Automatically Identifying Compiler Performance Anomalies

Tipp Moseley
Dirk Grunwald

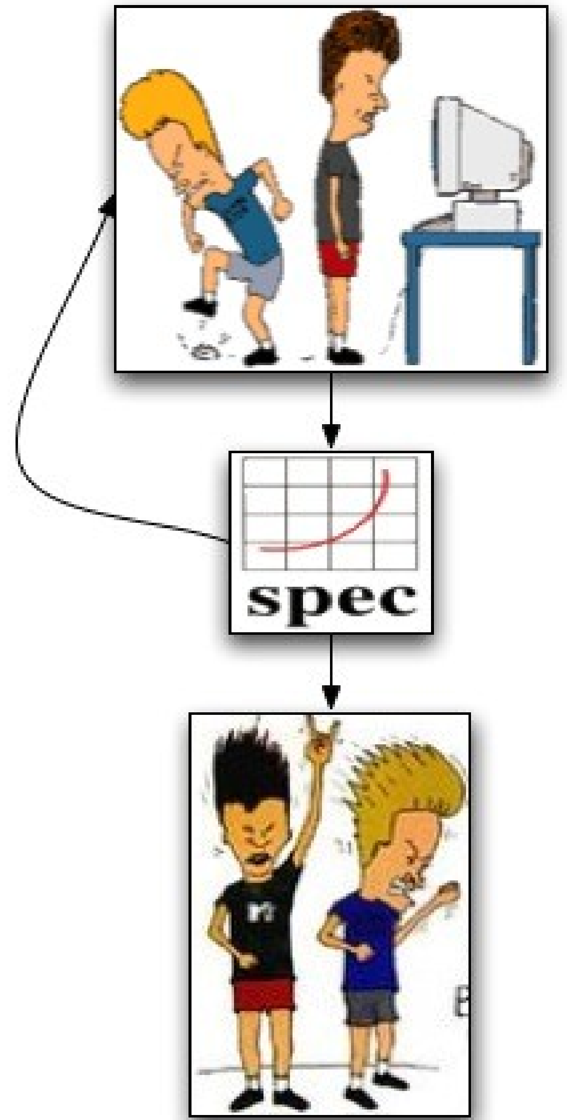
University of Colorado at Boulder

Ramesh Peri

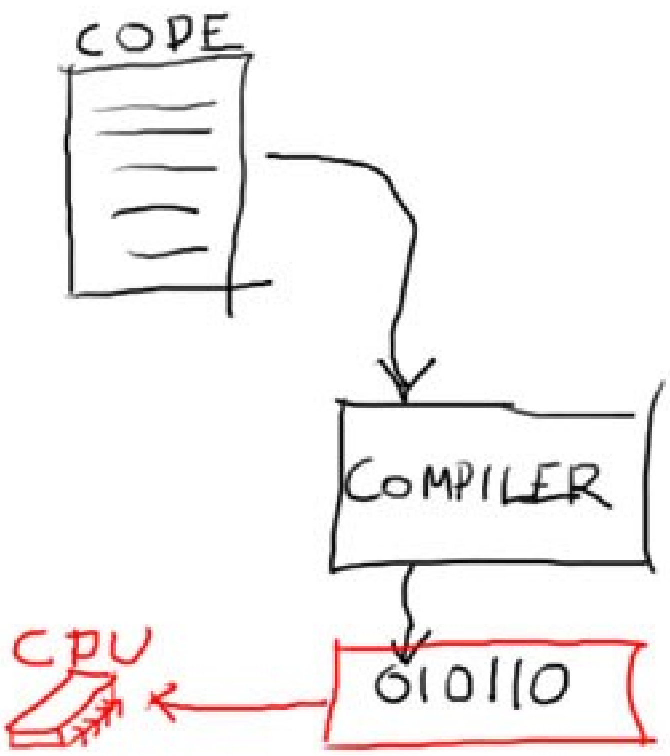
Intel

Primitive Development Cycle for Compilers

- Come up with an idea
- Test it
- If nothing improved, go to 1
- Celebrate

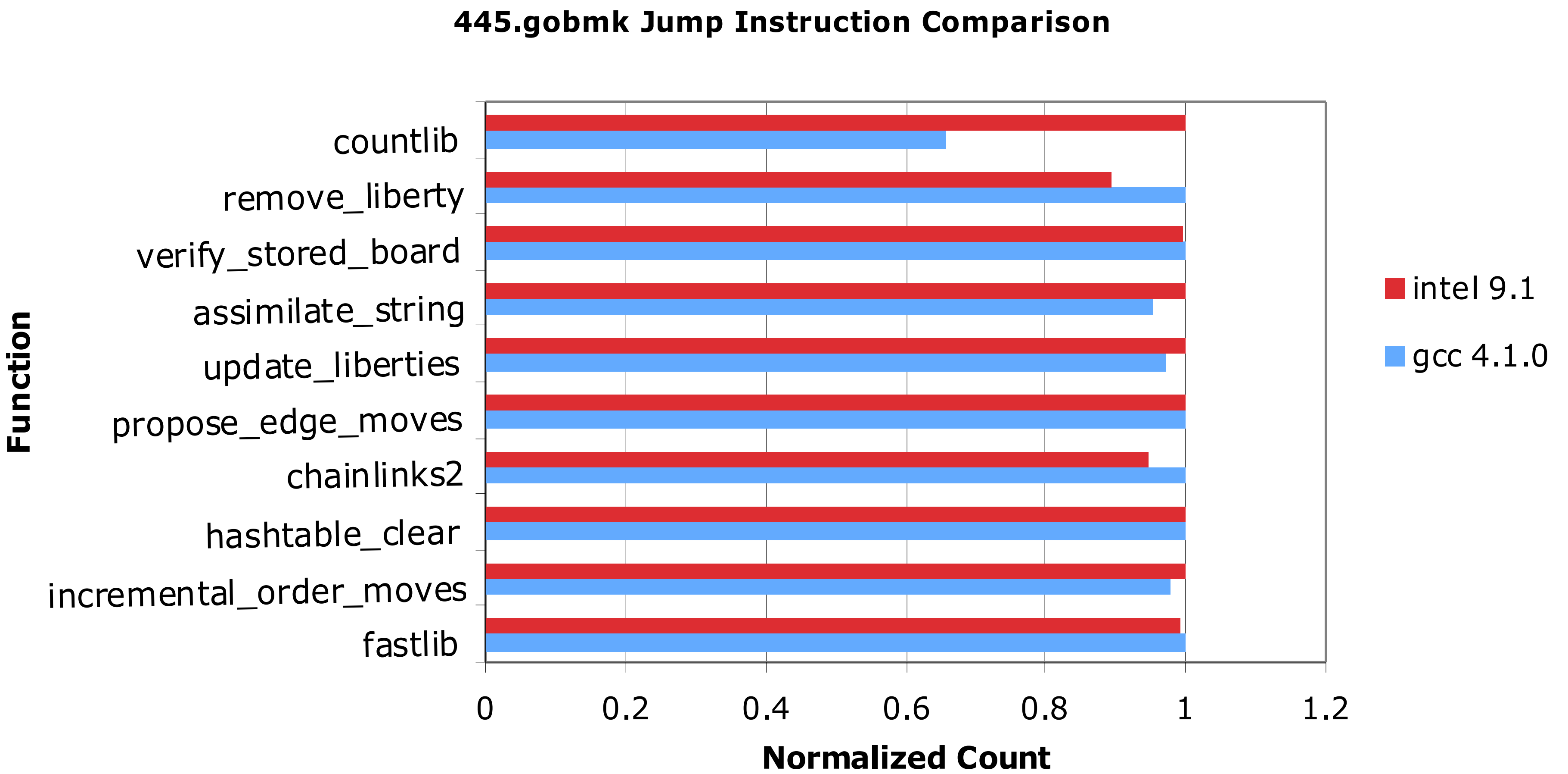


Understanding transformations is hard



- Code size vs. control flow
  - Inlining, loop transformations, superblocks, if-conversion
- Architectural Features
  - Superscalar, OoO, speculation, EPIC, multilevel memory hierarchy, prefetching

Unpredictable Results



Unpredictable Interactions

- Dozens of optimizations and parameters
- Selecting per-bench parameters for gcc on SPEC improves performance by 6%
- Best overall loop unrolling factor for 132.ijpeg is 2, but performance increases 8.81% if each function uses best parameter
- Iterative Compilation tries many combinations
  - Balances between compilation speed and search depth

Outline

- Relative Profile Data Analysis (RPDM)
  - Methodology
  - Screenshots
- Case Studies
- General Observations
- Future Work

Relative Profile Data Analysis

- Provide detailed metrics to measure impact of differently compiled benchmarks
  - Function-level comparison
- Usage scenarios
  - Identify missed opportunities and performance bugs
  - Understand impact of new optimizations
  - Regression testing

Relative Profile Data Analysis

1. Collect instruction mix profiles for many benchmarks with multiple compilers and optimization flags
2. Populate database with profile data
3. Brute force query database to identify most significant outlier functions (e.g., total ins, FP ops, jumps, stack r/w, mem r/w)
4. Visually inspect interesting cases

Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:2001/cgi-bin/cpdm.cgi

Step 1: Select some profiles (you probably want to select profiles for the sa

	linux em64t gnu 4.1.0 shared -O2	linux em64t gnu 4.1.0 shared -O3-mtune	linux em64t intel 9.1 shared -O2	linux em64t intel 9.1 shared -O3-mtune	linux em64t intel 9.1 shared -fast	linux ia32 gnu 3.2.3 shared -O2	linux ia32 gnu 3.4.6 shared -O2	linux ia32 gnu 4.1.0 shared -O2	linux ia32 gnu 4.1.0 shared -O3-mtune	linux ia32 gnu 4.1.1 shared -O2_ref0	-O2
cpu2000/253.perlbmk/train/0										☐	
cpu2000/253.perlbmk/train/1										☐	
cpu2000/253.perlbmk/train/2										☐	
cpu2000/253.perlbmk/train/3										☐	
cpu2000/253.perlbmk/train/4										☐	
cpu2000/253.perlbmk/train/5										☐	
cpu2000/253.perlbmk/train/6										☐	
cpu2006/400.perlbench/train/0	☐	☐	☐	☐	☐	☐	☐	☐	☐		
cpu2006/400.perlbench/train/1	☐	☐	☐	☐	☐	☐	☐	☐	☐		
cpu2006/400.perlbench/train/2	☐	☐	☐	☐	☐	☐	☐	☐	☐		
cpu2006/400.perlbench/train/3	☐	☐	☐	☐	☐	☐	☐	☐	☐		
cpu2006/400.perlbench/train/4	☐	☐	☐	☐	☐	☐	☐	☐	☐		
cpu2006/401.bzip2/train/0	☐	☐	☐	☐	☐	☐	☐	☐	☐		
cpu2006/401.bzip2/train/1	☐	☐	☐	☐	☐	☐	☐	☐	☐		
cpu2006/401.bzip2/train/2	☐	☐	☐	☐	☐	☐	☐	☐	☐		
cpu2006/403.gcc/train/0	☐	☐	☐	☐	☐	☐	☐				
cpu2006/410.bwaves/train/0	☐	☐						☐	☐		
cpu2006/416.gamess/train/0								☐	☐		
cpu2006/429.mcf/train/0	☐	☐	☐	☐	☐	☐	☐	☐	☐		
cpu2006/433.milc/train/0	☐	☐	☐	☐	☐	☐	☐	☐	☐		
cpu2006/434.zeusmp/train/0							☐	☐	☐		
cpu2006/435.gromacs/train/0	☐	☐	☐	☐	☐		☐	☐	☐		

Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:2001/cgi-bin/cpdm.cgi?cpu2006%2F433.milc%2Ftrain%2F0%2Flinux%2Fia32%2Fgnu%2F4.1.0%2Fshared%2F

Select a query:

Or type one in:

Get it!

Current Query: '^J'

Matched Functions

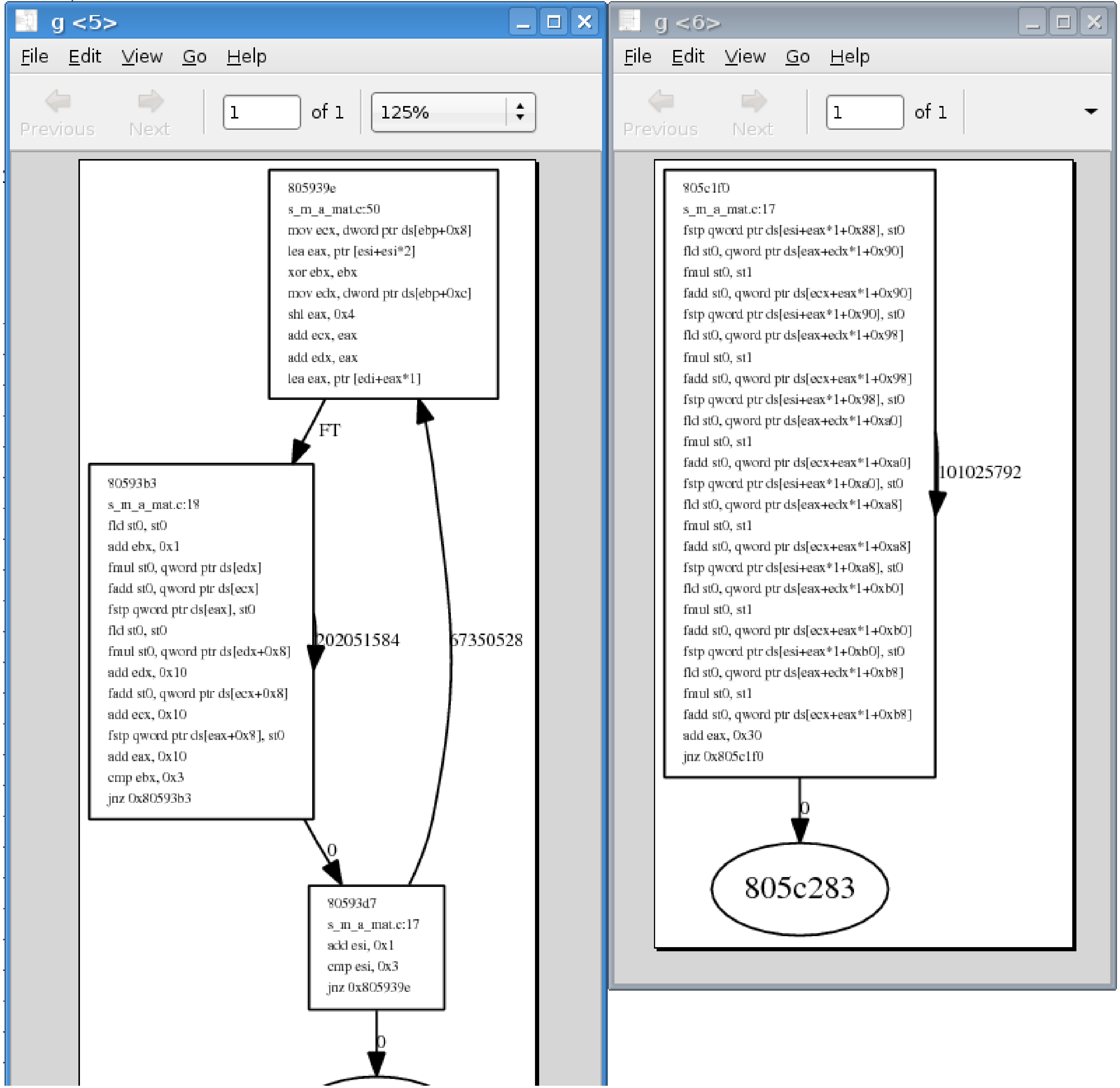
	MAX	linux ia32 gnu 4.1.0 shared -O2 433.milc train/0 NORMALIZED	linux ia32 intel 9.1 shared -O2 433.milc train/0 NORMALIZED
$dynamic-counts	1B	1.000	0.423
scalar_mult_add_su3_matrix	404M	1.000	0.333
su3_projector	254M	1.000	0.333
mult_su3_nn	206M	1.000	0.333
mult_su3_na	201M	1.000	0.333
su3_adjoint	74M	1.000	0.333
mult_su3_mat_vec	63M	1.000	0.000
mult_adj_su3_mat_vec	61M	1.000	0.000
start_gather_from_temp	59M	1.000	1.000
scalar_mult_add_su3_vector	39M	1.000	0.000
su3mat_copy	28M	1.000	1.000

Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://localhost:2001/cgi-bin/cpdm.cgi?mode=showfunc&func=scalar_mult_add_su3_matrix&cpu2006/433.milc/train/0/linux

	MAX	linux ia32 gnu 4.1.0 shared -O2 433.milc train/0 scalar_mult_add_su3_matrix IPO: 1774190592 NORMALIZED Disassemble Disassemble (dot) Disassemble (ps)	linux ia32 intel 9.1 shared -O2 433.milc train/0 scalar_mult_add_su3_matrix IPO: 1774190592 NORMALIZED Disassemble Disassemble (dot) Disassemble (ps)
Dyn Total	5B	100.0	50.9
Stack Reads	437M	100.0	46.2
Stack Writes	134M	100.0	25.0
Memory Reads	1B	100.0	85.7
Memory Writes	740M	100.0	86.4
mem-read-var	0	0.0	0.0
mem-write-var	0	0.0	0.0
mem-atomic	0	0.0	0.0
iprel-read	0	0.0	0.0
iprel-write	0	0.0	0.0
CMOV*	0	0.0	0.0
ADD*	1B	100.0	6.7
SUB*	0	0.0	0.0
J*	404M	100.0	33.3
CALL	0	0.0	0.0
RET	33M	100.0	100.0
MUL*	0	0.0	0.0
DIV*	0	0.0	0.0

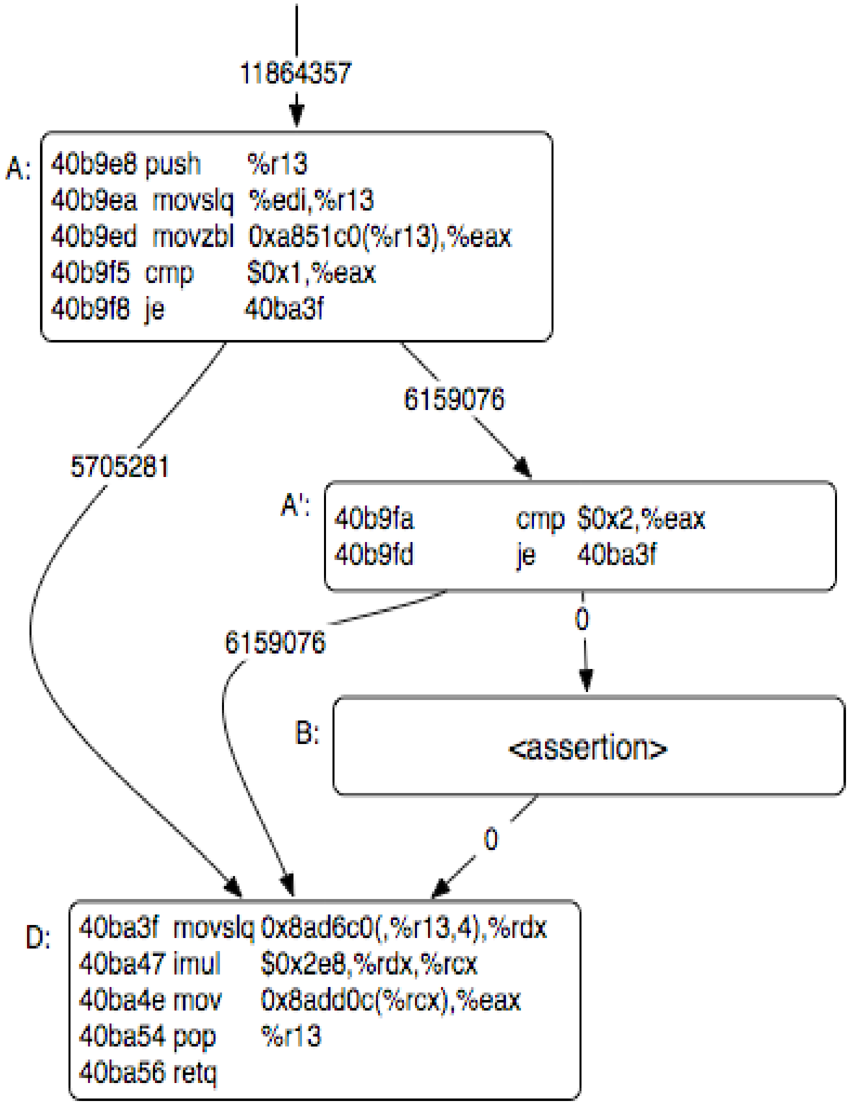

g <5>
File Edit View Go Help
Previous Next 1 of 1 125%
805939e
s_m_a_mat.c:50
mov ecx, dword ptr ds[ebp+0x8]
lea eax, ptr [esi+esi*2]
xor ebx, ebx
mov edx, dword ptr ds[ebp+0xc]
shl eax, 0x4
add ecx, eax
add edx, eax
lea eax, ptr [edi+eax*1]
FT
80593b3
s_m_a_mat.c:18
fld st0, st0
add ebx, 0x1
fmul st0, qword ptr ds[edx]
fadd st0, qword ptr ds[ecx]
fstp qword ptr ds[eax], st0
fld st0, st0
fmul st0, qword ptr ds[edx+0x8]
add edx, 0x10
fadd st0, qword ptr ds[ecx+0x8]
add ecx, 0x10
fstp qword ptr ds[eax+0x8], st0
add eax, 0x10
cmp ebx, 0x3
jnz 0x80593b3
202051584
67350528
0
80593d7
s_m_a_mat.c:17
add esi, 0x1
cmp esi, 0x3
jnz 0x805939e
0
g <6>
File Edit View Go Help
Previous Next 1 of 1
805c1f0
s_m_a_mat.c:17
fstp qword ptr ds[esi+eax*1+0x88], st0
fld st0, qword ptr ds[eax+edx*1+0x90]
fmul st0, st1
fadd st0, qword ptr ds[ecx+eax*1+0x90]
fstp qword ptr ds[esi+eax*1+0x90], st0
fld st0, qword ptr ds[eax+edx*1+0x98]
fmul st0, st1
fadd st0, qword ptr ds[ecx+eax*1+0x98]
fstp qword ptr ds[esi+eax*1+0x98], st0
fld st0, qword ptr ds[eax+edx*1+0xa0]
fmul st0, st1
fadd st0, qword ptr ds[ecx+eax*1+0xa0]
fstp qword ptr ds[esi+eax*1+0xa0], st0
fld st0, qword ptr ds[eax+edx*1+0xa8]
fmul st0, st1
fadd st0, qword ptr ds[ecx+eax*1+0xa8]
fstp qword ptr ds[esi+eax*1+0xa8], st0
fld st0, qword ptr ds[eax+edx*1+0xb0]
fmul st0, st1
fadd st0, qword ptr ds[ecx+eax*1+0xb0]
fstp qword ptr ds[esi+eax*1+0xb0], st0
fld st0, qword ptr ds[eax+edx*1+0xb8]
fmul st0, st1
fadd st0, qword ptr ds[ecx+eax*1+0xb8]
add eax, 0x30
jnz 0x805c1f0
101025792
0
805c283

Original Source

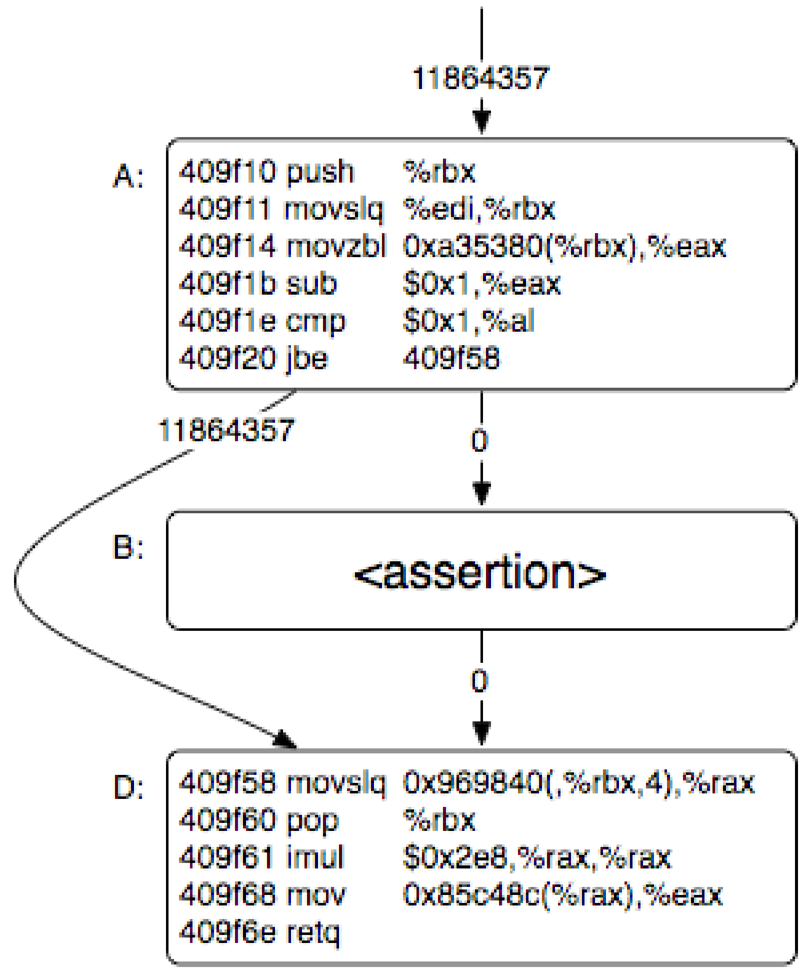
```
register int i,j;
for(i=0;i<3;i++) {
  for(j=0;j<3;j++) {
    c->e[i][j].real =
      a->e[i][j].real + s*b - e[i][j].real;
    c->e[i][j].imag =
      a->e[i][j].imag + s*b - e[i][j].imag;
  }
}
```

Case Study: 445.gobmk countlib()

Intel



GCC: **800% Faster!**



Case Study: 445.gobmk countlib()

```
/* Count the number of liberties of the string at pos. pos
   must not be empty. */
int
countlib(int str)
{
  ASSERT1(((board[str]) == WHITE || (board[str]) == BLACK),
          str);

  /* We already know the number of liberties. Just look it
   up. */
  return string[string_number[str]].liberties;
}
```

Case Study: 445.namd Patch::zeroforces()

- Both loads are loop-invariant (%ecx does not change)
- GCC 4.1 hoists the loads above the loop resulting in **6912** loads versus **1.6M** from ICC
- GCC's code is 25% faster

ICC 9.1 –O3 -mtune

```
head:
mov 0x74(%ecx),%esi
fstl (%eax,%esi,1)
fstl 0x8(%eax,%esi,1)
fstl 0x10(%eax,%esi,1)
mov 0x78(%ecx),%esi
fstl (%eax,%esi,1)
fstl 0x8(%eax,%esi,1)
fstl 0x10(%eax,%esi,1)
add $0x18,%eax
add $0x1,%edx
cmp (%ecx),%edx
jl head
```

Case Study: 462.libquantum

- For a simple, linear for loop, ICC inserts a speculative load that often fails (still don't fully understand why)
- Example of over-aggressive optimization
- GCC's version executes 60% less instructions and 46% less memory reads, 32.5% faster

IA-64 GCC 4.1.0 –O2

```
// quantum_addscratch():
for(i=0; i<reg->size; i++) {
  l = reg->node[i].state<<bits;
  reg->node[i].state = l;
}
```

```
HEAD:
{ld8 r14=[r33];;
nop.m 0x0
shl r14=r14,r32;;}
{st8 [r33]=r14,16
nop.i 0x0
br.cloop.sptk.few HEAD;;}
```

Case Study: 464.h264ref SetCoeffAndReconstruction8x8()

- Innermost loop of quadruply nested loop
- GCC 4.1.0 recomputes address calculations each iteration
- 93% less stack writes, 170% faster

GCC 3.4.6 -O2

```
head:
mov  (%edx,%ebx,4),%eax
mov  %eax,(%ecx,%ebx,4)
inc  %ebx
cmp  $0x40,%ebx
jle  head
```

GCC 4.1.0 -O2

```
head:
mov  0xffffffac(%ebp),%edx
mov  (%edi,%edx,1),%eax
mov  0xffffff9c(%ebp),%edx
mov  (%eax,%esi,1),%eax
mov  (%eax,%ebx,1),%eax
mov  %eax,0xffffffb0(%ebp)
mov  (%edi,%edx,1),%eax
mov  0xffffffb0(%ebp),%edx
mov  (%eax,%esi,1),%eax
mov  (%eax,%ebx,1),%eax
mov  (%eax,%ecx,1),%eax
mov  %eax,(%edx,%ecx,1)
add  $0x4,%ecx
cmp  $0x104,%ecx
jne  head
```

Observations

- Finding performance bugs is easy!
  - "Big" anomalies take 1-2 hours of analysis
  - Smaller differences more abundant
- Compilers *still* do silly things
  - Direct jump to next instruction
  - Jump to return instruction
  - Spill all registers, then immediately refill

Future Work

- Collect much more detailed profiles
  - Hardware perf data
  - Load-use distance
  - Stack/heap/text memory references
- Use binary matching to correlate profiles at finer grain (loop, bbl)
- Automate regression testing

Questions?