An Efficient Subsequence Similarity Search on Modern Intel Many-core Processors for Data Intensive Applications

Yana Kraeva and Mikhail Zymbler
South Ural State University, Chelyabinsk, Russia
kraevaya@susu.ru, mzym@susu.ru

This work was financially supported by the Russian Foundation for Basic Research (grant No. 17-07-00463), by Act 211 Government of the Russian Federation (contract No. 02.A03.21.0011) and by the Ministry of education and science of Russian Federation (government order 1.9624.2017/7.8).
Similarity search: typical applications

**Medicine:**
search for heart attack patterns

*Pattern of heart attack*

**Entomology:**
studying behavior of insects

*Behavior pattern*
Subsequence similarity search

Sliding window of length $n$

Number of subsequences

$N = m - n + 1$

$\exists i > 1 \forall j < N \quad DTW(Q, T_{i,n}) < DTW(Q, T_{j,n})$
Subsequence similarity search

Sliding window of length \( n \)

Number of subsequences
\[ N = m - n + 1 \]

\[ \exists i > 1 \quad \forall j < N \quad DTW(Q, T_{i,n}) < DTW(Q, T_{j,n}) \]
An Efficient Subsequence Similarity Search on Modern Intel MIC Processors for Data Intensive Applications

Subsequence similarity search

∃ i > 1 ∀ j < N \( DTW(Q, T_{i,n}) < DTW(Q, T_{j,n}) \)

Number of subsequences

\( N = m - n + 1 \)

Sliding window of length \( n \)

\( \exists i > 1 \forall j < N \)
Dynamic Time Warping measure

\[ DTW(Q, C) = d(n,n), \]

\[ d(i, j) = (q_i - c_j)^2 + \min\left\{ \begin{array}{ll}
  d(i-1, j) \\
  d(i, j-1) \\
  d(i-1, j-1)
\end{array} \right. \]

\[ d(0,0) = 0; d(i,0) = d(0, j) = \infty; 1 \leq i, j \leq n. \]
Dynamic Time Warping measure

\[ DTW(Q, C) = d(n, n), \]

\[ d(i, j) = (q_i - c_j)^2 + \min \begin{cases} 
  d(i - 1, j) \\
  d(i, j - 1) \\
  d(i - 1, j - 1), 
\end{cases} \]

\[ d(0,0) = 0; d(i,0) = d(0, j) = \infty; 1 \leq i, j \leq n. \]

Warping constraint (Sako–Chiba band)
• Z-normalization
  – The mean and standard deviation of query and each candidate subsequence must be normalized 0 and 1, respectively
  \[
  \hat{t}_i = \frac{t_i - \mu}{\sigma}, \quad \mu = \frac{1}{m} \sum_{i=1}^{m} t_i, \quad \sigma = \frac{1}{m} \sqrt{\sum_{i=1}^{m} t_i^2 - \mu^2}
  \]

• Lower bounding
  – Cascade of cheap-to-compute lower bounds (LB) helps to prune clearly dissimilar candidates
  – If LB has exceeded the threshold, the DTW measure will exceed the threshold as well
Lower bounding

Lower Bound

\[ \hat{L}_{B_{Kim}}(Q, C) = (q_1 - c_1)^2 + (q_n - c_n)^2 \]

Complexity

\[ O(1) \]
### Lower bounding

#### Lower Bound

<table>
<thead>
<tr>
<th>Complexity</th>
<th>Lower Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1)$</td>
<td>$LB_{Kim}(Q, C) = (q_1 - c_1)^2 + (q_n - c_n)^2$</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>$LB_{KeoghEC}(Q, C) = \sum_{i=1}^{n} \begin{cases} (c_i - u_i)^2, &amp; c_i &gt; u_i \ (c_i - \ell_i)^2, &amp; c_i &lt; \ell_i \ 0, &amp; \text{otherwise} \end{cases}$</td>
</tr>
</tbody>
</table>

$$u_i = \max_{i-r\leq k \leq i+r} q_k$$

$$\ell_i = \min_{i-r\leq k \leq i+r} q_k$$
### Lower Bound

<table>
<thead>
<tr>
<th>Lower Bound</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>( LB_{Kim}(Q, C) = (q_1 - c_1)^2 + (q_n - c_n)^2 )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>( LB_{Keogh\text{EC}}(Q, C) = \sum_{i=1}^{n} \begin{cases} \ (c_i - u_i)^2, &amp; c_i &gt; u_i \ \ (c_i - \ell_i)^2, &amp; c_i &lt; \ell_i \ \ 0, &amp; \text{otherwise} \end{cases} )</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>( u_i = \max_{i-r \leq k \leq i+r} q_k )</td>
<td></td>
</tr>
<tr>
<td>( \ell_i = \min_{i-r \leq k \leq i+r} q_k )</td>
<td></td>
</tr>
<tr>
<td>( LB_{Keogh\text{EQ}}(Q, C) = LB_{Keogh\text{EC}}(C, Q) )</td>
<td>( O(n) )</td>
</tr>
</tbody>
</table>
Lower bounding

\[ \text{best_so_far} = \min(\text{DTW}(Q, C), \text{bsf}_{\text{prev}}) \]

<table>
<thead>
<tr>
<th>Lower Bound</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{LB}_{\text{Kim}}(Q, C) = (q_1 - c_1)^2 + (q_n - c_n)^2 )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>( \text{LB}<em>{\text{Keogh}} \text{EC}(Q, C) = \sum</em>{i=1}^{n} \begin{cases} (c_i - u_i)^2, &amp; c_i &gt; u_i \ (c_i - \ell_i)^2, &amp; c_i &lt; \ell_i \ 0, &amp; \text{otherwise} \end{cases} )</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>( \text{LB}<em>{\text{Keogh}} \text{EQ}(Q, C) = \text{LB}</em>{\text{Keogh}} \text{EC}(C, Q) )</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>( \text{DTW}(Q, C) )</td>
<td>( O(n^2) )</td>
</tr>
</tbody>
</table>
Intel Xeon Phi many-core systems

• **Up to 72 compute cores**, each core provides
  – relatively weak computing power than the Intel Xeon CPU core
  – 512-bit wide **vector processing unit (VPU)** and **SIMD instructions**

• Based on **x86 architecture** and supports IDE and programming models for Intel CPUs
PhiBestMatch: novel algorithm for Intel MIC

Align subsequences

Z-normalize subsequences

Compute LBs for subsequences

... Z-normalize subsequences

Compute LBs for subsequences

Collect promising subsequences as matrix

Prune dissimilar subsequences by LBs

Collect best-so-far by DTW

Find minimum among all best-so-far

Prune dissimilar subsequences by LBs
Subsequence matrix

\[ T \in \mathbb{R}^m \]

Data alignment: \((n + pad) : width_{VPU}\)

\[ Q \in \mathbb{R}^n \]

\[ S \in \mathbb{R}^{N \times (n + pad)} \]
Matrix of Lower Bounds

\[ S \in \mathbb{R}^{N \times (n+\text{pad})} \]

\[ LB \in \mathbb{R}^{N \times lb_{\text{max}}} \]

<table>
<thead>
<tr>
<th>( LB_{\text{Kim}FL}(Q, S_i) )</th>
<th>( LB_{\text{Keogh}EC}(Q, S_i) )</th>
<th>( LB_{\text{Keogh}EQ}(S_i, Q) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( LB_{\text{Kim}FL} )</td>
<td>( LB_{\text{Keogh}EC} )</td>
<td>( LB_{\text{Keogh}EQ} )</td>
</tr>
<tr>
<td>( LB_{\text{Kim}FL} )</td>
<td>( LB_{\text{Keogh}EC} )</td>
<td>( LB_{\text{Keogh}EQ} )</td>
</tr>
<tr>
<td>( LB_{\text{Kim}FL} )</td>
<td>( LB_{\text{Keogh}EC} )</td>
<td>( LB_{\text{Keogh}EQ} )</td>
</tr>
<tr>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
</tr>
</tbody>
</table>

Parallelizable 😊
Auto-vectorizable 😊

An Efficient Subsequence Similarity Search on Modern Intel MIC Processors for Data Intensive Applications

12.10.2018

16/30
### Bitmap matrix

$$LB \in \mathbb{R}^{N \times lb_{max}}$$

<table>
<thead>
<tr>
<th>$LB_{KimFL}(Q, S_i)$</th>
<th>$LB_{KeoghEC}(Q, S_i)$</th>
<th>$LB_{KeoghEQ}(S_i, Q)$</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Graph" /></td>
<td><img src="image2" alt="Graph" /></td>
<td><img src="image3" alt="Graph" /></td>
</tr>
<tr>
<td><img src="image4" alt="Graph" /></td>
<td><img src="image5" alt="Graph" /></td>
<td><img src="image6" alt="Graph" /></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

$$BM \in \mathbb{R}^{N \times lb_{max}}$$

<table>
<thead>
<tr>
<th>$LB_{KimFL}(Q, S_i) &lt; bsf$</th>
<th>$LB_{KeoghEC}(Q, S_i) &lt; bsf$</th>
<th>$LB_{KeoghEQ}(S_i, Q) &lt; bsf$</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Parallelizable 😊
Auto-vectorizable 😊
Pruning clearly dissimilar candidates

\[
S \in \mathbb{R}^{N \times (n+pad)}
\]

Parallelizable 😊
Auto-vectorizable 😊
Matrix of promising candidates

\[ S \in \mathbb{R}^{N \times (n + pad)} \]

\[ C \in \mathbb{R}^{(k \cdot p) \times (n + pad)} \]

Balancing of threads load:

\[ p \ll m \] is number of threads

\[ k \] is parameter (e.g. 10, 100, 1000, ...)

Segment of \( k \) candidates

Matrix of \( p \) segments

Serial 😞
Computing best-so-far

\[ C \in \mathbb{R}^{(k \cdot p) \times (n + \text{pad})} \]

- Parallelizable
- Not auto-vectorizable

\[ bsf_1 \leftarrow DTW(C_1, Q) \]
\[ bsf_2 \leftarrow DTW(C_2, Q) \]
\[ bsf_{k \cdot p} \leftarrow DTW(C_{k \cdot p}, Q) \]

\[ bsf \leftarrow \min (bsf_{\text{prev}}, \min_{1 \leq i \leq k \cdot p} bsf_i) \]

Partly auto-vectorizable
Computing DTW

double DTW(a: array [1..m], b: array [1..m], r: int) {
    cost := array [1..m]
    cost_prev := array [1..m]

    for i := 1 to m
        cost[i] = infinity
        cost_prev[i] = infinity

    cost_prev[1] = dist(a[1], b[1])

    for j := max(2, i-r) to min(m, i+r)
        cost_prev[j] := cost_prev[j-1] + dist(a[1], b[j])

    for i := 2 to m
        for j := max(1, i-r) to min(m, i+r)
            c := d(a[i], b[j])
            cost[j] := c + min(cost[j-1], cost_prev[j-1], cost_prev[j])
        swap(cost, cost_prev)

    return cost_prev[m]
}
double DTW(a: array [1..m], b: array [1..m], r: int) {
    cost := array [1..m]
    cost_prev := array [1..m]
    for i := 1 to m
        cost[i] = infinity
        cost_prev[i] = infinity
    cost_prev[1] = dist(a[1], b[1])
    for j := max(2, i-r) to min(m, i+r)
        cost_prev[j] := cost_prev[j-1] + dist(a[1], b[j])
    for i := 2 to m
        for j := max(1, i-r) to min(m, i+r)
            cost[j] = min(cost_prev[j-1], cost_prev[j])
        for j := max(1, i-r) to min(m, i+r)
            c := dist(a[i], b[j])
            cost[j] := c + min(cost[j-1], cost[j])
    swap(cost, cost_prev)
    return cost_prev[m]
}
Experiments

• Hardware

<table>
<thead>
<tr>
<th>Feature</th>
<th>Device</th>
<th>2×Intel Xeon X5680</th>
<th>Intel Xeon Phi SE10X</th>
</tr>
</thead>
<tbody>
<tr>
<td># of physical cores</td>
<td>2×6</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>Hyper threading factor</td>
<td>2</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td># of logical cores</td>
<td>24</td>
<td>244</td>
<td></td>
</tr>
<tr>
<td>Frequency, GHz</td>
<td>3.33</td>
<td>1.1</td>
<td></td>
</tr>
<tr>
<td>Peak performance, TFLOPS</td>
<td>0.371</td>
<td>1.076</td>
<td></td>
</tr>
</tbody>
</table>

• Datasets
  – Random walk (synthetic): $|T|=10^6$
  – EPG (entomology): $|T|=2.5\times10^5$

• Measures
  – Performance, sec.
  – Speedup, $s(p) = \frac{t_1}{t_p}$
  – Parallel efficiency, $e(p) = \frac{s(p)}{p}$
Performance: Random Walk dataset

| Q | 128 |

UCR-DTW, Xeon CPU, serial
PhiBestMatch, 2×Xeon CPU, 24 threads
PhiBestMatch, Phi, 240 threads
PhiBestMatch, Phi, 60 threads

Warping constraint, r (as a fraction of |Q|)
An Efficient Subsequence Similarity Search on Modern Intel MIC Processors for Data Intensive Applications

Performance: EPG dataset

| Q | = 368

Run time (log scale), sec

Warping constraint, r (as a fraction of | Q |)

UCR-DTW, Xeon CPU, serial
PhiBestMatch, 2×Xeon CPU, 24 threads
PhiBestMatch, Phi, 240 threads
PhiBestMatch, Phi, 60 threads

\[ \frac{25}{30} \]
Performance: impact of query length

**EPG dataset**

$r=1.0$

**Comparison of Run Time with Different Query Lengths and Parallelisations**

- **UCR-DTW, Xeon CPU, serial**: 96.23 seconds
- **PhiBestMatch, 2×Xeon CPU, 24 threads**: 293.59 seconds
- **PhiBestMatch, Phi, 240 threads**: 112.20 seconds
- **PhiBestMatch, Phi, 60 threads**: 66.97 seconds
Scalability: Random Walk dataset

$r$ denotes a fraction of $|Q|$. 
Scalability: EPG dataset

$r$ denotes a fraction of $|Q|$.
Further work: cluster version
Conclusions

• We have proposed a novel parallel algorithm for subsequence similarity search for Intel many-core systems
• We have conducted experiments on synthetic and real data that show good scalability of our algorithm

Thank you for paying attention! Questions?
Mikhail Zymbler
mzym@susu.ru