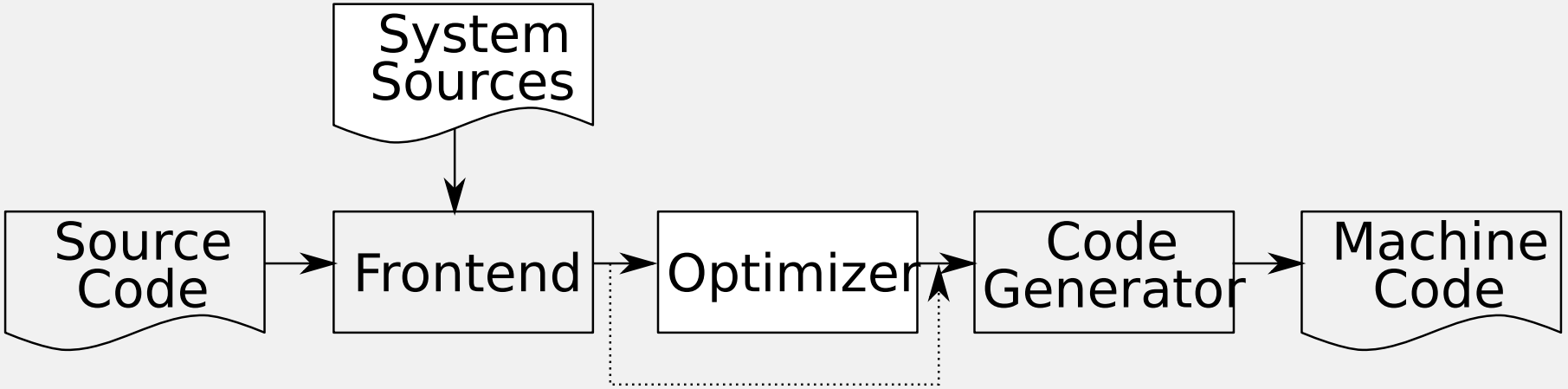


redhat.

Understanding Compiler Optimizations

Jakub Jelinek
Ulrich Drepper

Compiler Overview



Example Code

```
def euclid(a, b):
    while a != b:
        if a > b:
            a = a - b
        else:
            b = b - a
    return a
```

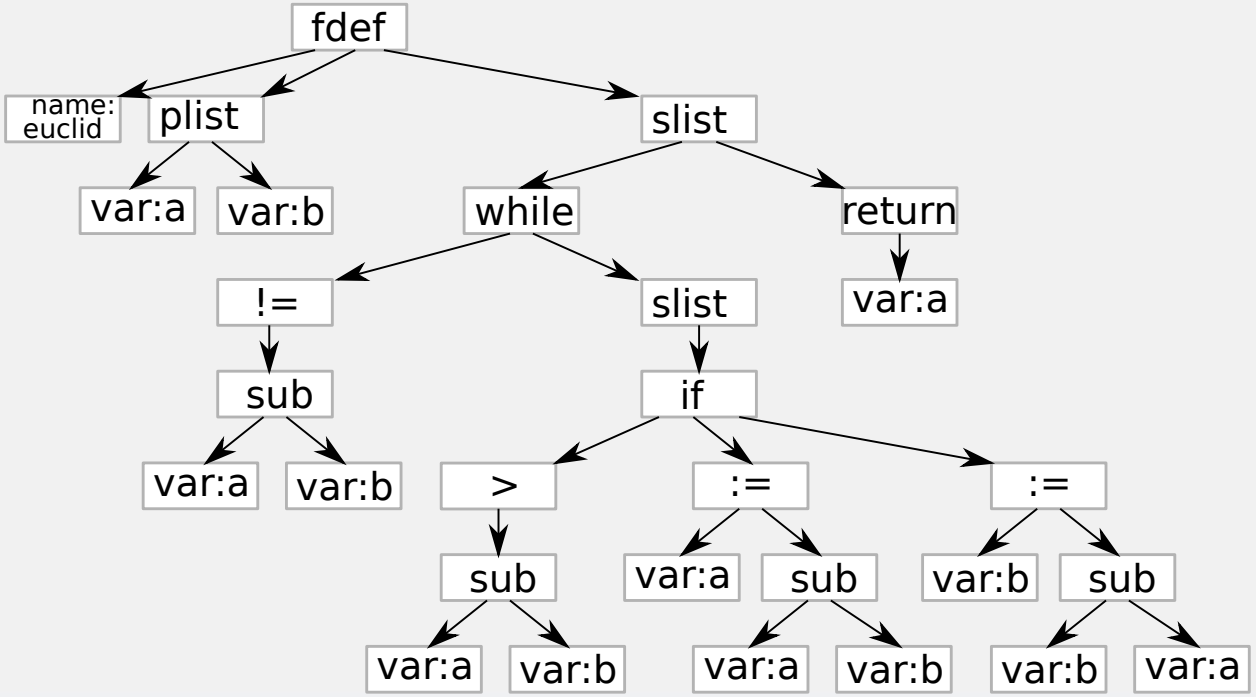
Tokenization

def | var:euclid | (| var:a | , | var:b |) | : | while | var:a | !=

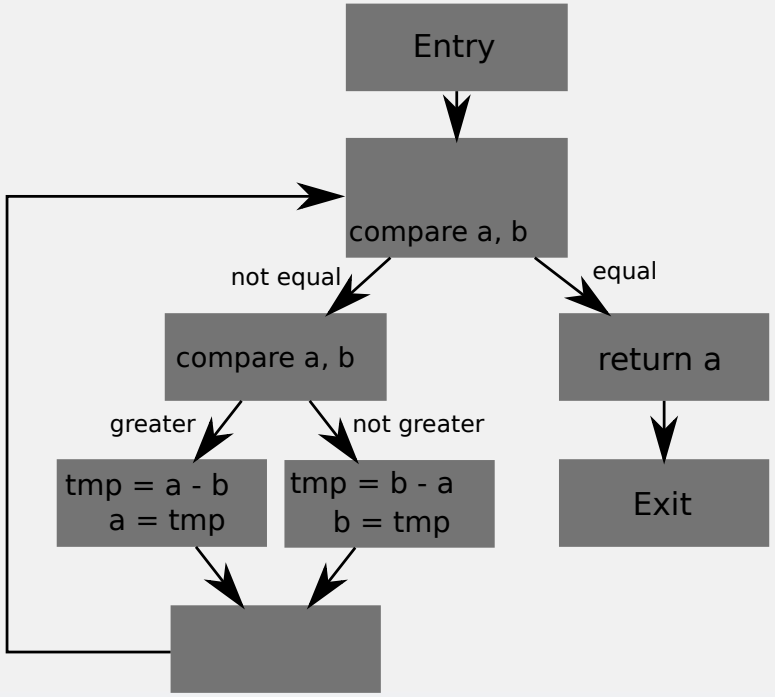
var:b | : | if | var:a | > | var:b | : | var:a | = | var:a | - | var:b

else | : | var:b | = | var:b | - | var:a | return | var:a

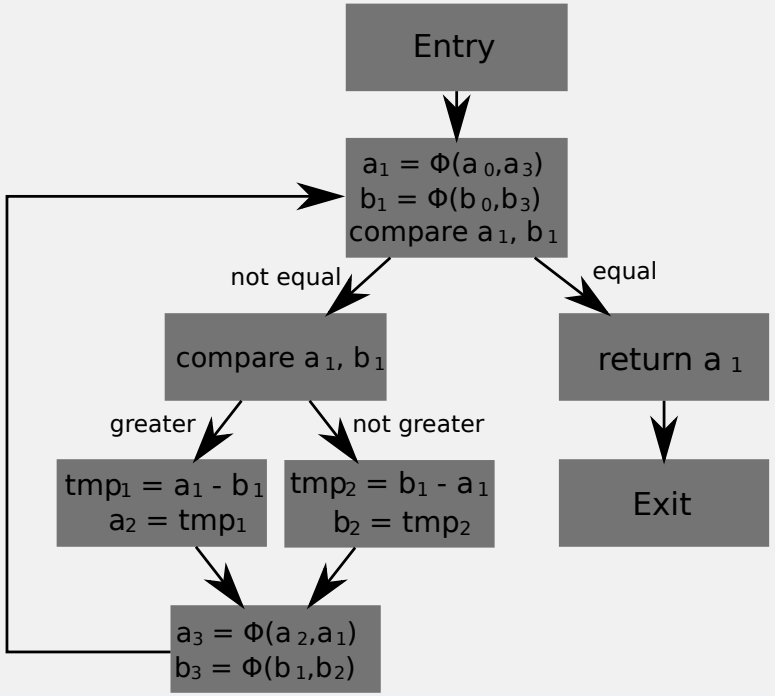
Abstract Syntax Tree



Basic Blocks



Single Static Assignment Form



Optimizations

- Definition

Transformation of the (tree) representation into one which, according to the rules of the programming language, is equivalent but can be executed with less resource use (time, memory, …).

gcc Optimizations

```
gcc-opt-enabled [master]$ ./gcc-opt-enabled.sh
```

	0	g	1	s	2	3	fast
-falign-functions	✘	✘	✘	✘	✘	✘	✘
-falign-jumps	✘	✘	✘	✘	✘	✘	✘
-falign-labels	✘	✘	✘	0	0	0	0
-falign-loops	✘	✘	✘	✘	✘	✘	✘
-fassociative-math	✘	✘	✘	✘	✘	✘	✔
-fasynchronous-unwind-tables	✔	✔	✔	✔	✔	✔	✔
-fauto-inc-dec	✔	✔	✔	✔	✔	✔	✔
-fbranch-count-reg	✘	✘	✔	✔	✔	✔	✔
-fbranch-probabilities	✘	✘	✘	✘	✘	✘	✘
-fbranch-target-load-optimize	✘	✘	✘	✘	✘	✘	✘
-fbranch-target-load-optimize2	✘	✘	✘	✘	✘	✘	✘
-fbtr-bb-exclusive	✘	✘	✘	✘	✘	✘	✘
-fcaller-saves	✘	✘	✘	✔	✔	✔	✔
-fcode-hoisting	✘	✘	✘	✔	✔	✔	✔
-fcombine-stack-adjustments	✘	✔	✔	✔	✔	✔	✔
-fcompare-elim	✘	✔	✔	✔	✔	✔	✔
-fconserve-stack	✘	✘	✘	✘	✘	✘	✘
-fcprop-registers	✘	✔	✔	✔	✔	✔	✔
-fcrossjumping	✘	✘	✘	✔	✔	✔	✔
-fcse-follow-jumps	✘	✘	✘	✔	✔	✔	✔
	0	g	1	s	2	3	fast
-fcx-fortran-rules	✘	✘	✘	✘	✘	✘	✘
-fcx-limited-range	✘	✘	✘	✘	✘	✘	✔
-fdce	✔	✔	✔	✔	✔	✔	✔
-fdefer-pop	✘	✔	✔	✔	✔	✔	✔
-fdelayed-branch	✘	✘	✘	✘	✘	✘	✘
-fdelete-dead-exceptions	✘	✘	✘	✘	✘	✘	✘
-fdelete-null-pointer-checks	✔	✔	✔	✔	✔	✔	✔

https://github.com/drepper/gcc-opt-enabled.git

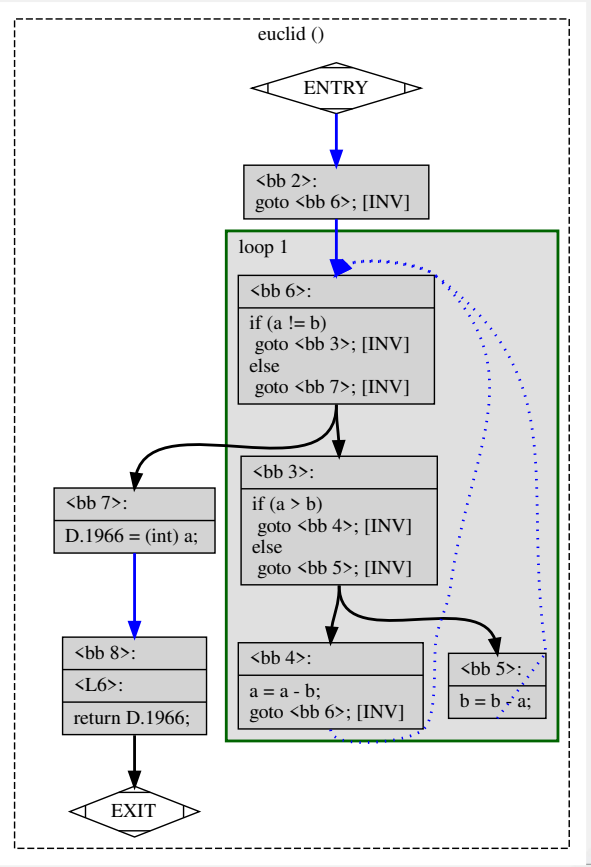
This Time in C

```
unsigned euclid(unsigned a, unsigned b)
{
  while (a != b)
    if (a > b)
      a = a - b;
    else
      b = b - a;
  return a;
}
```

Visualize Compiler Information

```
$ gcc -c -fdump-tree-cfg-graph euclid.c
$ dot -Tsvg euclid.c.011t.cfg.dot > out.svg
$ eog out.svg
$ cat euclid.c.011t.cfg
;; Function euclid (euclid, funcdef_no=0,
decl_uid=1957, cgraph_uid=0, symbol_order=0)

;; 2 loops found
…
euclid (unsigned int a, unsigned int b)
{
  int D.1966;
  <bb 2> :
  goto <bb 6>; [INV]
  <bb 3> :
  if (a > b)
    goto <bb 4>; [INV]
  else
    goto <bb 5>; [INV]
```



Now with Optimizations

```
$ gcc -c -Ofast -fdump-tree-all-graph -fplugin=./passes.so euclid.c
   pass *warn_unused_result
   pass omplower
   pass lower
   pass eh
   pass cfg
   pass *warn_function_return
   pass ompexp
   pass *build_cgraph_edges
simple IPA pass:
   pass *free_lang_data
   pass visibility
   pass build_ssa_passes
GIMPLE pass:
   pass    fixup_cfg
   pass    ssa
   pass    nothrow
   pass    *rebuild_cgraph_edges
simple IPA pass:
   pass opt_local_passes
```

```
GIMPLE pass:
   pass    fixup_cfg
   pass    *rebuild_cgraph_edges
   pass    local-fnsummary
   pass    einline
   pass    early_optimizations
   pass      *remove_cgraph_callee_edges
   pass      objsz
   pass      ccp
   pass      forwprop
   pass      ethread
   pass      esra
   pass      ealias
   pass      fre
   pass      evrp
   pass      mergephi
   pass      dse
   pass      cddce
   pass      eipa_sra
   pass      tailr
   pass      switchconv
   pass      profile_estimate
   pass      local-pure-const
   pass      fnsplit
   pass      *strip_predict_hints
   pass    release_ssa
   pass    *rebuild_cgraph_edges
   pass    local-fnsummary
```

```
simple IPA pass:
   pass targetclone
   pass free-fnsummary
IPA pass:
   pass whole-program
   pass profile_estimate
   pass icf
   pass devirt
   pass cp
   pass fnsummary
   pass inline
   pass pure-const
simple IPA pass:
   pass free-fnsummary
IPA pass:
   pass static-var
   pass single-use
   pass comdats
simple IPA pass:
   pass materialize-all-clones
   pass simdclone
GIMPLE pass:
   pass fixup_cfg
...
```

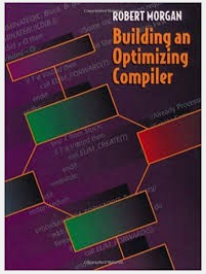
https://github.com/drepper/gcc-passes.git

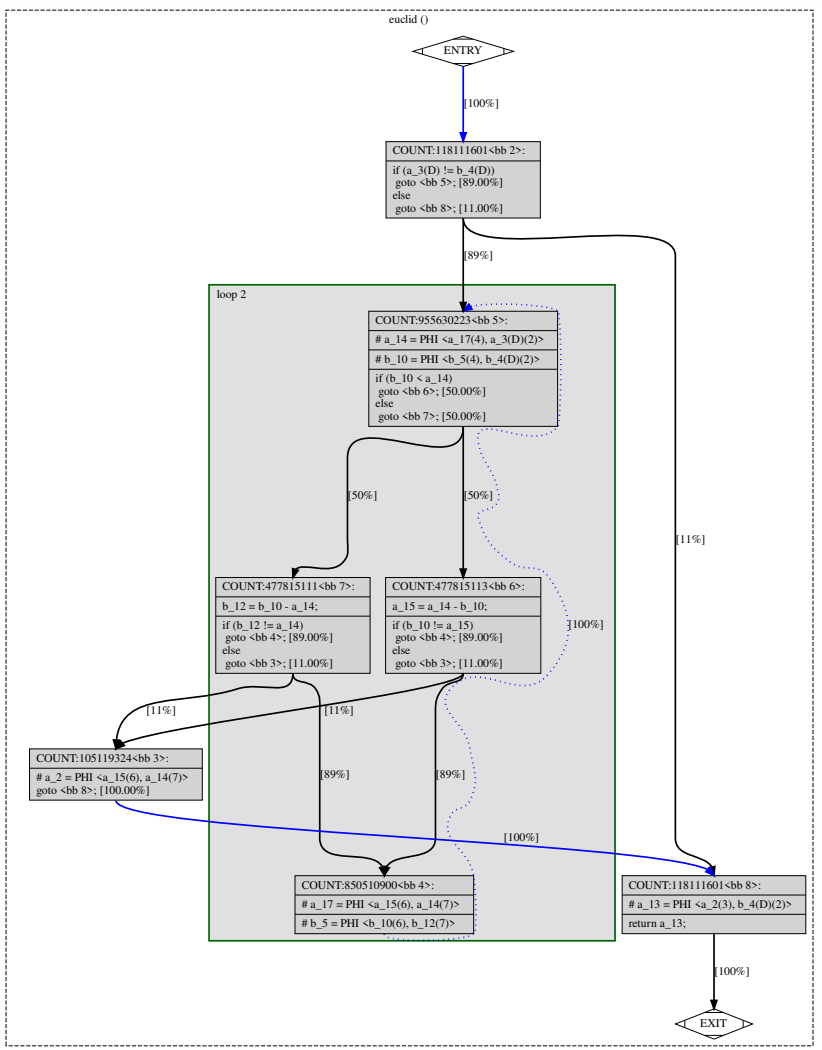
And Many Files

```
$ ls
euclid.c                                euclid.c.050t.local-fnsummary2      euclid.c.124t.reassoc1.dot      euclid.c.168t.lim4
euclid.o                                euclid.c.050t.local-fnsummary2.dot  euclid.c.125t.dce3              euclid.c.168t.lim4.dot
euclid.c.003t.original                  euclid.c.088t.fixup_cfg4            euclid.c.125t.dce3.dot          euclid.c.169t.loopdone
euclid.c.004t.gimple                    euclid.c.088t.fixup_cfg4.dot        euclid.c.126t.forwprop3         euclid.c.169t.loopdone.dot
euclid.c.006t.omplower                  euclid.c.093t.ccp2                  euclid.c.126t.forwprop3.dot     euclid.c.173t.veclower21
euclid.c.007t.lower                     euclid.c.093t.ccp2.dot              euclid.c.127t.phiopt2           euclid.c.173t.veclower21.dot
euclid.c.010t.eh                        euclid.c.095t.cunrolli              euclid.c.127t.phiopt2.dot       euclid.c.174t.recip
euclid.c.011t.cfg                       euclid.c.095t.cunrolli.dot          euclid.c.128t.ccp3              euclid.c.174t.recip.dot
euclid.c.011t.cfg.dot                   euclid.c.096t.backprop              euclid.c.128t.ccp3.dot          euclid.c.175t.printf-return-value2
euclid.c.012t.ompexp                    euclid.c.096t.backprop.dot          euclid.c.129t.sincos            euclid.c.175t.printf-return-value2.dot
euclid.c.012t.ompexp.dot                euclid.c.097t.phiprop               euclid.c.129t.sincos.dot        euclid.c.176t.reassoc2
euclid.c.013t.printf-return-value1      euclid.c.097t.phiprop.dot           euclid.c.130t.bswap             euclid.c.176t.reassoc2.dot
euclid.c.013t.printf-return-value1.dot  euclid.c.098t.forwprop2             euclid.c.130t.bswap.dot         euclid.c.177t.slsr
euclid.c.019t.fixup_cfg1                euclid.c.098t.forwprop2.dot         euclid.c.131t.laddress          euclid.c.177t.slsr.dot
euclid.c.019t.fixup_cfg1.dot            euclid.c.099t.objsz2                euclid.c.131t.laddress.dot      euclid.c.178t.split-paths
euclid.c.020t.ssa                       euclid.c.099t.objsz2.dot            euclid.c.132t.lim2              euclid.c.178t.split-paths.dot
euclid.c.020t.ssa.dot                   euclid.c.100t.alias                 euclid.c.132t.lim2.dot          euclid.c.180t.thread3
euclid.c.022t.nothrow                   euclid.c.100t.alias.dot             euclid.c.134t.pre               euclid.c.180t.thread3.dot
euclid.c.022t.nothrow.dot               euclid.c.101t.retslot               euclid.c.134t.pre.dot           euclid.c.181t.dom3
euclid.c.027t.fixup_cfg3                euclid.c.101t.retslot.dot           euclid.c.135t.sink              euclid.c.181t.dom3.dot
euclid.c.027t.fixup_cfg3.dot            euclid.c.102t.fre3                  euclid.c.135t.sink.dot          euclid.c.182t.strlen
euclid.c.028t.local-fnsummary1          euclid.c.102t.fre3.dot              euclid.c.139t.dce4              euclid.c.182t.strlen.dot
euclid.c.028t.local-fnsummary1.dot      euclid.c.103t.mergephi2             euclid.c.139t.dce4.dot          euclid.c.183t.thread4
euclid.c.029t.einline                   euclid.c.103t.mergephi2.dot         euclid.c.140t.fix_loops         euclid.c.183t.thread4.dot
euclid.c.029t.einline.dot               euclid.c.104t.thread1               euclid.c.140t.fix_loops.dot     euclid.c.184t.vrp2
euclid.c.030t.early_optimizations       euclid.c.104t.thread1.dot           euclid.c.141t.loop              euclid.c.184t.vrp2.dot
euclid.c.030t.early_optimizations.dot   euclid.c.105t.vrp1                  euclid.c.141t.loop.dot          euclid.c.186t.phicprop2
euclid.c.031t.objsz1                    euclid.c.105t.vrp1.dot              euclid.c.142t.loopinit          euclid.c.186t.phicprop2.dot
euclid.c.031t.objsz1.dot                euclid.c.107t.dce2                  euclid.c.142t.loopinit.dot      euclid.c.187t.dse3
euclid.c.032t.ccp1                      euclid.c.107t.dce2.dot              euclid.c.143t.unswitch          euclid.c.187t.dse3.dot
euclid.c.032t.ccp1.dot                  euclid.c.108t.stdarg                euclid.c.143t.unswitch.dot      euclid.c.188t.cddce3
euclid.c.033t.forwprop1                 euclid.c.108t.stdarg.dot            euclid.c.144t.sccp              euclid.c.188t.cddce3.dot
euclid.c.033t.forwprop1.dot             euclid.c.109t.cdce                  euclid.c.144t.sccp.dot          euclid.c.189t.forwprop4
euclid.c.034t.ethread                   euclid.c.109t.cdce.dot              euclid.c.145t.lsplit            euclid.c.189t.forwprop4.dot
....
$ ls|wc -l
252
```

Optimized Version

- SSA notation
- Branch probabilities





Performed Optimization

- Dead Code Elimination:

```
unsigned f(unsigned m)
{
  unsigned tmp = m + 4;
  if (m < 0)
    return f(tmp);
  return m;
}
```

- Dead Store:

```
int foo(int a, int b) {
  int l = a;
  l = g(b);
  return b + l;
}
```

- Common Subexpression Elimination

```
if (a + b * c < 10 || a + b * c + 1 > 42)
  break;
```

to

```
tmp = a + b * c;
if (tmp < 10 || tmp > 41)
  break;
```

- Reduce comparisons: to

```
tmp = a + b * c;
if ((unsigned)tmp - 10u > 31)
  break;
```

Let's Verify

```
extern int g(int);
int f(int a, int b, int c)
{
  if (a+b*c < 10 || a+b*c+1 > 42)
    return 0;
  return g(a);
}
```

```
f (int a, int b, int c)
{
  int _1;
  int _2;
  unsigned int _3;
  unsigned int _4;
  int _5;
  int _12;

  <bb 2> [local count: 1073741825]:
  _1 = b_7(D) * c_8(D);
  _2 = _1 + a_9(D);
  _3 = (unsigned int) _2;
  _4 = _3 + 4294967286;
  if (_4 > 31)
    goto <bb 4>; [51.12%]
  else
    goto <bb 3>; [48.88%]

  <bb 3> [local count: 524845004]:
  _12 = g (a_9(D)); [tail call]

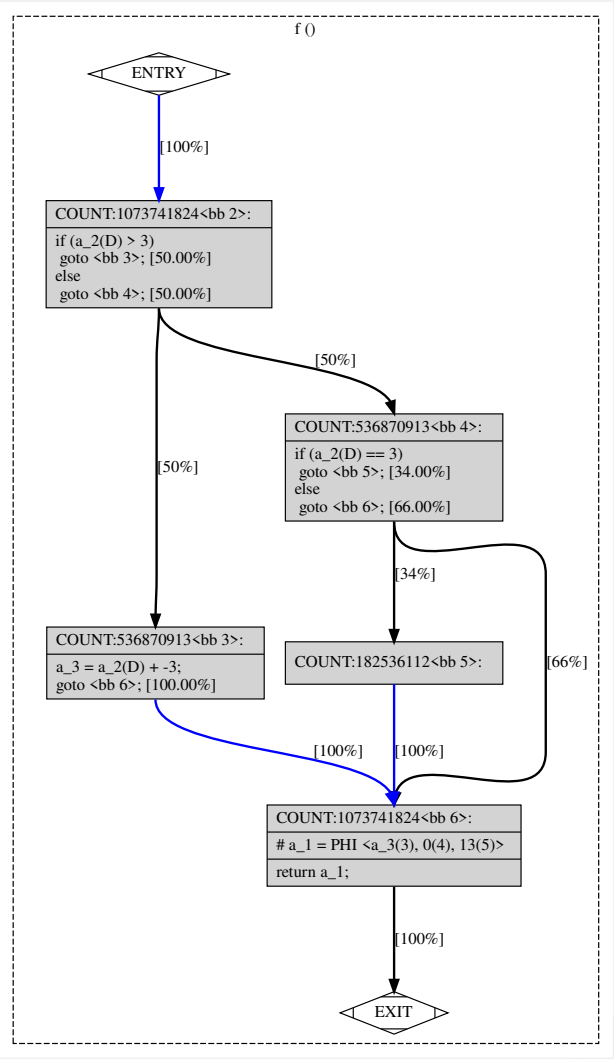
  <bb 4> [local count: 1073741825]:
  # _5 = PHI <_12(3), 0(2)>
  return _5;
}
```



Performed Optimizations II

- Value Range Propagation

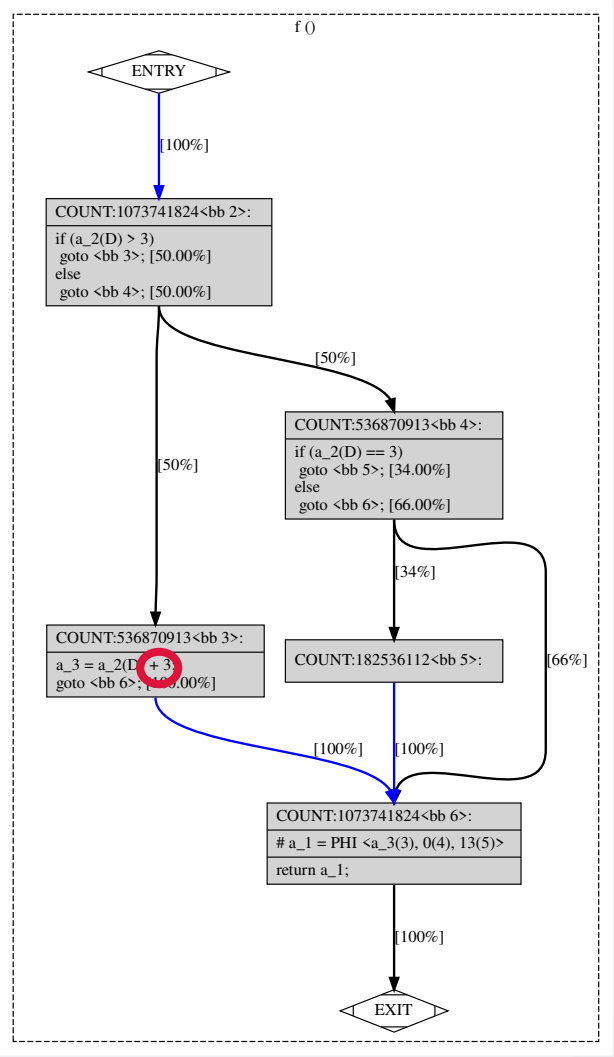
```
int f(int a) {
  if (a > 3)
    a -= 3;
  else
    switch (a) {
    case 3:
      a += 10;
      break;
    case 4:
      a = 10;
      break;
    default:
      a = 0;
      break;
    }
  return a < 0 ? -1 : a;
}
```



The Same?

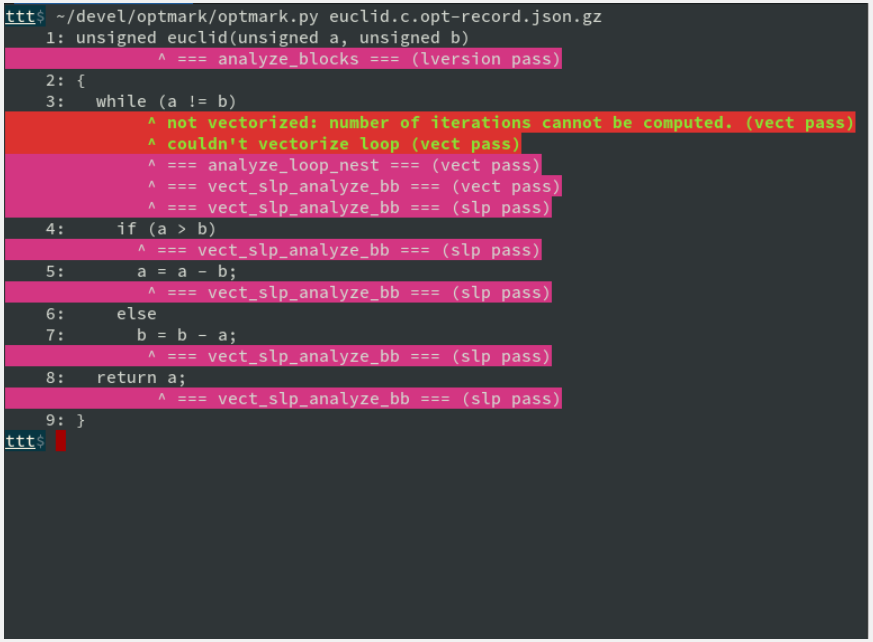
- Value Range Propagation

```
int f(int a) {
  if (a > 3)
    a += 3;
  else
    switch (a) {
    case 3:
      a += 10;
      break;
    case 4:
      a = 10;
      break;
    default:
      a = 0;
      break;
    }
  return a < 0 ? -1 : a;
}
```



Record Optimizations (gcc 9+)

```
$ gcc -c -Ofast euclid.c -fsave-optimization-record
$ optmark euclid.c.opt-record.json.gz
```



https://github.com/drepper/optmark

How To Improve Code

Example: Tail Calls

```
extern int tail(int);
int f(int a)
{
  return tail(a + 1);
}
```

```
_Z1fi:
        addl    $1, %edi
        jmp     _Z4taili
```

```
extern int tail(int);
int f(int a)
{
  return 1 + tail(a);
}
```

```
_Z1fi:
        subq    $8, %rsp
        call    _Z4taili
        addq    $8, %rsp
        addl    $1, %eax
        ret
```

Example: More Tail Calls

```
#include <string>
using namespace std::string_literals;
extern int tail(int);
extern void out(std::string&);
int f(int a) {
  if (a > 10) {
    std::string s = "a = "s + std::to_string(a);
    out(s);
  }
  return tail(a + 1);
}
```

```
f(int):
    pushq  %r13
    movl   %edi, %r8d
    movl   $.LC2, %ecx
    movl   $16, %edx
    pushq  %r12
    movl   $vsnprintf, %esi
    xorl   %eax, %eax
    pushq  %rbp
    pushq  %rbx
    movl   %edi, %ebx
    subq   $104, %rsp
    leaq   64(%rsp), %rdi
    leaq   48(%rsp), %rbp
    call   std::__cxx11::basic_string<char, std::char_traits<char>,…
    …
    call   out(std::__cxx11::basic_string<char, std::char_traits<char>,…
    leal   1(%rbx), %edi
    call   tail(int)
    movq   (%rsp), %rdi
    movl   %eax, %ebx
    leaq   16(%rsp), %rax
    cmpq   %rax, %rdi
    je     .L102
    call   operator delete(void*)
.L102:
    addq   $104, %rsp
    movl   %ebx, %eax
    popq   %rbx
    popq   %rbp
    popq   %r12
    popq   %r13
    ret
```

Fixing Tail Calls

```
#include <string>
using namespace std::string_literals;
extern int tail(int);
extern void out(std::string&);
namespace {
  [[gnu::noinline]] void inner(int a) noexcept
  {
    auto s = "a = "s + std::to_string(a);
    out(s);
  }
}
int f(int a)
{
  if (a > 10)
    inner(a);
  return tail(a + 1);
}
```

```
f(int):
        pushq   %rbx
        movl    %edi, %ebx
        cmpl    $10, %edi
        jg      .L121
        leal    1(%rbx), %edi
        popq    %rbx
        jmp     tail(int)
.L121:
        call    (anonymous namespace)::inner(int)
        leal    1(%rbx), %edi
        popq    %rbx
        jmp     tail(int)
```

Example: Assumptions

```
static inline int f(int a) {
  switch (a) {
  case 0:
    return 0;
  case 1 ... 3:
    return 1;
  case 4 ... 10:
    return -1;
  default:
    return 42;
  }
}

int g(int a) {


  return f(a);
}
```

```
g:
   0:    mov    $0x2a,%eax
   5:    cmp    $0xa,%edi
   8:    ja     13 <g+0x13>
   a:    mov    %edi,%edi
   c:    mov    CSWTCH.0(,%rdi,4),%eax
  13:    retq
```

Example: Assumptions

```
static inline int f(int a) {
  switch (a) {
  case 0:
    return 0;
  case 1 ... 3:
    return 1;
  case 4 ... 10:
    return -1;
  default:
    return 42;
  }
}

int g(int a) {
  if (a < 0 || a > 3)
    __builtin_unreachable();
  return f(a);
}
```

```
g:
   0:     mov    %edi,%edi
   2:     mov    CSWTCH.0(,%rdi,4),%eax
   9:     retq
```

Example: Vectorization

```
#include <cstddef>
#ifndef R
# define R __restrict
#endif
using float_type = double;
void f(size_t n, float_type* R res, const float_type* R a,
       float_type b)
{
#ifdef P
#pragma GCC ivdep
#elif defined OMP
#pragma omp simd
#endif
  for (size_t i = 0; i < n; ++i)
    res[i] = a[i] * b - 1.0;
}
```

```
        … prepare loop …

.L4:    vmovapd %xmm3, %xmm1
        vfmadd132pd     (%rdx,%rax), %xmm2, %xmm1
        vmovups %xmm1, (%rsi,%rax)
        addq    $16, %rax
        cmpq    %rax, %rcx
        jne     .L4
        movq    %rdi, %rax
        andq    $-2, %rax
        andl    $1, %edi
        je      .L11
.L3:    vmovsd  .LC1(%rip), %xmm4
        vfmadd132sd     (%rdx,%rax,8), %xmm4, %xmm0
        vmovsd  %xmm0, (%rsi,%rax,8)
.L11:   ret
```

Example: More Vectorization

```
#include <math.h>
void foo(double *p, double *q)
{
  for (int i = 0; i < 1024; ++i)
    p[i] = sqrt(q[i]);
}
```

```
vec1.c:2:35: missed:   not vectorized: control flow in loop.
vec1.c:2:35: missed:  bad loop form.
vec1.c:2:35: missed: couldn't vectorize loop
  _6 = .SQRT (_4);
  if (_4 u>= 0.0)
    goto <bb 8>; [99.95%]
  else
    goto <bb 4>; [0.05%]
...
  <bb 4> [local count: 531495]:
  sqrt (_4);
```

Example: Vectorization and Alignment

```
void foo(int *p, int *q, int *r)
{
#ifdef DECLARE_ALIGN
  p = __builtin_assume_aligned(p, 16);
  q = __builtin_assume_aligned(q, 16);
  r = __builtin_assume_aligned(r, 16);
#endif
  for (int i = 0; i < 1024; i++)
    p[i] = q[i] + r[i];
}
```

```
vec2.c:1:37: note:   Alignment of
access forced using peeling.
vec2.c:1:37: note:   Peeling for
alignment will be applied.
```



redhat.

Any Questions?

Jakub Jelinek
jakub@redhat.com

Ulrich Drepper
drepper@redhat.com
https://akkadia.org/drepper
https://github.com/drepper