LLVM Performance Improvements and Headroom

Gerolf Hoflehner
Apple

LLVM Developers' Meeting 2015
San Jose, CA

Messages

- Tuning and focused local optimizations
- Advancing optimization technology
- Getting inspired by ‘heroic’ optimizations
- Exposing performance opportunities to developers

Benchmarks

- SPEC® CINT2006 [1)]
- Kernels
- LLVM Tests —benchmarking-only
- SPEC® CFP2006 (7 C/C++ benchmarks)

[1)] SPEC is a registered trademark of the Standard Performance Evaluation Cooperation http://spec.org

Setup

- Clang-600 (~LLVM 3.5) vs Clang-700 (~LLVM 3.7)
- -O3 -FLTO -PGO
- -O3 -FLTO
- ARM64

Some Performance Gains

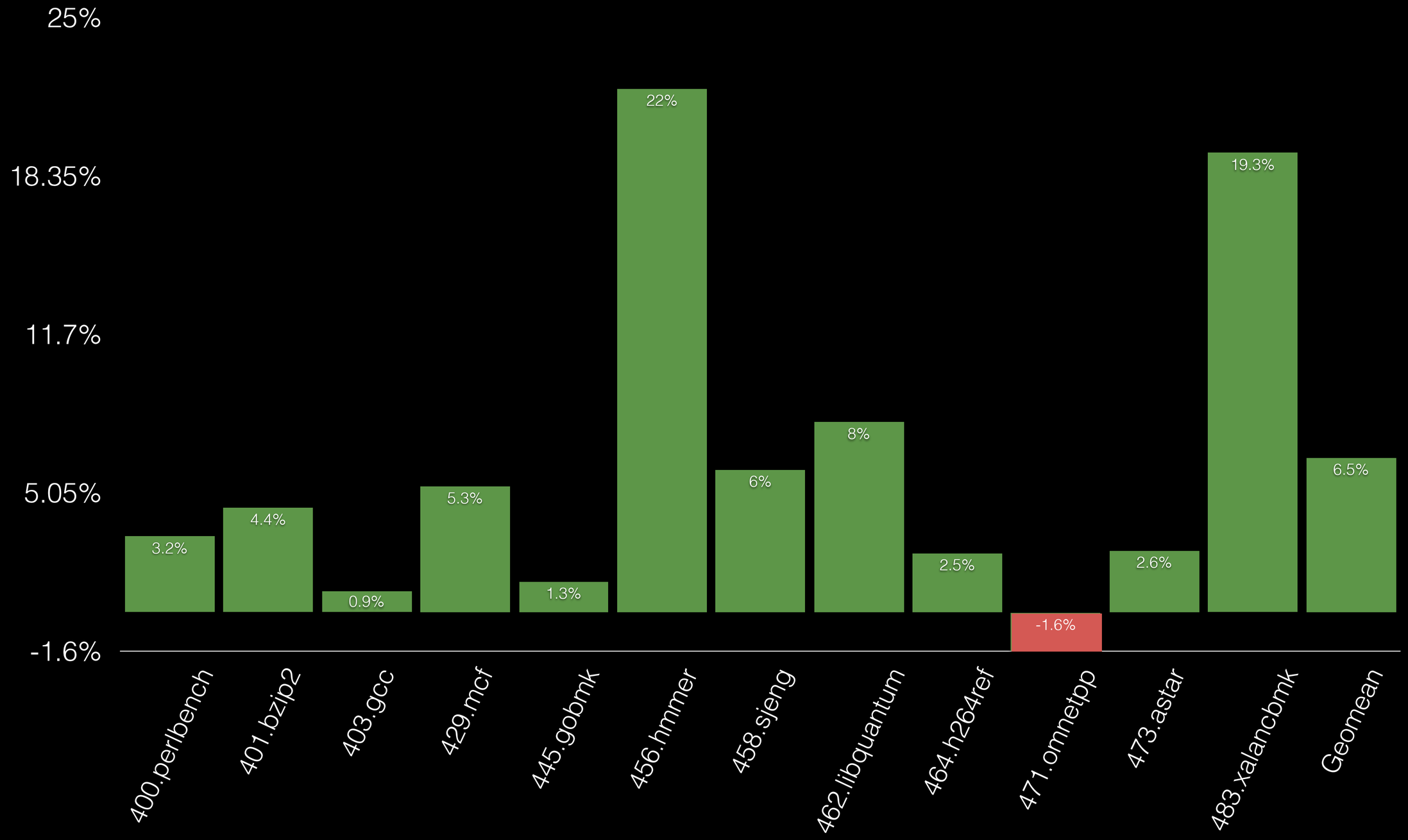
SPEC CINT2006: +6.5%

Kernels: up to 70%

Acknowledgements

- Adam Nemet, Arnold Schwaighofer, Chad Rossier, Chandler Carruth, James Molloy, Michael Zolotukhin, Tyler Nowicki, Yi Jiang and many other contributors of the LLVM community

SPEC CINT2006



Some Reasons For Gains

Condition folding: ((c) >= 'A' && (c) <= 'Z') || ((c) >= 'a' && (c) <= ‘z’)

cmp + br -> tbnz

Unrolling of loops with conditional stores

Register pressure aware loop rotation

Local (narrow) optimizations

Hot Loop: 456.hmmer

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k - 1] + t0[k - 1];
  …;


  d[k] = d[k - 1] + t1[k - 1];
  if ((sc = mc[k - 1] + t2[k - 1]) > d[k])
    d[k] = sc;
  …;
}
```

Hot Loop: 456.hmmer

Loop Distribution

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k - 1] + t0[k - 1];
  …;
}

for (k = 1; k <= M; k++) { // split
  d[k] = d[k - 1] + t1[k - 1];
  if ((sc = mc[k - 1] + t2[k - 1]) > d[k])
    d[k] = sc;
  …;
}
```

- Improves cache efficiency
- Partial vectorization

Hot Loop: 456.hmmer

Loop Distribution + Store To Load Forwarding

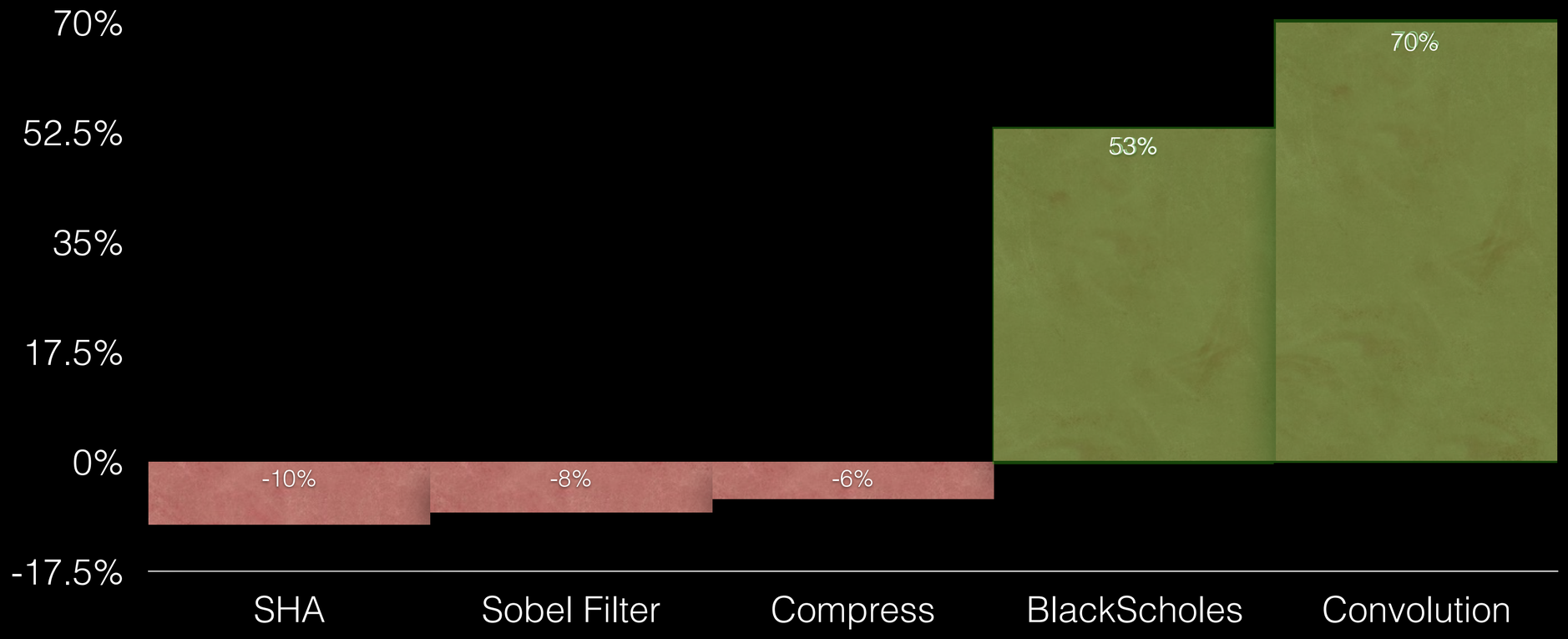
```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k - 1] + t0[k - 1];
  …;
}
T_{k-1} = d[0];
for (k = 1; k <= M; k++) { // split
  // d[k]= d[k-1] + t1[k-1]
  d[k] = T_k = T_{k-1} + t1[k - 1];
  if ((sc = mc[k - 1] + t2[k - 1]) > T_k)
    T_k = d[k] = sc;
  …
}
```

- Critical Path Shortening

Reflection

- Many local narrowly focused optimizations
- Loop Distribution advances capabilities of Loop Transformation Framework

Kernel Performance



“Convolution”: **Estimate Loop Unrolling Benefits**

No Unrolling

```
static const int k[] = { 0, 1, 5 };

for (v = 0; v < size; v++) {
  r += src[v] * k[v];
}
```

“Convolution”: **Estimate Loop Unrolling Benefits**

With Unrolling

```
static const int k[] = { 0, 1, 5 };

  …
  r += src[0] * k[0];
  r += src[1] * k[1];
  r += src[2] * k[2];
  …
```

“Convolution”: **Estimate Loop Unrolling Benefits**

With Unrolling

```
static const int k[] = { 0, 1, 5 };

  …
  r += src[0] * k[0];
  r += src[1] * k[1];
  r += src[2] * k[2];
  …
```

“Convolution”: **Estimate Loop Unrolling Benefits**

With Unrolling

```
static const int k[] = { 0, 1, 5 };

  …
  r += src[0] * 0;
  r += src[1] * 1;
  r += src[2] * 5;
  …
```

“Convolution”: **Estimate Loop Unrolling Benefits**

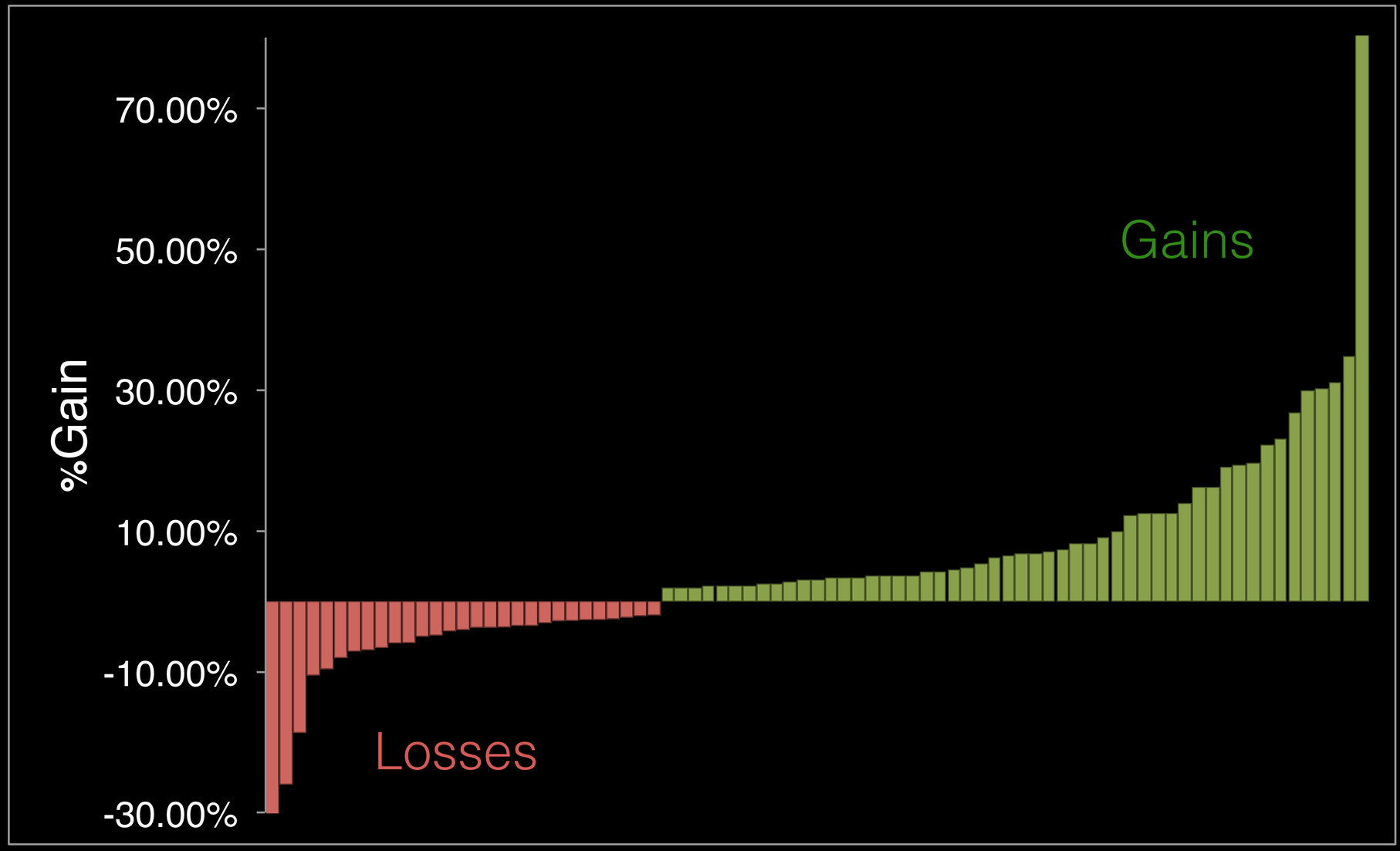
With Unrolling

```
static const int k[] = { 0, 1, 5 };

  …
  r += 0;                    saves mul + add
  r += src[1];               saves mul
  r += src[2] * 5;
  …
```

Kernel Regressions

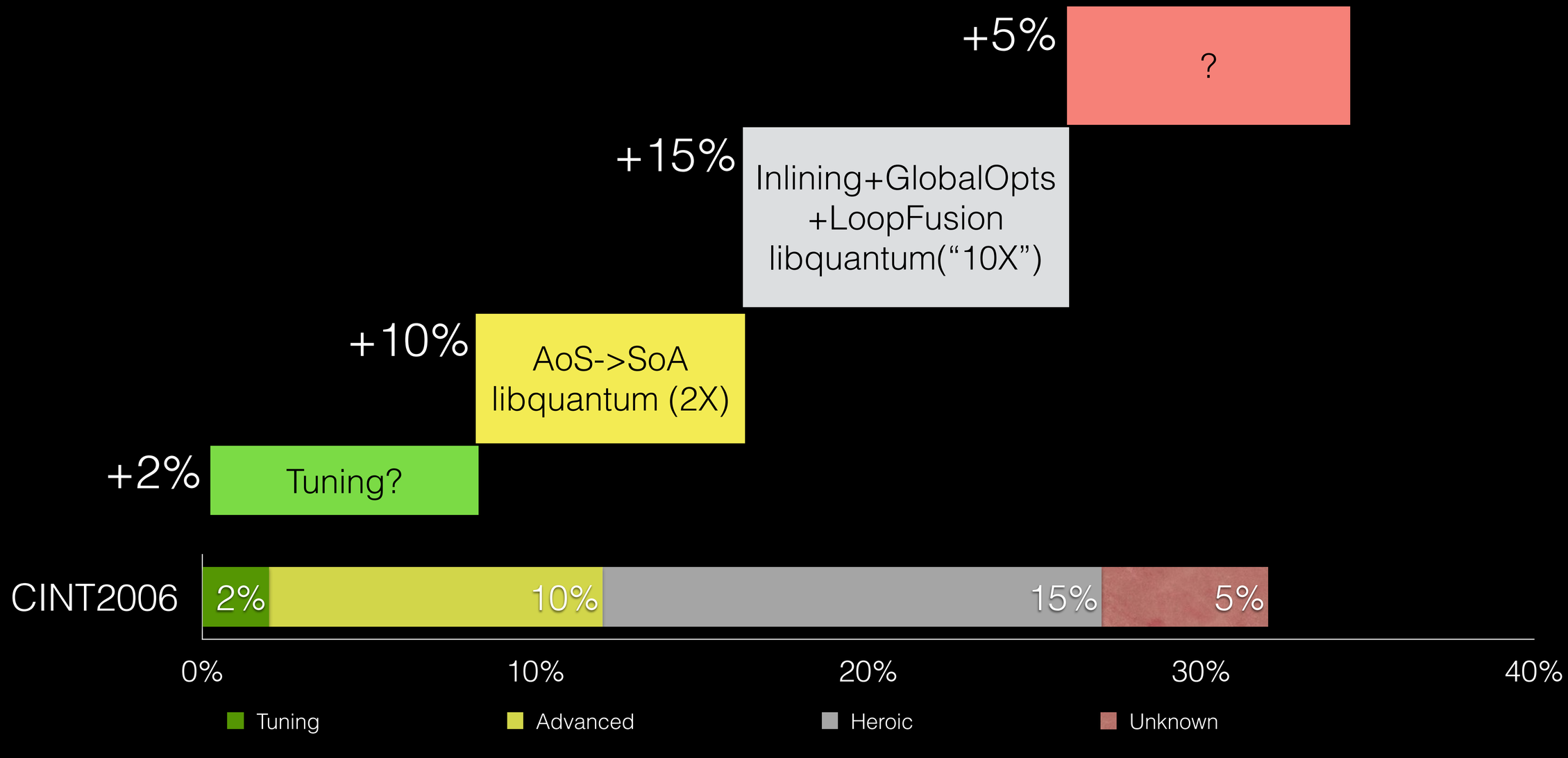
SHA (-10%)	Aggressive load hoisting	Tune Scheduler
Sobel Filter (-8%)	LSR expression normalization	Avoid GEP in base address calculation
Compress (-6%)	No CCMP optimization due to tbnz	Generalize CCMP

Performance Changes In LLVM Tests

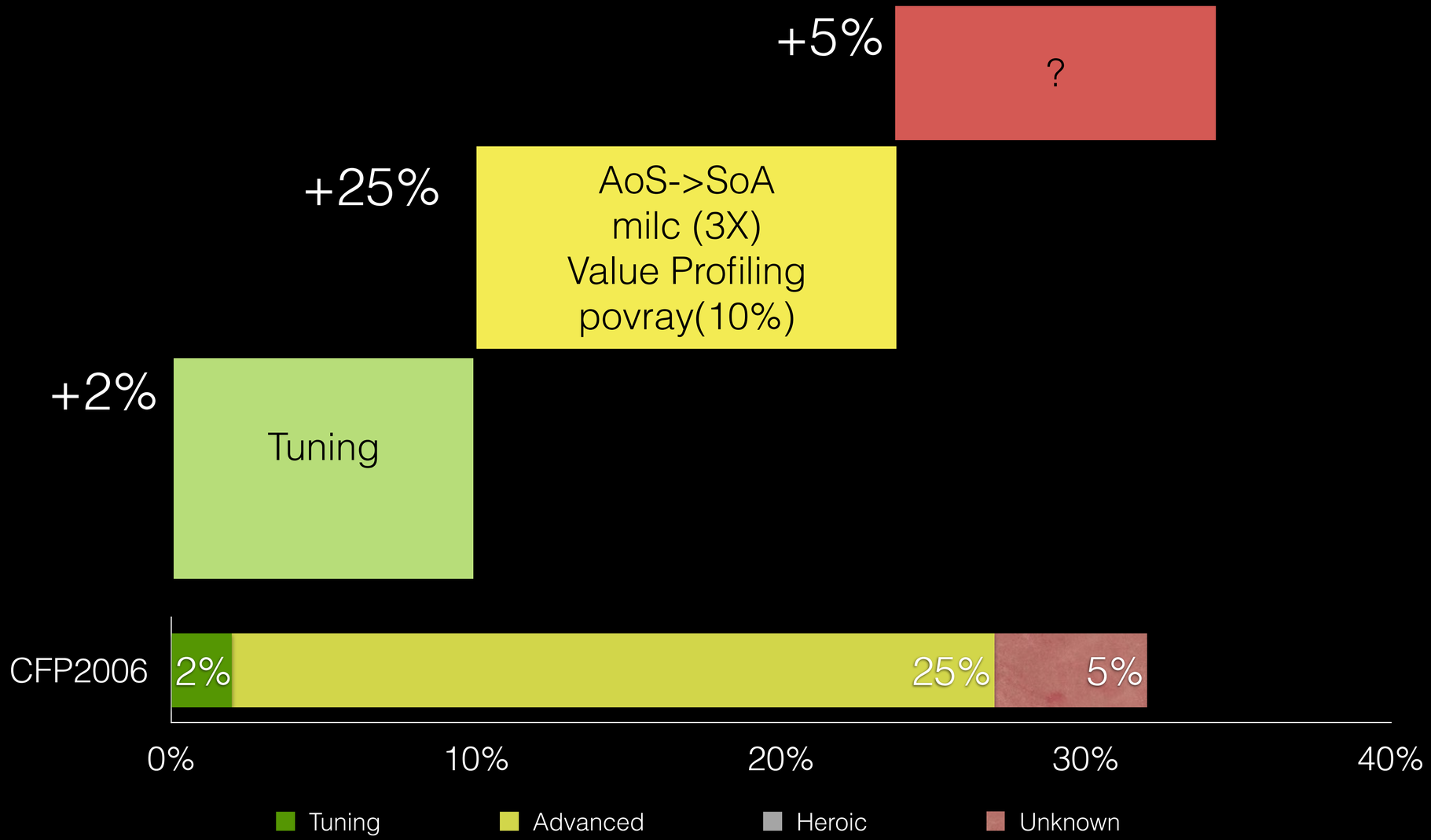


Headroom

CINT2006 Headroom



CFP2006 Headroom



Array of Structs (AoS) to Structs of Array (SoA)

Libquantum: AoS->SoA

```
hot_code(…, quantum_reg_node *reg) {
  int i;
  …
  for (i = 0; i < reg->size; i++) {
    if (reg->node[i].state & C) {
      reg->node[i].state ^= T;
    }
  }
  …
}
```

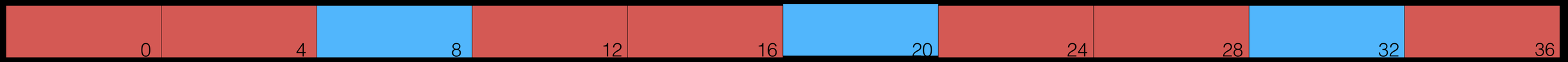
```
struct quantum_reg_node {
  COMPLEX_FLOAT amplitude;
  int state;
};
```

```
struct quantum_reg_node {
  int width;
  int size;
  int hashw;
  quantum_reg_node *node;
  int *hash;
};
```

Hot loop uses only some fields of a structure

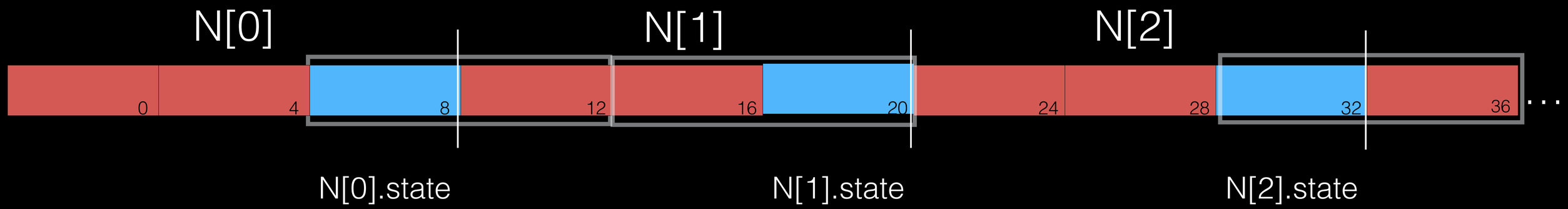
```
struct quantum_reg_node {
  COMPLEX_FLOAT amplitude;
  int state;
};
```

quantum_reg_node N[]

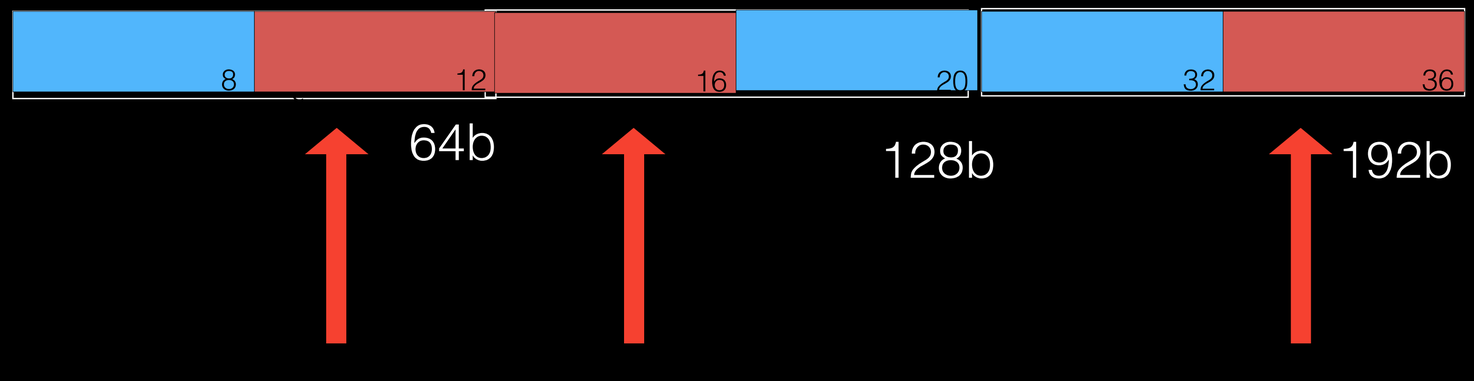



```
for(i=0; i<reg->size; i++){
  if(reg->N[i].state & C) {
    reg->N[i].state ^= T);
  }
}
```

quantum_reg_node N[]



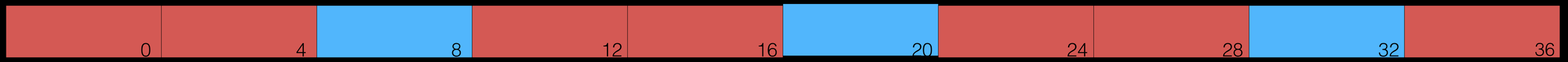
Cache:



fictitious cache line size!

```
struct quantum_reg_node {
  COMPLEX_FLOAT amplitude;
  int state;
};
```

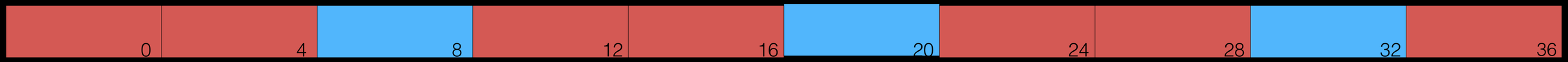
quantum_reg_node N[]




```
for(i=0; i<reg->size; i++){
  if(reg->N[i].state & C) {
    reg->N[i].state ^= T);
  }
}
```

```
COMPLEX_FLOAT *amplitude;
int *state;
```

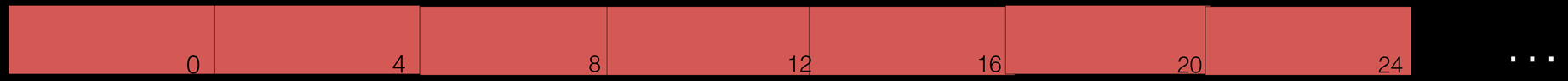
quantum_reg_node N[]




```
for(i=0; i<reg->size; i++){
  if(reg->N[i].state & C) {
    reg->N[i].state ^= T);
  }
}
```

```
COMPLEX_FLOAT *amplitude;
int *state;
```

COMLEX_FLOAT amplitude[]

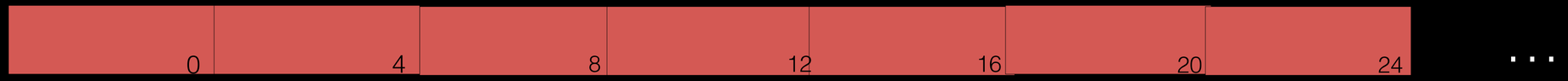


MAX_UNSIGNED state[]


```
for(i=0; i<reg->size; i++){
  if(reg->N[i].state & C) {
    reg->N[i].state ^= T);
  }
}
```

```
COMPLEX_FLOAT *amplitude;
int *state;
```

COMLEX_FLOAT amplitude[]

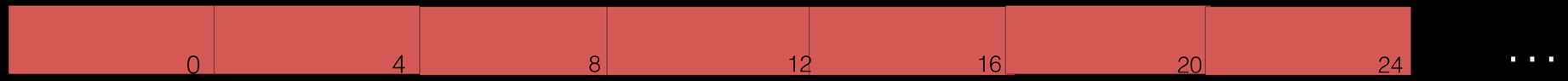


MAX_UNSIGNED state[]


```
for(i=0; i<reg->size; i++){
  if(reg->state[i] & C) {
    reg->state[i] ^= T);
  }
}
```

```
COMPLEX_FLOAT *amplitude;
int *state;
```

COMLEX_FLOAT amplitude[]



. . .

MAX_UNSIGNED state[]

. . .

Cache (after AoS):

64 128 192

AoS to SoA
speeds up benchmarks and applications
that are ***memory-bandwidth bound*** and/or
cache bound

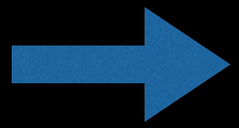
AoS->SoA: Challenges

- Legality
  - Casts to/from struct type, escaped types, address taken of individual fields, parameter and return values, semantic of constants
- Transformations
  - Data accesses, memory allocation
- Usability
  - Debugging

AoS->SoA: Changing Structure Definition and Accesses

```
struct quantum_reg_node {
  COMPLEX_FLOAT amplitude;
  MAX_UNSIGNED state;
};
```

```
struct quantum_reg_node {
  int width;
  int size;
  int hashw;
  quantum_reg_node *node;
  int *hash;
};
```

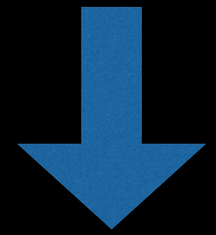


```
struct quantum_reg_node {
  int width;
  int size;
  int hashw;
  COMLEX_FLOAT *amplitude;
  int *state;
  int *hash;
};
```

reg->N[i].state to reg->state[i]

Constants

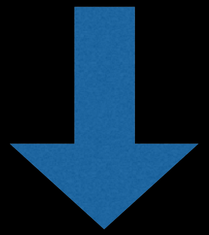
```
if (!reg.state) reg.node = calloc(r; eg.size, sizeof(quantum_reg_node))
```



```
%call10 = call i8* @calloc(i64 %conv, i64 16);
```

Parameter Passing

```
j = quantum_get_state(reg1->node)
```



```
j = quantum_get_state(reg1->amplitude, reg1->state);
```

References

- Implementing Data Layout Optimizations in LLVM Framework, Prashantha NR et al., 2014 LLVM Developer Meeting
- G. Chakrabarti, F. Chow, Structure Layout Optimizations in the Open64 Compiler: Design, Implementation and Measurements. Gautam Chakrabarti, Open64 Workshop at CGO, 2008
- O. Golovanevsky et al, https://www.research.ibm.com/haifa/Workshops/compiler2007/present/data-layout-optimizations-in-gcc.pdf, GCC Workshop, 2007
- R. Hundt et al, Practical Structure Layout Optimization and Advice, CGO, 2006

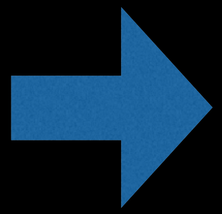
More Headroom: ‘Heroics’

Libquantum: Heroics

```
test_sum() {
  ...;
  cnot(2 * width - 1, width - 1, reg);
  sigma_x(2 * width - 1, reg);
  ...;
}
```

- 2 similar functions
- Hot loops

Whole program visibility
Alias analysis
GlobalModRef
…
Cost Model



Inline
+
Fuse

Interprocedural Loop Fusion

```
void cnot(int C, int T, q_reg *reg) {
  int i;
  int qec;
  status(&qec, NULL);
  if (qec)
    B1;
  else {
    …
    if (foo(x)) return;
    for (i = 0; i < reg->size; i++) {
      if (reg->state[i] & C)
        reg->state[i] ^= T;
    }
    decohere(reg);
  }
}
```

```
void sigma(int T, q_reg *reg) {
  int i;
  int qec;
  status(&qec, NULL);
  if (qec)
    B2;
  else {
    …
    if (foo(y)) return;
    for (i = 0; i < reg->size; i++) {

        reg->state[i] ^= T;
    }
    decohere(reg);
  }
}
```

Interprocedural Loop Fusion

```
void cnot(int C, int T, q_reg *reg) {
  int i;
  int qec;
  status(&qec, NULL);
  if (qec)
    B1;
  else {
    …
    if (foo(x)) return;
    for (i = 0; i < reg->size; i++) {
      if (reg->state[i] & C)
        reg->state[i] ^= T;
    }
    decohere(reg);
  }
}
```

```
void sigma(int T, q_reg *reg) {
  int i;
  int qec;
  status(&qec, NULL);
  if (qec)
    B2;
  else {
    …
    if (foo(y)) return;
    for (i = 0; i < reg->size; i++) {

        reg->state[i] ^= T;
    }
    decohere(reg);
  }
}
```

Interprocedural Loop Fusion

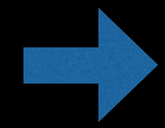
```
void

  status(&qec,

    B1;

    …

        reg->state[i] ^= T;
    }
    decohere(reg);

}
```

```
decohere(reg);
```

```
void

  status(&qec,

    B2

    …

        reg->state[i] ^= T;
    }
    decohere(reg);

}
```

Interprocedural Loop Fusion

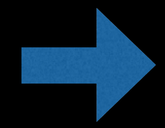
Global Value Prop



```
void decohere(q_reg) {
   if (status) {
     do_something;
   }
   return;
 }
```

Interprocedural Loop Fusion

Global Value Prop



```
void decohere(q_reg) {
   if (0) {
     do_something;
   }
   return;
 }
```

Interprocedural Loop Fusion

```
void cnot(int C, int T, q_reg *reg) {
  int i;
  int qec;
  status(&qec, NULL);
  if (qec)
    B1;
  else {
    …
    if (foo(x)) return;
    for (i = 0; i < reg->size; i++) {
      if (reg->state[i] & C)
        reg->state[i] ^= T;
    }
   decohere(reg);
  }
}
```

```
void sigma(int T, q_reg *reg) {
  int i;
  int qec;
  status(&qec, NULL);
  if (qec)
    B2;
  else {
    …
    if (foo(y)) return;
    for (i = 0; i < reg->size; i++) {

        reg->state[i] ^= T;
    }
    decohere(reg);
  }
}
```

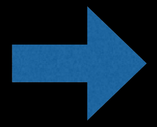
Interprocedural Loop Fusion

```
void

  status(&qec, NULL);
  if (qec)
    B1;

    …

        reg->state[i] ^= T;
    }
  ____________________

}
```

```
void

  status(&qec, NULL);
  if (qec)
    B2;

    …

        reg->state[i] ^= T;
    }
  ____________________

}
```

Interprocedural Loop Fusion

Inlining



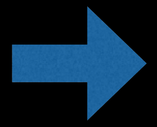
```
status(&qec, NULL);
if (qec)
```

```
void
status(qec)
  B1;
  …
      reg->state[i] ^= T;
  }
}
```

```
void
status(&qec,
  B2;
  …
      reg->state[i] ^= T;
  }
}
```

Interprocedural Loop Fusion

Inlining



```
if (globalVar)
```

```
void

  status(&qec,

    B1;

    …

        reg->state[i] ^= T;
    }

}
```

```
void

  status(&qec,

    B2;

    …

        reg->state[i] ^= T;
    }

}
```

Interprocedural Loop Fusion

```
void cnot(int C, int T, q_reg *reg) {
  int i;
  int qec;
  status(&qec, NULL);
  if (globalVar)
    B1;
  else {
    …
    if (foo(x)) return;
    for (i = 0; i < reg->size; i++) {
      if (reg->state[i] & C)
        reg->state[i] ^= T;
    }
    decohere(reg);
  }
}
```

```
void sigma(int T, q_reg *reg) {
  int i;
  int qec;
  status(&qec, NULL);
  if (globalVar)
    B2;
  else {
    …
    if (foo(y)) return;
    for (i = 0; i < reg->size; i++) {

        reg->state[i] ^= T;
    }
    decohere(reg);
  }
}
```

Interprocedural Loop Fusion

GlobalModRef

```
void
_____

 status(&qec, NULL);
 if (globalVar)
   B1;
 else {
  …

      reg->state[i] ^= T;
  }
 ____________

}
```

```
void
_____

 status(&qec, NULL);
 if (globalVar)
   B2;
 else {
   …

      reg->state[i] ^= T;
   }
  ____________

}
```

Interprocedural Loop Fusion

GlobalModRef

```
void

status(&qec,

  B1;

  …

      reg->state[i] ^= T;
  }

}
```

```
if (globalVar) {
  B1; B2;
} else {
```

```
void

…
if (foo(x)) return;
for (i = 0; i < reg->size; i++) {
  if (reg->state[i] & C)
    reg->state[i] ^= T;
}
…
if (foo(y)) return;
for (i = 0; i < reg->size; i++) {
    reg->state[i] ^= T;
}
  }
```

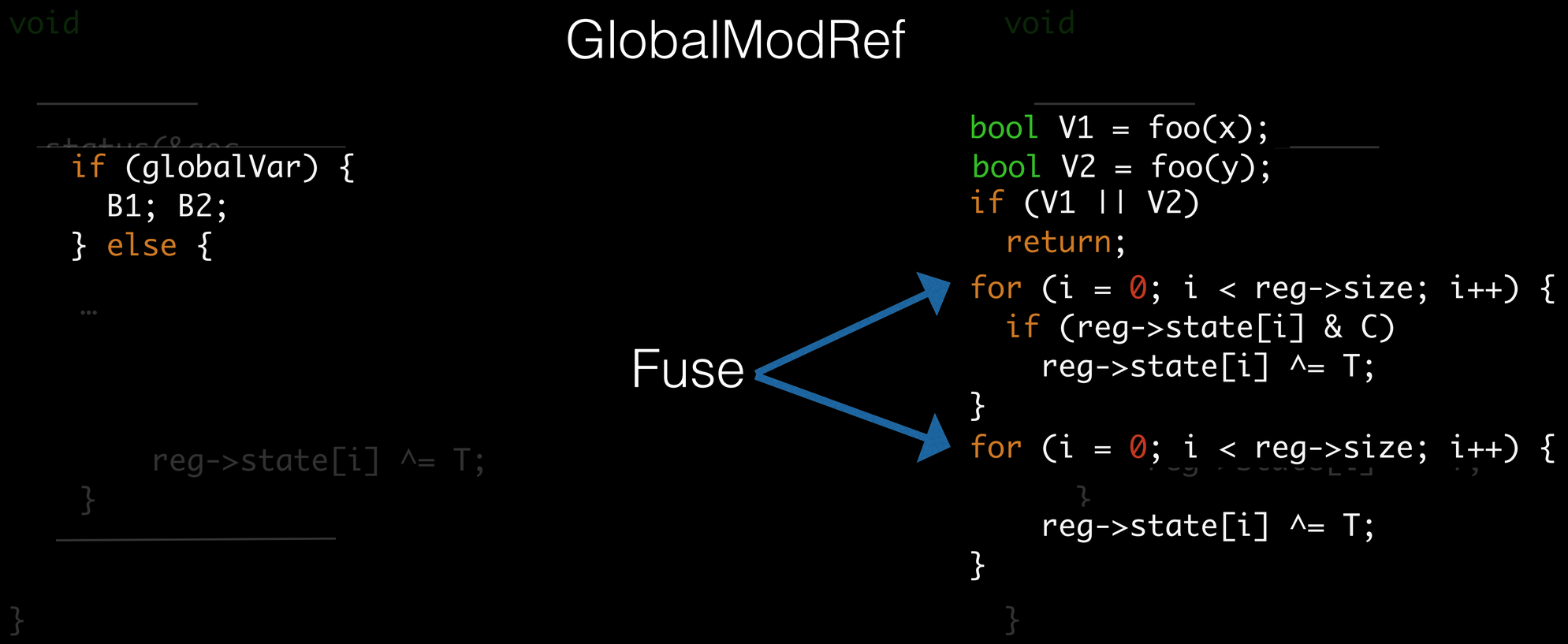
Interprocedural Loop Fusion

GlobalModRef

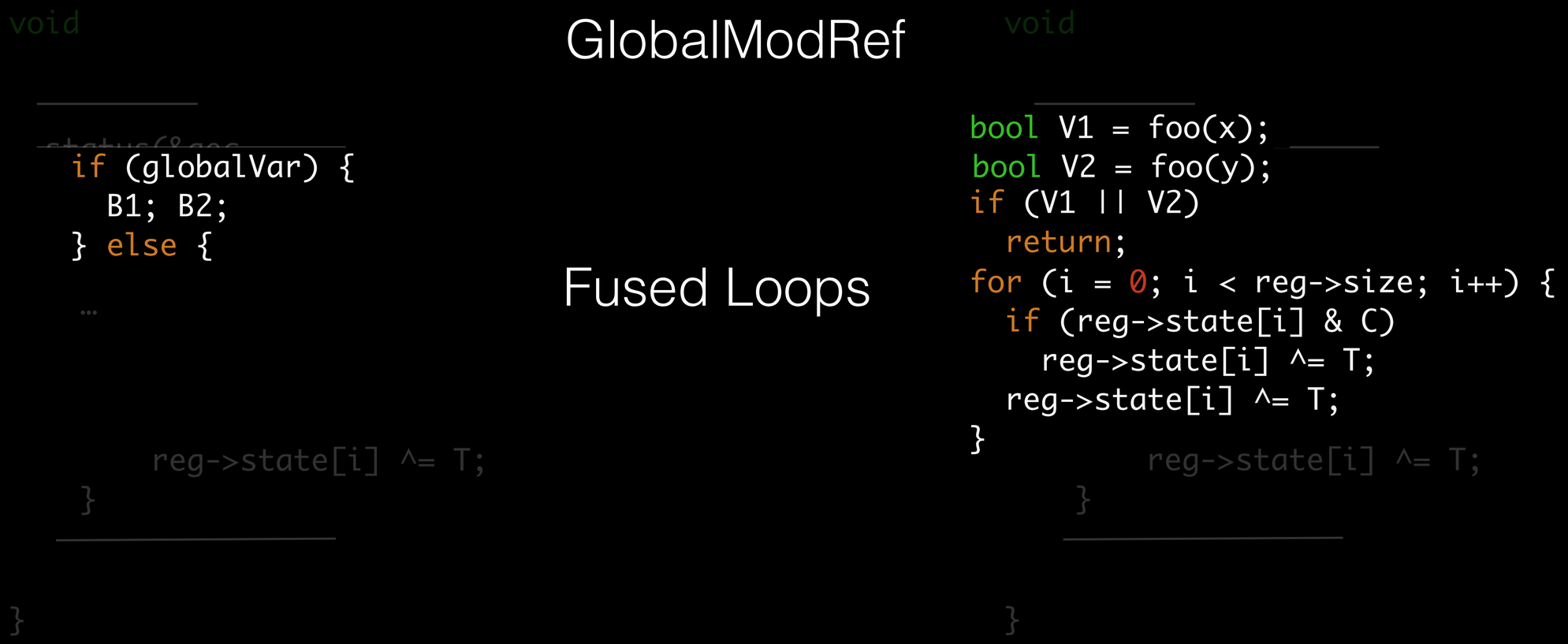
Interprocedural Loop Fusion

GlobalModRef

Interprocedural Loop Fusion



Interprocedural Loop Fusion



What can we learn?

- Optimization Scope: Call Chain
- Concept: Function Similarity
- Challenge: Hoist statements across loops
- Techniques: Global Value Prop, Partial Inlining, GlobalModRef, Loop Fusion, …

Take Aways

- Techniques needed for ‘heroics’ can be generalized to advance optimization technology
- Cost Model?

How to expose performance opportunities to developers?

__builtin_nontemporal_store

```
void scaledCpy(float *__restrict__ a, float *__restrict__ b, float S, int N) {
  for (int i = 0; i < N; i++)
    b[i] = S * a[i];
}
```

__builtin_nontemporal_store

```
void scaledCpy(float *__restrict__ a, float *__restrict__ b, float S, int N) {
  for (int i = 0; i < N; i++)
   //  b[i] = S * a[i];
    __builtin_nontemporal_store(S * a[i], &b[i]);
}
```

Vectorizer: Hints and Diagnostics

```
while (good) {

  for (i = 0; i < N; i++) {
    DW[i] = A[i - 3] + B[i - 2] + C[i - 1] + D[i];
    UW[i] = A[i] + B[i + 1] + C[i + 2] + D[i + 3];
  }
}
```

```
remark: loop not vectorized: … Avoid runtime pointer checking when you know
the arrays will always be independent by specifying '#pragma clang loop
vectorize(assume_safety)' before the loop or by specifying 'restrict' on
the array arguments. Erroneous results will occur if these options are
incorrectly applied!
```

Conclusions

The best days for LLVM performance are ahead of us

Questions?