DOI: xxx/xxxx

ARTICLE TYPE

Fast LRT Implementation on Parallel Computer Architectures

Author One*[1] | Author Two[2,3] | Author Three[3]

[1]Org Division, Org Name, State name, Country name
[2]Org Division, Org Name, State name, Country name
[3]Org Division, Org Name, State name, Country name

Correspondence
*Corresponding author name, This is sample corresponding address. Email: authorone@gmail.com

Present Address
This is sample for present address text this is sample for present address text

Summary

The Likelihood Ratio Test (LRT) is a method for identifying hotspots or anomalous rectangular regions for an spatial data grid that draws from an arbitrary distribution. LRT has wide-spread applications from social-media applications powered by mobile phones to medical image processing applications. Unfortunately, a naïve implementation of LRT exhibits a worst-case time complexity of $\mathcal{O}(n^4)$ for a two-dimensional grid.

For larger grids with a grid size of more than 64×64 cells, a naïve implementation of LRT becomes intractable (1) and it takes around several days to execute Grid(64,64) on a single-core CPU.

In this paper, we propose a parallel processing scheme for LRT computations for one parameter exponential (1EXP) distribution for multi-dimensional spatial data grids. The parallel processing scheme transforms LRT computations to a balanced embarrassingly parallel problem utilising a novel range mapping scheme. The new range mapping scheme relies on a dynamic pre-computation algorithm, and exhibits a worst-case complexity of $\mathcal{O}(n^2)$ for two dimensional spatial data-grids improving the complexity class by an order of two. Experiments were conducted for various computer architectures and platforms including shared-memory multi-core machines, multi-GPGPU computers, and the EC2 cloud computer. Extensive experiments are provided to demonstrate the utility of this approach and extensive performance analysis is given. In the experiments we could achieve speedups up to $20,000$ times using our parallel processing scheme on Grid $(1000, 1000)$.

KEYWORDS:
Spatial outlier, Likelihood Ratio Test, one parameter exponential distribution, Inclusive/Exclusive Principle, GPGPUs, Multi-core, EC2 Cloud Cluster

1 | INTRODUCTION

With the widespread availability of GPS-equipped smartphones and mobile sensors, there has been an urgent need to perform large scale spatial data analysis. For example, by carrying out a geographic projection of Twitter feeds, researchers are able to narrow down "hotspot" regions where a particular type of activity is attracting a disproportionate amount of attention. In neuroscience, high resolution MRIs facilitate the precise detection and localisation of regions of the brain which may indicate mental disorder. The statistical method of choice for identifying hotspots or anomalous regions is the Likelihood Ratio Test (LRT) statistic. Informally, the LRT of a spatial region compares the likelihood of the given spatial region with its complement, and hence can be used to identify hotspot regions. In (2), it was shown that the LRT value follows a χ^2 distribution, independent of the distribution of the underlying data.

For a $n \times n$ spatial grid, the worst-case execution time for identifying the most anomalous region is $\mathcal{O}(n^4)$. The reason for this high complexity class is that we have $\mathcal{O}(n^2)$ regions in the grid and computing the value of a single region R in the grid consumes $\mathcal{O}(n^2)$. As noted by Wu et al.(1), a naive implementation of LRT for a moderately sized 64×64 spatial grid may take nearly six hundred days[1]. Wu et al.(1) proposed a method which reduces the computation time to eleven days. However as noted previously in (3), this approach will not scale for larger data sets and the biggest spatial grid reported in(1) was 64×64.

The nature of LRT permits the independent computation of regions. This facilitates parallelization to some degree. However, the LRT computation of a region R involves the irregularly shaped computation of $\bar{R}$ (4). In a parallel environment, this can drastically reduce its computation performance(5, 6, 7). To reduce the overall enumeration cost and address the irregular computation, Pang et al. (4) presented a unified parallel approach for generalized LRT computation in spatial data grids on a GPGPU environment. The whole grid is partitioned into overlapped blocks and LRT computation is performed independently on each block in shared memory. For the regions which do not fit into shared memory, the computation is done on a CPU. While performance is greatly improved compared to the naive approach on a CPU, the granularity within each block is coarse. The computation of various sized sub-regions creates imbalanced workloads on each thread.

In this work, we focus on the parallel strategies for improving the LRT computation for the 1EXP family. We use a different strategy to ensure workload on each "parallel unit" (PU) [2] is balanced. We propose a novel range mapping scheme to transform irregular triangular shaped region space to a contiguously regular shaped region space with regards to iterate all of the regions in spatial grid (G). The fine-grained workload on each PU is produced by partitioning the regular space into different equal sized portions. Furthermore, a dynamic pre-computing scheme from our previous work(4) based on the Inclusive/Exclusive principle is presented for 1EXP. Four pre-computed data sets corresponding to four corners of the data grid are generated to reduce the cost of querying the intermediate statistics of each region to $\mathcal{O}(1)$.

Overall, the **contributions** in our work are:

- A novel range mapping scheme is proposed to provide the fine-grained parallelism for LRT computation.
- A dynamic pre-computation scheme is presented for fast computing.
- A k-best map reduce strategy is presented for accumulating distributed results on each "PU" and forms the final top-k regions at the end.
- The algorithms are implemented on various parallel architectures and corresponding performances are studied.

The rest of the paper is structured as follows. In Section 2, we provide background materials on LRT computation and its variation on 1EXP family. Motivations are presented in Section 3. In Section 4, we explain how we use Inclusive/Exclusive rule and dynamic programming to speed up the enumeration and processing of regions measurements. In Section 5, we present a novel range mapping scheme for 1EXP LRT computation for multi-dimensional data grid. To produce the final top-k regions, each parallel portion generates top k results and k-best reduction is done on CPU. The proof is provided in Section 6. The details of the implementation on Multi-core, GPGPU and EC2 cloud cluster are presented in Section 7. In Section 8, we evaluate experiments results on these different architectures and discussion is given. Related work is provided in Section 9. We give our conclusion in Section 10.

2 | BACKGROUND

We provide a brief but self-contained introduction for using LRT to find anomalous regions in a spatial setting. The regions are mapped onto a spatial grid G. Given a data set X, an assumed model distribution $f(X, \theta)$, a null hypothesis $H_0 : \theta \in \Theta_0$ and an alternate hypothesis $H_1 : \theta \in \Theta - \Theta_0$, LRT is the ratio

$$\lambda = \frac{sup_{\Theta_0}\{L(\theta|X)|H_0\}}{sup_{\Theta}\{L(\theta|X)|H_1\}} \quad (1)$$

where $L()$ is the likelihood function and θ is a set of parameters for the distribution (1). In a spatial setting, the null hypothesis is that the data in a region R (that is currently being tested) and its complement (denoted as $\bar{R}$) are governed by the same parameters. Thus if a region R is anomalous then the alternate hypothesis will most likely be a better fit and the denominator of λ will have a higher value for the maximum likelihood estimator of θ. A remarkable fact about λ is that under mild regularity conditions, the asymptotic distribution of $\Lambda \equiv -2log\lambda$ follows a χ^2_k distribution with k degrees of freedom, where k is the number of free parameters[3]. Thus regions whose Λ value falls in the tail of the χ^2 distribution are likely to be anomalous (1).

[1] The experiment results were reported in 2009.

[2] We define the (computational) granularity of a parallel architecture as the largest "parallel unit" that should run sequentially on an "application computing unit". It refers to "block/thread" for GPGPU, "core" for multi-core and "process" for cloud pc-cluster

[3] If the χ^2 distribution is not applicable then Monte Carlo simulation can be used to ascertain the p-value.

The one-parameter exponential family (1EXP) simplifies the computation of the LRT statistic. The distribution of a random variable $x \in X$ belongs to a one-parameter exponential family (8) (denoted by $x \sim 1EXP(\theta, \phi, T, B_e, a)$ if it has a probability density given by

$$f(x;\theta) = C(x,\phi)exp((\theta T(x) - B_e(\theta))/a(\phi)) \quad (2)$$

where $T(\cdot)$ is some measurable function, $a(\phi)$ is a function of some known scale parameter ($\phi > 0$), θ is an unknown parameter (called the natural parameter), $B_e(\cdot)$ is a strictly convex function and $C()$ is some constant value. Note that the support of $x : f(x;\theta) > 0$ is independent of θ.

Theorem 1. *(8): Let data set $X_R(R \in G)$ be independently distributed with $x_R \sim 1EXP(\theta, \phi, T, B_e, a)$. The log-likelihood ratio test statistic for testing $H_0 : \theta_R = \theta_{\bar{R}}$ versus $H_1 : \theta_R \neq \theta_{\bar{R}}$ is given by:*

$$\Lambda = m_R g_e(G\frac{m_R}{b_R}) - \frac{b_R}{G}B_e(g_e(G\frac{m_R}{b_R})) + (1-m_R)g_e(G\frac{1-m_R}{1-b_R}) - \frac{(1-b_R)}{G}B_e(g_e(G\frac{1-m_R}{1-b_R})) \quad (3)$$

Parameter m_R is the fraction measurement of Region R in total and b_R is the fraction of baseline measure of Region R in total. Correspondingly, $1 - m_R$ is the fraction measurement of Region $\bar{R}$ in total and $1 - b_R$ is the fraction of baseline measure of Region R in total. Parameters m_R and b_R are important measurements to calculate the statistic of 1EXP. More information about the functions g_e, G and B_e can be foudn in (8). For example, if we assume the counts m_R in a region R follow a Poisson distribution with baseline b and intensity λ, then a random variable $x \sim Poisson(\lambda\mu)$ is a member of 1EXP with $T(x) = x/\mu$, $\Phi = 1/\mu$, $a(\Phi) = \Phi$, $\theta = log(\lambda)$, $B_e\theta = exp(\eta)$, $g_e(x) = log(x)$. For any regions R and $\bar{R}$, m_R and $m_{\bar{R}}$ are independently Poisson distributed with a mean of $\{exp(\theta_R)b_R\}$ and $\{exp(\theta_{\bar{R}})b_{\bar{R}}\}$, respectively. The log-likelihood ratio is calculated by: $c(m_R log(\frac{m_R}{b_R}) + (1-m_R)log(\frac{1-m_R}{1-b_R}))$ (c.f. (8)). The closed-form formula for LRT generalizes to the 1EXP family of distributions (8).

3 | MOTIVATIONS

3.1 | Parallel Enumeration Schemes

In this section, we demonstrate two different parallel strategies for performing LRT statistics in a one-dimensional data grid by example. The strategies are (i) Overlapping Enumeration, and (ii) Non-overlapping Enumeration. The Overlapping Enumeration strategy has been implemented in (4). In the example, we simulate the architectural environment of (4), and illustrate its drawbacks. The non-overlapping Enumeration strategy, called Range Mapping scheme, will be presented in this paper to overcome the limitations of the Overlapping Enumeration strategy. For simplicity, we will describe the strategies on an abstract parallel machine not detailing its implementation on a concrete parallel platform. The abstract parallel machine has a fixed number of parallel units (PU) whose memory for each PU is finite.

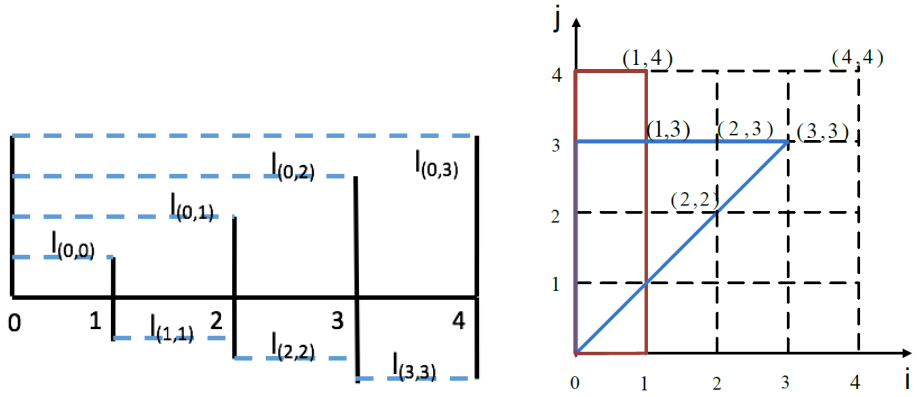
EXAMPLE: Assume we have a one-dimensional spatial data grid with five data points, denoted as $\{O_0, O_1, O_2, O_3, O_4\}$. We would like to compute the LRT of each interval for the one-dimensional spatial data grid. To find the outlier, we enumerate all intervals that can be spawned over the one-dimensional data grid by performing LRT calculation explained in above section 2. We express $l_{(i,j)}$ as an interval between point O_i and O_j, where $(0 \leq i < j \leq n)$ and there are $n+1$ points in one dimensional grid. There are $\frac{4\cdot(4+1)}{2} = 10$ intervals in total, i.e., number of pairs (i, j) for which $(0 \leq i < j \leq 4)$ holds. Figure 1 a visualizes some intervals among above five data points.

To illustrate how to compute LRT for each interval in this example, we assume each interval independently follows poisson distribution and the calculation of LRT is given in last part of section 2 of this paper. The formula is repeated here: $c(m_l log(\frac{m_l}{b_l}) + (1-m_l)log(\frac{1-m_l}{1-b_l}))$, where c is a constant factor and l denotes interval (c.f. (8)). Another detailed similar example in two-dimensional grid is provided in (9).Therefore, to compute LRT value of an interval in the formula, m_l (i.e. success count) and b_l (i.e. population count) are required. Table 1 provides sample value of (m_l, b_l) for each independent interval. To compute the LRT of an interval, there are two different kind of calculations: (a) compute LRT of a single independent interval l, (b) compute LRT of an interval l, which is union of multiple non-overlapped intervals $l = l_1 \cup l_2 \cup ...l_t$. For example, to obtain the LRT value of a single independent interval $l_{(0,0)}$, the computation is: $lrt_{(0,0)} = 2 \cdot log(\frac{2}{10}) + (1-2)log(\frac{1-2}{1-10}) = -1.0$. To obtain a larger interval interval $l_{(0,1)}$, which is the union of independent interval $l_{(0,0)}$ and $l_{(1,1)}$, (m_l, b_l) needs to be aggregated first. Since the data here follows poisson distribution, we know from above that $m_{l(0,1)} = m_{l(0,0)} + m_{l(1,1)} = 2 + 2 = 4$ and $b_{l(0,1)} = b_{l(0,0)} + b_{l(1,1)} = 10 + 10 = 20$. Thus $lrt_{(0,1)} = 4 \cdot log(\frac{4}{20}) + (1-4)log(\frac{1-4}{1-20}) = -0.9$. The LRT of each sub-intervals can be obtained using one of the two approaches. Since m_l and b_l need to be aggregated for calculating LRTs of some sub-intervals, dynamic pr-computation will be used in our following strategies to fully parallelize the computation.

From above, we know the computation of LRT for each interval. Now we move to present the strategies of parallelizing LRT computation in a data grid. We assume the parameters m_l and b_l of each independent single interval takes up 4 bytes of data storage, respectively. Let's assume that

TABLE 1 m_R and b_R statistics for each independent interval $l_(i,j)$

Interval	m_R	b_R
(0,0)	2	10
(1,1)	2	10
(2,2)	15	10
(3,3)	3	10



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

FIGURE 1 Parallel Enumeration Schemes

each parallel unit of the abstract parallel machine has at most 16 bytes of data storage. Hence, each parallel unit is limited to compute at most two intervals.

Overlapping Enumeration Parallel Strategy

Since each parallel unit of the parallel machine can only maximally store and process 16 bytes of data and the statistics values of each single independent interval takes up 8 bytes (i.e. 4 bytes of m_l + 4 bytes of b_l = 8 bytes), therefore the maximum number of independent interval can be processed is $\frac{max\_allowed\_bytes\_pu}{bytes\_of\_interval} = \frac{16}{8} = 2$ for a single parallel unit at once. In the overlapping strategy (4), each PU_i is associated to k continuous intervals. For this example, each PU enumerates every two continuous overlapping intervals (i.e., the interval (i,j) and the interval $(i,j+1)$ by a single parallel unit. The $LRTs$ of the remaining intervals will be sequentially processed and merged by the CPU (c.f. (4)). The reasons are: (a)any interval with length $|l| > k$ (k=2 in this example) cannot be fitted to a PU due to lack of memory of parallel unit; (b)every two non-continuous single intervals processed on a PU cannot process intervals optimally. On the other hand, if PU store k continuous single intervals, it can process all of the $\frac{k\cdot(k+1)}{2}$ sub-intervals on a single PU. Table 2 illustrates the allocation of continuous intervals to parallel units, and the allocation of the rest intervals to the CPU. For example, PU_0 is assigned the work to compute the LRT for the intervals $\{l_{(0,0)}, l_{(0,1)}\}$, and PU_1 is assigned for computing the LRT for intervals $\{l_{(1,1)}, l_{(1,2)}\}$. Figure 1 a illustrates the enumeration of some intervals for those 5 data points by applying over-lapping strategy.

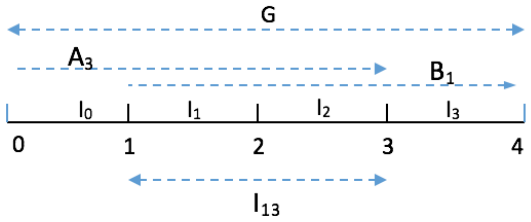
The aforementioned approach has a few drawbacks: First, non-continuous intervals cannot be processed by parallel units. Hence, there are around $n \cdot \binom{n}{k}$ (k>2 in above example) intervals that will be processed sequentially by the CPU. Second, some PUs may have fewer number of intervals as a consequence. For example, PU_4 has only one interval (i.e. $\{l_{(3,3)}\}$). Hence, the overlapping enumeration parallel strategy causes an unbalanced workload among PUs. Third, for each PU, each interval is assigned to a thread. The lengths of intervals are different and this can further creates different workloads among different threads while accumulating statistics for computing LRT. To overcome the drawbacks of this approach, the Inclusion-Exclusion parallel strategy is proposed.

Inclusion-Exclusion Parallel Strategy

For this new scheme, we apply a map-reduce scheme (10) (11) that distributes intervals among parallel units. The parallel units compute the k-best intervals from the assigned intervals. In the final stage, all k-best intervals per parallel unit are reduced to a single k-best list of regions. To overcome

TABLE 2 Example: PU assignment

	Intervals on PU		Intervals on CPU
$Grid$	$interval_1$	$interval_2$	$intervals$
PU_1	$l_{(0,0)}$	$l_{(0,1)}$	$l_{(0,2)}$
PU_2	$l_{(1,1)}$	$l_{(1,2)}$	$l_{(0,3)}$
PU_3	$l_{(2,2)}$	$l_{(2,3)}$	$l_{(1,3)}$
PU_4	$l_{(3,3)}$		



(a) inclusive exclusive strategy for 4 intervals

Notation	Interval	m_l	b_l
I_0	(0,0)	2	10
I_1	(1,1)	2	8
I_2	(2,2)	15	20
I_3	(3,3)	3	2

(b) m_l, b_l count for each interval

FIGURE 2 Example of Inclusive Exclusive Parallel Strategy

the issues of the overlapping enumeration scheme, we need to solve two challenges: First, how can we compute the LRT of a single interval of a spatial data-grid in $\mathcal{O}(1)$. Second, how can we enumerate intervals so that we can equally distribute them on parallel units without space and runtime overheads.

The first challenge can be solved by relying on the 1EXP property. For example, the parameters of the interval (i, j) can be deduced by having the calculation:

The region A_j counts all the events from the left most data-cell in the spatial grid to position j, i.e., $A_j = I_0 + \ldots + I_j$. The region B_i counts all the events from the right most data-cell to position i, i.e., $B_i = I_3 + \ldots + I_j$. The region $G = I_0 + \ldots + I_3$ denotes the whole spatial grid.

$$I_{ij} = A_j + B_i - G \tag{4}$$

For an interval I_{ij}, the counts can be deduced by applying the inclusion/exclusion principle on the set count. The sum of $|A_i|$ + $|B_j|$ over-counts the interval by G. The computation of A_j and B_i can be done in linear time, i.e., $\mathcal{O}(n)$, as a pre-computation step. The actual computation of an interval is reduced to one addition and subtraction, hence, the computation of a single interval is performed in $\mathcal{O}(1)$. In section 2, the property of the log-likelihood statistic (LRT) computation on 1EXP family allows us to simplify to aggregate the events count from a given interval I (denoted as $\sum$). The events count from actual measurement and baseline measurement m_l, b_l are obtained based on $\sum I$ without direct computation of the complement of $I(i.e.\bar{I})$. After collecting the aggregated events count of (m_l, b_l), theorem 1 is applied to get the LRT value of interval I. In Figure 2 a, it shows four interval composed by five points and counts of m_l, b_l for I_i are shown in Figure 2 b. For example, for interval $I_{13} = A_3 + B_1 - G$, we can get $m_{13} = (2+2+15)+(2+15+3)-(2+2+15+3) = 17$ and $b_{13} = (10+8+20)+(8+6+2)-(10+8+20+2) = 28$. From above, we know the LRT value of $l_{(1,3)}$ is calculated as: $lrt_{(1,3)} = m_{13} log(\frac{m_{13}}{b_{13}} + (1-m_{13}) log(\frac{1-m_{13}}{1-b_{13}}) = 17 \cdot log(17/28) - 16 \cdot log(16/27) = -0.11$

For balancing the workload, we require that intervals are equally distributed on parallel units. We solve this problem by enumerating all intervals from zero onwards. For our example, we enumerate intervals and assign each interval a unique number between 0 and $\frac{n(n+1)}{2} - 1 = 10$ where n is 4, which we refer to as representative number. By enumerating the intervals we can easily split the range of the enumeration into equal sub-ranges that we distribute on the parallel units. If there are p parallel units and $\frac{n(n+1)}{2}$ intervals, we achieve a balanced workload with approximately $\left\lceil \frac{n(n+1)}{2p} \right\rceil$ intervals per parallel unit since each interval takes the same constant runtime to compute.

Let's assume we have 5 parallel units. parallel unit can process two intervals at most. Hence, each parallel unit processes exactly 2 intervals. Intervals with representative numbers from 0 to 1 are mapped to the first parallel unit; intervals with representative numbers from 2 to 3 are mapped to the second parallel unit; and so forth.

The question arises how can we find good representative numbers for intervals in a spatial data-grid. We could simply enumerate all intervals exhaustively, assign numbers for each interval on the fly, and store the representative-number/interval map explicitly. Unfortunately, the explicit

TABLE 3 one digit number to two-digit number system

0→ $(0,0)$	$1 \rightarrow (0,1)$	$2 \rightarrow (0,2)$	$3 \rightarrow (0,3)$	$4 \rightarrow (0,4)$
5→ $(1,0)$	$6 \rightarrow (1,1)$	$7 \rightarrow (1,2)$	$8 \rightarrow (1,3)$	$9 \rightarrow (1,4)$

TABLE 4 two-digit system converts to intervals

(0,0)→ $(0,0)$	$(0,1) \rightarrow (0,1)$	$(0,2) \rightarrow (0,2)$	$(0,3) \rightarrow (0,3)$	$(0,4) \rightarrow (3,3)$
$(1,0) \rightarrow (1,1)$	$(1,1) \rightarrow (1,2)$	$(1,2) \rightarrow (1,3)$	(1,3)→ $(2,3)$	$(1,4) \rightarrow (2,2)$

construction of the array map renders to be expensive in the presence of multi-dimensional large spatial data grids. A one-dimensional spatial data grid exhibits a the space worst-case complexity of $\mathcal{O}(n^2)$ for computing the map. For a multi-dimensional spatial data grid, the space worst-case complexity grows exponential with the number of dimensions, i.e., $\mathcal{O}(n^{2k})$ where k is the dimensionality of the spatial data grid. Also, the map must be shared among all parallel units requiring communication and memory for storage. To overcome the high storage and communication costs of the map, we propose a closed-form for mapping representative numbers to intervals.

If we map each interval $l_{i,j}$ as a point (i, j) and plot these points onto two-dimensional space, a triangular shape is formed. In Figure 1 b, all of the 10 intervals in this example forms a triangle shape enclosed by points $(0,0), (3,3), (0,3)$, (i.e. intervals: $l_{(0,0)}, l_{(3,3)}, l_{(0,3)}$).

We notice that the total number of intervals is the product of $(n+1)$ and $\frac{(n)}{2}$. A rectangular shape can be formed by this product: $[width, height] = [n+1, \frac{n}{2}]$. In this example, $\{(0,0), (0,1), (1,4), (0,4)\}$ forms a rectangle R_1 and it has the exact same number of points as the formed triangular shape R_2. Therefore, each point in this rectangle can be mapped to one point in the triangle space.

The closed-form has a runtime and space worst-case complexity of $\mathcal{O}(1)$. The closed-form calculation is divided into two steps: First, we convert the representative number to a two-digit number system where the first base b_0 is n and the second base b_1 is $\frac{n+1}{2}$. We refer to the two-digits as Rectangular Space. Given a representative number r, the number r is expressed by $r = d_1 b_0 + d_0$ where d_1 and d_0 are the digits of the representative number in the two-digit number system with the base b_0 and b_1, respectively. Note that the digit d_1 ranges from 0 to $\frac{n+1}{2} - 1$ and the second digit ranges from 0 to $n-1$. The conversion follows the Horner scheme (12) for number conversion, i.e.,

$$d_0 = r \mod n$$
$$d_1 = (r \div n) \mod \frac{n+1}{2}$$

In the later section of this work, we refer to the mapping of a representative number to two digits as ϕ_1. The two-digit number system has exactly $\frac{n(n+1)}{2}$ numbers available. However, the two digits of a representative number in the two-digit number system with base n and $\frac{n+1}{2}$ do not represent intervals yet. There could be numbers whose first digit is larger than the second digit and vice versa, that would violate the constraints $0 \leq i \leq j < n$.

Hence, we require a further mapping that maps the two digits of a representative number to intervals. We refer to the second mapping as ϕ_2 in this work. The second mapping suffices the interval condition,

$$(d_1, d_0) \mapsto \begin{cases} (d_1, d_1 + d_0), & \text{if } d_1 + d_0 < n, \\ (n - d_1 - 1, 2n - d_1 - d_0 - 1), & \text{otherwise.} \end{cases}$$

For our example, we can convert a representative number to the two-digit number system, subsequently convert the two digits to intervals as shown in Table 3 and 4 .

In the above, we know that the number of intervals is treated as the product of $(n+1)$ and $\frac{n}{2}$ to form a rectangular shape. When n is odd, the multiplication of $\frac{n}{2}$ and $(n+1)$ won't produce the exact number of intervals. To generalize, we extend the calculation of number of intervals to $\lfloor \frac{n+1}{2} \rfloor \cdot (n+1)$. When n is even, $\lfloor \frac{n+1}{2} \rfloor = \frac{n}{2}$ and exact number of interval can be mapped. When n is odd, the generalized calculation of $\lfloor \frac{n+1}{2} \rfloor \cdot (n+1)$ is greater than $\lfloor \frac{n+1}{2} \rfloor \cdot n$ with $\frac{n+1}{2}$. In addition to satisfy the constraints $0 \leq i \leq j < n$, $d_1 \leq \frac{n+1}{2}$ is also applied to filter out these more points in the following:

$$(d_1, d_0) \mapsto \begin{cases} (d_1, d_1 + d_0), & \text{if } d_1 + d_0 < n, \\ (n - d_1 - 1, 2n - d_1 - d_0 - 1), & \text{if } (d_1 + 1) \leq \frac{n+1}{2}. \end{cases}$$

To illustrate the transformation, Figure 3 a and 3 b show small examples on how to transform the intervals when having data points $\{O_0, O_1, O_2, O_3, O_4\}$ and $\{O_0, O_1, O_2, O_3, O_4, O_5\}$ respectively. By plotting out all of intervals in first case in Figure 3 a, the triangular shaped space is bounded by $\{(0,0), (3,3), (0,3)\}$. And the rectangular shaped space used for transforming to triangular shape is bounded by

$\{(0,0),(1,0),(1,4),(0,4)\}$, which is converted by the representative number from the number of total intervals(i.e. 10). Similarly, for second case when $n = 5$ is odd, the triangular shaped space is bounded by $\{(0,0),(4,4),(0,4)\}$ and the rectangular shaped space is bounded by $\{(0,0),(2,0),(2,5),(0,5)\}$ by satisfying the constraints in above . Table 5 shows the transformations.

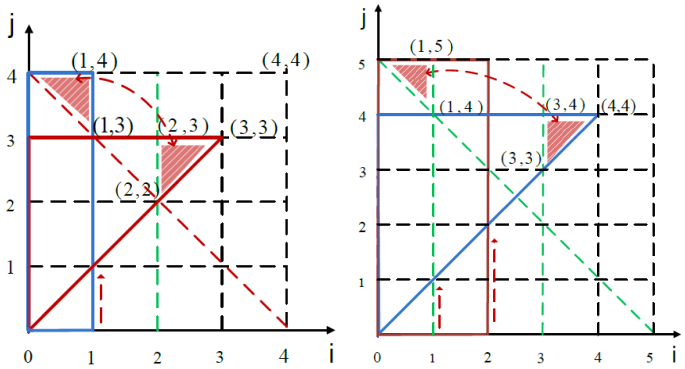


FIGURE 3 Example: 1-d Interval Transformation

(0,0)$\rightarrow (0,0)$	$(0,1) \rightarrow (0,1)$	$(0,2) \rightarrow (0,2)$	$(0,3) \rightarrow (0,3)$	$(0,4) \rightarrow (3,3)$
(1,4) $\rightarrow (2,2)$	$(1,3) \rightarrow (2,3)$	$(1,0) \rightarrow (1,1)$	$(1,1) \rightarrow (1,2)$	(1,2)$\rightarrow (1,3)$

Grid (4,4)

(0,0)$\rightarrow (0,0)$	$(0,1) \rightarrow (0,1)$	$(0,2) \rightarrow (0,2)$
(0,3)$\rightarrow (0,3)$	$(0,4) \rightarrow (0,4)$	$(0,5) \rightarrow (4,4)$
(1,0)$\rightarrow (1,1)$	$(1,1) \rightarrow (1,2)$	$(1,2) \rightarrow (1,3)$
(1,3)$\rightarrow (1,4)$	$(1,4) \rightarrow (3,4)$	$(1,5) \rightarrow (3,3)$
(2,0)$\rightarrow (2,2)$	$(2,1) \rightarrow (2,3)$	$(2,3) \rightarrow (2,4)$

Grid (5, 5)

TABLE 5 Example: Interval Transformation for even/odd number of points

Therefore, for our proposed strategy, in the first step, we convert the representative number to the two-digit number system. In the second step we map the two digits to an interval. Both steps can be performed with a constant number of arithmetic operations. Hence, there space and runtime worst-case complexity of the closed-form is $\mathcal{O}(1)$.

4 | DYNAMIC PRE-COMPUTATION SCHEME FOR MULTI-DIMENSIONAL GRIDS

From section 2, we know the log-likelihood statistic (LRT) computation on 1EXP family is simplified to aggregate the statistic values from a given region R (denoted as $\sum$). The fraction of the total from actual measurement and baseline measurement m_R, b_R are obtained based on $\sum R$ without direct computation of the complement of $R(i.e.\bar{R})$. After collecting the aggregated statistic values (m_R, b_R), theorem 1 is applied to get the LRT value of region R. This property works for any dimensional data grid.

4.1 | Pre-computation in one-dimension grid

From section 3, we already know inclusive/exclusive precomputation for calculating LRT of intervals can be done in one-dimension grid based on 1EXP property. Figure 4 shows $n-1$ intervals in one-dimensional grid. For any interval I_{ij}, it can be expressed as the overlapped interval between $I_{0,j}$ and $I_{i,n}$. Instead of computing its statistical counts (m_I, r_I) from position i to position j directly, (m_I, r_I) can be obtained in the following equation 5.

$$I_{ij} = A_j + B_i - G \tag{5}$$

, where interval A_j counts all the events from the left most data-cell in the spatial grid to position j. The interval B_i counts all the events from the right most data-cell to position i and G denotes counts of the whole spatial grid.

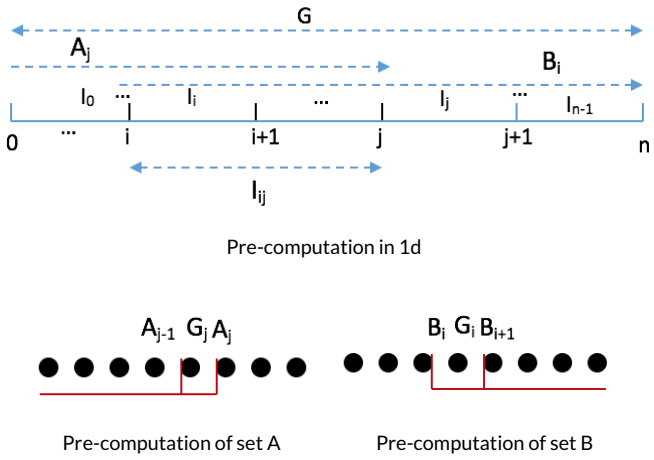


FIGURE 4 Inclusive Exclusive Pre-computation in 1d

From Equation 5, we can see that a query time of $\mathcal{O}(1)$ can be achieved by pre-computing statistics of sets A, B for all possible intervals in G. Since one of the corner of A, B is fixed, we can pre-compute the cardinalities of these sets in arrays of size $\mathcal{O}(n)$. This approach is based on the Inclusive/Exclusive principle and dynamic programming accumulates the statistic of an interval I.

To obtain the arrays for A, B, we employ dynamic programming. The dependency and statistic counts' propagation of data intervals for the array for A and B can be computed using the following recurrence relationship:

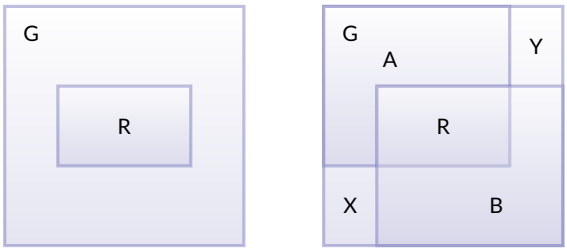
$$|A_j| = |A_{j-1}| + |G_j| \tag{6}$$

where $|G_j|$ represents the counts (m_j, r_j) of the unit interval at position $(j-1, j)$ .The first element $|G_0|$ needs to be populated (initialized) so that all cardinalities of A can be computed. The counts in the remaining array are accumulated through the dependency of the previous element in the array.

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

$$|B_i| = |B_{i+1}| + |G_i| \tag{7}$$

where $|G_i|$ represents the counts (m_i, r_i) of the unit interval at position $(i, i+1)$ .The first element $|G_{n-1}|$ needs to be populated (initialized) so that all cardinalities of B can be computed. The counts in the remaining array are accumulated through the dependency of the previous element in the array.



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

FIGURE 5 Set Relations for Region Problem

4.2 | Pre-computation in two-dimension grid

Consider Figure 5 (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 5 (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 of region $R(i_1, j_1, i_2, j_2)$, where (i_1, j_1) is the upper left corner and (i_2, j_2) is the lower right corner is expressed by (see Proof in appendix):

$$|R(i_1, j_1, i_2, j_2)| = |A(i_2, j_2)| + |B(i_1, j_1)| + |X(i_1, j_2)| + |Y(i_2, j_1)| - |G| \tag{8}$$

From Equation 8, we can see that a query time of $\mathcal{O}(1)$ can be achieved by pre-computing statistics of sets A, B, X, and Y for all possible regions in G. Since one of the corner of A, B, X, and Y is fixed, we can pre-compute the cardinalities of these sets in tables of size $\mathcal{O}(n^2)$.

This pre-computation scheme has been implemented in (4) and we give more details and related algorithms in this section. This novel and unique approach based on the Inclusive/Exclusive principle and dynamic programming accumulates the statistic of a region R. We build a table in time $\mathcal{O}(n^2)$ and then compute the statistic of any region R in $\mathcal{O}(1)$ time.

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 6 a, 6 b, 6 c and 6 d. For example, the table for A can be computed using the following recurrence relationship:

$$|A(i,j)| = |A(i,j-1)| + |A(i-1,j)| - |A(i-1,j-1)| + |G(i,j)| \tag{9}$$

where $|G(i,j)|$ [4]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 6 a shows the computation of set A and the implementations of it is listed in Algorithm 1 (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:

$$|B(i,j)| = |B(i+1,j)| + |B(i,j+1)| - |B(i+1,j+1)| + |G(i,j)| \tag{10}$$

$$|X(i,j)| = |X(i,j+1)| + |X(i-1,j)| - |X(i-1,j+1)| + |G(i,j)| \tag{11}$$

$$|Y(i,j)| = |Y(i+1,j)| + |Y(i,j-1)| - |Y(i+1,j-1)| + |G(i,j)| \tag{12}$$

[4](i,j) represents (x_i, x_j) for simplicity.

Algorithm 1 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*←1 to *n* **do**

  $A(0, j) \leftarrow G(0, j) + A(0, j - 1)$

end for

//accumulation of remaining elements in first column

for *i*←1 to *n* **do**

  $A(i, 0) \leftarrow G(i, 0) + A(i - 1, 0)$

end for

//accumulation of all the elements in remaining rows and columns

for *k*←1 to *n* **do**

  for *i*←k to *n* **do**

    $A(i, k) \leftarrow G(i, n + k) + A(i - 1, k) + A(i, k - 1) - A(i - 1, k - 1)$

  end for

  for *j*←k to *n* **do**

    $A(k, j) \leftarrow G(k, n + j) + A(k, j - 1) + A(k - 1, j) - A(k - 1, j - 1)$

  end for

end for

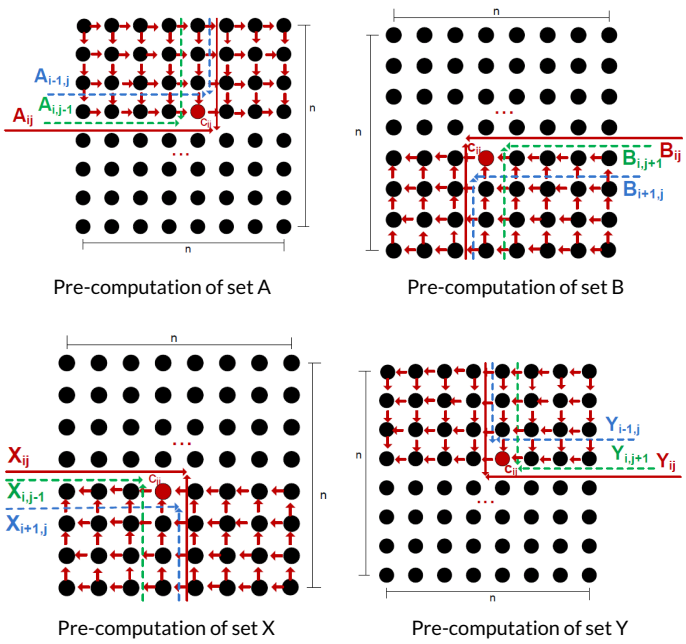


FIGURE 6 pre-computation of set A, B, X and Y

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.

4.3 | Pre-computation in Multi-dimension grid

The precomputation of counts in one dimension and two dimension grid can be directly extended to multi-dimensional grid. Let's assume the dimension of the grid G is k and each dimension of the grid is partitioned into n equal cells. A hyper-region R is embedded in this multi-dimensional grid G. Instead of counting the number of elements (m_R, b_R) in R directly, the count set of R is expressed as set intersections of 2^k sets denoted as A_1 ,A_2...,A_{2^k}. Set A_t $(1 \leq i \leq 2^k)$ is a hyper-region that starts from one corner of the grid, ends at the other corner of R by moving towards to the other end of grid to form a hyper-region . For example, the hyper-region A_1 is formed by starting at the corner of $(0, 0, ..., 0)$ and ending at the other corner of A_1 which moving towards to the corner $(n, n, ..n)$ of the grid G. For more details, each of the hyper-region, for instance, A_1 could be defined as counts between $(0, 0..., 0)$ to $(i, j..., k)$ in arbitrary coordinate (corresponding to the corner point (1,1,1) in the hyper-cube). A_2 could be defined as the counts between $(0, 0, ..., n)$ to $(i, j, ..., k)$ (corresponding to the corner point (0,0,1) in the hyper-cube).

For a (n, k) grid, there are 2^k corners. By applying De Morgan's law and inclusion/exclusion principle, the query of counting the number of elements of region $R(i_1, j_1, .., k_1, i_2, j_2, .., k_2)$, where $(i_1, j_1, ..., k_1)$ is one corner point of the grid and $(i_2, j_2, .., k_2)$ is the corner point of region R can be expressed in the following:

$$|R(i_1, j_1, ..., k_1, i_2, j_2, ..., k_2)| = |A_1(i_2, j_2, ..., k_2)| + |A_2(i_1, j_2, ..k_1)| + |A_3(i_2, j_1, .., k_1)| + ... + |A_{2^k}(i_1, j_1, ..., k_1)| - |G| \quad (13)$$

From Equation 13, we can see that a query time of $\mathcal{O}(1)$ can be achieved by pre-computing statistics of sets A_1, A_2,..., A_{2^k} for all possible regions in G.

To obtain a query time of $\mathcal{O}(1)$, we need to pre-compute sets A_t $(1 \leq t \leq 2^k)$ 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 $\mathcal{O}(n^k)$. The dependency and statistic counts' propagation of rows and columns for these tables are similar as the computation in one dimension and two dimension grid. For example, assuming A_1 is the hyper-region formed by starting at corner point $(0, 0, ..., 0)$ and moving towards the corner point $(n, n, .., n)$, the table for A_1 can be computed using the following recurrence relationship:

$$|A_1(i, j, .., k)| = |A_1(i, j-1, ..., k)| + |A_1(i-1, j, .., k)| + ... + |A_1(i, j, ..., k-1)| - |A_1(i-1, j-1, ..., k-1)| + |G(i, j)| \quad (14)$$

where $|G(i, j)|$ counts the element in the cell location (i, j). The first element and the first array along each coordinate need to be populated (initialized) so that all cardinalities of A_1 can be computed. The counts in the remaining coordinate are accumulated through the dependency of the previous array for each coordinate.

Similarly, the computation of other sets are listed as the following. For example, A_2 is formed by starting at corner point $(n, n, ..., n)$ and moving towards the corner point $(0, 0, ..., 0)$.

$$|A_2(i, j, ..., k)| = |A_2(i+1, j, .., k)| + |A_2(i, j+1, ..., k)| + ... + |A_2(i, j, ..., k+1)| - |A_2(i+1, j+1, ..., k+1)| + |G(i, j)| \quad (15)$$

Generally, depending on the coordinate direction to form a hyper-region, the previous array coordinate is obtained by subtracting 1 if the direction is towards n of the corner point of grid such as $(0, ..., n.., 0)$. Otherwise the previous array coordinate is obtained by adding 1 if the direction is towards 0 of the corner point of grid such as $(0, ..., n.., 0)$. For example, if A_t is formed by starting at corner point $(0, ..., n)$ and moving towards the corner point $(n, ..., 0)$ and we assume first coordinate and last coordinate starting from $(0, n)$ towards $(n, 0)$, the computation is shown as:

$$|A_t(i, ..., k)| = |A_t(i, ..., k-1)| + ... + |A_t(i+1, ..., k)| - |A_t(i+1, ..., k-1)| + |G(i, j)| \quad (16)$$

By applying Inclusive/Exclusive principle, the pre-computation strategy makes the computation complexity in multi-dimensional grid can be reduced from $O(n^{k^2})$ to $O(1)$ if we create above n^k tables on CPU and execute LRT computation on parallel architecture.

5 | RANGE MAPPING SCHEME

In this section, we study how parallelism helps to enumerate all of the rectangular regions (R) in a spatial grid (G). We will explain this scheme in one-dimensional, two-dimensional, and k-dimensional data-grids.

From section 3.1, we know that if all of the pairwise intervals between $n + 1$ data points, denoted as $l_{(i,j)}$, are plotted as point (i, j) 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" (PU) 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.

Figure 7 a plots out all of the intervals $l_{(i,j)}$ in two-dimensional space and we can see that these points (i, j) forms a triangle. Figure 7 b shows the rectangular-shaped space after transformation.

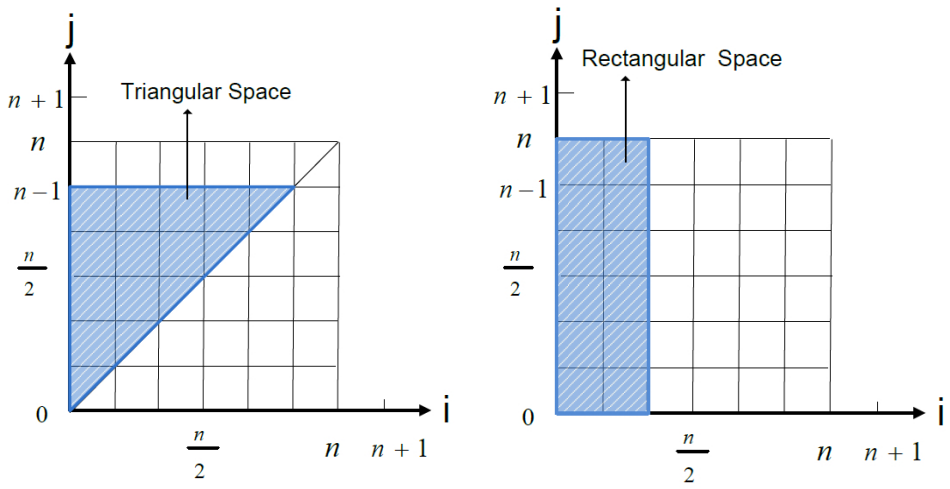


FIGURE 7 1-d Interval Transformation from $[n, n]$ to $[(n + 1)/2, n]$

To assign an interval $l_(i, j)$ to a PU_y, Horner scheme discussed above is applied (12). Then transform (i, j) from rectangular space to (i', j') to triangle space. We give details in the following:

5.1 | One-Dimensional Mapping

For one-dimensional range mapping scheme, we assume there are n data points, therefore having $\frac{n\cdot(n+1)}{2}$ intervals. There are two cases in one-dimensional grid: (1) The grid size n is even (2) The grid size n is odd.

Given the following information:

$$N = \{0, ..., n\} \tag{17}$$

$$R = \{(i, j) \in [0, ..., \lfloor \tfrac{n+1}{2} \rfloor - 1] \times [0, ..., n]\} \tag{18}$$

$$T = \{(i', j') \in [0, ..., n] \times [0, ..., n] | i' <= j'\} \tag{19}$$

$$M = \{0, ..., \lfloor \tfrac{n+1}{2} \rfloor \cdot (n + 1) - 1\} \tag{20}$$

$$\tag{21}$$

where N is the set of $n + 1$ data points which partitions one-dimensional grid into n intervals. R is the rectangular space formed by $\{[0, ..., \lfloor \frac{n+1}{2} \rfloor - 1] \times [0, ..., n]\}$ and each point (i, j) (i.e. interval) in this space is assigned to a certain PU_y. T is the triangular space formed by original intervals, where each interval is $l_(i', j')$ $(0 < i' < j' \leq n)$. M is total number of intervals for computing LRT.

To have a generalized form of the mapping scheme when grid size n is even or odd , we extend the calculation of total intervals from $\frac{n+1}{2} \cdot n$ to $\lfloor \frac{n+1}{2} \rfloor \cdot (n + 1)$. When n is even, we know $\lfloor \frac{n+1}{2} \rfloor = \frac{n}{2}$. When n is odd, we know $\lfloor \frac{n+1}{2} \rfloor = \frac{n+1}{2}$. The generalized computation of number of total intervals in our range mapping scheme is $\frac{n+1}{2} \cdot (n + 1)$ no matter grid size n is even or odd, this produces $\frac{n+1}{2}$ more intervals with enlarged set when n is odd but have exact calculation when n is even.

(1) The grid size n is even: A mapping Function $\phi : M \to R \to T$ is composed of two functions: $\phi_1 : M \to R$ and $\phi_2 : R \to T$. Firstly, ϕ_1 transforms the index value of $y \in M$ of a certain PU_y to a pair $(i, j) \in R$ in rectangular space. Secondly, ϕ_2 transforms the pair $(i, j) \in R$ from rectangular space to a pair $(i', j') \in T$ to triangular space.

Definition 1. IR Mapping Function

$$\begin{aligned} \phi_1 : & M \to R \\ & y \mapsto (i, j) \end{aligned}$$

where $i = y/(n+1)$ and $j = y \mod (n+1)$.

IR (Index to rectangular) mapping function assigns a point (i, j) in rectangular space R to y of PU_y by applying inverse horner scheme, where $y \in \{0, ..., \lfloor \frac{n+1}{2} \rfloor \cdot (n+1) - 1\}$. When n is even, $\lfloor \frac{n+1}{2} \rfloor = \frac{n}{2}$. Point (i, j) is computed by $i = y/(n+1)$ and $j = y \mod (n+1)$, where y is the index of a particular PU and constant value of n is the total number of data points in one-dimensional grid.

Lemma 1. *Mapping function ϕ_1 transforms $y \in M$ (i.e. $y \in \{0, ..., \frac{n \cdot (n+1)}{2} - 1\}$) to $(i, j) \in R$, where $i \in \{0, ..., \frac{n}{2} - 1\}$ and $j \in \{0, ..., n\}$.*

When n is even, we know $\lfloor \frac{n+1}{2} \rfloor = \frac{n}{2}$. Since $y \in M$ and size of M is equal to $\frac{n}{2} \cdot n$, it is obviously seen that $i \in \{0, ..., \frac{n}{2} - 1\}$ when $i = y/(n+1)$. Similarly, $j \in \{0, ..., n\}$ when $j = y \mod (n+1)$.

Lemma 2. *Mapping function $\phi_1 : M \to R, y \mapsto (i, j)$ is bijective.*

In one-dimension grid, for each PU_y, we can denote $y = i \cdot n + j$, where $y \in \{0, ..., \frac{n \cdot (n+1)}{2}\}$ and $0 \leq i, j < n$. From the horner scheme, we know lemma 1 produces one-to-one mapping from y to (i, j). At the same time, given (i, j), y can be uniquely constructed as $y = i \cdot n + j$. Therefore, $y \mapsto (i, j)$ is bijective.

Definition 2. RT Mapping Function

$$\phi_2 : \ R \to T \tag{22}$$

$$(i, j) \ \mapsto (i', j') = \begin{cases} (i, i + j) & \text{if } (i + j) < n \\ (n - i - 1, 2n - i - j - 1)) & \text{otherwise} \end{cases} \tag{23}$$

RT (rectangular to triangular) mapping function transforms point (i, j) in rectangular space to point (i', j') in triangular space shown in above equation. This transformation is achieved by satisfying the constraints: $(i, j) \in \{[0, ..., \lfloor (n+1)/2 \rfloor] \times [0, ..., n]\}$ and $0 \leq i' \leq j' < n$. To fulfill this constraint, line $i + j = n$ divide the rectangular space into two sub-spaces: (a) $i + j < n$, (b) $i + j \geq n$. In (a), transformed point (i', j') is calculated as $(i, i + j)$. In (b), transformed point (i', j') is calculated as $(n - i - 1, (n - i - 1) + (n - j))$ (i.e. $(n - i - 1, 2n - i - j - 1)$). It is clearly seen that (i', j') satisfies $0 \leq i' \leq j' < n$ after transformation. Figure 8 a visualizes the mapping.

Lemma 3. *Mapping function ϕ_2 transform $(i, j) \in R$ to $(i', j') \in T$, where $i' \in \{0, ..., n\}$ and $j' \in \{0, ..., n\}$.*

It is true that $i < n$ and $n - i - 1 < n$. Also when $(i + j) < n$ and $j' = i + j$, we can get $i' \leq j' < n$. When $(i + j) \geq n$, we get $i' \leq j' = 2n - i - j - 1 = (n - i) + (n - j - 1) < n$. Therefore,$i' \in \{0, ..., n\}$, $j' \in \{0, ..., n\}$ and $i' \leq j'$.

Lemma 4. *Mapping function $\phi_2 : R \to T, y \mapsto (i, j)$ is bijective.*

From above definition, we see that each (i, j) uniquely maps to (i', j'). We get $i' = i, j' = i + j$ when $i + j < n$. And we get $i' = n - i - 1, j' = 2n - i - j - 1$ when $i + j \geq n$. Correspondingly, given (i', j'), (i, j) can be uniquely obtained by $i = i', j = j' - i'$ or $i = n - i' - 1, j = i' + n - j'$. Therefore, $y \mapsto (i, j)$ is bijective.

Corollary 1. *The function $\phi : M \to R \to T, y \mapsto (i', j')$ is bijective.*

Since $M \to R$ and $R \to T$ are bijective respectively, this means $y \mapsto (i,j)$ and $(i,j) \mapsto (i',j')$ are bijective. Therefore $M \to R \to T, y \mapsto (i',j')$ is bijective.

(2) The grid size n is odd: Function $\phi : M \to R \to T$ is composed of two mapping functions: $\phi_1 : M \to R$ and $\phi_2 : R \to T$. Firstly, ϕ_1 transforms the index value of $y \in M$ of a certain PU_y to a pair $(i,j) \in R$ in rectangular space. Secondly, ϕ_2 transforms $(i,j) \in R$ from rectangular space to a pair $(i',j') \in T$ in triangular space.

Definition 3. IR Mapping Function

$$\phi_1 : M \to R \tag{24}$$

$$y \mapsto (i,j) \tag{25}$$

where $i = y/(n+1)$ and $j = y \mod (n+1)$.
IR (Index to rectangular) mapping function assigns a point (i,j) in rectangular space R to y of PU_y by applying inverse horner scheme, where $y \in \{0, ..., \lfloor \frac{n+1}{2} \rfloor \cdot (n+1) - 1\}$. Point (i,j) is computed by $i = y/(n+1)$ and $j = y \mod (n+1)$, where y is the index of a particular PU and constant value of n is the total number of data points in one-dimensional grid. Since n is odd in this case, there has $\frac{n+1}{2}$ more data points (intervals) in the extended form than the original total number of intervals and they won't be processed.

Lemma 5. *Mapping function ϕ_1 transforms $y \in M$ to $(i,j) \in R$, where $i \in \{0, ..., \frac{n+1}{2} - 1\}$ and $j \in \{0, ..., n\}$.*

When n is odd, we know $\lfloor \frac{n+1}{2} \rfloor = \frac{n+1}{2}$. Since $y \in M$ and size of M is extend to $\frac{n+1}{2} \cdot (n+1)$, we can derive that $i \in \{0, ..., \frac{n+1}{2} - 1\}$ when $i = y/(n+1)$. Similarly, $j \in (0, n)$ when $j = y \mod (n+1)$.

Lemma 6. *Mapping function $\phi_1 : M \to R, y \mapsto (i,j)$ is bijective.*

In one-dimension grid, for each PU_y, we can denote $y = i \cdot (n+1) + j$, where $y \in (0, \frac{(n+1)\cdot(n+1)}{2})$ and $0 \le i, j < n$. From the horner scheme and the quotient-remainder theorem , we know there is unique mapping from y to (i,j). Correspondingly, given (i,j), y can be uniquely constructed as $y = i \cdot (n+1) + j$. Therefore, $y \mapsto (i,j)$ is bijective.

Definition 4. Mapping Function

$$\phi_2 : \ R \to T \tag{26}$$

$$(i,j) \ \mapsto (i',j') = \begin{cases} (i, i+j) & \text{if } (i+j) < n \\ (n-i-1, 2n-i-j-1) & \text{if} (i+1) \le \frac{(n+1)}{2} \end{cases} \tag{27}$$

RT (rectangular to triangular) mapping function transforms point (i,j) in rectangular space to point (i',j') in triangular space shown in above equation. This transformation is achieved by satisfying the constraints: $(i,j) \in \{[0, ..., \lfloor (n+1)/2 \rfloor \times [0, ..., n+1]\}$ and $0 \le i' \le j' < n$. To fulfill this constraint, line $i + j = n$ divide the rectangular space into two sub-spaces: (a) $i + j < n$, (b) $i + j \ge n$. Furthermore, when n is odd, we have $\frac{(n+1)}{2}$ more points than needed and thus the rectangular shape is enlarged, the constraint of $(i+1) \le \frac{(n+1)}{2}$ is applied to limit the unnecessary transformation of points in rectangular space to triangular space. To explain this, when line $i + j = n$ intersection with line $(i+1) = \frac{(n+1)}{2}$, we can get $j = n - i = n - \frac{(n+1)}{2} + 1 = \frac{(n+1)}{2}$. Hence when $i + j < n$, the unnecessary points above the the intersection point on the line of $(i+1) = \frac{(n+1)}{2}$ are filtered out. In (a), transformed point (i',j') is calculated as $(i, i+j)$. In (b), transformed point (i',j') is calculated as $(n-i-1, (n-i-1) + (n-j))$ (i.e. $(n-i-1, 2n-i-j-1)$). This mapping strategy shows that (i',j') satisfies $0 \le i' \le j' < n$ after transformation. Figure 8 b visualizes the mapping.

Lemma 7. *Mapping function $\phi_2 : R \to T$ transform $(i,j) \in R$ to $(i',j') \in T$, where $i' \in \{0, ..., n\}$ and $j' \in \{0, ..., n\}$ in triangular space.*

It is true that $i < n$ and $n - i - 1 < n$. Also when $(i+j) < n$ and $j' = i + j$, we can get $i' \le j' < n$. When $(i+j) \ge n$, we get $i' \le j' = 2n - i - j - 1 = (n-i) + (n-j-1) < n$. Moreover, the constraint of $(i+1) \le \frac{(n+1)}{2}$ is applied to limit the unnecessary transformation of points when n is odd. Therefore,$i' \in \{0, ..., n\}$, $j' \in \{0, ..., n\}$ and $i' \le j'$ in triangular space.

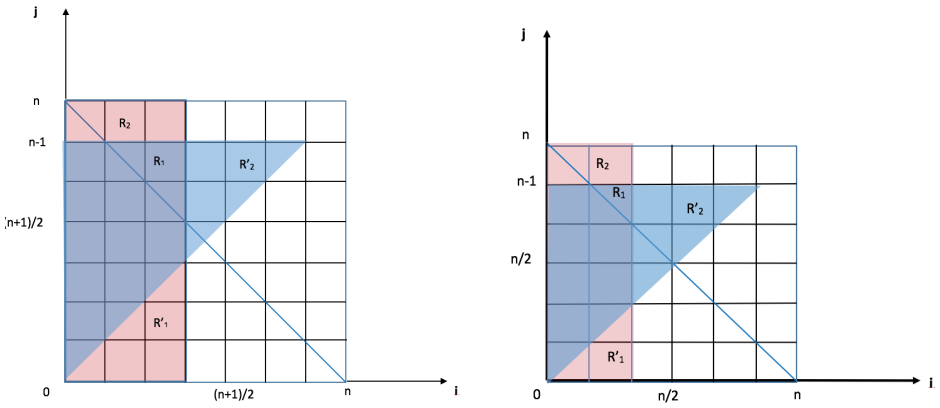
Lemma 8. *Mapping function $\phi_2 : R \to T, (i,j) \mapsto (i',j')$ is injective.*

From above mapping strategy, we see that each (i',j') has single mapping to (i,j) and some unnecessary (i,j) are not processed due to R is enlarged. There has $i' = i, j' = i+j$ when $i+j < n$. And there has $i' = n-i-1, j' = 2n-i-j-1$ when $i+j \geq n$. There is no overlapping mapped point (i',j') based on the constraint of $i+j < n$ and $(i+1) \leq \frac{(n+1)}{2}$. Correspondingly, given (i',j'), (i,j) can be obtained by $i = i', j = j'-i'$ or $i = n-i'-1, j = i'+n-j'$. However, due to R is enlarged space , $(i,j) \mapsto (i',j')$ is injective when n is odd.

Corollary 2. *Mapping function $\phi : M \to R \to T, y \mapsto (i',j')$ is injective.*

From above, we know $M \to R$ (i.e. $y \mapsto (i,j)$) is bijective and $R \to T$ (i.e. $(i,j) \mapsto (i',j')$) is injective. Therefore $M \to R \to T, y \mapsto (i',j')$ is injective.

Figure 8 a and Figure 8 b show the solutions and implementation is shown in Algorithm 2.



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

FIGURE 8 1-d Interval Transformation from $[n,n]$ to $[\lfloor (n+1)/2 \rfloor, n+1]$

5.2 | Two-Dimensional Mapping

Two-dimensional mapping is extended from one-dimensional mapping directly.
Given:

$$N^2 = N_1 \times N_2 = \{0..n_1\} \times \{0..n_2\} \tag{28}$$

$$R^2 = R_1 \times R_2 = \{(i_1,j_1),(i_2,j_2)\} \in \{[0..\lfloor \tfrac{n_1+1}{2} \rfloor - 1] \times [0..n_1], [0..\lfloor \tfrac{n_2+1}{2} \rfloor - 1] \times [0..n_2]\} \tag{29}$$

$$T^2 = T_1 \times T_2 = \{(i'_1,j'_1),(i'_2,j'_2)\} \in \{[0,...,n_1]^2, [0,...,n_2]^2\} | i'_{1,2} <= j'_{1,2} \tag{30}$$

$$M^2 = \{0..\lfloor \tfrac{n_1+1)}{2} \rfloor \cdot \lfloor \tfrac{n_2+1}{2} \rfloor \cdot (n_1+1) \cdot (n_2+1)\} \tag{31}$$

$$\tag{32}$$

where $\{N_1, N_2\}$ are set of two dimensional data points with size of (n_1+1, n_2+1) to partition the data grid into $n_1 \cdot n_2$ independent cells. $\{R_1, R_2\}$ is rectangular space formed by cell $\{(i_1,j_1),(i_2,j_2)\}$ and each such cell is assigned to a certain PU_y ($y \in M^2$). $\{T_1, T_2\}$ is triangular space formed by original cell $\{(i'_1,j'_1),(i'_2,j'_2)\}$, $(0 < i'_1 < j'_1 \leq n_1, 0 < i'_2 < j'_2 \leq n_2\})$in two dimensional grid. M^2 is the total number of cells for computing LRT in two-dimension grid. Similar as one dimensional grid, we generalize the range mapping scheme by extending the calculation of total cells to $\lfloor \frac{n_1+1)}{2} \rfloor \cdot \lfloor \frac{n_2+1}{2} \rfloor \cdot (n_1+1) \cdot (n_2+1)$ when either n_1 or n_2 is odd or even. In the following definitions and lemmas, we won't differentiate the scenario of even and odd number of partitions since we already shown the capability of our generalized form in one dimensional grid.

Definition 5. IR Mapping Function

$$\phi_1 : M^2 \to R^2$$
$$y \mapsto (i_1, j_1) \times (i_2, j_2)$$

where $i_1 \in \{0, ..., \lfloor \frac{n_1+1)}{2} \rfloor - 1\}$, $j_1 \in \{0, ..., n_1\}$ and $i_2 \in \{0, ..., \lfloor \frac{n_2+1)}{2} \rfloor - 1\}$, $j_2 \in \{0, ..., n_2\}$.

IR (Index to rectangular) mapping function assigns a cell denoted as $\{(i_1, j_1), (i_2, j_2)\}$ in rectangular space R^2 to y of PU_y by applying inverse horner scheme, where y is the index of a particular PU and constant value of $n_1 \cdot n_2$ is the number of independent cells partitioned in two-dimensional grid. Cell $\{(i_1, j_1), (i_2, j_2)\}$ is a region (i_1, j_1, i_2, j_2) defined in our pre-computing section **??** , where (i_1, j_1) is the upper left corner and (i_2, j_2) is the lower right corner.

The mapping function is performed in the following steps: (1) $i_1 = y mod \frac{n_1+1}{2}$; (2) $y = y/\frac{n_1+1}{2}$; (3) $j_1 = y mod(n_1 + 1)$; (4) $y = y/(n_1 + 1)$; (5) $i_2 = y mod \frac{n_2+1}{2}$; (6) $j_2 = y/(n_2 + 1)$

From above, we know that y is in the range of $\{0..\lfloor \frac{n_1+1)}{2} \rfloor \cdot \lfloor \frac{n_2+1}{2} \rfloor \cdot (n_1 + 1) \cdot (n_2 + 1)\}$, by performing the first step (1) using horner scheme , i_1 is generated in the range of $i_1 \in \{0, ..., \lfloor \frac{n_1+1)}{2} \rfloor - 1\}$. And j_1 is generated in the range of $\{0, ..., n_1\}$. In the step (4), y is recalculated in the range of $\{0, ..., \lfloor \frac{n_2+1)}{2} \rfloor \cdot (n_2 + 1)\}$ after we obtain (i_1, j_1). Similarly, i_2 is generated in the range of $i_2 \in \{0, ..., \lfloor \frac{n_2+1)}{2} \rfloor - 1\}$ and j_2 is generated in the range of $\{0, ..., n_2\}$, which is same as the mapping process in one dimensional grid.

Lemma 9. *Mapping function $\phi_1 : M^2 \to R^2, y \mapsto (i_1, j_1, i_2, j_2)$ is bijective.*

In two-dimension grid, for each PU_y, we can denote $y = (i_2 \cdot n_2 + j_2) \cdot i_1 \cdot n_1 + j_1$, where $y \in \{0..\lfloor \frac{n_1+1)}{2} \rfloor \cdot \lfloor \frac{n_2+1}{2} \rfloor \cdot (n_1 + 1) \cdot (n_2 + 1)\}$ with $0 \le i_1, j_1 < n_1$ and $0 \le i_2, j_2 < n_2$. From the horner scheme , we know i_1 and j_1 can be uniquely obtained by given n_1 through computing the above step (1), (2) and (3). Similarly, i_1 and j_1 are obtained by give n_2 through step (4), (5) and (6). This mapping produces one-to-one mapping from y to (i_1, j_1, i_2, j_2). At the same time, given (i_1, j_1, i_2, j_2) , y can be uniquely constructed as $y = (i_2 \cdot n_2 + j_2) \cdot i_1 \cdot n_1 + j_1$. Therefore, $y \mapsto (i_1, j_1, i_2, j_2)$ is bijective.

Definition 6. RT Mapping Function

$$\phi_2 : \quad R^2 \to T^2 \tag{33}$$

$$(i_1, j_1, i_2, j_2) \mapsto (i'_1, j'_1, i'_2, j'_2) = \begin{cases} (i_1, i_1 + j_1) & \text{if } (i_1 + j_1) < n_1 \\ (n_1 - i_1 - 1, 2n_1 - i_1 - j_1 - 1)) & \text{if } (i_1 + 1) \le \frac{n_1+1}{2} \\ (i_2, i_2 + j_2) & \text{if } (i_2 + j_2) < n_2 \\ (n_2 - i_2 - 1, 2n_2 - i_2 - j_2 - 1)) & \text{if } (i_2 + 1) \le \frac{n_2+1}{2} \end{cases} \tag{34}$$

RT (rectangular to triangular) mapping function transforms cell (i_1, j_1, i_2, j_2) in rectangular space to cell (i'_1, j'_1, i'_2, j'_2) in triangular space shown in above equation. This transformation is achieved by satisfying the constraints: $(i_1, j_1) \in \{[0, ..., \lfloor (n_1+1)/2 \rfloor \times [0, ..., n_1]\}$,$(i_1, j_1) \in \{[0, ..., \lfloor (n_2+1)/2 \rfloor \times [0, ..., n_2]\}$, $0 \le i'_1 \le j'_1 < n_1$ and $0 \le i'_2 \le j'_2 < n_2$. To fulfill this constraint, hyperplane $i_1 + j_1 = n_1$ and $i_2 + j_2 = n_2$ divide the rectangular space into sub-spaces on two dimensional coordinate system respectively : (a) $i_1 + j_1 < n_1$ and (b) $i_1 + j_1 \ge n_1$;(a') $i_2 + j_2 < n_2$ and (b') $i_2 + j_2 \ge n_2$. In (a) and (a'), transformed point (i'_1, j'_1, i'_2, j'_2) is calculated as $(i_1, i_1 + j_1, i_2, i_2 + j_2)$. In (b) and (b'), transformed point (i'_1, j'_1, i'_2, j'_2) is calculated as $(n_1 - i_1 - 1, (n_1 - i_1 - 1) + (n_1 - j_1))$ (i.e. $(n_1 - i_1 - 1, 2n_1 - i_1 - j_1 - 1)$) and $(n_2 - i_2 - 1, (n_2 - i_2 - 1) + (n_2 - j_2))$ (i.e. $(n_2 - i_2 - 1, 2n_2 - i_2 - j_2 - 1)$). It is clearly seen that (i'_1, j'_1, i'_2, j'_2) satisfies $0 \le i'_1 \le j'_1 < n_1$ and $0 \le i'_2 \le j'_2 < n_2$ after transformation.

Lemma 10. *Mapping function ϕ_2 transform $(i_1, j_1, i_2, j_2) \in R^2$ to $(i'_1, j'_1, i'_2, j'_2) \in T^2$, where $i'_1 \in \{0, ..., n_1\}$, $j'_1 \in \{0, ..., n_1\}$, $i'_2 \in \{0, ..., n_2\}$ and $j'_2 \in \{0, ..., n_2\}$.*

It is true that $i_1 < n$ and $n_1 - i_1 - 1 < n_1$. Also when $(i_1 + j_1) < n_1$ and $j'_1 = i_1 + j_1$, we can get $i'_1 \le j'_1 < n_1$. When $(i_1 + j_1) \ge n_1$, we get $i'_1 \le j'_1 = 2n_1 - i_1 - j_1 - 1 = (n_1 - i_1) + (n_1 - j_1 - 1) < n_1$. Therefore,$i'_1 \in \{0, ..., n_1\}$, $j'_1 \in \{0, ..., n_1\}$ and $i'_1 \le j'_1$. We can derive $i'_2 \in \{0, ..., n_2\}$ and $j'_2 \in \{0, ..., n_2\}$ similarly.

Lemma 11. *Mapping function $\phi_2 : R^2 \to T^2, (i_1, j_1, i_2, j_2) \mapsto (i'_1, j'_1, i'_2, j'_2)$ is injective.*

Here we deduce from (i_1, j_1) to (i'_1, j'_1) is injective, then similarly we can from (i_2, j_2) to (i'_2, j'_2) is injective. In one dimensional situation, We know that each (i'_1, j'_1) has single mapping to (i_1, j_1) and some unnecessary (i_1, j_1) are not processed due to R^1 is enlarged. We have $i'_1 = i_1, j'_1 = i_1 + j_1$ when $i_1 + j_1 < n_1$. Also $i'_1 = n_1 - i_1 - 1, j'_1 = 2n_1 - i_1 - j_1 - 1$ when $i_1 + j_1 \ge n_1$. There is no over-lapping mapped point (i'_1, j'_1)

based on the constraint of $i_1 + j_1 < n_1$ and $(i_1 + 1) \leq \frac{(n_1+1)}{2}$. Correspondingly, given (i'_1, j'_1), (i_1, j_1) can be obtained by $i_1 = i'_1, j_1 = j'_1 - i'_1$ or $i_1 = n_1 - i'_1 - 1, j_1 = i'_1 + n_1 - j'_1$. However, due to R^1 is enlarged space , $(i_1, j_1) \mapsto (i'_1, j'_1)$ is injective. Similarly, $(i_2, j_2) \mapsto (i'_2, j'_2)$ is injective. Therefore, mapping function $\phi_2 : R^2 \to T^2, (i_1, j_1, i_2, j_2) \mapsto (i'_1, j'_1, i'_2, j'_2)$ is injective.

Corollary 3. *Mapping function $\phi : M^2 \to R^2 \to T^2, y \mapsto (i'_1, j'_1, i'_2, j'_2)$ is injective.*

From above, we know $M^2 \to R^2$ (i.e. $y \mapsto (i_1, j_1, i_2, j_2)$) is bijective and $R^2 \to T^2$ (i.e. $(i_1, j_1, i_2, j_2) \mapsto (i'_1, j'_1, i'_2, j'_2)$) is injective. Therefore $M^2 \to R^2 \to T^2, y \mapsto (i'_1, j'_1, i'_2, j'_2)$ is injective.

5.3 | K-Dimensional Mapping

Similarly, in K-dimension, given:

$$N^k = N_1 ... \times N_k = \{0..n_1\} \times ... \times \{0..n_k\} \tag{35}$$

$$R^k = R_1 \times ... \times R_k = \{(i_1, j_1), ..., (i_k, j_k)\} \in \{[0..\lfloor \tfrac{n_1+1}{2} \rfloor - 1] \times [0..n_1], \times [0..n_k]\} \tag{36}$$

$$T^k = T_1 \times ... \times T_k = \{(i'_1, j'_1), .., (i'_k, j'_k)\} \in \{[0, ..., n_1]^2, ..., [0, ..., n_k]^2\} | i'_{1,..,k} <= j'_{1,...,k} \tag{37}$$

$$M^k = \{0..\lfloor \tfrac{n_1+1)}{2} \rfloor ... \cdot \lfloor \tfrac{n_k+1}{2} \rfloor \cdot (n_1 + 1) ... \cdot (n_k + 1)\} \tag{38}$$

$$\tag{39}$$

where $\{N_1, ..., N_k\}$ are set of k dimensional data points with size of $(n_1 + 1, ..., n_k + 1)$ to partition the data grid into $n_1 \cdot ... \cdot n_k$ independent k-dimensional hypecubes. $\{R_1, ..., R_k\}$ is hyperectangular space formed by hypecube $\{(i_1, j_1, ..., i_k, j_k)\}$ and each such hypecube is assigned to a certain PU_y ($y \in M^k$). $\{T_1, ..., T_k\}$ is hypetriangular space formed by original hypecube $\{(i'_1, j'_1, ..., i'_k, j'_k)\}$, ($0 < i'_1 < j'_1 \leq n_1, ..., 0 < i'_k < j'_k \leq n_k$})in k dimensional grid. M^k is the total number of cells for computing LRT in k-dimension grid. Similar as one dimensional grid, we generalize the range mapping scheme by extending the calculation of total cells to $\lfloor \frac{n_1+1)}{2} \rfloor \cdot ... \lfloor \frac{n_k+1}{2} \rfloor \cdot (n_1 + 1) ... \cdot (n_k + 1)$ when $n_1, ..., n_k$ is odd or even. Same as LRT calculation for two dimensional grid, in the following definitions and lemmas, we won't differentiate the scenario of even and odd number of partitions and won't provide proof.

Definition 7. IR Mapping Function

$$\begin{aligned} \phi_1 : & M^k \to R^k \\ & y \mapsto (i_1, j_1) \times ... \times (i_k, j_k) \end{aligned}$$

where $i_1 \in \{0, ..., \lfloor \frac{n_1+1)}{2} \rfloor - 1\}, j_1 \in \{0, ..., n_1\}$ ...,$i_k \in \{0, ..., \lfloor \frac{n_k+1)}{2} \rfloor - 1\}, j_k \in \{0, ..., n_k\}$.

IR (Index to rectangular) mapping function assigns a hypecube denoted as $\{(i_1, j_1, ..., i_k, j_k)\}$ in hyperectangular space R^k to y of PU_y by applying inverse horner scheme, where y is the index of a particular PU and constant value of $n_1 \cdot ... \cdot n_k$ is the number of independent hypecubes partitioned in k-dimensional grid.

Lemma 12. *Mapping function $\phi_1 : M^2 \to R^2, y \mapsto (i_1, j_1, i_2, j_2)$ is bijective.*

Definition 8. RT Mapping Function

$$\phi_2 : \quad R^k \to T^k \tag{40}$$

$$(i_1, j_1, ..., i_k, j_k) \mapsto (i'_1, j'_1, ..., i'_k, j'_k) = \begin{cases} (i_1, i_1 + j_1) & \text{if } (i_1 + j_1) < n_1 \\ (n_1 - i_1 - 1, 2n_1 - i_1 - j_1 - 1)) & \text{if } (i_1 + 1) \leq \frac{n_1+1}{2} \\ ...(i_k, i_k + j_k) & \text{if } (i_k + j_k) < n_k \\ (n_k - i_k - 1, 2n_k - i_k - j_k - 1)) & \text{if } (i_k + 1) \leq \frac{n_k+1}{2} \end{cases} \tag{41}$$

RT (rectangular to triangular) mapping function transforms hypecube $(i_1, j_1, ..., i_k, j_k)$ in hyperrectangular space to $(i'_1, j'_1, ..., i'_k, j'_k)$ in hypetriangular space shown in above equation. This transformation is achieved by satisfying the constraints: $(i_1, j_1) \in \{[0, ..., \lfloor (n_1 + 1)/2 \rfloor \times [0, ..., n_1]\}$,...,$(i_k, j_k) \in \{[0, ..., \lfloor (n_k+1)/2 \rfloor \times [0, ..., n_k]\}, 0 \leq i'_1 \leq j'_1 < n_1$,..., $0 \leq i'_k \leq j'_k < n_k$. To fulfill this constraint, hyperplane $i_1 + j_1 = n_1$, ..., $i_k + j_k = n_k$ divide the hyperectangular space into sub-spaces on k dimensional coordinate system respectively : (a) $i_1 + j_1 < n_1$ and (b) $i_1 + j_1 \geq n_1$;...;(a^k) $i_k + j_k < n_k$ and (b^k) $i_k + j_k \geq n_k$. In (a),..., (a^k), transformed point $(i'_1, j'_1, ..., i'_k, j'_k)$ is calculated as $(i_1, i_1 + j_1, ..., i_k, i_k + j_k)$. In (b),..., (b^k), transformed point $(i'_1, j'_1, ..., i'_k, j'_k)$ is calculated as $(n_1 - i_1 - 1, (n_1 - i_1 - 1) + (n_1 - j_1))$ (i.e. $(n_1 - i_1 - 1, 2n_1 - i_1 - j_1 - 1)$) ,...,

Algorithm 2 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:            end if
12:        end if
13:        LRT computation for interval (iı,jı)
14:    end for
15: end for
```

$(n_k - i_k - 1, (n_k - i_k - 1) + (n_k - j_k))$ (i.e. $(n_k - i_k - 1, 2n_k - i_k - j_k - 1)$). It is clearly seen that $(i'_1, j'_1, ..., i'_k, j'_k)$ satisfies $0 \leq i'_1 \leq j'_1 < n_1$,..., $0 \leq i'_k \leq j'_k < n_k$ after transformation.

Lemma 13. *Mapping function ϕ_2 transform $(i_1, j_1, ..., i_k, j_k) \in R^k$ to $(i'_1, j'_1, ..., i'_k, j'_k) \in T^k$, where $i'_1 \in \{0, ..., n_1\}$, $j'_1 \in \{0, ..., n_1\}$, ...,$i'_k \in \{0, ..., n_k\}$ and $j'_k \in \{0, ..., n_k\}$.*

Lemma 14. *Mapping function $\phi_2 : R^k \to T^k, (i_1, j_1, ..., i_k, j_k) \mapsto (i'_1, j'_1, ..., i'_k, j'_k)$ is injective.*

Corollary 4. *Mapping function $\phi : M^k \to R^k \to T^k, y \mapsto (i'_1, j'_1, ..., i'_k, j'_k)$ is injective.*

6 | K-BEST REDUCTION SCHEME

To find top-k[5] anomalous rectangular regions, a heap with maximium size of k is built from each "parallel computing component" (PU). A further reduction strategy is applied on these k heaps to get final k-best regions.

A LRT value set, denoted as $s = \{s_1, s_2, .., s_i, ..., s_n\}$, is generated from n rectangular regions set R. $\{R\}$ is divided into t equal portions: $\{p_1, p_2, .., p_i, ..., p_t\}$, where $1 \leq t \leq n$. Each portion is processed in parallel and a k-best result with heap structure is generated correspondingly. The k-best result from each portion is $p_{1_k}, p_{2_k}, ..., p_{i_k}, ..p_{t_k}$.

We denote the process of finding k-best using heap sort as $h()$. We also denote the k-best result from original data grid is $\{s_{r0}, s_{r1}, ..., s_{rk}\}$. And the k-best value from each parallel portion p_i is $\{s_{i0}, s_{i1}, ..., s_{ik}\}$. And we assume the k-best value are in ascending order. For example, $s_{r0} \leq s_{r1}, ..., \leq s_{rk}, s_{i0} \leq s_{i1}, ..., \leq s_{ik}$.

Lemma 15. $KBestReduction$: *The k-best LRT values from the value set s is equal to the k-best values obtained by performing k-best reduction from each parallel portion: $h(p_1 \cup p_2 \cup ... \cup p_i \cup ... \cup p_n) = h(h(p_1) \cup h(p_2) \cup ... \cup h(p_n))$.*

We give proof by contradiction:

$(1) \forall x \in h(p_1 \cup p_2 \cup ... \cup p_i \cup ... \cup p_n)$

We have

$$x \in (h(p_1) \cup h(p_2) \cup ... \cup h(p_n)), \tag{42}$$

$$x \in h(h(p_1) \cup h(p_2) \cup ... \cup h(p_n)) \tag{43}$$

[5] Top-k and k-best are exchangeable terms.

Algorithm 3 Naive top-k LRT search

Input: data grid (G(n,n)), k

Output: top k anomalous regions

```
1: //Search each rectangle R (x1,y1,x2,y2) in G (n,n)
2: for x1 ←0 to n do
3:     for y1 ←0 to n do
4:         for x2 ←x1 to n do
5:             for y2 ←y1 to n do
6:                 score←lrt(x1,y1,x2,y2)
7:             end for
8:         end for
9:     end for
10: end for
11: Sorting the scores and return top-k regions R
```

Because the total number of k-best values: $|(h(p_1) \cup h(p_2) \cup ... \cup h(p_n)| \geq k$,
and $x \notin h(p_1) \cup h(p_2) \cup ... \cup h(p_n) \rightarrow x \notin h(p_1 \cup p_2 \cup ... \cup p_i \cup ... \cup p_n)$.
and x is one of the topk value from all the results
Therefore, $x \in h((h(p_1) \cup h(p_2) \cup ... \cup h(p_n)))$
$(2) \forall x \in h(p_1 \cup p_2 \cup ... \cup p_i \cup ... \cup p_n)$
We have

$$x \in h(p_1) \cup h(p_2) \cup ... \cup h(p_n), \tag{44}$$

$$x \in h(p_1 \cup p_2 \cup ... \cup p_i \cup ... \cup p_n) \tag{45}$$

Because $h(h(p_1) \cup h(p_2) \cup ... \cup h(p_n)) \subseteq h(p_1) \cup h(p_2) \cup ... \cup h(p_n) \subseteq h(p_1 \cup p_2 \cup ... \cup p_i \cup ... \cup p_n)$
Therefore, $x \in h(p_1 \cup p_2 \cup ... \cup p_i \cup ... \cup p_n)$

From (1) and (2), So the k-best from the whole value set s is same as the reduction from the k-best results from each processes. We implemented $minheap$ and $maxheap$ to store the k-best results for each parallel threads/process.

7 | IMPLEMENTATIONS ON PARALLEL AND DISTRIBUTED ARCHITECTURE

To detect anomalous regions in spatial/spatial-temporal grid, for brute-force approach, all the contiguous and rectangular areas are searched and LRT computation is performed over each of them. The regions within top-k LRT statistic values are treated as potential outliers. However, it is very time-consuming to identity the outliers.

To improve performance, pre-computation method is presented based on inclusive/exclusive scheme. Furthermore, we devise range mapping scheme to transform the total rectangular regions from triangular shaped space to rectangular shaped space. This mapping have a contiguous space without gap to enable the workload perfectly partitioned into same amount for each PU.

We implemented the naive approach, inclusive/exclusive approach on a single CPU and range mapping scheme on different parallel and distributed architectures: Multi-core, GPGPU and EC2 Cloud Cluster in two-dimensional grid (n, n).[6]

7.1 | Naive Approach

The implementation of naive approach of LRT computation on a two-dimensional grid (n, n) is shown in algorithm 3. We can see the computation complexity of brute-force searching is $\mathcal{O}(n^4)$ and the wost case of the computation of single LRT for a region R is $\mathcal{O}(n^2)$.

[6] Please see source code (13).

Algorithm 4 Inclusive/Exclusive top-k LRT search

Input: data grid (G(n,n)), k

Output: top k anomalous regions

```
1: //Inclusive/Exclusive pre-computation
2: prefix_sums←compute_prefix_sums(A,B,X,Y)
3: //Search each rectangle R (x1,y1,x2,y2) in G (n,n)
4: for x1 ←0 to n do
5:     for y1 ←0 to n do
6:         for x2 ←x1 to n do
7:             for y2 ←y1 to n do
8:                 intermediate statistics←prefix_sums(x1,y1,x2,y2)
9:                 score←lrt(x1,y1,x2,y2,intermediate statistics)
10:                //keep kbest regions
11:                kbest ←heap(R,G)
12:            end for
13:        end for
14:    end for
15: end for
```

7.2 | Inclusive/Exclusive Approach

Inclusive/Exclusive scheme in section 3 is applied on the whole spatial grid (G) to generate four pre-computed data set. For any given region R, retrieving the intermediate statistic value is $\mathcal{O}(1)$ time. The overall enumeration cost is reduced in a sequential version. The k-best heap structure is built to keep global top-k regions. The implementation is shown in algorithm 4.

7.3 | Multi-core Approach

Multi-core computers, consisting two or more processors working together on a single integrated circuit, have produced breakthrough performance. They allow faster execution of applications by taking advantage of parallelism, or the ability to work on multiple problems simultaneously. The default mechanism for running parallel programs on multi-core processors is to run a separate process on each core. Threads enable a lighter weight approach than processes. A standardized programming interface called POSIX (Portable Operating System Interface for Unix) was developed to support multiple architecture and provide the capabilities of threads. Implementations adhering to this standard are refereed as Pthreads. Pthreads provide task and data parallelism and flexibility to the programmers.

There are two key advantages associated with LRT computation work on a multi-core architecture with Pthreads. Firstly, for a 1EXP family data grid, LRT computation of each rectangular region is independent. This provides the task parallelism. Secondly, the non-gap ranging mapping scheme clearly enables us to divide the workload to each thread equally, improving scaling.

The implementation of multi-core version is shown in algorithm 5 and algorithm 6. In the main program, each thread is assigned with the same workload and same size of range. It is equal to: total number of regions to be searched divides by the defined number of threads. The range mapping scheme enables each thread to process in a non-gap rectangular range. Otherwise, it is hard to provide each thread same size of range. Each thread performs LRT computation on its own part and keeps k-best regions using min, max heap structure. After all threads finishes computation, the heap function is applied to all the k-best results and get the final k-best regions.

7.4 | GPU/Multi-GPU Approach

The General Purpose Graphic Processing Unit (GPGPU) architecture allows graphic card to be used as general purpose parallel computing device. We briefly describe how the NVIDIA's Compute Unified Device Architecture (CUDA) framework supports massively parallel computing (14). A CUDAprogram consists of one or more phases that are executed on either the host (CPU) or the device (GPU). The host code executed on CPUexhibits little or no data parallelism and the device code (called kernel) executed on GPUside exhibits high amount of data parallelism. The kernel uses large number of threads to exploit data parallelism. The CUDAthreads are extremely light weight and require very few clock cycles to be

Algorithm 5 LRT implementation on $Multicore$ Architecture: Main Part

Input: data grid (G), likelihood function (f)
Output: top k anomalous regions

```
1: //declare threads array
2: pthread_t←thread[num_threads]
3: thread_param ←param[num_threads]
4: // do pre-computation
5: prefix_sums←compute_prefix_sums()
6: // total workload be partitioned
7: size← ((n + 1)(n + 1))^2/4
8: // compute the workload of each thread
9: stride← size/num_threads
10: // initialize thread parameters and spawn threads
11: for i ←0 to num_threads do
12:     thread_param[i].start←start
13:     thread_param[i].end←start+stride
14:     thread_param[i].prefix←prefix_sums
15:     create_pthread(thread_param)
16: end for
17: parallel processing (see the algorithm of parallel part)
18: //get kbest regions
19: kbest ←heap(kbest[0,...,num_threads])
```

Algorithm 6 LRT implementation on Multi-core: Parallel Part

Input: prefix-sum,thread id (i) portion index start, end
Output: kbest regions processed by thread i

```
1: // thread i process regions from (start,end)
2: for portion index ←start to end do
3:     LRT computation for regions obtained from index
4:     // keep kbest for current thread i
5:     kbest ←heap(i)
6: end for
```

generated and scheduled on the GPU. The key programming challenge is to map the computation and expressing them onto an abstract model consisting of grids, blocks and threads. The kernel is then executed by a grid of thread blocks. Blocks execute concurrently among multiprocessors while threads in a given block execute concurrently within a single multiprocessor. Only threads within a block can cooperate with each other.

To exploit CUDA for the LRT computation, all of regions are mapped from triangular range shape onto rectangular range shape. The rectangular range is divided into different non-overlap blocks with same size. Each block has a number of threads to process a number of rectangular regions. For fast computation, each block searches all of the rectangular regions which can be fitted into shared memory. Each thread is assigned to compute a subset of rectangle. To avoid transfering all the results back CPU, parallel reduction is performed. Each thread keeps a k-best heap. Furthermore, keeping the total k-best heaps not arbitrary, it might lead to another kernel for parallel reduction. By defining block number and thread number for different size of data grid, each thread is responsible for varied number of rectangles in different kernel. We did the implementation on single GPGPU and multi-GPGPU and measured the performance. The implementation of single GPGPU is shows in algorithm 11 . For multi-GPGPU

Algorithm 7 GPUkernel

Input: a spatial grid *G(n,n)*, number of blocks *(bx,by)*, number of threads per block *(tx,ty)*

Output: top k anomalous regions processed by each thread

```
1: //total regions to be enumerated
2: total_workload=(n(n+1)/2)^2
3: //compute the workload for each thread
4: tx_size ←(n(n+1)/2 + bx · tx − 1)/(bx · tx)
5: ty_size ←(n(n+1)/2 + by · ty − 1)/(by · ty)
6: //LRT computation of each thread
7: for i ←0 to tx_size do
8:     for j ←0 to ty_size do
9:         //get region coordinates from range mapping scheme
10:        (x1, y1, x2, y2)←reverse_range_mapping(i,j)
11:        //perform LRT computation on region R
12:        score←lrt(R(x1, y1, x2, y2))
13:        //keep kbest for current thread
14:        kbest←heap(current thread)
15:    end for
16: end for
```

implementation, we divide the whole workload equally to each GPGPU. Final k-best regions are obtained after each device return back all the k-best heaps.

7.5 | Amazon EC2 Cloud Cluster

Cloud computing offers a highly scalable infrastructure for high performance computing. Amazon Elastic Compute Cloud (Amazon EC2) enables ?compute? in the cloud. It is possible to get a set of computing instances on demand without requiring a lot of maintenance and financial resources a common cluster would need. We present our parallel implementation for cloud computing using MPI. The workload is equally assigned to each process based on our range mapping scheme and each process performs LRT computation on each rectangle and produces k-best regions with heap structure using reduce function. The implementation is shown in algorithm 8.

8 | EXPERIMENTS AND ANALYSIS

8.1 | Performance Analysis

We have designed and implemented a set of experiments to show and validate the performance speedup on the above different architectures. The experiments are performed on a Poisson distribution model and a randomly generated anomalous region was introduced for verification in synthetic data sets. Answers to the following questions were sought:

- What are the performance gains of naive versus Inclusive/Exclusive approach?
- How is the multi-core scaling of the LRT computation on Multi-core architecture?
- What are the performance gains of GPGPU approaches versus Inclusive/Exclusive approach?
- How is the MPI scaling of LRT computation on PC-cluster?
- How is the computation speed of LRT computation for (n, n) grid on these architectures?

Algorithm 8 MPI main program

Input: a spatial grid *G(n,n)*, number of processes *np*, top-k *kbest*

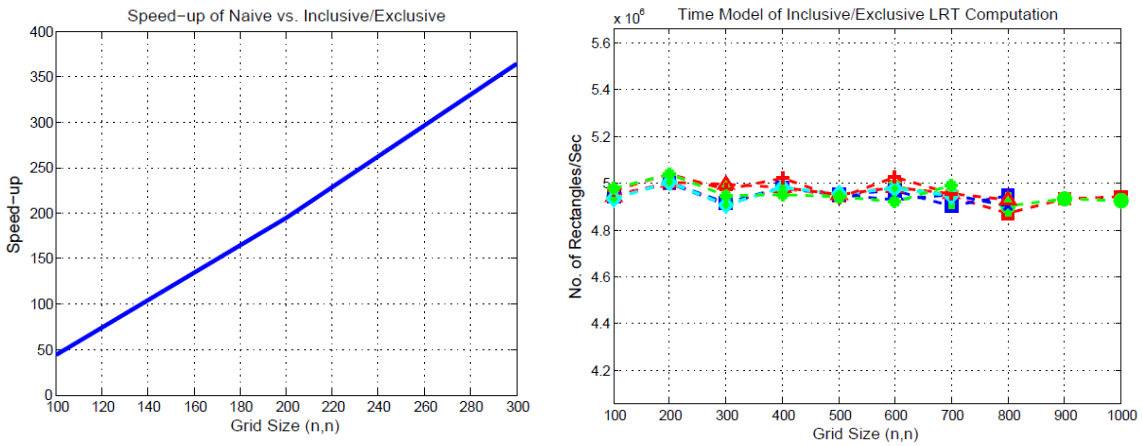
Output: top-k anomalous rectangular regions

```
1: //initialize
2: MPI_init()
3: //broadcast grid information to all nodes
4: MPI_Bcast(n,n,g)
5: total_workload=(n(n + 1)/2)^2
6: //compute workload for each process
7: stride←(total_workload+np-1)/np
8: start←processid · stride
9: end←start + stride
10: //parallel LRT computation
11: Rectangle r←compute(g,start,end)
12: //store kbest results for each process
13: kbest←heap(r)
14: // reduce operation
15: MPI_reduce(kbest)
16: //root process
17: if processid == 0 then
18:     kbest←qsort(reduced_kbest)
19: end if
20: MPI_Finalize()
```

8.1.1 | Naive vs. Inclusive/Exclusive Implementation

We run the naive and inclusive/exclusive implementation on an 8-core E5520 Intel server to compare their performance. Data grid varies from $(100, 100)$ to $(1000, 1000)$. The results are shown in Figure 9 a and Figure 9 b.

From the results in Figure 9 a, we can see that Inclusive/Exclusive approach has a significant speed-up. The number of rectangles per second processed by Inclusive/Exclusive approach is also plotted for various grid sizes in Figure 9 b, it can be seen the result is almost constant, validating that our Inclusive/Exclusive pre-computation approach search complexity is $\mathcal{O}(1)$.



(a) Speed-up of Inclusive/Exclusive approach

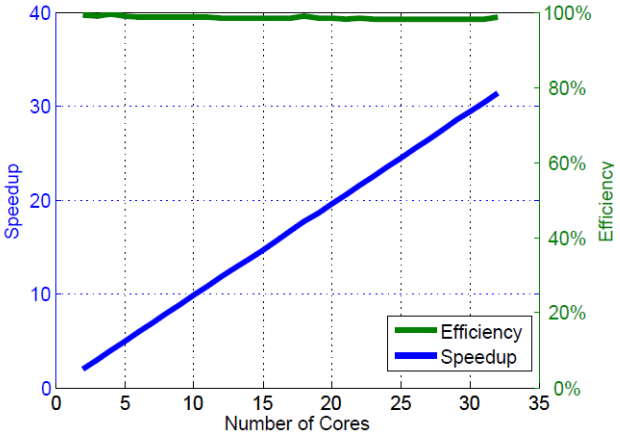
(b) number of rectangles per second processed

FIGURE 9 Naive vs. Inclusive/Exclusive Approach

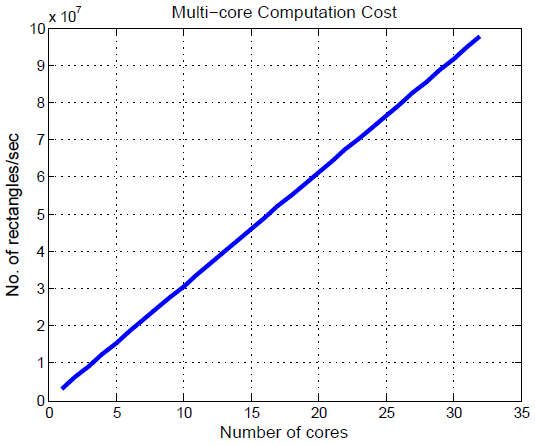
8.1.2 | Multi-core

The multi-core experiment is conducted on a 32-core AMD Opteron(tm) Processor 6128 server with 128GB of RAM and a 64-core AMD Opteron(tm) Processor 6378 server with 384GB of RAM. We use data grid with size of (1000,1000) to show the performance scaling. Figure 10 a and 10 c show that the 1EXP-LRT computation scales very well on multi-core architecture. The speedup increases near-linearly with the increase of number of cores (no_c) and is consistent with the $\mathcal{O}(n^4/no_c)$ running time on each core. A speedup of n on n cores is nearly achieved, indicating near perfect scaling.

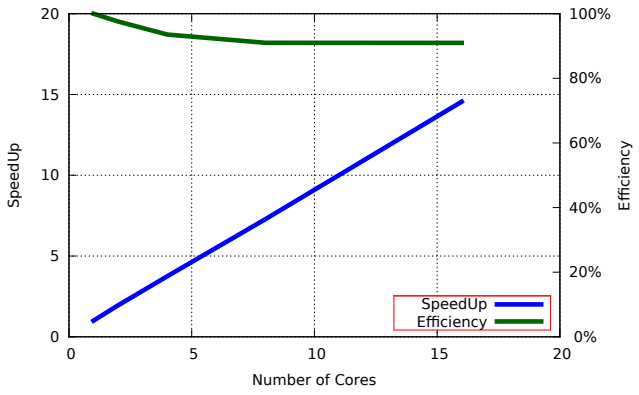
The number of rectangles processed per second is plotted against the number of cores in Figure 10 b and 10 d. The results show the linearity of increasing number of rectangles processed per second with increasing number of cores.



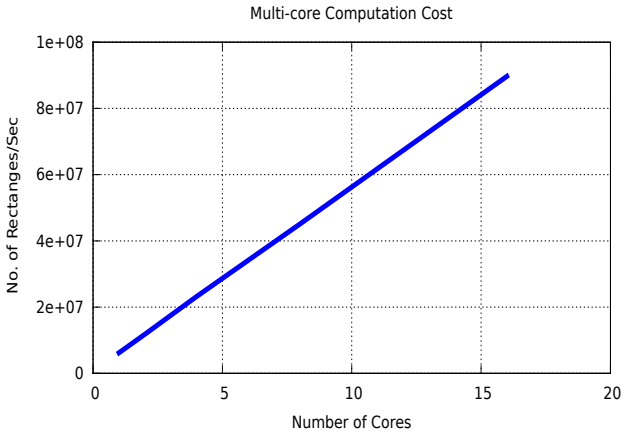
(a) Speed-up and Efficiency of Grid (1000,1000) on Multi-core



(b) The number of rectangles processed per second vs. number of cores on Grid (1000,1000)



(c) Speed-up and Efficiency of Grid (1000,1000) on Multi-core 2



(d) The number of rectangles processed per second vs. number of cores on Grid (1000,1000) on Multi-core2

FIGURE 10 Multi-core Evaluation

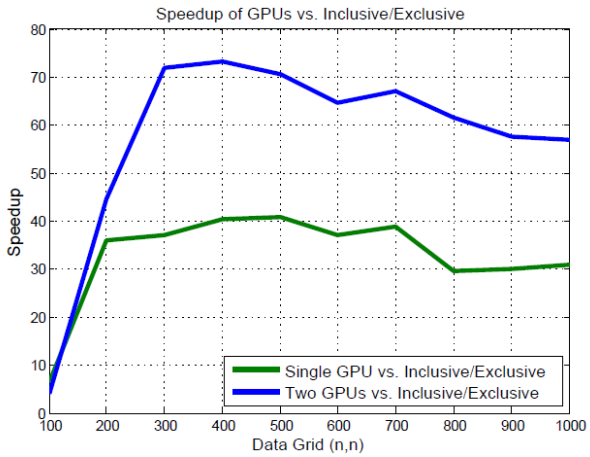
8.1.3 | Multi-GPGPU

The experiments are conducted on two different GPGPU configurations:

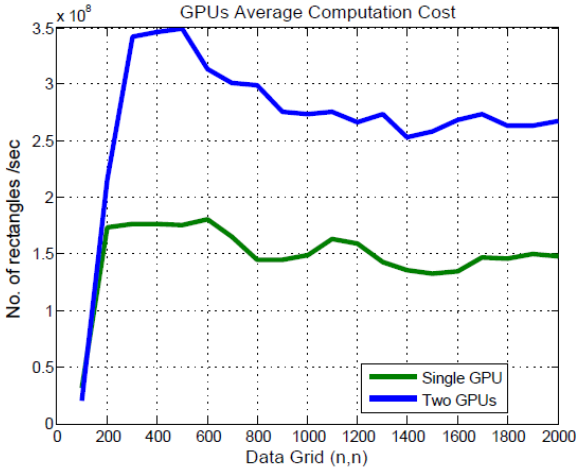
(1)An 8-core E5520 Intel server that is equipped with two GPGPU $Tesla\ C1060$ cards supporting CUDA 4.0. Each GPGPU card has $4GB$ global memory, $16KB$ shared memory, 240 cores and 30 multiprocessors. Experiments are conducted to compare the speed-up performance of GPGPUs with exclusive/inclusive cpu approach (see figure 11 a) on various data grid size (n,n). Furthermore, the performance of number of rectangles

processed per second on one single GPGPU vs. two GPGPUs are compared (see figure 11 b). The results show that LRT computation on two GPGPUs is around 2 times faster than that on one single GPGPU regardless of the data grid size. And the computation cost of per rectangle is almost constant with the increase of data grid for single GPGPU and two GPGPUs with exclusive/inclusive cpu approach.

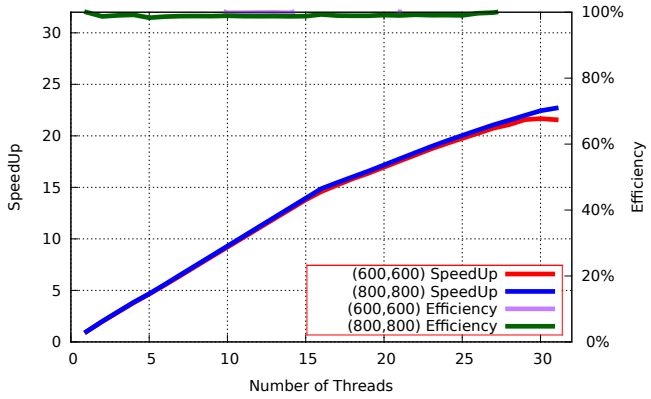
(2) An 8-core server that is equipped with two GPGPU $Tesla\ K20m$ cards supporting CUDA 6.5. Each GPGPU card has $5GB$ global memory,$64KB$ shared memory,2496 cores and 13 multiprocessors. In this architecture, we investigate the speed-up, efficiency and the number of rectangles processed per second with different number of threads on different grid size(see figure 11 c and figure 11 d). Figure 11 c shows the linearly increasing speed-up with increasing number of threads on a given grid size. The efficiency is almost 100%. Figure 11 d shows the linear increasing speed of processing number of rectangles with the increasing of threads on a given grid size.



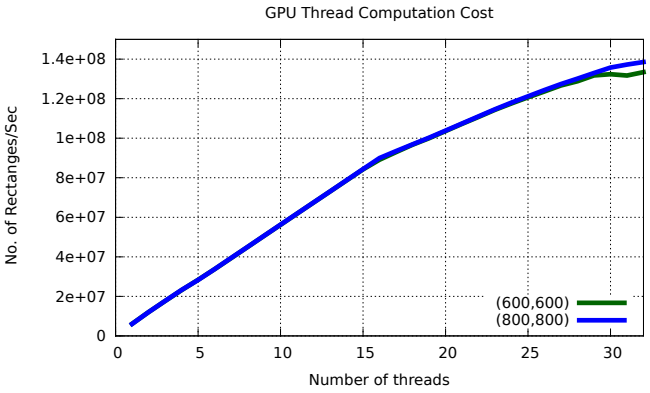
(a) GPGPU speed-up with inclusive/exclusive cpu approach



(b) Number of rectangles processed per second on single GPGPU vs. two GPGPUs.



(c) Speed-up and Efficiency of threads processing on different given grids in GPGPU architecture



(d) The number of rectangles processed per second vs. number of threads on different given grids

FIGURE 11 GPGPUs Cost

Given the $((((n+1)\times(n+1))/2)^2)$ regions to be searched for a (n,n) grid and the number of threads in each block is (tx,ty). The number of blocks (bx,by) changes the LRT computation performance. In our implementation,each region object takes $8byte$. To fully utilize the shared memory, each block has $(tx,ty)=(16,8)$ threads. We vary the value of (bx,by) for the grids with different size to find the optimal block configuration. To maximize performance:

- $Max(blocks)\leq(bx\times by)\leq((n+1)\times(n+1)/2)^2$, otherwise some blocks won't work.
- $((n+1)\times(n+1)/2)^2/(bx\times by)\geq(tx\times ty)$ makes each thread processes at least one region.

TABLE 6 Optimized Block Configuration

$Grid$	$block_x$	$block_y$
(500,500)	128	64
(600,600)	192	86
(700,700)	176	86

- $((n+1)\times(n+1)/2)^2/tx \leq (((n+1)\times(n+1))/2))/ty$ is better for reducing bank conflicts since it retrieves more data by rows.

If there are too few blocks, each thread processes a quite number of regions and thus performance is degraded. Table 6 gives the optimal block configuration. A speedup of up to two can be achieved by choosing the right block configuration using the slowest run for a given grid size as baseline.

8.1.4 | EC2 Cloud Cluster

We study a cluster composed of $20EC2$ high-CPU compute nodes and its effect on MPI application scaling. Instances of this family have proportionally more CPU resources than memory (RAM) and are well suited for compute-intensive applications. Figure 12 a shows the nearly linear speedup with the increase of number of processes for different grid size on cloud cluster. The computation speed of rectangles is plotted out in Figure 12 b, it shows the computation speed is faster with the increase number of processes on a given data grid and also verifies the constant computation speed for different data grid by a given number of processes.

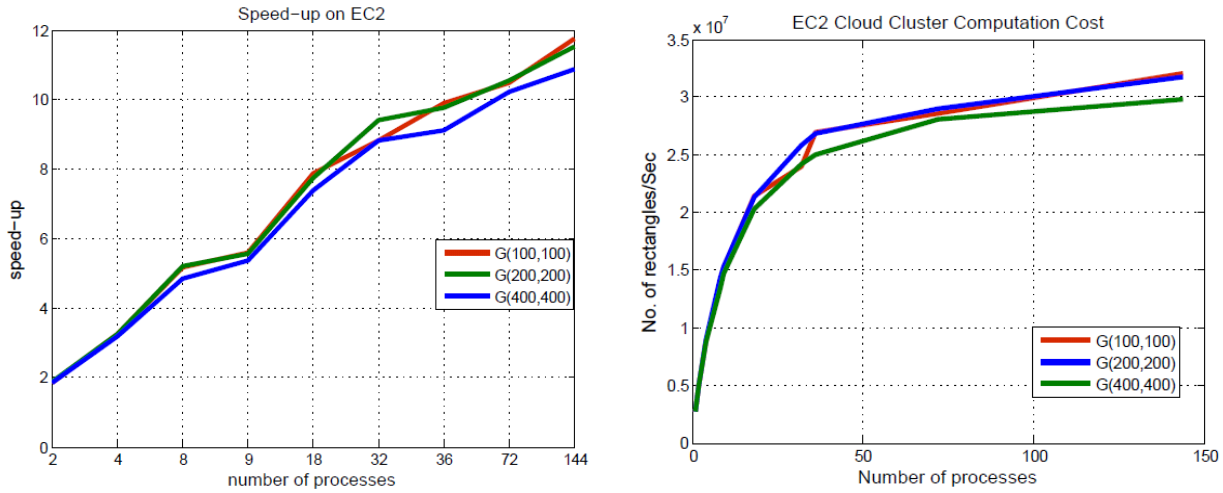


FIGURE 12 Speed-up of LRT computation on EC2 cloud cluster

8.2 | Discussion

From the above results, we further analyze of LRT computation on different architectures. We plot the number of rectangles processed per second for data grid $(1000, 1000)$ on the following different architectures: multi-core, GPGPU and EC2 cloud cluster. The results are shown in Figure **??**. The dashed line in the figure is the processing speed of Inclusive/Exclusive approach on single CPU. From the figure, we can see that the GPGPU approach performs much better than multi-core and EC2, which is almost one order of magnitude faster. Furthermore, we can see that, as the number of cores and the number of processes increase, the LRT processing speed is improved on these architectures.

9 | RELATED WORK

Previous attempts to parallelize LRT computation have only achieved limited success. For example, the Spatial Scan Statistic (SSS), which is a special case of LRT for Poisson data, is available as a program under the name SatScan (15). It has been parallelized for multi-core CPU environments and its extension for a GPGPU hardware (16) has achieved improved speed up of two over the multi-core baseline. The GPGPU implementation in (16) has proposed loading parts of the data into shared memory but has achieved only a modest speed up. The other attempt of (17) applied their own implementation of a spatial scan statistic program on the GPU to the epidemic disease dataset. This solution is only applicable to its special disease scenario. In each of these cases, we believe there is further room for optimising the algorithms for the parallel architectures by devising the fine-grained parallelism strategies. Furthermore, these existing parallel solutions perform LRT tests in a circular or cylindrical way, not in a grid-based scenario. Our parallel solution is different and provides a fully paralleled template for 1EXP-LRT computation in a grid.

Recently there are some work (18, 19, 20, 21) using LRT Framework to solve the big computation cost, however, they focus on improving pruning strategy, choosing subset or using sampling strategy , not utilizing parallel and distributed architecture to to speed up performance.

10 | CONCLUSION

The Likelihood Ratio Test Statistic (LRT) is a state-of-the-art method for identifying hotspots or anomalous regions in large spatial settings. To speed up the LRT computation for 1EXP family, this paper proposed three ideas: (i) a novel range mapping scheme is proposed to fully enumerate all the regions in a contiguous space. (ii) a dynamic pre-computation algorithm is implemented to reduce the cost of aggregating intermediate statistics. (iii) to save space and improve processing speed, kbest reduction scheme is presented to accumulate distributed results. We did the implementations on different parallel architectures: Multi-core, Multi-GPGPU and EC2 cloud cluster and extensive experiments are done correspondingly. From the results, we see that pre-computation approach has a linear speed-up with the data grid size comparing to the brute-force sequential approach. Then we compare the speed-up on different parallel architectures using pre-computation approach as baseline. The speed-up of these parallel approach increases near-linearly with the increase of the number of "parallel computing component" on different architectures. In concert, the parallel approaches yield a speed up of nearly four thousand times compared to their sequential counterpart. Further analysis on the processing speed of number of rectangles is given. This provides some recommended information for choosing the right architectures on various factors. Moving the computation of the LRT statistics to the parallel architectures enables the use of this sophisticated method of outlier detection for larger spatial grids than previously reported.

In future, we will apply the range mapping scheme on non-1EXP family distribution using pruning strategy (1). A unified parallel approach will be provided for generalized LRT computation in spatial grids.

SUPPORTING INFORMATION

11 | INCLUSIVE/EXCLUSIVE STATISTICS AGGREGATION

$$|R(x_1, y_1, x_2, y_2)| \;=\; |A(x_2, y_2)| + |B(x_1, y_1)| + |X(x_1, y_2)| + |Y(x_2, y_1)| - |G| \tag{46}$$

Proof.

$$\begin{aligned}
|R| &= \textstyle\sum_{i=1}^{x_2}\sum_{j=1}^{y_2} c_{i,j} + \sum_{i=x_1}^{n}\sum_{j=y_1}^{n} c_{i,j} + \sum_{i=x_2}^{n}\sum_{j=1}^{y_1} c_{i,j} + \sum_{i=1}^{x_1}\sum_{j=y_2}^{n} c_{i,j} && (47)\\
&\quad - \textstyle\sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j} && (48)\\
&= \textstyle\sum_{i=1}^{x_1}\sum_{j=1}^{y_2} c_{i,j} + \sum_{i=x_1}^{x_2}(\sum_{j=1}^{y_1} c_{i,j} + \sum_{j=y_1}^{y_2} c_{i,j}) + \sum_{i=x_1}^{n}\sum_{j=y_1}^{n} c_{i,j} && (49)\\
&\quad + \textstyle\sum_{i=x_2}^{n}\sum_{j=1}^{y_1} c_{i,j} + \sum_{i=1}^{x_1}\sum_{j=y_2}^{n} c_{i,j} && (50)\\
&= \textstyle\sum_{i=1}^{x_1}\sum_{j=1}^{n} c_{i,j} + \sum_{i=x_1}^{x_2}\sum_{j=1}^{y_1} c_{i,j} + |R| + |R| + \sum_{i=x_1}^{x_2}\sum_{j=y_2}^{n} c_{i,j} && (51)\\
&\quad + \textstyle\sum_{i=x_2}^{n}\sum_{j=1}^{n} c_{i,j} - \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j} && (52)\\
&= \textstyle\sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j} + |R| - \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j} && (53)\\
&= |R| && (54)
\end{aligned}$$

□

Definition 9.

$$A(i,j) = \sum_{k=1}^{i}\sum_{l=1}^{j} c_{i,j} \quad \text{for all } 1 \le i,j \le n. \tag{55}$$

Lemma 16.

$$A(i,j) = A_{i-1,j} + A_{i,j-1} - A_{i-1,j-1} + c_{i,j} \quad \textit{for all } 1 < i,j \le n. \tag{56}$$

Proof.

$$\begin{aligned}
A(i,j) &= \sum_{k=1}^{i-1}\sum_{l=1}^{j} c_{k,l} + a_{i,j-1} - a_{i-1,j-1} + c_{i,j} && (57)\\
&= \sum_{k=1}^{i-1}\left(\sum_{l=1}^{j-1} c_{k,l} + c_{k,j}\right) + a_{i,j-1} - \sum_{k=1}^{i-1}\sum_{l=1}^{j-1} c_{k,l} + c_{i,j} && (58)\\
&= \sum_{k=1}^{i-1} c_{k,j} + c_{i,j} + a_{i,j-1} && (59)\\
&= \sum_{k=1}^{i} c_{k,j} + a_{i,j-1} && (60)\\
&= \sum_{k=1}^{i} c_{k,j} + \sum_{k=1}^{i}\sum_{l=1}^{j-1} c_{k,l} && (61)\\
&= A(i,j) && (62)
\end{aligned}$$

□

Definition 10.

$$X_{i,j} = \sum_{k=i}^{n}\sum_{l=1}^{j} c_{i,j} \quad \text{for all } 1 \le i,j \le n. \tag{63}$$

Lemma 17.

$$X_{i,j} = X_{i,j-1} + X_{i+1,j} - X_{i+1,j-1} + c_{i,j} \quad \textit{for all } 1 < i,j \le n. \tag{64}$$

Proof.

$$\begin{aligned}
X_{i,j} &= \sum_{k=i+1}^{n}\sum_{l=1}^{j} c_{k,l} + X_{i,j-1} - X_{i+1,j-1} + c_{i,j}\\
&= \sum_{k=i+1}^{n}\left(\sum_{l=1}^{j-1} c_{k,l} + c_{k,j}\right) + X_{i,j-1} - \sum_{k=1+1}^{n}\sum_{l=1}^{j-1} c_{k,l} + c_{i,j}\\
&= \sum_{k=i+1}^{n} c_{k,j} + c_{i,j} + X_{i,j-1}\\
&= \sum_{k=i}^{n} c_{k,j} + X_{i,j-1}\\
&= \sum_{k=i}^{n} c_{k,j} + \sum_{k=i}^{n}\sum_{l=1}^{j-1} c_{k,l}\\
&= X_{i,j}
\end{aligned}$$

□

Definition 11.

$$X_{i,j} = \sum_{k=i}^{n}\sum_{l=1}^{j} c_{i,j} \quad \text{for all } 1 \le i,j \le n. \tag{65}$$

Lemma 18.

$$X_{i,j} = X_{i,j-1} + X_{i+1,j} - X_{i+1,j-1} + c_{i,j} \quad \textit{for all } 1 < i,j \le n. \tag{66}$$

Proof.

$$\begin{aligned}
X_{i,j} &= \sum_{k=i+1}^{n}\sum_{l=1}^{j} c_{k,l} + X_{i,j-1} - X_{i+1,j-1} + c_{i,j} \\
&= \sum_{k=i+1}^{n}\left(\sum_{l=1}^{j-1} c_{k,l} + c_{k,j}\right) + X_{i,j-1} - \sum_{k=1+1}^{n}\sum_{l=1}^{j-1} c_{k,l} + c_{i,j} \\
&= \sum_{k=i+1}^{n} c_{k,j} + c_{i,j} + X_{i,j-1} \\
&= \sum_{k=i}^{n} c_{k,j} + X_{i,j-1} \\
&= \sum_{k=i}^{n} c_{k,j} + \sum_{k=i}^{n}\sum_{l=1}^{j-1} c_{k,l} \\
&= X_{i,j}
\end{aligned}$$

□

Definition 12.

$$y_{i,j} = \sum_{k=1}^{i}\sum_{l=j}^{n} c_{i,j} \quad \text{for all } 1 \le i, j \le n. \tag{67}$$

Lemma 19.

$$Y_{i,j} = Y_{i-1,j} + Y_{i,j+1} - Y_{i-1,j+1} + c_{i,j} \quad \textit{for all } 1 < i, j \le n. \tag{68}$$

Proof.

$$\begin{aligned}
Y_{i,j} &= \sum_{k=1}^{i-1}\sum_{l=j}^{n} c_{k,l} + Y_{i,j+1} - Y_{i-1,j+1} + c_{i,j} \\
&= \sum_{k=1}^{i-1}\left(\sum_{l=j+1}^{n} c_{k,l} + c_{k,j}\right) + Y_{i,j+1} - \sum_{k=1}^{i-1}\sum_{l=j+1}^{n} c_{k,l} + c_{i,j} \\
&= \sum_{k=1}^{i-1} c_{k,j} + c_{i,j} + Y_{i,j+1} \\
&= \sum_{k=1}^{i} c_{k,j} + Y_{i,j+1} \\
&= \sum_{k=1}^{i} c_{k,j} + \sum_{k=1}^{i}\sum_{l=j+1}^{n} c_{k,l} \\
&= Y_{i,j}
\end{aligned}$$

□

12 | PRE-PROCESSING OF INCLUSIVE/EXCLUSIVE COMPUTATION

References

[1] M. Wu, X. Song, C. Jermaine, S. Ranka, J. Gums . A LRT Framework for Fast Spatial Anomaly Detection. In: :887–896.

[2] S. Wilks S.. The large sample distribution of the likelihood ratio for testing composite hypotheses. *Annals of Mathematical Statistics.* 1938;(9):60–62.

[3] Pang X. L., Chawla S., Liu W., Zheng. Y.. On Mining Anomalous Patterns in Road Traffic Streams. In: :237–251; 2011.

[4] X.L. Pang B. Scholz, G.Wilcox: . A Scalable Approach for LRT Computation in GPGPU Environments. *APWeb.* 2013;:595–608.

[5] Vuduc R., Chandramowlishwaranv A., Choi J., Guney M., Shringarpure A.. On the Limits of GPU Acceleration. In: :237–251; 2010.

[6] Gregerson A.. Implementing fast MRI gridding on GPUs via. CUDA. In: ; 2008.

Algorithm 9 Inclusive/Exclusive Pre-computation for Set B

Input: data grid (G)

Output: accumulated counts $B(i,j)$

```
1: //Initialize first element B(n-1,n-1)
2: B(n-1,n-1)←G(n-1,n-1)
3: //accumulation of remaining elements in last column
4: for j←(n-1) to 1 do
5:     B(j-1,n-1) ← G(i-1,n-1) +B(j,n-1)
6: end for
7: //accumulation of remaining elements in last row
8: for i←(n-1) to 1 do
9:     B(n-1,i-1)←G(n-1,i-1) + B(n-1,i)
10: end for
11: //accumulation of all the elements in remaining rows and columns
12: for k←1 to n do
13:     for i←(n-1) to k do
14:         B(i-k,n-1-k)← G(i-k,n-1-k)+B(i-k+1,n-1-k)+B(i-k,n-k)-B(i-k+1,n-k)
15:     end for
16:     for j←(n-1) to k do
17:         B(n-1-k,j-k)←G(n-1-k,j-k)+B(n-1-k,j-k+1)+ B(n-k,j-k)-B(n-k,j-k+1)
18:     end for
19: end for
```

Algorithm 10 Pre-processing of Inclusive/Exclusive Computation for Set X

```
1: //Initialize first column
2: for i←1 to n do
3:     X(i,0)←0
4: end for
5: //Initialize last row
6: for j←1 to n do
7:     X(n-1,j)←0
8: end for
9: //Iterate over all diagonal elements
10: for k←1 to n do
11:     //associate columns of diagonal elements
12:     for i←n-1 to k do
13:         X(i-k,k)←G(i-k+1,k-1)+X(i-k+1,k)+X(i-k,k-1)-X(i-k+1,k-1)
14:     end for
15:     // associate columns of diagonal element k
16:     for j←k to n do
17:         X(n-1-k,j)←G(n-k,j-1)+X(n-1-k,j-1)+X(n-k,j)-X(n-k,j-1)
18:     end for
19: end for
```

[7] Hong S., Kim S. K., Oguntebiv T., Olukotun K.. Efficient Parallel Graph Exploration on Multi-Core CPU and GPU. In: ; 2011.

Algorithm 11 Inclusive/Exclusive Pre-computation for Set Y

Input: data grid (G)

Output: accumulated counts $Y(i,j)$

```
1: //Initialize last column
2: for i←1 to n do
3:     Y(i,n-1)←0
4: end for
5: //Initialize first row
6: for j←1 to n do
7:     Y(0,j)←0
8: end for
9: //Iterate over all diagonal elements
10: for k←1 to n do
11:     //associate columns of diagonal elements
12:     for i←k to (n-1) do
13:         Y(i,n-1-k)←G(i-1,n-k)+Y(i-1,n-1-k)+ Y(i,n-k)-Y(i-1,n-k)
14:     end for
15:     // associate columns of diagonal element k
16:     for j←(n-1) to k do
17:         Y(k,n-k)←G(k-1,n-k+1)+Y(k,n-k+1)+Y(k-1,n-k)-Y(k-1,n-k+1)
18:     end for
19: end for
```


[8] Agarwal D., Phillips J. M., Venkatasubramanian S.. The hunting of the bump: On maximizing statistical discrepancy. In: :1137–1146; 2006.

[9] X.L. Pang B. Scholz, G.Wilcox: . A Scalable Approach for LRT Computation in GPGPU Environments. *APWeb.* 2013;:595–608.

[10] V.Efthymiou E. Ntoutsi. Top-k Computations in MapReduce: A Case Study on Recommendations. *IEEE BigData.* 2015;.

[11] M. Saouk A. Vlachou. Efficient Processing of Top-k Joins in MapReduce. *IEEE BigData.* 2016;.

[12] Horner Scheme .

[13] https://github.com/qlinsey/parallel-lrt .

[14] http://developer.nvidia.com/nvidia-gpu-computing-documentation .

[15] SatScan: http://www.SatScan.org .

[16] Larew S. G., Maciejewski R., Woo I., Ebert. D. S.. Spatial Scan Statistics on the GPGPU. In: ; 2010.

[17] Zhao S. S., Zhou CH.. Accelerating spatial clustering detection of epidemic disease with graphics processing unit. In: :1–6; 2010.

[18] W.Sheng Y.X.Yu R.G. An X.Zhou, X.Zhang . A Novel Emergency Detection Approach Leveraging Spatiotemporal Behavior for Power System. *T. Edutainmen.* 2016;:187–199.

[19] S.Speakman E.McFowland D.B. Neil. Penalized fast subset scanning. *Journal of Computational and Graphical Statistics.* 2016;:382Ű-404.

[20] X. Zhou A.Liu Z.Shafiq F. Zhang. A Traffic Flow Approach to Early Detection of Gathering Events. *24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems.* 2016;.

[21] M.Matheny K.Q. Wang L.Zhang J.M. Phillips. Scalable Spatial Scan Statistics through Sampling. *ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL).* 2016;.

How cite this article: Williams K., B. Hoskins, R. Lee, G. Masato, and T. Woollings (2016), A regime analysis of Atlantic winter jet variability applied to evaluate HadGEM3-GC2, *Q.J.R. Meteorol. Soc.*, *2017;00:1–6*.