Yuan Tang

Research Statement

My research interest is the performance engineering, an interdisciplinary research area at the intersection of parallel and concurrent computing, programming languages, systems and theory. The goal of my research is to obtain scalable and portable performance for real-world applications and algorithms on modern computing systems by exploiting caching and parallelism. The applications and algorithms I have been working on include, but not limited to, dense linear algebra, and dynamic-programming computations, which are widely used in computational biology and image processing, and would like to work on regular and irregular algorithms and data structures such as those used in artificial intelligence, data mining and machine learning. My approach is to design and implement high-performance programming systems based on solid theoretical foundations. My work ranges from traditional high-performance computing [YZ08; YCT07; YFD06; Yua+03; Yua+05; YS05], programming models [Bac+16; DSY16; Yua+15; Yua+14], domain-specific compilation [Bac+16; Yua+11a; Yua+11b], and provably efficient runtime schedulers [Cho+16; DSY16], to general algorithms and data structures [Mic+14; Lui+14].

My recent work on “cache-oblivious wavefront algorithms” and the “Nested Dataflow Model” [Yua+15; Yua+14] provides the possibility to achieve optimal time bounds within the optimal space and cache bounds processor-obliviously (i.e. optimality is achieved independently of processor count) without tuning on shared-memory parallel computing systems. My research suggests that people may have to re-think the way of parallel programming. I plan to extend my research to distributed-memory systems so that wider areas of applications such as traditional high-performance computing, cloud computing, big data, and the Internet of Things can also benefit from the research. I plan to incorporate my research into a strict mathematical model, possibly unified with polyhedral model for algorithm manipulation and derivation, as well as theoretical analyses, reasoning and automatic theorem / property proof. Another direction of future research will be developing a declarative domain-specific language so that my research can be easily used by the community.

Motivation and Overview

Performance is the currency in the world of computing. A programmer can always trade performance for new functionality such as security, reliability, among others. As the end of Moore’s Law approaches, free performance gains from hardware upgrades become unattainable. Every programmer has to be a performance engineer.

There are two metrics commonly considered in parallel computations: time complexity and cache complexity. The traditional objective for scheduling a parallel computation is to minimize the time complexity: the running time of a parallel computation on an infinite number processor system (the overhead of cache misses is counted separately). Alternatively, one can minimize the cache complexity: the number of cache misses incurred during the execution of program. Theoretical analyses often consider these metrics separately; in reality, the actual completion time of a program depends on both, since the number of cache misses has a direct impact on the running time and the time bound often serves as a good indicator of scalability.
Tuning algorithms to optimize time and/or cache complexity has limits in practice because the approach not only makes the code structure more complicated, but also requires searching over an exponentially-sized parameter space. Moreover, tuning is non-portable and non-cache-adaptive. Even a well-tuned code will suffer if the availability of cache fluctuates during its execution time.

Generally speaking, in the classic nested parallel programming model (also known as fork-join model), a 2-way divide-and-conquer algorithm asymptotically achieves optimal cache complexity but sub-optimal time bound; in contrast a straightforward parallel looping algorithm for the same problem achieves the optimal time bound but not cache bound. Traditional wisdom may suggest tuning for some balance point between time and caching to obtain “good” performance in practice. However, the intuition behind balance is that achieving optimal bounds for both metrics simultaneously is impossible. My research challenges this traditional wisdom from both algorithm’s perspective and programming model’s perspective. My research shows that both can be made optimal via either “cache-oblivious wavefront” algorithms or programming in the new “nested dataflow model”.

### Cache-Oblivious Wavefront algorithms

A cache-oblivious wavefront (COW) algorithm [Yua+15] performs the same divide-and-conquer as classic cache-oblivious parallel algorithm, but aligns the progress of derived sub-quadrants across different levels of recursion to a wavefront. Fundamentally, COW algorithm schedules all nodes, each of which represents a quadrant in some level of recursion, of a spawn tree (a parallel analogue of depth-first activation tree of a serial execution) based on their data dependencies thus yields an optimal time bound. Since COW algorithm retains the original divide-and-conquer behavior and maintains the same recursive computing order at all levels of recursion, the cache efficiency is not sacrificed compared to original cache-oblivious parallel algorithms. I have developed COW algorithms for several typical dynamic-programming computations such as stencil computation, Floyd-Warshall All-Pairs-Shortest-Paths, LCS, GAP, Parenthesis, and achieved the optimal time bounds within the optimal space and cache bounds. The set of COW algorithms outperform their counterparts of both the original cache-oblivious parallel algorithms and parallel loop tiling algorithms in practice if employing the same kernel functions for the base case computations.

### Nested Dataflow Model

In the nested parallel model (also known as the fork-join model), a spawn tree can be rooted by one of the two primitives, i.e. “∥” (parallel) or “;” (serial), and its left and right subtrees are then recursively defined up to leaves. The leaves of a spawn tree are computations that contain no primitives. Artificial dependency is the control dependency imposed by parallel constructs in addition to the fundamental data dependency that are essential to the correctness of algorithm. In contrast, artificial dependency is not necessary for the correctness but may hurt the time complexity of an algorithm. A close examination reveals that the insufficiency of nested parallel model in expressing “partial dependency” in a spawn tree is the fundamental reason that causes artificial dependency in many divide-and-conquer algorithms.

I propose extending the nested parallel model to the “Nested Dataflow (ND) Model” [DSY16] by simplifying the two primitives (“∥” or “;”) to only one but more powerful dataflow operator “∼” (Pronounced “Fire”). A spawn tree rooted by the “∼” operator indicates there is a partial...
dependency between its sink (right) subtree and source (left) subtree. The partial dependency at recursion level \( i \) can then be defined as a set of partial dependencies at recursion level \((i + 1)\) up to leaves. These semantics allow the original two primitives “\( || \)” and “;” to be defined as syntactic sugar for “\( \sim \)” in two extreme cases, allowing us to subsume the classic nested parallel model into the new nested dataflow model.

The design goal of ND Model is to make inter-processor execution work in a data-driven manner similar to the dataflow model, while the intra-processor execution is always in a depth-first order with respect to the spawn tree processor-obliviously. I redesigned several typical divide-and-conquer algorithms in the ND model and achieved optimality in time, space, and caching simultaneously and processor-obliviously without any tuning. The set of divide-and-conquer algorithms ranges from dense linear algebra to dynamic-programming. I proposed one extension on classic randomized work-stealing scheduler to map a multithreaded ND computation to a \( P \)-processor system with provably good bounds.

---

**The Pochoir Stencil Compiler**

I have worked at MIT since 2009 on “The Pochoir (Pronounced “PO-shwar”) Stencil Compiler” [Yua+14; Yua+11a; Yua+11b]. Stencil computation is a special case of dynamic-programming where each cell in a multi-dimensional table is updated by a constant number of neighboring cells. The stencil computation is widely used in iterative PDE solvers such as Jacobi, multi-grid, and adaptive mesh refinement, as well as in image processing and geometric modeling. In this project, I worked with the Pochoir team and achieved following results: 1) improving the parallelism asymptotically with the same cache efficiency of a cache-oblivious parallel algorithm for higher-dimensional simple stencil computation by inventing “hyperspace cut”; 2) handling periodic and aperiodic boundary condition in one unified algorithm; 3) designing domain-specific language (DSL) embedded in C++ for stencil computation; 4) designing and developing a novel two-phase compilation strategy that first uses any C++ toolchain to verify correctness and subsequently invokes the Pochoir compiler to do a source-to-source transformation for a highly optimized code. The two-phase compilation strategy reduces massive cost of parsing and type-checking of C++ language. Building on the success of the Pochoir project, the Pochoir team is currently working on developing the Bellmaniac software synthesis system for “general” dynamic-programming algorithms, where the number of dependent cells to update any cell in a multi-dimensional table does not have to be constant. The work on Bellmaniac system initiated the above ideas of the “cache-oblivious wavefront algorithms” and the “nested dataflow model”.

---

**Other achievements in algorithms and data structures**

I have broad research interests in wide area of parallel algorithms and data structures. For example, my recent study on Range 1 Query algorithms, a special case of Range Partial Sum Query algorithm, where all values on the discrete grid are either 1 or 0, was published in COCOON’14 [Mic+14]; Another piece of my work concerns weight balance on boundaries and skeletons, an inverse problem of the barycenter problem, was published in SoCG’14 [Lui+14]. Given a set of \( n \) weights \( W = \{w_1, w_2, \ldots, w_n\} \) and \( n \) arbitrary locations \( X = \{x_1, x_2, \ldots, x_n\} \) on the boundary of an arbitrary multi-dimensional polygon, one can easily calculate the barycenter in \( O(n) \) time by formula \( x = \sum_{i=1}^{n} w_i \cdot x_i \); The inverse problem is that given an arbitrary point
in/outside the convex/concave polygon and the weight set $W$, how fast can we actually find $n$ locations $X = \{x_1, x_2, \ldots, x_n\}$ on the boundary of polygon to place the $n$ given weights so that their barycenter is the given point?

---

**Summary and Future Directions**

I have been exploiting performance on modern computing systems by proposing the “cache-oblivious wavefront algorithms” and the “nested dataflow programming model”. Going forward, my work opens up new opportunities in the research areas on parallel programming model, parallel programming language, provably efficient runtime scheduler, algorithms and data structures.

The restricted expressiveness of nested parallel model is known in the community. Several approaches such as futures, synchronization variables, pipeline parallelism, and task graph parallel execution, were proposed to express more general class of parallel computations. However, none of these approaches consider the possibility of achieving the optimal time bounds within the optimal space and cache bounds processor-obliviously without tuning.

### Parallel Programming Language:

A good programming model should be presented by elegantly designed linguistic constructs such as `spawn` and `sync` in Cilk language. Since “$\sim$” operator of the ND model is a natural extension to the “$\parallel$” and “$;$” primitives in the nested parallel model, I plan to investigate the linguistic constructs and optimizing implementation at the level of programming language to make the ND model widely usable in the parallel programming community. Conceivably, many existing parallel algorithms and data structures have to be redesigned in the new ND model.

### Domain-Specific Compilation and Language:

From stencil to dynamic-programming computation, almost all applications of my research fall into the category of problems that can be formulated in the polyhedral model. Though my approaches are from a perspective of divide-and-conquer based cache-oblivious and processor-oblivious methodology, I would like to explore the possibility of formalizing the research into one strict mathematical model, possibly unified with the polyhedral model, for easy algorithm manipulation and derivation, as well as theoretical analyses, reasoning, and automatic theorem / property proof. Moreover, I plan to investigate designing and developing a declarative domain-specific language for wide acceptance in the community.

### Applications to computational biology, image processing, Big Data, and large-scale machine learning:

The core algorithms behind computational biology and image processing are stencil or more generally dynamic-programming computations. For instances, “Basic RNA Secondary Structure Prediction”, which is a fundamental problem in computational biology, and Viterbi algorithm, which is widely used in speech recognition, are essentially dynamic-programming problems with non-constant dependencies; the H.264 video coding algorithm has a similar data dependency pattern to that of LCS (Longest Common Subsequence), thus a similar computing algorithm. My expertise in optimizing general dynamic-programming algorithms can give me big advantages in dealing with these applications.

The key of Big Data is to process larger data sets faster, which lies exactly in the sweet spot of my research area, the “performance engineering”. Some potential application of my research on the Big Data can be high-performance / parallel data analytics, among others.

One of the hot researchs on large-scale machine learning focuses on the scheduling algorithm for efficient resource management, minimizing the communication and caching overhead on
modern computing systems, including both distributed and shared-memory systems. My expertise in provably efficient scheduling algorithms and cache-oblivious / processor-oblivious wavefront algorithms, as well as parallel programming model, can be of big help in this research field.

References


