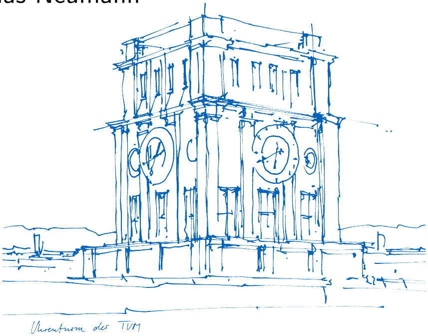
Exploiting Code Generation for Efficient LIKE Pattern Matching

Adrian Riedl, Philipp Fent, Maximilian Bandle, Thomas Neumann

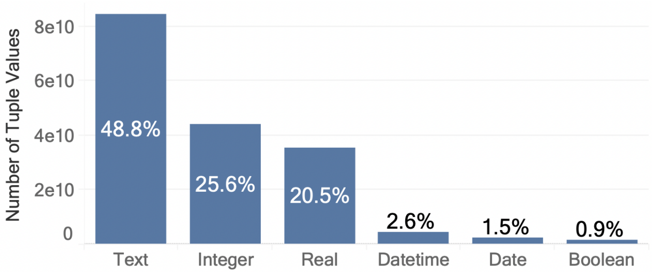
Technical University of Munich

ADMS @ VLDB 2023



Strings are everywhere

➔ Require efficient text operations



From: Vogelsgesang et al., "Get Real: How Benchmarks Fail to Represent the Real World"

Strings are everywhere

- Require efficient text operations
- Should a DBMS use third-party libraries for string processing?

Strings are everywhere

- Require efficient text operations
- Should a DBMS use third-party libraries for string processing?
- Expensive string conversions due to DBMS specific format

Strings are everywhere

- Require efficient text operations
- Should a DBMS use third-party libraries for string processing?
- Expensive string conversions due to DBMS specific format
- Per-tuple overhead causes significant performance loss

Strings are everywhere

- ➔ Require efficient text operations
- ➔ Should a DBMS use third-party libraries for string processing?
- ⚠ Expensive string conversions due to DBMS specific format
- ⚠ Per-tuple overhead causes significant performance loss

➔ **We need to integrate text operations better!**

Approaches

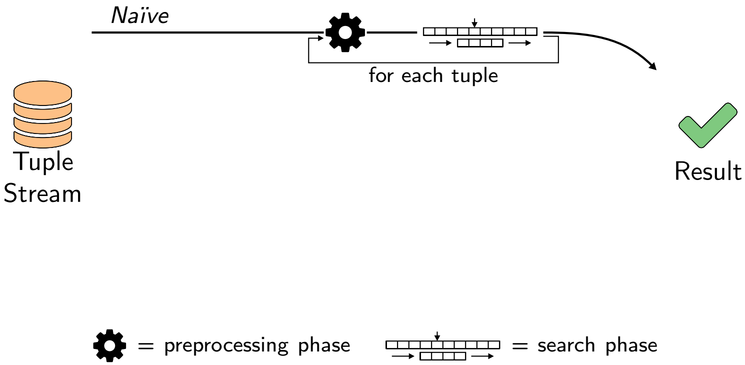
```sql
select count(*) from uni where name like '%TUM%';
```

Tuple
Stream

Result

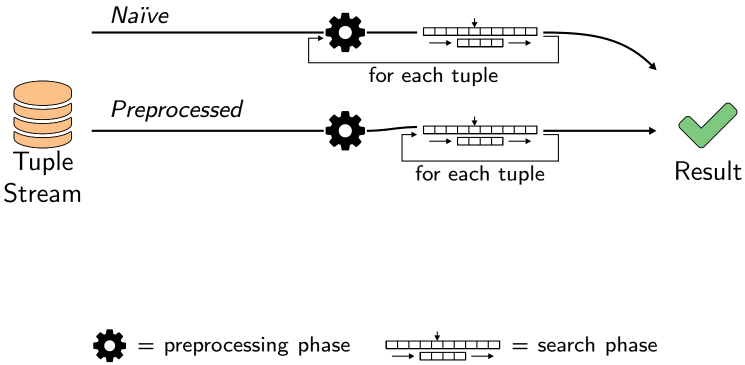
Approaches

```
select count(*) from uni where name like '%TUM%';
```



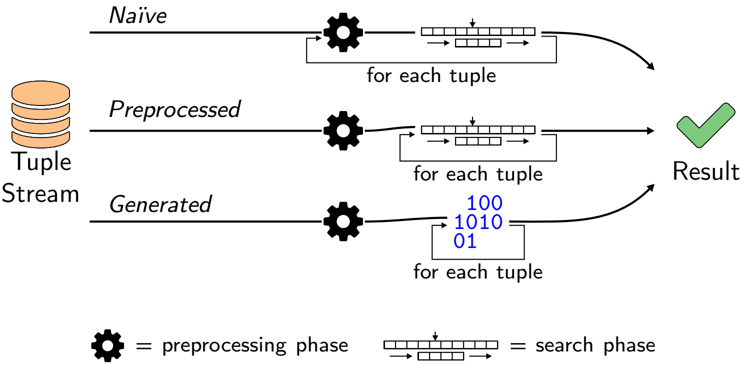
Approaches

```sql
select count(*) from uni where name like '%TUM%';
```



Approaches

```
select count(*) from uni where name like '%TUM%';
```



Knuth-Morris-Pratt Algorithm

From Naïve to Preprocessed

Naïve approach

```
KMP(text, pattern):
  lpsTable = preprocess(pattern);
  pPos = 0;
  pSize = pattern.size();
  tPos = 0;
  tSize = text.size();
  while (tPos - pPos + pSize <= tSize)
    if (pattern[pPos] == text[tPos])
      pPos++; tPos++;
      if (pPos == pSize) return true;
    else
      shift = lpsTable[pPos];
      if (shift < 0) pPos = 0; tPos++;
      else pPos = shift;
  return false;
```

Knuth-Morris-Pratt Algorithm

From Naïve to Preprocessed

Naïve approach

```
KMP(text, pattern):
  lpsTable = preprocess(pattern);
  pPos = 0;
  pSize = pattern.size();
  tPos = 0;
  tSize = text.size();
  while (tPos - pPos + pSize <= tSize)
    if (pattern[pPos] == text[tPos])
      pPos++; tPos++;
      if (pPos == pSize) return true;
    else
      shift = lpsTable[pPos];
      if (shift < 0) pPos = 0; tPos++;
      else pPos = shift;
  return false;
```

Knuth-Morris-Pratt Algorithm

From Naïve to Preprocessed

Naïve approach

```
KMP(text, pattern):
  lpsTable = preprocess(pattern);
  pPos = 0;
  pSize = pattern.size();
  tPos = 0;
  tSize = text.size();
  while (tPos - pPos + pSize <= tSize)
    if (pattern[pPos] == text[tPos])
      pPos++; tPos++;
      if (pPos == pSize) return true;
    else
      shift = lpsTable[pPos];
      if (shift < 0) pPos = 0; tPos++;
      else pPos = shift;
  return false;
```

Knuth-Morris-Pratt Algorithm

From Naïve to Preprocessed

Naïve approach

```
KMP(text, pattern):
  lpsTable = preprocess(pattern);
  pPos = 0;
  pSize = pattern.size();
  tPos = 0;
  tSize = text.size();
  while (tPos - pPos + pSize <= tSize)
    if (pattern[pPos] == text[tPos])
      pPos++; tPos++;
      if (pPos == pSize) return true;
    else
      shift = lpsTable[pPos];
      if (shift < 0) pPos = 0; tPos++;
      else pPos = shift;
  return false;
```

Knuth-Morris-Pratt Algorithm

From Naïve to Preprocessed

Naïve approach

```
KMP(text, pattern):
  lpsTable = preprocess(pattern);
  pPos = 0;
  pSize = pattern.size();
  tPos = 0;
  tSize = text.size();
  while (tPos - pPos + pSize <= tSize)
    if (pattern[pPos] == text[tPos])
      pPos++; tPos++;
      if (pPos == pSize) return true;
    else
      shift = lpsTable[pPos];
      if (shift < 0) pPos = 0; tPos++;
      else pPos = shift;
  return false;
```

Knuth-Morris-Pratt Algorithm

From Naïve to Preprocessed

Naïve approach

```
KMP(text, pattern):
  lpsTable = preprocess(pattern);
  pPos = 0;
  pSize = pattern.size();
  tPos = 0;
  tSize = text.size();
  while (tPos - pPos + pSize <= tSize)
    if (pattern[pPos] == text[tPos])
      pPos++; tPos++;
      if (pPos == pSize) return true;
    else
      shift = lpsTable[pPos];
      if (shift < 0) pPos = 0; tPos++;
      else pPos = shift;
  return false;
```

Extract pre-processing →

Knuth-Morris-Pratt Algorithm

From Naïve to Preprocessed

Naïve approach

```
KMP(text, pattern):
  lpsTable = preprocess(pattern);
  pPos = 0;
  pSize = pattern.size();
  tPos = 0;
  tSize = text.size();
  while (tPos - pPos + pSize <= tSize)
    if (pattern[pPos] == text[tPos])
      pPos++; tPos++;
      if (pPos == pSize) return true;
    else
      shift = lpsTable[pPos];
      if (shift < 0) pPos = 0; tPos++;
      else pPos = shift;
  return false;
```

Extract pre-processing →

Preprocessed approach

```
KMP(text, pattern, lpsTable):

  pPos = 0;
  pSize = pattern.size();
  tPos = 0;
  tSize = text.size();
  while (tPos - pPos + pSize <= tSize)
    if (pattern[pPos] == text[tPos])
      pPos++; tPos++;
      if (pPos == pSize) return true;
    else
      shift = lpsTable[pPos];
      if (shift < 0) pPos = 0; tPos++;
      else pPos = shift;
  return false;
```

Knuth-Morris-Pratt Algorithm

From Preprocessed to Generated for '%TUM%'

Preprocessed approach

```
KMP(text, pattern, lpsTable):
  pPos = 0;
  pSize = pattern.size();
  tPos = 0;
  tSize = text.size();
  while (tPos - pPos + pSize <= tSize)
    if (pattern[pPos] == text[tPos])
      pPos++; tPos++;
      if (pPos == pSize) return true;
    else
      shift = lpsTable[pPos];
      if (shift < 0) pPos = 0; tPos++;
      else pPos = shift;
  return false;
```

Knuth-Morris-Pratt Algorithm

From Preprocessed to Generated for '%TUM%'

Preprocessed approach

```
KMP(text, pattern, lpsTable):
  pPos = 0;
  pSize = pattern.size();
  tPos = 0;
  tSize = text.size();
  while (tPos - pPos + pSize <= tSize)
    if (pattern[pPos] == text[tPos])
      pPos++; tPos++;
      if (pPos == pSize) return true;
    else
      shift = lpsTable[pPos];
      if (shift < 0) pPos = 0; tPos++;
      else pPos = shift;
  return false;
```

Specialize →

Generated approach

Knuth-Morris-Pratt Algorithm

From Preprocessed to Generated for `'%TUM%'`

Preprocessed approach

```
KMP(text, pattern, lpsTable):
  pPos = 0;
  pSize = pattern.size();
  tPos = 0;
  tSize = text.size();
  while (tPos - pPos + pSize <= tSize)
    if (pattern[pPos] == text[tPos])
      pPos++; tPos++;
      if (pPos == pSize) return true;
    else
      shift = lpsTable[pPos];
      if (shift < 0) pPos = 0; tPos++;
      else pPos = shift;
  return false;
```

Specialize →

Generated approach

whileLoopHeader:
check tPos - pPos + 3 $\leq$ text.size()

`return false`

Knuth-Morris-Pratt Algorithm

From Preprocessed to Generated for '%TUM%'

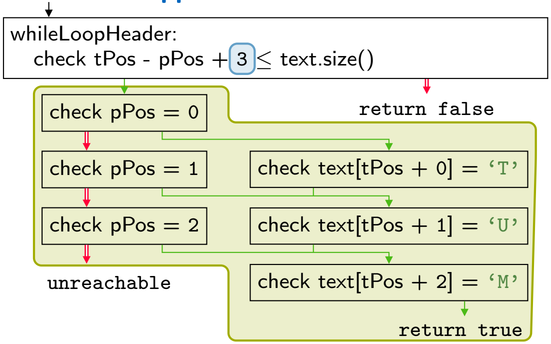
UMBRA

Preprocessed approach

```
KMP(text, pattern, lpsTable):
  pPos = 0;
  pSize = pattern.size();
  tPos = 0;
  tSize = text.size();
  while (tPos - pPos + pSize <= tSize)
    if (pattern[pPos] == text[tPos])
      pPos++; tPos++;
      if (pPos == pSize) return true;
    else
      shift = lpsTable[pPos];
      if (shift < 0) pPos = 0; tPos++;
      else pPos = shift;
  return false;
```

Specialize

Generated approach



Knuth-Morris-Pratt Algorithm

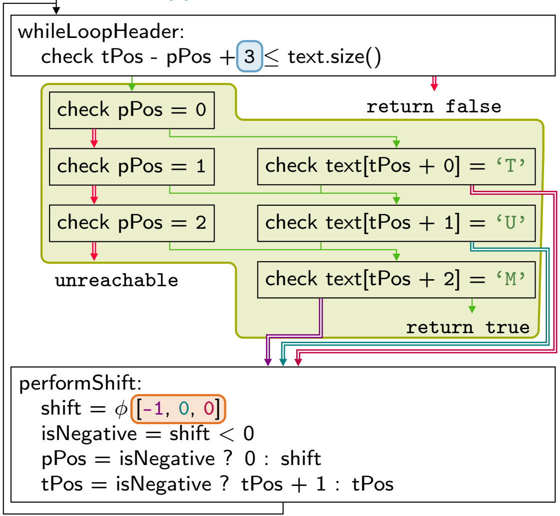
From Preprocessed to Generated for `'%TUM%'`

Preprocessed approach

```
KMP(text, pattern, lpsTable):
  pPos = 0;
  pSize = pattern.size();
  tPos = 0;
  tSize = text.size();
  while (tPos - pPos + pSize <= tSize)
    if (pattern[pPos] == text[tPos])
      pPos++; tPos++;
      if (pPos == pSize) return true;
    else
      shift = lpsTable[pPos];
      if (shift < 0) pPos = 0; tPos++;
      else pPos = shift;
  return false;
```

Specialize

Generated approach

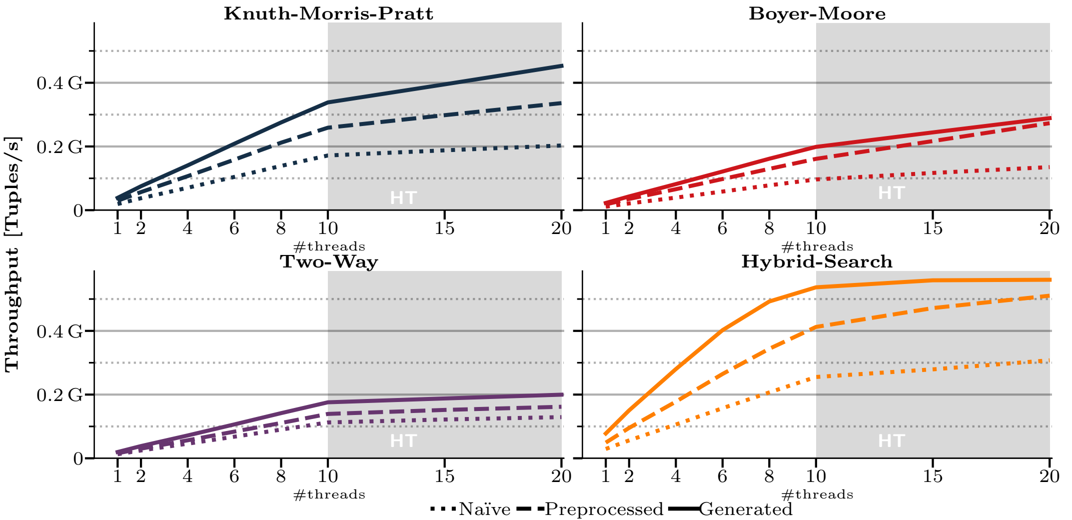


Evaluation

Multi-threaded performance on Clickbench: `url like '%google%'`

Evaluation

Multi-threaded performance on Clickbench: `url like '%google%'`



Hybrid-Search

UMBRA TUM

- Uses SSE instruction: `pcmpistri`

```
HS(text, pattern):
  if (pattern.size() <= 16 && text.size() >= 16)
    iter = text.begin(), end = text.end();
    safeMatch = 17 - pattern.size();
    pattern16 = load16(pattern);
    while ((iter + 16) < end)
      match = pcmpistri(pattern16, load16(iter));
      if (match < safeMatch) return true;
      iter += safeMatch;
    if (iter < end)
      match = pcmpistri(pattern16, load16(end - 16));
      return match < safeMatch;
    return false;
  return fallback(text, pattern);
```

Hybrid-Search

- Uses SSE instruction: `pcmpistri`
- No preprocessing needed

```
HS(text, pattern):
  if (pattern.size() <= 16 && text.size() >= 16)
    iter = text.begin(), end = text.end();
    safeMatch = 17 - pattern.size();
    pattern16 = load16(pattern);
    while ((iter + 16) < end)
      match = pcmpistri(pattern16, load16(iter));
      if (match < safeMatch) return true;
      iter += safeMatch;
    if (iter < end)
      match = pcmpistri(pattern16, load16(end - 16));
      return match < safeMatch;
    return false;
  return fallback(text, pattern);
```

Hybrid-Search

UMBRA

- Uses SSE instruction: `pcmpistri`
- No preprocessing needed
- Consumes 16 bytes of input text at once

```
HS(text, pattern):
  if (pattern.size() <= 16 && text.size() >= 16)
    iter = text.begin(), end = text.end();
    safeMatch = 17 - pattern.size();
    pattern16 = load16(pattern);
    while ((iter + 16) < end)
      match = pcmpistri(pattern16, load16(iter));
      if (match < safeMatch) return true;
      iter += safeMatch;
    if (iter < end)
      match = pcmpistri(pattern16, load16(end - 16));
      return match < safeMatch;
    return false;
  return fallback(text, pattern);
```

Hybrid-Search

UMBRA TUM

- Uses SSE instruction: `pcmpistri`
- No preprocessing needed
- Consumes 16 bytes of input text at once
- ✔ Generated approach is straightforward

```
HS(text, pattern):
  if (pattern.size() <= 16 && text.size() >= 16)
    iter = text.begin(), end = text.end();
    safeMatch = 17 - pattern.size();
    pattern16 = load16(pattern);
    while ((iter + 16) < end)
      match = pcmpistri(pattern16, load16(iter));
      if (match < safeMatch) return true;
      iter += safeMatch;
    if (iter < end)
      match = pcmpistri(pattern16, load16(end - 16));
      return match < safeMatch;
    return false;
  return fallback(text, pattern);
```

Hybrid-Search

UMBRA

- Uses SSE instruction: `pcmpistri`
- No preprocessing needed
- Consumes 16 bytes of input text at once
- Generated approach is straightforward
- Limited to short patterns

```
HS(text, pattern):
  if (pattern.size() <= 16 && text.size() >= 16)
    iter = text.begin(), end = text.end();
    safeMatch = 17 - pattern.size();
    pattern16 = load16(pattern);
    while ((iter + 16) < end)
      match = pcmpistri(pattern16, load16(iter));
      if (match < safeMatch) return true;
      iter += safeMatch;
    if (iter < end)
      match = pcmpistri(pattern16, load16(end - 16));
      return match < safeMatch;
    return false;
  return fallback(text, pattern);
```

Hybrid-Search

- Uses SSE instruction: `pcmpistri`
- No preprocessing needed
- Consumes 16 bytes of input text at once
- ✔ Generated approach is straightforward
- ⚠ Limited to short patterns

```
HS(text, pattern):
  if (pattern.size() <= 16 && text.size() >= 16)
    iter = text.begin(), end = text.end();
    safeMatch = 17 - pattern.size();
    pattern16 = load16(pattern);
    while ((iter + 16) < end)
      match = pcmpistri(pattern16, load16(iter));
      if (match < safeMatch) return true;
      iter += safeMatch;
    if (iter < end)
      match = pcmpistri(pattern16, load16(end - 16));
      return match < safeMatch;
    return false;
  return fallback(text, pattern);
```

→ **Code generation allows to generate code specifically for longer patterns**

SSE Search

Example: `'%Technical University of Munich%'`

SSE Search

Example: `'%Technical University of Munich%'`

whileLoopHeader:
check tPos + $\boxed{30}$ $\leq$ text.size()

`return false`

SSE Search

Example: `'%Technical University of Munich%'`

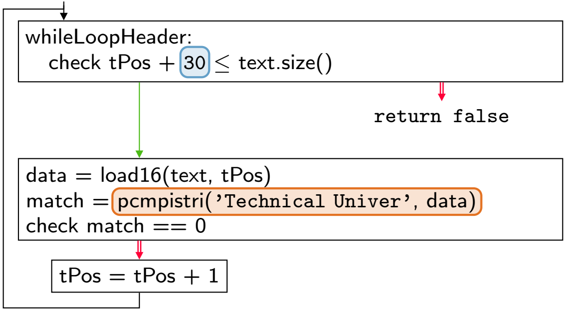
whileLoopHeader:
  check tPos + $\boxed{30} \leq$ text.size()

`return false`

data = load16(text, tPos)
match = $\boxed{\text{pcmpistri('Technical Univer', data)}}$
check match == 0

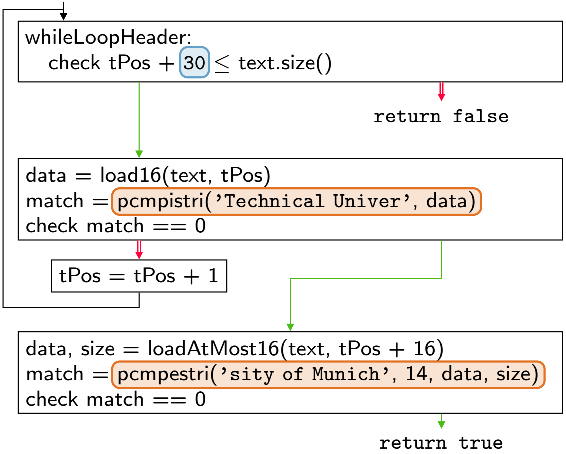
SSE Search

Example: `'%Technical University of Munich%'`



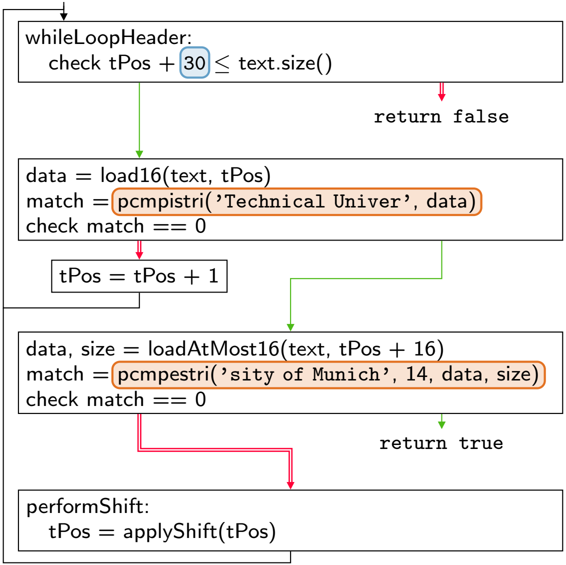
SSE Search

Example: `'%Technical University of Munich%'`



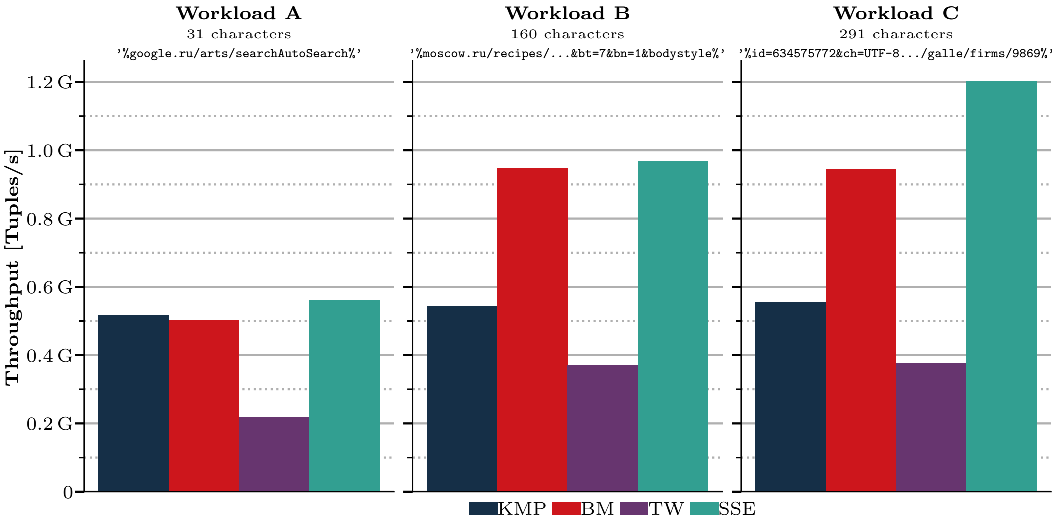
SSE Search

Example: `'%Technical University of Munich%'`



Evaluation

Long patterns



Exploiting Code Generation for Efficient LIKE Pattern Matching

➔ Replacing generic function calls with pattern-specific code

Adrian Riedl
Technical University of Munich
Adrian.Riedl@in.tum.de

Exploiting Code Generation for Efficient LIKE Pattern Matching

- ➔ Replacing generic function calls with pattern-specific code
- ➔ Throughput improvement by up to 2.5×

Adrian Riedl
Technical University of Munich
Adrian.Riedl@in.tum.de

Exploiting Code Generation for Efficient LIKE Pattern Matching

- Replacing generic function calls with pattern-specific code
- Throughput improvement by up to 2.5$\times$
- Generating specialized code with SSE instructions for longer pattern

Adrian Riedl
Technical University of Munich
Adrian.Riedl@in.tum.de