Brief Announcement: STAR (Space-Time Adaptive and Reductive) Algorithms for Dynamic Programming Recurrences with more than O(1) Dependency

Yuan Tang, Shiyi Wang*
School of Software, Fudan University
Shanghai, China
yuantang@fudan.edu.cn

ABSTRACT

It's important to hit a space-time balance for a real-world algorithm to achieve high performance on modern shared-memory multi-core and many-core systems. However, a large class of dynamic programs with more than $O(1)$ dependency achieved optimality either in space or time, but not both. In the literature, the problem is known as the fundamental space-time tradeoff. We propose the notion of "Processor-Adaptiveness". In contrast to the prior "Processor-Awareness", our approach does not partition statically the problem space to the processor grid, but uses the processor count P to just upper bound the space and cache requirement in a cache-oblivious fashion. In the meantime, our processor-adaptive algorithms enjoy the full benefits of "dynamic load-balance", which is a key to achieve satisfactory speedup on a shared-memory system, especially when the problem dimension n is reasonably larger than P. By utilizing the "busy-leaves" property of runtime scheduler and a program managed memory pool that combines the advantages of stack and heap, we show that our STAR (Space-Time Adaptive and Reductive) technique can help these dynamic programs to achieving sublinear time bounds while keeping to be asymptotically work-, space-, and cache-optimal. The ***key achievement*** of this paper is to obtain the first sublinear $O(n^{3/4} \log n)$ time and optimal $O(n^3)$ work GAP algorithm; If we further bound the space and cache requirement of the algorithm to be asymptotically optimal, there will be a factor of P increase in time bound without sacrificing the work bound. If $P = o(n^{1/4}/\log n)$, the time bound stays sublinear and may be a better tradeoff between time and space requirements in practice.

CCS CONCEPTS

•Theory of computation →Divide and conquer; Dynamic programming; •Computing methodologies →Shared memory algorithms;

KEYWORDS

space-time balance, cache-oblivious algorithm, matrix multiplication, dynamic program with more than $O(1)$ dependency, work-time model, the shared-memory multicore system

1 INTRODUCTION

It's important to hit a space-time balance for a real-world algorithm to achieve high performance on modern shared-memory multi-core and many-core systems. However, a large class of DP (Dynamic Programming) recurrences with more than $O(1)$ dependency, including the general MM (matrix multiplication), Strassen-like fast MM, LWS, GAP, and Parenthesis, have algorithms with either sublinear (parallel) time bound (time bound for short) [1] but sub-optimal space and cache bound [10], or optimal space and cache but super-linear time bound [4, 6, 8]. To the best of our knowledge, there are no prior approach that can simultaneously achieve a sublinear time as well as optimal work, space and cache bound in one algorithm for DP recurrences with more than $O(1)$ dependency, especially in a cache-oblivious fashion.

Figure 1: Acronyms and notations

MM	Matrix Multiplication
DP	Dynamic Programming
COP	Cache-Oblivious Parallel
RWS	Randomized Work-Stealing
CAS	Compare-And-Swap
n	Problem dimension
P	Number of processing cores
ϵ_i	small constant
M	Cache size
B	cache line size
T_1	Work
T_∞	Time (span, depth, critical path length)
T_p	Parallel running time on p cores
T_1/T_∞	Parallelism
Q_1	Serial cache complexity
Q_p	Parallel cache complexity on P threads
ND	Nested Dataflow
$a \parallel b$	task b has *no* dependency on a
$a \; ; \; b$	task b has *full* dependency on a
$a \rightsquigarrow b$	task b has *partial* dependency on a

Let's take the general MM $C = A \otimes B$ on a closed semiring $SR = (S, \oplus, \otimes, 0, 1)$ as an example, where S is a set of elements, $\oplus$ and $\otimes$ are binary operations on S, and 0, 1 are additive and multiplicative identities, respectively. The general MM not only is a DP problem with $O(n)$ dependency [2], but also serves as a basic building block for more complicated DP algorithms such as LWS, GAP, and Parenthesis to achieve sublinear time bounds [10]. The general MM can be computed in a recursive divide-and-conquer fashion as follows. At each level of recursion, the computation of an MM of dimension n (i.e. n-by-n) is divided into four equally sized quadrants, which require updates from eight

*Yuan Tang is the corresponding author and also affiliated with Shanghai Key Lab. of Intelligent Information Processing and the State Key Lab. of Comp. Arch. ICT, CAS

[1]If we view a parallel computation as a DAG, the time bound T_∞ denotes the critical path length.
[2]The update of each cell of the output matrix requires $O(n)$ reads and computation from the two input matrices

sub-MMs of dimension $n/2$ as shown in Equation (1).

$$\begin{aligned}\begin{bmatrix}C_{00} & C_{01}\\ C_{10} & C_{11}\end{bmatrix} &= \begin{bmatrix}A_{00} & A_{01}\\ A_{10} & A_{11}\end{bmatrix} \otimes \begin{bmatrix}B_{00} & B_{01}\\ B_{10} & B_{11}\end{bmatrix}\\ &= \begin{bmatrix}A_{00}\otimes B_{00} & A_{00}\otimes B_{01}\\ A_{10}\otimes B_{00} & A_{10}\otimes B_{01}\end{bmatrix} \oplus \begin{bmatrix}A_{01}\otimes B_{10} & A_{01}\otimes B_{11}\\ A_{11}\otimes B_{10} & A_{11}\otimes B_{11}\end{bmatrix}\end{aligned} \tag{1}$$

Depending on the availability of extra space, the computation of the eight sub-MMs can be scheduled to run either completely in parallel (Figure 2a) or in two parallel steps (Figure 2b). The Figures 2a and 2b show the two algorithms. More sophisticated approaches are feasible in the literature and will be discussed in Section 2.

We can calculate the time and space bounds of the two algorithms by the recurrences of Equations (2), (3), and (4). The MM-n^2-SPACE algorithm (Figure 2b) uses no extra space than the input and output matrices so there is no recurrece for its space requirement. We can see that the MM-n^3-SPACE algorithm (Figure 2a) has an optimal $O(\log n)$ time bound if counting only the data dependency but a poor $O(n^3)$ space bound; By contrast, the MM-n^2-SPACE algorithm has an optimal $O(n^2)$ space bound, but a sub-optimal $O(n)$ time bound. We care about an algorithm's space bound not only because operating system will disable a computation from executing if it exceeds the space quota, but also because it's a good indicator of cache bound. The cache bound characterizes the amount of data movement (communication) between levels of cache hierarchy throughout the computation. On modern computing system with a hierarchy of caches, data movement usually has a heavier unit weight than arithmetic operations, thus has more impact on the overall performance. By a similar recurrence calculation, we can see that the MM-n^3-SPACE algorithm has a sub-optimal $O(n^3/B)$ serial cache bound [3], in contrast to the optimal $O(n^3/(B\sqrt{M}))$ bound of the MM-n^2-SPACE algorithm. In the literature, it is known as the fundamental space-time tradeoff.

$$T_{\infty,\text{MM-}n^3\text{-SPACE}}(n) = T_{\infty,\text{MM-}n^3\text{-SPACE}}(n/2) + T_{\infty,\text{MADD}}(n) \tag{2}$$

$$S_{\text{MM-}n^3\text{-SPACE}}(n) = 8S_{\text{MM-}n^3\text{-SPACE}}(n/2) + n^2 \tag{3}$$

$$T_{\infty,\text{MM-}n^2\text{-SPACE}}(n) = 2T_{\infty,\text{MM-}n^2\text{-SPACE}}(n/2) \tag{4}$$

A real-world MM algorithm may employ some tuning technique (e.g. 2.5D MM algorithm [16]) to go somewhere in the middle ground of the two extremes. However, an interesting research question is if it is possible to achieve a sublinear time bound, while in the meantime keeping to be asymptotically work-, space-, and cache-optimal.

Our Contributions

- ***Key Achievement:*** We solve an open problem raised in Galil and Park's paper [10] more than 20 years ago. That is, we have the first sublinear $O(n^{3/4}\log n)$ time and optimal $O(n^3)$ work GAP algorithm. If we further bound the space and cache requirement of the algorithm to be asymptotically optimal, i.e. $O(n^2)$ and $O(n^3/(B\sqrt{M}))$ respectively, there will be a factor of P increase in time bound without sacrificing the work bound. If $P = o(n^{1/4}/\log n)$, the time bound stays sublinear and may be a better tradeoff between time and space in practice.
- We propose the notion of "***Processor-Adaptiveness***". In contrast to the prior "Processor-Awareness", our approach does not partition statically the problem space to the processor grid, but uses the processor count P to just upper bound the space and cache requirement in a cache-oblivious fashion. Moreover, our processor-adaptive approach enjoys the full benefits of "*dynamic load-balance*", which is a key to achieving satisfactory speedup on a shared-memory multi-core and many-core system, especially when the problem dimension n is reasonably larger than P. We argue that taking the processor count P as a parameter to algorithm design and implementation is easy and straightforward in most state-of-the-art multi-threaded programming languages such as Cilk and OpenMP. Moreover, the parameter P does not require any tuning during the computation in contrast to the cache parameters.
- By utilizing the "busy-leaves" property of the runtime scheduler, we can bound the space requirement of our STAR (Space-Time Adaptive and Reductive) algorithms to be asymptotically optimal; By a program-managed memory pool that combines the advantages of stack and heap, our STAR technique can further bound the serial cache misses to be asymptotically optimal in a cache-oblivious fashion.
- We show by experiments that our STAR algorithms do improve the parallel cache misses due to a better time bound and can outperform the classic cache-oblivious parallel algorithms when the problem dimension is reasonably large, especially when the application is more "memory-intensive" than "computation-intensive", provided that the same kernel function is employed to compute the same-sized base cases.

[3]The parallel cache complexity is determined in large by the runtime scheduler and is proportional to the serial cache bound.

2 RELATED WORKS

Galil and Park [9, 10] proposed to solve the dynamic programming recurrences by the methods of matrix closure, matrix product, and indirection. Their work is a great motivation for this paper. By their methods, they achieved sublinear parallel time bounds algorithms for a large class of dynamic programming recurrences with more than $O(1)$ dependency. Their work, however, doesn't consider the space or cache requirements. Moreover, their GAP algorithm is not work-optimal.

Hybrid r-way divide-and-conquer algorithms with different values of r at different levels of recursion have been considered in by Chowdhury et al [5]. These algorithms can reach parallel cache complexity matching the best serial cache bounds. Their approach are processor-aware.

Tang et al. [17] proposed Eager and Lazy cache-oblivious wavefront (COW) technique to strike the best of cache complexity and parallelism for a large class of dynamic programming problems. Dinh et al. [7] extended the research to the ND (Nested Dataflow) parallel programming model.

The optimizations, as well as various tradeoffs among work, space, time, communication bounds, on the general matrix multiplication on a semiring or Strassen-like fast algorithm has been

```
MM-$n^3$-SPACE($C$, $A$, $B$)
// $C \leftarrow A \times B$
if (sizeof($C$) $\leq$ BASE_SIZE)
    BASE-KERNEL($C$, $A$, $B$)
    return
$D \leftarrow$ alloc(sizeof($C$))
// Run all 8 sub-MMs concurrently
MM-$n^3$-SPACE($C_{00}$, $A_{00}$, $B_{00}$) || MM-$n^3$-SPACE($C_{01}$, $A_{00}$, $B_{01}$)
|| MM-$n^3$-SPACE($C_{10}$, $A_{10}$, $B_{00}$) || MM-$n^3$-SPACE($C_{11}$, $A_{10}$, $B_{01}$)
|| MM-$n^3$-SPACE($D_{00}$, $A_{01}$, $B_{10}$) || MM-$n^3$-SPACE($D_{01}$, $A_{01}$, $B_{11}$)
|| MM-$n^3$-SPACE($D_{10}$, $A_{11}$, $B_{10}$) || MM-$n^3$-SPACE($D_{11}$, $A_{11}$, $B_{11}$)
; // sync
// Merge matrix $D$ into $C$ by addition
madd($C$, $D$)
free ($D$)
return
```

(a) The $O(n^3)$ space recursive MM algorithm

```
MM-$n^2$-SPACE($C$, $A$, $B$)
// $C \leftarrow A \times B$
if (sizeof($C$) $\leq$ BASE_SIZE)
    BASE-KERNEL($C$, $A$, $B$)
    return
// Run the first 4 sub-MMs concurrently
MM-$n^2$-SPACE($C_{00}$, $A_{00}$, $B_{00}$) || MM-$n^2$-SPACE($C_{01}$, $A_{00}$, $B_{01}$)
|| MM-$n^2$-SPACE($C_{10}$, $A_{10}$, $B_{00}$) || MM-$n^2$-SPACE($C_{11}$, $A_{10}$, $B_{01}$)
; // sync
// Run the next 4 sub-MMs concurrently
MM-$n^2$-SPACE($C_{00}$, $A_{01}$, $B_{10}$) || MM-$n^2$-SPACE($C_{01}$, $A_{01}$, $B_{11}$)
|| MM-$n^2$-SPACE($C_{10}$, $A_{11}$, $B_{10}$) || MM-$n^2$-SPACE($C_{11}$, $A_{11}$, $B_{11}$)
; // sync
return
```

(b) The $O(n^2)$ space recursive MM algorithm

Figure 2: Recursive Divide-And-Conquer MM algorithms. "||" and ";" are symbols of linguistic constructs of the ND (Nested Dataflow) parallel programming model [7] (Figure 1).

studied for decades, including at least [1–3, 11–13, 15, 16]. The basic idea of these prior works on the tradeoffs between work, space, time and / or communication overheads for general matrix multiplication or Strassen-like fast algorithms is to switch manually back and forth between the serial algorithm to save and reuse space and the parallel algorithm to increase the parallelism. There are some differences between these prior works and the STAR techniques. First, our work focus on the shared-memory multicore architecture, on which the dynamic load-balance is a key to achieve satisfactory speedup, especially when the problem dimension n is reasonably large compared to the processor count P; By utilizing the "busy-leaves" property of the runtime scheduler, our STAR technique can upper bound the space requirement to be optimal without tuning; By a program managed memory pool, we combine the advantages of stack and heap, thus bound the serial cache-optimality; By having a sublinear critical path length, we reduce asymptotically the parallel cache misses.

Shun et al. [14] alleviates the problem of "concurrent writes" to the same memory location by prioritizing the operations, thus reduce the number of writes. However, not all operations can be prioritized and reduced such as those for the general matrix multiplication.

ACKNOWLEDGMENTS

We gratefully thank the PACMAN group at Tsinghua University for providing some equipment for experiments. This research is supported in part by the Open Funding of State Key Laboratory of Computer Architecture, ICT, CAS (No. CARCH201606).

REFERENCES

[1] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. Communication-optimal parallel algorithm for strassen's matrix multiplication. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '12, pages 193–204, New York, NY, USA, 2012. ACM.

[2] A. R. Benson and G. Ballard. A framework for practical parallel fast matrix multiplication. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP 2015, pages 42–53, New York, NY, USA, 2015. ACM.

[3] B. Boyer, J.-G. Dumas, C. Pernet, and W. Zhou. Memory efficient scheduling of strassen-winograd's matrix multiplication algorithm. In *Proceedings of the 2009 International Symposium on Symbolic and Algebraic Computation*, ISSAC '09, pages 55–62, New York, NY, USA, 2009. ACM.

[4] R. Chowdhury. *Cache-efficient Algorithms and Data Structures: Theory and Experimental Evaluation.* PhD thesis, Department of Computer Sciences, The University of Texas at Austin, Austin, Texas, 2007.

[5] R. Chowdhury and V. Ramachandran. Cache-efficient Dynamic Programming Algorithms for Multicores. In *Proceedings of ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 207–216, 2008.

[6] R. Chowdhury and V. Ramachandran. The cache-oblivious Gaussian elimination paradigm: Theoretical framework, parallelization and experimental evaluation. *Theory of Computing Systems*, 47(4):878–919, 2010.

[7] D. Dinh, H. V. Simhadri, and Y. Tang. Extending the nested parallel model to the nested dataflow model with provably efficient schedulers. In *SPAA'16*, Pacific Grove, CA, USA, 11 – 13 2016.

[8] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. *ACM Trans. Algorithms*, 8(1):4:1–4:22, Jan. 2012.

[9] Z. Galil and R. Giancarlo. Speeding up dynamic programming with applications to molecular biology. *Theoretical Computer Science*, 64:107–118, 1989.

[10] Z. Galil and K. Park. Parallel algorithms for dynamic programming recurrences with more than $O(1)$ dependency. *Journal of Parallel and Distributed Computing*, 21:213–222, 1994.

[11] J. Huang, T. M. Smith, G. M. Henry, and R. A. van de Geijn. Implementing strassen's algorithm with blis. *CoRR*, 2016.

[12] B. Kumar, C. Huang, P. Sadayappan, and R. W. Johnson. A tensor product formulation of strassen's matrix multiplication algorithm with memory reduction. *Scientific Programming*, 4(4):275–289, 1995.

[13] F. W. McColl and A. Tiskin. Memory-efficient matrix multiplication in the bsp model. *Algorithmica*, 24(3):287–297, 1999.

[14] J. Shun, G. E. Blelloch, J. T. Fineman, and P. B. Gibbons. Reducing contention through priority updates. In *SPAA*, pages 152–163, 2013.

[15] T. M. Smith, R. v. d. Geijn, M. Smelyanskiy, J. R. Hammond, and F. G. V. Zee. Anatomy of high-performance many-threaded matrix multiplication. In *Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium*, IPDPS '14, pages 1049–1059, Washington, DC, USA, 2014. IEEE Computer Society.

[16] E. Solomonik and J. Demmel. Communication-optimal parallel 2.5d matrix multiplication and lu factorization algorithms. In *Proceedings of the 17th International Conference on Parallel Processing - Volume Part II*, Euro-Par'11, pages 90–109, Berlin, Heidelberg, 2011. Springer-Verlag.

[17] Y. Tang, R. You, H. Kan, J. J. Tithi, P. Ganapathi, and R. A. Chowdhury. Cache-oblivious wavefront: Improving parallelism of recursive dynamic programming algorithms without losing cache-efficiency. In *PPoPP'15*, San Francisco, CA, USA, Feb.7 – 11 2015.