Partition Around Medoids Clustering on the Intel Xeon Phi Many-core Coprocessor

Timofey Rechkalov

South Ural State University, Russia

This work was financially supported by the Ministry of education and science of the Russian Federation contract No. 14.574.21.0035.
Intel Xeon Phi
Partitioning Clustering
PAM properties

• **PAM algorithm (Partitioning Around Medoids)** – partitioning clustering algorithm which selects cluster centers among clustered objects

• Such objects called *medoids*

• Iteration time complexity is $O(k(n-k)^2)$, where
  • $n$ is the number of clustered objects
  • $k$ is the number of clusters
Objective function

• Objective function

\[ E = \sum_{j=1}^{n} \min_{1 \leq i \leq k} \rho(c_i, o_j). \]

where \( c_i \) is the medoid, \( o_j \) is the clustered object, \( \rho \) is the distance metric
PAM pseudocode

Input: Set of objects $O$, number of clusters $k$
Output: Set of clusters $C$

1. Initialize $C$; // BUILD phase
2. repeat // SWAP phase
3. Find best swapping estimation $T_{min}$;
4. Swap $c_{min}$ and $o_{min}$, determined by $T_{min}$;
5. until $T_{min} < 0;$
Calculating distance matrix

<table>
<thead>
<tr>
<th></th>
<th>$o_1$</th>
<th>$o_2$</th>
<th>$o_3$</th>
<th>...</th>
<th>$o_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$o_1$</td>
<td>$\rho(o_1, o_1)$</td>
<td>$\rho(o_1, o_2)$</td>
<td>$\rho(o_1, o_3)$</td>
<td>...</td>
<td>$\rho(o_1, o_n)$</td>
</tr>
<tr>
<td>$o_2$</td>
<td>$\rho(o_2, o_1)$</td>
<td>$\rho(o_2, o_2)$</td>
<td>$\rho(o_2, o_3)$</td>
<td>...</td>
<td>$\rho(o_2, o_n)$</td>
</tr>
<tr>
<td>$o_3$</td>
<td>$\rho(o_3, o_1)$</td>
<td>$\rho(o_3, o_2)$</td>
<td>$\rho(o_3, o_3)$</td>
<td>...</td>
<td>$\rho(o_3, o_n)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$o_n$</td>
<td>$\rho(o_n, o_1)$</td>
<td>$\rho(o_n, o_2)$</td>
<td>$\rho(o_n, o_3)$</td>
<td>...</td>
<td>$\rho(o_n, o_n)$</td>
</tr>
</tbody>
</table>
BUILD phase

$k=3$

\[ E = \infty \quad E = 2,250 \quad E = 1,578 \quad E = 1,014 \]

Time complexity \( O(kn^2) \)
SWAP phase

\[ E = 1,014 \quad E = 0,865 \]

Time complexity \( O(k(n - k)^2) \) per iteration
Used Optimizations

• Parallelizing with OpenMP
• Loops with arithmetic operations were reorganized for vectorized execution
  – Data consists of 32 element blocks
• Tiling for better locality and cache performance
PAM implementation

Input: Set of objects $O$, number of clusters $k$
Output: Set of clusters $C$

1. $M \leftarrow \text{PrepareDistanceMatrix}(O)$;
2. $C \leftarrow \text{BuildMedoids}(M)$; // BUILD phase
3. repeat // SWAP phase
4. $T_{\text{min}} \leftarrow \text{FindBestSwap}(M, C)$;
5. Swap $c_{\text{min}}$ and $o_{\text{min}}$, determined by $T_{\text{min}}$;
6. until $T_{\text{min}} < 0$;
void prepareDistanceMatrix(const float* rowData, const float* colData,
float* distances, const int n, const int pointWidth)
{
    const int vecLen = 32;
    #pragma omp parallel
    {
        float point[pointWidth] __attribute__((aligned(64)));
        float result[vecLen] __attribute__((aligned(64)));
        #pragma omp for
        for(int i=0; i<n; ++i)
        {
            point[] = rowData[i*pointWidth:pointWidth];
            for(int ii = 0; ii < n; ii += vecLen)
            {
                result[] = 0;
                for(int j=0; j < pointWidth; ++j)
                {
                    const float* restrict point2 = colData+ii*pointWidth;
                    result[] += (point[j] - point2[j*vecLen:vecLen])*
                    (point[j] - point2[j*vecLen:vecLen]);
                }
            }
            distances[i*n+ii:vecLen] = sqrtf(result[]);
        }
    }
}
Low level property

- $C_{jih}$ — is a contribution of non selected object $o_j$ to the objective function changing in case of swapping of medoid $c_i$ and non-medoid $o_h$

Вход : $o_j, c_i, o_h, d_j, s_j$
Выход: $C_{jih}$
if $\rho(o_j, c_i) > d_j$ and $\rho(o_j, o_h) > d_j$ then
  $C_{jih} \leftarrow 0$
else if $\rho(o_j, c_i) = d_j$ then
  if $\rho(o_j, o_h) < s_j$ then
    $C_{jih} \leftarrow \rho(o_j, o_h) - d_j$
  else
    $C_{jih} \leftarrow s_j - d_j$
else if $\rho(o_j, o_h) < d_j$ then
  $C_{jih} \leftarrow \rho(o_j, o_h) - d_j$
end
PAM-1 and PAM-2

• Calculating of $C_{jih}$ is hard to vectorize so we implemented two versions of PAM
• PAM-1 uses «forced» vectorization
  – #pragma vector always
• PAM-2 stores more auxiliary data which simplifies vectorization but degrade cache effectiveness
• PAM-1 and PAM-2 have differences in SWAP phase only
# PAM-1 and PAM-2 (2)

<table>
<thead>
<tr>
<th>PAM-1</th>
<th>PAM-2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Auxiliary data size</strong></td>
<td><strong>nk</strong></td>
</tr>
<tr>
<td>$3n$</td>
<td></td>
</tr>
</tbody>
</table>

## Logical condition for $C_{ijh}$ calculating

```java
if (val1 > val2){
    delta += val2 - val1;
} else if (val3 == const1) {
    if (val2 < val4){
        delta += val2 - val1;
    } else {
        delta += val4 - val1;
    }
} else {
    delta += val4 - val1;
}
```

```java
if (val1 > val2){
    delta += val2 - val1;
}
```
Experimental evaluation

• Hardware
  – Intel Xeon Phi 60 cores
  – Intel Xeon 12 cores

• Parameters
  – Data type: float
  – Intel Xeon Phi mode: offload
  – Maximum clustered objects number (caused by Intel Xeon Phi memory size): 40 thousand

• Purpose
  – Compare work time of PAM algorithm on CPU and Intel Xeon Phi
## Dataset properties

<table>
<thead>
<tr>
<th>Набор данных</th>
<th>$p$</th>
<th>$k$</th>
<th>$n, \times 2^{10}$</th>
<th>Max data size, Mb</th>
<th>Time to transfer to coprocessor, sec</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>min</td>
<td>max</td>
<td></td>
</tr>
<tr>
<td>FCS Human</td>
<td>423</td>
<td>10</td>
<td>2</td>
<td>18</td>
<td>29.74</td>
</tr>
<tr>
<td>Corel Image Histogram</td>
<td>32</td>
<td>15</td>
<td>5</td>
<td>35</td>
<td>4.38</td>
</tr>
<tr>
<td>MixSim</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>35</td>
<td>0.68</td>
</tr>
<tr>
<td>Letter Recognition</td>
<td>16</td>
<td>26</td>
<td>2</td>
<td>18</td>
<td>1.13</td>
</tr>
</tbody>
</table>

- $p$ – size of real-valued tuple which describes clustering object
- $k$ – the number of clusters
- $n$ – the number of clustering objects
FCS Human evaluation

- PAM-1, Intel Xeon
- PAM-1, Intel Xeon Phi
- PAM-2, Intel Xeon
- PAM-2, Intel Xeon Phi
- PAM-1, single thread, Intel Xeon
- Prepare Distance Matrix, Intel Xeon
- Prepare Distance Matrix, Intel Xeon Phi

Execution time, sec

Number of objects, x1024
Corel Image Histogram evaluation

[Graph showing execution time vs. number of objects for different configurations]

21.12.2015
MixSim evaluation

![Graph showing execution time vs. number of objects for different configurations. The graph plots the execution time in seconds on the y-axis against the number of objects (x1024) on the x-axis. Different markers and line types represent various configurations, such as PAM-1 and PAM-2 with different processors and thread settings.](image-url)
Letter Recognition evaluation
Intermediate summary

• PAM-2 is a better implementation for the Intel Xeon Phi. This is confirmed by experiments. In all tests PAM-2 is twice better on Intel Xeon Phi.

• PAM-1 is the best with the Intel Xeon only once. In other tests there is no significant difference.

• To investigate this fact deeper we made more experiments to see contribution of every PAM subalgorithm.
MixSim detailed evaluation

![Graphs showing execution time vs. number of objects for different phases and iterations of MixSim, with Intel Xeon and Intel Xeon Phi processors.](image)
Letter Recognition detailed evaluation

![Graph showing execution time vs. number of objects for different processes and hardware configurations.](image)

Number of objects, $x1024$
Conclusion

• The paper has described a parallel version of Partitioning Around Medoids clustering algorithm for the Intel Xeon Phi many-core coprocessor
  – OpenMP
  – Vectorization
  – Tiling
• Experiments show that PAM performance depends on clustered data nature
• Distance Matrix calculation and BUILD phase perform better on Intel Xeon Phi
• SWAP phase performs better on Intel Xeon

21.12.2015