POSTER: STAR (Space-Time Adaptive and Reductive) Algorithms for Real-World Space-Time Optimality

Yuan Tang [*], Ronghui You

School of Computer Science, School of Software, Fudan University
Shanghai Key Lab. of Intelligent Information Processing
State Key Lab. of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
P. R. China
[yuantang, 15210240027]@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 or many-core systems. However, a large class of dynamic programs with more than $O(1)$ dependency achieve optimality either in space or time, but not both. In the literature, the problem is known as the fundamental space-time tradeoff. By exploiting properly on the runtime system, we show that our STAR (Space-Time Adaptive and Reductive) technique can help these dynamic programs to achieve sublinear parallel time bounds while still maintaining work-, space-, and cache-optimality in a processor- and cache-oblivious fashion.

Keywords space-time balance, cache-oblivious algorithm, dynamic program, 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 or many-core systems. However, a large class of DP (Dynamic Programming) algorithms with more than $O(1)$ dependency, including the general MM (matrix multiplication), Strassen-like fast MM, LWS, GAP, and Parenthesis [2], achieve optimality either in space or time, but not both.

Let's take the general MM on a closed semiring as an example. The general MM not only is itself a dynamic programming algorithm with $O(n)$ dependency since the full update of each cell of the output matrix requires $O(n)$ reads and computation from the two input matrices, but also serves as a basic building block for more complicated DP algorithms such as LWS, GAP, and Parenthesis to achieve sublinear parallel time bounds (or time bounds for short) [2] [1]. 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 sub-MMs of dimension $n/2$ as shown in the Equation (1). Depending on the availability of extra space,

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

the computation of the eight sub-MMs can be scheduled to run either completely in parallel (Figure 1a) or in two parallel steps (Figure 1b).

We can calculate the time and space bounds of the two algorithms by recurrences and see that the MM-n^3-SPACE algorithm (Figure 1a) has an optimal time bound of $O(\log n)$ if counting only the data dependency and a poor space bound of $O(n^3)$; By contrast, the MM-n^2-SPACE algorithm (Figure 1b) has a an optimal space bound of $O(n^2)$, but a sub-optimal time bound of $O(n)$. In the literature, it is known as the fundamental space-time tradeoff. A real-world MM algorithm usually employs some tuning technique (e.g. 2.5D MM algorithm) to go somewhere in the middle ground of the two extremes. However, one interesting research question is if it is possible to achieve a sublinear time bound while still

[*] This work is supported in part by the Open Funding of State Key Laboratory of Computer Architecture, ICT, CAS (No. CARCH201606)

PPoPP '17 February 04-08, 2017, Austin, TX, USA
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-4493-7/17/02...$15.00
DOI: http://dx.doi.org/10.1145/3018743.3019029

[1] This paper uses the notion "time bound" to stand for the running time of a parallel algorithm on infinite number of processors, or equivalently the critical path length of the computation DAG, and uses the notion "work" to stand for the running time of the same algorithm on a single processor, or equivalently the total number of vertices of the computation DAG.

```
MM-$n^3$-SPACE($C$, $A$, $B$)
    // $C \leftarrow A \times B$
    if ($C$ is small enough)
        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 matrices $C$ and $D$ 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 ($C$ is small enough)
        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 1: Recursive Divide-And-Conquer MM algorithms. "||" (Parallel) and ";" (Serial) are symbols for the linguistic constructs of the ND (Nested Dataflow) parallel programming model [1].

bounding the total space requirement to be asymptotically optimal.

We care about an algorithm's space bound not only because the 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, an even more important factor than the computational bound in deciding an algorithm's real performance on any modern computing system with hierarchical caches. By a similar recurrences analysis, we can see that the MM-n^3-SPACE algorithm has a sub-optimal $O(n^3/B)$ cache miss bound [2], in contrast to the optimal $O(n^3/(B\sqrt{M}))$ bound of the MM-n^2-SPACE algorithm. These cache bounds match proportionally to their space bounds. But we have to remind the audience that an algorithm can repeatedly allocate and free space throughout its computation to bound the total space to be some optimal number. This strategy will inevitably incur a lot of cold cache misses. In other words, a good space bound does **not** necessary imply a good cache bound. Thus, we have to calculate both bounds.

Contributions

- We propose the property of "remote-blocking". If a runtime system stands by the property, we show that our STAR (Space-Time Adaptive and Reductive) technique can bound the total space requirement of a large class of DP algorithms with more than $O(1)$ dependency to be asymptotically optimal while still being work-optimal and having a sub-linear time bound.
- We propose a parallel stack-like memory management system for large chunks of space in a multi-threaded setting. By the system, our STAR technique can bound the cache misses to be asymptotically optimal as well in a processor- and cache-oblivious fashion.

  Though our STAR technique has the processor count "P" in the algorithm design, we use this parameter only to bound the total space requirement and cache misses. Our algorithms do **not** partition "statically" according to the parameter, thus still have the full benefits of "scalable to any number of processors without changing a single line of the algorithm" , "dynamic multi-threading" and "dynamic load-balance" as any processor- and cache-oblivious algorithm.
- By combining the STAR and the cache-oblivious wavefront technique [3], or the ND (nested dataflow) in general, we solve one open problem raised in Galil and Park's paper [2]. More precisely, we have a better GAP algorithm with an optimal $O(n^3)$ work, a sublinear $O(P^{1/2}n^{3/4}\log n)$ time bound conditional on that the processor count $P = o(n^{1/2}/\log^2 n)$, an optimal $O(n^2)$ space and optimal $O(n^3/(B\sqrt{M}))$ cache bound.

[2] This paper calculates only the serial cache miss bound under the ideal cache model since the parallel cache complexity is determined in large by the runtime scheduler.

References

[1] 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.

[2] 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.

[3] 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.