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

Given an (n, n) spatial data grid draw from an arbitrary distribution, the Likelihood Ratio Test (LRT) is a method for identifying hotspots or anomalous rectangular regions. The naive approach has $\mathcal{O}(n^4)$ time complexity. In this paper, we study how parallel processing can accelerate LRT computation for one parameter exponential (1EXP) distributions.
A novel range mapping scheme is proposed to balance workloads, and this is confirmed with a dynamic pre-computation algorithm for fast computing when LRT computation is from 1EXP family. Various implementations are devised to adapt to different parallel and distributed architectures: Multi-core, Multi-GPGPU and EC2 cloud cluster. Extensive experiments are provided to demonstrate the utility of this approach and extensive performance analysis is given.

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 localization 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 (1), 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 total execution time for identifying the most anomalous region is $\mathcal{O}(n^4)$. The worst case of computing single LRT for a region R is $\mathcal{O}(n^2)$. is As noted by Wu et al.(2), a naive implementation of LRT for a moderately sized 64×64 spatial grid may take nearly six hundred days.[1] Wu et al.(2) 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(2) 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 an 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.

[1]The experiment results were reported in 2009.

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 computing component" (PCC) [2] is balanced. We propose a novel range mapping scheme to transform irregular 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 PCC 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 kbest reduction strategy is presented for accumulating distributed results on each "PCC" 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 present a novel range mapping scheme for 1EXP LRT computation for multi-dimensional data grid. In Section 5, we explain how we use Inclusive/Exclusive rule and dynamic programming to speed up the enumeration and processing of regions measurements. To produce the final top-k regions, each parallel portion generates top k results and kbest 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

2.1 | The Likelihood Ratio Test (LRT)

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

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

where $L()$ is the likelihood function and θ is a set of parameters for the distribution (2). 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 (2).

2.2 | One-parameter Exponential Family (1EXP)

We briefly introduce the one-parameter exponential family (1EXP) and the simplified LRT statistic on 1EXP.

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 probability density given by

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

[2]We define the (computational) granularity of a parallel architecture as the largest "computational component" that should run sequentially on an "application computing component". 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

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), and $B_e(\cdot)$is a strictly convex function. 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:*

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

where 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. m_R and b_R are important measurements to calculate the statistic of 1EXP. See the detail of the function of g_e, G and B_e 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 mean $\{exp(\theta_R)b_R\}$ and $\{exp(\theta_{\bar{R}})b_{\bar{R}}\}$ respectively. Then $b_R = \frac{b_R}{b_R+b_{\bar{R}}}$ and $m_R = \frac{m_R}{m_R+m_{\bar{R}}}$. 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 use a small example to demonstrate two different parallel strategies for performing LRT statistics in one-dimension data grid. The strategies are: (i) Overlapping Enumeration; (ii) Non-overlapping Enumeration. For simplicity, we will describe the strategies on GPGPU computing environment and they can be easily extended to multi-core and pc-clusters computing environments. In this example, we assume that each block can only maximally process 2 intervals in GPGPU environment.

Example: A data set has five data points, denoted as $\{O_0, O_1, O_2, O_3, O_4\}$. Our task is how to enumerate all of the intervals among these five points in parallel. See Figure 1 .

We express $l_{(i,j)}$ as an interval between point O_i and O_j, where $(0 \leq i < n)$, $(0 < j \leq n)$ and $(j > i)$. The total number of intervals among five points is computed as: $\frac{4\cdot(4+1)}{2} = 2 \times 5 = 10$.

In the case of Overlapping Enumeration parallel strategy, every two continuous overlapped intervals (i.e. intervals (i,j), $(i, j+1)$) are assigned to a block. From Figure 1 (a), $Block_0$ contains intervals: $\{l_{(0,0)}, l_{(0,1)}\}$; $Block_1$ contains intervals: $\{l_{(1,1)}, l_{(1,2)}\}$, we can see that the overlapping scheme enumerates all of the intervals up to size of two. However, some blocks may have fewer number of intervals in this way. For example, $Block_4$ has only one interval(i.e. $\{l_{(3,3)}\}$). This leads to an observation of workload unbalancing among blocks. Furthermore, in each block, different interval is assigned to different thread. The lengths of intervals are different and this can further creates different workload among different threads during brute-force calculation.

If we plot each interval $l_{i,j}$ as a point (i,j) onto two-dimensional space, a triangular shape is formed. In Figure 1 b, all of the ten intervals 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 can be treated as the product of $(n+1)$ and $\frac{(n)}{2}$. A rectangular shape can be formed by $[width, height] = [n+1, \frac{n}{2}]$. In our example, $\{(0,0), (0,1), (1,4), (0,4)\}$ forms a rectangle and it has the exact same number of points as the formed triangular shape. Therefore, each point in this rectangle can be mapped to one point in the triangle. In a rectangular shape, we can divide the entire workload equally and assign each to each block. This solves the workload imbalance among blocks. In each block, each thread process the same number of intervals. However, this still does not solve the workload imbalance among each thread because of the LRT computation on different sized sub-grid within each block. In next section, we presents a dynamic programming scheme to pre-compute subsets of LRT statistic value and the computation of each thread becomes $\mathcal{O}(1)$, which will solve the workload imbalance issue among threads.

The Overlapping Enumeration strategy has been implemented in (4). In this paper, the Non-overlapping Enumeration strategy, called Range Mapping scheme, is presented to overcome the limitations of the Overlapping Enumeration strategy.

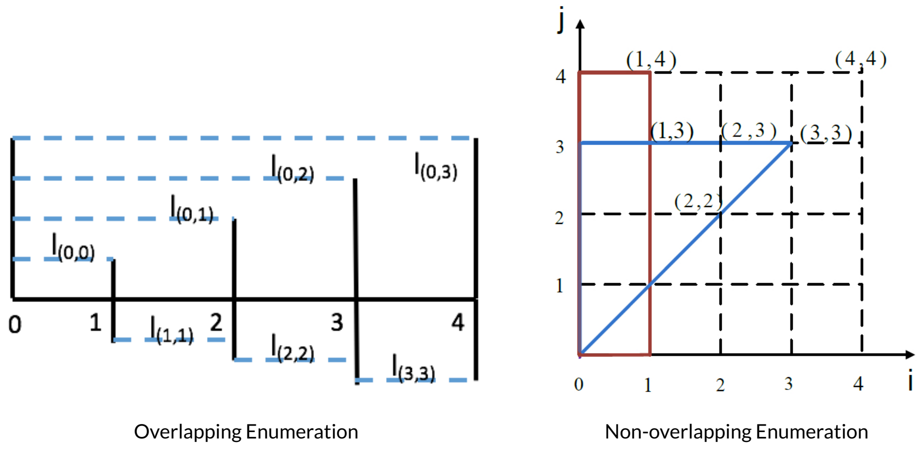
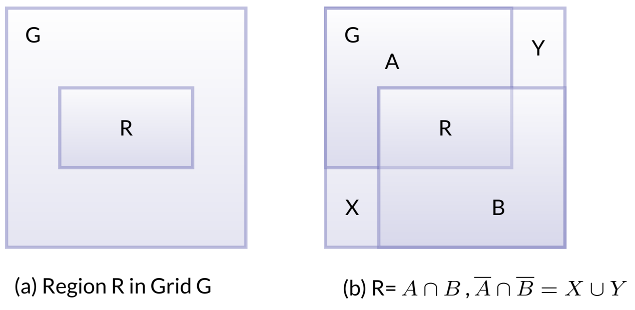


FIGURE 1 Parallel Enumeration Schemes

3.2 | Dynamic Pre-computation

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.



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

FIGURE 2 Set Relations for Region Problem

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

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

$$|R(x_1, y_1, x_2, y_2)| = |A(x_2, y_2)| + |B(x_1, y_1)| + |X(x_1, y_2)| \\ + |Y(x_2, y_1)| - |G| \tag{4}$$

From Equation 4, 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 will give more details and related algorithms in this paper.

4 | RANGE MAPPING SCHEME

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

From section 3.1, we know that if all of the pairwise intervals between n data points, denoted as $l_{(i,j)}$, are plotted onto two-dimensional coordinate system, they form triangular shape. Each pair of (i, j) in triangular-shaped space can be transformed to pair (i', j') in rectangular-shaped space. After transformation, it enables the whole rectangular space to be partitioned into equal portions. Each portion consists of same amount of pairs and is distributed onto different "parallel computing component" (PCC) to balance workload and facilitate parallelization. We name this transforming scheme as range mapping scheme. This scheme can be directly extended to two dimensional spatial grid and multi-dimensional grid. Figure 3 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 3 b shows the rectangular-shaped space after transformation.

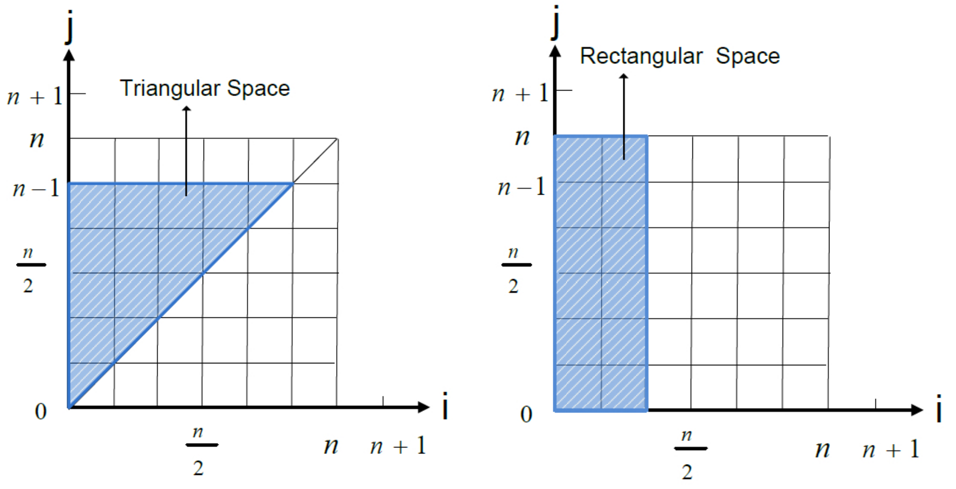


FIGURE 3 1-d Interval Transformation from $[n, n]$ to $[(n + 1)/2, n]$

If an interval index is given, Horner scheme (9) can be applied to generate unique index in desired dimension. For example, given an interval x, where x is integer number $(0 \le x \le n)$, to transform x to 2-dimentional value (i,j), we can use horner scheme: $i = x/n$ and $j = x \mod n$. Then transformation can be done to map current index (i,j) to the index in different shaped space. We give details in the following:

4.1 | One-Dimensional Mapping

First, we consider the one-dimensional range mapping scheme. We assume there are n data points.

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

Given the following information:

$$N = \{0..n - 1\} \tag{5}$$
$$R = \{0..\lfloor \tfrac{n+1}{2} \rfloor - 1\} \times \{0..n\} \tag{6}$$
$$T = \{(i, j) \in N^2 | i <= j\} \tag{7}$$
$$M = \{0..\lfloor \tfrac{n+1}{2} \rfloor \times (n + 1) - 1\} \tag{8}$$
$$\tag{9}$$

where N is the number of data points, R forms rectangular space , T forms triangular space and M is number of total intervals.

The grid size n is even: 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 $x \in M$ to a pair in $(i, j) \in R$ in rectangular space. Secondly, ϕ_2 transforms the pair $(i, j) \in R$ to a pair $(i', j') \in T$ in triangular space.

Definition 1. Mapping Function

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

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

In this function, x is an interval index and (i, j) is mapped to x by applying horner scheme.

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

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

Definition 2. Mapping Function

$$\phi_2 : \ R \to T \tag{10}$$

$$(i, j) \mapsto \begin{cases} (i, i + j) & \text{if } (i + j) \le n \\ (n - i - 1, n - j + n - i - 1) & \text{otherwise} \end{cases} \tag{11}$$

The transformation is achieved by applying two lines: $i = j$ and $i + j = n$ to divide the range space and thus map points among different shape. Figure 4 a shows the mapping.

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

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

Corollary 1. *The function $\phi : MR \to T, x \mapsto (i, j)$ is bijective.*

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 a coordinate index $x \in M$ to a pair in $(i, j) \in R$ in rectangular space. Secondly, ϕ_2 transforms the rectangular pair $(i, j) \in R$ to a pair $(c, d) \in T$ in triangular space.

Definition 3. Mapping Function

$$\phi_1 : M \to R \tag{12}$$

$$x \mapsto (i, j) \tag{13}$$

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

Lemma 5. *The range of ϕ_1 is R. That means: i is in the range of $\{0, ..., \frac{n+1}{2} - 1\}$ and j is in the range of $\{0, ..., n\}$.*

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

Definition 4. Mapping Function

$$\phi_2 : \ R \to T \tag{14}$$

$$(i,j) \ \mapsto \begin{cases} (i, i+j) & \text{if } (i+j) \le n \\ (n-i-1, n-j+n-i-1) & \text{if} (i+1) \le \frac{(n+1)}{2} \end{cases} \tag{15}$$

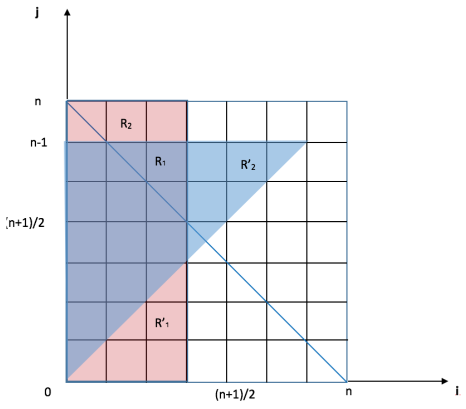
The transformation is achieved by applying two lines: $i = j$ and $i + j = n$ to divide the range space and thus map points among different shape. Since n is odd, the rectangular shape is enlarged and a straight line $i = \frac{(n+1)}{2} - 1$ is used to limit the number of points to triangle shape. Figure 4 b shows the mapping.

Lemma 7. *The range of ϕ_2 is T. That means: i is in the range of $\{0, ..., \frac{n}{2} - 1\}$ and j is in the range of $\{0, ..., n\}$.*

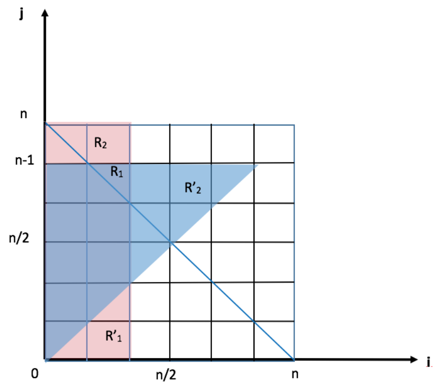
Lemma 8. *The function $\phi_2 : R \to T, x \mapsto (i,j)$ is surjective.*

Corollary 2. *The function $\phi : MR \to T, x \mapsto (i,j)$ is surjective.*

Figure 4 a and Figure 4 b show the solutions and implementation is shown in Algorithm 1.



(a) Solution when Grid size n is even number



(b) Solution when Grid size n is odd number

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

Example: To illustrate the transformation, Figure 5 a and 5 b show small examples on how to transform the intervals when having data points $\{0,1,2,3,4\}$ and $\{0,1,2,3,4,5\}$ respectively. For data point $(i,j) \in \{0,1,2,3,4\}$, the total intervals is 10. By plotting out all of them in Figure 5 a, the triangular shaped space is bounded by $\{(0,0),(3,3),(0,3)\}$ and there are exactly 10 points in total. The rectangular shaped space is bounded by $\{(0,0),(1,0),(1,4),(0,4)\}$ and total number of points is exactly 10. Similarly, for data point $(i,j) \in \{0,1,2,3,4,5\}$, the triangular shaped space is bounded by $\{(0,0),(4,4),(0,4)\}$ and total number of points is . The rectangular shaped space is bounded by $\{(0,0),(2,0),(2,5),(0,5)\}$ and total number of points is 15. Table 1 shows the transformations.

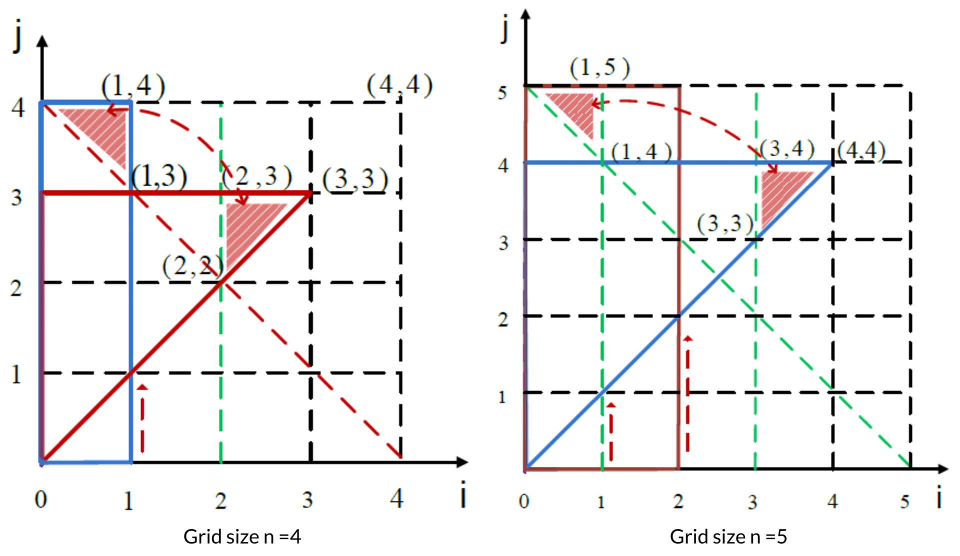


FIGURE 5 Example: 1-d Interval Transformation

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

4.2 | Two-Dimensional Mapping

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

$$N_{1,2} = \{0..n_{1,2} - 1\} \tag{16}$$
$$R_{1,2} = \{0..\lfloor \frac{n_{1,2}+1}{2} \rfloor - 1\} \times \{0..n_{1,2}\} \tag{17}$$
$$T_{1,2} = \{(a_{1,2}, b_{1,2}) \in N_{1,2}^2 | a_i <= b_{1,2}\} \tag{18}$$
$$M_{1,2} = \{0..\lfloor \frac{n_{1,2}+1}{2} \rfloor \times (n_{1,2} + 1) - 1\} \tag{19}$$
$$\tag{20}$$

where N_1, N_2 are coordinate range, R_1, R_2 are rectangular space , T_1, T_2 are triangular space and M_1, M_2 are number range in two-dimension.

Mapping function $\phi^2 : M^2 \rightarrow R^2 \rightarrow T^2$ does transformation. Horner scheme is applied to get two-dimensional tuple (x_1, x_2). Then ϕ^2 is used to transform (x_1, x_2) to (i_1, j_1, i_2, j_2).

Algorithm 1 Parallel Range Mapping in 1-d array

Input: n data points

Output: mapping all the intervals from rectangular to triangular space

———————————————————————

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

4.3 | K-Dimensional Mapping

Similarly, in K-dimension, by given:

$$N_k = \{0..n_k - 1\} \tag{21}$$

$$R_k = \{0..\lfloor \tfrac{n_k+1}{2} \rfloor - 1\} \times \{0..n_k\} \tag{22}$$

$$T_k = \{(a_k, b_k) \in N_k^2 | a_i <= b_k\} \tag{23}$$

$$M_k = \{0..\lfloor \tfrac{n_k+1}{2} \rfloor \times (n_k + 1) - 1\} \tag{24}$$

$$\tag{25}$$

where k is the number of dimension transformed, N_k is coordinate range , R_k is rectangular space , T_k is triangular space and M_k is number range in K-dimension.

Mapping function $\phi^k : M^k \to R^k \to T^k$ performs transformation. Horner's scheme is used to transform a number in M^k to a k-dimensional tuple $(x_1, ..., x_k)$. Then mappings ϕ^k performs the translation from each element x_i to $(i_{1,...,k}, j_{1,...,k})$.

5 | INCLUSIVE/EXCLUSIVE PRE-COMPUTATION SCHEME

Motivated by section 3.2, we now present a novel and unique approach, based on the Inclusive/Exclusive principle and dynamic programming to accumulate 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:

$$|A(i,j)| = |A(i,j-1)| + |A(i-1,j)| - |A(i-1,j-1)| + |G(i,j)| \tag{26}$$

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 2 (see the proofs and the rest implementation of set B, X, Y in Appendix.)

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

$$|B(i,j)| \; = |B(i+1,j)| + |B(i,j+1)| - |B(i+1,j+1)| + |G(i,j)| \tag{27}$$

$$|X(i,j)| \; = \; |X(i,j+1)| + |X(i-1,j)| - |X(i-1,j+1)| + |G(i,j)| \tag{28}$$

$$|Y(i,j)| \; = \; |Y(i+1,j)| + |Y(i,j-1)| - |Y(i+1,j-1)| + |G(i,j)| \tag{29}$$

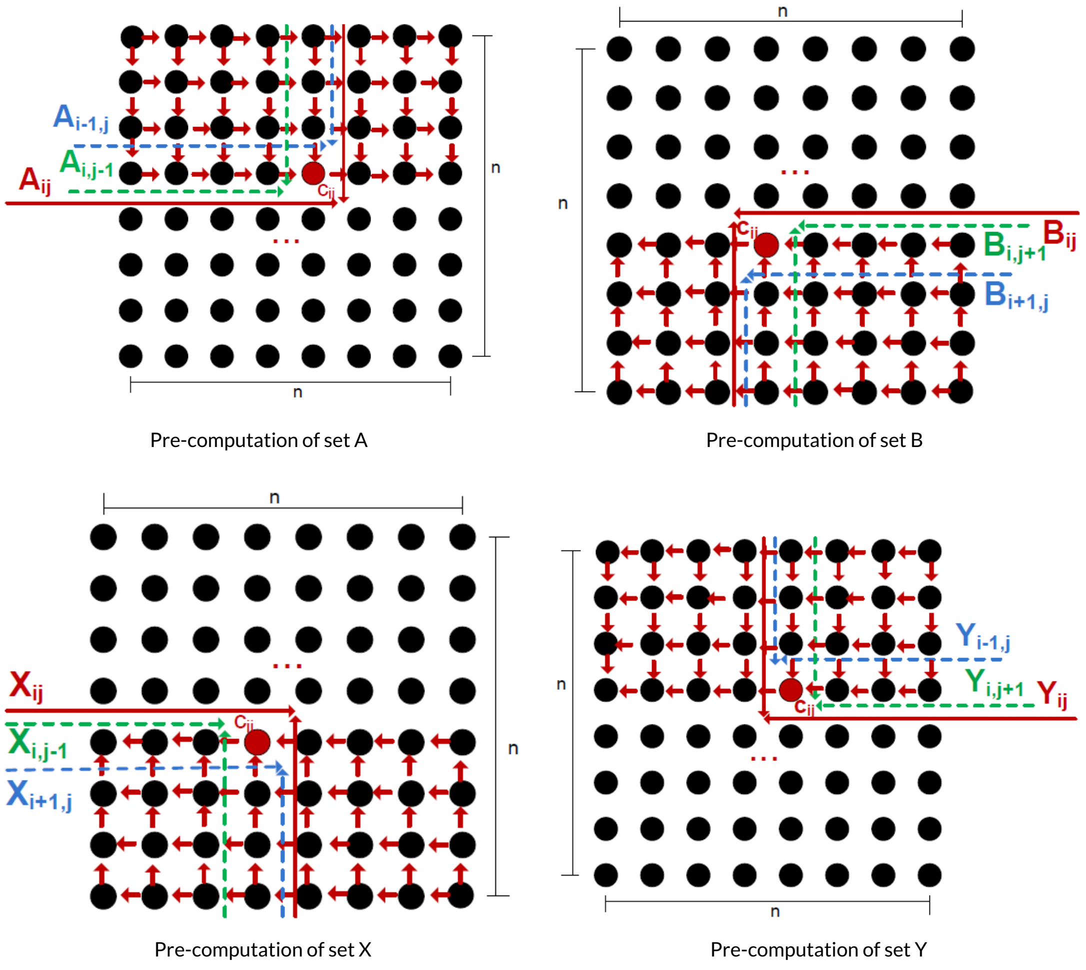


FIGURE 6 pre-computation of set A, B, X and Y

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

Algorithm 2 Inclusive/Exclusive Pre-computation for Set A

Input: data grid (G)

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

————————————————————

```
1:  //Initialize first element $A(0,0)$
2:  $A(0,0) \leftarrow G(0,0)$
3:  //accumulation of remaining elements in first row
4:  for j←1 to n do
5:      $A(0,j) \leftarrow G(0,j)$ +$A(0,j-1)$
6:  end for
7:  //accumulation of remaining elements in first column
8:  for i←1 to n do
9:      $A(i,0) \leftarrow G(i,0)$ + $A(i-1,0)$
10: end for
11: //accumulation of all the elements in remaining rows and columns
12: for k←1 to n do
13:     for i←k to n do
14:         $A(i,k) \leftarrow G(i,n+k) + A(i-1,k) + A(i,k-1) - A(i-1,k-1)$
15:     end for
16:     for j←k to n do
17:         $A(k,j) \leftarrow G(k,n+j) + A(k,j-1) + A(k-1,j) - A(k-1,j-1)$
18:     end for
19: end for
```

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.

6 | KBEST REDUCTION SCHEME

To find top-k[5] anomalous rectangular regions, a heap with maximium size of k is built from each "parallel computing component" (PCC). A further reduction strategy is applied on these k heaps to get final kbest 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 $1 \le t \le n$. Each portion is processed in parallel and a $kbest$ result with heap structure is generated correspondingly. The $kbest$ result from each portion is $p_{1_k}, p_{2_k}, ..., p_{i_k}, ..p_{t_k}$.

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

Lemma 9. $KBestReduction$: *The $kbest$ LRT values from the value set s is equal to the $kbest$ values obtained by performing $kbest$ 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

$$x \in (h(p_1) \cup h(p_2) \cup ... \cup h(p_n)), \tag{30}$$

$$x \in h(h(p_1) \cup h(p_2) \cup ... \cup h(p_n)) \tag{31}$$

Because the total number of $kbest$ values: $|(h(p_1) \cup h(p_2) \cup ... \cup h(p_n)| \ge k$,

and $x \notin h(p_1) \cup h(p_2) \cup ... \cup h(p_n) \rightarrow x \notin h(p_1 \cup p_2 \cup ... \cup p_i \cup ... \cup p_n)$.

[5]Top-k and kbest 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
```

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

$$x \in h(p_1) \cup h(p_2) \cup ... \cup h(p_n), \tag{32}$$

$$x \in h(p_1 \cup p_2 \cup ... \cup p_i \cup ... \cup p_n) \tag{33}$$

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 $kbest$ from the whole value set s is same as the reduction from the $kbest$ results from each processes. We implemented $minheap$ and $maxheap$ to store the $kbest$ 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 PCC.

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 (10).

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 kbest 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 $kbest$ regions using min, max heap structure. After all threads finishes computation, the heap function is applied to all the $kbest$ results and get the final $kbest$ 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 (11). 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← $((n+1)(n+1))^2/4$

 9: // compute the workload of each thread
10: stride← size/num_threads

12: // initialize thread parameters and spawn threads
13: for i ←0 to num_threads do
14:     thread_param[i].start←start
15:     thread_param[i].end←start+stride
16:     thread_param[i].prefix←prefix_sums
17:     create_pthread(thread_param)
18: end for
19: parallel processing (see the algorithm of parallel part)
20: //get kbest regions
21: 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 $kbest$ heap. Furthermore, keeping the total $kbest$ 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 9 . 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 ←$(n(n+1)/2 + bx \cdot tx - 1)/(bx \cdot tx)$
5: ty_size ←$(n(n+1)/2 + by \cdot ty - 1)/(by \cdot ty)$
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:        $(x_1, y_1, x_2, y_2)$←reverse_range_mapping(i,j)
11:        //perform LRT computation on region R
12:        score←lrt($R(x_1, y_1, x_2, y_2)$)
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 $kbest$ regions are obtained after each device return back all the $kbest$ 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 $kbest$ 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

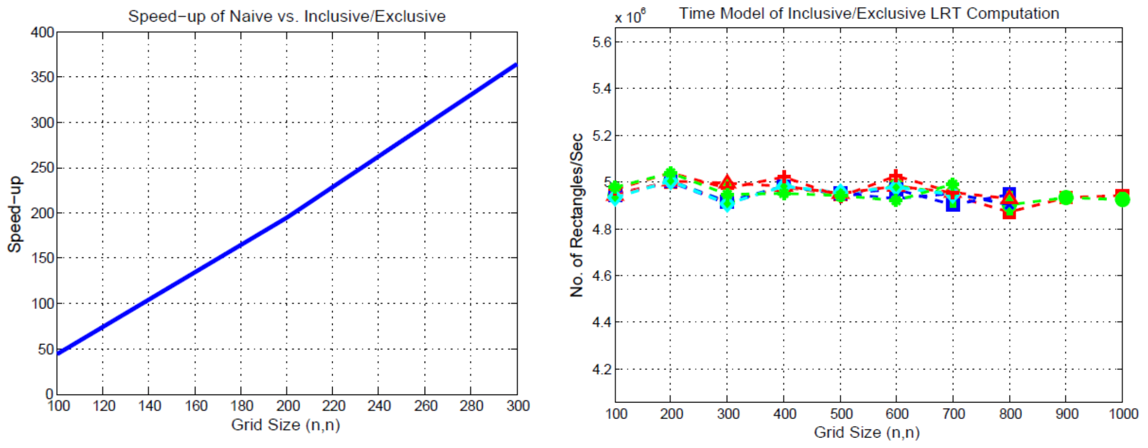
——————————————————————

```
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←$processid \cdot stride$
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 7 a and Figure 7 b.

From the results in Figure 7 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 7 b, it can be seen the result is almost constant, validating that our Inclusive/Exclusive pre-computation approach search complexity is $\mathcal{O}(1)$.



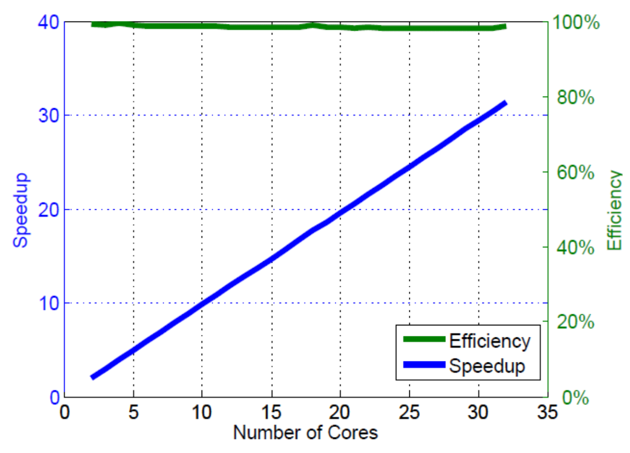
(a) Speed-up of Inclusive/Exclusive approach (b) number of rectangles per second processed

FIGURE 7 Naive vs. Inclusive/Exclusive Approach

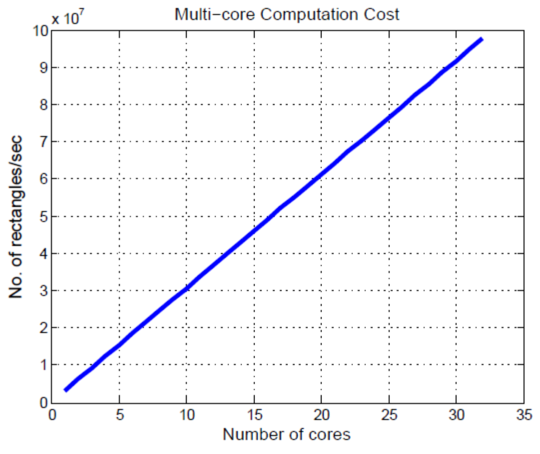
8.1.2 | Multi-core

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 8 a and 8 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.

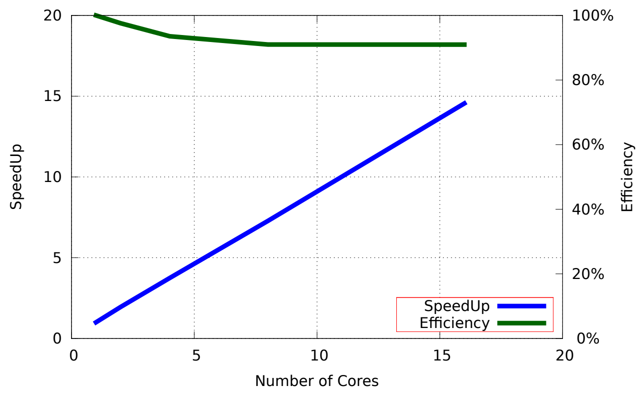
The number of rectangles processed per second is plotted against the number of cores in Figure 8 b and 8 d. The results show the linearity of increasing number of rectangles processed per second with increasing number of cores.



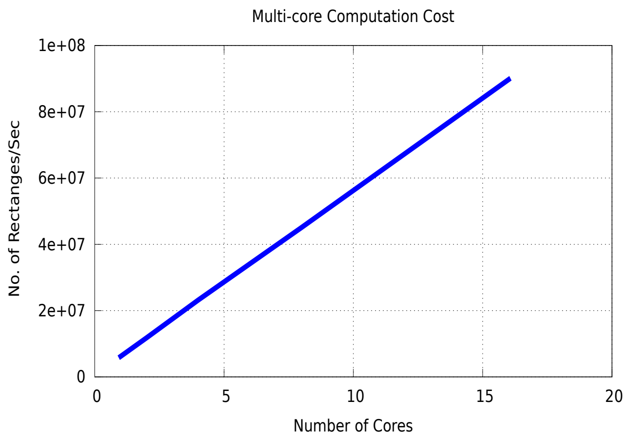
(a) Speed-up and Efficiency of Grid (1000,1000) on Multi-core



(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

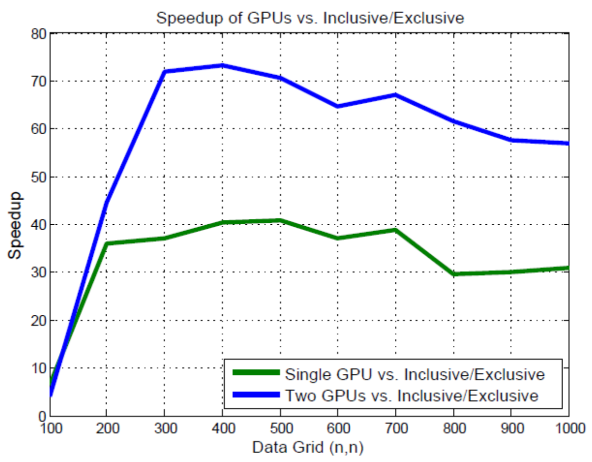
FIGURE 8 Multi-core Evaluation

8.1.3 | Multi-GPGPU

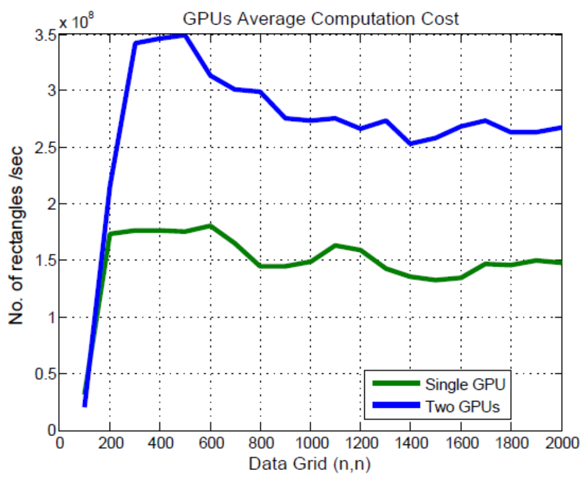
The experiments are conducted on two different GPGPU configurations:

(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

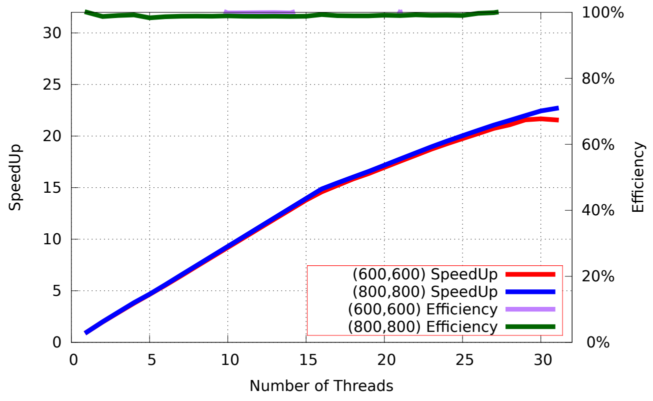
with exclusive/inclusive cpu approach (see figure 9 a) on various data grid size (n,n). Furthermore, the performance of number of rectangles processed per second on one single GPGPU vs. two GPGPUs are compared (see figure 9 b). The results show that LRT computation on two GPGPUs is around 2 times faster 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 9 c and figure 9 d). Figure 9 c shows the linearly increasing speed-up with increasing number of threads on a given grid size. The efficiency is almost 100%. Figure 9 d shows the linear increasing speed of processing number of rectangles with the increasing of threads on a given grid size.



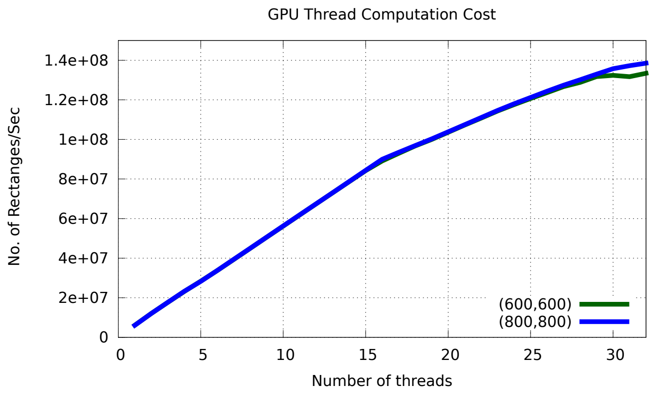
(a) GPGPU speed-up with inclusive/exclusive cpu approach



(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 9 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.

TABLE 2 Optimized Block Configuration

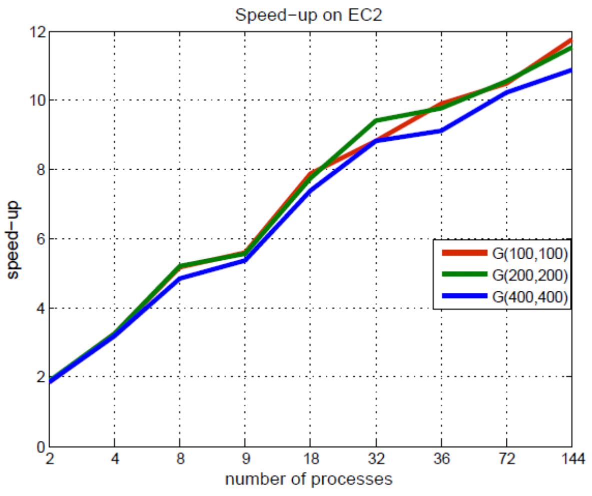
$Grid$	$block_x$	$block_y$
(500,500)	128	64
(600,600)	192	86
(700,700)	176	86

- $((n+1) \times (n+1)/2)^2/(bx \times by) \geq (tx \times ty)$ makes each thread processes at least one region.
- $((n+1) \times (n+1)/2)^2/tx \leq (((n+1) \times (n+1))/2))/ty$ 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 2 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 $20 EC2$ 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 10 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 10 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.



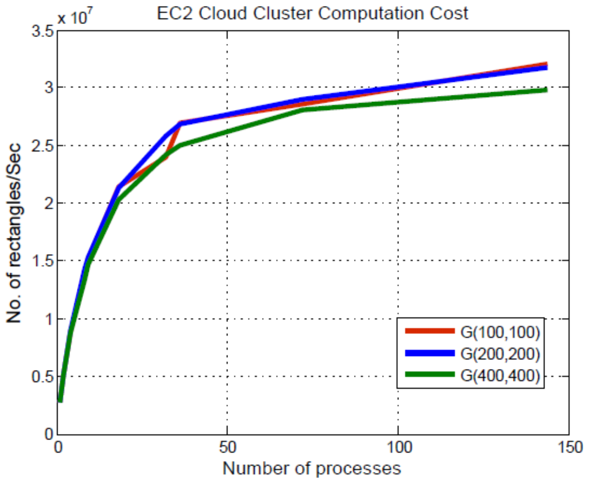


FIGURE 10 Speed-up of LRT computation on EC2 cloud cluster

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 (12). It has been parallelized for multi-core CPU environments and its extension for a GPGPU hardware (13) has achieved improved speed up of two over the multi-core baseline. The GPGPU implementation in (13) has proposed loading parts of the data into shared memory but has achieved only a modest speed up. The other attempt of (14) 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, all the 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.

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 (2). A unified parallel approach will be provided for generalized LRT computation in spatial grids.

SUPPORTING INFORMATION

11 | INCLUSIVE/EXCLUSIVE STATISTICS AGGREGATION

$$|R(x_1, y_1, x_2, y_2)| = |A(x_2, y_2)| + |B(x_1, y_1)| + |X(x_1, y_2)| + |Y(x_2, y_1)| - |G| \tag{34}$$

Proof.

$$|R| = \textstyle\sum_{i=1}^{x_2}\sum_{j=1}^{y_2} c_{i,j} + \sum_{i=x_1}^{n}\sum_{j=y_1}^{n} c_{i,j} + \sum_{i=x_2}^{n}\sum_{j=1}^{y_1} c_{i,j} + \sum_{i=1}^{x_1}\sum_{j=y_2}^{n} c_{i,j} \tag{35}$$

$$- \textstyle\sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j} \tag{36}$$

$$= \textstyle\sum_{i=1}^{x_1}\sum_{j=1}^{y_2} c_{i,j} + \sum_{i=x_1}^{x_2}(\sum_{j=1}^{y_1} c_{i,j} + \sum_{j=y_1}^{y_2} c_{i,j}) + \sum_{i=x_1}^{n}\sum_{j=y_1}^{n} c_{i,j} \tag{37}$$

$$+ \textstyle\sum_{i=x_2}^{n}\sum_{j=1}^{y_1} c_{i,j} + \sum_{i=1}^{x_1}\sum_{j=y_2}^{n} c_{i,j} \tag{38}$$

$$= \textstyle\sum_{i=1}^{x_1}\sum_{j=1}^{n} c_{i,j} + \sum_{i=x_1}^{x_2}\sum_{j=1}^{y_1} c_{i,j} + |R| + |R| + \sum_{i=x_1}^{x_2}\sum_{j=y_2}^{n} c_{i,j} \tag{39}$$

$$+ \textstyle\sum_{i=x_2}^{n}\sum_{j=1}^{n} c_{i,j} - \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j} \tag{40}$$

$$= \textstyle\sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j} + |R| - \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j} \tag{41}$$

$$= |R| \tag{42}$$

□

Definition 5.

$$A(i,j) = \sum_{k=1}^{i}\sum_{l=1}^{j} c_{i,j} \quad \text{for all } 1 \le i,j \le n. \tag{43}$$

Lemma 10.

$$A(i,j) = A_{i-1,j} + A_{i,j-1} - A_{i-1,j-1} + c_{i,j} \quad \textit{for all } 1 < i,j \le n. \tag{44}$$

Proof.

$$\begin{aligned}
A(i,j) &= \sum_{k=1}^{i-1}\sum_{l=1}^{j} c_{k,l} + a_{i,j-1} - a_{i-1,j-1} + c_{i,j} &(45)\\
&= \sum_{k=1}^{i-1}\left(\sum_{l=1}^{j-1} c_{k,l} + c_{k,j}\right) + a_{i,j-1} - \sum_{k=1}^{i-1}\sum_{l=1}^{j-1} c_{k,l} + c_{i,j} &(46)\\
&= \sum_{k=1}^{i-1} c_{k,j} + c_{i,j} + a_{i,j-1} &(47)\\
&= \sum_{k=1}^{i} c_{k,j} + a_{i,j-1} &(48)\\
&= \sum_{k=1}^{i} c_{k,j} + \sum_{k=1}^{i}\sum_{l=1}^{j-1} c_{k,l} &(49)\\
&= A(i,j) &(50)
\end{aligned}$$

□

Definition 6.

$$X_{i,j} = \sum_{k=i}^{n}\sum_{l=1}^{j} c_{i,j} \quad \text{for all } 1 \le i,j \le n. \tag{51}$$

Lemma 11.

$$X_{i,j} = X_{i,j-1} + X_{i+1,j} - X_{i+1,j-1} + c_{i,j} \quad \textit{for all } 1 < i,j \le n. \tag{52}$$

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 7.

$$X_{i,j} = \sum_{k=i}^{n}\sum_{l=1}^{j} c_{i,j} \quad \text{for all } 1 \le i,j \le n. \tag{53}$$

Lemma 12.

$$X_{i,j} = X_{i,j-1} + X_{i+1,j} - X_{i+1,j-1} + c_{i,j} \quad \textit{for all } 1 < i,j \le n. \tag{54}$$

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 8.

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

Lemma 13.

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

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

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

[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: $B(n-1, n-1) \leftarrow G(n-1, n-1)$
3: //accumulation of remaining elements in last column
4: for j←(n-1) to 1 do
5:     $B(j-1, n-1) \leftarrow G(i-1, n-1) + B(j, n-1)$
6: end for
7: //accumulation of remaining elements in last row
8: for i←(n-1) to 1 do
9:     $B(n-1, i-1) \leftarrow G(n-1, i-1) + B(n-1, i)$
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] Horner Scheme .

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

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

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

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

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

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*.