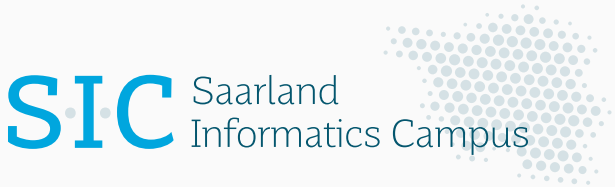
Polyhedral Driven Optimizations on Real Codes

LLVM Performance Workshop – February 4th, 2017 – Austin, TX

Johannes Doerfert and Sebastian Hack

Compiler Design Lab
Saarland University
http://compilers.cs.uni-saarland.de



Agenda

- Optimizing SPEC with Polly
  - 456.hmmer
  - 462.libquantum
  - 470.lbm
  - Conclusion
- Polyhedral Value Analysis
  - Design Goals and Motivation
  - Comparison with ScalarEvolution
  - Use Cases

Optimizing SPEC with Polly

SPEC 2006 — 456.hmmer — fast_algorithms.c

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k – 1] + tpmm[k – 1];
  if ((sc = ip[k – 1] + tpim[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k – 1] + tpdm[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < –INFTY) mc[k] = –INFTY;


  dc[k] = dc[k – 1] + tpdd[k – 1];
  if ((sc = mc[k – 1] + tpmd[k – 1]) > dc[k]) dc[k] = sc;
  if (dc[k] < –INFTY) dc[k] = –INFTY;



  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < –INFTY) ic[k] = –INFTY;
}}
```

SPEC 2006 — 456.HMMER — FAST_ALGORITHMS.C

```
#pragma clang loop vectorize(enable)
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k – 1] + tpmm[k – 1];
  if ((sc = ip[k – 1] + tpim[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k – 1] + tpdm[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < –INFTY) mc[k] = –INFTY;
}
for (k = 1; k <= M; k++) {
  dc[k] = dc[k – 1] + tpdd[k – 1];
  if ((sc = mc[k – 1] + tpmd[k – 1]) > dc[k]) dc[k] = sc;
  if (dc[k] < –INFTY) dc[k] = –INFTY;



  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < –INFTY) ic[k] = –INFTY;
}}
```

SPEC 2006 — 456.HMMER — FAST_ALGORITHMS.C

```
#pragma clang loop vectorize(enable)
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k – 1] + tpmm[k – 1];
  if ((sc = ip[k – 1] + tpim[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k – 1] + tpdm[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < –INFTY) mc[k] = –INFTY;
}
for (k = 1; k <=
  dc[k] = dc[k –
  if ((sc = mc[k
  if (dc[k] < –INFTY) dc[k] = –INFTY;



  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < –INFTY) ic[k] = –INFTY;
}}
```

up to 30% speedup

SPEC 2006 — 456.hmmer — fast_algorithms.c

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k – 1] + tpmm[k – 1];
  if ((sc = ip[k – 1] + tpim[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k – 1] + tpdm[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < –INFTY) mc[k] = –INFTY;



  dc[k] = dc[k – 1] + tpdd[k – 1];
  if ((sc = mc[k – 1] + tpmd[k – 1]) > dc[k]) dc[k] = sc;
  if (dc[k] < –INFTY) dc[k] = –INFTY;
}
#pragma clang loop vectorize(enable)
for (k = 1; k <= M; k++) {
  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < –INFTY) ic[k] = –INFTY;
}}
```

SPEC 2006 — 456.hmmer — fast_algorithms.c

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k – 1] + tpmm[k – 1];
  if ((sc = ip[k – 1] + tpim[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k – 1] + tpdm[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < –INFTY) mc[k] = –INFTY;

  dc[k] = dc[k –
  if ((sc = mc[k
  if (dc[k] < –INFTY) dc[k] = –INFTY;
}
#pragma clang loop vectorize(enable)
for (k = 1; k <= M; k++) {
  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < –INFTY) ic[k] = –INFTY;
}}
```

up to 30% speedup

SPEC 2006 — 456.hmmer — fast_algorithms.c

```
#pragma clang loop vectorize(enable)
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k – 1] + tpmm[k – 1];
  if ((sc = ip[k – 1] + tpim[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k – 1] + tpdm[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < –INFTY) mc[k] = –INFTY;
}
for (k = 1; k <= M; k++) {
  dc[k] = dc[k – 1] + tpdd[k – 1];
  if ((sc = mc[k – 1] + tpmd[k – 1]) > dc[k]) dc[k] = sc;
  if (dc[k] < –INFTY) dc[k] = –INFTY;
}
#pragma clang loop vectorize(enable)
for (k = 1; k <= M; k++) {
  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < –INFTY) ic[k] = –INFTY;
}}
```

SPEC 2006 — 456.HMMER — FAST_ALGORITHMS.C

```
#pragma clang loop vectorize(enable)
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k – 1] + tpmm[k – 1];
  if ((sc = ip[k – 1] + tpim[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k – 1] + tpdm[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < –INFTY) mc[k] = –INFTY;
}
for (k = 1; k <=
  dc[k] = dc[k –
  if ((sc = mc[k
  if (dc[k] < –INFTY) dc[k] = –INFTY;
}
#pragma clang loop vectorize(enable)
for (k = 1; k <= M; k++) {
  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < –INFTY) ic[k] = –INFTY;
}}
```

up to 50% speedup

SPEC 2006 — 456.hmmer — fast_algorithms.c

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k – 1] + tpmm[k – 1];
  if ((sc = ip[k – 1] + tpim[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k – 1] + tpdm[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < –INFTY) mc[k] = –INFTY;


  dc[k] = dc[k – 1] + tpdd[k – 1];
  if ((sc = mc[k – 1] + tpmd[k – 1]) > dc[k]) dc[k] = sc;
  if (dc[k] < –INFTY) dc[k] = –INFTY;



  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < –INFTY) ic[k] = –INFTY;
}}
```

SPEC 2006 — 456.HMMER — FAST_ALGORITHMS.C

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k – 1] + tpmm[k – 1];
  if ((sc = ip[k – 1] + tpim[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k – 1] + tpdm[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < –INFTY) mc[k] = –INFTY;


  dc[k] = dc[k – 1] + tpdd[k – 1];
  if ((sc = mc[k – 1] + tpmd[k – 1]) > dc[k]) dc[k] = sc;
  if (dc[k] < –INFTY) dc[k] = –INFTY;



  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < –INFTY) ic[k] = –INFTY;
}}
```

SPEC 2006 — 456.HMMER — FAST_ALGORITHMS.C

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k – 1] + tpmm[k – 1];
  if ((sc = ip[k – 1] + tpim[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k – 1] + tpdm[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < –INFTY) mc[k] = –INFTY;

  dc[k] = dc[k –
  if ((sc = mc[k                                    sc;
  if (dc[k] < –INFTY) dc[k] = –INFTY;

  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < –INFTY) ic[k] = –INFTY;
}}
```

non-affine conditionals
can be approximated

SPEC 2006 — 456.HMMER — FAST_ALGORITHMS.C

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k - 1] + tpmm[k - 1];
  if ((sc = ip[k - 1] + tpim[k - 1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k - 1] + tpdm[k - 1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < -INFTY) mc[k] = -INFTY;

  dc[k] = dc[k -
  if ((sc = mc[k                                    sc;
  if (dc[k] < -INFTY) dc[k] = -INFTY;

  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < -INFTY) ic[k] = -INFTY;
}}
```

conditionals can be lowered to selects

SPEC 2006 — 456.HMMER — FAST_ALGORITHMS.C

```
for (k = 1; k <= M; k++) {
  mc[k] = mpp[k – 1] + tpmm[k – 1];
  if ((sc = ip[k – 1] + tpim[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = dpp[k – 1] + tpdm[k – 1]) > mc[k]) mc[k] = sc;
  if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
  mc[k] += ms[k];
  if (mc[k] < –INFTY) mc[k] = –INFTY;



  dc[k] = dc[k – 1] + tpdd[k – 1];
  if ((sc = mc[k – 1] + tpmd[k – 1]) > dc[k]) dc[k] = sc;
  if (dc[k] < –INFTY) dc[k] = –INFTY;



  if (k < M) {
    ic[k] = mpp[k] + tpmi[k];
    if ((sc = ip[k] + tpii[k]) > ic[k]) ic[k] = sc;
    ic[k] += is[k];
    if (ic[k] < –INFTY) ic[k] = –INFTY;
}}
```

Polyhedral Statement Granularity

Pluto *like* | Polly | Polly^{+}

```
mc[k] = mpp[k − 1] + tpmm[k − 1];
if ((sc = ip[k − 1] + tpim[k − 1]) > mc[k]) mc[k] = sc;
if ((sc = dpp[k − 1] + tpdm[k − 1]) > mc[k]) mc[k] = sc;
if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
mc[k] += ms[k];
if (mc[k] < −INFTY) mc[k] = −INFTY;
```

```
dc[k] = dc[k − 1] + tpdd[k − 1];
if ((sc = mc[k − 1] + tpmd[k − 1]) > dc[k]) dc[k] = sc;
if (dc[k] < −INFTY) dc[k] = −INFTY;
```

Polyhedral Statement Granularity

Pluto *like* | Polly | Polly^{+}

memory write (9)

```
mc[k] = mpp[k − 1] + tpmm[k − 1];
if ((sc = ip[k − 1] + tpim[k − 1]) > mc[k]) mc[k] = sc;
if ((sc = dpp[k − 1] + tpdm[k − 1]) > mc[k]) mc[k] = sc;
if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
mc[k] += ms[k];
if (mc[k] < −INFTY) mc[k] = −INFTY;
```

```
dc[k] = dc[k − 1] + tpdd[k − 1];
if ((sc = mc[k − 1] + tpmd[k − 1]) > dc[k]) dc[k] = sc;
if (dc[k] < −INFTY) dc[k] = −INFTY;
```

Polyhedral Statement Granularity

Pluto *like*	Polly	Polly$^+$
memory write (9)	basic blocks (1)	

```
mc[k] = mpp[k − 1] + tpmm[k − 1];
if ((sc = ip[k − 1] + tpim[k − 1]) > mc[k]) mc[k] = sc;
if ((sc = dpp[k − 1] + tpdm[k − 1]) > mc[k]) mc[k] = sc;
if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
mc[k] += ms[k];
if (mc[k] < −INFTY) mc[k] = −INFTY;
```

```
dc[k] = dc[k − 1] + tpdd[k − 1];
if ((sc = mc[k − 1] + tpmd[k − 1]) > dc[k]) dc[k] = sc;
if (dc[k] < −INFTY) dc[k] = −INFTY;
```

Polyhedral Statement Granularity

Pluto *like*	Polly	Polly$^+$
memory write (9)	basic blocks (1)	

```
mc[k] = mpp[k − 1] + tpmm[k − 1];
if ((sc = ip[k − 1] + tpim[k − 1]) > mc[k]) mc[k] = sc;
if ((sc = dpp[k − 1] + tpdm[k − 1]) > mc[k]) mc[k] = sc;
if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
mc[k] += ms[k];
if (mc[k] < −INFTY) mc[k] = −INFTY;
```

```
dc[k] = dc[k − 1] + tpdd[k − 1];
if ((sc = mc[k − 1] + tpmd[k − 1]) > dc[k]) dc[k] = sc;
if (dc[k] < −INFTY) dc[k] = −INFTY;
```

Polyhedral Statement Granularity

Pluto *like*	Polly	Polly$^+$
memory write (9)	basic blocks (1)	semantic blocks (2)

```
mc[k] = mpp[k − 1] + tpmm[k − 1];
if ((sc = ip[k − 1] + tpim[k − 1]) > mc[k]) mc[k] = sc;
if ((sc = dpp[k − 1] + tpdm[k − 1]) > mc[k]) mc[k] = sc;
if ((sc = xmb + bp[k]) > mc[k]) mc[k] = sc;
mc[k] += ms[k];
if (mc[k] < −INFTY) mc[k] = −INFTY;

// semantic block ends, split basic block here

dc[k] = dc[k − 1] + tpdd[k − 1];
if ((sc = mc[k − 1] + tpmd[k − 1]) > dc[k]) dc[k] = sc;
if (dc[k] < −INFTY) dc[k] = −INFTY;
```

Polyhedral Statement Granularity

Pluto *like*	Polly	Polly$^+$
memory write (9)	basic blocks (1)	semantic blocks (2)

```
mc[k] = m
if ((sc =
if ((sc =
if ((sc =
mc[k] +=
if (mc[k] < -INFTY) mc[k] = -INFTY;

// semantic block ends, split basic block here

dc[k] = dc[k - 1] + tpdd[k - 1];
if ((sc = mc[k - 1] + tpmd[k - 1]) > dc[k]) dc[k] = sc;
if (dc[k] < -INFTY) dc[k] = -INFTY;
```

30% speedup out-of-the-box,
up to 50% possible

SPEC2006 — 462.libquantum — oaddn.c

SPEC2006 — 462.LIBQUANTUM — OADDN.C

```
void test_sum(int compare, int width, quantum_reg *reg) {

    quantum_sigma_x(2 * width - 1, reg);
    quantum_cnot(2 * width - 1, width - 1, reg);
```

SPEC2006 — 462.LIBQUANTUM — OADDN.C

```
void test_sum(int compare, int width, quantum_reg *reg) {
  if (compare & ((MAX_UNSIGNED)1 << (width - 1))) {
    quantum_cnot(2 * width - 1, width - 1, reg);
    quantum_sigma_x(2 * width - 1, reg);
    quantum_cnot(2 * width - 1, 0, reg);
  } else {
    quantum_sigma_x(2 * width - 1, reg);
    quantum_cnot(2 * width - 1, width - 1, reg);
  }
  for (i = (width - 2); i > 0; i--) {
    if (compare & (1 << i)) {
      quantum_toffoli(i + 1, width + i, i, reg);
      quantum_sigma_x(width + i, reg);
      quantum_toffoli(i + 1, width + i, 0, reg);
    } else {
      quantum_sigma_x(width + i, reg);
      quantum_toffoli(i + 1, width + i, i, reg);
  } }
  if (compare & 1) {
    quantum_sigma_x(width, reg);
    quantum_toffoli(width, 1, 0, reg);
  }
```

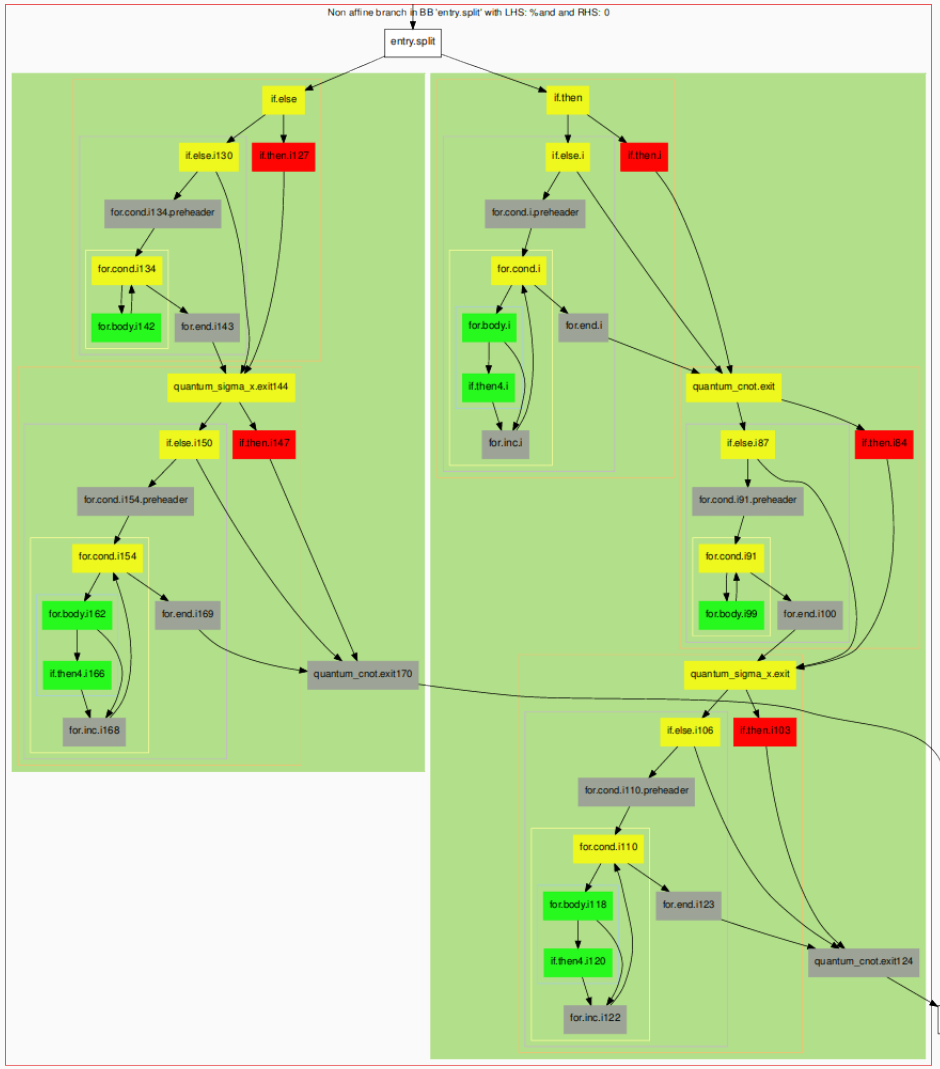
SPEC2006 — 462.LIBQUANTUM — OADDN.C

```
  quantum_toffoli(2 * width + 1, 0, 2 * width, reg);
  if (compare & 1) {
    quantum_toffoli(width, 1, 0, reg);
    quantum_sigma_x(width, reg);
  }
  for (i = 1; i <= (width - 2); i++) {
    if (compare & (1 << i)) {
      quantum_toffoli(i + 1, width + i, 0, reg);
      quantum_sigma_x(width + i, reg);
      quantum_toffoli(i + 1, width + i, i, reg);
    } else {
      quantum_toffoli(i + 1, width + i, i, reg);
      quantum_sigma_x(width + i, reg);
  } }
  if (compare & (1 << (width - 1))) {
    quantum_cnot(2 * width - 1, 0, reg);
    quantum_sigma_x(2 * width - 1, reg);
    quantum_cnot(2 * width - 1, width - 1, reg);
  } else {
    quantum_cnot(2 * width - 1, width - 1, reg);
    quantum_sigma_x(2 * width - 1, reg);
} }
```

SPEC2006 — 462.LIBQUANTUM — OADDN.C

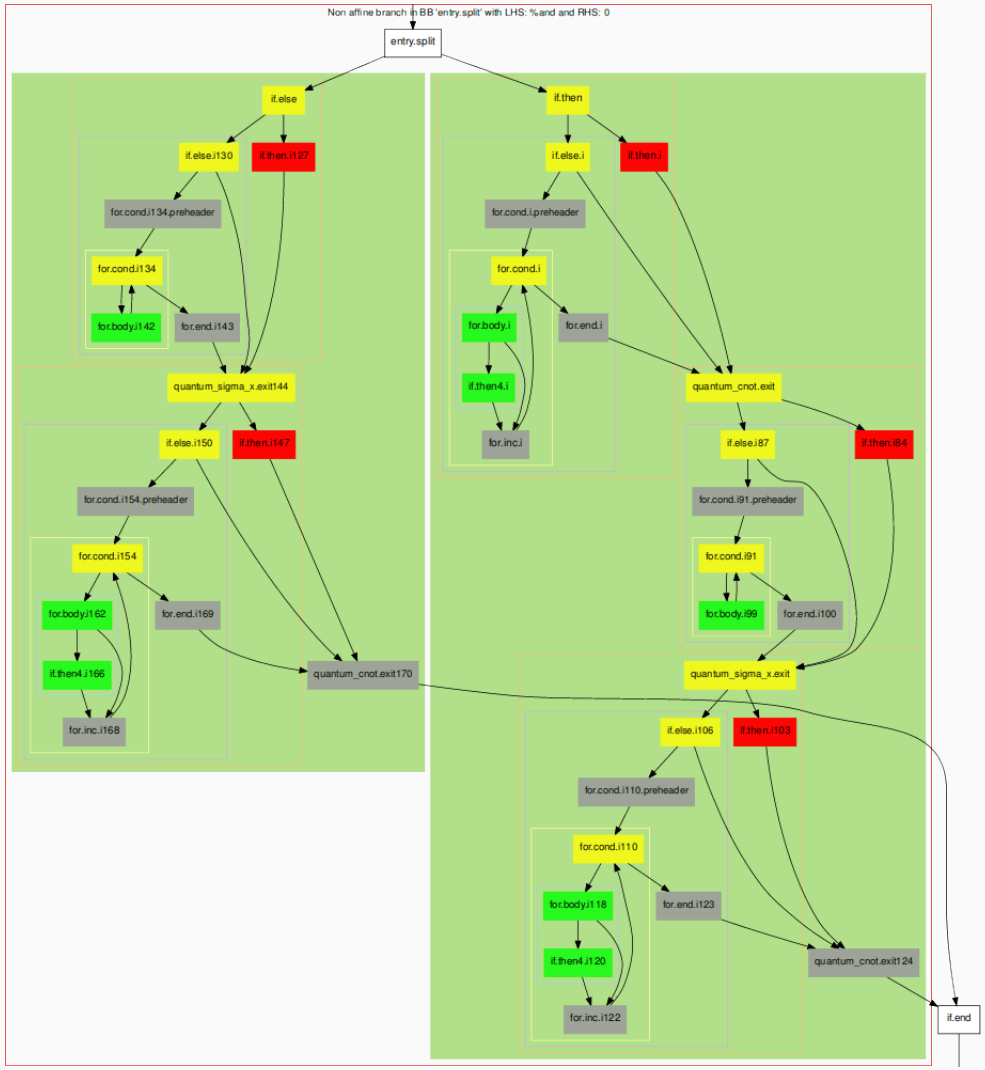
```
if (!(compare&(1<<(width-1)))) {
  quantum_sigma_x(2*width-1, reg);
  quantum_cnot(2*width-1,width-1,reg)
} else {
  quantum_cnot(2*width-1,width-1,reg)
  quantum_sigma_x(2*width-1, reg);
  quantum_cnot(2*width-1, 0, reg);
}
```

SPEC2006 — 462.LIBQUANTUM — OADDN.C




```
if (!(compare&(1<<(width-1)))) {
  quantum_sigma_x(2*width-1, reg);
  quantum_cnot(2*width-1,width-1,reg)
} else {
  quantum_cnot(2*width-1,width-1,reg)
  quantum_sigma_x(2*width-1, reg);
  quantum_cnot(2*width-1, 0, reg);
}
```

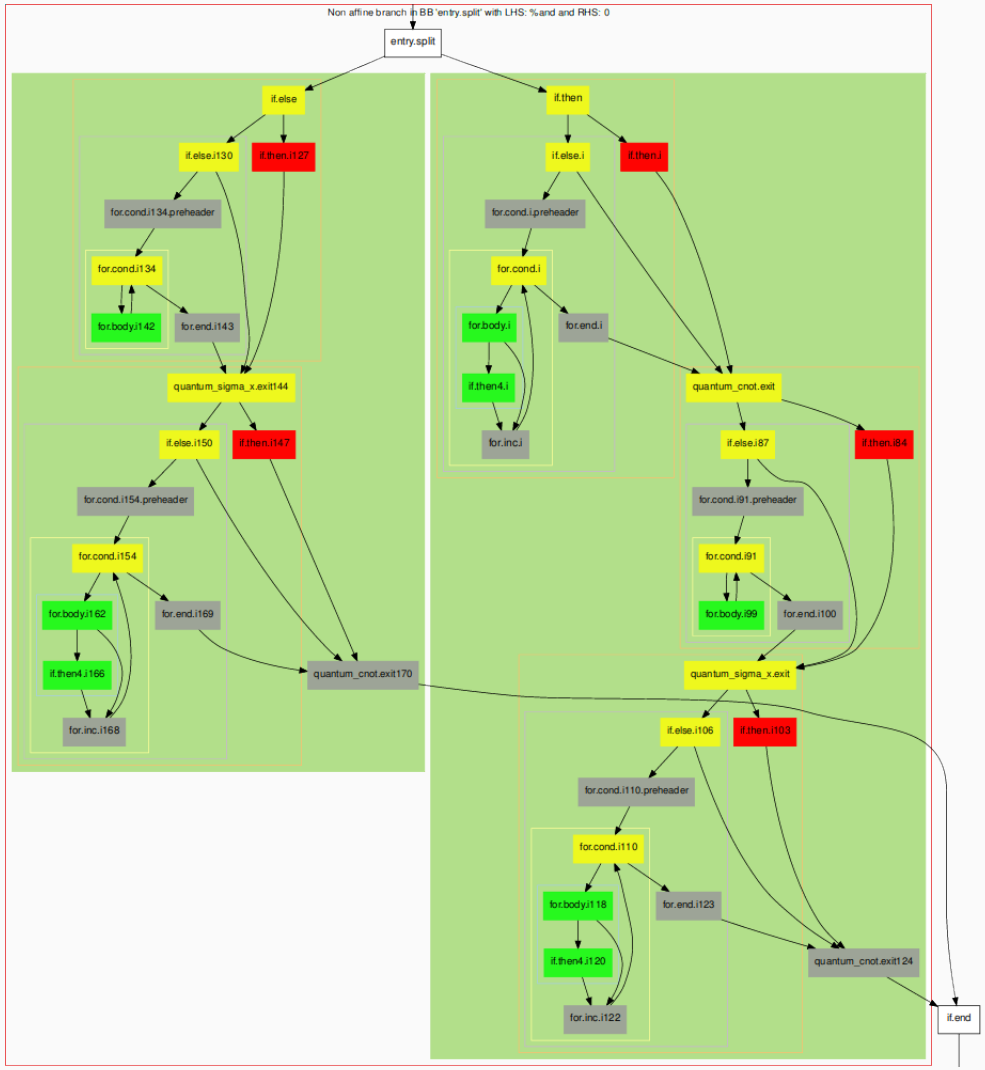
SPEC2006 — 462.LIBQUANTUM — OADDN.C




```
if (!(compare&(1<<(width-1)))) {
  quantum_sigma_x(2*width-1, reg);
  quantum_cnot(2*width-1,width-1,reg)
} else {
  quantum_cnot(2*width-1,width-1,reg)
  quantum_sigma_x(2*width-1, reg);
  quantum_cnot(2*width-1, 0, reg);
}
```

control conditions

SPEC2006 — 462.LIBQUANTUM — OADDN.C

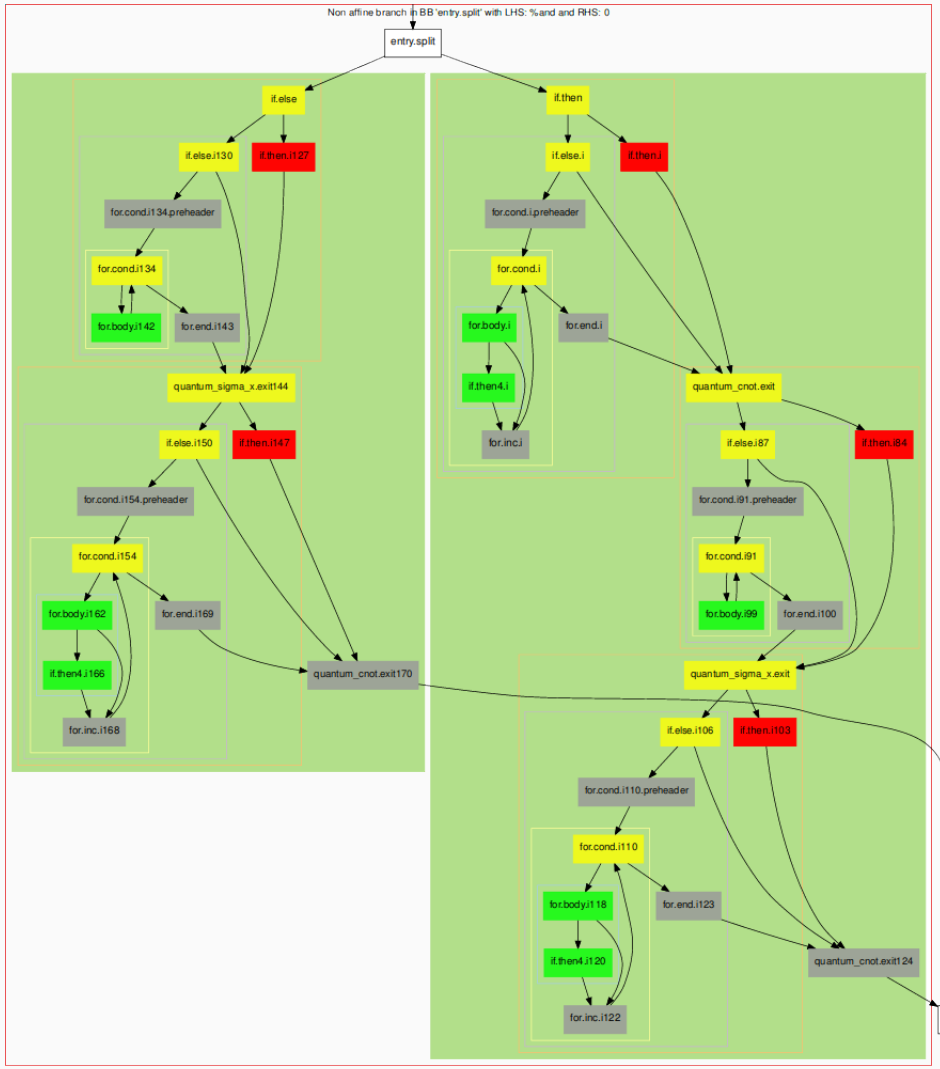



```
if (!(compare&(1<<(width-1)))) {
  quantum_sigma_x(2*width-1, reg);
  quantum_cnot(2*width-1,width-1,reg)
} else {
  quantum_cnot(2*width-1,width-1,reg)
  quantum_sigma_x(2*width-1, reg);
  quantum_cnot(2*width-1, 0, reg);
}
```

control conditions

trivial block

SPEC2006 — 462.LIBQUANTUM — OADDN.C



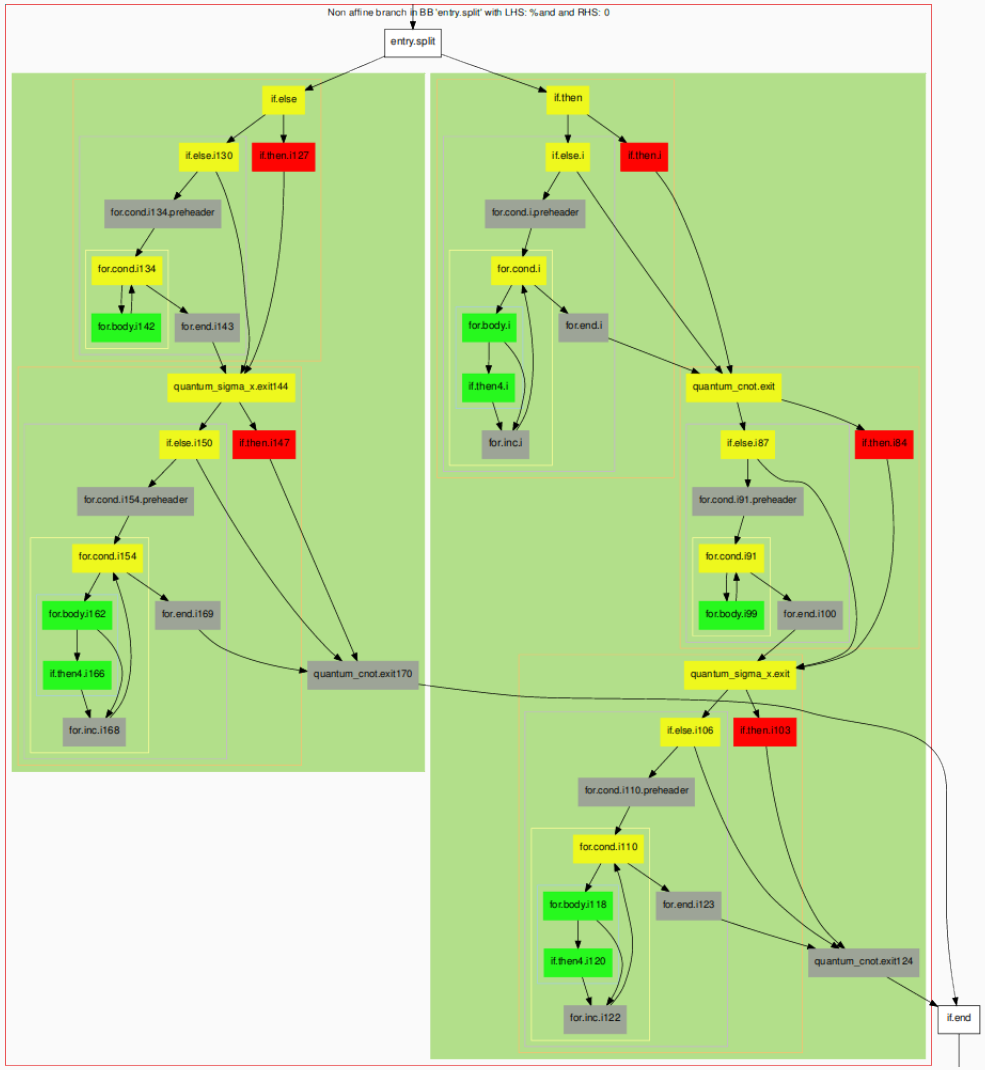

```
if (!(compare&(1<<(width-1)))) {
  quantum_sigma_x(2*width-1, reg);
  quantum_cnot(2*width-1,width-1,reg)
} else {
  quantum_cnot(2*width-1,width-1,reg)
  quantum_sigma_x(2*width-1, reg);
  quantum_cnot(2*width-1, 0, reg);
}
```

control conditions

trivial block

side-effect block

SPEC2006 — 462.LIBQUANTUM — OADDN.C




```
if (!(compare&(1<<(width-1)))) {
  quantum_sigma_x(2*width-1, reg);
  quantum_cnot(2*width-1,width-1,reg)
} else {
  quantum_cnot(2*width-1,width-1,reg)
  quantum_sigma_x(2*width-1, reg);
  quantum_cnot(2*width-1, 0, reg);
}
```

control conditions

trivial block

side-effect block

assumed to be dead

SPEC2006 — 462.LIBQUANTUM — OADDN.C

```
void quantum_decohere(quantum_reg *reg) {

  /* Increase the gate counter */
  global_gate_counter += 1;

  if (status) {
    /* Complex code, system calls, etc. */
  }
}
```

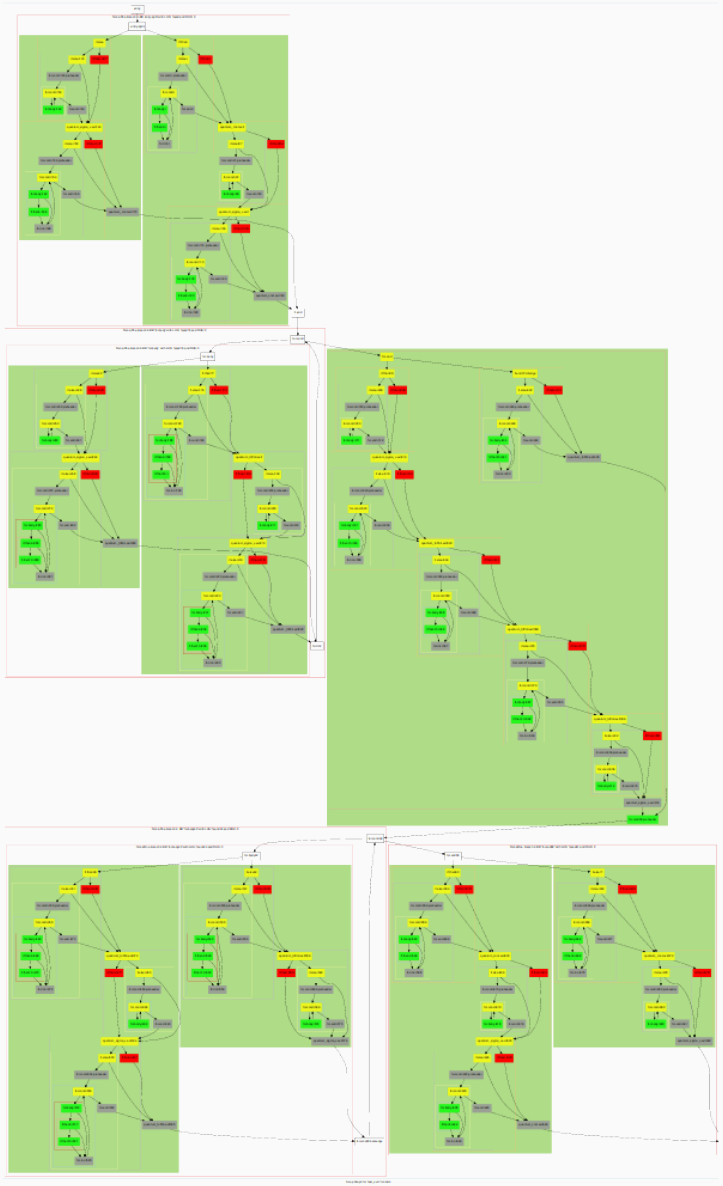
SPEC2006 — 462.LIBQUANTUM — OADDN.C

```
void quantum_decohere(quantum_reg *reg) {

  /* Increase th
  global_gate_co

  if (status) {
    /* Complex code, system calls, etc. */
  }
}
```

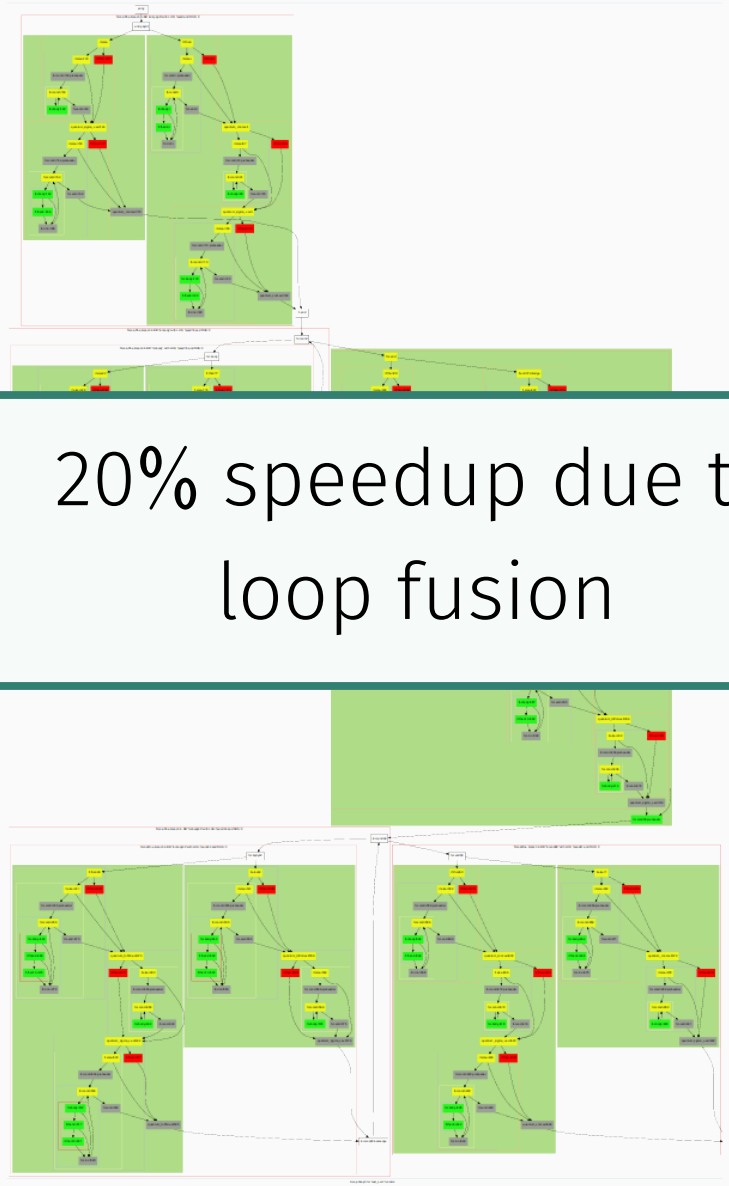
Can be analyzed for

`status == 0`

SPEC2006 — 462.libquantum — oaddn.c



SPEC2006 — 462.LIBQUANTUM — OADDN.C



20% speedup due to loop fusion

SPEC2006 — 470.lbm — lbm.c

SPEC2006 — 470.lbm — lbm.c

```
SWEEP_START( 0, 0, 0, 0, 0, SIZE_Z )
  if( TEST_FLAG_SWEEP(srcGrid, OBSTACLE)) {
    DST_C(dstGrid) = SRC_C(srcGrid);
    // 18 more lines
    continue;
  }

  rho = + SRC_C(srcGrid) + SRC_N(srcGrid) + SRC_WB(srcGrid);
  // 8 more lines

  ux = + SRC_E(srcGrid) — SRC_W(srcGrid) + /* ... */;
  // 4 more lines, and similar code for uy and uz

  if( TEST_FLAG_SWEEP(srcGrid, ACCEL)) {
    ux = 0.005; uy = 0.002; uz = 0.000;
  }

  u2 = 1.5 * (ux*ux + uy*uy + uz*uz);
  DST_C(dstGrid) = (1.—OMEGA)*SRC_C(srcGrid)+DFL1*OMEGA*rho*(1. — u2);
  // 18 more lines
SWEEP_END
```

SPEC2006 — 470.LBM — LBM.C

```
SWEEP_START( 0, 0, 0, 0, 0, SIZE_Z )
  if( TEST_FLAG_SWEEP(srcGrid, OBSTACLE)) {
    DST_C(dstGrid) = SRC_C(srcGrid);
    // 18 more lines
    continue;
  }

  rho = + SRC_C(srcGrid) + SRC_N(srcGrid) + SRC_WB(srcGrid);
  // 8 more lines

  ux = + SRC_E(srcGrid) — SRC_W(srcGrid) + /* ... */;
  // 4 more lines, and similar code for uy and uz

  if( TEST_FLAG_SWEEP(srcGrid, ACCEL)) {
    ux = 0.005; uy = 0.002; uz = 0.000;
  }

  u2 = 1.5 * (ux*ux + uy*uy + uz*uz);
  DST_C(dstGrid) = (1.—OMEGA)*SRC_C(srcGrid)+DFL1*OMEGA*rho*(1. — u2);
  // 18 more lines
SWEEP_END
```

SPEC2006 — 470.LBM — LBM.C

```
SWEEP_START( 0, 0, 0, 0, 0, SIZE_Z )
  if( TEST_FLAG_SWEEP(srcGrid, OBSTACLE)) {
    DST_C(dstGrid) = SRC_C(srcGrid);
    // 18 more lines
    continue;
  }

  rho = + SRC_C(srcGrid) + SRC_N(srcGrid) + SRC_WB(srcGrid);
  // 8 more l
  ux = + SRC_
  // 4 more lines, and similar code for uy and uz

  if( TEST_FLAG_SWEEP(srcGrid, ACCEL)) {
    ux = 0.005; uy = 0.002; uz = 0.000;
  }

  u2 = 1.5 * (ux*ux + uy*uy + uz*uz);
  DST_C(dstGrid) = (1.-OMEGA)*SRC_C(srcGrid)+DFL1*OMEGA*rho*(1. - u2);
  // 18 more lines
SWEEP_END
```

three loops collapsed to one

SPEC2006 — 470.lbm — lbm.c

```
SWEEP_START( 0, 0, 0, 0, 0, SIZE_Z )
  if( TEST_FLAG_SWEEP(srcGrid, OBSTACLE)) {
    DST_C(dstGrid) = SRC_C(srcGrid);
    // 18 more lines
    continue;
  }

  rho = + SRC_C(srcGrid) + SRC_N(srcGrid) + SRC_WB(srcGrid);
  // 8 more lines

  ux = + SRC_E(srcGrid) — SRC_W(srcGrid) + /* ... */;
  // 4 more lines, and similar code for uy and uz

  if( TEST_FLAG_SWEEP(srcGrid, ACCEL)) {
    ux = 0.005; uy = 0.002; uz = 0.000;
  }

  u2 = 1.5 * (ux*ux + uy*uy + uz*uz);
  DST_C(dstGrid) = (1.—OMEGA)*SRC_C(srcGrid)+DFL1*OMEGA*rho*(1. — u2);
  // 18 more lines
SWEEP_END
```

SPEC2006 — 470.LBM — LBM.C

```
SWEEP_START( 0, 0, 0, 0, 0, SIZE_Z )
  if( TEST_FLAG_SWEEP(srcGrid, OBSTACLE)) {
    DST_C(dstGrid) = SRC_C(srcGrid);
    // 18 more lines
    continue;
  }

  rho = + SRC                                          );
  // 8 more l

  ux = + SRC_
  // 4 more l

  if( TEST_FLAG_SWEEP(srcGrid, ACCEL)) {
    ux = 0.005; uy = 0.002; uz = 0.000;
  }

  u2 = 1.5 * (ux*ux + uy*uy + uz*uz);
  DST_C(dstGrid) = (1.-OMEGA)*SRC_C(srcGrid)+DFL1*OMEGA*rho*(1. - u2);
  // 18 more lines
SWEEP_END
```

parallel performance scales linearly with the # threads

SPEC2006 — 470.lbm — lbm.c

```
SWEEP_START( 0, 0, 0, 0, 0, SIZE_Z )
  if( TEST_FLAG_SWEEP(srcGrid, OBSTACLE)) {
    DST_C(dstGrid) = SRC_C(srcGrid);
    // 18 more lines
    continue;
  }

  rho = + SRC_C(srcGrid) + SRC_N(srcGrid) + SRC_WB(srcGrid);
  // 8 more lines

  ux = + SRC_E(srcGrid) - SRC_W(srcGrid) + /* ... */;
  // 4 more lines, and similar code for uy and uz

  if( TEST_FLAG_SWEEP(srcGrid, ACCEL)) {
    ux = 0.005; uy = 0.002; uz = 0.000;
  }

  u2 = 1.5 * (ux*ux + uy*uy + uz*uz);
  DST_C(dstGrid) = (1.-OMEGA)*SRC_C(srcGrid)+DFL1*OMEGA*rho*(1. - u2);
  // 18 more lines
SWEEP_END
```

SPEC2006 — 470.lbm — lbm.c

```
SWEEP_START( 0, 0, 0, 0, 0, SIZE_Z )
  if( TEST_FLAG_SWEEP(srcGrid, OBSTACLE)) {
    DST_C(dstGrid) = SRC_C(srcGrid);
    // 18 more lines
    continue;
  }

  rho = + SRC_C(srcGrid) + SRC_N(srcGrid) + SRC_WB(srcGrid);
  // 8 more lines

  ux = + SRC_E(srcGrid) — SRC_W(srcGrid) + /* ... */;
  // 4 more lines, and similar code for uy and uz

  if( TEST_FLAG_SWEEP(srcGrid, ACCEL)) {
    ux = 0.005; uy = 0.002; uz = 0.000;
  }

  u2 = 1.5 * (ux*ux + uy*uy + uz*uz);
  DST_C(dstGrid) = (1.—OMEGA)*SRC_C(srcGrid)+DFL1*OMEGA*rho*(1. — u2);
  // 18 more lines
SWEEP_END
```

SPEC2006 — 470.lbm — lbm.c

```
SWEEP_START( 0, 0, 0, 0, 0, SIZE_Z )
  if( TEST_FLAG_SWEEP(srcGrid, OBSTACLE)) {
    DST_C(dstGrid) = SRC_C(srcGrid);
    // 18 more lines
    continue;
  }

  rho = + SRC_C(srcGrid) + SRC_N(srcGrid) + SRC_WB(srcGrid);
  // 8 more lines

  ux = + SRC_E(srcGrid) — SRC_W(srcGrid) + /* ... */;
  // 4 more lines, and similar code for uy and uz

  if( TEST_FLAG_SWEEP(srcGrid, ACCEL)) {
    ux = 0.005; uy = 0.002; uz = 0.000;
  }

  u2 = 1.5 * (ux*ux + uy*uy + uz*uz);
  DST_C(dstGrid) = (1.—OMEGA)*SRC_C(srcGrid)+DFL1*OMEGA*rho*(1. — u2);
  // 18 more lines
SWEEP_END
```

SPEC2006 — 470.lbm — lbm.c

```
SWEEP_START( 0, 0, 0, 0, 0, SIZE_Z )
  if( TEST_FLAG_SWEEP(srcGrid, OBSTACLE)) {
    DST_C(dstGrid) = SRC_C(srcGrid);
    // 18 more lines
    continue;
  }

  rho = + SRC                                                    );
  // 8 more l

  ux = + SRC_
  // 4 more l

  if( TEST_FLAG_SWEEP(srcGrid, ACCEL)) {
    ux = 0.005; uy = 0.002; uz = 0.000;
  }

  u2 = 1.5 * (ux*ux + uy*uy + uz*uz);
  DST_C(dstGrid) = (1.-OMEGA)*SRC_C(srcGrid)+DFL1*OMEGA*rho*(1. - u2);
  // 18 more lines
SWEEP_END
```

scalars can sequentialize every surrounding loop

SPEC2006 — 470.LBM — LBM.C

```
SWEEP_START( 0, 0, 0, 0, 0, SIZE_Z )
  if( TEST_FLAG_SWEEP(srcGrid, OBSTACLE)) {
    DST_C(dstGrid) = SRC_C(srcGrid);
    // 18 more lines
    continue;
  }

  rho[0] = + SRC_C(srcGrid) + SRC_N(srcGrid) + SRC_WB(srcGrid);
  // 8 more lines

  ux[0] = + SRC_E(srcGrid) — SRC_W(srcGrid) + /* ... */;
  // 4 more lines, and similar code for uy and uz

  if( TEST_FLAG_SWEEP(srcGrid, ACCEL)) {
    ux[0] = 0.005; uy[0] = 0.002; uz[0] = 0.000;
  }

  u2[0] = 1.5 * (ux[0]*ux[0] + uy[0]*uy[0] + uz[0]*uz[0]);
  DST_C(dstGrid) = (1.—OMEGA)*SRC_C(srcGrid)+DFL1*OMEGA*rho[0]*(1.—u2[0])
  // 18 more lines
SWEEP_END
```

My To Do List

My To Do List

- Improve the statement granularity

Trade-off between compile-time and transformation potential

My To Do List

- Improve the statement granularity

Trade-off between compile-time and transformation potential

- Improve inlining heuristic

Trade-off between code size and transformation potential

My To Do List

- Improve the statement granularity

Trade-off between compile-time and transformation potential

- Improve inlining heuristic

Trade-off between code size and transformation potential

- Improve interprocedural analysis

Summarize side-effects and return values

My To Do List

- Improve the statement granularity

Trade-off between compile-time and transformation potential

- Improve inlining heuristic

Trade-off between code size and transformation potential

- Improve interprocedural analysis

Summarize side-effects and return values

- Improve handling of scalars

Privatize and propagate scalars aggressively

Polyhedral Value Analysis

Motivation:

1. Augment Scalar Evolution in LLVM passes
2. Foundation for low-level polyhedral tooling

Design Goals:

Design Goals:

- Iteration and flow sensitive

Distinguish loop iterations and control flow paths

Design Goals:

- Iteration and flow sensitive

Distinguish loop iterations and control flow paths

- Applicable and optimistic

Use partial representations and a variable scope

Design Goals:

- Iteration and flow sensitive

Distinguish loop iterations and control flow paths

- Applicable and optimistic

Use partial representations and a variable scope

- Demand driven and caching

Carefully spend compile-time

Design Goals:

- Iteration and flow sensitive

Distinguish loop iterations and control flow paths

- Applicable and optimistic

Use partial representations and a variable scope

- Demand driven and caching

Carefully spend compile-time

- Intuitive, easy to use API

Allow usage for non-polyhedral people

Comparison SE and PVA

```
j = 0;

while (j < N) {
  use(j);


  j++;
}
```

Comparison SE and PVA

```
j = 0;

while (j < N) {
  use(j);


  j++;
}
```

Scalar Evolution	Polyhedral Value Analysis
`AddRecExpr(0, +, 1)`	`{[i] -> [i] : 0 <= i < N }`

Comparison SE and PVA

```
j = 0;
assume(N > 10);
while (j < N) {
  use(j);
  if (j < 10)
    j++;
  j++;
}
```

Scalar Evolution	Polyhedral Value Analysis
CouldNotCompute	{[i] -> [2i] : 0 <= i < 5; [i] -> [5 + i] : 4 < i < N - 5 }

Comparison SE and PVA

```
if (a > b)
  x = a;
else
  x = b;
use(x);
```

Scalar Evolution	Polyhedral Value Analysis
`MaxExpr(a, b)`	`{[] -> [a] : a > b;` `[] -> [b] : b <= a }`

Comparison SE and PVA

```
if (a > b)
  x = a;
else
  x = a - 1;
use(x);
```

Scalar Evolution	Polyhedral Value Analysis
`CouldNotCompute`	`{[] -> [a] : a > b;` `[] -> [a - 1] : b <= a }`

Comparison SE and PVA

```
if (a > b)
  x = a;
else if (a < b)
  x = a - 1;
else
  x = unknown_call();
use(x);
```

Scalar Evolution	Polyhedral Value Analysis
`CouldNotCompute`	`{[] -> [a] : a > b;`
	`[] -> [a - 1] : b < a;`
	`[] -> ⊤ : a == b }`

Comparison SE and PVA

```
if (a > b)
  x = a;
else if (a < b)
  x = a - 1;
else
  x = unknown_call();
/* ... */
if (a != b)
  use(x);
```

Scalar Evolution	Polyhedral Value Analysis
`CouldNotCompute`	`{[] -> [a] : a > b;`
	`[] -> [a - 1] : b < a }`

Approximated and Derived Context

Approximated Context

Constraints under which the value is not *represented*

Approximated and Derived Context

Approximated Context

Constraints under which the value is not *represented*

Derived Context

Constraints derived from the IR, e.g., `nsw`

Approximated and Derived Context

```
signed char x = /* ... */;
if (c)
  x = a + b; // nsw
else
  x = a + b; // no nsw
use(x);
```

Scalar Evolution	Polyhedral Value Analysis
`AddExpr(a, b)`	`{[] -> [a + b] : c == true;`
	`[] -> [(a + b) mod 128] : c == false}`

Approximated and Derived Context

```
signed char x = /* ... */;
if (c)
  x = a + b; // nsw
else
  x = a + b; // no nsw
use(x);
```

Scalar Evolution	Polyhedral Value Analysis

`AddExpr(a, b)`	`{[] -> [a + b] : c == true;`
	`[] -> ⊤ : c == false}`
	Approximated: `{ c == false }`

Approximated and Derived Context

```
signed char x = /* ... */;
if (c)
  x = a + b; // nsw
else
  x = a + b; // no nsw
use(x);
```

Scalar Evolution | Polyhedral Value Analysis

Derived: { c and −128 <= a + b < 128 }

Approximated and Derived Context

```
signed char x = /* ... */;
if (c)
  x = a + b; // nsw
else
  x = a + b; // no nsw
use(x);
if (c)
  a + b; // nsw derived
```

Scalar Evolution	Polyhedral Value Analysis
`AddExpr(a, b)`	`{[] -> [a + b]}`
	Derived: `{ c and -128 <= a + b < 128 }`

Use Cases

Use Cases

- Annotate the IR with derived constraints, e.g. nsw.

Improves analysis results of other passes.

Use Cases

- Annotate the IR with derived constraints, e.g. `nsw`.

Improves analysis results of other passes.

- Symbolic domain description for CFG parts.

Improves the cost analysis for inlining and parallelization.

Use Cases

- Annotate the IR with derived constraints, e.g. `nsw`.

Improves analysis results of other passes.

- Symbolic domain description for CFG parts.

Improves the cost analysis for inlining and parallelization.

- Fall-back replacement for ScalarEvolution.

Orthogonal strengths and different applicability.

Use Cases

- Annotate the IR with derived constraints, e.g. `nsw`.

Improves analysis results of other passes.

- Symbolic domain description for CFG parts.

Improves the cost analysis for inlining and parallelization.

- Fall-back replacement for ScalarEvolution.

Orthogonal strengths and different applicability.

- Foundation for polyhedral tooling.

Demand-driven dependence analysis, loop transformations, ...

Conclusion

- Optimizing SPEC with Polly
  - 456.hmmer
  - 462.libquantum
  - 470.lbm
  - Conclusion
- Polyhedral Value Analysis
  - Design Goals and Motivation
  - Comparison with ScalarEvolution
  - Use Cases

Specializing 462.libquantum

```
void quantum_decohere(quantum_reg *reg) {

  /* Increase the gate counter */
  quantum_gate_counter(1);

  if (status) {
    nrands = calloc(reg->width, sizeof(float));
    if (!nrands) {
      printf("Not enough memory for %i-sized array!\n", reg->width);
      exit(1);
    }
    quantum_memman(reg->width * sizeof(float));

    /* ... */
```

Specializing 462.libquantum

```
void quantum_cnot(int control, int target, quantum_reg *reg) {
  int i, qec;

  quantum_qec_get_status(&qec, NULL);

  if (qec)
    quantum_cnot_ft(control, target, reg); // Multiple recursive calls
  else {
    if (quantum_objcode_put(CNOT, control, target))
      return;

    for (i = 0; i < reg->size; i++)
      if ((reg->node[i].state & ((MAX_UNSIGNED)1 << control)))
        reg->node[i].state ^= ((MAX_UNSIGNED)1 << target);

    quantum_decohere(reg); // Conditional system calls
  }
}
```

Optimistic Loop Optimization

Optimistic Loop Optimization

```
/* loop nest */
```

Optimistic Loop Optimization

1. Take *Optimistic Assumptions* to model the loop nest

```
/* loop nest */
```

Optimistic Loop Optimization

1. Take *Optimistic Assumptions* to model the loop nest
2. Optimize the loop nest

```
/* optimized loop nest */

/* loop nest */
```

Optimistic Loop Optimization

1. Take *Optimistic Assumptions* to model the loop nest
2. Optimize the loop nest
3. Version the code

```
if (                                  )
  /* optimized loop nest */
else
  /* loop nest */
```

Optimistic Loop Optimization

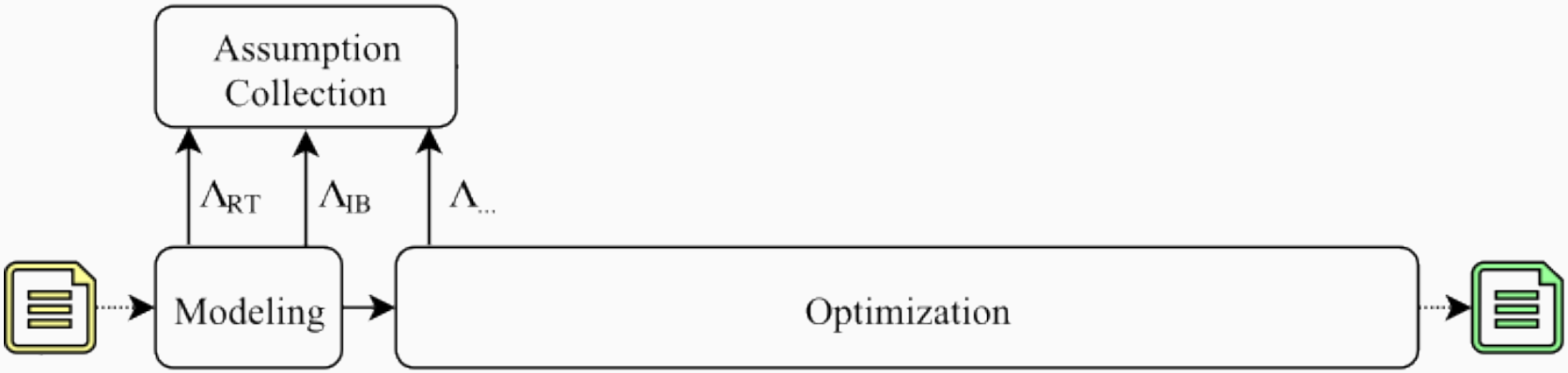
1. Take *Optimistic Assumptions* to model the loop nest
2. Optimize the loop nest
3. Version the code
4. Create a *general* and *simple* runtime check

```
if (/* simple runtime check */)
  /* optimized loop nest */
else
  /* loop nest */
```

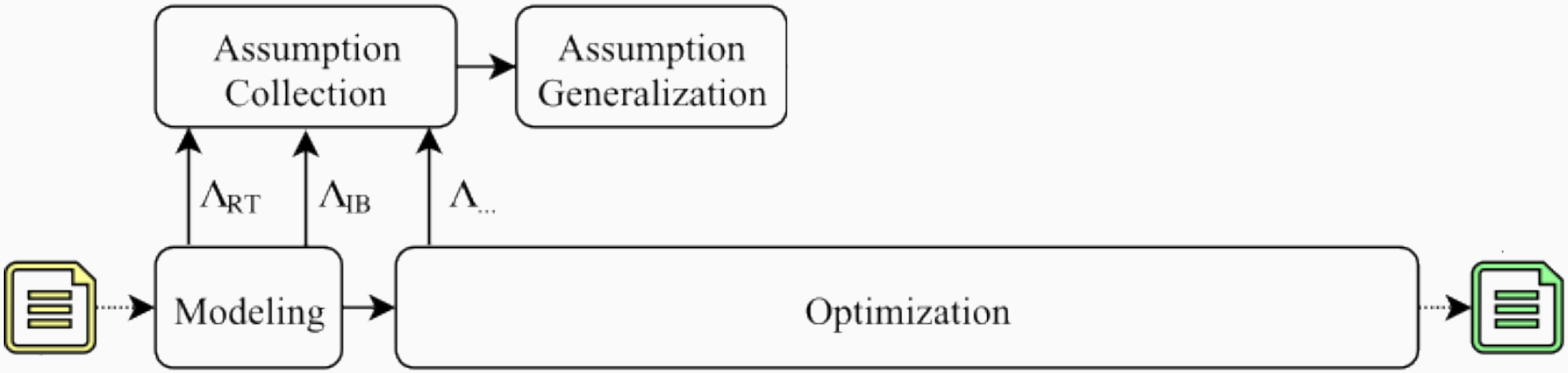
Architecture Overview

Architecture Overview

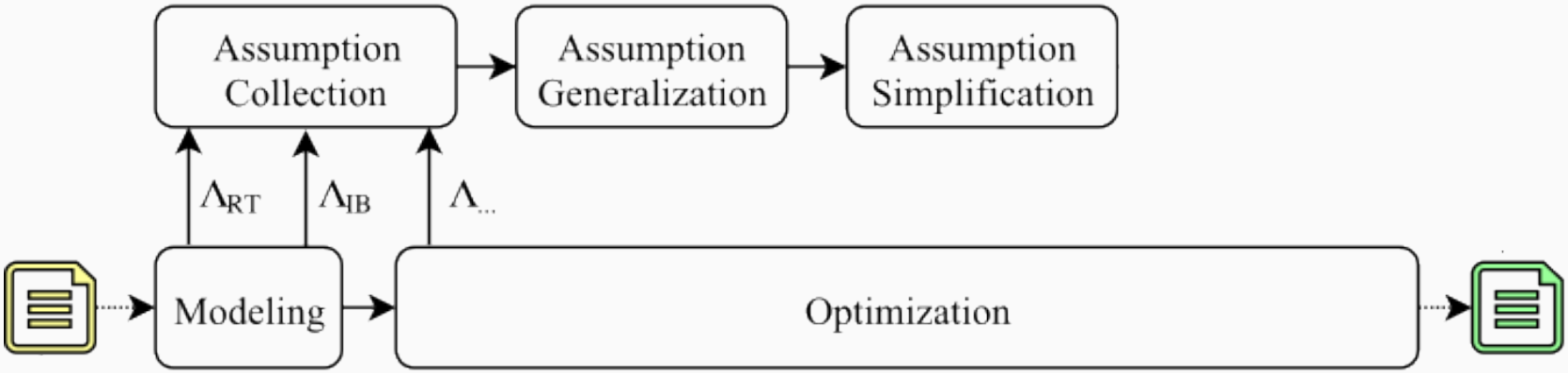
Architecture Overview



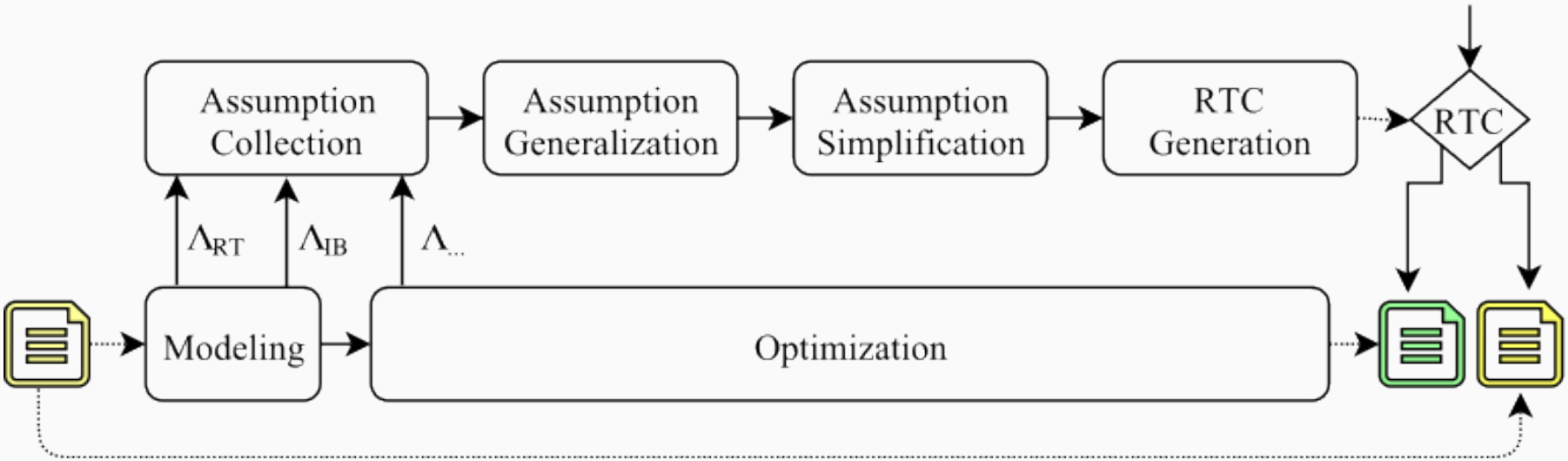
Architecture Overview



Architecture Overview



Architecture Overview



Architecture Overview