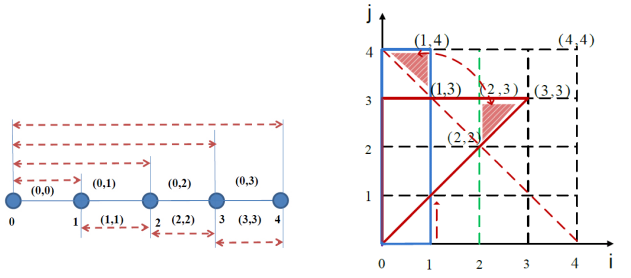
Fast LRT Implementation on Parallel Computer Architectures

1 MOTIVATIONS

1.1 Parallel Enumeration Schemes

In this section, we use a small example to demonstrate two different parallel strategies for performing LRT statistics in one-dimension data grid. The strategies are: (i) Overlapping Enumeration; (ii) Non-overlapping Enumeration. For simplicity, we will describe the strategies on GPGPU computing environment and they can be easily extended to multi-core and pc-clusters computing environments.

Example: A data set has five data points, denoted as $\{0, 1, 2, 3, 4\}$. Our task is how to enumerate all of the intervals among these five points in parallel. See Figure 1.



(a) Overlapping Enumeration (b) Non-overlapping Enumeration

Fig. 1: Overlapping scheme and Non-overlapping scheme

We express an interval between point i_1 and i_2 as $(i_1, i_2 - 1)$, where $i_2 > i_1$. We assume that each block can only maximally process 2 intervals. The total number of intervals among five points is computed as: $\frac{4 \cdot (4+1)}{2} = 2 \times 5 = 10$.

In the case of Overlapping Enumeration parallel strategy, every two overlapped intervals is assigned to a block. The next/previous interval always overlaps by one interval and the length of next interval is larger than previous interval. From Figure 1(a), $\{(0,0),(0,1)\} \rightarrow$ block 0; $\{(1,1),(1,2)\} \rightarrow$block 1, we can see that the overlapping scheme enumerates all of the intervals up to size of two. However, some blocks may have fewer number of intervals when the last interval has been enumerated. For block 4, it has only one interval(i.e. $\{(3,3)\}$). This leads to an observation of workload unbalancing among blocks. Furthermore, in each block, different interval is assigned to different thread. The lengths of intervals are different and this can further creates different workload among different threads during brute-force calculation.

Interestingly, if we plot each interval between point i_1 and i_2 onto two-dimensional space, a triangular shape is formed. In Figure 5b, all of the ten intervals forms a triangle shape enclosed by $\{(0,0) \rightarrow (3,3) \rightarrow (0,3)\}$. Furthermore, we notice that the total number of intervals can be treated as the product of $(n+1)$ and $\frac{(n)}{2}$. A rectangular shape can be formed by $(width, height) = (n+1, \frac{n}{2})$. In our example, $\{(0,0) \rightarrow (0,1) \rightarrow (1,4) \rightarrow (0,4)\}$ forms a rectangle and it has the exact same number of points as the formed triangular shape. Therefore, each point in this rectangle can be mapped to an interval. In a rectangular shape, we can divide the entire workload equally and assign each to each block. This solves the workload imbalance among blocks. In each block, each thread process the same number of intervals. However, this still does not solve the workload imbalance among each thread. In next section, we presents a dynamic programming scheme to pre-compute subsets of LRT statistic value and the computation of each thread becomes $O(1)$, which will solve the workload imbalance issue among threads.

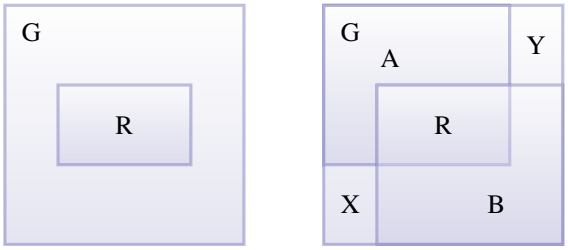
The Overlapping Enumeration strategy has been implemented in [apweb]. In this paper, the Non-overlapping Enumeration strategy, called Range Mapping scheme, is presented to overcome the limitations of the Overlapping Enumeration strategy.

1.2 Dynamic Pre-computation

The log-likelihood statistic (LRT) computation on 1EXP family is simplified to aggregate the statistic values from a given region R (denoted as $\sum$), which is composed one or multiple cells. This makes

Consider Figure 2(a) showing a rectangular area R embedded in a grid G. Instead of counting the number of elements in R directly, we express set R as set intersections of two sets A and B as shown in Figure 2(b). Set A is a rectangular region that starts in the upper left corner of the grid and ends at the lower right corner of R. Set B is a rectangular region that starts at the upper left corner of R and ends at the lower right corner of the grid. Hence, $R = A \cap B$. We denote the region from the lower left corner of G to the lower left corner of R by X and the region from the upper right corner of R to the upper right corner of G by Y.

By applying De Morgan's law and inclusion/exclusion principle, the query of counting the number of elements



(a) Region R in Grid G (b) R= $A \cap B$, $\overline{A} \cap \overline{B} = X \cup Y$

Fig. 2: Inclusive/Excusive Scheme

of region $R(x_1, y_1, x_2, y_2)$, where (x_1, y_1) is the upper left corner and (x_2, y_2) is the lower right corner is expressed by (see proof in appendix):

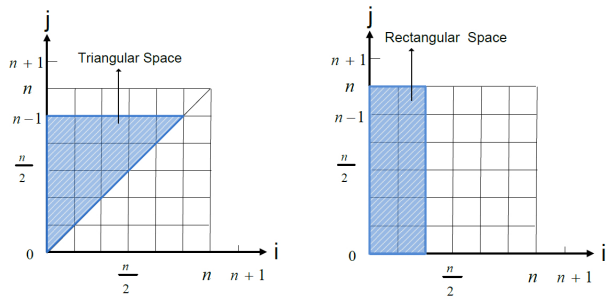
$$|R(x_1, y_1, x_2, y_2)| = |A(x_2, y_2)| + |B(x_1, y_1)| + |X(x_1, y_2)| + |Y(x_2, y_1)| - |G| \tag{1}$$

2 Range Mapping Scheme

In this section, we study how parallelism helps to enumerate all of the rectangular regions (R) in a spatial grid (G).

From section 1.1, we know that if all of the pairwise intervals between n data points, denoted as (i, j), are plotted onto two-dimensional coordinate system, they form triangular shape. Each pair of (i, j) in triangular-shaped space can be transformed to pair (i', j') in rectangular-shaped space. After transformation, it enables the whole rectangular space to be partitioned into equal portions. Each portion consists of same amount of pairs and is distributed onto different "parallel computing component" (PCC) to balance workload and facilitate parallelization. We name this transforming scheme as range mapping scheme. This scheme can be directly extended to two dimensional spatial grid and multi-dimensional grid. We give details in the following:

Figure 3a plots out all of the intervals (i, j) in two-dimensional space and we can see that these points (i, j) forms a triangle. Figure 3b shows the rectangular-shaped space after transformation.



(a) (b)

Fig. 3: 1-d Interval Transformation from $[n, n]$ to $[(n+1)/2, n]$

2.1 One-Dimensional Mapping

First, we consider the one-dimensional range mapping scheme. The axis oriented rectangles become intervals on one-dimensional space. We assume n is the size of the data grid (i.e. array).

Given the following information:

$$\mathbb{N} = \{0..n-1\}$$
$$\mathbb{R} = \{0..\frac{n}{2} - 1\} \times \{0..n\}$$
$$\mathbb{T} = \{(i, j) \in \mathbb{N}^2 | i <= j\}$$
$$\mathbb{M} = \{0..\frac{n \times (n+1)}{2} - 1\}$$

where $\mathbb{N}$ is coordinate range , $\mathbb{R}$ is rectangular space , $\mathbb{T}$ is triangular space and $\mathbb{M}$ is number range of intervals.

There are two cases for the range mapping in one-dimensional grid: (1) The grid size n is even (2) The grid size n is odd. Please see the following:

The grid size n ***is even:*** Function $\phi : \mathbb{M} \to \mathbb{R} \to \mathbb{T}$ is composed of two mapping functions: $\phi_1 : \mathbb{M} \to \mathbb{R}$ and $\phi_2 : \mathbb{R} \to \mathbb{T}$. Firstly, ϕ_1 transforms $x \in \mathbb{M}$ to a pair in $(i, j) \in \mathbb{R}$ in rectangular space. Secondly, ϕ_2 transforms the pair $(i, j) \in \mathbb{R}$ to a pair $(i', j') \in \mathbb{T}$ in triangular space.

Definition 1: Mapping Function

$$\phi_1 : \mathbb{M} \to \mathbb{R}$$
$$x \mapsto (i, j)$$

where $i = x/(n+1)$ and $j = x \mod (n+1)$.
In this function, x is an interval index and (i, j) is mapped to x by applying horner scheme.

Lemma 1: The range of ϕ_1 is R. That means: i is in the range of $\{0, ..., n/2-1\}$ and j is in the range of $\{0, ..., n\}$.

Lemma 2: The function $\phi_1 : \mathbb{M} \to \mathbb{R}, x \mapsto (i, j)$ is bijective.

Definition 2: Mapping Function

$$\phi_2 : \mathbb{R} \to \mathbb{T}$$
$$(i, j) \mapsto \begin{cases} (i, i+j) & \text{if } (i+j) \leq n \\ (n-i-1, n-j+n-i-1) & \text{otherwise} \end{cases}$$

Lemma 3: The range of ϕ_2 is T. That means: i is in the range of $\{0, ..., n/2-1\}$ and j is in the range of $\{0, ..., n\}$.

Lemma 4: The function $\phi_2 : \mathbb{R} \to \mathbb{T}, x \mapsto (i, j)$ is bijective.

Corollary 1: The function $\phi : \mathbb{MR} \rightarrow \mathbb{T}, x \mapsto (i, j)$ is bijective.

The grid size n ***is odd:*** We define a mapping function $\phi : M \rightarrow R \rightarrow T$. It is composed of two mapping functions: $\phi_1 : M \rightarrow R$ and $\phi_2 : R \rightarrow T$. Firstly, ϕ_1 transforms a coordinate index $x \in \mathbb{M}$ to a pair in $(i, j) \in R$ in rectangular space. Secondly, ϕ_2 transforms the rectangular pair $(i, j) \in R$ to a pair $(c, d) \in T$ in triangular space.

Definition 3: Mapping Function

$$\phi_1 : \mathbb{M} \rightarrow \mathbb{R}$$
$$x \mapsto (a, b)$$

where $a = x/(n+1)$ and $b = x \mod (n+1)$.

Lemma 5: The range of ϕ_1 is R. That means: a is in the range of $\{0, ..., n/2-1\}$ and b is in the range of $\{0, ..., n\}$.

Lemma 6: The function $\phi_1 : \mathbb{M} \rightarrow \mathbb{R}, x \mapsto (a, b)$ is bijective.

Definition 4: Mapping Function

$$\phi_2 : \mathbb{R} \rightarrow \mathbb{T}$$
$$(a, b) \mapsto \begin{cases} (a, a+b) & \text{if } (a+b) \leq n \\ (n-a-1, n-b+n-a-1) & \text{if} (a+1) \leq \frac{(n+1)}{2} \end{cases}$$

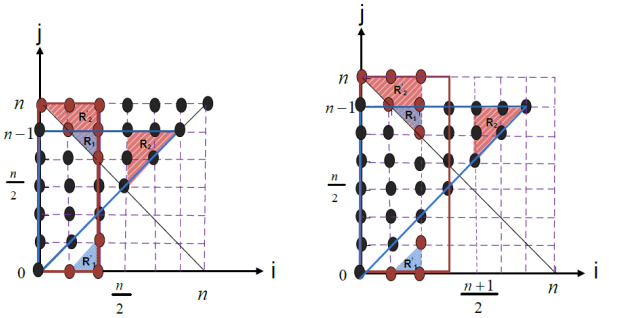
Lemma 7: The range of ϕ_2 is T. That means: a is in the range of $\{0, ..., n/2-1\}$ and b is in the range of $\{0, ..., n\}$.

Lemma 8: The function $\phi_2 : \mathbb{R} \rightarrow \mathbb{T}, x \mapsto (a, b)$ is injective.

Corollary 2: The function $\phi : \mathbb{MR} \rightarrow \mathbb{T}, x \mapsto (a, b)$ is injective.

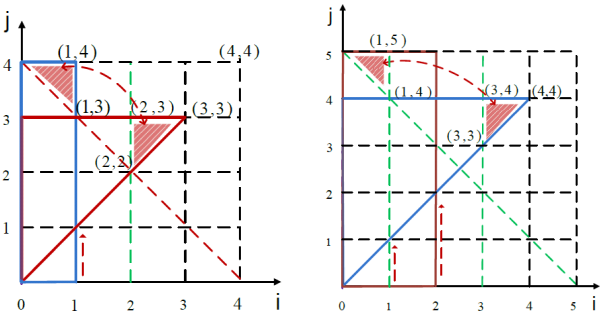
Figure 4a and Figure 4b show the solutions and implementation is shown in Algorithm 1. The detailed transform approach is given in the following part.

Example: To illustrate the transformation, Figure 5a and 5b show small examples on how to transform the intervals when having data points $\{x_0, x_1, x_2, x_3, x_4\}$ and $\{x_0, x_1, x_2, x_3, x_4, x_5\}$ respectively. For data point $(i, j) \in \{x_0, x_1, x_2, x_3, x_4\}$, the total intervals is 10. By plotting out all of them in Figure 5a, the triangular shaped space is bounded by $(0,0) \rightarrow (3,3) \rightarrow (0,3)$ and there are exactly 10 points in total. The rectangular shaped space is bounded by $(0,0) \rightarrow (1,0) \rightarrow (1,4) \rightarrow (0,4)$ and total number of points is exactly 10. Similarly, for data point $(i, j) \in \{x_0, x_1, x_2, x_3, x_4, x_5\}$, the triangular shaped space is bounded by $(0,0) \rightarrow (3,3) \rightarrow (0,3)$ and total number of points is . The rectangular shaped space is bounded by $(0,0) \rightarrow (1,0) \rightarrow (1,4) \rightarrow (0,4)$ and total number of points is exactly 10. Table 1 shows the transformations.



(a) Solution when Grid size n is even number (b) Solution when Grid size n is odd number

Fig. 4: 1-d Interval Transformation from $[n, n]$ to $[\lfloor (n+1)/2 \rfloor, n+1]$



(a) Grid size n =4 (b) Grid size n =5

Fig. 5: Example: 1-d Interval Transformation

(0,0)→ (0, 0)	(0, 1) → (0, 1)	(0, 2) → (0, 2)
(0,3)→ (0, 3)	(0, 4) → (3, 3)	(1, 4) → (2, 2)
(1,3)→ (2, 3)	(1, 0) → (1, 1)	(1, 1) → (1, 2)
(1,2)→ (1, 3)		

(a) Grid (4,4)

(0,0)→ (0, 0)	(0, 1) → (0, 1)	(0, 2) → (0, 2)
(0,3)→ (0, 3)	(0, 4) → (0, 4)	(0, 5) → (4, 4)
(1,0)→ (1, 1)	(1, 1) → (1, 2)	(1, 2) → (1, 3)
(1,3)→ (1, 4)	(1, 4) → (3, 4)	(1, 5) → (3, 3)
(2,0)→ (2, 2)	(2, 1) → (2, 3)	(2, 3) → (2, 4)

(b) Grid (5, 5)

TABLE 1: Example: Interval Transformation for even/odd number of points

2.2 Two-Dimensional Mapping

By extending one-dimensional mapping directly, two-dimensional mapping is given in the following:

$$\mathbb{N}_k = \{0..n_k - 1\}$$
$$\mathbb{R}_k = \{0..\frac{n_k}{2} - 1\} \times \{0..n_k\}$$
$$\mathbb{T}_k = \{(a_k, b_k) \in \mathbb{N}_k^2 | a_i <= b_k\}$$
$$\mathbb{M}_k = \{0..\frac{n \times (n_k + 1)}{2} - 1\}$$

Algorithm 1 Parallel Range Mapping in 1-d array

Input: n data points
Output: mapping all the intervals from rectangular to triangular space

```
1: //The interval (i',j') in rectangular space is transformed to
   interval (i,j) in triangular space
2: for i' ←0 to (n+1)/2 do
3:    for j' ←0 to n do
4:       if j'<(n − i') then
5:          i←i'
6:          j←(i'+j')
7:       else
8:          if 2(i' + 1)<(n + 1) then
9:             i←(n-i'+1)
10:            j←(n-j'+i)
11:      LRT computation for interval (i,j)
```

where $k = 1, 2$, N_1, N_2 are coordinate range , R_1, R_2 are rectangular space , T is triangular space and M is number range.

2.3 k-Dimensional Mapping

Mapping function $\phi^k : M^k \to R^k \to T^k$
by using Horner's scheme to transform a number in M^k to a k-dimensional tuple $(x_1, ..., x_k)$. Then we can use mappings ϕ_1 and ϕ_2 to do the translation from each element x_i to (l_i, u_i).

Definition 5: Mapping Function

$$\phi_1^k : \mathbb{M}^k \to \mathbb{R}^k$$
$$x \mapsto (x_1, ..., x_k)$$

where $x_k = x \div (n + 1)$ and $b = x \mod (n + 1)$.

2.4 Inclusive/Exclusive Pre-computation Scheme

Motivated by section 1.2, to obtain a query time of $\mathcal{O}(1)$, we need to pre-compute sets A, B, X, and Y for all possible regions in G. Since one of the corner is fixed we can pre-compute the cardinalities of these sets in tables of size $\mathcal{O}(n^2)$.

To obtain the tables for A, B, X, and Y, we employ dynamic programming. The dependency and statistic counts' propagation of rows and columns for these tables are shown in Figure 6a, 6b, 6c and 6d. For example, the table for A can be computed using the following recurrence relationship:

$$|A(i,j)| = |A(i,j-1)| + |A(i-1,j)| - |A(i-1,j-1)| + |G(i,j)| \tag{2}$$

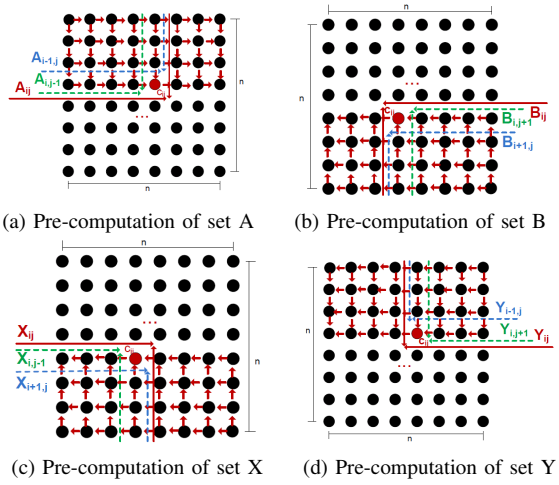
where $|G(i,j)|$ counts whether there is an element in the cell location (i,j). The first element and the first column and row need to be populated (initialized) so that all cardinalities of A can be computed. The counts in the remaining rows and columns are accumulated through the dependency of the previous row and column. Figure 6a shows the computation of set A and the implementations of it is listed in Algorithm 2 (see the proofs and the rest implementation of set B, X, Y in Appendix.)

Similarly, the computation of set B, X, Y is listed as the following:

$$|B(i,j)| = |B(i+1,j)| + |B(i,j+1)| - |B(i+1,j+1)| + |G(i,j)| \tag{3}$$

$$|X(i,j)| = |X(i,j+1)| + |X(i-1,j)| - |X(i-1,j+1)| + |G(i,j)| \tag{4}$$

$$|Y(i,j)| = |Y(i+1,j)| + |Y(i,j-1)| - |Y(i+1,j-1)| + |G(i,j)| \tag{5}$$



(a) Pre-computation of set A (b) Pre-computation of set B

(c) Pre-computation of set X (d) Pre-computation of set Y

Fig. 6: pre-computation of set A, B, X and Y

Due to the high dependency among rows and columns, pre-computing is hard to parallelise and the computation is very fast on a CPU. In our work, the pre-computation of set A, B, X, Y is done on a CPU.

Algorithm 2 Inclusive/Exclusive Pre-computation for Set A

Input: data grid (G)
Output: accumulated counts $A(i, j)$

//Initialize first element $A(0, 0)$
$A(0, 0) \leftarrow G(0, 0)$
//accumulation of remaining elements in first row
for $j \leftarrow 1$ to n **do**
    $A(0, j) \leftarrow G(0, j) + A(0, j - 1)$
//accumulation of remaining elements in first column
for $i \leftarrow 1$ to n **do**
    $A(i, 0) \leftarrow G(i, 0) + A(i - 1, 0)$
//accumulation of all the elements in remaining rows and columns
for $k \leftarrow 1$ to n **do**
    for $i \leftarrow$ k to n **do**
        $A(i, k) \leftarrow G(i, n + k) + A(i - 1, k) + A(i, k - 1) - A(i - 1, k - 1)$
    for $j \leftarrow$ k to n **do**
        $A(k, j) \leftarrow G(k, n + j) + A(k, j - 1) + A(k - 1, j) - A(k - 1, j - 1)$