Partitioned Local Depth Algorithm (PaLD) is a cluster algorithm discovered by Dr. Kenneth S. Berenhaut at Wake Forest University Mathematics and Statistics Department. The idea of this algorithm is to find the probability that how likeliy points within a range will support a given point. For example, given point i and j we get their distance as $d(i,j)$, then we create a group of points that are closer to either point i or point j comparing with the distance between point i and point j. We call this group local conflict focus. Each pari of points will have their own local conflict focus and within each local conflict we calcuate the probability that how likeliy a point in the local conflict focus will support that pair of points. The algorithm is shown below

Algorithm 1 Partitioned Local Depth Original Algorithm

1: ***Input:*** distance matrix of the data set D where $D_{i,j} = d(i,j)$, and $D \in \mathbb{R}^{n\times n}$
2: ***Output:*** cohension matrix C of the data set D
3: **procedure** OrigPaLD(D)
4: $C \leftarrow \boldsymbol{0}$
5: **for** `i = 1 to n` **do**
6: **for** `j = 1 to n` satisfying $j \neq i$ **do**
7: $U_{i,j} \leftarrow \{k | d(k,i) \leq d(j,i)$ *or* $d(k,j) \leq d(j,i)\}$
8: **for** l in $U_{i,j}$ **do**
9: **if** $d(l,i) < d(l,j)$ **then**
10: $C_{i,l} \leftarrow C_{i,l} + \frac{1}{size(U_{i,j})}$
11: **if** $d(l,i) > d(l,j)$ **then**
12: $C_{j,l} \leftarrow C_{j,l} + \frac{1}{size(U_{i,j})}$
13: **if** $d(l,i) = d(l,j)$ **then**
14: $C_{i,l} \leftarrow C_{i,l} + \frac{1}{2*size(U_{i,j})}$
15: $C_{j,l} \leftarrow C_{j,l} + \frac{1}{2*size(U_{i,j})}$
16: $C \leftarrow \frac{C}{n-1}$
17: **return** C

There are three nested loops and one more loop in line 7 and each loop has $\mathcal{O}(n)$ except the loop in line 8 which is related to the size of the local conflict focus. Therefore, this algorithm is at least $\mathcal{O}(n^3)$. After a careful discuss we find that the size of a local conflict focus is about linear with n^2 (See $U_{x,y}$ size analysis.), which means the loop in line 8 has $\mathcal{O}(n^2)$ flops. So the overall computational cost of original PaLD algorithm is $\mathcal{O}(n^3)$ and the running time of PaLD algorithm will be large if we use the original algorithm given a large data set.

In order to reduce the running time of PaLD, here we apply a modifed optimal blocked matrix multiplication algorithm to PaLD. Before we dive into the modified algorithm, let's see how the optimal blocked matrix multiplication algorithm works.

Algorithm 2 Optimal Blocked Matrix Multiplication Algorithm

Input: A, B are $n \times n$ matrices, b divides n, $b \leq \sqrt{M+1} - 1$, A is partitioned into $b \times n$ block rows so that $A_{I,:} = A((I-1)b+1 : Ib, 1 : n)$, B is partitioned into $n \times b$ block columns so that $B_{:,J} = B(1 : n, (J-1)b+1 : Jb)$, C is partitioned into $b \times b$ blocks so that $C_{IJ} = C((I-1)b+1 : Ib, (J-1)b+1 : Jb)$.
Ensure: C = C + AB is $n \times n$ matrix
Output: C, where C is the output of A times B
procedure MATMUL-BLOCKED-BETTER(C, A, B, b)
    $C \leftarrow \boldsymbol{0}$
    for `I = 1 to n/b` **do**
        for `J = 1 to n/b` **do**
            Read C_{IJ} inot fast memory
            for `k = 1 to n` **do**
                Read $A_{I,:}(:,k)$ and $B_{:,J}(k,:)$ inot fast memory
                for `i = 1 to b` **do**
                    for `j = 1 to b` **do**
                        $C_{IJ}(i,j) + 1 = A_{I,:}(i,k)B_{:,J}(k,j)$
                Write C_{IJ} to slow memory
    return C

The optimal blocked matrix multiplication algorithm take advantage of the blocks that are stored in the fast memory and reuse those blocks as much as possible to save the data movement time between fast memroy and slow memory. We will use the same strategy to reduce the running time of PaLD algorithm. But the challenge in this converting is that PaLD use comparsion instead of multiplication and the comparsion procedure is not exactly the same as matrix multiplication. What's more, we also need to compute the local conflict foci in this algorithm.

Based on the optimal blocked matrix multiplication algorithm, we can embeded the calculation of local conflict foci within the calculation of cohension matrix C. The major challenge of this algorithm is to figure out how to block the conflict foci and distance matrix to get the cohension matrix.

Algorithm 3 Optimal Blocked PaLD Algorithm

Input: distance matrix of the data set D where $D_{i,j} = d(i,j)$, and $D \in \mathbb{R}^{n\times n}$, b divides n, $b \leq \sqrt{M+1}-1$
Ensure: D is a symmetric matrix, and triangle inequality is satisified in the distance matrix
Output: cohension matrix C of the data set D
procedure PaLD-Opt(D, b)
  $C \leftarrow \boldsymbol{0}$
  for `I = 1 to n/b` **do**
    Ib = I:min(I+b-1,n)
    for `J = 1 to n/b` **do**
      Jb = J:min(J+b-1,n)
      Read $D_{Ib,Jb}$ block inot fast memory
      $U_{IJ} \leftarrow \boldsymbol{0}$, where $\boldsymbol{0} \in \mathbb{R}^{min(b,n-I+1)\times min(b,n-J+1)}$
      if I=J **then**
        $U_{IJ} = U_{IJ} + diag(Inf * \boldsymbol{1})$, where $\boldsymbol{1} \in \mathbb{R}^{min(b,n-I+1)\times 1}$
        for `k = 1 to n` **do**
          $C(Ib,k) = C(Ib,k) + \big((D(Ib,k) < D(Jb,k)^T)\&(D(Ib,k) <= D_{Ib,Jb})\big)./U_{IJ}$
          $C(Ib,k) = C(Ib,k) + 0.5 * \big((D(Ib,k) = D(Jb,k)^T)\&(D(Ib,k) <= D_{Ib,Jb})\big)./U_{IJ}$
      else
        for `k = 1 to n` **do**
          determine contributions to C(Ib,k)
          $C(Ib,k) = C(Ib,k) + \big((D(Ib,k) < D(Jb,k)^T)\&(D(Ib,k) <= D_{Ib,Jb})\big)./U_{IJ}$
          $C(Ib,k) = C(Ib,k) + 0.5 * \big((D(Ib,k) = D(Jb,k)^T)\&(D(Ib,k) <= D_{Ib,Jb})\big)./U_{IJ}$
          determine contributions to C(Jb,k)
          $C(Jb,k) = C(Jb,k) + \big((D(Ib,k) < D(Jb,k)^T)\&(D(Ib,k) <= D_{Ib,Jb})\big)./U_{IJ}$
          $C(Jb,k) = C(Jb,k) + 0.5 * \big((D(Ib,k) = D(Jb,k)^T)\&(D(Ib,k) <= D_{Ib,Jb})\big)./U_{IJ}$
      Write C_{IJ} to slow memory
  return $C \leftarrow \frac{C}{n-1}$

As we can see from the above algorithm, we successfully reduced the data movement between fast memroy (cache) and slow memroy (disk). The overall saved running time of this algorithm comparing with the original PaLD algorithm is $\mathcal{O}(M)$ given the size of cache as M.

This research project is mentored by Dr. Grey Ballard at Wake Forest University Computer Science Department. We are currently working on a parallelized version of PaLD with Yixin Zhang as our new project member.