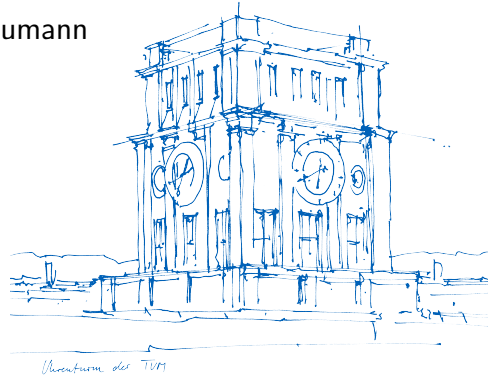
Efficiently Compiling Dynamic Code for Adaptive Query Processing

Tobias Schmidt, Philipp Fent, Thomas Neumann

Technische Universität München

ADMS'22



Adaptive Query Processing (AQP)

```
select count(*)
from lineitem
where commitdate < receiptdate
  and shipdate < commitdate
```

Adaptive Query Processing (AQP)

```
for l in lineitem:
  if not shipdate < commitdate:
    continue
  if not commitdate < receiptdate:
    continue

  counter++
```

```
select count(*)
from lineitem
where commitdate < receiptdate
  and shipdate < commitdate
```

Figure: Variant Ⓐ

Adaptive Query Processing (AQP)

```
for l in lineitem:
  if not shipdate < commitdate:
    continue
  if not commitdate < receiptdate:
    continue

  counter++
```

Figure: Variant Ⓐ

```
select count(*)
from lineitem
where commitdate < receiptdate
  and shipdate < commitdate
```

	branch-miss	instr.	exec. time
Ⓐ	0.63	7.62	18.4 ms

Table: Performance characteristics

Adaptive Query Processing (AQP)

```
for l in lineitem:
  if not shipdate < commitdate:
    continue
  if not commitdate < receiptdate:
    continue

  counter++
```

Figure: Variant Ⓐ

```
select count(*)
from lineitem
where commitdate < receiptdate
  and shipdate < commitdate
```

	branch-miss	instr.	exec. time
Ⓐ	0.63	7.62	18.4 ms

Table: Performance characteristics

Adaptive Query Processing (AQP)

```
for l in lineitem:
  if not shipdate < commitdate:
    continue -- 51% taken
  if not commitdate < receiptdate:
    continue -- 75% taken

  counter++
```

Figure: Variant Ⓐ

```
select count(*)
from lineitem
where commitdate < receiptdate
  and shipdate < commitdate
```

	branch-miss	instr.	exec. time
Ⓐ	0.63	7.62	18.4 ms

Table: Performance characteristics

Adaptive Query Processing (AQP)

```
for l in lineitem:
  if not commitdate < receiptdate:
    continue -- 37% taken
  if not shipdate < commitdate:
    continue -- 81% taken

  counter++
```

Figure: Variant Ⓑ

```
select count(*)
from lineitem
where commitdate < receiptdate
  and shipdate < commitdate
```

	branch-miss	instr.	exec. time
Ⓐ	0.63	7.62	18.4 ms
Ⓑ	0.58	7.91	17.7 ms

Table: Performance characteristics

Adaptive Query Processing (AQP)

```
for l in lineitem:
  if not (shipdate < commitdate
     and commitdate < receiptdate):
    continue -- 88% taken

  counter++
```

Figure: Variant Ⓒ

```
select count(*)
from lineitem
where commitdate < receiptdate
  and shipdate < commitdate
```

	branch-miss	instr.	exec. time
Ⓐ	0.63	7.62	18.4 ms
Ⓑ	0.58	7.91	17.7 ms
Ⓒ	0.13	11.67	**12.7 ms**

Table: Performance characteristics

Adaptive Query Processing (AQP)

UMBRA TUM

```
for l in lineitem:
  if not (shipdate < commitdate
     and commitdate < receiptdate):
    continue -- 88% taken

  counter++
```

Figure: Variant Ⓒ

```
select count(*)
from lineitem
where commitdate < receiptdate
  and shipdate < commitdate
```

	branch-miss	instr.	exec. time
Ⓐ	0.63	7.62	18.4 ms
Ⓑ	0.58	7.91	17.7 ms
Ⓒ	0.13	11.67	**12.7 ms**

Table: Performance characteristics

⇒ 50% performance improvement

AQP in Umbra

Vision: evaluate variations of the pipeline (*exploration*) and choose the best one (*exploitation*) during execution.

AQP in Umbra

Vision: evaluate variations of the pipeline (*exploration*) and choose the best one (*exploitation*) during execution.

Challenge 1: compiling pipelines multiple times is prohibitively expensive

Challenge 2: support high-level plan changes and low-level optimizations

Challenge 3: minimize the runtime overhead and avoid slowdowns

AQP in Umbra

UMBRA TUM

Vision: evaluate variations of the pipeline (*exploration*) and choose the best one (*exploitation*) during execution.

Challenge 1: compiling pipelines multiple times is prohibitively expensive

Challenge 2: support high-level plan changes and low-level optimizations

Challenge 3: minimize the runtime overhead and avoid slowdowns

⇒ **Dynamic Blocks code generation framework**

Dynamic Blocks Framework

```
for l in lineitem:

  if not shipdate <
         commitdate:
    continue

  if not commitdate <
         receiptdate:
    continue

  counter++
```

Variant Ⓐ

```
for l in lineitem:

  if not commitdate <
         receiptdate:
    continue

  if not shipdate <
         commitdate:
    continue

  counter++
```

Variant Ⓑ

```
for l in lineitem:

  if not (shipdate <
          commitdate
     and commitdate <
          receiptdate):
    continue

  counter++
```

Variant Ⓒ

Dynamic Blocks Framework

UMBRA TUM

```
for l in lineitem:
  if not shipdate <
         commitdate:
    continue
  if not commitdate <
         receiptdate:
    continue
  counter++
```

Variant Ⓐ

```
for l in lineitem:
  if not commitdate <
         receiptdate:
    continue
  if not shipdate <
         commitdate:
    continue
  counter++
```

Variant Ⓑ

```
for l in lineitem:
  if not (shipdate <
          commitdate
     and commitdate <
          receiptdate):
    continue
  counter++
```

Variant Ⓒ

Dynamic Blocks Framework

UMBRA TUM

```
for l in lineitem:
  if not shipdate <
         commitdate:
    continue
  if not commitdate <
         receiptdate:
    continue
  counter++
```

Variant Ⓐ

```
for l in lineitem:
  if not commitdate <
         receiptdate:
    continue
  if not shipdate <
         commitdate:
    continue
  counter++
```

Variant Ⓑ

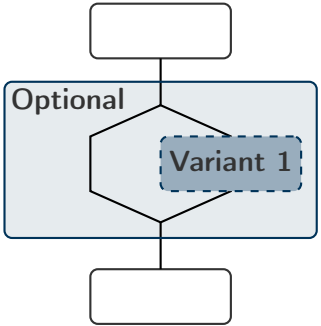
```
for l in lineitem:
  if not (shipdate <
          commitdate
     and commitdate <
          receiptdate):
    continue
  counter++
```

Variant Ⓒ

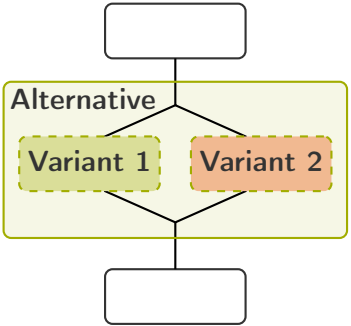
⇒ compile once with all code fragments

Dynamic Blocks Framework

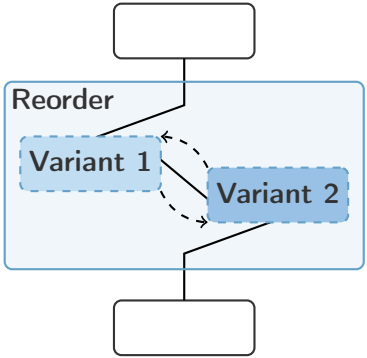
Optional Block



Alternative Block



Reorder Block



Dynamic Blocks Framework

```
for l in lineitem:
```

```
Alternative Block                    Variant 1
if not (shipdate < commitdate
  and commitdate < receiptdate):
    continue
```

```
Alternative Block                    Variant 2
Reorder Block                        Variant 2.1
if not commitdate < receiptdate:
    continue
Reorder Block                        Variant 2.2
if not shipdate < commitdate:
    continue
```

```
counter++
```

Dynamic Blocks Framework

1. Code Generation

```
for l in lineitem:
Alternative Block                          Variant 1
  if not (shipdate < commitdate
      and commitdate < receiptdate):
    continue
                                           Variant 2
  Reorder Block                            Variant 2.1
    if not commitdate < receiptdate:
      continue
                                           Variant 2.2
    if not shipdate < commitdate:
      continue

  counter++
```

```
...
Alternative Block                          Variant 1
  %3156 = cmpult i32 %3116, %2736
  %3180 = getelementptr int8 %2551,
                          i64 3670016
  %3202 = load int32 %3180, %localTid
  %3224 = cmpult i32 %3202, %3116
  %3238 = and bool %3152, %3224
  condbr %3238 %cont %contScan
                                           Variant 2
  Reorder Block                            Variant 2.1
  %3292 = cmpult i32 %3116, %2736
  condbr %3292 %cont2 %contScan
                                           Variant 2.2
  %3354 = getelementptr int8 %2551,
                          i64 3670016
  %3376 = load int32 %3354, %localTid
  %3398 = cmpult i32 %3376, %3116
  condbr %3398 %cont3 %contScan
...
```

⇒ integrate Dynamic Blocks into Umbra IR

Dynamic Blocks Framework

2. Compilation

```
...
Alternative Block                                  Variant 1
  %3156 = cmpult i32 %3116, %2736
  %3180 = getelementptr int8 %2551,
                                  i64 3670016
  %3202 = load int32 %3180, %localTid
  %3224 = cmpult i32 %3202, %3116
  %3238 = and bool %3152, %3224
  condbr %3238 %cont %contScan
                                                   Variant 2
  Reorder Block                                    Variant 2.1
    %3292 = cmpult i32 %3116, %2736
    condbr %3292 %cont2 %contScan
                                                   Variant 2.2
    %3354 = getelementptr int8 %2551,
                                  i64 3670016
    %3376 = load int32 %3354, %localTid
    %3398 = cmpult i32 %3376, %3116
    condbr %3398 %cont3 %contScan
...
```

Flying Start Compiler →

```
...
Alternative Block                                  Variant 1
  cmp r14d, ebx
  setb r13b
  mov r15, [rsp+80]
  mov r15d, [r15+r12*4+3670016]
  cmp r15d, r14d
  setb r15b
  and r13b, r15b
  cmp r13b, 1
  jnz contScan
                                                   Variant 2
  Reorder Block                                    Variant 2.1
    cmp r14d, ebx
    jae contScan
                                                   Variant 2.2
    mov r12, [rsp+80]
    mov r13, [rsp+32]
    mov r12d, [r12+r13*4+3670016]
    cmp r12d, r14d
    jae contScan
...
```

Dynamic Blocks Framework

3. Rewriting

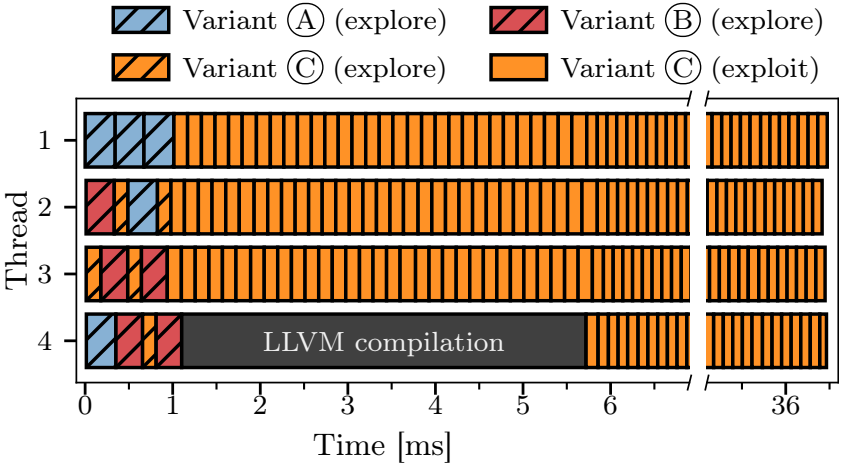
```
1   ...
Alternative Block                    Variant 1
2   cmp r14d, ebx
3   setb r13b
4   mov r15, [rsp+80]
5   mov r15d, [r15+r12*4+3670016]
6   cmp r15d, r14d
7   setb r15b
8   and r13b, r15b
9   cmp r13b, 1
10  jnz contScan
                                     Variant 2
Reorder Block                        Variant 2.1
11  cmp r14d, ebx
12  jae contScan
                                     Variant 2.2
13  mov r12, [rsp+80]
14  mov r13, [rsp+32]
15  mov r12d, [r12+r13*4+3670016]
16  cmp r12d, r14d
17  jae contScan
18  ...
```

memcpy →

```
1   ...
Alternative Block                    Variant 2
Reorder Block                        Variant 2.2
2   mov r12, [rsp+80]
3   mov r13, [rsp+32]
4   mov r12d, [r12+r13*4+3670016]
5   cmp r12d, r14d
6   jae contScan
                                     Variant 2.1
7   cmp r14d, ebx
8   jae contScan
9   ...
```

Dynamic Blocks Framework

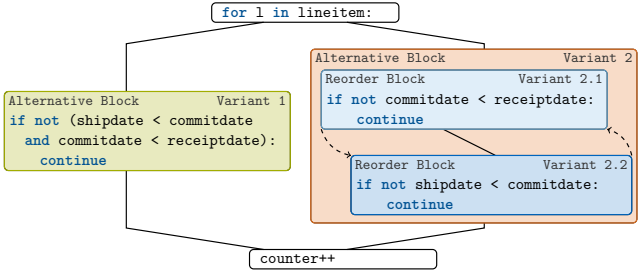
4. Execution



Dynamic Blocks Framework

Optimization: Dynamic Predicates

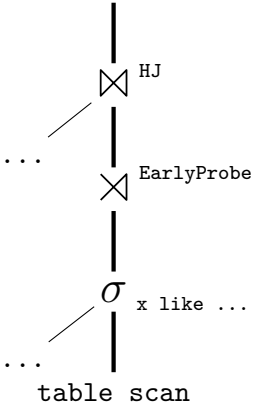
- Adapt n-ary conjunctions in joins $\bowtie$ and selections σ
- Reorder the terms or evaluate them together



Dynamic Blocks Framework

Optimization: Dynamic Join Probes

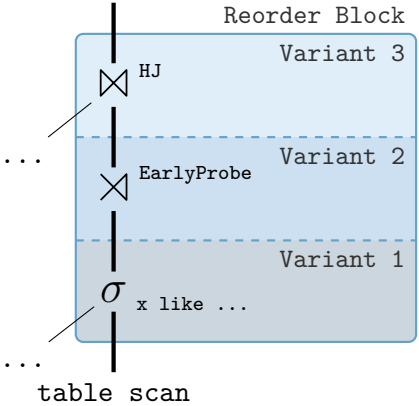
- Reorder hash joins $\bowtie$, early probes $\rtimes$, and selections σ
- Limited to operators in the same pipeline
- Pullup predicates during execution



Dynamic Blocks Framework

Optimization: Dynamic Join Probes

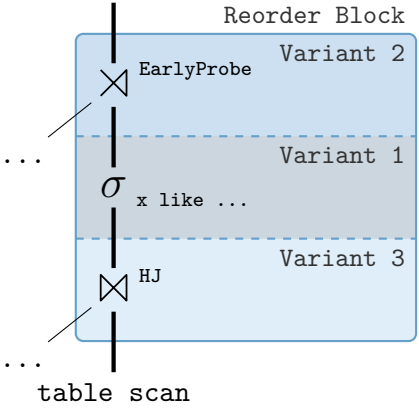
- Reorder hash joins $\bowtie$, early probes $\rtimes$, and selections σ
- Limited to operators in the same pipeline
- Pullup predicates during execution



Dynamic Blocks Framework

Optimization: Dynamic Join Probes

- Reorder hash joins $\bowtie$, early probes $\rtimes$, and selections σ
- Limited to operators in the same pipeline
- Pullup predicates during execution



Evaluation

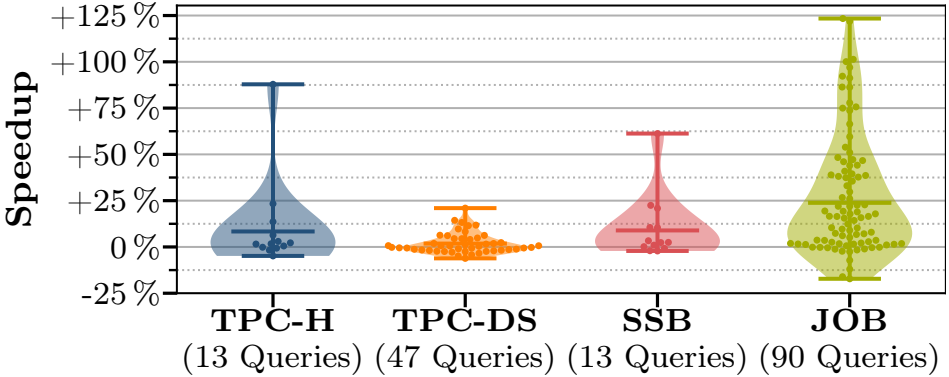
Setup

- Four OLAP benchmarks: TPC-H, TPC-DS, SSB, and JOB
- Dynamic optimizations are used by 163 queries

System:

- Intel Xeon Gold 6338 (Icelake, 2.0 GHz - 3.2 GHz)
- 32 cores
- 256 GB memory
- Ubuntu 21.10

Evaluation



⇒ more than 2× speedup, almost no overhead, and up to 25 % performance improvement on average

Conclusion

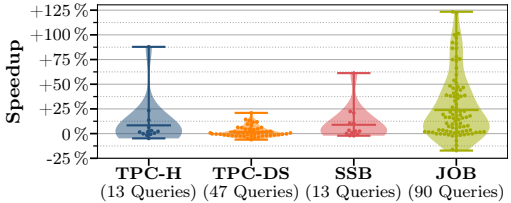
Dynamic Blocks combine the benefits of Adaptive Query Processing and Query Compilation.

Conclusion

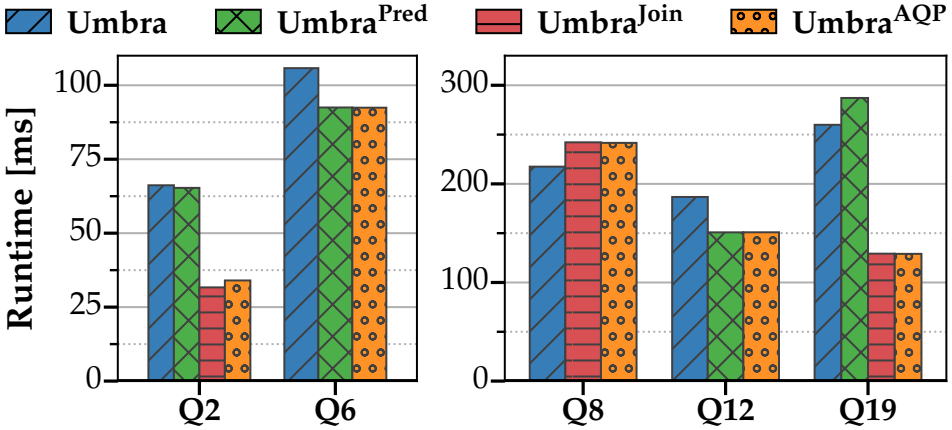
UMBRA

Dynamic Blocks combine the benefits of Adaptive Query Processing and Query Compilation.

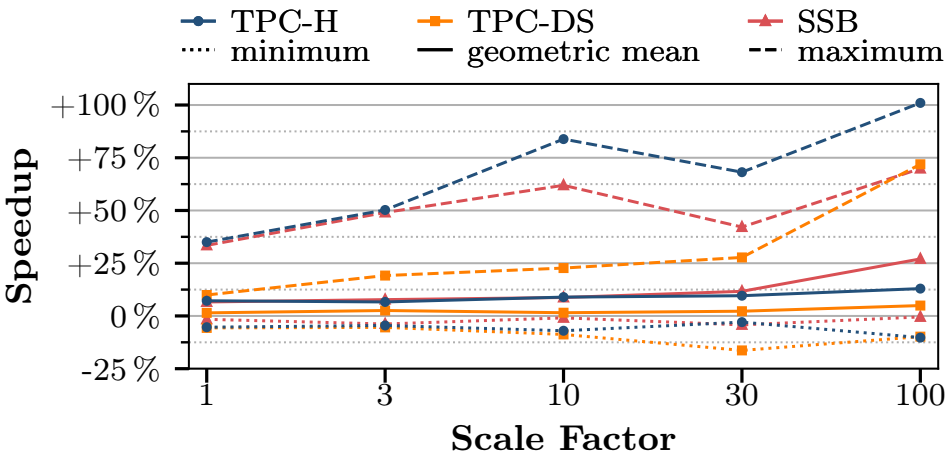
- postpone query optimization decision and perform low-level optimizations.
- generate dozens variations in a single compilation pass.
- find the best-performing implementation at run-time.



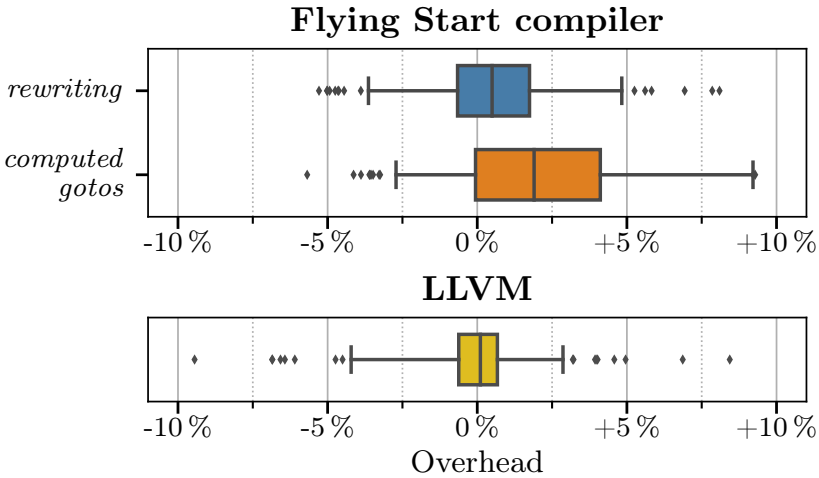
Evaluation


Umbra
Umbra^Pred
Umbra^Join
Umbra^AQP
Runtime [ms]
100
75
50
25
0
300
200
100
0
Q2
Q6
Q8
Q12
Q19

Evaluation


TPC-H
TPC-DS
SSB
minimum
geometric mean
maximum
+100 %
+75 %
+50 %
+25 %
0 %
-25 %
Speedup
1
3
10
30
100
Scale Factor

Evaluation


Flying Start compiler
rewriting
computed gotos
-10 %
-5 %
0 %
+5 %
+10 %
LLVM
-10 %
-5 %
0 %
+5 %
+10 %
Overhead