SUPPLEMENTARY DATA

Massive parallelization pattern for obtaining canonical m-mers (with or without signature restriction) through hybrid granularity and tiles.

Algorithm 1: Obtaining the first canonical m-mer by tile (High granularity parallelism)

---

ts -> Tile size
m -> m-mer size
x -> x index of the thread
RSK[] -> Array in local/shared memory where reads are stored and canonical m-mers are rewritten
MT[] -> Array in local/shared memory where the "current" m-mer of each tile is stored.
RCMT[] -> Array in local/shared memory where the reverse complement of the "current" m-mer of each tile is stored.

Function block executed by each of the m threads of each tile of each read
{

- ***idt = x/m***

  Through an integer division, each thread obtains the index of the tile to which it corresponds

- ***b = RSK [idt*ts + x%m]***

  Each thread of the tile copies a base (its representation "base 4") from local/shared memory to private memory.

- ***MT[idt] = OR Atómico (MT[idt] , (b << (m-(x%m)-1)*2))***
  RCMT[idt] = OR Atómico (RCMT[idt] , (((~b) & 3) << ((x%m)*2)))

  Through logical negations, shifts and atomic operations OR, each of the threads of the tile processes the base that it has read to implement the equations that postulates that given the m-mer = b_0, b_1, ..., b_{m-1} its decimal value and the one of its inverse complement will be:
  $M\text{-}mer_{10} = b_0<<(2m-2) + b_1<<(2m-4) +...+ b_{m-1}<<(0)$
  $Rcm\text{-}mer_{10} = \sim b_{m-1}<<(2m-2) + \ldots + \sim b_1<<(2) + \sim b_0<<(0)$.

}

Local synchronization

Function block executed by a thread in each tile of each read
{

- ***RSK[idt*ts + m -1] = min/sig (MT[idt], RCMT[idt])***

  The thread assigned to the tile applies the function min[1] or sig[2] to the decimal values (m-mer and its inverse complement) and overwrites the result in the read (input) in the position corresponding to the last base of the m-mer computed.

  [1] If this algorithm is used to find the canonical m-mers of a set of readings, the min function is applied (it obtains the smaller of the two inputs).
  [2] If this algorithm is used to find the super k-mers of a set of reads under the signatures criterion, the sig function is applied which before taking the minimum of the values applies the constraints of a signature (prohibited AAA or ACA at the start and prohibited AA in any location except at the start). If neither the m-mer nor its inverse complement comply with the constraints of a signature, the function returns the decimal value equivalent to the largest decimal value that an m-mer with that length can have, incremented by one.

```
}

Local synchronization
```

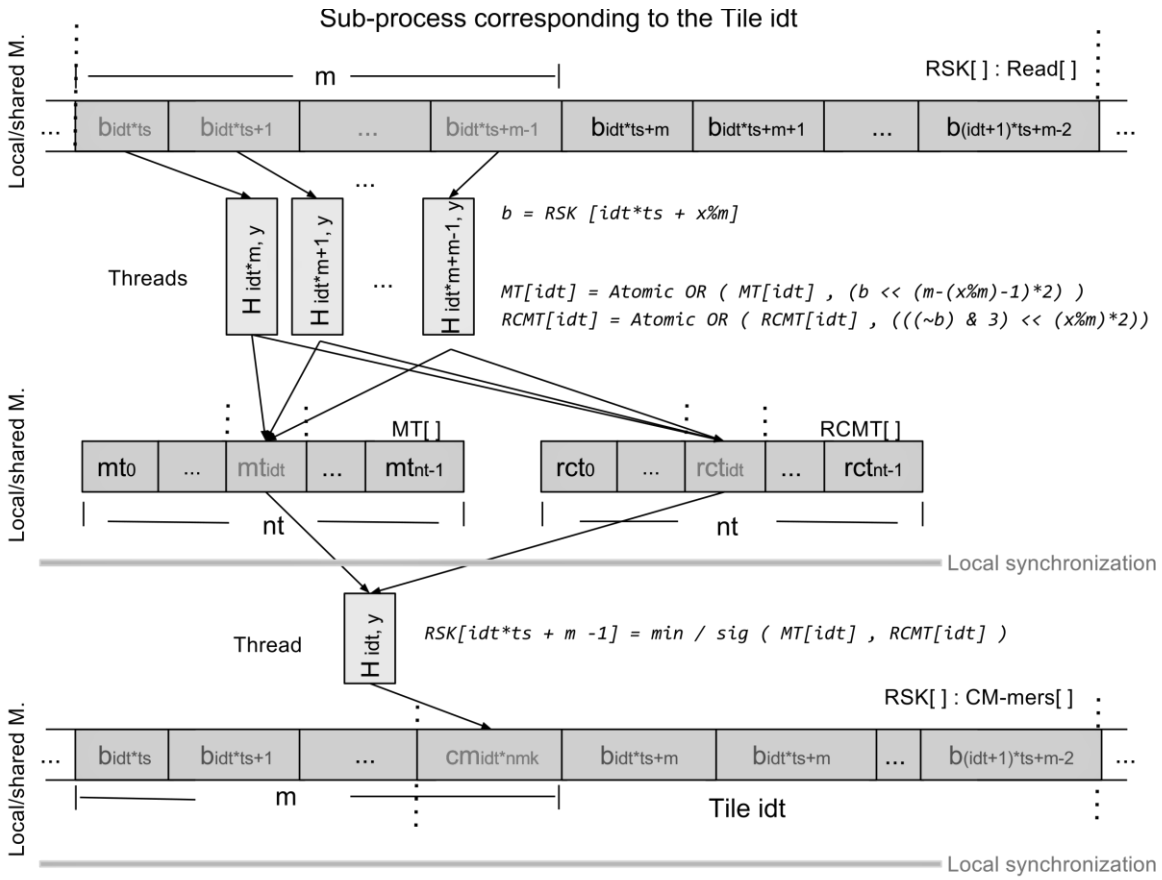

Figure 1. Illustration of the algorithm for obtaining the first canonical m-mer by tile (high granularity parallelism). The conventions, acronyms and nomenclature in general are the same as those used in Algorithm 1.

Algorithm 2: Obtaining the rest of canonical m-mers by tile (Moderate granularity parallelism)

```
idt -> Index of the tile calculated for each thread in the previous algorithm
ts -> Tile size
m -> m-mer size
x -> x index of the thread
RSK[] -> Array in local/shared memory where reads are stored and canonical m-mers are rewritten
MT[] -> Array in local/shared memory where the "current" m-mer of each tile is stored.
RCMT[] -> Array in local/shared memory where the reverse complement of the "current" m-mer of each
tile is stored.

Function block executed by a thread in each tile of each read
{
```

- ***c = MT[idt]***
  d = RCMT[idt]

  The thread assigned to the tile reads the decimal value of the first m-mer of the tile and that of its inverse complement (from local/shared memory to private memory).

  Function block that is repeated for the conditions (z = 0; z <(ts-1); z ++) and breaks if it tries to read a base in a position outside of the reading.
  {
  - ***b = RSK [idt*ts + m + z]***

    The thread assigned to the tile copies from the local/shared memory to the private memory only the last base (its representation "base 4") of the m-mer to be calculated.

  - ***c = ((c & (2^(2m-2)-1)) << 2)|b***
    d = ((d>>2) & (2^(2m-2)-1)) | (((~b) & 3)<<((m-1)*2))

    Using only the numeric value (base 4) of the last base of the m-mer to be calculated (b_n_{m-1}) and the decimal value of the previous m-mer and that of its inverse complement (which are already in the private memory), the thread assigned to the tile implements the equations that postulate that given the previous m-mer; m-mer_o = b_o_0 ,b_o_1 ,..., b_o_{m-1} the decimal value of the next m-mer and that of its inverse complement will be:
    $New\_m\text{-}mer_{10} = (M\text{-}mer_{10} << 2) + new\_b_{m-1}$
    $New\_rcm\text{-}mer_{10} = (Rcm\text{-}mer_{10} >> 2) + (new\_b_{m-1} << (2m-2))$

  - ***RSK[idt*ts + m + z] = min / sig (c, d)***

    The thread assigned to the tile applies the function min or sig to the decimal values of the m-mer and the one of its inverse complement (according to the same conditions exposed in the algorithm of the first m-mer per tile) and overwrites the result in the RSK[] array (input read) in the position corresponding to the last base of the m-mer that has just been computed (Overwrite strategy to avoid deleting bases that may be required by other threads)
  }
}

Local synchronization

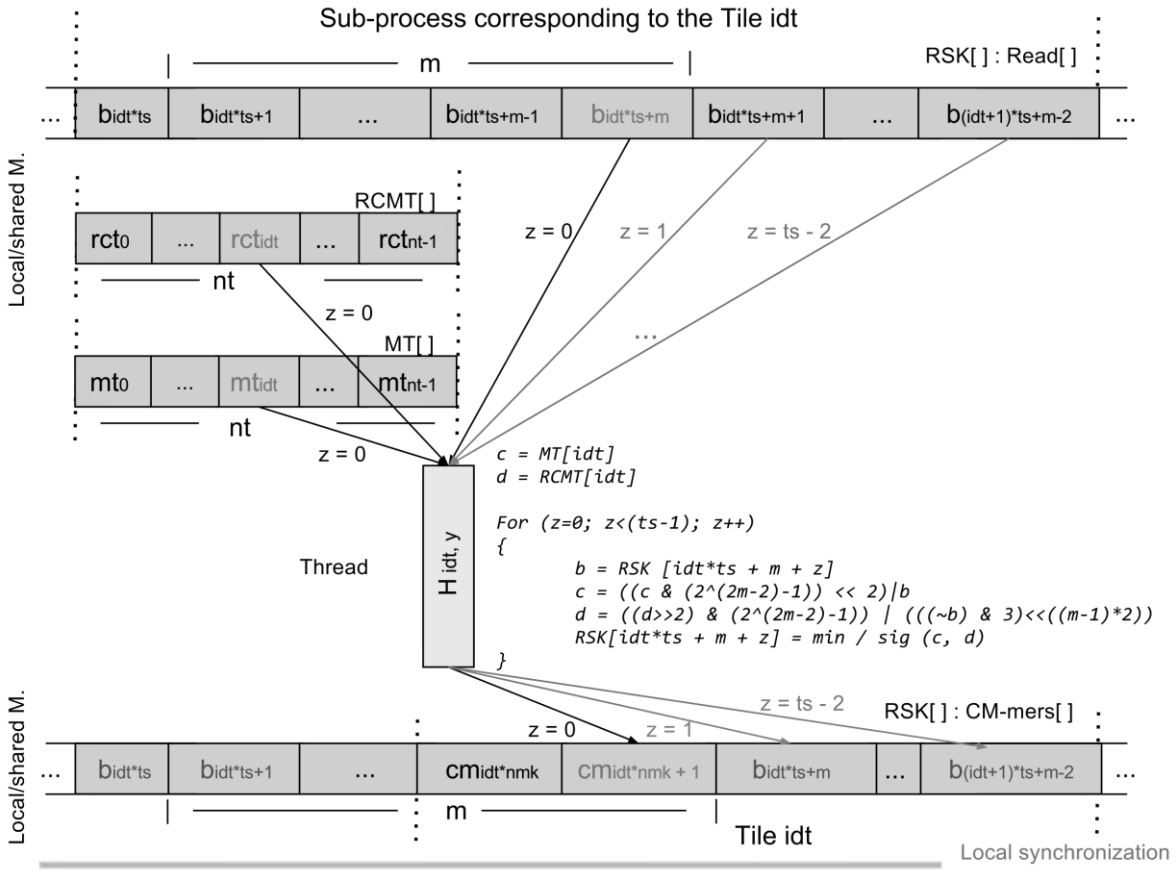

Figure 2. Illustration of the algorithm for obtaining the rest of m-mers of each tile. (Parallelism of moderate granularity). The conventions, acronyms and nomenclature in general are the same used in Algorithm 2

Parallelization pattern for the search of super k-mers through a window of sliding by jumps and mixed and adaptive reduction

Algorithm 3: Evaluation of the zone without reference seed (Mixed and adaptive reduction pattern)

bs -> Reduction block size.
m -> m-mer size
cm -> Decimal value of a canonical m-mer
nmk -> Number of m-mers per k-mers
x -> x index of the thread
p -> Position of the element of the vector Cm-mer [] that the thread is processing.
Mp -> Position of the "current" seed (minimizer or signature)
RSK[] -> Array in local/shared memory where the CM-mer[] vector is stored, which contains the canonical m-mers.
MT[] -> Array in local/shared memory where the minimum of each block is stored.
nb -> Number of reduction blocks (if k <= 36 -> nb = 6; if k <= 64 -> nb = 8; if k <= 100 -> nb = 10; if k > 100 -> nb = 12)
Minimizer -> Decimal value of a minimizer

Function block executed by nmk threads per read
{
Function block that executes if p <sz (Those threads whose element processed in the previous zone is in a position lower than the beginning of the current zone - caterpillar type displacement)
{

- ***p = p + nmk***

  The position of the element of the vector Cm-mer [] that each thread will read and will process (caterpillar displacement) is updated

- ***cm = RSK[p + m - 1]***

  Each thread reads its new element to be processed from the vector Cm-mer[] (the offset m-1 is due to the strategy used to rewrite Cm-mer[] in the RSK[] array where the bases of the read were initially stored)

}

- ***MT[(x%bs] = Min atómico (MT[x%bs], cm)***

  By means of a modular operation, the threads belonging to the same block perform a "minimum" atomic operation with the same location of the vector MT []. In this way each block is reduced to its smallest element.

}

Local synchronization

Function block executed by nb threads per read
{

- ***Minimizer = Min atómico (MT[x], Minimizer)***

  Through an atomic operation each of the blocks is reduced to its smallest element

}

Local synchronization

Function block executed by nmk threads per read
{
Those threads whose value cm equals the seed execute the following function
{

- ***Mp = Max atómico (Mp, p)***

Finally, the position of the minimum is obtained. If in the evaluated area more than one minimum was found with the same value, the one that has a greater position is chosen as seed (to favor the construction of more extensive super k-mers)

}

}

Local synchronization

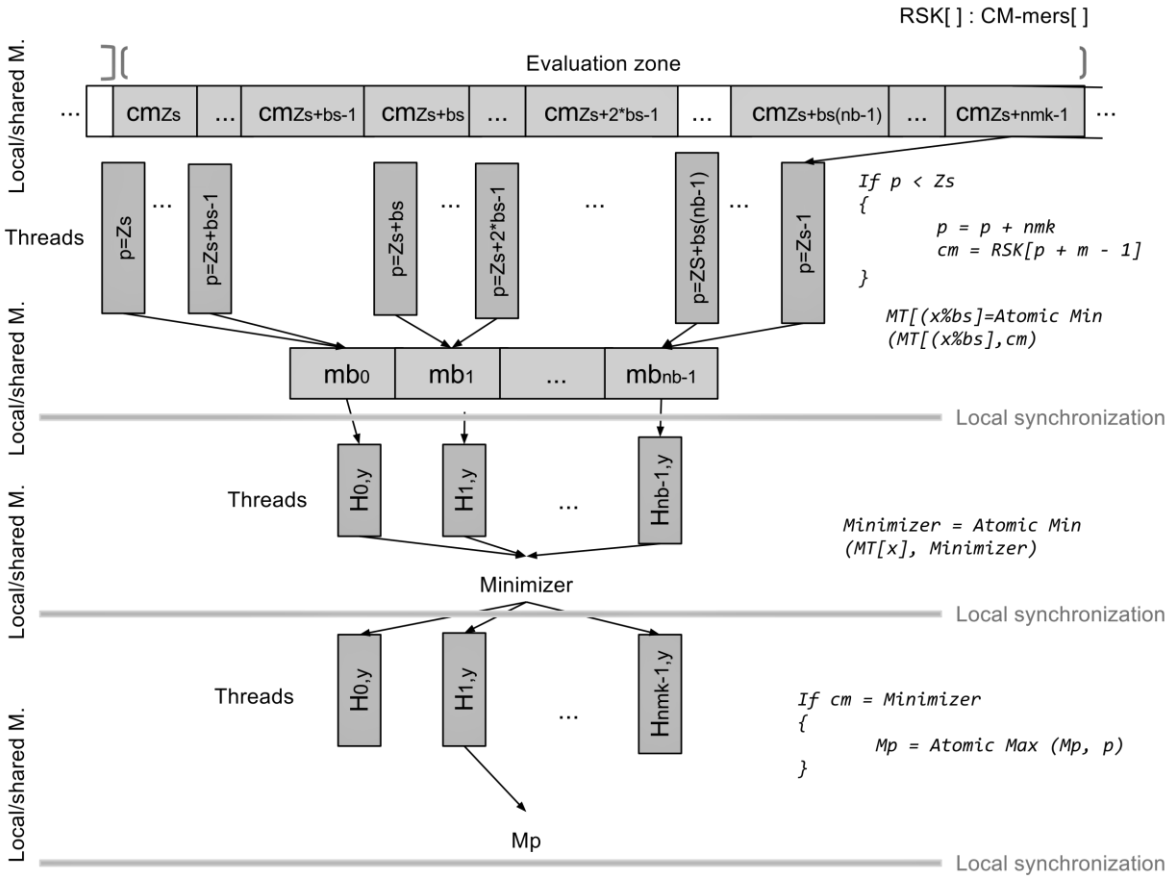

Figure 3. Illustration of the algorithm to evaluation of the zone without reference seed (Mixed and adaptive reduction pattern). The conventions, acronyms and nomenclature in general are the same used in Algorithm 3

Algorithm 4: Evaluation of zone with reference seed
(Nearest smaller search algorithm)

```
m -> m-mer size
nmk -> Number of m-mers per k-mers
cm -> Decimal value of a canonical m-mer
p -> Position of the  element of the vector Cm-mer [] that the thread is processing.
Mp -> Position of the "current" seed (minimizer or signature)
NMp -> Position of the "new" seed (minimizer or signature)
RSK[] -> Array in local/shared memory where the CM-mer[] vector is stored, which contains the canonical
m-mers.
Minimizer -> Decimal value of a minimizer.

Function block executed by nmk threads per read
{
        Function block that executes if p <sz (Those threads whose element processed in the previous
        zone is in a position lower than the beginning of the current zone - caterpillar type
        displacement)
        {
            -   p = p + nmk

                The position of the element of the vector Cm-mer [] that each
                thread will read and will process (caterpillar displacement) is updated

            -   cm = RSK[p + m - 1]

                Each thread reads its new element to be processed from the vector Cm-mer[] (the offset
                m-1 is due to the strategy used to rewrite Cm-mer[] in the RSK[] array where the bases
                of the read were initially stored)
        }

        Those threads whose cm value is less than the seed execute the following function
        {
            -   NMp = Min atómico (NMp, p)

                The position of the closest element whose value is less than the seed is obtained
        }
}

Local synchronization

Function block executed by 1 thread per read only if NMp is different from Mp (only if a new seed was
found)
{
    -   Mp = NMp
        Minimizer = RSK [Mp + m -1]

        The position and value of the seed found is updated

}

Local synchronization
```

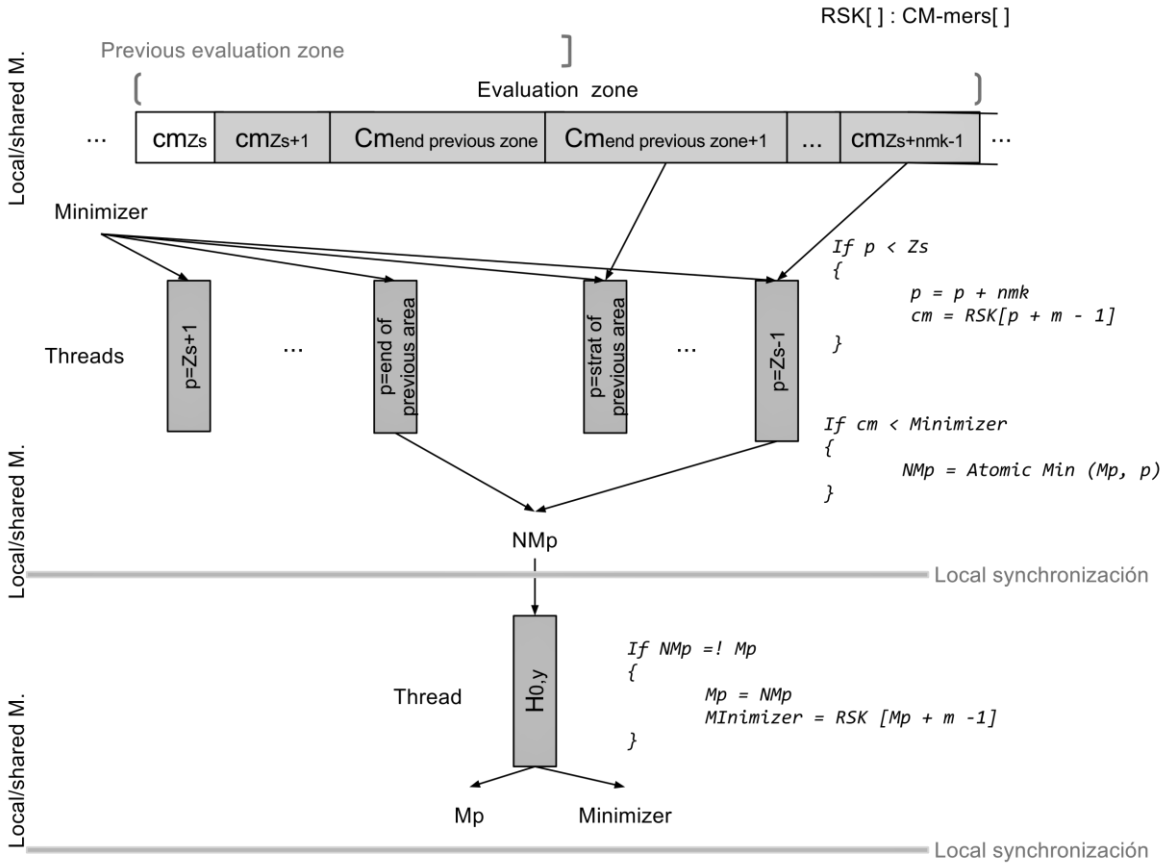

Figure 4. Illustration of the zone evaluation algorithm with reference seed (nearest smaller search algorithm). The conventions, acronyms and nomenclature in general are the same used in Algorithm 4.