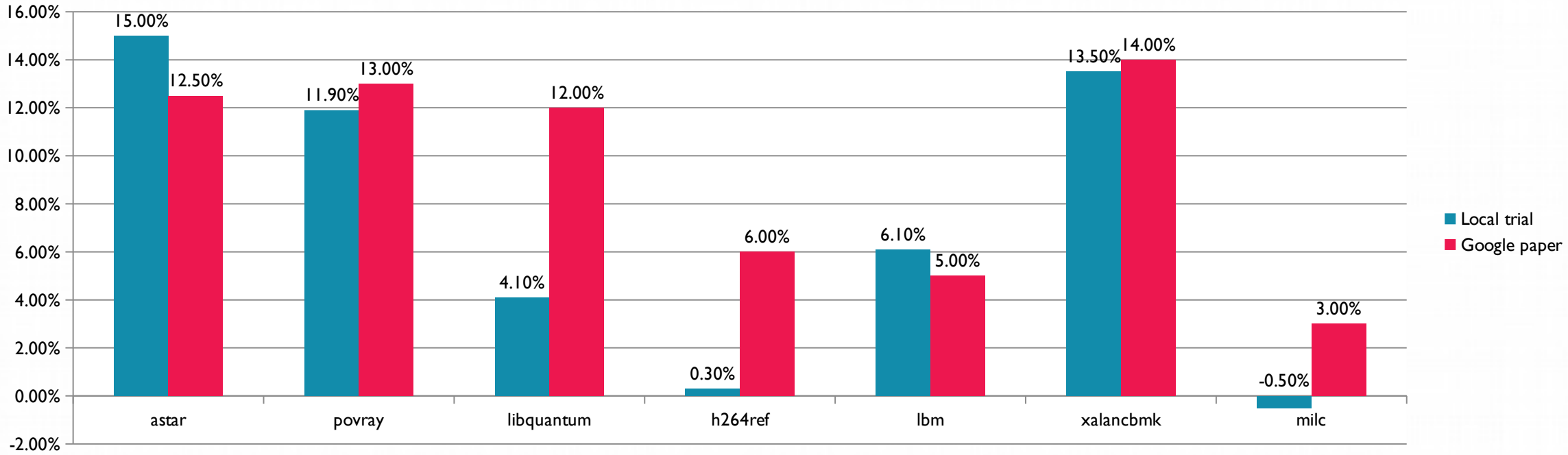
Cortex-A57

Sampled profile quality

- Sampled profiles are inaccurate by nature
- To analyze branch frequency, samples should be focused on branches
  - Precise sampling on "branch executed" events
  - Branch history stack (`perf record -b`)
  - Processor trace
- All this require hardware support
- Branch history drastically improves statistical profile quality with little overhead
  - "Taming hardware event samples for precise and versatile feedback directed optimization" paper
- Processor trace provides accurate branch information but increases overhead
  - May be reasonable for performance critical portions

SPEC2006 results

- Google's AutoFDO gcc branch provided real improvements up to 15%, as described in "Hardware Counted Profile-Guided Optimization" paper



gcc-google-4.8, x86_64, SPEC2006 result improvement with "–O2 –fauto-profile=autofdo.gcov" over "–O2"

ARM

Challenges with sample based FDO

- Not 100% mature tools
  - Profile compatibility issues
- Requires detailed debug information for binaries
  - Sometimes hard to achieve in production releases
- Observed instability of results
  - Profile generated for AutoFDO optimized binary can cause performance regression in the next build
  - Usually result of lost information about execution hotspots, eg:

```
if (cond) x = a; else x = b;
```

converted into

```
csel x, a, b, cond
```

FDO in LLVM

- Instrumentation based FDO

```
$ clang -O3 -fprofile-instr-generate bubble.c \
                              -o bubble-O3-profile-instr-generate
$ clang -O3 -fprofile-instr-use=bubble.profdata bubble.c \
                                    -o bubble-O3-profile-instr-use
```

- AutoFDO support currently catching up with gcc results

```
$ clang -O3 -g bubble.c -o bubble-O3
$ perf record -b bubble-O3
$ create_llvm_prof --binary=bubble-O3 --profile=perf.data \
                               --out=bubble-O3.prof –format=text
$ clang -O3 -gline-tables-only \
                             -fprofile-sample-use=bubble-O3.prof \
                          bubble.c -o bubble-O3-profile-sample-use
```

Example LLVM AutoFDO profile

```
0: void Proc_3 (Rec_Pointer *Ptr_Ref_Par)
1: /*******************/
2: /* executed once */
3: /* Ptr_Ref_Par becomes Ptr_Glob */
4: {
5:     if (Ptr_Glob != Null)
6:         /* then, executed */
7:         *Ptr_Ref_Par = Ptr_Glob->Ptr_Comp;
8:     Proc_7 (10, Int_Glob, &Ptr_Glob->variant.var_1.Int_Comp);
9: } /* Proc_3 */
```

```
Proc_3:728:14
5: 14
7: 14
8: 14  Proc_7:10
```

Deployments

- Commercial products
  - Often only for performance critical portions
- Open source projects like CPython and Firefox
  - Support for FDO available in build system but not turned on by default
- Google data center
  - Origins of AutoFDO
- Chrome & ChromeOS
  - Cross profiling
- ClearLinux

AutoFDO at Google data center

- At data center scale, even fractional improvement translates into significant financial savings
- "AutoFDO: Automatic Feedback-Directed Optimization for Warehouse-Scale Applications" paper discusses Google's infrastructure:

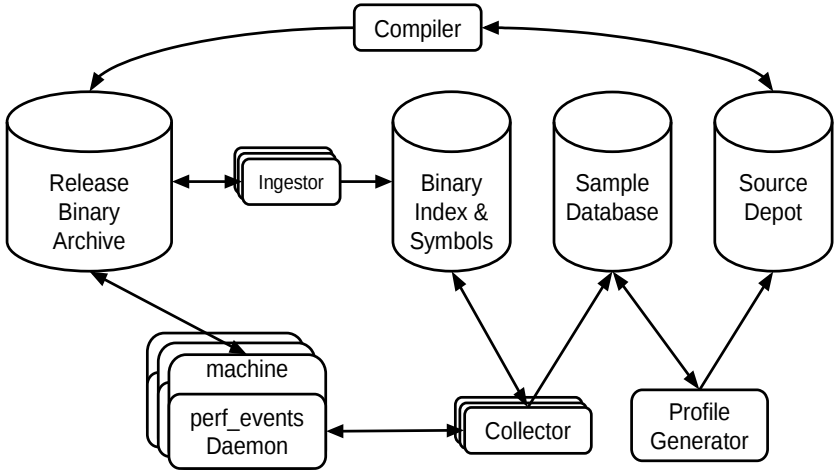


Figure 1. System Diagram.

Future

- Intensive development in LLVM
  - Fueled by Google work on replacing gcc in their work flows
- More hardware providing relevant data
  - Intel PT already available in mainline kernel
  - ARM's CoreSight trace mostly merged
  - New PMU features in both architectures
- Wider deployment in managed environments
  - Very natural technique for JITs, can avoid most static environment challenges
  - Many use FDO already

Summary

- There is no magic "Make My Program Faster" compilation option
  - Although, carefully used, FDO can bring significant improvements
- Instrumentation based FDO known since mainframes era
  - And yet surprisingly rarely used in practice
- Sample based AutoFDO lowers entry barrier
  - But still requires careful maintenance
- Do give it a try!
  - **Just make sure to measure effects**