A Hybrid MPI+OpenMP Application for Processing Big Trajectory Data

Natalija STOJANOVIC, Dragan STOJANOVIC
University of Nis, Faculty of Electronic Engineering,
14, A. Medvedeva, 18000 Nis, Serbia
natalija.stojanovic@elfak.ni.ac.rs; dragan.stojanovic@elfak.ni.ac.rs

Abstract: In this paper, we present the use of parallel/distributed programming frameworks, MPI and OpenMP, in processing and analysis of big trajectory data. We developed a distributed application that initially performs a spatial join between big trajectory data and regions of interest, and further aggregates join results to provide analysis of movement. The solution was implemented using hybrid distributed/parallel programming model, based on MPI and OpenMP programming interfaces. The experimental evaluation in detecting the most popular places in the city, based on large-scale trajectory dataset, demonstrates the performance gains and feasibility of our approach.

Keywords: High performance computing, Big data processing, Geospatial analysis, MPI, OpenMP.

1. Introduction

Many today's computing and data intensive applications, such as computer games, database and Web searching, financial and economic forecasting, climate modelling, environment monitoring, and bioinformatics, demand acceleration and significant performance improvements [1]. There are several approaches that can be employed to improve the performance of advanced computing and data intensive applications. All of them are based on parallel computing paradigm on single multi/many core computer systems, or over distributed computing infrastructure within a cluster or cloud architecture. Such solutions employ parallel and distributed programming models, frameworks and interfaces such as: multi-core CPU, many-core graphics processing unit (GPU), network/cluster of workstations and PaaS in computer clouds [2].

Advances in remote sensing technologies, sensor networks and the proliferation of mobile devices in everyday use have resulted in an acquisition of massive amounts of geospatial data and moving object trajectories. Also, large-scale geo-scientific modelling and simulations, as well as geo-social network activities (e.g. Twitter and Facebook) are generating petabytes of spatio-temporal data per day. These ever-increasing volumes of spatio-temporal data call for new models and computationally effective algorithms in order to efficiently store, process, analyze and visualize such a big data in advanced data-intensive systems and applications. Recently, high-performance computing (HPC) is promoted to meet the requirements of advanced Geographic Information Systems (GIS) applications [3].

The recent proliferation of distributed and cloud computing infrastructures and platforms, both public clouds (e.g., Amazon EC2) and private computer clouds and computer clusters, has given a further rise for processing and analysis of complex Big data. Especially, the implementation that can work on clusters of multicore shared-memory computers (nodes), have set this paradigm as an emerging research and development topic. In this paper, we employ MPI (Message Passing Interface), a message passing parallel programming over cluster of computers/nodes, and OpenMP for shared memory parallel programming within a node, to implement an application for large scale trajectory data processing and analysis.

The Message Passing Interface (MPI) has become the major model for programming distributed-memory applications. Message passing works by creating processes which are distributed among the group of computing nodes. When a MPI program runs, all processes execute the same code.

OpenMP notation can be added to a sequential program to define how the work can be shared among the threads that execute on different processor's cores and to order access to shared data as needed. OpenMP supports the so-called fork-join and shared memory programming model.

Hybrid MPI+OpenMP approach integrates different levels of parallelism. This approach employs features of a distributed memory using message-passing and a shared-memory using multithreading. MPI is used for process communication between multicore nodes and OpenMP is used for thread communication within a multicore node.

The main contributions of the paper are:

- We propose the use of a hybrid programming model and develop a hybrid MPI+OpenMP application that performs spatial join between trajectory data set and spatial regions around points/places of interest (POI), and further aggregation of join results, to detect the most popular POIs in the city (Popular Places algorithm).
- We perform the experimental evaluation that indicates the improvements in performance with respect to pure MPI-based and sequential (single node) solutions and show feasibility of using hybrid programming model for data-intensive GIS computing.
- We analyze and examine the effects of hybrid MPI/OpenMP implementation and propose hints for large scale spatio-temporal data processing in other GIS domains.

The rest of the paper is structured as follows. Section II presents the research work related to high-performance processing and analysis of large-scale spatial and spatio-temporal data using existing HPC paradigms. In section III we describe the pure MPI and the hybrid MPI+OpenMP implementation for processing of big trajectory data set over set of places of interest (POI). Section IV gives the results and presents the evaluation of hybrid MPI+OpenMP, pure MPI and sequential implementation of Popular Places algorithm. Section V concludes the paper and gives directions for future research.

2. HPC for Big Geospatial Data

Big data is emerging paradigm and research topic related to collection, processing and analysis of huge amounts of structured and unstructured data and its applications in enterprises, scientific and government organisations [4]. Recently, there is also a growing research interest in management, processing analysis and mining of massive geo-spatial data in various GIS domains using high performance parallel and distributed computing methodologies, techniques and platforms [5]. Advanced GIS applications, such as real-time disaster management, high-fidelity terrain visualization, global climate change analysis, traffic monitoring, etc., impose strengthen performance and response time constraints which cannot be met by contemporary GISs and spatial databases. Thus, high-performance computing is promoted to meet the requirements of these applications [3]. Various researchers propose methods and techniques for achieving high performance through parallelization in the processing and analysis of big geospatial data based on cluster and cloud computing [6], as well as on personal computers equipped with multiprocessor CPUs and massively parallel GPUs [7].

Akhter et al. [8] develop a methodology and propose GRASS GIS module extension with parallel and distributed computing for remote sensing image processing. Different implementations for distributed GRASS modules are examined on three different programming platforms (MPI, Ninf-G and OpenMP) and their performance are presented. Zhang in [7] proposes a framework for high-performance processing of geospatial data in a personal computing environment. His approach is based on personal computers equipped with multi-core CPU and many-core GPU that provide excellent support for spatial data processing comparing with cluster and cloud computing approaches, such as MPI and MapReduce. Shi in [9] discusses fundamental research challenges that need to be solved to effectively apply HPC techniques in a service-oriented GIS. Wang et al. in [10] propose a framework for efficient retrieval, indexing and management of spatial data in the cloud environment. They develop a spatial object model, spatial index structures and algorithms for the cloud computing environment, and implement them on Google App Engine cloud environment.

One of the first HPC approach related to spatio-temporal data and massive trajectory management is presented in [11]. The authors present a framework for spatial query processing over trajectory data based on MapReduce in a computer cluster. The experiments performed show the scalability of proposed framework in terms of the size of trajectory data set [12]. SpatialHadoop is developed as the MapReduce extension with support for spatial data types and operations implemented within Hadoop software framework [13]. SpatialHadoop employs a spatial high level language, a two-level spatial index structure, and three basic spatial operations: range queries, k-NN queries, and spatial join. SpatialHadoop demonstration has been done on an Amazon EC2 cluster against two real spatial data sets.

Kunaseth et al. in [14] propose analysed algorithms for hybrid MPI/OpenMP parallelization of molecular-dynamics simulation, which are scalable on large multicore clusters. Proposed hybrid MPI+OpenMP implementation achieves 2.58 and 2.16 speedups with respect to pure-MPI implementation for 0.84 and 1.68 million particle simulations, respectively. In [15] a hybrid MPI+OpenMP approach is proposed for pseudo-spectral computations of fluid turbulence. The implementation scales well up to 20,000 computing cores with a maximum parallel efficiency of 89%. They propose a method to reduce the number of MPI tasks, and increase network bandwidth, resulting in the implementation competitive with the pure MPI-based method.

Although there is a considerable recent research interest related to spatial and spatio-temporal data management on parallel and distributed computing infrastructures, there is limited research work dedicated to trajectory (mobility) data processing and analysis using MPI and OpenMP parallel programming frameworks. Our work aims to provide efficient solution based on MPI and OpenMP for fundamental mobility data processing task related to Big trajectory data sets..

3. MPI and OpenMP in Big Trajectory Data Processing

The research presented in this paper aims to provide high-performance solution for a Big data processing and analysis task related to the trajectory data set representing movement of mobile users and the points/places of interest (POI) they visit.

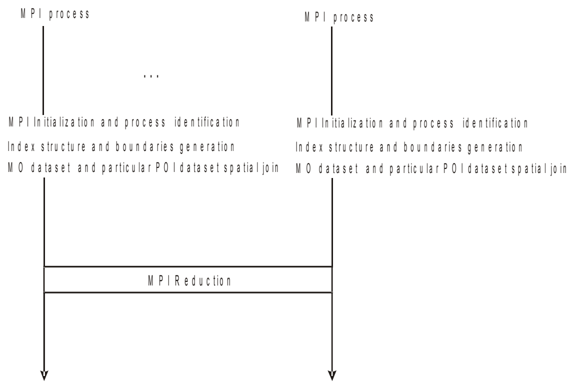
The problem we investigate in this work represents the spatial join between a big set of spatio-temporal, trajectory (mobility) data T and a (potentially large) set of spatial regions R. The trajectory data represent the movement of a large collection of moving objects/mobile users (MO) tracked for a certain time period with the specified frequency of location updates. Each trajectory is seen as a collection of points <oid, xi, yi, ti>, where xi, yi represent the location in a geographic/geometric reference system and ti is the corresponding time stamp at which the moving object (oid) is detected at the specified location. Trajectories of a large number of moving objects collected for a long time period are characterized by large volumes, considered as Big data and therefore their processing and analysis is a challenging issue.

Each spatial region R represents an area around a point/place of interest (POI) visited by mobile users that stay there for certain time periods. A mobile user visits particular POI if its recording location at a corresponding time stamp is within the spatial region R around POI; otherwise a mobile user is considered to be on a trip between two POIs. The objective of our implementation is to provide detection of the most popular places regarding the number of users that visited them. For such analysis we develop efficient application over the trajectory and region data sets. We consider strategy where spatial relation between objects from the first dataset (trajectory dataset containing time stamped locations of moving objects) and objects from the second dataset, places of interest (POI), is analyzed. To improve the performance of spatial join we implement simple, in-memory grid index structure for POI data set. Thus, the area of interest is divided by a grid consisting of rectangular regions of specific dimensions along x and y axes. The grid can be regular, or irregular (quad tree) to efficiently cope with POI data skewness. In this strategy, the POI data set (work) is evenly divided among processes and each process is assigned with POI data located in particular geographical region.

In the pure MPI implementation, we perform uniform splitting of the first dataset (trajectory dataset, MO dataset) where distribution depends on the size of dataset. The first dataset is evenly distributed among processes. Each process is responsible for as possible equal chunk of data. Within each process, the spatial join between moving objects location data in the trajectory dataset and the objects from the POI dataset that are within particular geographical region, is performed. In this manner, each moving object location data from the trajectory dataset is joined, according spatial proximity, with POIs that are within the corresponding grid cell in which both objects reside. Pseudocode and schematic workflow that describes computation steps performed by MPI processes are shown in Figure 1 and 2.

```
Parallel execution of MPI processes
- MPI initialization and process identification
- Generating POI index structure
- Generating boundaries for chunk of MO data assigned to a particular process
- Performing spatial join between MO dataset (inside assigned boundaries) and a particular POI dataset (using obtained quad tree index structure)
- Summarizing results from different processes in order to generate final result using MPI reduction
```

Figure 1. Pseudocode of pure MPI computation



MPI process
MPI process
...
MPI Initialization and process identification
Index structure and boundaries generation
MO dataset and particular POI dataset spatial join
MPI Initialization and process identification
Index structure and boundaries generation
MO dataset and particular POI dataset spatial join
MPI Reduction

Figure 2. Schematic workflow of pure MPI computation

By dividing the spatial area of interest using quad tree index structure, the number of POIs in each spatial rectangular region is at most possible similar but not absolutely equal (Figure 3). Thus, the time spent for performing spatial join operation assigned to each process is slightly different. Therefore, we perform the second level of parallelism, i.e. fine-grained parallelism using threads within the single node and shared memory. To better distribute the iterations assigned to each process, each process spawns multiple threads that are dynamically balanced. Pseudocode and schematic workflow that describes hybrid MPI/OpenMP computation are shown in Figure 4 and 5.

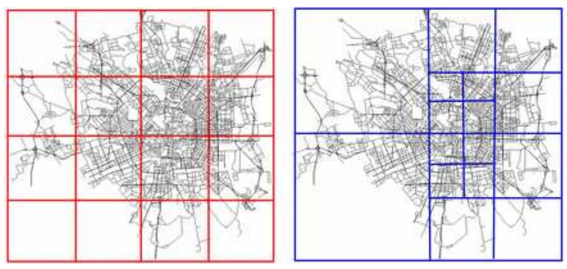
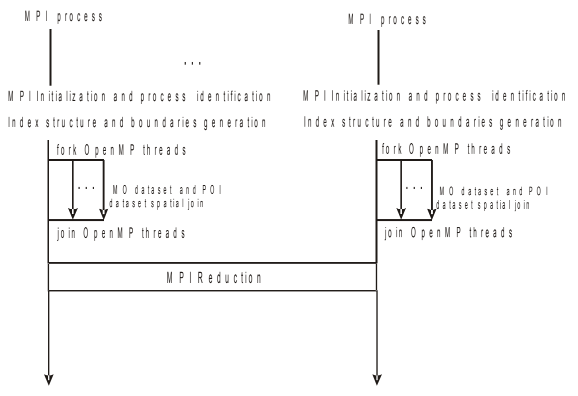


Figure 3. The area of interest (city of Milan) a) regular grid, b) quad tree index structure

```
Parallel execution of MPI processes
- MPI initialization and process identification
- Generating POI index structure
- Generating boundaries for chunk of MO data assigned to particular process
- Performing spatial join between process MO dataset and particular POI dataset by multiple OpenMP threads
- Summarizing results from different processes in order to generate final result using MPI reduction
```

Figure 4. Pseudocode of hybrid MPI+OpenMP computation

In the case of proposed MPI+OpenMP implementation, the number of users that visited a particular POI, which determines popularity of a place, is stored in a shared memory array on each multicore node. Since there is possibility that multiple threads try to modify the same element of the shared array, the synchronization is necessary. The atomic directive is used for that purpose. Also, the access pattern is defined in the form that multiple threads modify the same cache line. This fact can have significant influence on performance degradation.



MPI process
MPI process
...
MPI Initialization and process identification
Index structure and boundaries generation
fork OpenMP threads
...
MO dataset and POI dataset spatial join
join OpenMP threads
MPI Initialization and process identification
Index structure and boundaries generation
fork OpenMP threads
...
MO dataset and POI dataset spatial join
join OpenMP threads
MPI Reduction

Figure 5. Schematic workflow of hybrid MPI+OpenMP computation

Therefore, an array extending, or padding, is introduced. It is achieved by dimensioning array d[n] as d[n][x] where x=cache_line_size/size_of_int, in our case x=16, and by changing the indexing from d[i] to d[i][0] which eliminates false sharing. When all processes finish their computation, summation of all the results assigned for a particular place, is performed. Since the results that must be summarized are placed in columns of the matrix D, respectively, in order to apply MPI reduction, the user defined function for summation is created.

4. Experimental Evaluation

In this section, we present the experimental evaluation of the implemented algorithm. The parallel application has been developed using C++ programming language in Visual Studio 2010 extended with MPICH2 implementation of MPI and with support for OpenMP. We implement and compare two implementations: the pure MPI and the proposed hybrid MPI+OpenMP implementation.

For estimating performances of the proposed solutions we used two system architectures. The first architecture represents a cluster of 16 commodity computers equipped with Intel Pentium G2030 (dual core) 3GHz CPU and 8GB RAM. The second architecture is a private IaaS cloud containing 16 virtual machines equipped with Intel Xeon E5430 (quad core) 2.6GHz CPU and 2GB RAM.

In our study, we use *MilanoByNight* simulated datasets that have been provided by the EveryWare Lab, University of Milano [16] for the implementation and evaluation of large-scale trajectory data processing algorithms. The data correspond to a typical city mobility scenario where a large number of people are tracked during the evening in a city of Milan looking for friends and places for amusement. The simulation includes a total of 30,000 home buildings, 10,000 office buildings and 1,000 entertainment places which represent a POI dataset. The trajectory data set contains 18 million of records for 100,000 mobile users moving over the city of Milan while location updates are made at every 2 minutes. The movement has been recorded over 6 hours long time period, from 7 pm - 1 am, which amounts for about 1.3 GB in total.

We have developed two parallel applications that implement Popular Places algorithm and performed experiments for two implementations using pure MPI and hybrid MPI+OpenMP approach. The algorithm firstly processes spatial join between trajectory data set and spatial regions around points/places of interest (POI), and then perform aggregation of spatial join results. By counting unique moving objects that visited specific POIs the algorithm provides detection of the most popular POIs in the city during the evening. To provide the most possible uniform splitting of the second (POI) data set, we divide the data space of interest into a quad tree which contains 16 disjoint rectangular regions with similar numbers of POIs within each region (Figure 3b). We compare performances of both applications running on 2, 4, 8 and 16 nodes.

To examine scalability issues, we have performed our experiments for two different sizes of the trajectory data set: 1.8 and 18 million moving objects. The number of processes running on processor cores at each node depends on the number of nodes specified in the experiment. For MPI+OpenMP implementation each process spawns corresponding number of worker threads, which depends on the processor architecture. Thus, for the cluster computing architecture which has dual core processors, each MPI process forks two OpenMP threads, while for the cloud architecture nodes, having quad core processors, each MPI process forks four OpenMP threads. The resulting execution times obtained for the cluster architecture of physical machines are shown in Tables 1 and 2 expressed in seconds. The value of S1 represents a speedup of MPI+OpenMP implementation, i.e. $T_{SEQ}/T_{MPI+OMP}$, while S2 is

Table 1. PopularPlaces application run on the cluster for 1.8 millions moving objects

1.8 millions of moving objects				
p	2	4	8	16
T_{SEQ}	40.610	40.610	40.610	40.610
$T_{MPI+OMP}$	11.570	5.789	2.909	1.762
T_{MPI}	18.384	7.942	2.707	2.460
$T_{MPI}/T_{MPI+OpenMP}$	1.588	1.371	0.930	1.395
S1	3.509	7.014	13.957	16.508
S2	2.208	5.113	14.997	14.199

Table 2. *PopularPlaces* application run on the cluster for 18 millions moving objects

18 millions of moving objects				
p	2	4	8	16
T_{SEQ}	409.396	409.396	409.396	409.396
$T_{MPI+OMP}$	113.980	57.123	28.388	14.320
T_{MPI}	183.677	78.550	31.098	25.945
$T_{MPI}/T_{MPI+OpenMP}$	1.611	1.375	1.095	1.811
S1	3.591	7.166	14.421	28.588
S2	2.228	5.211	13.164	15.778

an appropriate speedup of pure MPI implementation, i.e. T_{SEQ}/T_{MPI}.

The speedup of the pure MPI and the hybrid MPI+OpenMP implementation related to sequential *PopularPlaces* application for 18 millions of moving objects is shown in Figure 6. The execution time comparison between the MPI and the MPI+OpenMP implementation for 18 millions of moving objects is shown in Figure 7.

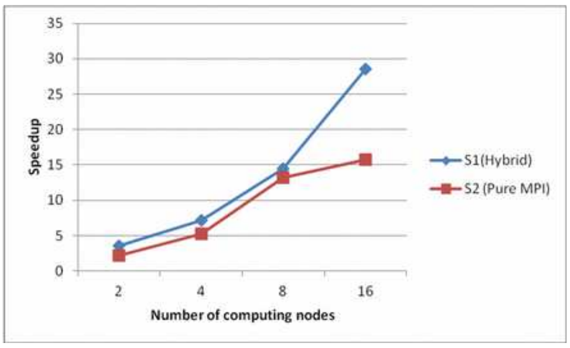


Figure 6. The speedup of the pure MPI and the hybrid MPI+OpenMP implementation related to sequential *PopularPlaces* application

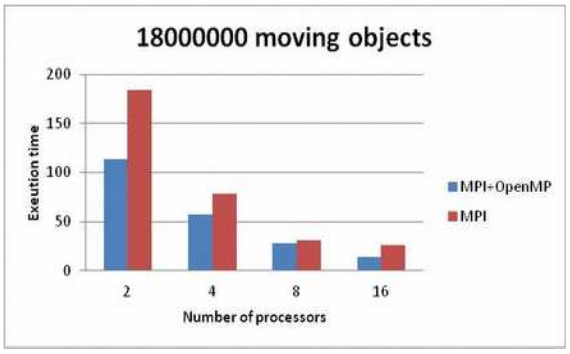


Figure 7. The execution time comparison between the MPI and the MPI+OpenMP implementation

Hybrid MPI+OpenMP implementation gives better results than pure MPI implementation for both data sets: maximum 1.6 for 1.8 million moving objects and maximum 1.8 for 18 millions moving objects. In the case of pure MPI implementation, the communication overhead outweighs the computation advantage, when number of processors is 16.

The application running over the cloud architecture shows the same tendency as the cluster architecture (Table 3). The obtained values of execution times are higher than the appropriate values for a cluster of physical machines, as expected. MPI+OpenMP implementation gives better results than pure MPI implementation on the cloud for both data sets: maximum 1.9 for 18 millions of moving objects.

Table 3. The experimental results for 18 million of moving objects on a cloud

18 million of moving objects on CLOUD			
p	4	8	16
T_{SEQ}	1568.706	1568.706	1568.706
$T_{MPI+OMP}$	394.985	204.064	103.322
T_{MPI}	413.072	216.179	196.573
$T_{MPI}/T_{MPI+OpenMP}$	1.045	1.059	1.902
S1	3.972	7.687	15.183
S2	3.798	7.256	7.980

As stated in the related work section, the superiority of one model, Pure MPI or Hybrid MPI+OpenMP, depends on the level (overhead caused by) of inter-process communication and shared memory parallelization. The problem we implemented of finding the popular places in the city represents the classical GIS analytics problem. The solution consists of two steps; the first step includes spatial join between two spatial or spatio-temporal data sets, and the second step performs appropriate processing and aggregation of join results. Our approach consists in assigning to each MPI process an equal set of moving objects data and indexing the POI data set using a spatial index structure, quad tree, to cope with spatial data skewness. By dividing the datasets for each MPI process we avoid the inter-process communication overhead and reduce the effect of MPI communication to the parallel implementation. We have used fine-grain OpenMP parallelization technique, by implementing OpenMP threads forked by each MPI process. In that way we increase the utilization of cores with sharing memory resources.

Our evaluation shows that the best overall performance is achieved when one MPI process is started on each node, and all other cores within the node are filled with OpenMP threads. In the pure MPI implementation, with the increasing number of nodes, the overall execution time decreases. Also, the execution time slightly decreases, when the number of nodes changes from 8 to 16, since there is the same number of processes per core. The hybrid implementation performs better then the pure MPI, especially when the number of nodes in the experiment changes from 8 to 16, since each OpenMP thread executes at separate core.

It is expected that other similar GIS algorithms that include processing of large-scale geospatial data and spatial join between two large spatial data sets will demonstrate the same performance behaviour for hybrid MPI+OpenMP approach.

5. Conclusion

With advances in remote sensing, geo-sensor networks and pervasive positioning, the amount of geospatial data that needs to be processed and analyzed has exploded in recent years. It leads to the rising of interest in processing and analysis of massive geospatial data, particularly trajectories of moving objects. This paper shows that the use of MPI and OpenMP programming models and framework can significantly improve the performance of data intensive GIS algorithms, such as the spatial join between large-scale trajectory data set and spatial regions representing places of interest (POI), as well as aggregation of join results in order to analyze popularity of POIs in the city. We show that employing distributed architecture consisting of a cluster of multicore PC nodes and integration of appropriate parallel programming techniques represents a promising research and development direction in applying high-performance computing in GIS applications. The hybrid implementation using both MPI and OpenMP frameworks, that employs parallelism between distributed computing nodes, as well as within multi-core computer nodes, eliminate communication overhead of the pure MPI solutions and achieve better overall performance by employing parallelism at the thread level. Our implementation provides excellent scalability regarding increasing trajectory data set and computing resources. Future work will consider other HPC techniques and platforms, such as MapReduce framework and its Hadoop implementation, in processing of Big geospatial and trajectory data.

REFERENCES

1. PATEL, S., W HWU, WEN-MEI, **Accelerator Architectures**, IEEE Micro, vol. 28, no. 4, 2008, pp. 4-12.

2. PACHECO, P., **An Introduction to Parallel Programming**, Morgan Kaufman, 2011.

3. SHEKHAR, S., **High Performance Computing with Spatial Datasets**, in Proceedings of the ACM SIGSPATIAL International Workshop on High Performance and Distributed Geographic Information Systems - HPDGIS, 2010, pp. 1-2.

4. MAYER-SCHÖNBERGER, V., K. CUKIER, K., **Big Data: A Revolution That Will Transform How We Live, Work, and Think**, Eamon Dolan/ Houghton Mifflin Harcourt, 2013, p. 256.

5. CLEMATIS, A., MINETER, M., MARCIANO, R., **High performance computing with geographical data** , Parallel Computing, vol. 29, no. 10, 2003, pp. 1275–1279, Oct. 2003.

6. AJI, A., WANG, F., VO, H., LEE, R., LIU, Q., ZHANG, X., SALTZ, J., **Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce**, Proceedings VLDB Endowment, vol. 6, no. 11, Aug. 2013.

7. ZHANG, J., **Towards Personal High-Performance Geospatial Computing (HPC-G)**, in Proceedings of the ACM SIGSPATIAL International Workshop on High Performance and Distributed

Geographic Information Systems - HPDGIS, 2010, pp. 3-10.

8. AKHTER, S., K. AIDA, Y. CHEMIN, **GRASS GIS on High Performance Computing with MPI, OpenMP and Ninf-G Programming Framework**, Proceeding of ISPRS, 2010, Japan.

9. SHI, X., **High Performance Computing: Fundamental Research Challenges in Service Oriented GIS**, ACM SIGSPATIAL- HPDGIS 2010 workshop: International Workshop on High Performance and Distributed Geographic Information Systems, San Jose, California, 2010, pp. 31-34.

10. WANG, Y., S. WANG, D. ZHOU, **Retrieving and Indexing Spatial Data in the Cloud Computing Environment**, Cloud Computing, First International Conference-CloudCom Beijing, China, 2009, pp. 322-331.

11. MA, Q., B. YANG, W. QIAN, A. ZHOU, **Query Processing of Massive Trajectory Data based on MapReduce**, in Proceeding of the First International Workshop on Cloud Data Management - CloudDB , 2009, pp. 9-16.

12. YANG, B., Q. MA, W. QIAN, A. ZHOU, **Truster: Trajectory Data Processing on Clusters**, in Proceedings of 14th International Conference Database Systems for Advanced Applications DASFAA, 2009, pp. 768-771.

13. ELDAWY, A., M. F. MOKBEL, **A Demonstration of SpatialHadoop: An Efficient MapReduce Framework for Spatial Data**, Proc. VLDB Endow., vol. 6, no. 12, 2013, pp. 1230–1233 .

14. KUNASETH, M.,. RICHARDS, D., GLOSLI, J., KALIA, R., NAKANO, A., VASHISHTA, P., **Analysis of scalable data-privatizati,on threading algorithms for hybrid MPI/OpenMP parallelization of molecular dynamics**, Journal of Supercomputing, vol. 66, 2013, pp. 406-430.

15. MINNINI, P., D. ROSENBERG, R. REDDY, A. POUQUET, **A Hybrid MPI–OpenMP Scheme for Scalable Parallel Pseudospectral Computations for Fluid Turbulence**, Parallel Computing, vol. 37, no. 6, 2011, pp. 316-326.

16. MASCETTI, S., D. FRENI, C., BETTINI, X. S. WANG, S. JAJODIA, **On the Impact of User Movement Simulations in the Evaluation of LBS Privacy- Preserving Techniques**, in Proceedings of the 1st International Workshop on Privacy in Location-Based Applications, 2008, vol. 397, 2008.