My research interest is the **performance engineering**, an interdisciplinary research area at the intersection of parallel and concurrent computing, computational and data science, 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 work, time, or equivalently parallelism for parallel computations, space, and caching. The applications and algorithms I have been working on include, but not limited to, dense linear algebra, and dynamic-programming computations with constant or non-constant dependencies, which are widely used in computational biology and image processing, and would love to work on regular and irregular algorithms and data structures such as those used in artificial intelligence, Big Data, 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 \[ChowdhuryGaTa16; DSY16\], to general algorithms and data structures \[Mic+14; Lui+14\].

My recent work on “cache-oblivious wavefront algorithms” and the “Nested Dataflow (ND) Parallel Programming Model” \[Yua+15; Yua+14\] provides the possibility to achieve optimal time bounds within the optimal space and cache bounds without tuning in a processor- and cache-oblivious fashion. By processor-oblivious, I mean that the optimality is achieved independently of processor count; By cache-oblivious, I mean that the algorithm doesn’t parametrize on the cache model / hierarchy. 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 my research.

### 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. Alternatively, one can minimize the amount of cache misses incurred during the computation. Theoretical analyses often consider these metrics separately; in reality, the actual running time of a computation depends on both, since the unit cost of cache misses is even more expensive than that of an arithmetic operation in modern computing systems, while the parallel time bound often serves as a good indicator of the scalability of the algorithm.

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.

Generally speaking, in the classic Nested Parallel (NP) programming model (also known as fork-join model), a 2-way divide-and-conquer algorithm achieves asymptotically optimal cache complexity but sub-optimal time bound; in contrast a straightforward parallel looping algorithm for the same problem can achieve the optimal time but sub-optimal cache bound. Traditional wisdom may suggest to tune for some balance point between the two 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 optimal is possible via either the “cache-oblivious wavefront” technique or programming in the “Nested Dataflow (ND) 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-computations across different levels of recursion to a wavefront. Fundamentally, a COW algorithm schedules all sub-computations based on their data dependencies thus yields an optimal time bound while inheriting the recursive computing order from the divide-and-conquer algorithm to preserve the cache efficiency in a cache-oblivious fashion. 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 work, time, space, and cache bounds simultaneously. These COW algorithms not only have provably optimal bounds in theory, but also outperform their cache-oblivious parallel and parallel loop tiling counterparts in practice if using the same kernel functions for the base case computations.

### Nested Dataflow Model

In the Nested Parallel (NP) programming model (also known as the fork-join model), there are two primitives to construct a parallel program, i.e. “||” (Parallel) and “;” (Serial). The notation “a || b” denotes that task b has no dependency on a at all, thus these two tasks can execute in arbitrary order, i.e. concurrently; “a ; b” says that task b has a full dependency on all subtasks of a, thus has to wait until a completes. However, in a recursive divide-and-conquer algorithm, there is a possibility that only a subset of subtasks of b depends on a subset of subtasks of a, the relation of which we call partial dependency. Due to the fact that the classic NP model is unable to express the partial dependency, programming in the NP model can cause artificial dependency, i.e. the control dependency that is un-necessary for the correctness of an algorithm. The existence of artificial dependency, or equivalently the inability of the NP model in expressing the partial dependency, is the fundamental reason that lots of cache-oblivious parallel algorithms that base on the recursive divide-and-conquer behavior can have optimal cache bounds but sub-optimal parallel time bounds.

We propose to extend the NP model to the “Nested Dataflow (ND) Parallel Programming Model” [DSY16] by generalizing the two primitives (“||” and “;”) to one more powerful dataflow construct “~” (Pronounced “Fire”). The notation “a ~ b” indicates that task b has only a
partial dependency on \(a\), i.e. a subset of subtasks of \(b\) depends on a subset of subtasks of \(a\). The partial dependency at recursion level \(i\) will then be refined by a corresponding fire rule to a set of partial dependencies at recursion level \((i + 1)\) until both its left and right tasks are leaves, i.e. one unit of basic computation. The \(\rightsquigarrow\) construct between two leaves will reduce to a \(\parallel\) or \(;\) primitive as needed. This semantics of the \(\rightsquigarrow\) construct makes the classic primitives of \(\parallel\) and \(;\) a syntactic sugar for two special cases. Thus, the ND model completely subsumes the classic NP model.

The design goal of the ND Model is to make the inter-processor execution work in a dataflow fashion to maximize the parallelism, i.e. a task is ready for scheduling as soon as all of its source data are ready, while the intra-processor execution to be aligned with the depth-first order of the sequential execution to minimize cache misses. I redesigned several typical divide-and-conquer algorithms in the ND model and achieved optimality in work, time, space, and caching without tuning simultaneously in a processor- and cache-oblivious fashion. The set of divide-and-conquer algorithms ranges from dense linear algebra to dynamic-programming.

---

**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 for higher-dimensional simple stencil computation by inventing “hyperspace cut”; 2) unifying the handling of periodic and aperiodic boundary conditions in one algorithm; 3) designing and developing a domain-specific language (DSL) embedded in C++ for the stencil computation; 4) designing and developing a novel two-phase compilation strategy that reduce the massive cost of parsing and type-checking of the Stencil DSL embedded in C++. Building on the success of the Pochoir project, the Pochoir team is currently working on the Bellmaniac software synthesis system for the “general” dynamic-programming problems, where the number of dependent cells to update any cell in a multi-dimensional table does not have to be constant. The work on the 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 wider area of parallel algorithms and data structures. For instance, my recent study on the Range 1 Query algorithms, a special case of the Range Partial Sum Query algorithm, was published in COCOON’14 [Mic+14]; Another piece of my work concerns weight balance on the boundaries of convex / concave polygons, an inverse problem of the barycenter problem, was published in SoCG’14 [Lui+14].

---

**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.

**Provably Optimal Runtime Scheduling Algorithm:** Though the nested dataflow model provides and maintains necessary data structures to achieve optimality in work, time, space and caching, significant efforts needs to be leveraged to fully exploit the data structures to have a provably optimal runtime scheduling.

**Parallel Programming Language:** A good programming model should be presented by elegantly designed linguistic constructs such as `spawn` and `sync` in Cilk language. Since the “∼” construct of the ND model is a natural extension to the “∥” and “;” primitives, I plan to investigate on the linguistic construct and efficient implementations to make the ND model widely usable in the parallel programming community. Conceivably, many existing parallel algorithms and data structures may have to be redesigned in the new ND model.

**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 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 runtime scheduling algorithms and cache- / processor-oblivious wavefront algorithms, as well as parallel programming model, can be of big help in this research field.

**“Conditional dependency”** : Conditional dependency is the dependency that can be changed by runtime values. Conditional dependencies are common in irregular computation and data structures such as graphs, trees and sets. Correspondingly, there have been intensive studies on the best way to formulate regular or irregular parallelism, respectively. I am interested in working on both fields and how to possibly connect one field to the other.

---

**References**


