Fast LRT Implementation on Parallel Computer Architectures

** *,** * and ** *

Abstract—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 $O(n^4)$ 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 of this approach and a performance analysis given.

Index Terms—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 an $n \times n$ spatial grid, the execution time for identifying the most anomalous region is $\mathcal{O}(n^4)$. 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. While performance is greatly improved compared to the naive approach on a CPU, the granularly 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 ensure workload on each "parallel computing unit" (PCU) [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 PCU 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.

- *M. Shell is with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332. E-mail: see http://www.michaelshell.org/contact.html*
- *J. Doe and J. Doe are with Anonymous University.*

1. The experiment results were reported in 2009.

2. We define the (computational) granularity of a parallel architecture as the largest "computational 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

- A dynamic pre-computation scheme is presented for fast computing.
- A kbest reduction strategy is presented for accumulating distributed results on each "PCU" 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. Related work is given in Section 8. In Section 3, we explain how we use Inclusive/Exclusive rule and dynamic programming to speed up the enumeration and processing of regions measurements. In Section 4, we propose a novel range mapping scheme for 1EXP LRT computation for multi-dimensional data grid. To produce the final top-k regions, each parallel portion generate top k results and kbest reduction is done on CPU. The proof is provided in Section 5. The details of the implementation on Multi-core, GPGPU and EC2 cloud cluster are presented in Section 6. In Section 7, we evaluate experiments' results on these different architectures and discussion is given . We give our conclusions in Section 9.

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 $\Lambda \equiv -2\log\lambda$ 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.

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

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

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

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:

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

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].

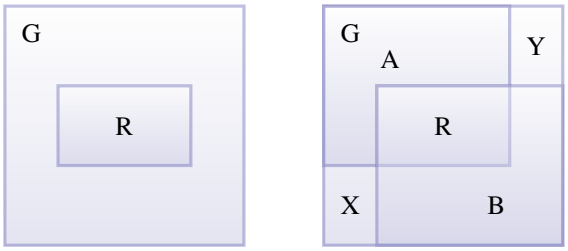
3 INCLUSIVE/EXCLUSIVE PRE-COMPUTATION SCHEME

3.1 Inclusive/Excusive Scheme

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 $O(n^2)$ and then compute the statistic of any region R in $O(1)$ time.

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 directly 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= $A \cap B$, $\overline{A} \cap \overline{B} = X \cup Y$

Fig. 1: Set Relations for Region Problem

Consider Figure 1(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 1(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 $R(x_1, y_1, x_2, y_2)$, where (x_1, y_1) is the upper left corner and (x_2, y_2) is the lower right corner is expressed by (see proof in appendix):

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

3.2 Dynamic Pre-Computation

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 2a, 2b, 2c and 2d. For example, the table for A can be computed using the following recurrence relationship:

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

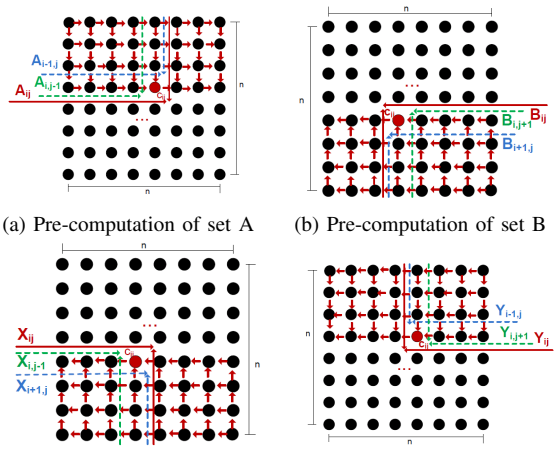
where $|G(i,j)|$ counts whether there is an element in the cell location (i,j). The first element and the first column and row need to be populated (initialized) so that all cardinalities of A can be computed. The counts in the remaining rows and columns are accumulated through the dependency of the previous row and column. Figure 2a shows the computation of set A and the implementations of it is listed in Algorithm 13 (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:

$$|B(i,j)| = |B(i+1,j)| + |B(i,j+1)| - |B(i+1,j+1)| + |G(i,j)| \tag{6}$$

$$|X(i,j)| = |X(i,j+1)| + |X(i-1,j)| - |X(i-1,j+1)| + |G(i,j)| \tag{7}$$

$$|Y(i,j)| = |Y(i+1,j)| + |Y(i,j-1)| - |Y(i+1,j-1)| + |G(i,j)| \tag{8}$$



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

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

Fig. 2: 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 Range Mapping Scheme

The main problem we study in this section is how to parallelize the enumeration of all rectangular regions (R) in a spatial grid (G).

Firstly, we propose a range mapping scheme to transform all of the pairwise intervals between data points from non-consecutive space to consecutive space for n data points in one dimension. Then we extend this scheme to two dimensional spatial grid and multi-dimensional grid. After transformation, the consecutive space enables the overall intervals/regions to be partitioned into equal portions and distributed onto each "parallel computing unit" (PCU).

4.1 One Dimensional Point Set

In one dimension, axis oriented rectangles become intervals that have both endpoints in the unit interval and we assume $X = \{x_0, x_1, x_2, ..., x_n\}$ is a set of $(n+1)$ distinct points.

Algorithm 1 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** $j \leftarrow 1$ to n **do**
5: $A(0,j) \leftarrow G(0,j) + A(0,j-1)$
6: //accumulation of remaining elements in first column
7: **for** $i \leftarrow 1$ to n **do**
8: $A(i,0) \leftarrow G(i,0) + A(i-1,0)$
9: //accumulation of all the elements in remaining rows and columns
10: **for** $k \leftarrow 1$ to n **do**
11: **for** $i \leftarrow$k to n **do**
12: $A(i,k) \leftarrow G(i,n+k) + A(i-1,k) + A(i,k-1) - A(i-1,k-1)$
13: **for** $j \leftarrow$k to n **do**
14: $A(k,j) \leftarrow G(k,n+j) + A(k,j-1) + A(k-1,j) - A(k-1,j-1)$

Algorithm 2 Brute-force Range Mapping in 1-d array

Input: n data points
Output: LRT of all the intervals among n points

1: **for** $i \leftarrow 0$ to *n-1* **do**
2: **for** $j \leftarrow$i to *n-1* **do**
3: *LRT computation for interval i,j*

Algorithm 3 Parallel Interval Enumeration in 1-d array

Input: n data points
Output: LRT of all the intervals in parallel

1: *F*or each i $\in \{0, .., \frac{n}{2} - 1\}$,
2: //The interval (i,j) is determined through two independent loops
3: //loop 1
4: **for** $j \leftarrow$i to *n-1* **do**
5: *LRT computation for interval (i,j)*
6: //loop 2
7: **for** $j \leftarrow$n-i to *n-1* **do**
8: *LRT computation for interval (i,j)*

To enumerate all of the pairwise points between these points, the interval set is defined as: $\mathbb{R} = \{I(i,j) | 0 \le i \le n, i \le j \le n\}$ in 2D coordinate system. We assume each interval $I(i,j)$ is corresponding to a point (i,j) in 2D coordinate system. Regions are formed by these points.

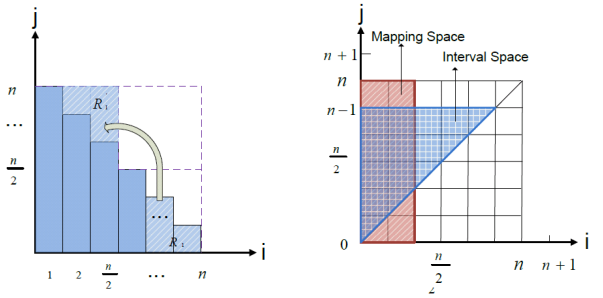
Definition 1: **Interval I(i,j):** An interval $I(i,j) \in \mathbb{R}$ represents the distance from start point x_i to end point $x_{(j+1)}$. For example, to denote the interval from the start point x_0 to the end point x_1, we define it as $I(0,0)$. $I(0,1)$ denotes the interval from the start point x_0 to the end point x_2, etc.

Definition 2: **Closed Bounded Region** $CBR(i,j) \rightarrow CBR(i_k,j_k) ... \rightarrow CBR(i_n,j_n)$: This closed bounded region represents the closed area from point (i,j) to (i_k,j_k) ,..., at last to end point (i_n,j_n).

Lemma 1: **Range Mapping-1D:** In one dimensional space, there has a bijective function $f_{1d} : I(i,j) \rightarrow I(i',j')$, where each interval $I(i,j) \in \{0 \le i \le n, i \le j \le n\}$ is mapped to by exactly one interval $I(i',j') \in \{0 \le i' < \frac{n}{2}, 0 \le j' \le n\}$.

Proof: The cardinality of the intervals in one dimensional point set X is $|R| = \frac{n \cdot (n+1)}{2}$. To visualize, each different interval (i,j) from set X is plotted out in Figure 3a in 2D coordinate system. It is clearly to be seen that this domain has triangular shape and the total amount of intervals can be split into the product of $\frac{n}{2}$ and $(n+1)$. It has the exactly same amount of all the points within the rectangular space of $[\frac{n}{2}, n+1]$. Therefore, there exists a one-to-one onto mapping of the data points in the triangular shaped space to the data points in the rectangular shaped space.

Figure 3b shows the original point (interval) space and the transformed point (interval) space. Both of them consist of the same number of points, which creates the same amount of intervals. The implementations of brute-force approach and workable parallel approach from (i,j) to (i,j') are shown in the Algorithm 2 and 3. The detailed transform approach is given in the following part.



(a) (b)

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

□

Lemma 1 is applicable only when the number of the points in set X is even, to make it more applicable, we extend it workable for set X having even/odd number of points.

Lemma 2: **Range Mapping-EO:** There has a injective function $f_{eo} : (i,j) \rightarrow (i',j')$, where each interval $I(i,j) \in \{0 \le i \le n, i \le j \le n\}$ is mapped to at most one interval $I(i',j') \in \{0 \le i' \le \lfloor \frac{n+1}{2} \rfloor, 0 \le j' \le n+1\}$.

Proof: We know the cardinality of the intervals in one dimensional point set X is $|R| = \frac{n \cdot (n+1)}{2}$. This is the product of $(n+1)$ and $\frac{n}{2}$. Dependent on n, the number of points in rectangular shaped space generated by the second part $\frac{n}{2}$ together with $(n+1)$, will be less than or equal to the total number of points required by triangular shaped space. If we extend $\frac{n}{2}$ to $\frac{(n+1)}{2}$ ($\frac{n}{2} \le \lfloor \frac{n+1}{2} \rfloor$), each interval

Algorithm 4 Parallel Range Mapping in 1-d array

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

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

included in the triangular shaped space is exactly mapped to one point in the transformed rectangular space.

Figure 4a and Figure 4b show the solutions and implementation is shown in Algorithm 4. There are points $\{x_0, x_1, x_2, ..., x_n\}$, to transform the rectangular range space to the triangular space, two scenarios and corresponding transformation steps are considered:

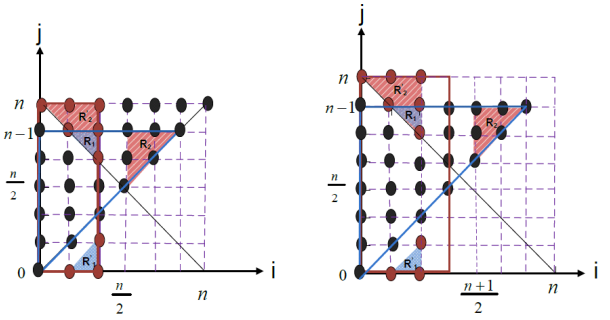
Scenario I: When there are even number of intervals (i.e. $(n+1)\ mod 2 \equiv 0$), the rectangular shaped space and triangular shaped space are shown in Figure 4a and bounded by:
$CBR_{rec} = CBR_{(0,0)} \rightarrow CBR_{(\frac{n}{2}-1,0)} \rightarrow CBR_{(\frac{n}{2}-1,n)}$ $\rightarrow CBR_{(0,n)}$ and
$CBR_{tri} = CBR(0,0) \rightarrow CBR_{(n-1,n-1)} \rightarrow CBR_{(0,n-1)}$.
To perform mapping, two lines $i = j$ and $i + j = n$ divide CBR_{rec} into three parts: $(1)T_1 = \{(i,j)|(i+j) \geq n\}$, $(2)T_2 = \{(i,j)|j<i\}$, $(3)T_3 = \overline{T1 \cap T2}$. The steps are:

- $\forall (i,j) \in T_2$ are shifted upwards by $|i|$ into CBR_{tri}. $R_1{}'$ are transformed to R_1 in the figure.
- $\forall (i,j) \in T_1$ are shifted downwards by $|j - n + 1|$ and shifted rightwards in a by $|2(\frac{(n)}{2} - i) - 1|$. $R_2{}'$ is transformed to R_2 in the figure.
- $\forall (i,j) \in T_3$ are shifted upwards by $|i|$ until reaching line $i + j = n$.

Scenario II: When there are odd number of intervals (i.e. $(n+1)\ mod 2 \neq 0$), the rectangular and triangular shaped space are shown in Figure 4b and bounded by:
$CBR_{rec} = CBR_{(0,0)} \rightarrow CBR_{(\frac{(n-1)}{2},0)} \rightarrow CBR_{(\frac{(n-1)}{2},n)}$ $\rightarrow CBR_{(0,n)}$ and
$CBR_{tri} = CBR(0,0) \rightarrow CBR_{(n-1,n-1)} \rightarrow CBR_{(0,n-1)}$.
The steps are almost same as the steps for transforming even number of intervals. The only difference is the intervals in the rectangular space of $CBR_{(0,0)} \rightarrow CBR_{(\frac{(n-1)}{2}-1,0)} \rightarrow CBR_{(\frac{(n-1)}{2}-1,n)}$ $\rightarrow CBR_{(0,n)}$ are transformed. The intervals $(i,j) \in CBR_{tri}$ when $j = (n+1)/2$ are remained at their positions.

□

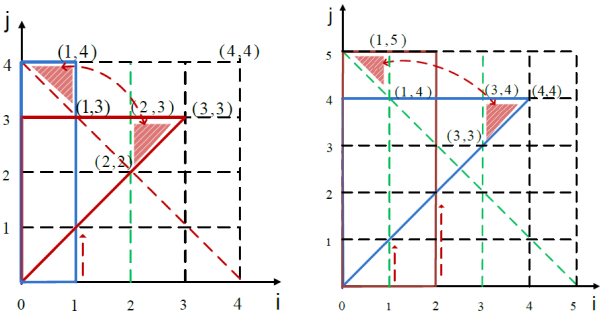
Example: To illustrate the transformation, Figure 5a and 5b show small examples on how to transform the



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

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

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



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

Fig. 5: Example: 1-d Interval Transformation

4.2 Two Dimensional Region Set

Lemma 3: **Range Mapping-2D:** In two dimensional grid (m,n), there has a injective function $f_{2d} : R(i_1, j_1, i_2, j_2) \rightarrow R(i_1', j_1', i_2', j_2')$ is mapped to at most one region:
$\{(i_1', j_1', i_2', j_2') | 0 \leq i_1' \leq \lfloor \frac{m+1}{2} \rfloor, 0 \leq j_1' \leq m+1,$
$0 \leq i_2' \leq \lfloor \frac{n+1}{2} \rfloor, 0 \leq j_2' \leq n+1\}$.

Proof: We know that a region is bounded by two vertical lines $(x = i_1, x = i_2)$ and two horizontal lines $(y = j_1, y = j_2)$. The coordinates of x and y are independent of each other for defining a region. That means (i_1, i_2) is independent to (j_1, j_2). From above, there has injective function to transform each interval in

$from(i,j)$	$to(i',j')$
(0,0)	(0,0)
(0,1)	(0,1)
(0,2)	(0,2)
(0,3)	(0,3)
(0,4)	(3,3)
(1,4)	(2,2)
(1,3)	(2,3)
(1,0)	(1,1)
(1,1)	(1,2)
(1,2)	(1,3)

(a) Grid (4,4)

$from(i,j)$	$to(i',j')$
(0,0)	(0,0)
(0,1)	(0,1)
(0,2)	(0,2)
(0,3)	(0,3)
(0,4)	(0,4)
(0,5)	(4,4)
(1,0)	(1,1)
(1,1)	(1,2)
(1,2)	(1,3)
(1,3)	(1,4)
(1,4)	(3,4)
(1,5)	(3,3)
(2,0)	(2,2)
(2,1)	(2,3)
(2,3)	(2,4)

(b) Grid (5, 5)

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

one dimension. Therefore, all rectangular regions in grid (m, n) can be transformed to 2D rectangular region with $\{(\lfloor\frac{m+1}{2}\rfloor, m+1), (\lfloor\frac{n+1}{2}\rfloor, n+1)\}$. □

4.3 Multi-Dimensional Point Set

Lemma 4: **Range Mapping-mD:** In multi-dimension grid $(n_1, n_2, ..., n_m)$, there has a injective function f_{md} : $R(i_1, j_1, i_2, j_2, ..., i_m, j_m) \rightarrow R(i_1', j_1', i_2', j_2', ..., i_m', j_m')$ is mapped to by at most one region:
$\{(i_1{}', j_1{}', i_2{}', j_2{}', ..., i_m{}', j_m{}')$
$|0 \leq i_1{}' \leq \lfloor\frac{n_1+1}{2}\rfloor, 0 \leq j_1{}' \leq n_1+1,$
$0 \leq i_2{}' \leq \lfloor\frac{n_2+1}{2}\rfloor, 0 \leq j_2{}' \leq n_2+1, ...,$
$0 \leq i_m{}' \leq \lfloor\frac{n_m+1}{2}\rfloor, 0 \leq j_m{}' \leq n_m+1\}.$

5 KBEST REDUCTION SCHEME

To find top-k[4] anomalous rectangular regions, a heap with maximium size of k is built from each "parallel computing unit" (PCU). A further reduction strategy is applied on these k heaps to get final kbest regions.

A LRT value set $s = s_1, s_2, .., s_i, ..., s_n$ is generated from n rectangular regions set $\mathbb{R}$. $\{R\}$ 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 $kbest$ 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, $s_{r0} \leq s_{r1}, ..., \leq s_{rk}$, $s_{i0} \leq s_{i1}, ..., \leq s_{ik}$.

Lemma 5: $KBestReduction$: The $kbest$ LRT values from the value set s is equal to the $kbest$ values obtained by performing $kbest$ reduction from each parallel portion: $h(p_1 \cup p_2 \cup ... \cup p_i \cup ... \cup p_n) = h(h(p_1) \cup h(p_2) \cup ... \cup h(p_n))$.

4. We use top-k and kbest exchangeably.

We give proof by contradiction:

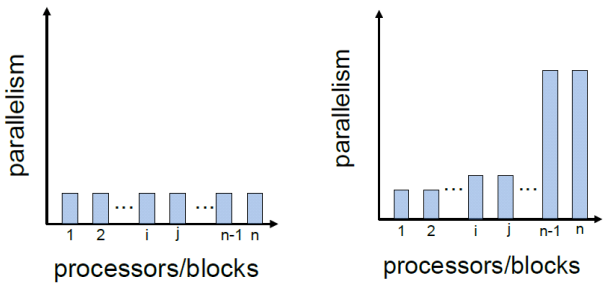
(a) $\forall x \in h(p_1 \cup p_2 \cup ... \cup p_i \cup ... \cup p_n) \rightarrow x \in (h(p_1) \cup h(p_2) \cup ... \cup h(p_n)) \rightarrow x \in h(h(p_1) \cup h(p_2) \cup ... \cup h(p_n))$.
$\because$ Total number of $kbest$ values of $|(h(p_1) \cup h(p_2) \cup ... \cup h(p_n)| \geq k$.
And $x \notin h(p_1) \cup h(p_2) \cup ... \cup h(p_n) \rightarrow x \notin h(p_1 \cup p_2 \cup ... \cup p_i \cup ... \cup p_n)$.
Furthermore, since x is one of the topk value from all the results, therefore, $x \in h((h(p_1) \cup h(p_2) \cup ... \cup h(p_n)))$

(b) $\forall x \in h(h(p_1) \cup h(p_2) \cup ... \cup h(p_n)) \rightarrow x \in h(p_1) \cup h(p_2) \cup ... \cup h(p_n) \rightarrow x \in h(p_1 \cup p_2 \cup ... \cup p_i \cup ... \cup p_n)$
$\because h(h(p_1) \cup h(p_2) \cup ... \cup h(p_n)) \subseteq h(p_1) \cup h(p_2) \cup ... \cup h(p_n) \subseteq h(p_1 \cup p_2 \cup ... \cup p_i \cup ... \cup p_n)$

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. Please see algorithm

6 IMPLEMENTATIONS ON PARALLEL AND DISTRIBUTED ARCHITECTURE

From the above discussion of 2D range mapping scheme, the total rectangular regions can be transformed 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 multi-core, pc-cluster and GPGPU architectures. The effects of coarse-grained and fine-grained parallelism are shown in Figure 6a and 6b.



(a) Workload Balance of Rectangular Range Mapping Scheme

(b) Workload Imbalance of Triangular Range Scheme

Fig. 6: The parallelism comparison between rectangular and triangular range mapping scheme applied to enumeration of regions

We implemented the range mapping scheme on different parallel and distributed architectures: Multi-core, GPGPU and EC2 Cloud Cluster.

6.1 Naive Approach

Firstly, we introduce the naive approach of LRT computation on a two-dimensional grid (n, n). In spatial anomaly detection, 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

Algorithm 5 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: Sorting the scores and return top-k regions R
```

Algorithm 6 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)
```

Algorithm 7 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: parallel processing (see the algorithm of parallel part)
17: //get kbest regions
18: kbest ←heap(kbest[0,...,num_threads])
```

Algorithm 8 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)
```

treated as potential outliers. The implementation is shown in algorithm 5. We can see the computation complexity of brute-force searching is $\mathcal{O}(n^4)$ and the computation complexity of single LRT for a region R is $\mathcal{O}(cn^2)$. Therefore, the total computation complexity of naive approach in 1EXP family is $\mathcal{O}(cn^6)$.

6.2 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 6.

6.3 Multi-core Approach

Multi-core computers, 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 7 and algorithm 8. 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.

6.4 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 [9]. A CUDA program consists of one or more phases that are executed on either the host (CPU) or the device (GPU). The host code executed on CPU exhibits little or no data parallelism and the device code (called kernel) executed on GPU side exhibits high amount of data parallelism. The kernel uses large number of threads to exploit data parallelism. The CUDA threads are extremely light weight and require very few clock cycles to be 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 first and extends to $(n+1)(n+1)/2$ range. 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 transfer 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 implementation, we divide the whole workload equally to each GPGPU. Final $kbest$ regions are obtained after each device return back all the $kbest$ heaps.

6.5 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 10.

Algorithm 9 GPU kernel

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)
```

Algorithm 10 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: MPI_Finalize()
```

7 Experiments and Analysis

7.1 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?

7.1.1 Naive vs. Inclusive/Exclusive Implementation

A naive approach requires $\mathcal{O}(n^2)$ computation for a given region. In order to improve the computation efficiency, an Inclusive/Exclusive pre-computation approach is devised, which reduces the 1EXP-LRT computation for each region to $\mathcal{O}(1)$. We run the two implementations on an 8-core E5520 Intel server to compare their performance. The data grid varies from $(100, 100)$ to $(1000, 1000)$. In the Inclusive/Exclusive approach, pre-computed sets take up $\mathcal{O}(4n^2)$ space and only the $kbest$ results kept in heap structure at the end, therefore the total space complexity is $\mathcal{O}(n^2)$. For a naive approach, the computation result of each region is stored into memory and $kbest$ regions are obtained by sorting all of the results at the end. The total results takes $\mathcal{O}(n^4)$ space. Due to large memory space required and long computing, the naive approach was only be able to generate the $kbest$ results from $(100, 100)$ to $(300, 300)$.

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

7.1.2 Multi-core

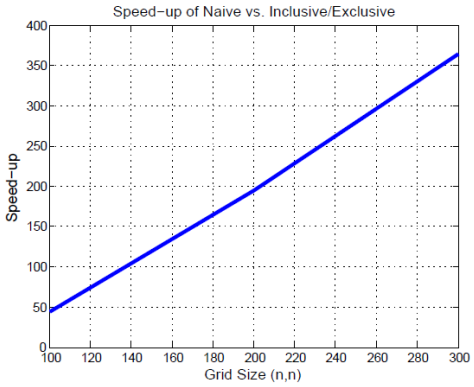
The multi-core experiment is conducted on a 32-core AMD Opteron(tm) Processor 6128 server with 128GB of RAM. We use data grid with size of (1000,1000) to show the performance scaling. Figure 8 shows that the 1EXP-LRT computation scales very well on multi-core architecture. The speedup increases near-linearly 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 (see Figure 8(a)).

The number of rectangles processed per second is plotted against the number of cores in Figure 8(b).

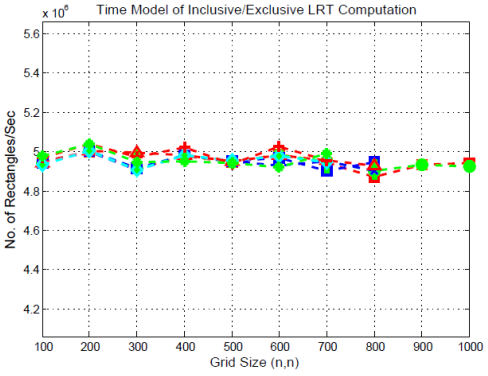
7.1.3 Multi-GPGPU

The experiments were conducted on 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. See Figure 9a and 9b.The results show that LRT computation on two GPGPUs is around 2 times faster than that on one single GPGPU. And the computation cost of per rectangle is almost constant with the increase of data grid for single GPGPU and two GPGPUs.

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



(a) Running time of Naive vs. Inclusive/Exclusive Approach



(b) No. of Rectangles per second in Inclusive/Exclusive Approach

Fig. 7: Naive vs. Inclusive/Exclusive Approach

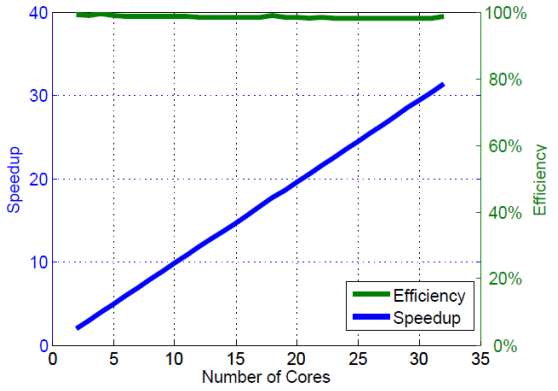
TABLE 2: Optimized Block Configuration

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

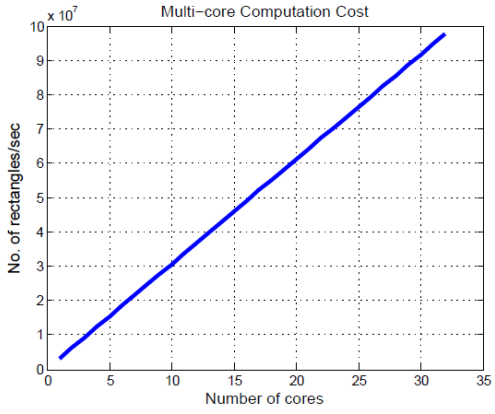
region object takes $8byte$. To fully utilize the shared memory, each block has $(tx, ty) = (16, 8)$ threads. We vary the value of (bx, by) for the grids with different size to find the optimal block configuration. To maximize performance:

- $Max(blocks) \leq (bx \times by) \leq ((n+1) \times (n+1)/2)^2$, 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.
- $((n+1)\times(n+1)/2)^2/tx \leq (((n+1)\times(n+1))/2))/ty$ 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. Figure 11 shows the speedup curves using the slowest run for a given grid size as baseline and Table 2 gives the optimal block configuration. The speedup curves tell us that a speedup of up to two can be achieved by choosing the right block configuration.

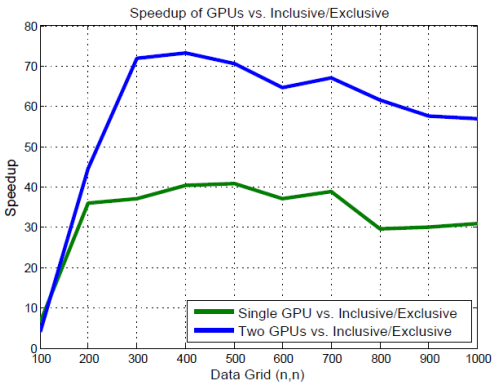


(a) Speed-up and Efficiency of Grid (1000,1000) on Multi-core

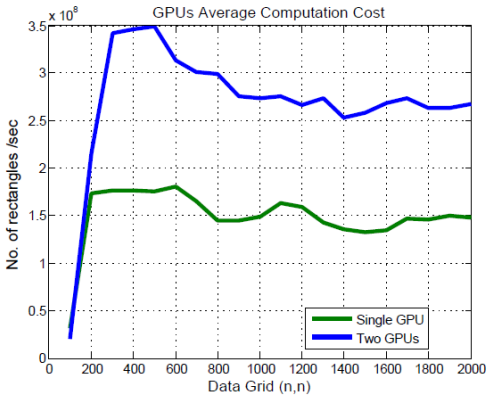


(b) The number of rectangles processed per second vs. number of cores on Grid (1000,1000)

Fig. 8: Multi-core Evalution

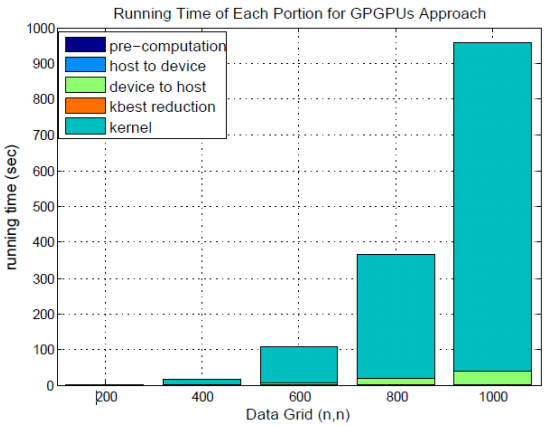


(a)



(b)

Fig. 9: GPGPUs Cost



(a)

Fig. 10: GPGPUs Breakdown parts

7.1.4 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 12a 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 12b, 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.

7.2 Discussion

From the above results, we further analyze of LRT computation on different architectures. Table **??** shows the related factors to the LRT performance on different architectures, the speedup and processing speed of rectangles from these factors. 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 13. 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. As the number of cores and the number of processes increase, the LRT processing speed is improved.

8 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

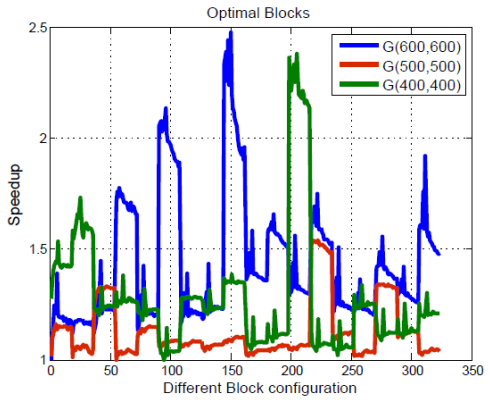
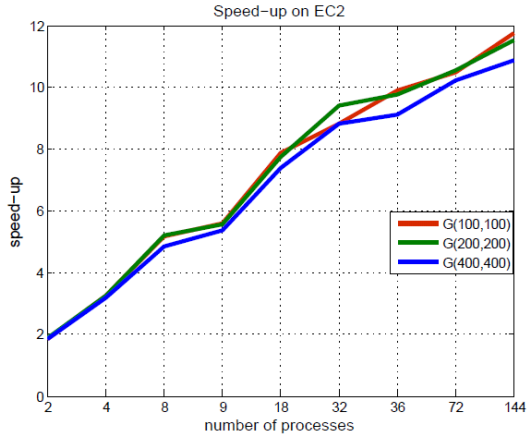
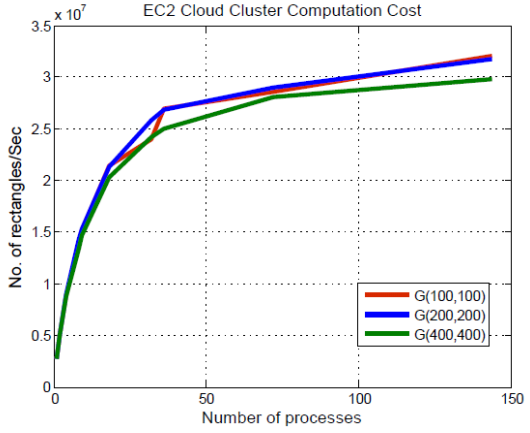


Fig. 11: GPGPUs Optimal Blocks Configuration



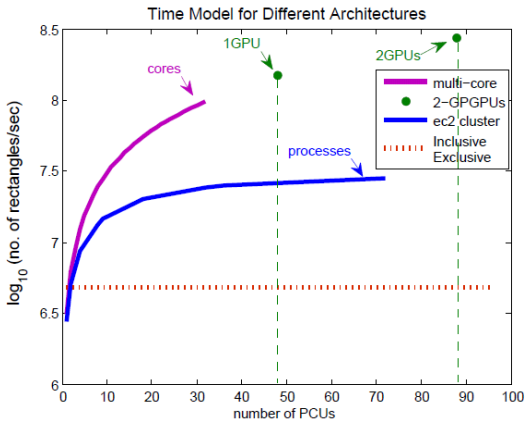
(a)



(b)

Fig. 12: Speed-up of LRT computation on EC2 cloud cluster

data, is available as a program under the name SatScan [10]. It has been parallelized for multi-core CPU environments and its extension for a GPGPU hardware [11] has achieved improved speed up of two over the multi-core baseline. The GPGPU implementation in [11] has proposed loading parts of the data into shared memory but has achieved only a modest speed up. The other attempt of [12] applied their own implementation of a spatial scan statistic program on



(a)

Fig. 13: Time Model of Different Architectures

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.

9 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 unit" 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.

APPENDIX A
INCLUSIVE/EXCLUSIVE STATISTICS AGGREGATION

$$|R(x_1,y_1,x_2,y_2)| = |A(x_2,y_2)| + |B(x_1,y_1)| + |X(x_1,y_2)| + |Y(x_2,y_1)| - |G| \tag{9}$$

Proof:

$$\begin{aligned}
|R| &= \sum_{i=1}^{x_2}\sum_{j=1}^{y_2} c_{i,j} + \sum_{i=x_1}^{n}\sum_{j=y_1}^{n} c_{i,j} + \sum_{i=x_2}^{n}\sum_{j=1}^{y_1} c_{i,j} + \sum_{i=1}^{x_1}\sum_{j=y_2}^{n} c_{i,j} - \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j} \\
&= \sum_{i=1}^{x_1}\sum_{j=1}^{y_2} c_{i,j} + \sum_{i=x_1}^{x_2}(\sum_{j=1}^{y_1} c_{i,j} + \sum_{j=y_1}^{y_2} c_{i,j}) + \sum_{i=x_1}^{n}\sum_{j=y_1}^{n} c_{i,j} \\
&+ \sum_{i=x_2}^{n}\sum_{j=1}^{y_1} c_{i,j} + \sum_{i=1}^{x_1}\sum_{j=y_2}^{n} c_{i,j} \\
&= \sum_{i=1}^{x_1}\sum_{j=1}^{n} c_{i,j} + \sum_{i=x_1}^{x_2}\sum_{j=1}^{y_1} c_{i,j} + |R| + |R| + \sum_{i=x_1}^{x_2}\sum_{j=y_2}^{n} c_{i,j} \\
&+ \sum_{i=x_2}^{n}\sum_{j=1}^{n} c_{i,j} - \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j} \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j} + |R| - \sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j} \\
&= |R|
\end{aligned} \tag{10}$$

□

Definition 3:

$$A(i,j) = \sum_{k=1}^{i}\sum_{l=1}^{j} c_{i,j} \quad \text{for all } 1 \le i,j \le n. \tag{11}$$

Lemma 6:

$$A(i,j) = A_{i-1,j} + A_{i,j-1} - A_{i-1,j-1} + c_{i,j} \quad \text{for all } 1 < i,j \le n \tag{12}$$

Proof:

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

□

Definition 4:

$$X_{i,j} = \sum_{k=i}^{n}\sum_{l=1}^{j} c_{i,j} \quad \text{for all } 1 \le i,j \le n. \tag{13}$$

Lemma 7:

$$X_{i,j} = X_{i,j-1} + X_{i+1,j} - X_{i+1,j-1} + c_{i,j} \quad \text{for all } 1 < i,j \le n. \tag{14}$$

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 5:

$$X_{i,j} = \sum_{k=i}^{n}\sum_{l=1}^{j} c_{i,j} \quad \text{for all } 1 \le i,j \le n. \tag{15}$$

Lemma 8:

$$X_{i,j} = X_{i,j-1} + X_{i+1,j} - X_{i+1,j-1} + c_{i,j} \quad \text{for all } 1 < i,j \le n. \tag{16}$$

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 6:

$$y_{i,j} = \sum_{k=1}^{i}\sum_{l=j}^{n} c_{i,j} \quad \text{for all } 1 \le i,j \le n. \tag{17}$$

Lemma 9:

$$Y_{i,j} = Y_{i-1,j} + Y_{i,j+1} - Y_{i-1,j+1} + c_{i,j} \quad \text{for all } 1 < i,j \le n. \tag{18}$$

Proof:

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

□

APPENDIX B
PRE-PROCESSING OF INCLUSIVE/EXCLUSIVE COMPUTATION

Algorithm 11 Inclusive/Exclusive Pre-computation for Set B

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

```
//Initialize first element B(n-1,n-1)
B(n-1,n-1)←G(n-1,n-1)
//accumulation of remaining elements in last column
for j←(n-1) to 1 do
    B(j-1,n-1) ← G(i-1,n-1) +B(j,n-1)
//accumulation of remaining elements in last row
for i←(n-1) to 1 do
    B(n-1,i-1)←G(n-1,i-1) + B(n-1,i)
//accumulation of all the elements in remaining rows and columns
for k←1 to n do
    for i←(n-1) to k do
        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)
    for j←(n-1) to k do
        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)
```

Algorithm 12 Pre-processing of Inclusive/Exclusive Computation for Set X

```
//Initialize first column
for i←1 to n do
    X(i,0)←0
//Initialize last row
for j←1 to n do
    X(n-1,j)←0
//Iterate over all diagonal elements
for k←1 to n do
    //associate columns of diagonal elements
    for i←n-1 to k do
        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)
    // associate columns of diagonal element k
    for j←k to n do
        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)
```

REFERENCES

[1] W. S. S., "The large sample distribution of the likelihood ratio for testing composite hypotheses," *Annals of Mathematical Statistics*, no. 9, pp. 60–62, 1938.

[2] M. Wu, X. Song, C. Jermaine, S. Ranka, J. Gums, "A LRT Framework for Fast Spatial Anomaly Detection," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '09)*, pp. 887–896.

[3] X. L. Pang, S. Chawla, W. Liu, and Y. Zheng., "On mining anomalous patterns in road traffic streams," in *ADMA: In the 7th International Conference on Advanced Data Mining and Applications*, 2011, pp. 237–251.

[4] B. S. X.L. Pang, S. Chawla and G.Wilcox:, "A scalable approach for lrt computation in gpgpu environments," *APWeb*, pp. 595–608, 2013.

[5] R. Vuduc, A. Chandramowlishwaranv, J. Choi, M. Guney, and A. Shringarpure, "On the limits of gpu acceleration," in *HotPar'10 Proceedings of the 2nd USENIX conference on Hot topics in parallelism*, 2010, pp. 237–251.

[6] A. Gregerson, "Implementing fast mri gridding on gpus via. cuda," in *Nvidia Tech. Report on Medical Imaging using CUDA*, 2008.

[7] S. Hong, S. K. Kim, T. Oguntebiv, and K. Olukotun, "Efficient parallel graph exploration on multi-core cpu and gpu," in *PPoPP '11 Proceedings of the 16th ACM symposium on Principles and practice of parallel programming*, 2011.

[8] D. Agarwal, J. M. Phillips, and S. Venkatasubramanian, "The hunting of the bump: On maximizing statistical discrepancy," in *SODA*, 2006, pp. 1137–1146.

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

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

[11] S. G. Larew, R. Maciejewski, I. Woo, and D. S. Ebert., "Spatial scan

Algorithm 13 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: //Initialize first row
5: for j←1 to n do
6:     Y(0,j)←0
7: //Iterate over all diagonal elements
8: for k←1 to n do
9:     //associate columns of diagonal elements
10:    for i←k to (n-1) do
11:        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)
12:    // associate columns of diagonal element k
13:    for j←(n-1) to k do
14:        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)
```

statistics on the gpgpu," in *Proceedings of the Visual Analytics in Healthcare Workshop at the IEEE Visualization Conference*, 2010.

[12] S. S. Zhao and C. Zhou, "Accelerating spatial clustering detection of epidemic disease with graphics processing unit," in *In Proceedings of Geoinformatics*, 2010, pp. 1–6.