Profiling Dataflow Systems on Multiple Abstraction Levels

Alexander Beischl, Timo Kersten, Maximilian Bandle
Jana Giceva, Thomas Neumann

Technische Universität München
Compiling Dataflow Systems are Everywhere!

Dataflow systems in different areas

<table>
<thead>
<tr>
<th>Machine- and deep learning</th>
<th>Graph and stream-processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>TensorFlow</td>
<td>Flink</td>
</tr>
<tr>
<td>PyTorch</td>
<td>Ligra</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Big-data processing</th>
<th>Relational DBMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Spark</td>
<td>Microsoft Hekaton</td>
</tr>
<tr>
<td>Microsoft Naiad</td>
<td>SAP HANA</td>
</tr>
<tr>
<td></td>
<td>UMBRA</td>
</tr>
</tbody>
</table>
Profiling a Compiling Dataflow System

Trying to optimize the system

Query

```python
df_sales.join(df_CPUs,
  col("df_sales.cpuID" ===
  col("df_CPUs.ID"), "inner")
```
Profiling a Compiling Dataflow System

Trying to optimize the system

```
for tuple t in table T
  if t[1] > 5
    load int32 %40, i64 %13
    isnotnull ptr %12
    mov rax, [4 * rbx]
    cmp rax, 0
    je ...
```

```
df_sales.join(df_CPUs,
  col("df_sales.cpuID" ===
  col("df_CPUs.ID"), "inner")
```

```
for tuple t in table T
  if t[1] > 5
    load int32 %40, i64 %13
    isnotnull ptr %12
    mov rax, [4 * rbx]
```
Profiling a Compiling Dataflow System

Trying to optimize the system

Query

df_sales.join(df_CPUs, col("df_sales.cpuID" == col("df_CPUs.ID")), "inner")

Dataflow System

\[
\text{for tuple } t \text{ in table } T \\
\text{load int32 } %40, \text{ i64 } %13 \\
\text{mov rax, [4 * rbx]} \\
\text{cmp rax, 0} \\
\text{je...}
\]

Execution Time

5067ms
Profiling a Compiling Dataflow System

Trying to optimize the system

Query

```plaintext
df_sales.join(df_CPUs,
col("df_sales.cpuID" ===
col("df_CPUs.ID"), "inner")
```

Dataflow System

```plaintext
for tuple t in table T
  load int32 %40, i64 %13
  isnotnull ptr %12
  mov rax, [4 * rbx]
  cmp rax, 0
  je ...
```

Execution Time

5067ms
Profiling a Compiling Dataflow System

Trying to optimize the system

Query

```
df_sales.join(df_CPUs, col("df_sales.cpuID" == col("df_CPUs.ID"), "inner")
```
Profiling Dataflow Systems on Multiple Abstraction Levels

Profiling a Compiling Dataflow System

Trying to optimize the system

Query
\[
\text{df\_sales}\text{.join}(\text{df\_CPUs}, \\
\text{col}("\text{df\_sales}\text{.cpuID}" \text{==} \text{col}("\text{df\_CPUs}\text{.ID}")), "inner")
\]

Dataflow System

 Execution Time
5067ms

Profiling Dataflow Systems on Multiple Abstraction Levels

beischl@in.tum.de
Profiling a Compiling Dataflow System

Trying to optimize the system

Query

df_sales.join(df_CPUs, col("df_sales.cpuID" == col("df_CPUs.ID"), "inner")

Dataflow System

for tuple t in table T
load int32 %40, i64 %13
mov rax, [4 * rbx]

Execution

Execution Time

5067ms

Perf report

<table>
<thead>
<tr>
<th>loop</th>
<th>tuples:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>%localTid = phi [%1, %loopBlocks %2, %contScan]</td>
</tr>
<tr>
<td>0.1%</td>
<td>%3 = getelementptr int8 %state, i64 320</td>
</tr>
<tr>
<td>0.1%</td>
<td>%4 = getelementptr int8 %3, i64 262144</td>
</tr>
<tr>
<td>2.2%</td>
<td>%5 = load int32 %4, %localTid</td>
</tr>
<tr>
<td>2.3%</td>
<td>%7 = crc32 i64 5961697176435608501, %5</td>
</tr>
<tr>
<td>1.5%</td>
<td>%8 = crc32 i64 2231409791114444147, %5</td>
</tr>
<tr>
<td>1.2%</td>
<td>%9 = rotr i64 %8, 32</td>
</tr>
<tr>
<td>2.3%</td>
<td>%10 = xor i64 %7, %9</td>
</tr>
<tr>
<td>2.2%</td>
<td>%11 = mul i64 %10, 2685821657736338717</td>
</tr>
<tr>
<td>1.2%</td>
<td>%12 = shr %11, 16</td>
</tr>
<tr>
<td>2.4%</td>
<td>%13 = getelementptr int8 %5, i64 %12</td>
</tr>
<tr>
<td>32.1%</td>
<td>%14 = load int32 %40, i64 %13</td>
</tr>
<tr>
<td>0.2%</td>
<td>%15 = isnotnull ptr %12</td>
</tr>
<tr>
<td>0.3%</td>
<td>condbr %15 %loopHashChain %nextTuple</td>
</tr>
</tbody>
</table>

beischl@in.tum.de
Profiling a Compiling Dataflow System

Trying to optimize the system

```python
df_sales.join(df_CPUs,
    col("df_sales.cpuID" ==
    col("df_CPUs.ID"), "inner")
```

```
hashjoin
```

```
567ms
```

```
567ms
```

```
hashjoin
```

```
567ms
```

```
hashjoin
```

```
567ms
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

```
hashjoin
```

`````
Why do we have this problem?

Identifying the gap

Dataflow System

- Dataflow Graph
- Imperative Prog.
- Machine IR
- x86 Assembly

Query
Why do we have this problem?

Identifying the gap

Query

Dataflow System

- Dataflow Graph
- Imperative Prog.
- Machine IR
- x86 Assembly

```python
df_sales.join(df_CPUs, col("df_sales.cpuID") ...
```
Why do we have this problem?

Identifying the gap

Query

Dataflow System

Dataflow Graph

Imperative Prog.

Machine IR

x86 Assembly

for tuple t in table T
if t[1] > 5
...
df_sales.join(df_CPUs,
col("df_sales.cpuID")
...
Why do we have this problem?

Identifying the gap

```
for tuple t in table T
if t[1] > 5
...
```
Why do we have this problem?

Identifying the gap

---

Query

Dataflow System

- Dataflow Graph
- Imperative Program
- Machine IR
- x86 Assembly
Why do we have this problem?

Identifying the gap

```
mov rax, [4 * rbx]
cmp rax, 0
je @ ...
```
Why do we have this problem?

Identifying the gap
Why do we have this problem?
Identifying the gap

Query

Dataflow System

Dataflow Graph
Imperative Prog.
Machine IR
x86 Assembly

Execution

Profiling

Result

Profiling Samples

Machine IR Results
Why do we have this problem?

Identifying the gap

Query → Dataflow System:
- Dataflow Graph
- Imperative Prog.
- Machine IR
- x86 Assembly

Profiling Results:
- Machine IR Results
- Profiling Samples

Execution → Profiling → Result

CodeGen Dev
Why do we have this problem?

Identifying the gap
Why do we have this problem?

Identifying the gap
Why do we have this problem?

Identifying the gap

Query

Dataflow System

- Dataflow Graph
- Imperative Prog.
- Machine IR
- x86 Assembly

Connection lost

Profiling Samples

Machine IR Results

for tuple t in table T
if t[1] > 5

Domain Expert

Optimizer Dev

CodeGen Dev

Execution

Result

Profiling

beischl@in.tum.de
Tailored Profiling

Closing the gap

- Track connection between components of all abstraction levels down to generated code
- Map profiling samples back to higher abstraction levels

- Ingredients
  - Tagging Dictionary & Register Tagging

Profiling Samples
Tailored Profiling

Tagging Dictionary

① *Connection tracking* of abstraction components for each lowering step (top-down)
Tailored Profiling

Tagging Dictionary

② Store mapping in the Tagging Dictionary

Dataflow System

- Dataflow Graph
- Imperative Prog.
- Machine IR
- x86 Assembly

Connection Info → Tagging Dictionary
Tailored Profiling
Tagging Dictionary

3 Map profiling results to each abstraction level’s components (bottom-up)
Tailored Profiling

Tagging Dictionary

④ Aggregate the data for profiling results
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

Generated Query Code

for each tuple $t$ in sales

... 

if $t$.price > 500

...
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

...  
| Filter_{price>500}  
| Scan sales

Generated Query Code

for each tuple t in sales

...  

if t.price > 500

...

{for each tuple t in sales s -> Scan sales}

Tagging Dictionary
Tailored Profiling
Tagging Dictionary and Register Tagging

Dataflow Graph

- Scan sales
- Filter \( \text{price} > 500 \)

Generated Query Code

- for each \( \text{tuple} \ t \) in \( \text{sales} \)
  - ...
  - if \( t.\text{price} > 500 \)
    - ...

{if \( t.\text{price} > 500 \) -> Filter}

Tagging Dictionary
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

... 
\(\text{Filter}_{\text{price} > 500}\) 
\(\text{Scan sales}\)

Generated Query Code

\[
\text{for each tuple } t \text{ in sales} \\
\quad \text{call malloc(...)} \\
\quad \text{if } t.\text{price} > 500 \\
\quad \quad \text{call malloc(...)}
\]
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

```
...  
Filter_{price>500}  
Scan sales
```

Generated Query Code

```
for each tuple t in sales

call malloc(...)

if t.price > 500

call malloc(...)
```
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

Generated Query Code

for each tuple \( t \) in sales

\[\text{call malloc(...)}\]

if \( t.price > 500 \)

\[\text{call malloc(...)}\]

Call-Stack Sample

Recorded Call-Stack

\[\text{malloc(...)}\]

Scan: call malloc()

\[\text{...}\]
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

```
...  
Filter_{price>500}  
Scan sales  
```

Generated Query Code

```
for each tuple t in sales
    call malloc(...)  
    if t.price > 500  
        call malloc(...)  
```

Machine Register

```  
```
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

```
...  
Filter_{price>500}  
  
Scan sales
```

Generated Query Code

```
for each tuple t in sales
  setTag(Scan)
  call malloc(...)
  unsetTag()
  if t.price > 500
    setTag(Filter)
    call malloc(...)
    unsetTag()
```

Machine Register
Tailored Profiling
Tagging Dictionary and Register Tagging

Dataflow Graph

Generated Query Code

for each tuple t in sales
  setTag(Scan)
  call malloc(...)
  unsetTag()
  if t.price > 500
    setTag(Filter)
    call malloc(...)
    unsetTag()
Tailored Profiling
Tagging Dictionary and Register Tagging

Dataflow Graph

...  
Filter\(\text{price}>500\)  
Scan sales

Generated Query Code

```python
for each tuple \(t\) in sales
  setTag(Scan)
  call malloc(...)
  unsetTag()
  if \(t.\text{price} > 500\)
  setTag(Filter)
  call malloc(...)
  unsetTag()
```

Machine Register

Scan

Profiling Sample

<table>
<thead>
<tr>
<th>Source Line</th>
<th>Register Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>malloc(...)</td>
<td>Scan</td>
</tr>
</tbody>
</table>
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

```
... 
Filter_{price>500} 
Scan sales
```

Generated Query Code

```
for each tuple t in sales
  setTag(Scan)
  call malloc(...)
  unsetTag()
  if t.price > 500
    setTag(Filter)
    call malloc(...)
    unsetTag()
```

Machine Register
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

Generated Query Code

for each tuple t in sales
    setTag(Scan)
    call malloc(...)
    unsetTag()
    if t.price > 500
        setTag(Filter)
        call malloc(...)
        unsetTag()
Tailored Profiling

Tagging Dictionary and Register Tagging

Dataflow Graph

... 
Filter_{\text{price}>500} 
→ Scan sales

Generated Query Code

\begin{verbatim}
for each tuple t in sales
    setTag(Scan)
    call malloc(...) 
    unsetTag()
    if t.price > 500
        setTag(Filter)
        call malloc(...) 
        unsetTag()
\end{verbatim}

Machine Register

Filter

Profiling Sample

<table>
<thead>
<tr>
<th>Source Line</th>
<th>Register Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>malloc(...)</td>
<td>Filter</td>
</tr>
</tbody>
</table>
Insights with Tailored Profiling
Insights with Tailored Profiling

Time per operator

\( T_{l_{\text{orderkey,avg}}} \) (65.1%)
\( \sigma_{o_{\text{ord}}<1995 \cdot} \) (0.3%)
\( \xi_{o_{\text{ord}}=l_{\text{ord}}} \) (32.4%)

Tablescan lineitem (1.6%)

Tablescan orders (0.6%)

\( T_{l_{\text{orderkey}}} \)
Insights with Tailored Profiling

Time per operator

Context-aware profiling over time

\[ \Gamma_{l_{\text{orderkey}}, \text{avg}(\ldots)} (65.1\%) \]

\[ \land_{o_{\text{ord}}=l_{\text{ord}}} (32.4\%) \]

\[ \sigma_{o_{\text{ord}}< '1995'} (0.3\%) \]

Tablescan

lineitem (1.6%)

orders (0.6%)

groupby

join

products

sales

Time [ms]

Operator Activity [%]

grooby

join

products

sales
Insights with Tailored Profiling

Time per operator

Context-aware profiling over time

Memory access patterns
Overhead and Integration Effort

Is it worth it?
Overhead and Integration Effort

Is it worth it?

- Little implementation effort

<table>
<thead>
<tr>
<th>Component</th>
<th>Lines Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Gen.</td>
<td>56</td>
</tr>
<tr>
<td>Sample Processing</td>
<td>1176</td>
</tr>
<tr>
<td>Visualizations</td>
<td>510</td>
</tr>
</tbody>
</table>
Overhead and Integration Effort

Is it worth it?

- Little implementation effort
- High accuracy

<table>
<thead>
<tr>
<th>Component</th>
<th>Lines Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Gen.</td>
<td>56</td>
</tr>
<tr>
<td>Sample Processing</td>
<td>1176</td>
</tr>
<tr>
<td>Visualizations</td>
<td>510</td>
</tr>
<tr>
<td>Attributed Samples</td>
<td>98 %</td>
</tr>
</tbody>
</table>
Overhead and Integration Effort

Is it worth it?

- Little implementation effort
- High accuracy
- Lightweight
  - Small query execution overhead
    - Tagging Dictionary: populated at compile time
    - Register Tagging: 3%
  - Cheaper than call-stack sampling

<table>
<thead>
<tr>
<th>Component</th>
<th>Lines Added</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Gen.</td>
<td>56</td>
</tr>
<tr>
<td>Sample Processing</td>
<td>1176</td>
</tr>
<tr>
<td>Visualizations</td>
<td>510</td>
</tr>
<tr>
<td>Attributed Samples</td>
<td>98%</td>
</tr>
</tbody>
</table>

![Sampling Frequency vs. Overhead Graph]

- Profiling
  - Sampling
  - Sampling, Register Tagging
  - Call-stack
Impact of Tailored Profiling

Where can you apply it?

- Preserve connection information to close gap
- Profiling results on high abstraction levels
Impact of Tailored Profiling

Where can you apply it?

- Preserve connection information to close gap
- Profiling results on high abstraction levels
- Lightweight, high accuracy
- Easy to integrate
- Applicable to many systems
Impact of Tailored Profiling

Where can you apply it?

- Preserve connection information to close gap
- Profiling results on high abstraction levels
- Lightweight, high accuracy
- Easy to integrate
- Applicable to many systems

- Already supported: profiling code on CPUs (multi-socket and multicore)
- Future work: heterogenous compute resources, distributed systems
What else can be found in the Paper

- Technical and implementation details
- Tagging Dictionary for multiple abstraction levels
- Register Tagging and call-stack sampling
- Supported optimizations
What else can be found in the Paper

- Technical and implementation details
- Tagging Dictionary for multiple abstraction levels
- Register Tagging and call-stack sampling
- Supported optimizations
- Integration details for our compiling DBMS Umbra
What else can be found in the Paper

- Technical and implementation details
- Tagging Dictionary for multiple abstraction levels
- Register Tagging and call-stack sampling
- Supported optimizations

- Integration details for our compiling DBMS Umbra
- More detailed evaluation, runtime overhead and use cases
Thank you for watching!