Research Statement

Massive computing power and applications running on this power, primarily confined to expensive supercomputers a decade ago, have now become mainstream through the availability of clusters with commodity computers and high-speed interconnects running big-data era applications. The challenges associated with programming such systems, for effectively utilizing the computing power, have led to the creation of intuitive abstractions and implementations targeting average users, domain experts, and savvy (parallel) programmers. There is often a trade-off between the ease of programming and performance when using these abstractions. My research focuses on developing tools to bridge the gap between ease of programming and performance of *irregular programs*—programs that involve one or more of irregular- data structures, control structures, and communication patterns—on distributed-memory systems.

For example, the use of trees as spatial acceleration structures (kdtree, octree etc.) is a common optimization in *n-body* problems, which in a naïve implementation, require comparing each of a set of items with every element in the data set. Such data-dependent computation manipulating complicated data structures such as trees and graphs causes unpredictability in control-flow and communication patterns. And this unpredictability in irregular programs is the chief obstacle in effectively optimizing them for parallelism, locality, and communication, which are crucial for achieving a scalable performance. As irregular programs typically involve data sets that overwhelm the memory or compute power of a single compute node, distributed-memory solutions become necessary. Summarizing my research gives a glimpse of the proposed solutions. I have introduced:

- A framework [3] consisting of an abstraction and a *space-adaptive* runtime system for simplifying the creation of distributed implementations of recursive irregular programs based on spatial acceleration structures. Through a set of optimizations, the system achieves a scalable performance, and outperforms implementations done in contemporary distributed graph processing frameworks.

- An *ontology* and a benchmark suite [4] for better understanding of optimizations and tree-algorithms in general. Using the ontology, we can generalize existing optimizations from one domain to another and find out optimizations applicable for new tree algorithms.

- A system [2] for automatically generating distributed-memory implementations of recursive divide-conquer algorithms. The system generates implementations that execute significantly faster than manual implementations done using similar frameworks and often outperform hand-written distributed implementations.

This research has explored topics in high-performance computing, programming languages, and distributed systems, and has touched upon a number of application domains such as data mining, numerical computing, statistics, and bioinformatics. As a result, it has opened up new directions that I would like to explore. In the short term, I would like to optimize important irregular algorithms from the domains of numerical computing, bioinformatics, and image retrieval for large-scale data processing. In the long term, given the increasingly heterogeneous computing capabilities of nodes in a cluster, I would like to focus on developing techniques for efficiently utilizing the heterogeneous computing power for executing irregular programs, and making these techniques accessible to average programmer. In this regard, I have open-sourced all my previous works so that expert programmers can improve the system and domain-specific application programmers can quickly deploy their algorithm.

Research approach

I have adopted the principle of generalization (of optimizations and methods) by abstracting common structural properties of irregular programs from various application domains. This approach has been beneficial for me, since, understanding structural properties of an algorithm and optimizations has demanded a deep-dive into the application domain, and has fostered my interest in newer areas. While SPIRIT [3] employed a generalization of a Barnes-Hut specific optimization, Treelogy [4] tried to identify the conditions under which a generalization of an optimization could be possible. With D2P [2], I believe I can generalize the inspector-executor based method, applied to recursive divide-conquer algorithms with certain properties, to a broader class of irregular applications.

Prior research

SPIRIT [3] provides a set of APIs and a runtime system to automate the creation and traversal of distributed spatial trees. I developed SPIRIT to address the insufficiency of traditional data-parallel approaches and existing systems in effectively parallelizing a subset of irregular applications based on spatial trees. As there is an abundant amount of parallelism in these applications due to repeated independent tree traversals, data-parallelism seems natural. However, a data-parallel approach to parallelizing traversals by executing them on multiple tree replicas is ruled out for large data sets because there is not enough space to store the tree in memory. So the tree must be distributed across compute nodes. In SPIRIT, I show that traversals on distributed trees can be executed efficiently through the use of optimizations targeting parallelism, communication overhead, locality, and load-balance. The SPIRIT system provides *space-adaptivity* by partially replicating the tree when space permits. This feature allows for seamless performance transition from a data-parallel approach to purely distributed solution. Also, I collaborated on developing an inspector-executor guided scheduling optimization for executing spatial tree traversals on GPUs [1].

Whether a new optimization or a tree algorithm, it is challenging to device an efficient implementation strategy considering a rich history of application- and platform- specific optimizations and tree algorithms. Treelogy [4] was conceived with the idea of understanding the connection between optimizations and tree-algorithms. Treelogy provides an ontology and a benchmark suite of a broader class of tree algorithms to help answer: (i) is there any existing optimization that is applicable or effective for a new tree algorithm? (ii) can a new optimization developed for a particular tree algorithm be applied to existing tree algorithms from other domains? I show that a categorization (ontology) based on structural properties of tree-algorithms is useful for both developers of new optimizations and new tree algorithm creators. With the help of a suite of tree traversal kernels spanning the ontology, my collaborators and I show that GPU, shared-, and distributed-memory implementations are scalable and the two-point correlation algorithm with vptree performs better than the 'standard' kdtree implementation.

Given the complexities of distributed-memory programming, is it possible to completely automate the creation of distributed-memory implementations of irregular programs? Here, a broader goal is to let a computer generate a distributed-memory implementation of *any* irregular program starting from its shared-memory artefact (specification, pseudocode, implementation). I take the first step towards this goal through specifications for recursive divide-conquer algorithms having certain properties: i) inclusive—a recursive method's parameters summarize the data access done within the method body. ii) Intersection—data-set intersection tests among method invocations can be computed efficiently when a hierarchical decomposition creates disjoint partitions of data, which are computed in a specific order (preorder) as determined by the control-flow of the program. Recursive formulations of Dynamic Programming (DP) algorithms are well-known examples having these properties. As recursive divide-conquer algorithms with inclusive and intersection properties have irregularity only in their communication patterns, they are a couple of degrees less complex compared to irregular applications with all three degrees of irregularity—data-structures, control-flow, and communication. As a result, they provide a convenient starting point for automatic distributed-memory code generation. D2P [2] is a framework for generating distributed-memory implementations of recursive divide-conquer algorithms starting from their *specifications*. In D2P, I show that it is possible to achieve a scalable performance, and even outperform hand-written distributed-memory implementations with the help of intuitive user-configurable knobs to control parallelism.

Future directions

Short term based on the frameworks and optimizations that I have built and used, I believe existing solutions for some of the problems in image retrieval, numerical computing, and bioinformatics can be improved using efficient data partitioning strategies, data-structures, and formulations targeting locality optimizations. The specific goal would be to create abstractions and new algorithms for processing data at scale: (1) building a scalable system for finding similar images from a repository using efficient spatial acceleration structures such as distributed vptrees, (2) creating libraries of efficient distributed implementations leveraging recent advances in Eigen-value solvers, and (3) creating efficient implementations of problems in bioinformatics based on their recursive formulations.

Long term I would like to develop techniques for simplifying the creation of performance-oriented programs of irregular applications for large-scale data processing. Considering the heterogeneity of cores—latency tolerant, NUMA-aware, GPUs—in nodes of state-of-the-art and future clusters, effective utilization of computing power asks for careful designing of algorithms and orchestrating the computation. As a result we expect to see increased programming complexity, thus calling for automation. To realize the automation goal, I believe there are multiple concrete directions such as a programming-by-examples/program-synthesis from specifications approach, using intuitive design of programming languages / domain-specific languages, source to source translation aided by efficient abstractions and runtime systems. However, in spirit, I believe the path towards realizing this goal requires understanding of the computation patterns in irregular applications, employing optimizations that are effective for those patterns, and mapping those computation patterns to hardware. In this regard, experience from the past is reassuring: SPIRIT and Treelogy explored algorithms based on a common pattern of repeated tree traversals, and D2P explored algorithms with the inclusivity and intersection properties.

References

[1] Jianqiao Liu, **Nikhil Hegde**, and Milind Kulkarni. Hybrid CPU-GPU Scheduling and Execution of Tree Traversals. In *Proceedings of the 2016 International Conference on Supercomputing*, ICS '16, pages 2:1–2:12, New York, NY, USA, 2016. ACM.

[2] **Nikhil Hegde**, Qifan Chang, and Milind Kulkarni. D2P: An Inspector-Executor Based Approach to Automatically Creating Distributed Dynamic Programming Codes (in preparation).

[3] **Nikhil Hegde**, Jianqiao Liu, and Milind Kulkarni. SPIRIT: A Framework for Creating Distributed Recursive Tree Applications. In *Proceedings of the International Conference on Supercomputing*, ICS '17. ACM, 2017.

[4] **Nikhil Hegde**, Jianqiao Liu, Kirshanthan Sundararajah, and Milind Kulkarni. Treelogy: A benchmark suite for tree traversals. In *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 227–238, April 2017.