Chainsaw: Using Binary Matching for Relative Instruction Mix Comparison

Tipp Moseley

Dirk Grunwald

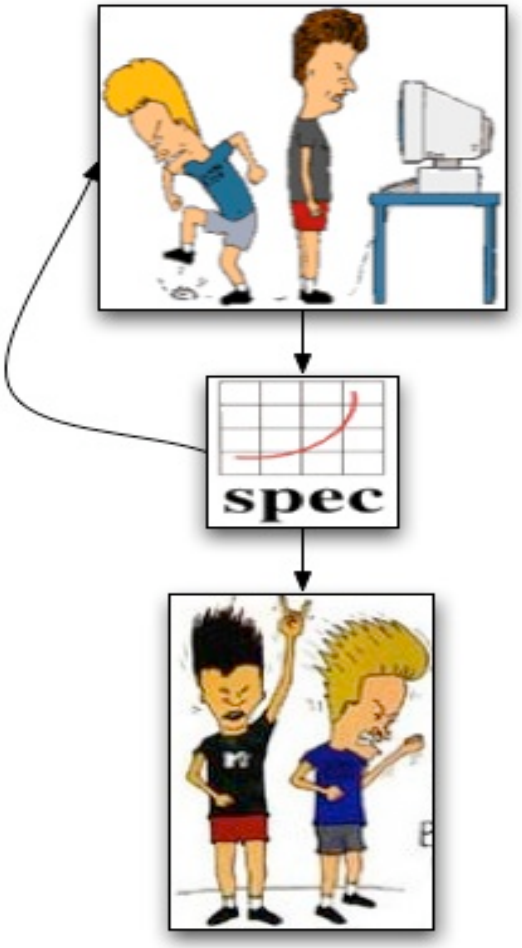
U. Of Colorado

Ramesh Peri

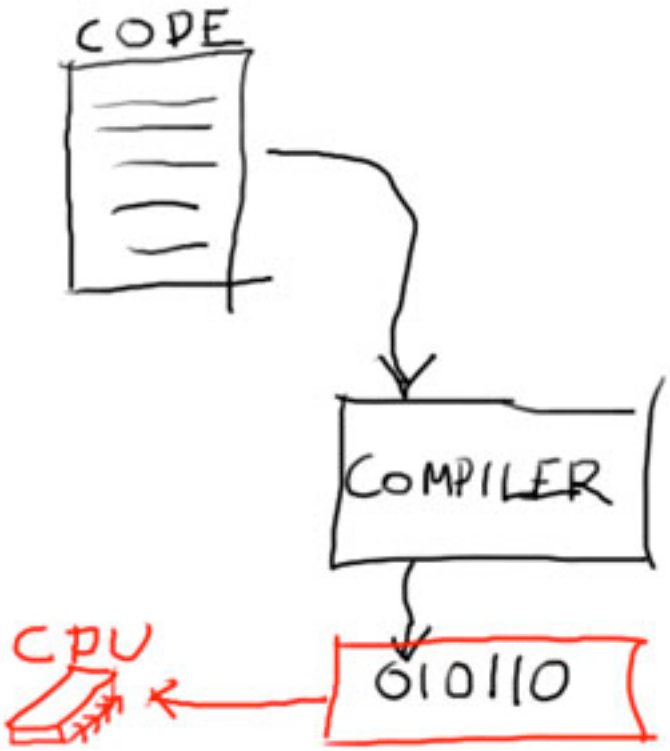
Intel

Primitive Development Cycle for Compilers

1. Come up with an idea
2. Test it
3. If nothing improved, go to 1
4. Celebrate

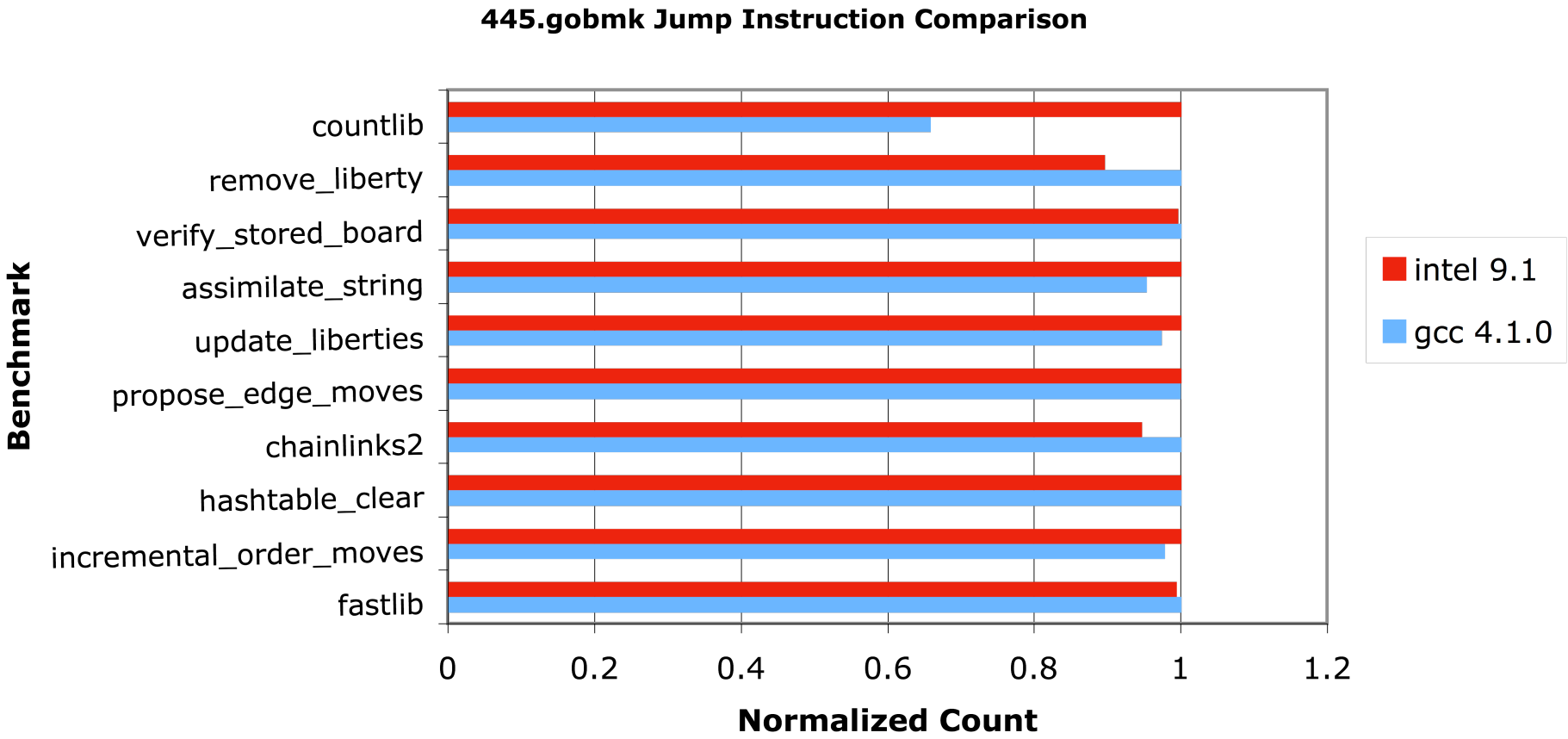


Understanding Transformations Is Hard



- Code size vs. control flow
  - Inlining, loop transformations, superblocks, if-conversion
- Architectural Features
  - Alignment, speculation, instruction selection, multilevel memory hierarchy, prefetching

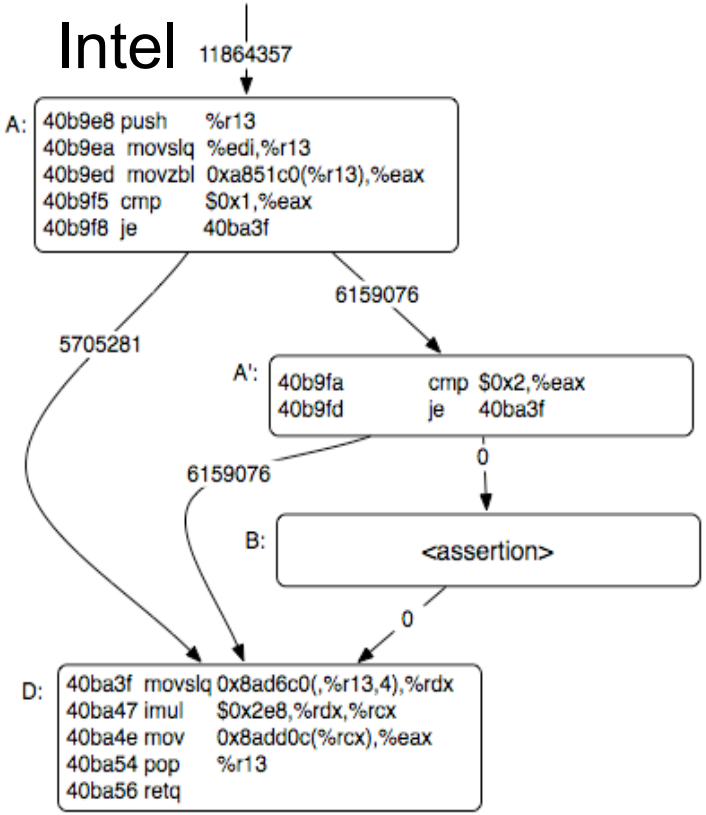
Unpredictable Results



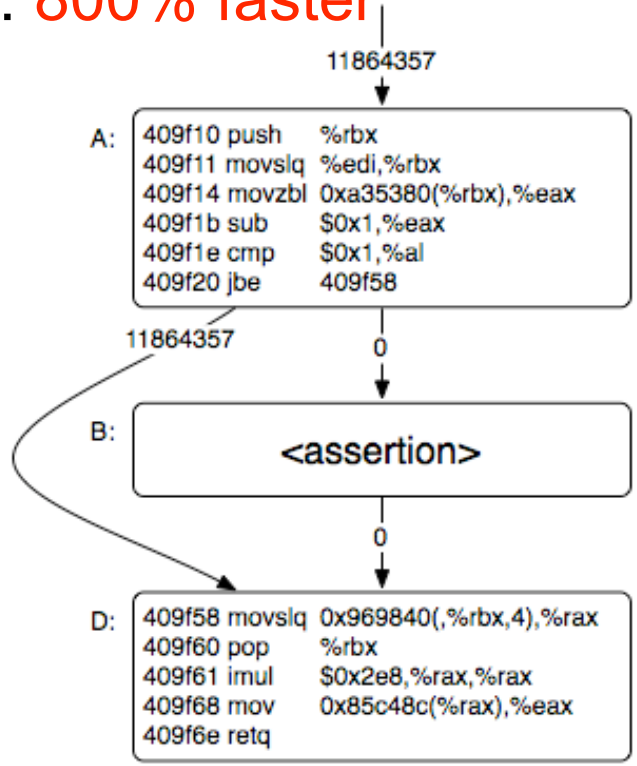
Motivating Example

```
assert(x == 1 || x == 2);
return array[x];
```

Intel



GCC: 800% faster



What if…

…this case is insignificant in the compiler vendor's regression suite?

…it is in the kernel of an important customer's application?

We need better tools!

Chainsaw Approach

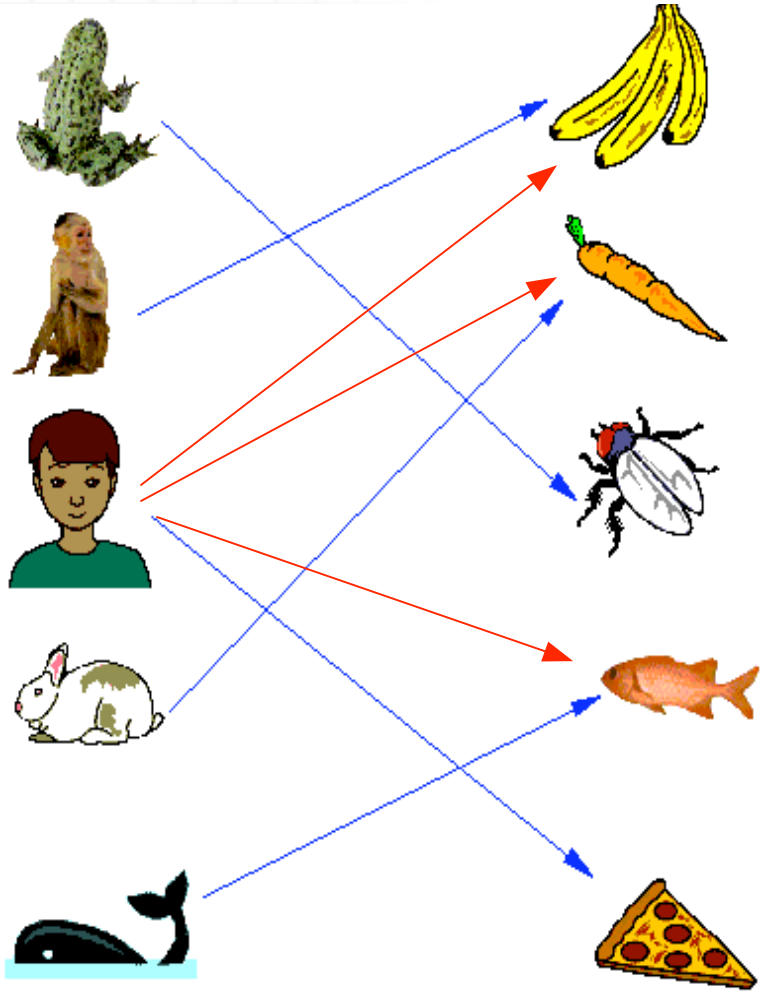
- Chainsaw offers high resolution relative performance comparison for arbitrarily compiled binaries (including IPO)
- Match a log of execution events from two differently compiled programs
  - Use BASE as reference point
- Compare profiles of matched intervals to identify outliers

Matching Resolution

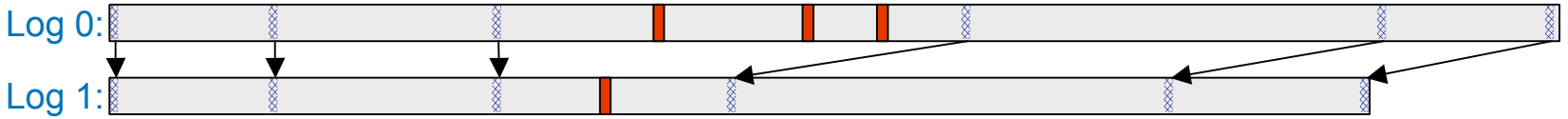
- Program- and function-level
  - Too big; miss opportunities
- Instructions and basic blocks
  - Too small; automatic comparisons are too noisy
- Loops
  - Just right; corresponds well with source
  - Be conservative; match innermost loop that makes sense

Challenges to Binary Matching

- Compiler can perform very aggressive optimization
  - Unrolling, peeling, splitting, merging, multiple entrypoints, versioning, …
- Larger functions can have many loops that look identical



Log-based Event Matching



- ***Event***: call, return, or backedge
- ***Anchor***: a pair of events that
  - Occurs the same number of times in both logs
  - Is semantically similar in both logs (e.g., same symbol names)
  - Has the same respective ordering to all other anchors in both logs
- ***Interval***: execution history between two anchored events

- Similar to sequence alignment, but much simpler
  - Don't deal with insertions/deletions; only align tokens that occur same number of times

Log-based Event Matching

Compilers can sometimes order function calls arbitrarily, so semantics and count alone are insufficient

- x = f() + g();
- LOG 0: CALL f, RET, CALL g, RET
- LOG 1: CALL g, RET, CALL f, RET

Ordering Signature

- Two anchors have the same ordering signature (OrdSig) **iff** they occur the same number of time and in the same order in both logs
- For each log, an ordering table (OrdTable) is filled with the OrdSig for every pair of events in the log

Computing Ordering Signatures

Traverse the log and maintain a stack-like structure ordering events

1. If the event, *i*, is not on the stack, run Update(*i, j*) for every *j* residing on the stack. Place i on the stack.
2. If the event, *i*, is on the stack, run Update(*i, j*) for every event *j* that has been seen since the last time *i* was seen. Move *i* to the back of the stack.
3. When the end of an interval is reached, run Update(*i, j*) for every event pair where *i* has been seen more recently than *j* on the stack.

Computing Ordering Signatures

Update function:

Modify ordtable[i, j] and ordtable[j, i] by xor with event count, circular shift, and multiply with large prime seed.

Ordering Signatures Example

Log: ABCBBA

	A	B	C
A	2	0x21	0x67
B	0x11	3	0x03
C	0xeb	0x75	1

Log: AXYBCXXBYBYAX

	A	B	C	X	Y
A	2	0x21	0x67	0x9a	0x1b
B	0x11	3	0x03	0xd	0x9f
C	0xeb	0x75	1	0x81	0x8c
X	0xb2	0x86	0x63	4	0x47
Y	0x19	0xfe	0x88	0x5e	3

Ordering Signatures Example

Initial state:

Sequence: ABCBBA

stack = []

	A	B	C
A	0	0	0
B	0	0	0
C	0	0	0

Ordering Signatures Example

Step 1:
Sequence: **A**BCBBA
Actions: OrdTable[A, A]++
stack = [A]

	A	B	C
A	1	0	0
B	0	0	0
C	0	0	0

Ordering Signatures Example

Step 2:

Sequence: A**B**CBBA

Actions: OrdTable[B, B]++, Update(B, A)

stack = [A, B]

	A	B	C
A	1	0xa1	0
B	0xb7	1	0
C	0	0	0

Ordering Signatures Example

Step 3:
Sequence: AB**C**BBA
Actions: OrdTable[C, C]++, Update(C, B),
Update(C, A)
stack = [A, B, C]

	A	B	C
A	1	0xa1	0xa1
B	0xb7	1	0xa1
C	0xb7	0xb7	1

Ordering Signatures Example

Step 4:

Sequence: ABC**B**BA

Actions: OrdTable[B, B]++, Update(B, C)

stack = [A, C, B]

	A	B	C
A	1	0xa1	0xa1
B	0xb7	2	0x5a
C	0xb7	0x13	1

Ordering Signatures Example

Step 5:
Sequence: ABCB**B**A
Actions: OrdTable[B, B]++
stack = [A, C, B]

	A	B	C
A	1	0xa1	0xa1
B	0xb7	3	0x5a
C	0xb7	0x13	1

Ordering Signatures Example

Step 6:

Sequence: ABCBB**A**

Actions: OrdTable[A, A]++, Update(A, B)

Update(A, C)

stack = [C, B, A]

	A	B	C
A	2	0x12	0x4e
B	0x61	3	0x5a
C	0x9f	0x13	1

Ordering Signatures Example

End of interval:

Sequence: <>

Actions: Update(A, B), Update(A, C),
Update(B, C)

stack = [C, B, A]

	A	B	C
A	2	0x21	0x67
B	0x11	3	0x03
C	0xeb	0x75	1

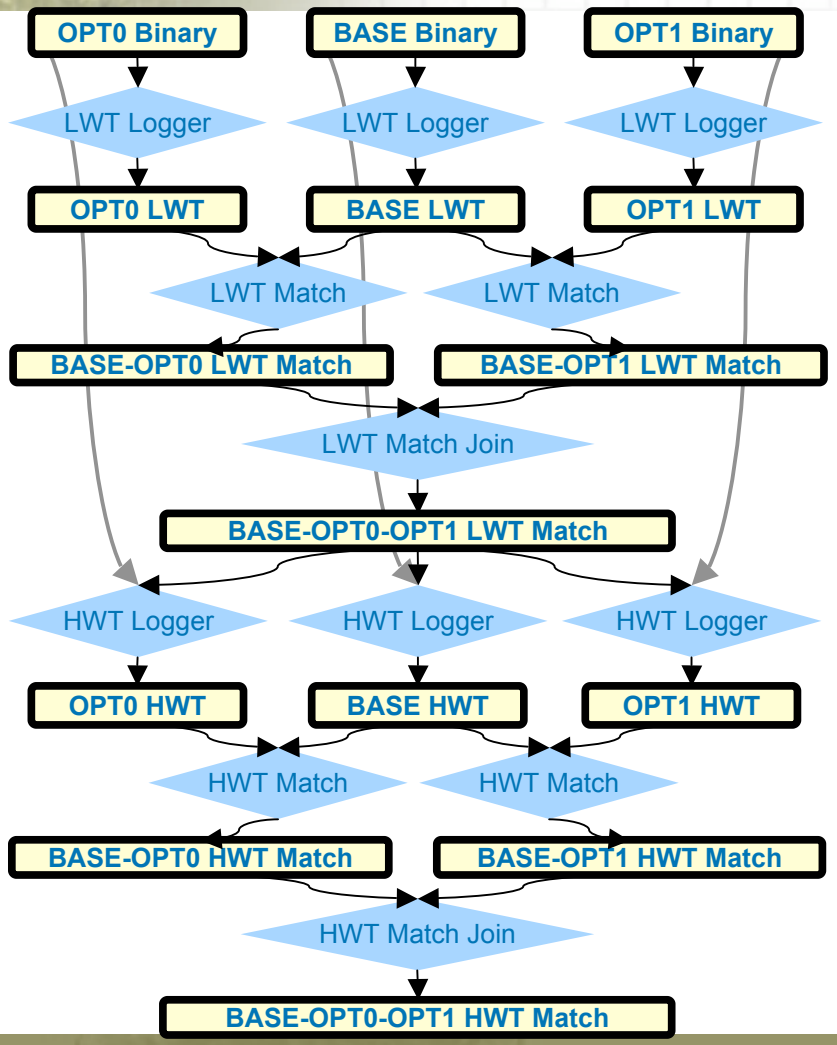
Selecting Matches

- Score potential anchors
  - Sort by number of common OrdSigs in each log
  - Event counts must be equal
  - Events must be semantically similar
- Select first match with highest score
- Subsequent matches must be legally ordered with respect to previous matches

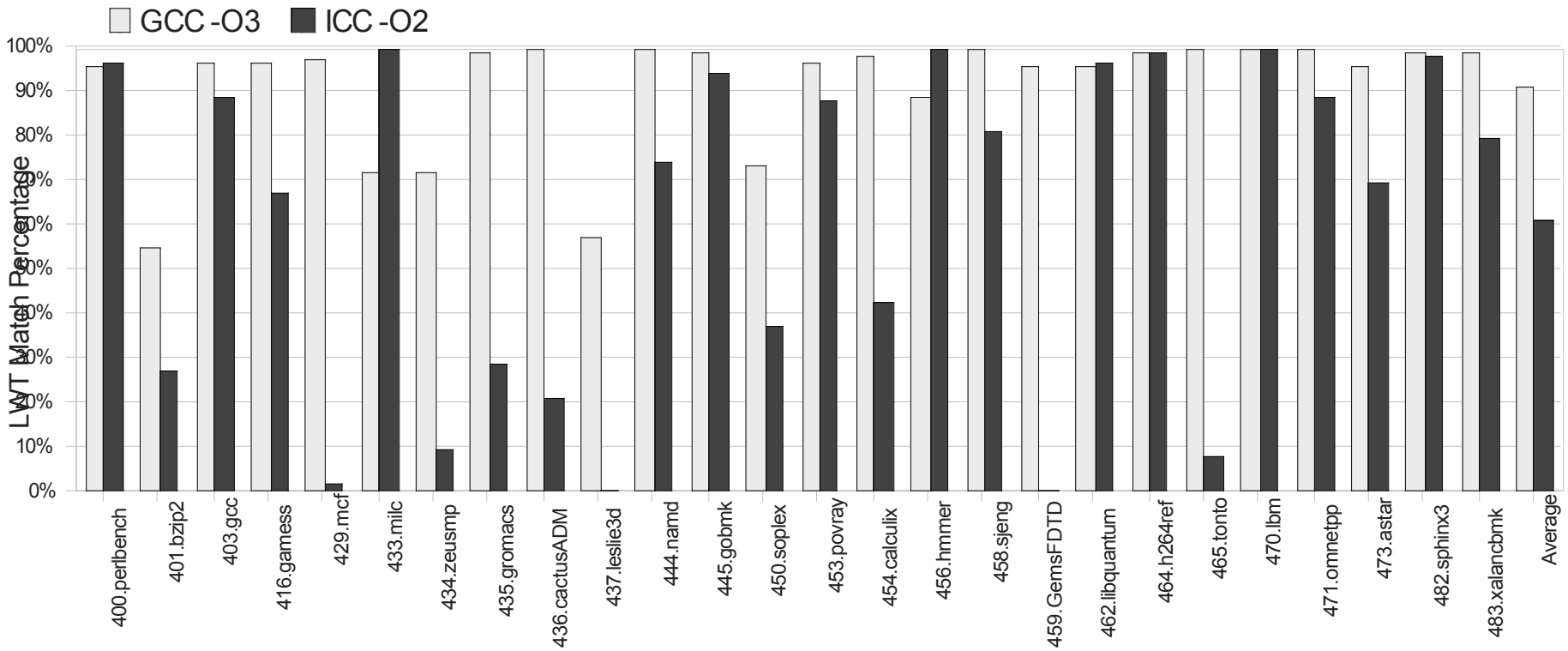
Real-world Challenges

- Event logs are enormous and slow to collect
  - Two-pass algorithm
    - Lightweight log (LWT)
      - Only CALL, RET, and non-inner loop BACKEDGE
    - Heavyweight log (HWT) – inner loops too
- Comparing optimized (OPT) binaries leads to ambiguity
  - Compare each log to unoptimized (BASE) and infer mapping

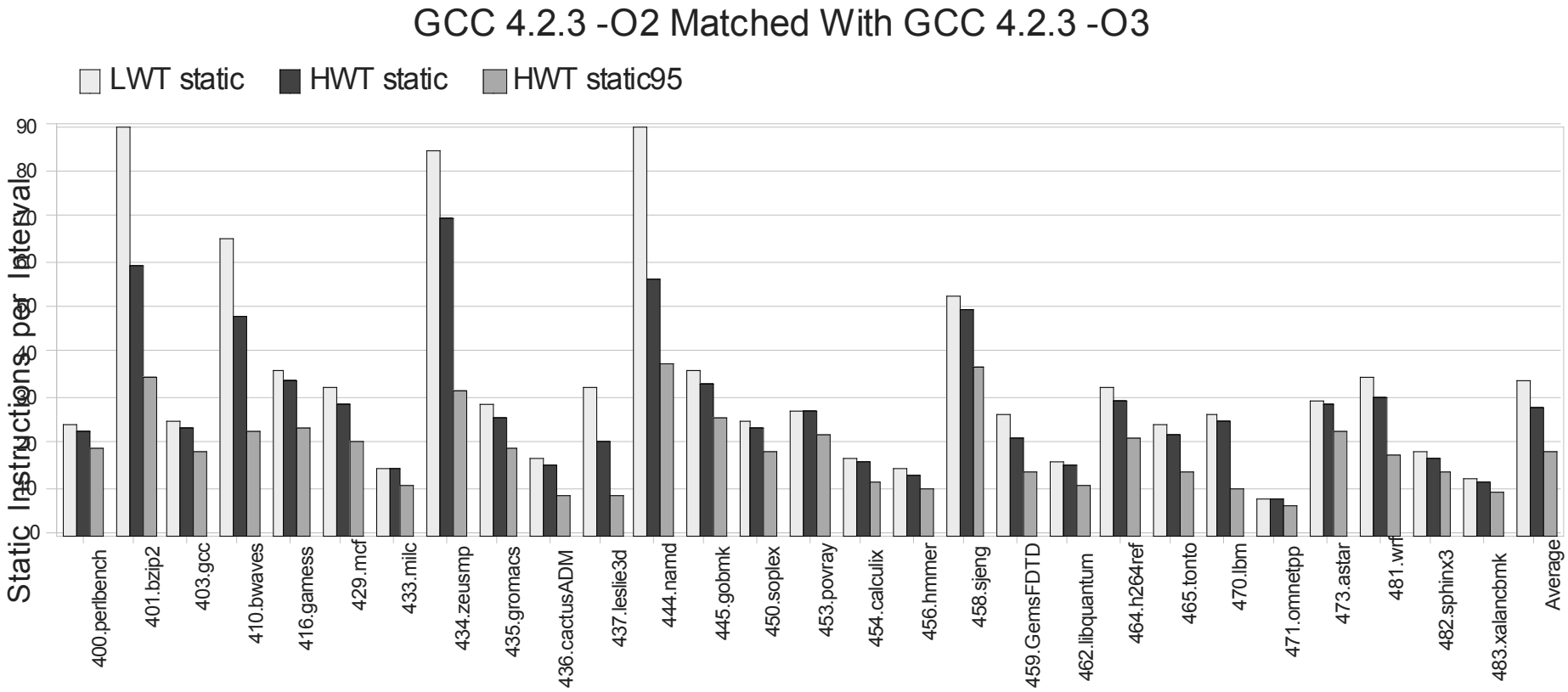
Chainsaw Overview



LWT Matching Rate

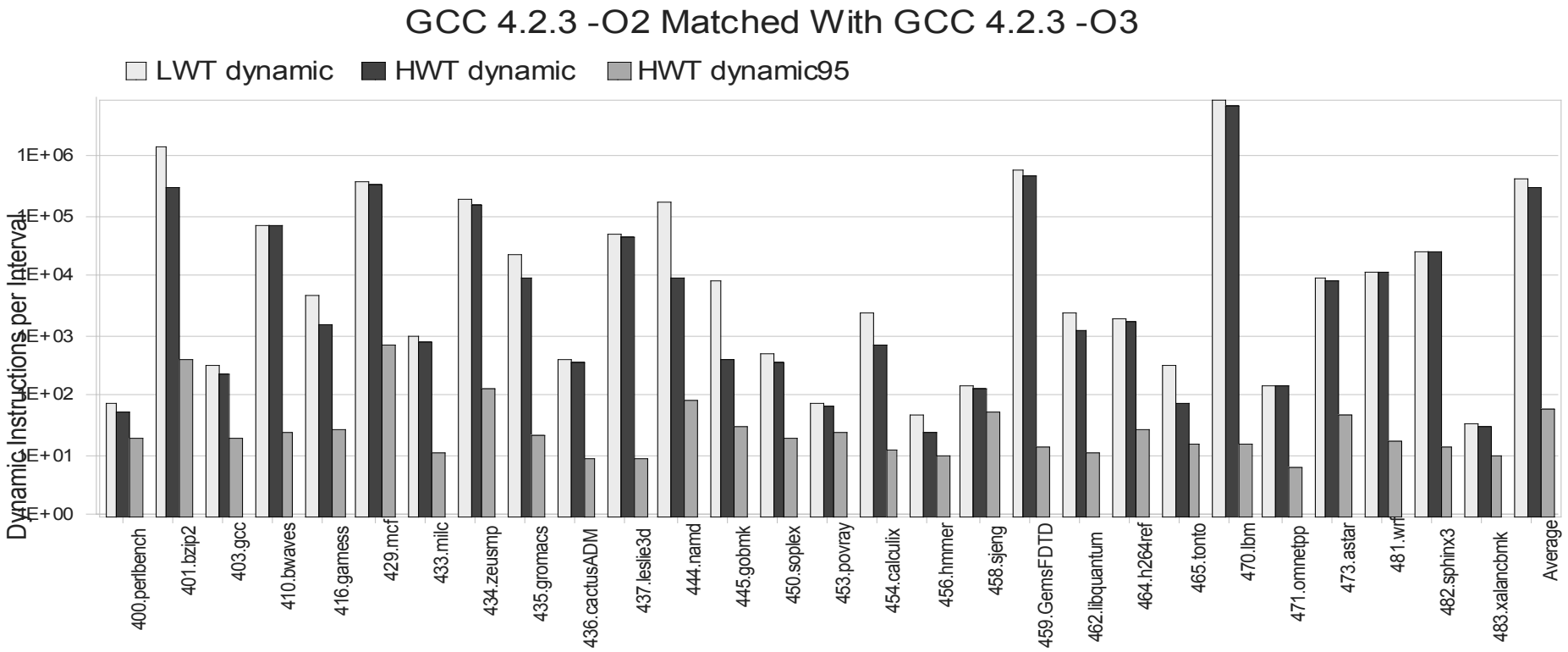


Static Instructions per Interval



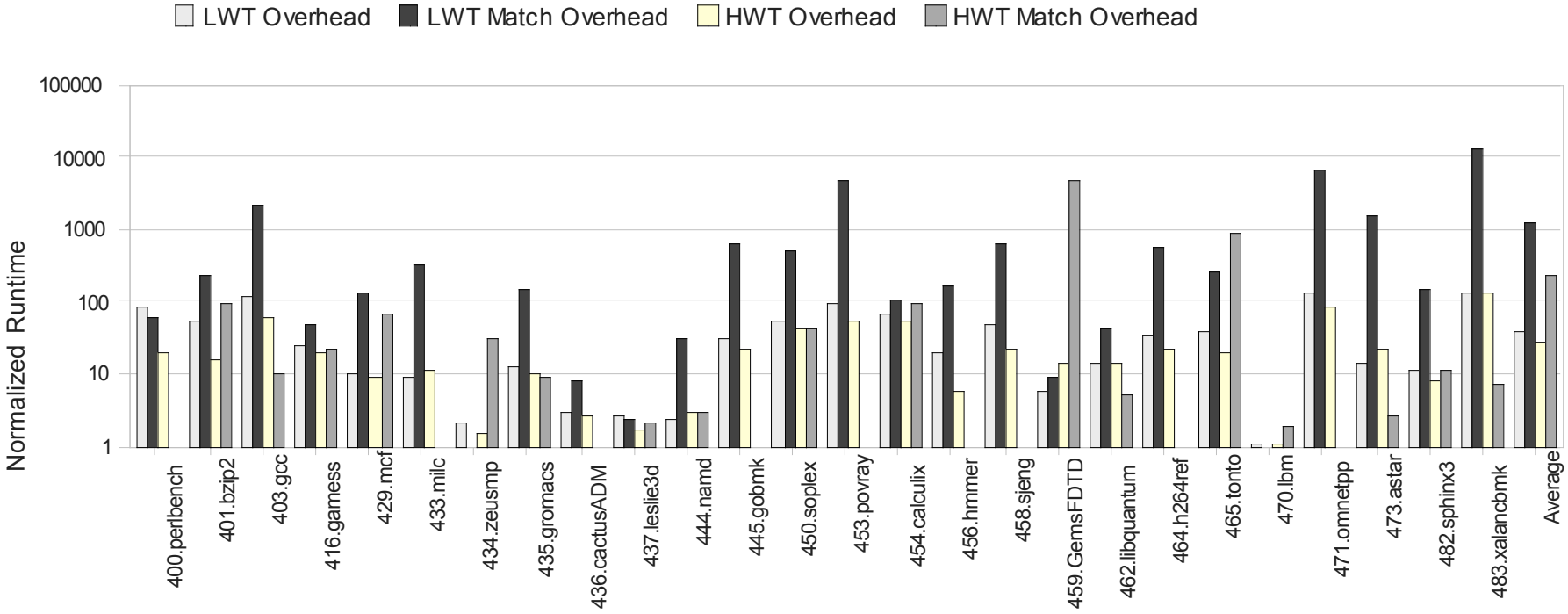
- 95% of intervals have less than 19 static instructions

Dynamic Instructions per Interval



- 95% of intervals have less than 64 dynamic instructions

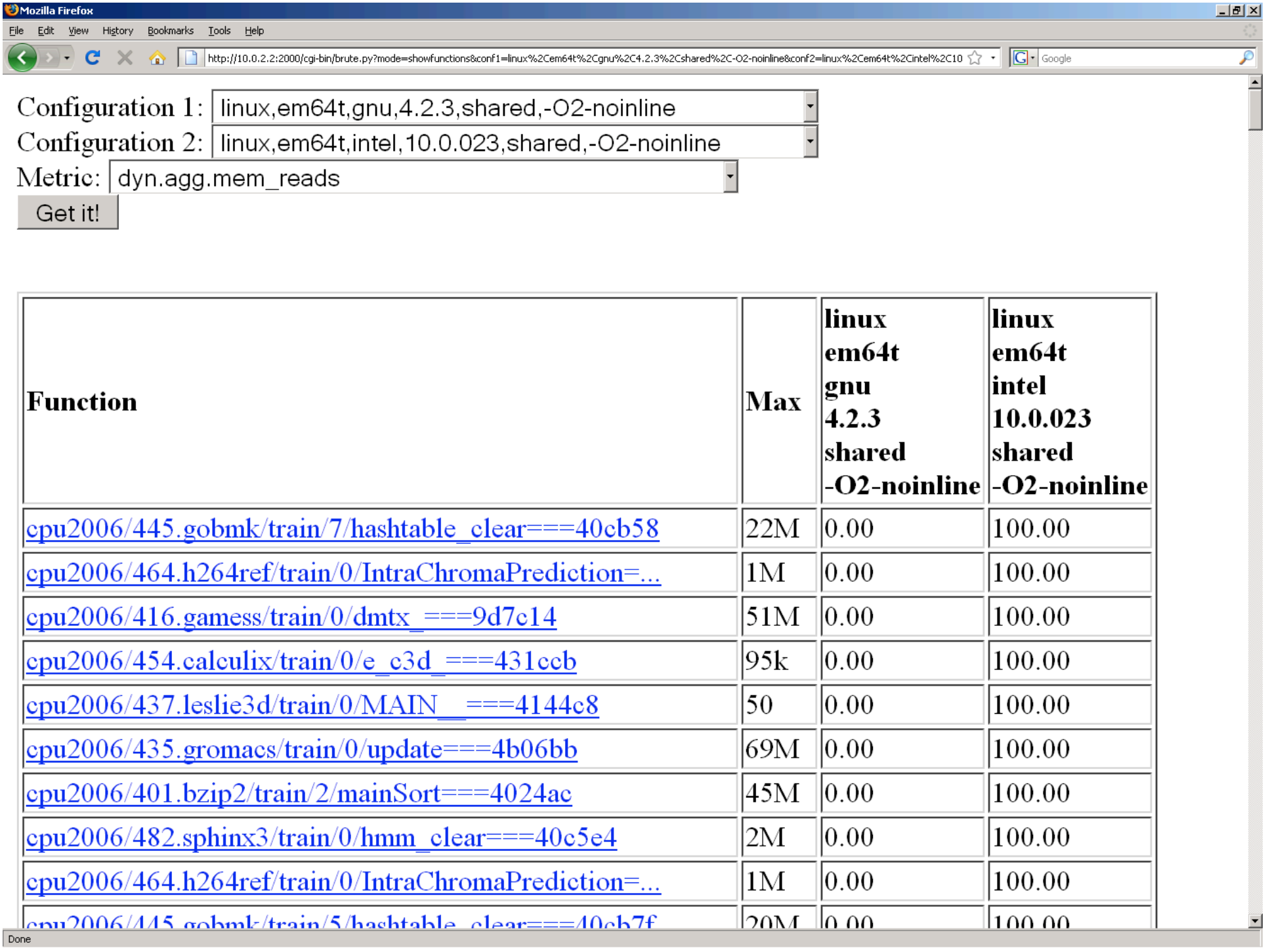
Analysis Overhead



- Total overhead around 1600X

Storage Overhead

- SPEC2006 test suite takes about 1.2GB



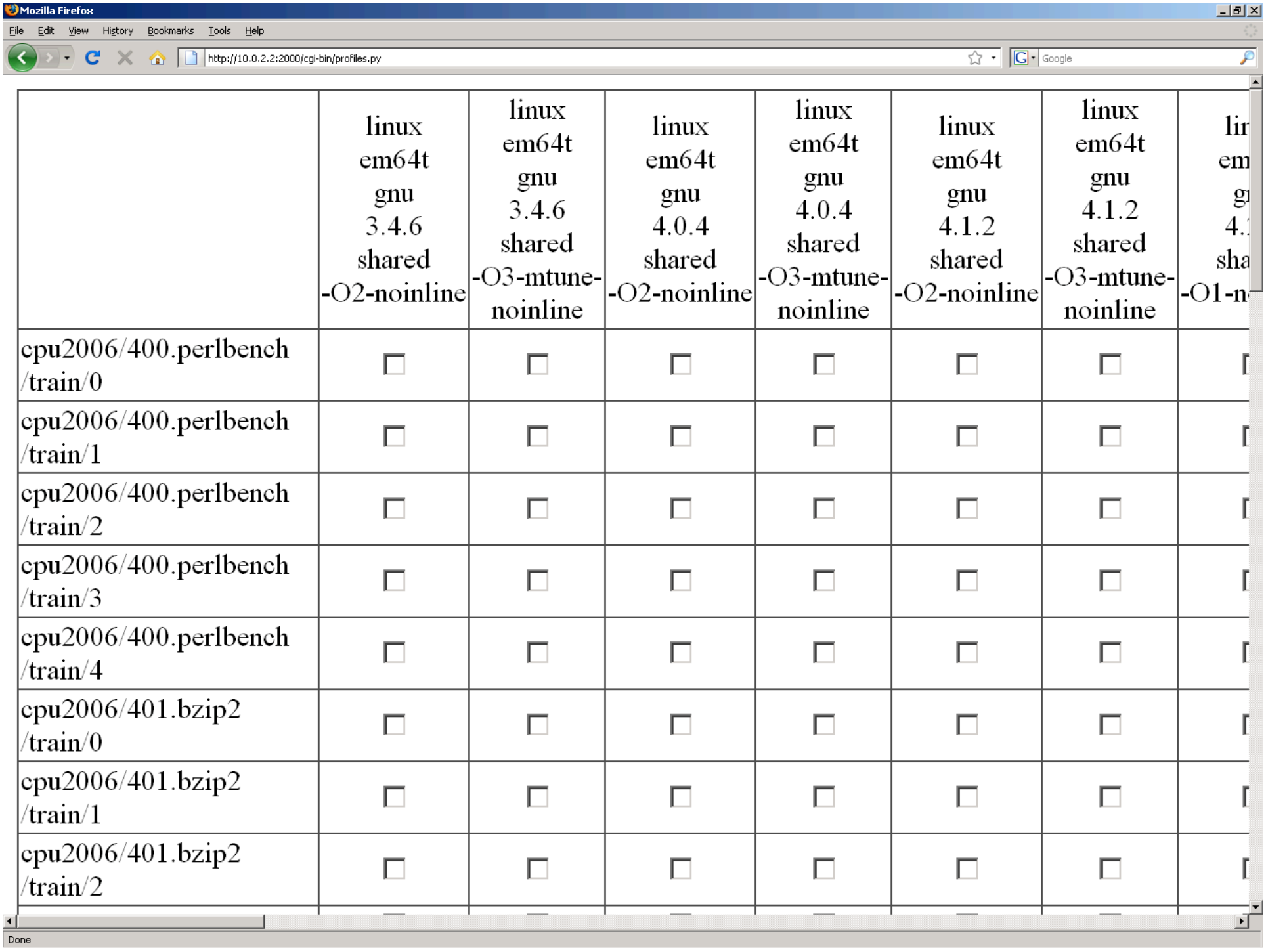
Configuration 1: linux,em64t,gnu,4.2.3,shared,-O2-noinline

Configuration 2: linux,em64t,intel,10.0.023,shared,-O2-noinline

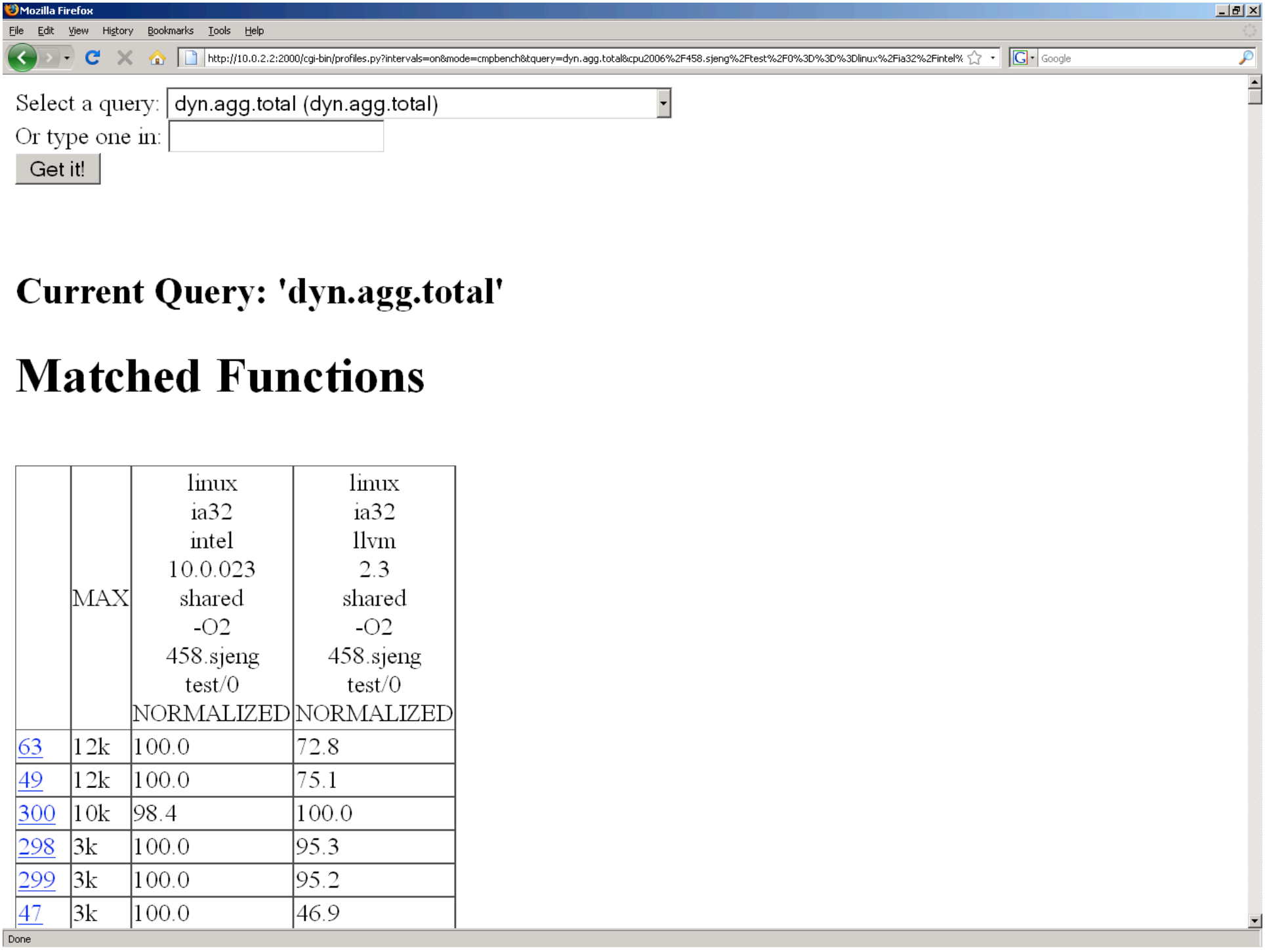
Metric: dyn.agg.mem_reads

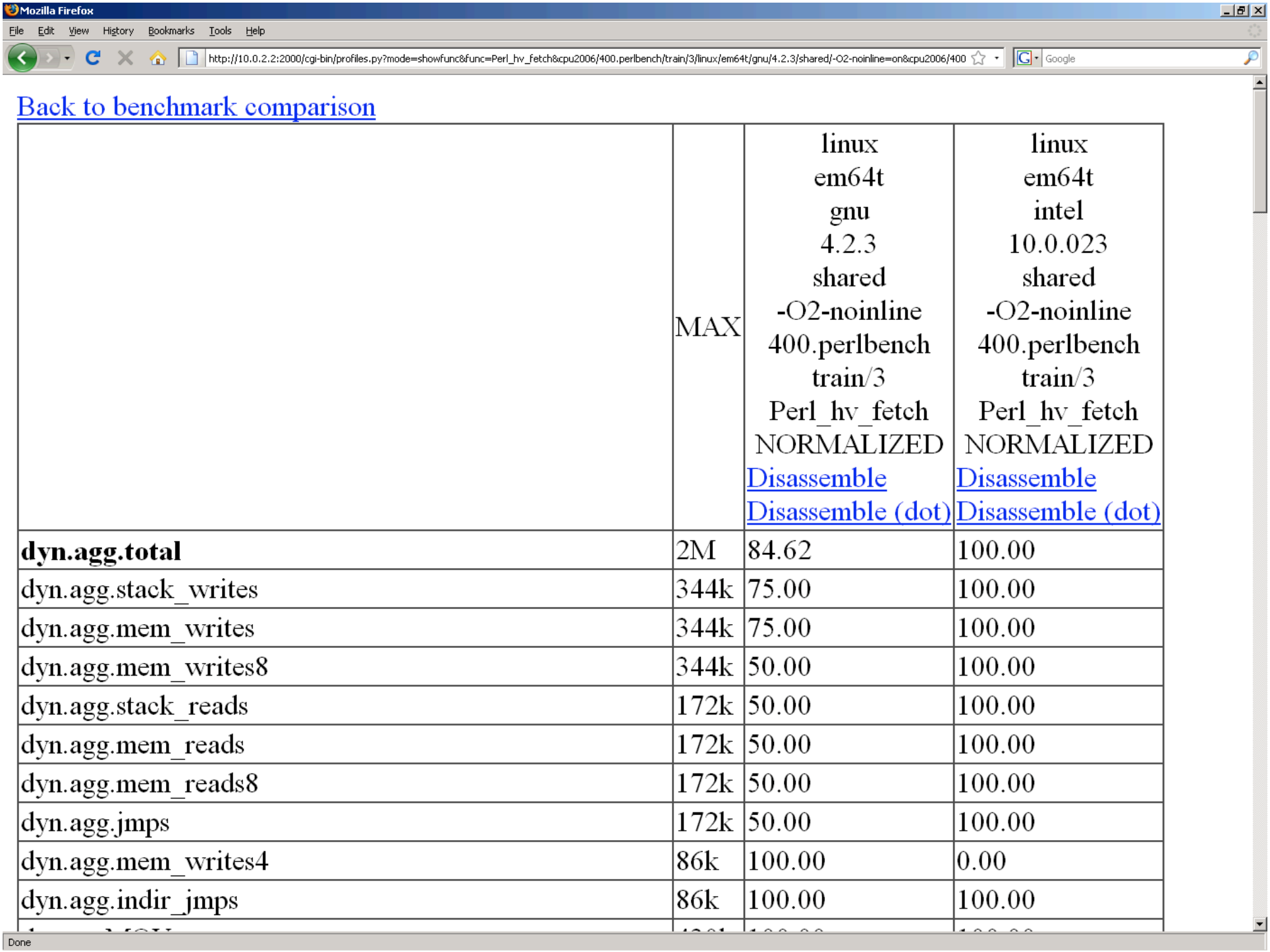
Get it!

Function	Max	linux em64t gnu 4.2.3 shared -O2-noinline	linux em64t intel 10.0.023 shared -O2-noinline
cpu2006/445.gobmk/train/7/hashtable_clear===40cb58	22M	0.00	100.00
cpu2006/464.h264ref/train/0/IntraChromaPrediction=...	1M	0.00	100.00
cpu2006/416.gamess/train/0/dmtx_===9d7c14	51M	0.00	100.00
cpu2006/454.calculix/train/0/e_c3d_===431ccb	95k	0.00	100.00
cpu2006/437.leslie3d/train/0/MAIN__===4144c8	50	0.00	100.00
cpu2006/435.gromacs/train/0/update===4b06bb	69M	0.00	100.00
cpu2006/401.bzip2/train/2/mainSort===4024ac	45M	0.00	100.00
cpu2006/482.sphinx3/train/0/hmm_clear===40c5e4	2M	0.00	100.00
cpu2006/464.h264ref/train/0/IntraChromaPrediction=...	1M	0.00	100.00
cpu2006/445.gobmk/train/5/hashtable_clear===40cb7f	20M	0.00	100.00



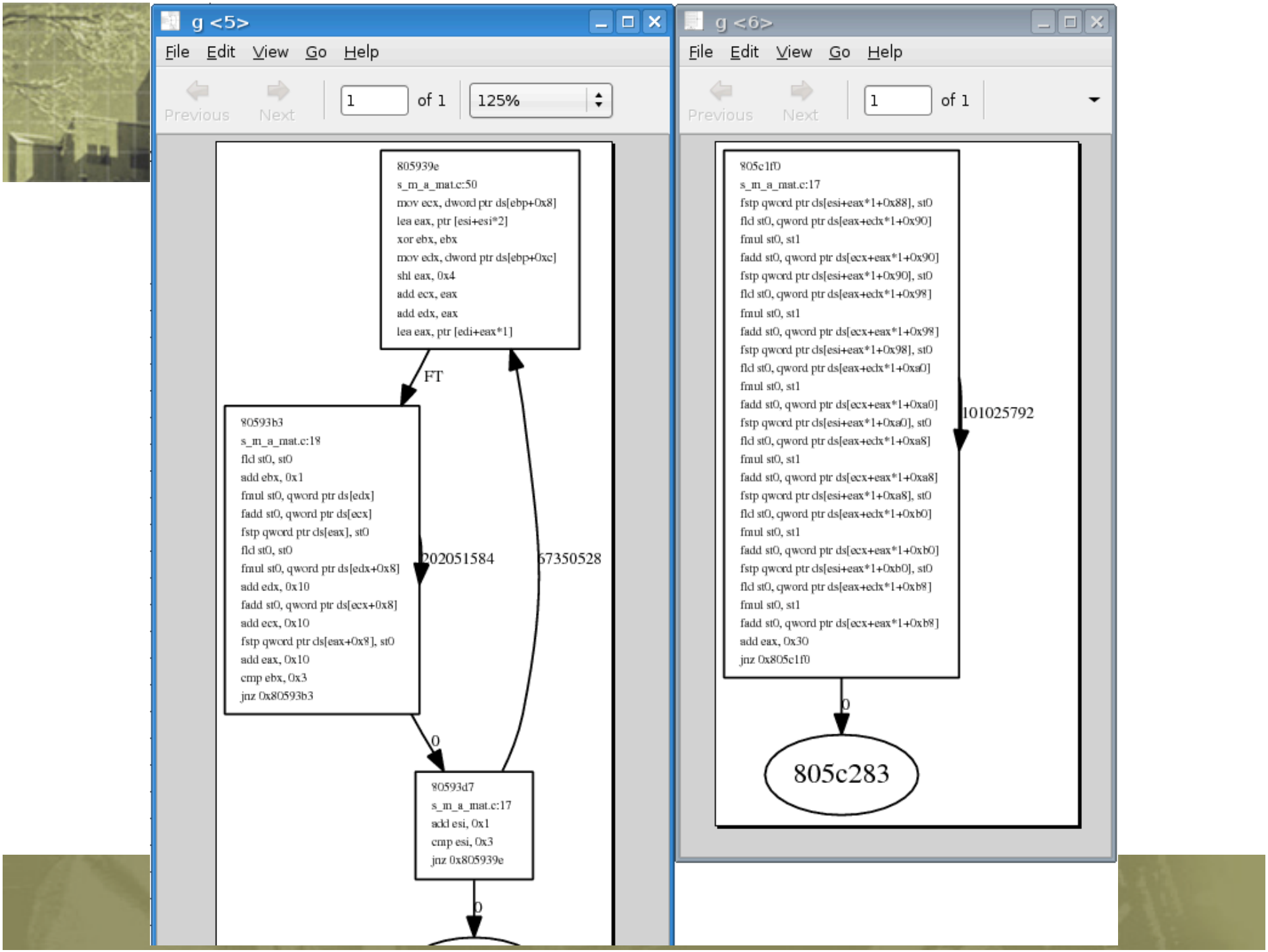
	linux em64t gnu 3.4.6 shared -O2-noinline	linux em64t gnu 3.4.6 shared -O3-mtune-noinline	linux em64t gnu 4.0.4 shared -O2-noinline	linux em64t gnu 4.0.4 shared -O3-mtune-noinline	linux em64t gnu 4.1.2 shared -O2-noinline	linux em64t gnu 4.1.2 shared -O3-mtune-noinline	lir em g 4. sha -O1-n
cpu2006/400.perlbench /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	☐	☐	☐	☐	☐	☐	☐


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://10.0.2.2:2000/cgi-bin/profiles.py?intervals=on&mode=cmpbench&query=dyn.agg.total&cpu2006%2F458.sjeng%2Ftest%2F0%3D%3D%3Dlinux%2Fia32%2Fintel%
Google
Select a query: dyn.agg.total (dyn.agg.total)
Or type one in:
Get it!
Current Query: 'dyn.agg.total'
Matched Functions
MAX
linux ia32 intel 10.0.023 shared -O2 458.sjeng test/0 NORMALIZED
linux ia32 llvm 2.3 shared -O2 458.sjeng test/0 NORMALIZED
63 12k 100.0 72.8
49 12k 100.0 75.1
300 10k 98.4 100.0
298 3k 100.0 95.3
299 3k 100.0 95.2
47 3k 100.0 46.9
Done



[Back to benchmark comparison]

	MAX	linux em64t gnu 4.2.3 shared -O2-noinline 400.perlbench train/3 Perl_hv_fetch NORMALIZED Disassemble Disassemble (dot)	linux em64t intel 10.0.023 shared -O2-noinline 400.perlbench train/3 Perl_hv_fetch NORMALIZED Disassemble Disassemble (dot)
dyn.agg.total	2M	84.62	100.00
dyn.agg.stack_writes	344k	75.00	100.00
dyn.agg.mem_writes	344k	75.00	100.00
dyn.agg.mem_writes8	344k	50.00	100.00
dyn.agg.stack_reads	172k	50.00	100.00
dyn.agg.mem_reads	172k	50.00	100.00
dyn.agg.mem_reads8	172k	50.00	100.00
dyn.agg.jmps	172k	50.00	100.00
dyn.agg.mem_writes4	86k	100.00	0.00
dyn.agg.indir_jmps	86k	100.00	100.00


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+ecx*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+ecx*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+ecx*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+ecx*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+ecx*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+ecx*1+0xb8]
fmul st0, st1
fadd st0, qword ptr ds[ecx+eax*1+0xb8]
add eax, 0x30
jnz 0x805c1f0
101025792
0
805c283

Case Studies

- ISAs
  - IA-32, EM64T, IA-64
- Compilers
  - GCC 3.2.3, 3.4.6, 4.0.4, 4.1.2, 4.2.4, 4.3.0
  - ICC 9.1, 10.0, 11.0
  - LLVM 2.3 (IA-32 only)
- Machines
  - 3.0 GHz Core 2 Xeon
  - 900 MHz Itanium 2 McKinley

Algorithmic Impact

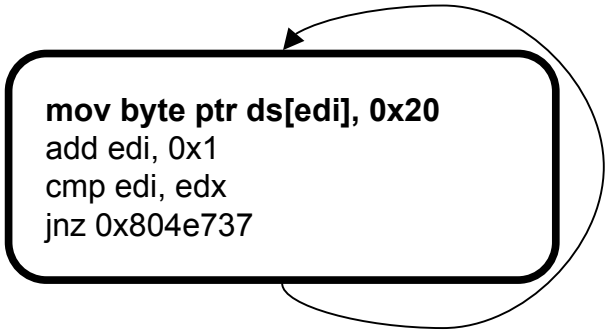
```
// Source: O(N)
int gx;
void Tailcall(int c) {
  if (c > 0) {
    gx += 100;
    Tailcall(c - 1);
  }
}
```

```
// ICC: O(N)
while (c > 0) {
  gx += 100;
   c--;
}
```

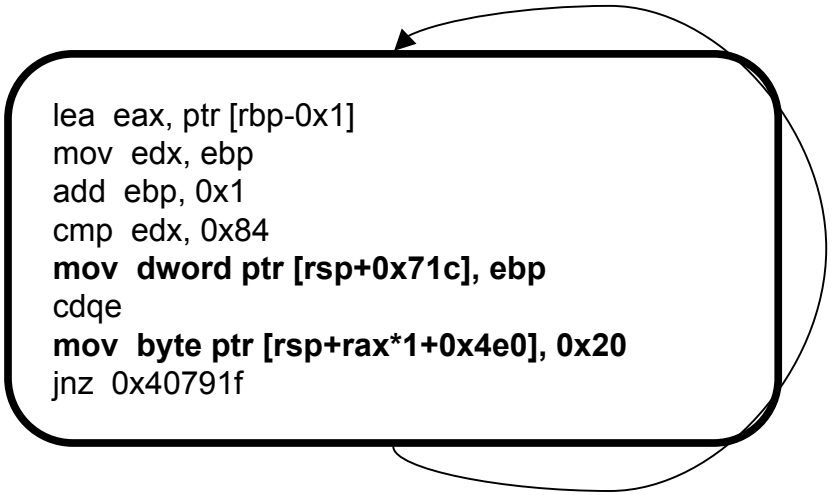
```
// GCC: O(1)
if (c > 0) {
  gx += c * 100;
}
```

Cross-ISA Comparison

```
do i=1,132
  text(i:i)=' '
```



(a) IA-32 GCC 4.2.3 -O2



(b) EM64T GCC 4.3.0 -O2

Data Alignment

Metric	Max Count	ICS 10.0 –O2	ICS 10.0 –O3 –mtune
dyn.agg.total	5B	100%	81.75%
dyn.agg.mem_reads	1B	100	84.09
dyn.agg.mem_writes	738M	100	94.74
dyn.cat.X87_ALU	4B	100	0
dyn.cat.DATAXFER	2B	20.54	100
dyn.cat.SSE	1B	0	100
dyn.cat.COND_BR	19M	100	100
dyn.pmu.MISALIGN_MEM_REF	**347k**	**0**	**100**

- With –O3 enabled, ICS doesn't align array variable on the stack for loop from 444.namd
  - 87% slower

Performance Regression

```
// 464.h264ref loop in image.c:1441
jj = max(0, min(s->size_y - 1, j)
```

- GCC 4.2.3 uses CMOV instruction
- GCC 4.3.0 uses branches
  - 90% slower,
    25X more cond. branches

Inlining

```
for(XalanDOMString::size_type i = start; i < n; ++i) {
     accumCharUTF(chars[i]);
   }
}
```

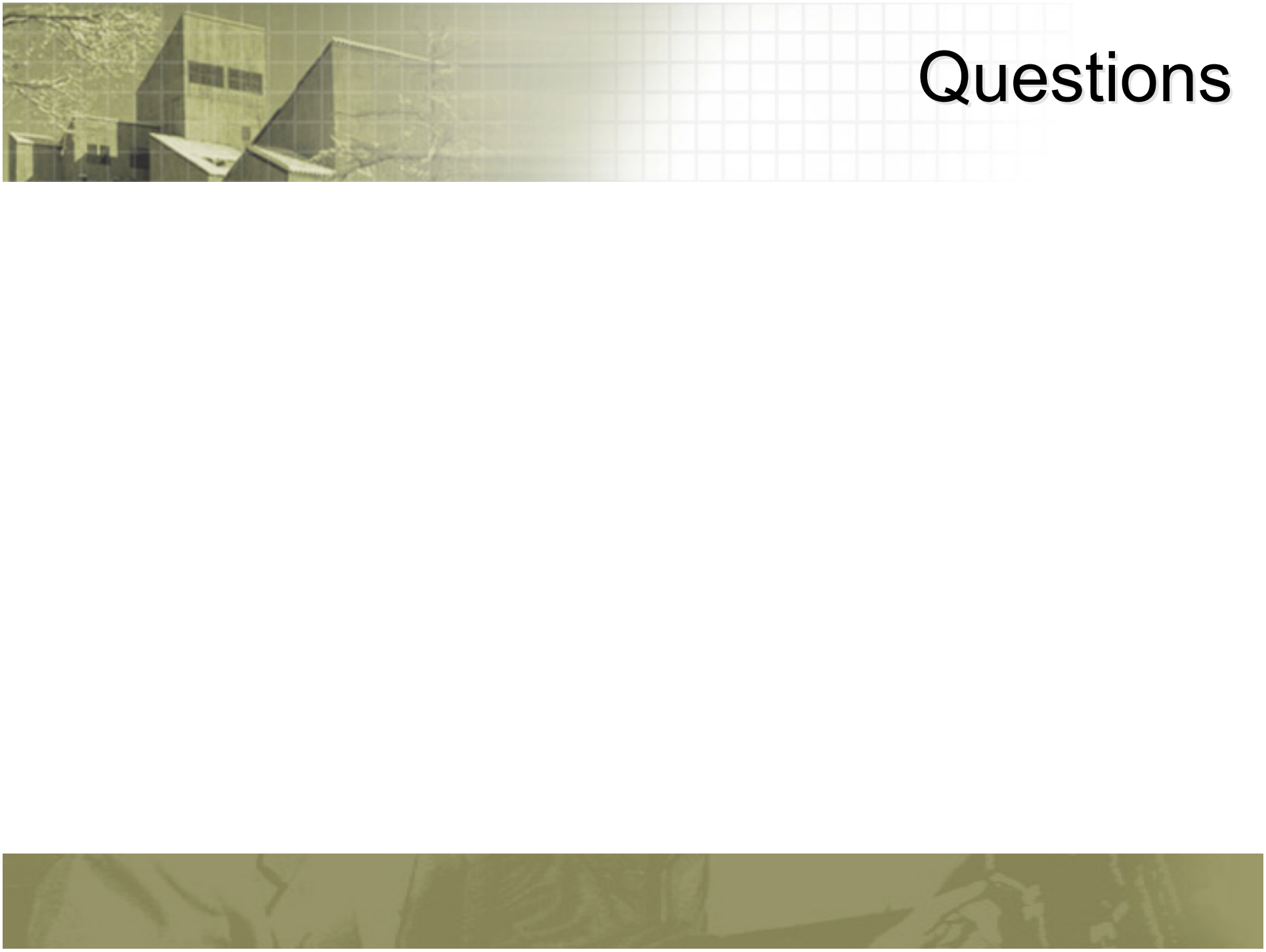
- ICPC inlines accumCharUTF, does not unroll
- LLVM unrolls loop 8 times
  - 8 calls to `accumCharUTF`
  - 5X more space, 2X more dyn. insts.,
    3X more mem reads, 7X more mem writes

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

Summary

- Chainsaw: Infrastructure for relative profile comparison tools
  - Binary matching
  - Web-based profile viewer


Questions