Automatic Discovery of Efficient Divide-&-Conquer Algorithms for Dynamic Programming Problems

Pramod Ganapathi

Vision and motivation. In our vision of the not-so-distant future of computing, machines will take the responsibility of designing and implementing most, if not all, machine-specific highly efficient nontrivial algorithms with provable correctness and performance guarantees. Indeed, as the complexity of computing hardware grows and their basic architectures keep changing rapidly, manually redesigning and reimplementing key algorithms for high performance on every new architecture is quickly becoming an impossible task. Automation is needed.

We will talk about designing algorithms that can design other algorithms. Our focus is on auto-generating algorithms for solving dynamic programming (DP) recurrences efficiently on state-of-the-art parallel computing hardware. While DP recurrences can be solved very easily using nested or even tiled nested loops, for high performance on modern processors/coprocessors with cache hierarchies, portability across machines, and automatic adaptivity to runtime fluctuations in the availability of shared resources (e.g., cache space) highly nontrivial recursive divide-and-conquer algorithms are needed. But these recursive algorithms are difficult to design and notoriously hard to implement for high performance. Furthermore, new DP recurrences are being encountered by scientists every day for solving brand new problems in diverse application areas ranging from economics to computational biology. But how does an economist or a biologist without any formal training in computer science design an efficient algorithm for evaluating his/her DP recurrence on a computer? Well, we can help!

Dissertation statement. It is possible to develop algorithm(s) / framework(s) to automatically / semi-automatically discover algorithms that are simultaneously nontrivial, simple, fast, portable, and robust, which can be used to solve to a wide class of DP problems.

My contributions. I am the major contributor in the research projects that I will be describing in further sections. I have designed algorithms, developed frameworks, proved correctness, analyzed complexities, and have written major portions of the papers under the guidance of my adviser. The implementations are not mine.

Major research highlights. The dissertation will be an excursion into computer-generated algorithms and computer-aided algorithm design.

We present Autogen – an algorithm that given any blackbox (e.g., inefficient naive serial loops) implementation of a DP recurrence from a wide class of DP problems can automatically discover a fast recursive divide-and-conquer algorithm for solving that problem on a shared-memory multicore machine. We mathematically formalize Autogen, prove its correctness, and provide theoretical performance guarantees for the auto-discovered algorithms. These auto-generated algorithms are shown to be efficient (e.g., highly parallel with highly optimizable kernels, and cache-, energy- and bandwidth-efficient), portable (i.e., cache- and processor-oblivious), and robust (i.e., cache- and processor-adaptive).

We present AUTOGEN-WAVE – a framework for computer-assisted discovery of fast divide-and-conquer wavefront versions of the algorithms already generated by AUTOGEN. These recursive wavefront algorithms retain all advantages of the AUTOGEN-discovered algorithms on which they are based, but have better and near-optimal parallelism due to the wavefront order. We also show how to schedule these algorithms with provable performance guarantees on multicore machines.

As a first step toward extending AUTOGEN to handle DP recurrences with irregular data-dependent dependencies, we design an efficient cache-oblivious parallel algorithm to solve the Viterbi recurrence. Our algorithm beats the current fastest Viterbi algorithm in both theory and practice.

We present AUTOGEN-FRACTILE – a framework to design high-performing and easily portable GPU algorithms based on recursive divide-and-conquer and works for a wide class of DP problems. Our CPU-GPU algorithms have excellent cache locality and can be easily extended to work on external-memory.

We also plan to extend AUTOGEN to discover efficient DP algorithms for the MapReduce framework as well as for distributed-memory clusters.

AUTOGEN. AUTOGEN [Chowdhury et al., 2016b] is an algorithm that given any blackbox implementation of a DP recurrence from a wide class of DP problems can automatically discover a fast recursive divide-and-conquer algorithm for solving that problem on a shared-memory multicore machine. AUTOGEN analyzes the set of DP table locations accessed by the input algorithm (e.g.: serial iterative) when run on a DP table of small size, and automatically identifies a recursive access pattern and a corresponding provably correct recursive algorithm for solving the DP recurrence.

Consider the parenthesis dynamic program [Galil and Park, 1994]. Variations of the algorithm are used for chain matrix multiplication, database joins, parsing context-free grammars, and so on. Figure 1 shows an iterative algorithm, a parallel iterative code generated from a parallelizer, and a recursive divide-and-conquer algorithm auto-discovered from AUTOGEN.

We use AUTOGEN to auto-discover efficient recursive algorithms for several well-known problems. Table 1 summarizes the superiority of the recursive algorithms over their iterative counterparts. Our experimental results show that several auto-discovered algorithms significantly outperform parallel looping and tiled loop-based algorithms. Also, these algorithms are less sensitive to fluctuations of memory and bandwidth compared with their looping counterparts, and their running times and energy profiles remain relatively more stable. To the best of our knowledge, AUTOGEN is the first algorithm that can automatically discover new nontrivial divide-and-conquer algorithms.

AUTOGEN-WAVE. AUTOGEN-WAVE [Chowdhury et al., 2017b] is a framework for computer-assisted discovery of fast divide-and-conquer wavefront versions of the algorithms already generated by AUTOGEN. Standard cache-oblivious recursive divide-and-conquer algorithms have optimal serial cache complexity but often have low parallelism due to artificial dependencies among subtasks. AUTOGEN-WAVE can be used to systematically transform standard cache-oblivious recursive divide-and-conquer algorithms into recursive wavefront algorithms to achieve optimal parallel cache complexity and high parallelism under state-of-the-art schedulers for fork-join programs.

The divide-and-conquer wavefront algorithms use closed-form formulas to compute at what time each divide-and-conquer function must be launched in order to achieve high parallelism without losing cache performance. The resulting implementations are arguably much simpler than implementations of known cache-oblivious wavefront algorithms. We present theoretical analyses and experimental performance and scalability results showing a superiority of these new algorithms over existing algorithms.

Cache-oblivious parallel Viterbi algorithm. As a first step toward extending AUTOGEN to handle DP recurrences with irregular data-dependent dependencies, we design an efficient cache-oblivious parallel algorithm to solve the Viterbi recurrence. The Viterbi algorithm is used to find the most likely path through a hidden Markov model given an observed sequence and has nu-

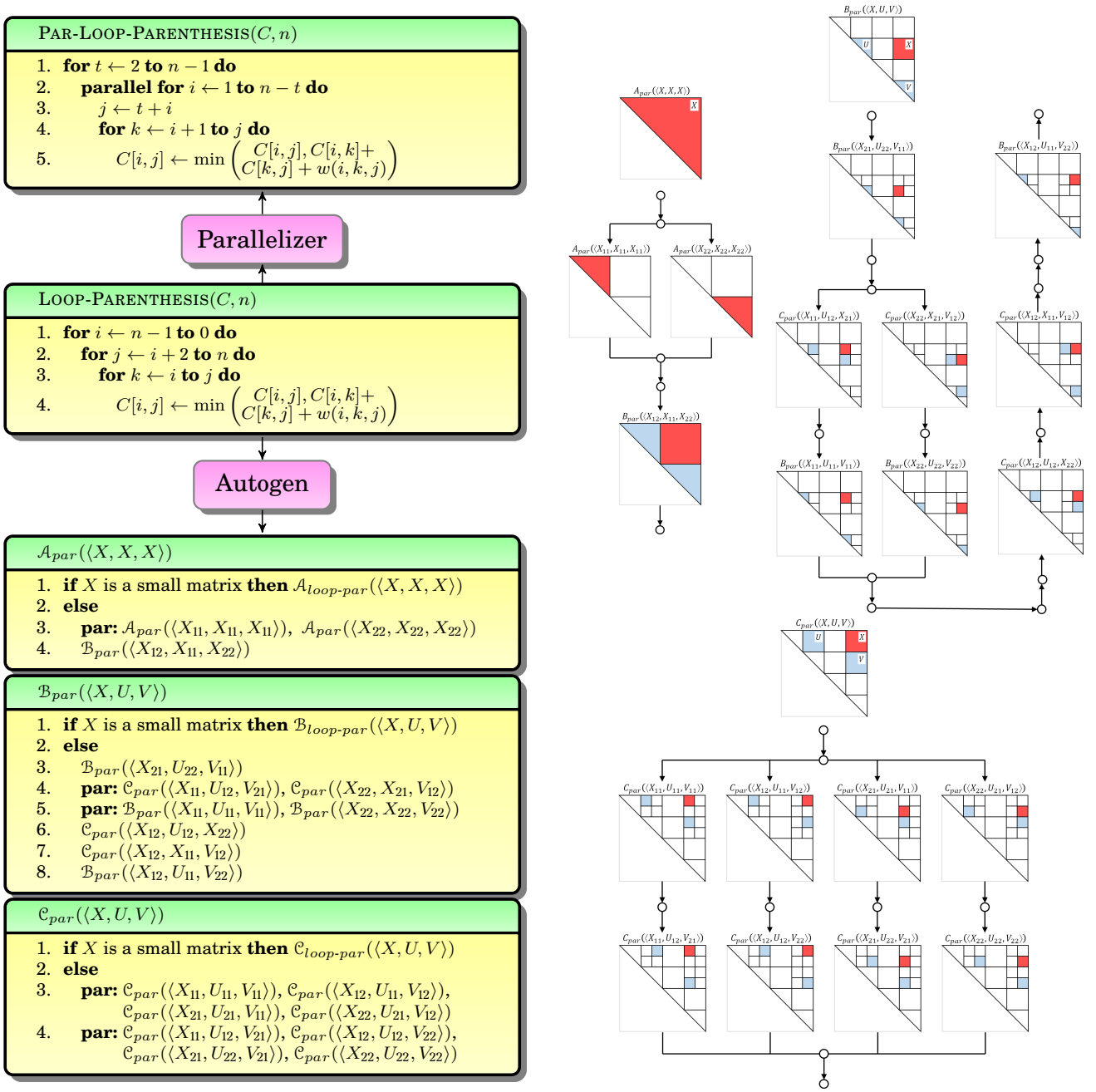

Figure 1: Left upper half: A parallel looping code that for the parenthesis algorithm. Left lower half: AUTOGEN takes the serial parenthesis algorithm as input and automatically discovers a recursive divide-and-conquer cache-oblivious parallel algorithm. Initial call to the divide-and-conquer algorithm is $\mathcal{A}_{par}(\langle C,C,C\rangle)$, where C is an $n \times n$ DP table and n is a power of 2. The iterative base-case kernel of a function $\mathcal{F}_{par}$ is $\mathcal{F}_{loop\text{-}par}$. Right: Pictorial representation of the recursive divide-and-conquer algorithm discovered by AUTOGEN. Data in the dark red blocks are updated using data from light blue blocks.

merous applications. Existing algorithms for the Viterbi decoding problem are not simultaneously cache-efficient and cache-oblivious.

We design our efficient cache-oblivious parallel Viterbi algorithm [Chowdhury et al., 2016a] for a single instance by combining our efficient cache-oblivious Viterbi algorithm for multiple instances with the cache-inefficient parallel Viterbi algorithm of Maleki et al. [Maleki et al., 2016] based on rank convergence. We provide an empirical analysis of our algorithm by comparing it with Maleki

Problem	Work (T_1)	Iterative algorithm: Serial cache comp. (Q_1)	Iterative algorithm: Span (T_∞)	Iterative algorithm: Parallelism (T_1/T_∞)	Recursive algorithm: Serial cache comp. (Q_1)	Recursive algorithm: Span (T_∞)	Recursive algorithm: Parallelism (T_1/T_∞)
Parenthesis problem	$\Theta(n^3)$	$\Theta(n^3)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta\left(n^3/(B\sqrt{M})\right)$	$\Theta\left(n^{\log 3}\right)$	$\Theta\left(n^{3-\log 3}\right)$
Floyd-Warshall's APSP 2-D	$\Theta(n^3)$	$\Theta(n^3/B)$	$\Theta(n\log n)$	$\Theta(n^2/\log n)$	$\Theta\left(n^3/(B\sqrt{M})\right)$	$\Theta\left(n\log^2 n\right)$	$\Theta\left(n^2/\log^2 n\right)$
LCS / Edit distance	$\Theta(n^2)$	$\Theta(n^2/B)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n^2/(BM))$	$\Theta\left(n^{\log 3}\right)$	$\Theta\left(n^{2-\log 3}\right)$
Multi-instance Viterbi	$\Theta(n^3t)$	$\Theta(n^3t/B)$	$\Theta(nt)$	$\Theta(n^2)$	$\Theta\left(n^3t/(B\sqrt{M})\right)$	$\Theta(nt)$	$\Theta(n^2)$
Gap problem	$\Theta(n^3)$	$\Theta(n^3)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta\left(n^3/(B\sqrt{M})\right)$	$\Theta\left(n^{\log 3}\right)$	$\Theta\left(n^{3-\log 3}\right)$
Protein accordion folding	$\Theta(n^3)$	$\Theta(n^3/B)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta\left(n^3/(B\sqrt{M})\right)$	$\Theta(n\log n)$	$\Theta(n^2/\log n)$
Spoken-word recognition	$\Theta(n^2)$	$\Theta(n^2/B)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n^2/(BM))$	$\Theta\left(n^{\log 3}\right)$	$\Theta\left(n^{2-\log 3}\right)$
Function approximation	$\Theta(n^3)$	$\Theta(n^3/B)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta\left(n^3/(B\sqrt{M})\right)$	$\Theta\left(n^{\log 3}\right)$	$\Theta\left(n^{3-\log 3}\right)$
Binomial coefficient	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2/B)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n^2/(BM))$	$\Theta\left(n^{\log 3}\right)$	$\Theta\left(n^{2-\log 3}\right)$
Bitonic traveling salesman	$\Theta(n^2)$	$\Theta(n^2/B)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n^2/(BM))$	$\Theta(n\log n)$	$\Theta(n/\log n)$
Matrix multiplication	$\Theta(n^3)$	$\Theta(n^3/B)$	$\Theta(n)$	$\Theta(n^2)$	$\Theta\left(n^3/(B\sqrt{M})\right)$	$\Theta(n)$	$\Theta(n^2)$
Bubble sort	$\Theta(n^2)$	$\Theta(n^2/B)$	$\Theta(n^2)$	$\Theta(1)$	$\Theta(n^2/(BM))$	$\Theta(n)$	$\Theta(n)$
Selection sort	$\Theta(n^2)$	$\Theta(n^2/B)$	$\Theta(n^2)$	$\Theta(1)$	$\Theta(n^2/(BM))$	$\Theta(n)$	$\Theta(n)$
Insertion sort	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2/B)$	$\mathcal{O}(n^2)$	$\Theta(1)$	$\mathcal{O}\left(n^{\log 3}/(BM^{\log 3-1})\right)$	$\mathcal{O}(n)$	$\Omega(n)$

Table 1: Work (T_1), serial cache complexity (Q_1), span (T_∞), and parallelism (T_1/T_∞) of serial and recursive algorithms for several DP problems. Here, n = problem size, M = cache size, B = block size, and p = #cores. By T_p we denote running time on p processing cores. We assume that the DP table is too large to fit into the cache, and $M = \Omega\left(B^d\right)$ when $\Theta\left(n^d\right)$ is the size of the DP table. On p cores, the running time is $T_p = \mathcal{O}\left(T_1/p + T_\infty\right)$ and the parallel cache complexity is $Q_p = \mathcal{O}\left(Q_1 + p(M/B)T_\infty\right)$ with high probability when run under the randomized work-stealing scheduler on a parallel machine with private caches. The problems in the lower section are non-DP problems. For insertion sort, T_1 for recursive algorithm is $\mathcal{O}\left(n^{\log 3}\right)$.

et al.'s algorithm. To the best of our knowledge, this is the first work that presents a provably cache-efficient cache-oblivious parallel Viterbi algorithm.

AUTOGEN-FRACTILE. AUTOGEN-FRACTILE [Chowdhury et al., 2017a] is a framework for computer-assisted discovery of fast external-memory GPU algorithms based on recursive tiling. The approaches used in the existing GPU implementations, most of them tiled-iterative-based, suffer from several limitations: the approaches use different ways of tiling and it is nontrivial to choose the one that leads to the best performance, the approaches might lead to asymptotically non-optimal number of data transfers, it is difficult to extend the existing algorithms to multiple GPUs and external memory, and the existing approaches typically do not provide strong theoretical performance guarantees for I/O complexity and span. The proposed framework solves all the aforementioned problems.

Our GPU implementations run much faster than their CPU counterparts.

References

[Chowdhury et al., 2017a] Chowdhury, R., Das, R., Ganapathi, P., Javanmard, M. M., and Tschudi, S. (2017a). A framework for designing external-memory RAM-oblivious GPU algorithms using recursive tiling. In Under preparation.

[Chowdhury et al., 2016a] Chowdhury, R., Ganapathi, P., Pradhan, V., Tithi, J. J., and Xiao, Y. (2016a). An efficient cache-oblivious parallel Viterbi algorithm. In Euro-Par.

[Chowdhury et al., 2017b] Chowdhury, R., Ganapathi, P., Tang, Y., and Tithi, J. J. (2017b). Provably efficient scheduling of cache-oblivious wavefront algorithms. In Under review.

[Chowdhury et al., 2016b] Chowdhury, R. A., Ganapathi, P., Tithi, J. J., Bachmeier, C., Kuszmaul, B. C., Charles E. Leiserson, A. S.-L., and Tang, Y. (2016b). AutoGen: Automatic discovery of cache-oblivious parallel recursive algorithms for solving dynamic programs. In PPoPP, page 10.

[Galil and Park, 1994] Galil, Z. and Park, K. (1994). Parallel algorithms for dynamic programming recurrences with more than o(1) dependency. JPDC, 21(2):213–222.

[Maleki et al., 2016] Maleki, S., Musuvathi, M., and Mytkowicz, T. (2016). Low-rank methods for parallelizing dynamic programming algorithms. ACM TOPC, 2(4):26.