Research Statement

Yan Gu

1 Introduction

We are fast approaching a data-centric and data-driven world. Due to the rapid growth of the data size, the efficiency for processing large-scale data has become more crucial. As a result, new hardware technologies (e.g., multicore/many-core processors, caching, new storage hardware) and new efficient algorithms are designed for this purpose.

As mathematical problems, algorithms have been studied for about 100 years, even prior to the emergence of computers. Numerous classic algorithms that are simple and elegant have been invented. However, to fully utilize the advantages of the new hardware technologies to handle large-scale data, practically efficient algorithms must be designed to address the architectural concerns brought up by new hardware.

The goal of my research is to bridge the gap between theoretical algorithm research and the evolving of software and hardware. My research considers *architectural aspects*, and I develop *theoretical models* to capture the new features or properties in the new architectures. Such models allow theoretical analysis and investigate the bottlenecks and hardness of a problem algorithmically. From the other direction, I design simple and theoretically-efficient *algorithms* on these models, most of which are also implemented and shown to be practical. Since the improvements are from the algorithmic side with provable guarantees, the algorithms that I have designed show robust performance on a wide range of platforms (e.g., CPU models, number of processors and sockets, hyperthreading or not) on various data distributions. My new ongoing work at MIT will also consider software-level support including *programming techniques, languages and system-level optimizations*.

My research has focused on designing efficient algorithms for modern architectures with an emphasis on 1) *multi-core parallelism* and on 2) future platforms based on *non-volatile memories*, with considerations of *cache performance* for both cases. I will first summarize my research results, and show more in-depth details in the two following sections. Finally, I will conclude with my research vision.

Non-Volatile Memories (NVMs). New NVM technologies are projected to become the dominant type of main memory in the very near future, that promise persistence, significantly lower energy costs and higher density than DRAMs. However, one key property of NVMs is the asymmetric read and write cost—write operations are much more expensive than reads with respect to energy, bandwidth, and latency. This property contradicts previous classic algorithm research and poses the need to develop write-efficient algorithms that use fewer writes than the classic approaches.

My PhD thesis [1] provides a comprehensive study of the design and analysis of write-efficient algorithms, building the foundation of a systematic study of write-read asymmetry. The work starts with modeling the computation on various settings (e.g., sequential, parallel, I/O algorithms) with the consideration of write-read asymmetry [2, 5, 6]. Then I show a number of lower bounds on these models [6, 15], and new write-efficient algorithms for fundamental algorithms, from basic algorithmic building blocks [2, 5, 6, 11], to graph algorithms [2, 3, 6], geometric algorithms and data structures [2, 11], as well as many cache-oblivious algorithms for dynamic programming and linear algebra problems [15]. I also proposed the first experimental framework for write-efficient algorithms in practice [19], and experimentally showed the performance improvements of the write-efficient algorithms.

Shared-memory Parallelism. Today, parallelism plays an important role in improving the performance of programs, given that the performance of a single CPU core has barely increased in more than ten years. Meanwhile, within the last decade, commodity multicore machines have become prevalent, and today these machines support up to terabytes of memory, which is more than enough for a majority of applications. As a result, it is of significant importance to study parallel algorithms.

I have designed, analyzed, and implemented many new parallel algorithms that are *simple*, and *efficient both in theory and practice*. Some of them have answered long-standing *open problems* theoretically (e.g., algorithms for incremental Delaunay triangulation and strongly connected components [10]). Most of them are significantly simpler than previous algorithms [4, 10, 13, 18, 24]. For all algorithms with implementations, their performance is considerably faster than the best previously existing ones [4, 10, 17, 18, 20, 24].

Because of the simplicity and efficiency, my algorithms are influential in many aspects. For example, the framework for directly parallelizing the sequential iterative algorithms introduced by myself and my collaborators has been introduced in keynote talks at top conferences and seminars. My algorithm for efficiently constructing space-partitioning trees [17] has become the programming assignments for related undergraduate courses at *CMU* and *Stanford*. Many of my other algorithms are integrated in graduate courses at CMU.

2 Write-Efficient Algorithms

Almost all previous algorithm research has focused on settings in which reads and writes (to the memory) have similar (symmetric) cost. This symmetric assumption simplifies the model, and roughly fits the current architecture. However, as stated, the new non-volatile memory technologies that will arrive soon have writes significantly more expensive than reads. Roughly speaking, the reason for this asymmetry is that writing to memory requires a change to the state of the material, while reading only requires detecting the current state. Based on the information currently available to us, the cost ratio between writes and reads is between 5 to 40 for the next-generation memories. This asymmetry poses the need to develop **write-efficient algorithms** that use significantly fewer writes than their classic counterparts. Many other motivations for studying read-write asymmetry have been discussed in my PhD thesis [1].

The motivations and challenges of asymmetric read and write costs inevitably lead to the following new questions. For example, how should the asymmetry and the new memory be modeled? How does this asymmetry impact the design of algorithms, and what new techniques are generally useful? What are the fundamental limitations on such trade-offs (lower bounds)? Can write-efficient algorithms also be parallel? Do new algorithms actually reduce the number of writes in practice, and if so, by how much? My PhD thesis [1] provides a comprehensive study of the read-write asymmetry in response to these questions, which are crucial for understanding both the theory and practice in the asymmetric setting.

Models. My first step is to introduce several computation and cost models to address read-write asymmetry [2, 5, 6]. These models represent the asymmetry with a parameter ω, indicating the cost of a write relative to a read. They consider various settings including sequential vs. parallel, explicit vs. implicit memory management, etc. These models allow theoretical analysis for write-efficient algorithms. Meanwhile, some system-level supports are needed in these models, which I have been discussed in these papers (e.g., asymmetry-aware cache-replacement policies and scheduling parallel algorithms on asymmetric memory).

Lower Bounds. With these models, the first straightforward question is whether we can achieve asymptotic improvement for existing algorithms. I proved lower bounds for many fundamental problems (e.g., sorting, fast Fourier transform, and many dynamic programming and linear algebra problems) [6, 15], indicating the (asymptotical) required number of writes under certain assumptions. Other researchers have also shown a list of interesting results based on these models.

Algorithms. The main contribution of this research series [2, 3, 5, 6, 10, 11, 15] consists of new write-efficient algorithms for fundamental algorithms in Computer Science, from basic algorithmic building blocks (sorting, reduce, filter, etc.), to graph algorithms ((bi)connectivity, shortest paths, MST, BFS, etc.), geometric algorithms and data structures (convex hull, Delaunay triangulation, k-d trees, augmented trees, etc.), as well as many cache-oblivious algorithms for dynamic programming and linear algebra problems. The numbers of writes in all these algorithms are significantly reduced asymptotically. Furthermore, most of these algorithms are also highly parallel. This is because new NVMs provide a much larger (terabyte-level) capacity than the systems they replace, so good parallelism is necessary to process data with such sizes in a timely manner.

When designing these algorithms, many techniques that are newly invented are of independent interest, since they are applicable to many other problems outside those studied in this thesis, and lead to improved algorithms in the classic setting without read-write asymmetry.

Experiments. I also proposed the first experimental framework to evaluate, reason, and analyze the performance of write-efficient algorithms in practice [19]. This framework is simple, clean and hardware-independent. Within the framework, a variety of different algorithms and data structures and their write-efficient implementations are discussed (including many of those introduced in my algorithm papers). Many of the implementations are non-trivial and require careful algorithmic design, analysis, and coding. Under the new measure with asymmetric read and write costs, I showed better implementations on all the problems that were evaluated, in comparison to their traditional and commonly-used counterparts in the symmetric setting. Many interesting algorithmic techniques that can be useful in designing and engineering write-efficient algorithms in the future have also been summarized.

3 Shared-Memory Algorithm Design

Given the performance of a single CPU core has not grown for more than a decade, the major solutions to improve the performance remains in seeking the parallelism in the computations. Traditional high-performance large-scale solutions are based on supercomputers or distributed clusters. However, within the past decade, commodity multicore machines have become prevalent and ubiquitous, and support up to terabytes of shared-memory[1], which is sufficient for a majority of applications. Recent research [25, 26] shows a single multicore machine can provide simple, efficient, and scalable solutions to many large-scale problems. My new algorithms are designed in this setting (i.e., shared-memory parallelism).

Design Goal: Simplicity, and Theoretical and Practical Efficiency. Parallel algorithms have been studied for about 40 years, and the traditional goal is to design parallel algorithms that are theoretically efficient with polylogarithmic parallel depth (highly-parallelism), and sequential work that matches that of the best sequential algorithm (work-efficiency). Work-efficiency for a parallel algorithm guarantees asymptotically no more work is wasted because of parallelism. Polylogarithmic depth limits the overhead caused by parallelism in practice. Many theoretically efficient algorithms have been designed, but unfortunately, many (not all) of these are sophisticated in the algorithmic details. As a result, they are either too complicated to be implemented, or have large hidden constant factors in the complexities. Conversely, many parallel algorithms are actually implemented in practice are not theoretically efficient, which prevents these delicate optimizations for more general settings. The goal of my research is to bridge this gap between theory and practice by developing parallel algorithms which are *simple* enough so that other researchers and even undergraduate students can learn and use them, and *theoretically and practically efficient*, which allows for good performance across all possible inputs, scalability across a wide range of core counts and to larger datasets. Achieving all three goals is the ultimate goal in designing parallel algorithms, and I admit that not all the works listed in this section can achieve them simultaneously. However, most of my results can improve at least two criteria from the best existing algorithms.

Framework for Randomized Iterative Algorithms. In this work, I focus on the list of problems including random permutation, list/tree contraction, comparison sort, d-dimensional convex hull and Delaunay triangulation, graph algorithms for least-element lists and strongly connected components (SCC) [7, 9, 10, 24]. All of these are fundamental problems that have already been widely studied. Unfortunately, the theoretically efficient parallel algorithms (except for list/tree contraction) are usually much more complicated than the sequential iterative algorithms. Such complication in some of the algorithms makes them impractical. Taking the random permutation as an example, the algorithm of Knuth (Durstenfeld's) shuffle contains only a few lines of pseudocode:

```
1 for $i \leftarrow 1$ to $n$ do
2     $A[i] \leftarrow i$
3     swap($A[i]$, $A[\textsc{Rand}(i)]$)
```

where $\textsc{Rand}(i)$ returns a random integer between 1 and i, and $A[\cdot]$ is the output random permutation. This sequential algorithm is trivial and can be taught to high school students. Previous parallel algorithms, however, are quite complicated and have reasonably large overhead (more discussions in [24]).

My research shows that these simple sequential iterative algorithms are readily parallel. This is achieved by an analytical framework of the dependence structure of the computation in the algorithms. Many of these algorithms have deep dependence chains in the worst case, but shallow chains (polylogarithmic with high probability) if the elements are randomly ordered. As such, these sequential iterative algorithms are highly parallel inherently, and only require slightly additional details to run the iterations in parallel. These algorithms are simple and theoretically efficient (work-efficient and polylogarithmic). Since not many additional operations are added to the simple sequential algorithms, the performance of the new parallel algorithm in practice is more efficient than the previous ones [24, 26].

The new framework leads to many interesting results, and answers several *long-standing open problems*. For example, it shows the first algorithm for incremental Delaunay triangulation with optimal work and polylogarithmic depth, which had been open for 30 years. This result is important since most implementations of parallel Delaunay triangulation use the incremental approach. For the widely-used Coppersmith et al.'s SCC algorithm, which is proposed in the year 2000, this framework shows the parallelism in the computation ($O(\log n)$ reachability queries), and the corresponding parallel algorithm. This result has been conjectured for a long while but had never been shown. The analysis of dependence structures can also be extended for write-efficiency [11], so that all these new parallel algorithms are write-efficient. While the existing results are already very interesting, I plan to extend this framework to more new problems and implementations.

[1]For example, Amazon AWS service rents multicore machines with 768 GB shared-memory. The existing Dell PowerEdge R940 can be configured with 96 cores and 6 TB of memory. The new Intel's Cascade Lake AP processors support 6 TB of system memory per socket based on the new 3D XPoint NVMs.

Graph Analytics. I have designed various graph algorithms and solutions for fundamental problems in graph analytics. As discussed in the previous section, my research shows a simple and parallel algorithm to compute strongly connected components (SCCs), which by far has the best theoretical bounds. The algorithm was later implemented and shown experimentally to be practical by Dhulipala et al. [26].

The single-source shortest path problem is a fundamental graph problem that has been extremely well-studied. The version with nonnegative weights is notoriously difficult to solve efficiently in parallel, and the previous algorithms with good work and depth bounds are very complicated. I have shown a theoretically efficient algorithm [13] that has the same bounds as the best previous algorithms, but this algorithm and its analysis are much simpler and can easily be taught in regular courses.

I also studied graph embedding and distance oracles. Querying pairwise distances is one of the fundamental problems in graph analytics. When there are many queries, computing the exact distances can be costly, so a common compensation is to tolerate a small error in trading for faster query speed. A distance oracle is the data structure to support such queries. This structure has been studied extensively both theoretically and experimentally, but the existing practical algorithms are still slow, and theoretical works are even less practical. My new approach comprises embedding the graphs into trees and subsequently using the tree distances to approximate graph distances. The idea of tree embedding (FRT embedding) is very simple, and I have discussed how to apply them to graphs efficiently regarding good sequential work [12] and parallel depth [10]. My ongoing work [4] implements and experiments the algorithm, which indicates that this approach is significantly (more than 10x) faster on construction and can work on larger graphs with orders of magnitude more vertices, while achieving a competitive approximation of the distances.

Sorting and Searching. Sorting and searching are fundamental problems in general computing and programming. I have designed a variety of parallel algorithms that are theoretically and practically efficient.

I designed an algorithm for semisorting [18] which is the problem of reordering an input array of keys such that equal keys are contiguous but different keys are not necessarily in sorted order. Semisorting collects equal values and is a widely-used primitive in parallel computing applications (e.g., the shuffle step in MapReduce paradigm, or grouping in databases). My algorithm is the first theoretically efficient algorithm (optimal work and logarithmic depth) for semisorting. Further, it is very simple, and the experimental results show that it outperforms a parallel integer sorting and comparison sorting algorithms by a factor of 1.5–2, and achieves an almost perfect parallel speedup.

I am currently working on improving the performance of sorting [20]. The improvement is achieved by utilizing the numerical values of the keys. By applying an optimization based on the key distribution, each item can find the approximate destination quickly. Experiments are run based on a variety of inputs, and the results show an consistent 20-30% performance improvement as compared to the best existing algorithms on comparison sorting.

I also designed new algorithms to improve the quality and construction speed of bounding volume hierarchies (BVHs), which are the most commonly used accelerating structures for low-dimensional nearest neighbor search and graphics applications like ray tracing and collision detection. Usually, a bottom-up approach based on agglomerative clustering produces hierarchies with better quality, but it suffers from low construction speed (by about 10x slower than the top-down approaches sequentially, and cannot be parallelized efficiently). My new algorithm [17] is an approximate version of agglomerative clustering, which significantly accelerates the sequential construction time and provides good parallelism, while the deterioration of BVH quality caused by approximation is negligible. Thus, my new algorithm was much faster than existing algorithms at the time when I invented this algorithm, with the output BVHs to have similar quality. Due to its simplicity and efficiency, this algorithm was chosen as the programming assignment in undergraduate courses in related areas at *Stanford* and *CMU*. Many later algorithms on this topic are based on this idea, and further improve the performance of this algorithm by adding more details, hybrid algorithms with the top-down approaches, or applying this algorithm to other platforms (such as GPUs).

Other than construction efficiency, I have also designed a post-processing algorithm (i.e., after the construction of BVHs) to improve the quality [16]. The algorithm is based on a data-driven approach that can learn statistics from a few samples and adjust the shape of the BVHs accordingly. This is the first learning-based algorithm (previous algorithms are based on static heuristics), which inspires many later algorithms.

Efficient Approaches on Other Topics. Prior to CMU, I designed parallel and efficient algorithms to manipulate images [22], new geometric representations for origami pop-ups for efficient availability checking and simulation [23], and a theoretically efficient algorithm (the first polynomial algorithm) for a barrier coverage problem in wireless sensor networks [14]. In my first PhD year, I studied new abstractions to improve shading efficiency on GPUs [21].

4 Research Vision

The main theme of my research is to design efficient algorithms and approaches to improve the performance of many fundamental problems and applications. Given the continued growth in the volume of data and the emergence of many new computing technologies (such as more powerful multicore machines and non-volatile memories), more challenges are arising, and I plan to help address them in my future research.

My research is more than analyzing the algorithms mathematically or restructuring a computation to take advantages of the architecture. The high-level goal is to bridge the gap between theory and practice. This goal is achieved by investigating efficient theoretical models, and based on them, I design, analyze and implement simple but efficient algorithms. Since the improvements are from the algorithmic side, a robust speedup is shown on a wide range of platforms. I also consider other approaches based on *programming techniques, languages and system-level optimizations*. My research considers all aspects to improve the performance, and my future research seeks a unified approach to solve more problems on various architectures.

Today is a critical time for studying computing, given that new hardware technologies have arisen with significantly useful properties. However, fully utilizing the new technologies is non-trivial, because they may force us to rethink the philosophy to design algorithms. Meanwhile, it is also less likely for computer scientists in all areas to continually follow up on the entire range of all these techniques. As a result, my long-term goal is to *develop efficient solutions that consider both theoretical efficiency and new architectural evolution, and apply them to different research areas*.

I will conclude my potential future work in three categories. My research during the PhD period focused on shared-memory parallelism and write-efficient algorithms. These topics are new and the need of the hour, so there remain numerous topics for future work that are worth exploring. I am also interested in investigating more platforms (e.g., GPUs). Lastly, given that seeking better performance is also crucial in many other research areas, I am looking forward to extending my research area to a larger scope in the future, and collaborating with my future colleagues and other researchers in a variety of areas.

Shared-Memory/In-Memory Computing. Classic approaches to handle large-scale data are based on distributed or external memories. With the recent evolving of hardware technologies, today's computing platform supports up to approximately 100 CPU cores sharing terabytes of memory. Once the memory size is sufficient for an application, such an approach simplifies implementations and eliminates the expensive accesses to the external memories or communication on distributed systems. My previous research has enhanced the research of this topic by proposing multicore parallel algorithms with simplicity and better performance, and a comprehensive study on write-efficient algorithms for NVMs which will support a much larger memory size.

Given the significant amount of evidence that shared-memory/in-memory computing will be a main trend in the future, I plan to continue working on related topics. Since this setting is relatively new, there still remains much room for improving the performance and simplicity of many algorithms. For write-efficient algorithms, given that the new hardware will be available very soon, my future work plans to integrate highly-optimized implementations in a library, so that researchers and programmers can directly utilize them. Other than write-read asymmetry, NVMs are persistent, meaning that they have the capability of surviving power outages and other failures without losing data. I plan to continue working on extending my previous work [8] and designing simplified programming models to support algorithms and data structures resilient to faults. In conclusion, it is a fascinating time to do research on related topics to support efficient shared-memory/in-memory computation.

GPUs and Other Newly Emerged Hardware. Multicore parallelism provides a general solution for performance improvement. Many other hardware technologies developed recently provide alternative options for some particular applications and yield good performance for them. For example, general-purpose GPUs have much more less-powerful processors and weaker memory consistency. While they do show better performance on many applications, the performance is hard to predict without implementations, and programming on GPUs is more complicated. I will move to MIT and join a project there in the next (spring) semester to investigate theoretical models, simplified programming techniques and support for graph analytics on GPUs. A long-term goal is to understand GPU parallelism and performance as well as what we know about CPUs. Other interesting topics also include near-memory computing, as well as low-precision computation with applications in machine learning.

Efficient Algorithms in More Areas. Performance is important for almost all areas in computer science. For instance, in graph mining and computational biology, many graph algorithms are used to gather useful information from input graph instances with millions or billions of vertices. Computer graphics requires various space-partitioning data structures to handle meshes with up to billions of vertices for generating high-resolution output images. Databases support many functions to manipulate organized data in terabyte size. Machine learning uses different algorithms to cluster massive datasets. Reducing the execution time of these applications is valuable for increasing productivity and saving resources. Meanwhile, the computations of these algorithms or problems are input dependent, and therefore improving the performance is generally challenging.

My research during the PhD period mostly focuses on the classic problems and algorithms (e.g., those in *Introduction to Algorithms*). These algorithms have been extensively studied for decades; thus, improving their performance is more difficult, and this experience has equipped me with a deep understanding of algorithm design and performance engineering. I am also very interested in addressing the performance challenges in other research areas with high real-world impacts. I am looking forward to collaborating with my potential future colleagues who are experts in these areas. I can provide my fresh mind as an outsider, and my experience and expertise in designing efficient algorithms on modern and future platforms.

References

(Author lists for papers [2]–[14] are in alphabetical order, following the convention in mathematics and theoretical computer science. Others are listed by contribution.)

[1] **Yan Gu**. *Write-Efficient Algorithms*. Ph.D. Thesis, Carnegie Mellon University, 2018.

[2] Naama Ben-David, Guy E Blelloch, Jeremy T Fineman, Phillip B Gibbons, **Yan Gu**, Charles McGuffey, and Julian Shun. Parallel algorithms for asymmetric read-write costs. In *ACM symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2016.

[3] Naama Ben-David, Guy E Blelloch, Jeremy T Fineman, Phillip B Gibbons, **Yan Gu**, Charles McGuffey, and Julian Shun. Implicit decomposition for write-efficient connectivity algorithms. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2018.

[4] Guy E Blelloch, Laxman Dhulipala, and **Yan Gu**. Efficient parallel constructions of all-distance sketches and distance oracles. *Manuscript*.

[5] Guy E Blelloch, Jeremy T Fineman, Phillip B Gibbons, **Yan Gu**, and Julian Shun. Sorting with asymmetric read and write costs. In *ACM symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2015.

[6] Guy E Blelloch, Jeremy T Fineman, Phillip B Gibbons, **Yan Gu**, and Julian Shun. Efficient algorithms with asymmetric read and write costs. In *European Symposium on Algorithms (ESA)*, 2016.

[7] Guy E Blelloch, Jeremy T Fineman, **Yan Gu**, and Yihan Sun. Optimal parallel algorithms in binary-forking model. *Manuscript*.

[8] Guy E Blelloch, Phillip B Gibbons, **Yan Gu**, Charles McGuffey, and Julian Shun. The parallel persistent memory model. In *ACM symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2018.

[9] Guy E Blelloch, and **Yan Gu**. Sequential incremental convex hull are highly parallel. *Manuscript*.

[10] Guy E Blelloch, **Yan Gu**, Julian Shun, and Yihan Sun. Parallelism in randomized incremental algorithms. In *ACM symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2016.

[11] Guy E Blelloch, **Yan Gu**, Julian Shun, and Yihan Sun. Parallel write-efficient algorithms and data structures for computational geometry. In *ACM symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2018.

[12] Guy E Blelloch, **Yan Gu**, and Yihan Sun. Efficient construction of probabilistic tree embeddings. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2017.

[13] Guy E Blelloch, **Yan Gu**, Yihan Sun, and Kanat Tangwongsan. Parallel shortest paths using radius stepping. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2016.

[14] Danny Z Chen, **Yan Gu**, Jian Li, and Haitao Wang. Algorithms on minimizing the maximum sensor movement for barrier coverage of a linear domain. *Discrete & Computational Geometry*, 50(2):374–408, 2013.

[15] **Yan Gu**. Improved parallel cache-oblivious algorithms for dynamic programming and linear algebra. *arXiv preprint:1809.09330*, 2018.

[16] **Yan Gu**, Yong He, and Guy E Blelloch. Ray specialized contraction on bounding volume hierarchies. In *Computer Graphics Forum*, volume 34, pages 309–318, 2015.

[17] **Yan Gu**, Yong He, Kayvon Fatahalian, and Guy E Blelloch. Efficient BVH construction via approximate agglomerative clustering. In *High-Performance Graphics Conference*, 2013.

[18] **Yan Gu**, Julian Shun, Yihan Sun, and Guy E Blelloch. A top-down parallel semisort. In *ACM symposium on Parallelism in algorithms and architectures (SPAA)*, 2015.

[19] **Yan Gu**, Yihan Sun, and Guy E Blelloch. Algorithmic building blocks for asymmetric memories. In *European Symposium on Algorithms (ESA)*, 2018.

[20] **Yan Gu**, Yibin Yang, Yihan Sun, and Guy E Blelloch. Improving sorting performance via interpolation. *Manuscript*.

[21] Yong He, **Yan Gu**, and Kayvon Fatahalian. Extending the graphics pipeline with adaptive, multi-rate shading. *ACM Transactions on Graphics (TOG)*, 33(4):142, 2014.

[22] Xian-Ying Li, **Yan Gu**, Shi-Min Hu, and Ralph R Martin. Mixed-domain edge-aware image manipulation. *IEEE Transactions on Image Processing*, 22(5):1915–1925, 2013.

[23] Xian-Ying Li, Tao Ju, **Yan Gu**, and Shi-Min Hu. A geometric study of v-style pop-ups: theories and algorithms. In *ACM Transactions on Graphics (TOG)*, volume 30, page 98. ACM, 2011.

[24] Julian Shun, **Yan Gu**, Guy E Blelloch, Jeremy T Fineman, and Phillip B Gibbons. Sequential random permutation, list contraction and tree contraction are highly parallel. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2015.

[25] Julian Shun. *Shared memory parallelism can be simple, fast, and scalable*. Ph.D. Thesis, Carnegie Mellon University, 2015 ACM Dissertation Award.

[26] Laxman Dhulipala, Guy E Blelloch, and Julian Shun. Theoretically efficient parallel graph algorithms can be fast and scalable. In *ACM symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2018, best paper.