Research Summary

Sandeep Gupta

I am interested in data-intensive, graph, semantic, and informatics computing using modern hardware and architectural features such as large array of multi-cores, high-bandwidth networks (infiniband), various atomics, SSDs, scratchpad memory, dynamic voltage frequency scaling etc. I did my Bachelors in Computer Science from the Indian Institute of Technology (IIT), Guwahati. Following that I completed my Ph.D. in databases under Professor Chinya Ravishankar from University of California, Riverside (UCR) in Sept 2006. After that I joined San Diego SuperComputer Center where I am now an assistant research scientist working on architecture, data intensive and high performance computing, and, informatics data management.

Over the course of my PhD and as a research scientist I have learned and worked across various disciplines of computing: access methods, algorithmics, query-engines and runtimes, parallel and distributed computing and architecture. In most cases my work has focused around accelerating graph or informatics processing. In my research I consider the following as my chief results:

- Developed an algorithm to scale reachability queries over acyclic graphs on massive multi-core systems. Compared to sequential algorithm, my approach, using 256 cores on UV Blacklight supercomputer, is more than 2 orders of magnitude faster. Reachability is an important problem due to prominence of semantic web and ontology centric life-sciences data. Its access pattern lacks locality (spatial and temporal) and is difficult to optimize on current cache-based architectures. It is also harder to parallelize than BFS exploration because of low degree of parallelism and higher requirements for synchronization. The algorithm that I propose is successfully able to overcome these barriers.

- Proposed an acyclic graph based data model, query language, and distributed evaluation strategy for heterogeneous life-sciences data. My scheme can handle complex navigational queries over acyclic structure that would not be feasible in the traditionally XML setting.

- Developed a hybrid pthreads/MPI based algorithm for graph exploration. The algorithm scales up to 2048 cores spread across 128 compute nodes and can perform exploration on a 8 billion node and 256 billion edge scale free graph in less then 3 minutes yielding roughly a TEPS[1] of 2 billion. By graph500 benchmark this algorithms ranks at number 20 (on Trestles machine).

- Developed concurrent data structures to build and maintain streaming social network graphs such as Twitter. Developed a parallel algorithm using this structure to track popular/highly ranked nodes in such settings.

- As part of my Ph.D. dissertation, developed data-structures for solving spatio-temporal and shortest distance queries on road networks using recursive graph separators. These data structures perform well on real world datasets of Los Angeles and U.S. Highway road networks and have provable theoretical bounds. Specifically, the NDL data structure [19] requires $O(n\sqrt[4]{n}\log n)$ space to compute shortest distance in $O(\sqrt[4]{n}\log^2 n)$ time and $O(\sqrt[4]{n})$ disk I/O. Further, they form a very effective route/trajectory hashing method for the domain, allowing significant optimization of novel spatio-temporal queries such as: An object o_p (the "pursuer") starts at a specified node, and is required to catch a second object o_q (the "quarry"), whose route $\langle o_q \rangle$ is given.

In addition, I am currently working on following problems:

- distributed joins that utilize infiniband hardware and multicore to scale processing beyond 1K cores.
- a pruning based method for accelerating reachability queries.
- architecture for energy efficient graph exploration.

In the rest of this write-up, I present a brief description of each of these works along with my results.

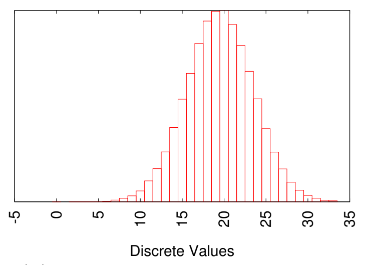
[1]a metric devised by the graph500 committee to measure the performance of graph exploration

1 Parallel Acyclic Graph Processing Using Multi-Cores and Applications

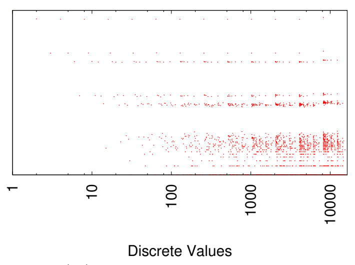
Acyclic graphs, much like scale-free graphs, appear in many areas of computation and engineering. These include, but are not limited to, knowledge representation, binary decision diagrams, dependency graphs, semantic web, and, binaries of computer programs. In the field of life-sciences and bio-informatics, in particular, such structures are used to create ontologies that represent the compendium of factual information and are central towards organizing and cataloging scientific data. Given their abundance and their increasing prominence in data management efficient algorithms that can process large acyclic graphs are highly desirable. I explored the problem of acyclic graph centric data engine for modern supercomputers. The research in this direction has lead to following three developments: (i) an acyclic graph generator (ii)a scheme for answering reachability queries over acyclic graphs, and (iii) a data model and query language based on acyclic graphs.

1.1 Scalable Acyclic Graph Generator [8]

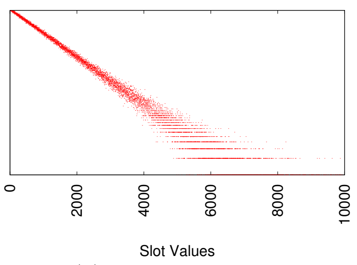
In this work, I show why most naive generators might produce degenerate (bipartite graphs) and propose a generator that can produce a stochastic acyclic graph satisfying constraints on the in-degree, the out-degree, the depth, and the path-length[2] attributes of the resulting dag. I provide suit of distribution functions that yield non-degenerate real-world acyclic graphs. Figure 1(a)–1(c) show the input (or desired) distribution functions and figures 1(d)– 1(j) show the frequency distribution of the various attributes of the resulting dag.



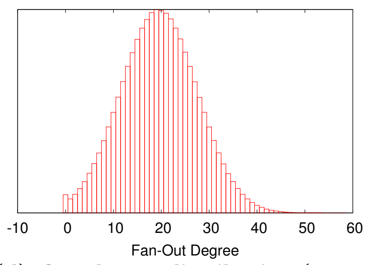
(a) ***Out-,In-degree distribution***



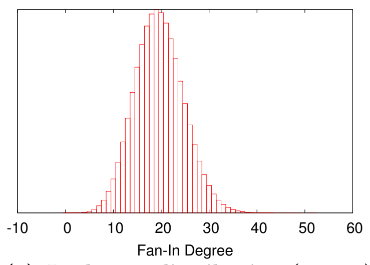
(b) ***Depth distribution***



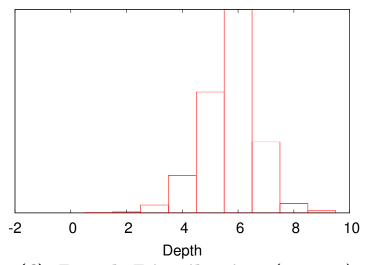
(c) ***Slot distribution***



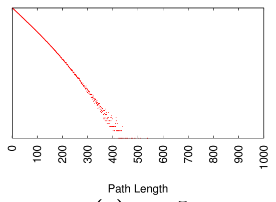
(d) ***Out-degree distribution (m=20)***



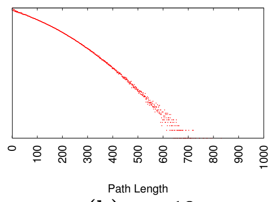
(e) ***In-degree distribution (m=20)***



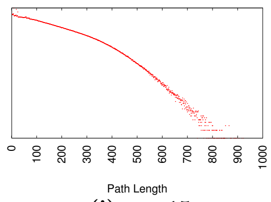
(f) ***Depth Distribution (m=20)***



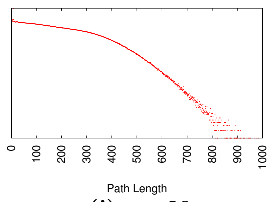
(g) ***m=5***



(h) ***m=10***



(i) ***m=15***



(j) ***m=20***

Figure 1: Input distributions (a)–(c) specify required degree and depth characteristics in the dag. Output (d)–(f) show the distribution, which satisfies the input requirement, of various attributes of the dag obtained from our generator. Distributions (g)–(j) illustrate variation in depth path distribution when varying the mean of the degree distribution.

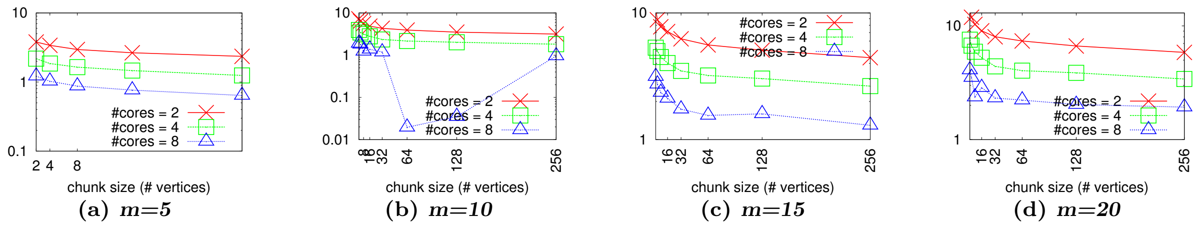
[2]distribution of random walks from source to sink

This method is linear with respect to number of edges and can produce acyclic graphs up to 64 million nodes and 320M edges. Previous method of generating acyclic graphs [22] require solving a linear program where each node represents a variable in the formulation and therefore is not scalable to very large graphs. Scalable graph generators that can model and produce domain specific acyclic graphs can help in algorithm design and database engineering.

1.2 Answering Reachability Queries using massive multi-cores [5]

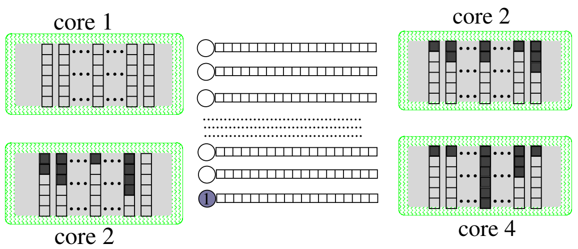
Reachability query is a fundamental operation while processing acyclic structured data. The operation is, however, difficult to accelerate using indexing and caching strategies due to low locality of access (both spatial and temporal). They are also difficult to parallelize due to low compute to communication ratio and requirement for fine-grained synchronization. This combined with high cost of synchronization on massive multi-cores make scaling reachability queries even more challenging.

In this work, I developed a parallel algorithm that overcome these barriers and scale acyclic graph processing to 100's of cores. On large acyclic graphs, my scheme is more than two order of magnitude faster than the sequential counterpart. In order to alleviate the high synchronization cost the algorithm partitions the node in chunks. Each core maintains a local queue of discovered nodes per chunk. Once the local queue is filled it locks the chunk and marks all discovered nodes as illustrated in the figures 3(a) and 3(b). The strategy for synchronization and graph exploration is such that it avoids busy-waiting and contention for resources. This allows wait-free progress and scalability up to 100's of cores.

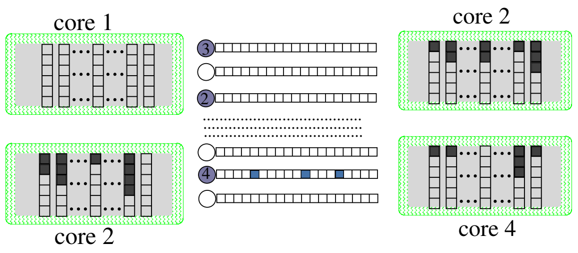


(a) ***m=5*** **(b)** ***m=10*** **(c)** ***m=15*** **(d)** ***m=20***

Figure 2: Figure2(a)–2(d) shows query performance for increasing graph sizes (64M nodes and 160M–640M edges). In each plot I vary the chunk size and number of cores. It show that performance improves by increasing the both the chunk size and number of threads. Our algorithm is able to find the right trade-off between extreme points of no synchronization (i.e. each thread performs independent exploration) and fully synchronized (i.e. synchronous update of the discovered vertices state).

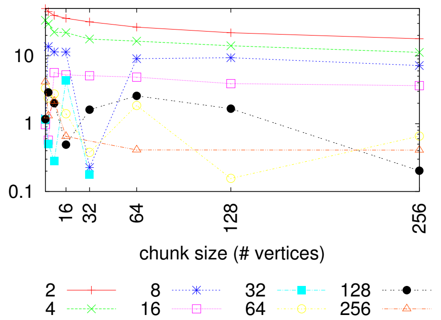


(a) ***Before chunked visited***

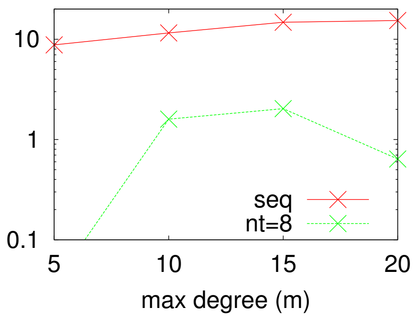


(b) ***Post chunked visited***

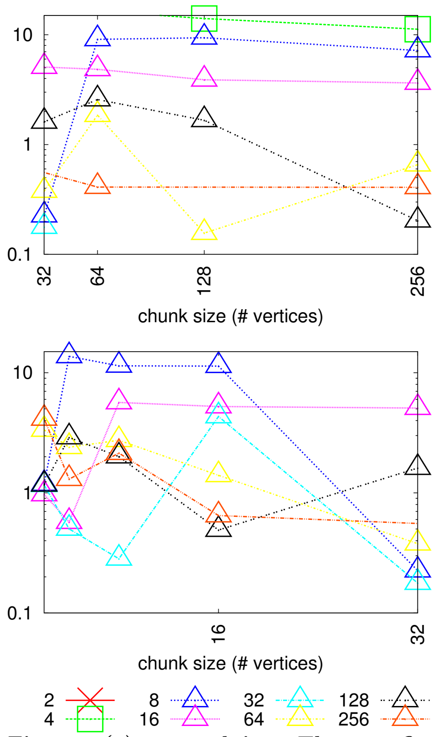
Figure 3: Shows the working of the algorithm. Each core maintains a collection of queues, one for each chunk. The cores explore the graph independently and place the discovered nodes in the designated queue. Once the queue is full the core locks the chunk and marks all the unvisited nodes as visited (we refer this as asynchronous update). In this figure this is depicted for core 4 whose queue is full. It locks the chunk and marks all the nodes in the queue as visited and finally unlocking it. The queue_sz and the chunk_sz control the trade-off between synchronization and computation. Small queue sizes and large chunk size imply high degree of synchronization and while larges values implies that cores will perform more exploration than communication.



(a) ***Preliminary study using UV blacklight supercomputer hosted at Pittsburg Supercomputer Center (PSC). Each line on the plot corresponds to number of cores used for exploration. We see that our algorithm scales up to 256 cores. It is able to overcome the communication and synchronization cost associated with massive core supercomputers. Compared to sequential algorithms our method performs three orders of magnitude faster.***



(b) ***Comparison with sequential algorithm with varying dag sizes using Triton supercomputer host at San Diego Supercomputer Center (SDSC). The parallel algorithm using 8 cores is couple of orders of magnitude faster. Furthermore, unlike sequential algorithm, the parallel algorithm is able to scale with increasing dag density.***



(c) ***Figure 4(a) zoomed in. The top figure shows the data points for chunk sizes of 32–256 while bottom row shows the portion from 0–32. The chunk size parameter controls the communication vs. synchronization trade-off of the algorithms. As the number of cores is increased the synchronization cost goes up and therefore larger chunk sizes are needed to amortize this increased synchronization cost and maintain performance. For smaller chunk sizes the performance is roughly the same irrespective of the number of the cores used.***

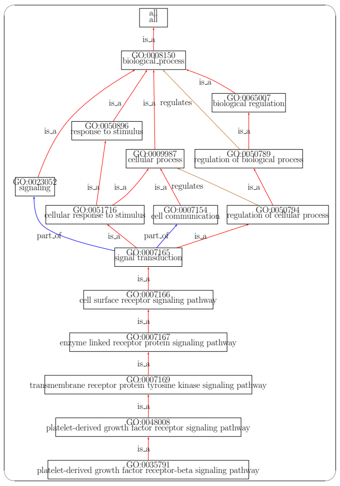
Figure 4: Scaling reachability queries to 100's of cores.

1.3 Data Model and query languages for scalable processing of heterogeneous biological datasets [10, 4]

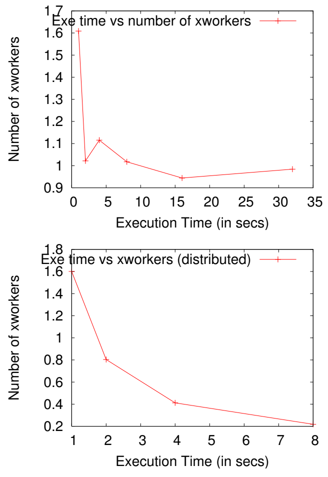
Current tools and techniques fail to adequately model the rich and heterogeneous data types like those encountered in life sciences. Moreover, the querying using standard database and RDF tools is lacking in capabilities and restricts users from formulating and answering complex domain specific queries. They also do not capture (in straight forward manner) the full spectrum of data types and queries prevalent in domain science. Finally, they lack mechanisms for translation of those queries to efficient execution plan. For example, consider the ontology data regarding biological processes. Such kinds of data is organized as acyclic graphs can be very large. A biologically relevant query against this example dataset is *"Return the type of regulation signaling of biological processes that have been linked to Alzheimer's study"*.

Solving such queries requires the algorithm traverses the acyclic graph along the descendants of the "biological processes" till it reaches the leaf nodes. If the leaf node has annotations in the publications about Alzheimer's it then navigate towards the root node until it reaches the signaling node. Such forward and backward navigation along aylic graphs is not feasible within the XML framework. Moreover it can be shown that such queries are computationally expensive.

My work in this direction show how to model most, if not all, biological data using acyclic graphs. I propose a data-model and an associated query language (based on XPath). My proposed data model and query language duo allows the user to express complex biological entities and query expressions. I have also developed a hybrid parallel/distributed evaluation strategy to evaluate queries on modern SSD enabled supercomputers. Figure 5 shows scalability of query evaluation in shared memory and distributed computing platform.


all
GO:0008150 biological_process
GO:0065007 biological regulation
GO:0050896 response to stimulus
GO:0009987 cellular process
GO:0050789 regulation of biological process
GO:0023052 signaling
GO:0051716 cellular response to stimulus
GO:0007154 cell communication
GO:0050794 regulation of cellular process
GO:0007165 signal transduction
GO:0007166 cell surface receptor signaling pathway
GO:0007167 enzyme linked receptor protein signaling pathway
GO:0007169 transmembrane receptor protein tyrosine kinase signaling pathway
GO:0048008 platelet-derived growth factor receptor signaling pathway
GO:0035791 platelet-derived growth factor receptor-beta signaling pathway
is_a
regulates
part_of

(a) ***Ontology graphs are common in life-sciences and have acyclic graph structure.***



(b) ***Parallel evaluation of XDPath query on multi-cores and on distributed (MPI) based system.***

Figure 5: Data intensive computing using ontologies. (a) The data model and query language allow complex expression over acyclic graphs and thereby present a management technique for heterogeneous and cross-referenced biological data. (b) Unlike XML queries,XDPath queries are computationally expensive. However, unlike queries on RDF data,their evaluation can be scaled using parallel processing. Figure shows scalability on a million node graph for SMP and distributed systems. We see that the query has almost linear scalability on the distributed platform. This work provides evidence that supercomputing capabilities are essential towards dealing with challenges in biological data management.

2 Algorithms for massive scale social network graphs

Growth in graph structured data in last decade (due to various social networks and Internet) has out-paced the growth in computing capabilities. At SDSC, I had the opportunity to study massive scale graph processing using large number of cores both in shared and distributed memory setting . In particular, my contributions were: (i) massively parallel reachability answering (discussed previously), (ii) a mechanism for tracking popular/higly ranked nodes in Twitter network, (iii) a distributed graph exploration algoritm and (iv) a graph exploration mechanism for scratchpad/embedded systems memory.

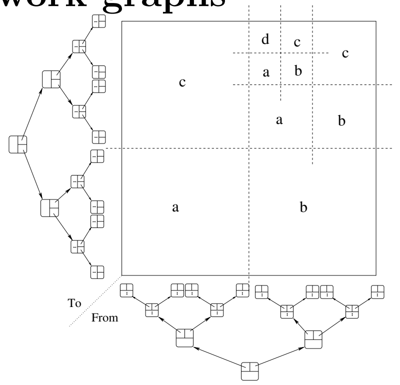
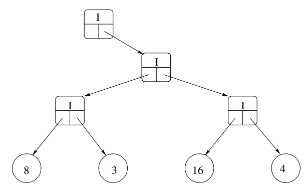


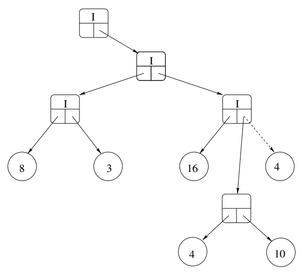
Figure 6: Using B_{dh}-tree for recursive R-MAT graph generation. Joint probability walk along the source (to) and destination (from) B_{dh}-tree

2.1 B_{dh}-tree and applications [6]

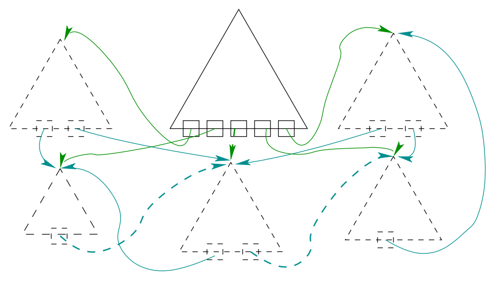
In this work, I propose a new data structure, called B_{dh}-tree, and associated algorithms and show its usage for streaming social networks such as Twitter. The data structures in such domains are massive in sizes and also need to be updated while being queried. Hence, concurrent yet scalable access and manipulation of the data-structures is an essential requirement.



(a) ***The B_{dh}-tree data-structure.***



(b) ***Insertion in the set.***



(c) ***Dynamic graph data structure using $B_{dh}-$ trees.***

Figure 7: (a) As oppose to other search trees entries at only at leaf nodes. (b) Use of double-word (16 byte word) compare-and-swap(cmpchx16) atomic instruction allows for concurrent access. Both the node type and pointer to child is update in a single atomic instruction. (c) Each triangle represents a $B_{dh} - tree$. The tree in the center (with bold outlines) is the node-set tree while those on the perimeter are the adjacency-set tree. The node-set-tree has vertex id as keys and pointers to root of adjacency-tree as values. Each adjacency tree contains adjacent vertices as keys and their adjacency tree pointers as values.

This work shows how to address this problem using B_{dh}-trees and modern hardware instructions such as double-word-compare-and-swap (DCAS).

B_{dh}-trees are structurally equivalent to binary-trees (not necessarily balanced). However, unlike binary search trees, B_{dh}-trees are stored only at the leaf nodes. The location of the key is determined by a family of hash functions $\mathcal{H}_0, \mathcal{H}_1, \mathcal{H}_2, \ldots \mathcal{H}_d$ where $H_i : I \longrightarrow \{0, 1\}$. Given a key k and depth d, $\mathcal{H}_d$ determines if k is in left or right subtree. B_{dh}-trees trades-off storage space for simple insertion and deletion of key-value pairs.

I developed concurrent B_{dh}-trees using DCAS (cmpchx16b) available on modern x86 architectures. This allows multiple threads to insert, delete, and search in the same tree structure simultaneously without introducing any race conditions. Since atomic instructions allow for fine grained synchronization my approach avoids the need for locking and mutual exclusion is scalable with respect to number of threads.

Dynamic social network graph generator

R-MAT is a scale-free graph generator that is widely used to generate graphs that represent web-link graph, social networks, and other real world graphs. I show how to utilize the binary structure of the B_{dh}-tree to perform joint-probabilistic walks on the tree to produce R-MAT graphs over the dynamic graph structure. Recursive partitioning of the R-MAT algorithm is “simulated” by choosing the left- or the right- subtree based on outcome of random variables and R-MAT parameters a, b, c, and d. Figure 6 illustrates the walk till 3 levels of the tree. Figures in 8 illustrates the resulting degree distributions at different stages of node and edge insertions.

Random Walkers on streaming graph (Twitter) for continuous tracking of popular nodes

I demonstrate how to use B_{dh} structure to represent and maintain a graph structure with streaming updates such as in Twitter. Using the dynamic social network generator (described above) I generated a synthetic workload of 4 million edge insertions over a million node graph. The algorithm perform 1 million random walks concurrently (average path length of 15–20) using 2–32 cores. This allows for continuous tracking of popular/highly ranked users in the network network with high probability. The connection between random walkers and popular nodes in social networks has been explored previously [21].

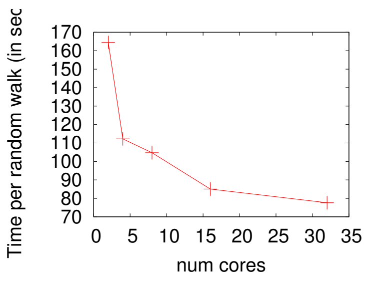


Figure 9: Concurrent access time vs. number of cores. Concurrent 4M random walks (avg. walk length=15) and 1M edge insertion using 2–32 cores.

I also investigated its performance with respect to increasing number of concurrent walkers when a single thread is performing 4M edge insertion in the background. The experiments

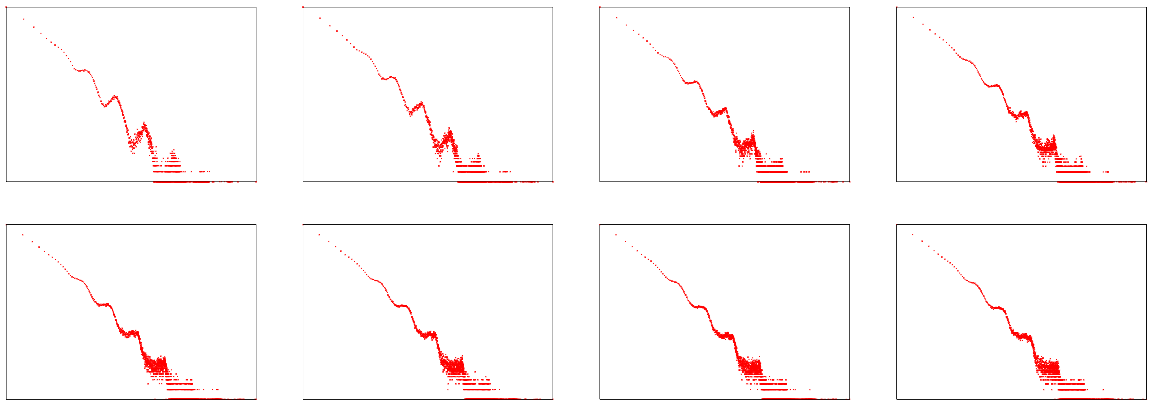


Figure 8: Evolution of degree distribution under addition and deletion of nodes and edges. Consequent plots differ in 200, 000 edge insertions. We see that the that our generator is able is produce shapes similar to the static R-MAT generator and is stable over the modification of the graph. Moreover, local biases/ kinks disappear as more nodes and edges are inserted.

show that B_{dh}-trees is scalable with increase in number of threads. On the other hand, a naive approach using sequential data structures and locks does not scale with increasing number of cores (see figure 9). Moreover, unlike binary-tree data-structure B_{dh} tree allows selection of a random node in $O(2 * \log n)$. In comparison, without additional storage other set data-structure require $O(n)$ time to pick a random element.

2.2 Hybrid pthread/MPI based processing of massive scale graph [7]

Graph500 is a benchmark for assessing data intensive capabilities of big machines and supercomputers. In this work I developed hybrid shared (pthreads/atomics) and distributed (MPI) algorithm for processing graphs of size up to scale 33, i.e., 8 billion nodes and 256 billion edges have storage cost of 8 Terabytes. My algorithm scales up to 2048 cores distributed across 128 compute nodes.

In order to scale to such large sizes, the algorithm makes extensive use of vast arrays of SSDs and asynchronous I/O so as to overlap computation and communications. The current performance the algorithm is roughly 2 billion TEPS (edges traversed per second) and ties with the number 20 ranked algorithm in the current graph500 list.

2.3 Massive scale graph exploration on scratchpad/embedded hardware [9]

It is well known that graph exploration on current architecture does not scale -well. The cache subsystem (L1-, L2-, and TLB) that were boon to relational data-engines turn out to be performance impediments for unstructured and graph-centric databases. We designed a parallel graph exploration strategy that does not rely on the caching subsystem for performance but carries out its own data-movement.

Such a strategy turns out to be also effective for cached based system and performs twice as fast than its counterpart that does not perform any queue management. Figure 10 illustrate the methodology.

3 Data Structures for Spatio-Temporal and Shortest Distance Queries on Road Networks

Efficient methods exist for spatial and spatio-temporal queries if distances between points are defined using the Euclidean metric. However, for motions along road networks, it is more appropriate and useful to consider *road network distances*, measured along roads. Unfortunately, this change makes shortest distance and path computation far more expensive, severely degrading performance of traditional methods.

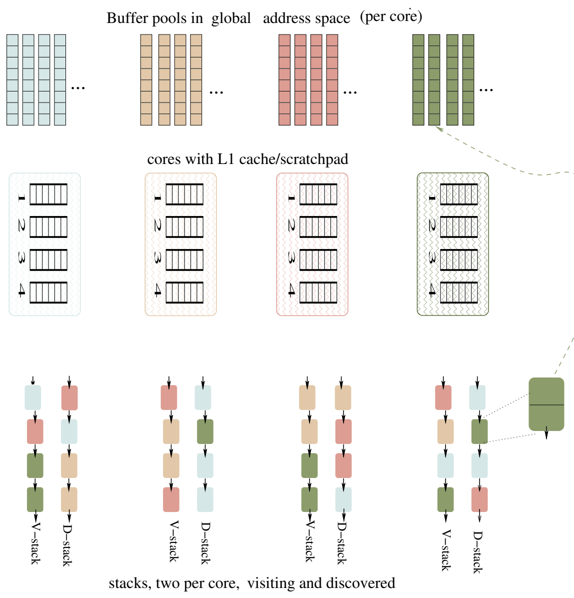
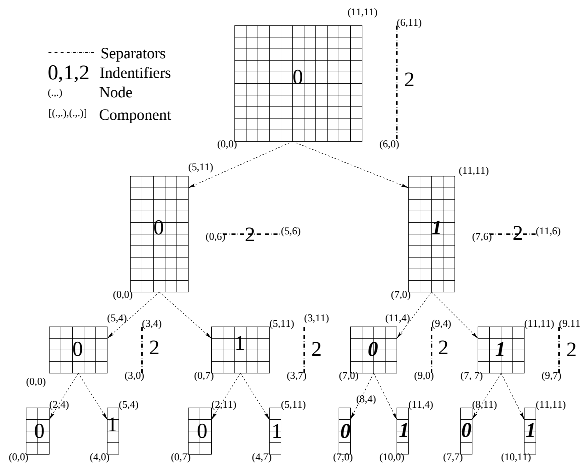

Buffer pools in global address space (per core)
cores with L1 cache/scratchpad
stacks, two per core, visiting and discovered
V-stack
D-stack

Figure 10: Illustration of the scratchpad algorithm using 4 cores. The three main data-structures are the buffer-pool (top row), the scratchpad-queue (middle row, shown as internal to the core) and the two stacks implemented as liked list. The V- and D- link stands for visiting and discovered. Each core performs exploration of the graphs and discovered nodes are placed in the designated queue that is internal to the core. Once the queue is full the core copies the data from internal memory to the global memory and makes an entry in the V-stack of the designated core. This is achieved by push a node on top of the queue using atomic instructions. Further improvement is feasible by reducing contention by performing asynchronous atomics. However most current architecture do not support this feature.

3.1 PaL: Partial labelings For Efficient Spatial and Spatiotemporal Queries on Road Networks [19, 18]

I developed PaL, a novel approach that uses graph separators to assign labels to a subset of graph nodes, such that shortest network distances between any two nodes can be found quickly from their labels. The labeling is based on hierarchical recursive decomposition of the road network using separators (see figure below).



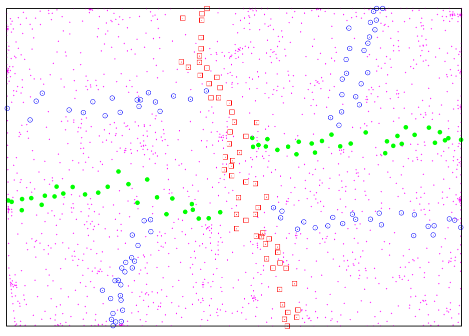


Figure 11: Hierarchical recursive decomposition (HRD) of grid graph till level 3. HRD of real planar graph (edges omitted for clarity) using separators. Empty squares are level-0 separators, dark circles are at level 1 and empty circles are at level 2. The decomposition of the graph is done without the knowledge of the planar embedding. Yet it is able to perform “good” separation as is clear when compared with grid graph decomposition, which is optimal. Each connected component is referred to as ”fragments”.

We show how to reduce storage requirements by selective labeling of nodes, without significant performance penalty. We also describe an index structure for static and mobile objects that allows pruning of the search space based on spatial and connectivity properties of the road network.

The proposed methods form a powerful and unified framework for spatial and spatio-temporal query processing on road networks. Their performance for spatial queries, even with distances measured along roads, is comparable to that of traditional methods using Euclidean distances. Our experiments on real road networks show that our shortest distance method is several orders of magnitude faster than schemes based on Dijkstra algorithm, that our index structures for trajectories perform much better than R-tree based indices, and that our optimizations speed up spatio-temporal queries significantly.

I proved the following results regarding labeling for shortest distance computations on planar graphs:

Lemma 3.1 *Let $G = (V, E)$ be a planar graph. For a graph admitting $O(\sqrt{n})$ recursive vertex separators, the DL data structure requires $O(n\sqrt{n}\log n)$ space, $O(\log n)$ time and $O(1)$ disk I/O, while the NDL requires $O(n\sqrt[4]{n}\log n)$ space, $O(\sqrt[4]{n}\log^2 n)$ time, and $O(\sqrt[4]{n})$ disk I/O. [19]*

	component ID	**distances to level-i separator nodes**
$L_{(3,0)} = \langle$	$(id^0_{(3,0)} = 0,$	$A^0_{(3,0)} = \langle 3, 4, 5, \dots 12\rangle),$
	$(id^1_{(3,0)} = 0,$	$A^1_{(3,0)} = \langle 9, \dots, 6, \dots, 9\rangle),$
	$(id^2_{(3,0)} = 2,$	$A^2_{(3,0)} = \langle 0, 1, 2, 3, 4\rangle)\rangle$

(a) ***Label for nodes*** $(3, 0)$ ***in the labeling associated with HRD of the grid-graph over the sample grid graph.***

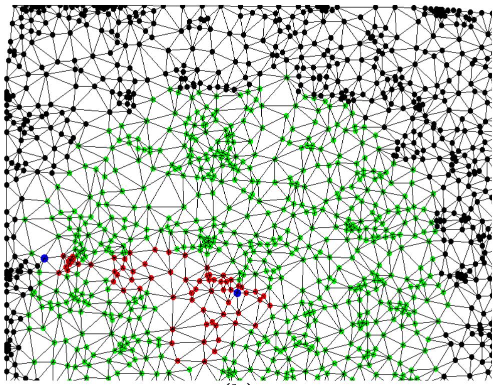

(b)

Figure 12: (a) Every separator node has a label and stores distances to nodes in $O(\log n)$ separators. (b) Decoding strategy using the labels narrows the region of the graph explored while finding the shortest distance. Region of the network explored by PaL algorithm (red dots) and those visited due to Dijkstra's algorithm (red and green dots) is shown. We also propose a path decoding algorithm that uses only the distance labels and the fragments within which the path lies.

3.2 Roads, codes, and spatiotemporal queries [12, 17]

This work provides the theoretical underpinning for our approach for solving spatial and spatio-temporal queries when distances are measured along road network. It is based on the work of [2] regarding hypercube embedding of graphs using isometric separators. We developed strategies to find good quality "isometric" separators and a associated coding mechanism that assigns a binary string to each vertex of the graph. The coding maps a vertex to a point in a high-dimensional hybercube. The isometric property guarantees that (section 5.6 in [12]) that the Hamming distance between codes of any two vertices would be equal to the shortest distance on the graph.

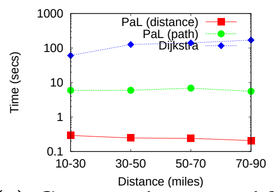
This codes are also used as very effective route/trajectory hashing method for the domain, allowing significant optimization of novel spatio-temporal queries such as:

- Given a time interval $[t_1, t_2]$, we are required to find all pairs of (o_i, o_j), such that o_i and o_j are on the same edge during $[t_i, t_2]$.
- Given a query vertex v_q and time interval $[t_1, t_2]$, an incidence query requests all objects that pass through v_q during $[t_1, t_2]$.
- An object o_p (the "pursuer") starts at a specified node, and is required to catch a second object o_q (the "quarry"), whose route $\langle o_q\rangle$ is given.

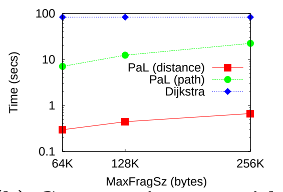
4 Other Works

4.1 On the Complexity of 2-Dimensional K-Nearest Neighbors Query in Parallel [11]

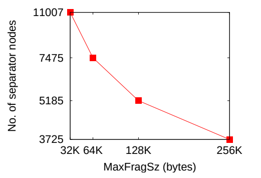
The problem of finding k nearest neighbors from a given set S of n points arise frequently in the GIS, biology, and, machine learning. We study the parallel version of this problem , also known as k-nn declustering i.e.



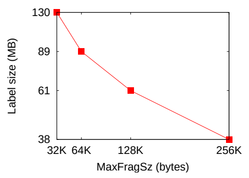
(a) ***Computation cost with varying distance. Computation cost for Dijkstra's algorithm increase with increase in distance.***



(b) ***Computation cost with varying fragment size.***

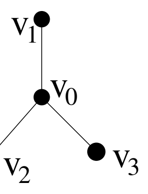


(c) ***Number of separator with varying fragment size***

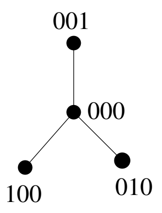


(d) ***Total labeling cost with respect to fragment size***

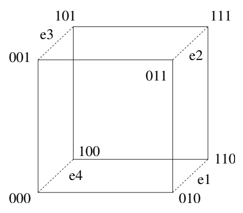
Figure 13: Storage and distance costs for PaL with respect to varying fragment size. Our algorithm is able to trade-off storage cost with respect for increased shortest distance and path computation. Shortest distance computation using PaL is more then 2 orders of magnitude faster then Dijkstra's algorithm. The path computation is an order of magnitude faster.



(a) ***A graph not embeddable in*** $\mathbb{R}^m$ ***for any*** m



(b) ***A star that can be embedded in*** H_3



(c) H_3 ***and its structure***

Figure 14: Motivation for Hypercube Embedding

find a decomposition of a given set $S \subset \mathbb{R}^2$ of n points so that the load per link per query l is minimized. In particular, if $nn_p^k(S)$ be the set of k-nn of p in S, then given constant c, it seeks to find c disjoint partitions of S such that no partition contains more than l elements from $nn_p^k(S)$.

We devise declustering strategy that achieves $O(\log^{\frac{3}{2}} k\sqrt{\log nk})$ load for $c = 2$. The best known bound for this problem was $O(\log^{\frac{3}{2}} n)$ due to Matousek et. al [15].

We introduce a novel relaxation of this problem called *partial declustering*, which declusters a subset $S' \subseteq S$, so that the load is less than a given parameter l_t, where $l_t \geq k/c$. We develop methods that yield small and bounded number of ignored points $(S \setminus S')$ for given load l_t. Our findings motivates its potential applicability over traditional formulation.

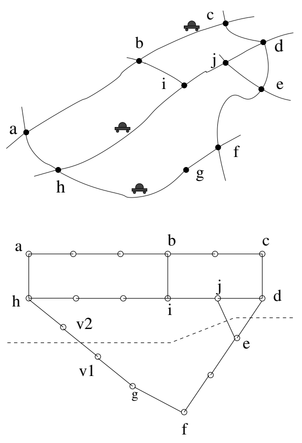
We also propose a simple and scalable partial declustering which creates *randomized* partitions, *refines* them locally, and *ignores* data points until load is below l_t. We conduct experiments to validate our findings, show the superiority of the partial declustering and demonstrate that parallel k-nn has weak scaling property with respect to load. To the best of our knowledge analysis declustering k-nn queries and its scaling properties has not been studied earlier.

4.2 Efficient data dissemination using locale covers [14, 13]

Location dependent data are central to many emerging applications, ranging from traffic information services to sensor networks. The standard pull- and push-based data dissemination models become unworkable since the data volumes and number of clients are high.

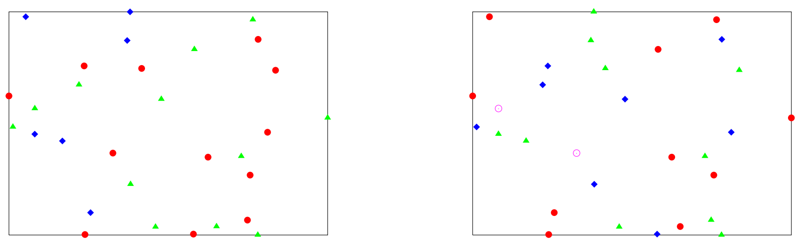
We address this problem using *locale covers*, a subset of the original set of locations of interest, chosen to include at least one location in a suitably defined neighborhood of any client. We also introduce a nested locale cover scheme that ensures fair access latencies, and allows clients to refine the accuracy of their information over time. We also prove two important results: One regarding the greedy algorithm for sensor covers and other pertaining to randomized locale covers for k-nearest neighbor queries [3].

[3]This paper is an extended version of [13]



Node	Code
a	01011111111011
b	00000001011011
c	00000000000111
d	00000000000100
e	10000000000000
f	11111000000000
g	11111110000000
h	01011111111000
i	00000001011000
j	00000000001000

Figure 15: A sample road network. Its planar graph representation and illustration of isometric separators. The embedding in H_{14} of the example road map



(a) ***Three partitions,*** $l = 4$

(b) ***Partial Declustering,*** $c = 3$, $l = 3$, $|U| = 2$

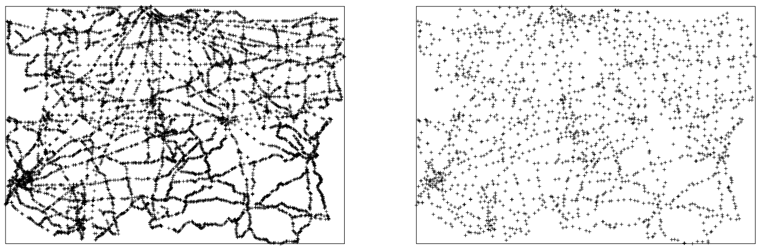
Figure 16: Traditional vs. Partial Declustering: Each color corresponds to a partition. Traditional clustering (figure 16(a)) has optimal load of 4. Partial declustering (figure 16(b)) can achieve load of 3 if two the points, shown as empty circle, are refrained from the declustering.

5 Current and non-peer reviewed manuscripts

- Currently developing a distributed sort merge join algorithm that utilizes asynchronous communication and SSDs to scale join processing to large number of cores.
- Using the SESC simulator to study the energy efficiecy of various scientific and graph codes.
- Developed theoretical bounds for arrangement of sensor that collaborate to determine the approximate location of mobile and static objects when beacons from an object are received by all sensors that are within its distance R [16].
- Develop a technique to identify spams in online message networks based upon random projections [3]
- Developed v-Trees to index and queries large collection of tracjectories in euclidean plane [15].
- Currently developing a pruning based method to answer reachability queries over very large acyclic graphs.
- Studied the performance of graph and other data-intensive applications for SSD-enabled supercomputers [20]
- Was part of the team that participated and won the 2009 Supercomputer Storage Challenge [1]

References

[1] Research team wins sc09 'storage challenge' award. `http://cacm.acm.org/news/52941-research-team-wins-sc09-storage-challenge-award/fulltext`, Nov 2009.



(a) *CH* **(b) *10-domain locale cover.***

Figure 17: Sensor locations within the Chicago's road network and its locale cover assuming k-nearest neighbor criteria. The locale cover size is 1/10 of the original yet guarantees at least one of the top-10 nearest neighbor for any client. Our approach reduces bandwidth consumption while maintaining quality guarantees as location-dependent values are highly correlated with location

[2] V. Chepoi and M. Deza. Clin d'oeil on l_1-embeddable planar graphs. *In Discrete Appl. Math. 80*, pages 3–19, 1997.

[3] S. Dixit, S. Gupta, and C. Ravishankar. Lohit: An online detection & control system for cellular sms spam. In M. H. Hamza, editor, *Proceedings of IASTED International Conference on Communication, Network and Information Security (CNIS)*, Phoenix, USA, November.

[4] S. Gupta. A unified data model and declarative query language for heterogenous life sciences data. Technical report, San Diego Supercomputer Center, 2011.

[5] S. Gupta. Informatics supercomputing:part 1: Parallel reachability queries over acyclic graphs using multi-cores. Technical Report TR-2011-3, San Diego Supercomputer Center, 2012.

[6] S. Gupta. Random walkers on streaming graph (twitter) for continuous tracking of popular nodes. In *International Conference on Parallel Processing (to be submitted)*, 2012.

[7] S. Gupta. A relational and graph processing toolking using hybrid mpi+pthread primitives (in prepration). 2012.

[8] S. Gupta. Scalable acyclic graph exporation. In *IEEE Transactions on Knowledge and Data Engineering (submitted)*, 2012.

[9] S. Gupta, A. Agarwal, K. Seger, A. Snavely, and J. Torrellas. Architectures for energy efficient graphs exploration (in prepration). 2012.

[10] S. Gupta, C. Condit, and A. Gupta. Graphitti: An annotation management system for heterogeneous objects. In *ICDE*, pages 1568–1571. IEEE, 2008.

[11] S. Gupta, D. Gunopulos, and C. V. Ravishankar. On the complexity of 2-dimensional k-nearest neighbors query in parallel. *CGTA: Computational Geometry: Theory and Applications (under review)*.

[12] S. Gupta, S. Kopparty, and C. Ravishankar. Roads, codes, and spatiotemporal queries. In *Proceedings of the twenty-third ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '04, pages 115–124, New York, NY, USA, 2004. ACM.

[13] S. Gupta, J. Ni, and C. V. Ravishankar. Efficient data dissemination using locale covers. In O. Herzog, H.-J. Schek, N. Fuhr, A. Chowdhury, and W. Teiken, editors, *CIKM*, pages 243–244. ACM, 2005.

[14] S. Gupta, J. Ni, and C. V. Ravishankar. Efficient data dissemination using locale covers. *Pervasive Mob. Comput.*, 4:254–275, April 2008.

[15] S. Gupta and C. Ravishankar. Using vtree indices for queries over objects with complex motions. In *Proceedings of the 20th International Conference on Data Engineering*, ICDE '04, pages 831–, Washington, DC, USA, 2004. IEEE Computer Society.

[16] S. Gupta and C. Ravishankar. Lower bounds for arrangement-based range-free localization in sensor networks. http://arxiv.org/abs/1203.1042, 2012.

[17] S. Gupta and C. V. Ravishankar. Spatio-temporal queries on road networks, coding based methods. In S. Shekhar and H. Xiong, editors, *Encyclopedia of GIS*, pages 1122–1125. Springer, 2008.

[18] S. Gupta and C. V. Ravishankar. Pal: Partial labelings for efficient spatial and spatiotemporal queries on road networks. 2012.

[19] S. K. Gupta. *External memory algorithms for shortest distance and spatio-temporal queries on road network*. PhD thesis, USA, 2006. Adviser-Ravishankar, Chinya.

[20] J. He, A. Jagatheesan, S. Gupta, J. Bennett, and A. Snavely. Dash: a recipe for a flash-based data intensive supercomputer. *SC Conference*, 0:1–11, 2010.

[21] J. D. Noh and H. Rieger. Random walks on complex networks. *Phys. Rev. Lett.*, 92:118701, Mar 2004.

[22] Y. Theoharis, G. Georgakopoulos, and V. Christophides. On the synthetic generation of semantic web schemas. In V. Christophides, M. Collard, and C. Gutierrez, editors, *SWDB-ODBIS*, volume 5005 of *Lecture Notes in Computer Science*, pages 98–116. Springer, 2007.