Closing the Gap Between Cache-oblivious and Cache-adaptive Analysis

Michael A. Bender
Rezaul A. Chowdhury
Rathish Das
Stony Brook University
Stony Brook, NY, USA
{bender,rezaul,radas}@cs.stonybrook.edu

Rob Johnson
VMware Research
Palo Alto, CA, USA
robj@vmware.com

William Kuszmaul
Andrea Lincoln
Quanquan C. Liu
Jayson Lynch
Helen Xu
MIT CSAIL
Cambridge, MA, USA
{kuszmaul,andreali,quanquan}@mit.edu
{jaysonl,hjxu}@mit.edu

ABSTRACT

Cache-adaptive analysis was introduced to analyze the performance of an algorithm when the cache (or internal memory) available to the algorithm dynamically changes size. These memory-size fluctuations are, in fact, the common case in multi-core machines, where threads share cache and RAM. An algorithm is said to be efficiently cache-adaptive if it achieves optimal utilization of the dynamically changing cache.

Cache-adaptive analysis was inspired by cache-oblivious analysis. Many (or even most) optimal cache-oblivious algorithms have an (a, b, c)-regular recursive structure. Such (a, b, c)-regular algorithms include Longest Common Subsequence, All Pairs Shortest Paths, Matrix Multiplication, Edit Distance, Gaussian Elimination Paradigm, etc. Bender et al. (2016) showed that some of these optimal cache-oblivious algorithms remain optimal even when cache changes size dynamically, but that in general they can be as much as logarithmic factor away from optimal. However, their analysis depends on constructing a highly structured, worst-case memory profile, or sequences of fluctuations in cache size. These worst-case profiles seem fragile, suggesting that the logarithmic gap may be an artifact of an unrealistically powerful adversary.

We close the gap between cache-oblivious and cache-adaptive analysis by showing how to make a smoothed analysis of cache-adaptive algorithms via random reshuffling of memory fluctuations. Remarkably, we also show the limits of several natural forms of smoothing, including random perturbations of the cache size and randomizing the algorithm's starting time. Nonetheless, we show that if one takes an arbitrary profile and performs a random shuffle on when "significant events" occur within the profile, then the shuffled profile becomes optimally cache-adaptive in expectation, even when the initial profile is adversarially constructed.

These results suggest that cache-obliviousness is a solid foundation for achieving cache-adaptivity when the memory profile is not overly tailored to the algorithm structure.

CCS CONCEPTS

• **Theory of computation → Caching and paging algorithms**; **Parallel algorithms**.

KEYWORDS

Cache-adaptive algorithms; smoothed analysis; cache-oblivious algorithms

ACM Reference Format:
Michael A. Bender, Rezaul A. Chowdhury, Rathish Das, Rob Johnson, William Kuszmaul, Andrea Lincoln, Quanquan C. Liu, Jayson Lynch, and Helen Xu. 2022. Closing the Gap Between Cache-oblivious and Cache-adaptive Analysis. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 30 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

On multi-threaded and multi-core systems, the amount of cache available to any single process constantly varies over time as other processes start, stop, and change their demands for cache. On most multi-core systems, each core has a private cache and the entire system has a cache shared between cores. A program's fraction of the private cache of a core can change because of time-sharing, and its fraction of the shared cache can change because multiple cores use it simultaneously [14, 23, 24].

Cache-size changes can be substantial. For example, there is frequently a winner-take-all phenomenon, in which one process grows to monopolize the available cache [25]; researchers have suggested periodically flushing the cache to counteract this effect [57]. With this policy, individual processes would experience cache allocations that slowly grow to the maximum possible size, then abruptly crash down to 0.

Furthermore, even small cache-size changes can have catastrophic effects on the performance of algorithms that are not designed to handle them. When the size of cache shrinks unexpectedly, an algorithm tuned for a fixed-size cache can thrash, causing its performance to drop by orders of magnitude. (And if the cache grows, an algorithm that assumes a fixed cache size can leave performance on the table.)

This is such an important problem that many systems provide mechanisms to manually control the allocation of cache to different processes. For example, Intel's Cache Allocation Technology [46] allows the OS to limit each process's share of the shared processor cache. Linux's cgroups mechanism [43] provides control over each application's use of RAM (which serves as a cache for disk). Although these mechanisms can help avoid thrashing, they require manual tuning and can leave cache underutilized. Furthermore, systems may be forced to always leave some cache unused in order to be able to schedule new jobs as they arrive.

A more flexible approach is to solve this problem in the algorithms themselves. If algorithms could gracefully handle changes in their cache allocation, then the system could always fully utilize the cache. Whenever a new task arrives, the system could reclaim some cache from the running tasks and give it to the new task, without causing catastrophic slowdowns of the older tasks. When a task ends, its memory could be distributed among the other tasks on the system. The OS could also freely redistribute cache among tasks to prioritize important tasks.

Practitioners have proposed many algorithms that heuristically adapt to cache fluctuations [13, 31, 44, 45, 47, 48, 64–66]. However, designing algorithms with guarantees under cache fluctuations is challenging and most of these algorithms have poor worst-case performance.

Theoretical approaches to adaptivity. Barve and Vitter [2, 3] initiated the theoretical analysis of algorithms under cache fluctuations over twenty years ago by generalizing the external-memory/disk-access machine (DAM) model [1] to allow the cache size to change. They gave optimal algorithms under memory fluctuations for sorting, FFT, matrix multiplication, LU decomposition, permutation, and buffer trees. In their model, the cache can change size only periodically and algorithms know the future size of the cache and adapt explicitly to these forecasts.

Writing programs and analyzing algorithms that explicitly adapt to changing memory is complicated because the algorithm needs to pay attention to the changing parameter of cache sizes. Moreover, it's hard to have performance guarantees that apply to all possible ways that memory can change size. Thus, most prior work by practitioners is empirical without guarantees, and even the Barve and Vitter work only has guarantees for a restricted class of memory profiles, which limits its generality.

More recently, Bender et al. [5, 6] proposed using techniques from cache-oblivious algorithms to solve the adaptivity problem. Since cache-oblivious algorithms are oblivious to the size of the cache, it is compelling that the algorithms should work well when the cache size changes dynamically. They defined the cache-adaptive model, which is akin to the ideal-cache model [28, 29] from cache-oblivious analysis, except that the size of memory can change arbitrarily over time. They showed that many cache-oblivious algorithms remain optimal even when the size of cache changes dynamically. However, they also showed that some important cache-oblivious algorithms are *not* optimal in the cache-adaptive model.

Concretely, they define optimality in terms of how much progress an algorithm makes under a given ***memory profile*** and they also show that only a restricted class of memory profiles need to be considered. A memory profile $m(t)$ is a function specifying the size of memory at each time t. Prior results show that, for cache-oblivious algorithms, and up to a constant factor of resource augmentation, we need only consider ***square profiles***, i.e., memory profiles which can be decomposed into a sequence of boxes $(\square_1, \square_2, \ldots)$, where a box of size x means that memory remains at size x blocks for x time steps. In its strongest form, cache adaptivity requires that for an algorithm $\mathcal{A}$, the total amount of progress that $\mathcal{A}$ makes on a series of boxes $(\square_1.\square_2, \ldots)$ should be within a constant factor of the total potential $\sum_i \rho(|\square_i|)$ of those boxes, where the potential $\rho(|\square_i|)$ of a box $\square_i$ is defined to be the maximum number of operations that $\mathcal{A}$ could possibly perform in $\square_i$, where the max is taken over all possible places that $\square_i$ could occur in the execution of $\mathcal{A}$.[1]

Thus, Bender et al.'s results show that cache-obliviousness is a powerful technique for achieving adaptivity without the burden of having to explicitly react to cache-size changes. They define optimality in terms of worst-case, adversarial memory profiles, which makes their optimality criteria very strong, but also tough to meet. It's natural to ask how algorithms perform under less adversarial profiles. This is important because for a large class of cache-oblivious algorithms, there exists highly structured and pernicious worst-case profiles on which the algorithms do *not* run optimally.

Cache-oblivious algorithms and (a, b, c)-regularity. One of the fundamental insights in the design of cache-oblivious algorithms is that, by using certain basic recursive structures in the design of an algorithm, one can get optimal cache-obliviousness for free. The algorithms with this recursive structure are known as ***(a, b, c)-regular algorithms***. If an algorithm is (a, b, c)-regular, its I/O-complexity satisfies a recurrence of the form $T(N) = aT(N/b) + \Theta(1 + N^c/B)$,

[1] Several variations on this definition have also been used [5, 6] when considering particular problems (e.g., matrix multiplication). For (a, b, c)-regular algorithms, which are the focus of this paper, the used definitions are equivalent (and thus, as a convention, we use the most general definition).

where B is the block size of the cache and $\Theta(1 + N^c/B)$ represents the cost of scanning an array of size N^c.

For the purposes of cache-adaptivity, the only interesting cases are when $a > b$ and $c \leq 1$.[2] When $a > b$, an algorithm's performance is sensitive to the size of the cache, and so adaptivity becomes important.

If cache-oblivious algorithms were always cache-adaptive, then we could view adaptivity as a solved problem. Unfortunately, this is not the case. Bender et al. showed that, for $a > b$, (a, b, c)-regular algorithms are adaptive if and only if $c < 1$.

The worst-case gap between obliviousness and adaptivity. When $c = 1$, there can be a logarithmic gap between an algorithm's performance in the ideal cache and cache-adaptive models.[3]

Although many classical cache-oblivious algorithms are (a, b, c)-regular [17, 18, 28, 30, 49, 51, 56], many notable algorithms, including cache-oblivious dynamic programming algorithms [17], naive matrix multiplication [28], sub-cubic matrix multiplications (e.g., Strassen's algorithm [55]), and Gaussian elimination [17], have $a > b$ and $c = 1$ and hence fall into this gap. These algorithms are kernels of many other algorithms, including algorithms for solving linear equations in undirected and directed Laplacian matrices (see e.g., [26, 37, 52]), APSP [53, 54, 67], triangle counting [9], min-weight cycle [60], negative triangle detection and counting [60], and the replacement paths problem [60]. Lincoln, et al. [40] show that some algorithms can be rewritten to reduce c, making them adaptive, but the transformation is complex, introduces overhead, and doesn't work for all algorithms.

The goal of this paper is to show that this gap closes under less stringent notions of optimality.

Beyond the worst-case gap. Previous analysis shows that in the worst case there is a gap between the cache-adaptive and ideal-cache/cache-oblivious models. However, the logarithmic gap may just be an artifact of an unrealistically powerful adversary. The proof depends on exhibiting worst-case memory profiles that force the algorithm to perform poorly. The worst-case profiles mimic the recursive structure of the algorithm and maintain a tight synchronization between the algorithm's execution and the fluctuations in memory size. A concrete example of the worst-case profile for matrix multiplication can be found in Section 3.

The natural question to ask is: what happens to these bad examples when they get tweaked in some way, so that they no longer track the program execution so precisely? Is this gap robust to the inherent randomness that occurs in real systems?

Results

Our main result shows that, given any probability distribution Σ over box-sizes, if each box has size chosen i.i.d. from the distribution Σ, (a, b, c)-regular algorithms achieve optimal performance in the cache-adaptive model, matching their performance in the ideal cache model.

Theorem 1. *Consider an (a, b, c)-regular algorithm $\mathcal{A}$, where $a > b$ are constants in $\mathbb{N}$ and $c = 1$. Let Σ be a probability distribution over box sizes, and consider a sequence of boxes $(\square_1, \square_2, \square_3, \ldots)$ drawn independently from the distribution Σ. If all boxes in Σ are sufficiently large in $\Omega(1)$, then $\mathcal{A}$ is cache-adaptive in expectation on the random sequences $(\square_1, \square_2, \ldots)$.*

Proving this requires a number of new combinatorial ideas, an overview of which appear in Section 4. The full version of the paper formally proves this positive result.

The proof begins by reinterpreting cache-adaptivity in expectation in terms of the expected stopping time of a certain natural martingale. We then show a relationship between the expected stopping time for a problem and the expected stopping times for the child subproblems. A key obstacle, however, is that the linear scans performed by the algorithm can cause the natural recurrence on stopping times to break. In particular, the recurrence is able to relate the time to complete subproblems (including scans) and the time to complete their parent problems (excluding scans); but is unable to consider the parent problems in their entirety (including scans). We fill in this gap by showing that the total effect of the scans at all levels of the recursion on the expected stopping time is at most a constant factor. By analyzing the aggregate effect of scans across all levels of the recursion, we get around the fact that certain scans at specific levels can have far more impact on the expected stopping time than others.

Robustness to weaker smoothings. We further show that drawing box-sizes indepenently from one-another is necessary in the sense that several weaker forms of smoothing fail to close the logarithmic gap between the ideal-cache and cache-adaptive models. We show that worst-case profiles are robust to all of the following perturbations: randomly tweaking the size of each box by a constant factor, randomly shifting the start time of the algorithm, and randomly (or even adversarialy) altering the recursive structure of the profile.

These smoothings substantially alter the overall structure of the profile, and eliminate any initial alignment between the program and the structure of the memory profile. Nonetheless, the smoothed profiles remain worst-case in expectation. That is, as long as some recursive structure is maintained within the profile, the algorithm is very likely to gradually synchronize its execution to the profile in deleterious ways. In this sense, even a small amount of global structure between the sizes of consecutive boxes is enough to cause the logarithmic gap.

Map. This paper is organized as follows. Section 2 gives the definitions and conventions used in the rest of the paper. Section 3 provides intuition for the fragility of worst-case memory profiles. Section 4 explains the intuition for the proofs of the main theorems and sketch the combinatorial ideas behind the proofs. The full proofs can be found in the full version of this paper. Section 5 gives an in-depth examination of previous work, and Section 6 gives concluding remarks.

[2] When $a < b$ and $c = 1$, the algorithm runs in linear time independent of the cache size, and hence is trivially cache-adaptive. We are not aware of any (a, b, c)-regular cache-oblivious algorithms with $c > 1$.

[3] (a, b, c)-regular algorithms are cache-adaptive when $a < b$ or $c < 1$. When $a = b$ and $c = 1$, no algorithm can be optimally cache-adaptive because such algorithms are already a $\Theta(\log \frac{M}{B})$ factor away from optimal in the DAM model [22]. This is why two-way merge sort, classic (i.e., not cache-oblivious) FFT, etc. cannot be optimal DAM algorithms.

2 PRELIMINARIES

The cache-adaptive model. The ***cache-adaptive*** (CA) model [5, 6] extends the classical disk access model (DAM) [1] to allow for the cache to change in size in each time step. In the DAM, the machine has a two-level memory hierarchy consisting of a fast ***cache*** (sometimes also referred to as ***memory*** or ***RAM***) of size M words and a slow ***disk***. Data is transferred between cache and disk in blocks of size B. An algorithm's running time is precisely the number of block transfers that it makes. Similarly, each I/O is a unit of time in the CA model.

In the cache-adaptive model, the size of fast memory is a (non-constant) function $m(t)$ giving the size of memory (in blocks) after the tth I/O. We use $M(t) = B \cdot m(t)$ to represent the size, in words, of memory at time t. We call $m(t)$ and $M(t)$ ***memory profiles*** in blocks and words, respectively. Although the cache-adaptive model allows the size of cache to change arbitrarily from one time-step to the next, prior work showed that we need only consider ***square profiles*** [5, 6]. Throughout this paper, we use the terms ***box*** and ***square*** interchangeably.

Definition 1 (Square Profile [5])**.** *A memory profile M or m is a **square profile** if there exist boundaries $0 = t_0 < t_1 < \ldots$ such that for all $t \in [t_i, t_{i+1})$, $m(t) = t_{i+1} - t_i$. In other words, a square memory profile is a step function where each step is exactly as long as it is tall. We will use the notation $(\square_1, \square_2, \ldots)$ to refer to a profile in which the i-th square is size $|\square_i|$.*

For convenience, we assume that cache is cleared at the start of each square. The paging results underlying cache-adaptivity [6] explain that this assumption is w.l.o.g. With this assumption, an algorithm gets to reference exactly X distinct blocks in a square of size X. Since any memory profile can be approximated with a square profile up to constant factors [5], any random distribution of generically produced profiles has a corresponding random distribution over square profiles that approximates it.

Recursive algorithms with (a, b, c)-regular structure. This paper focuses on algorithms with a common recursive structure.

Definition 2. *Let $a, b \in \mathbb{N}$ be constants, $b > 1$ and $c \in [0, 1]$. An **(a, b, c)-regular algorithm** is a recursive algorithm that, when run on a problem of size n blocks (equivalently $N = nB$ words), satisfies the following:*

- *On a problem of size n blocks, the algorithm accesses $\Theta(n)$ distinct blocks.*
- *Until the base case (when $n \in \Theta(1)$), each problem of size b blocks recurses on exactly a subproblems of size n/b.*
- *Within any non-base-case subproblem, the only computation besides the recursion is a **linear scan of size N^c/B**. This linear scan is any sequence of N^c contiguous memory accesses satisfying the property that a cache of a sufficiently large constant size can complete the sequence of accesses in time $O(N^c/B)$. Parts of the scan may be performed before, between, and after recursive calls.*

Remark 1. *When referring to the size of a subproblem, box, scan, etc., we use blocks, rather than machine words, as the default unit. We define (a, b, c)-regular algorithms to have a base case of size $O(1)$ blocks. This differs slightly from previous definitions [5, 6] which recurse down to $O(1)$ words.*

Remark 2. *The definition of linear scans ensures the following useful property. Consider a linear scan of size N^c/B. Consider any sequence of squares $(\square_1, \square_2, \ldots, \square_j)$, where each $|\square_i|$ is a sufficiently large constant, and where $\sum_{i=1}^{j} |\square_i| = \Omega(N^c/B)$, for a sufficiently large constant in the Ω. Then the sequence of squares can be used to complete the scan in its entirety.*

The following theorem gives a particularly simple rule for when an (a, b, c)-regular algorithm is optimal.

Theorem 2 ((a, b, c)-regular optimality [5], informal)**.** *Suppose $\mathcal{A}$ is an (a, b, c)-regular algorithm that is optimal in the DAM model. Then $\mathcal{A}$ is optimal in the cache-adaptive model if $c < 1$ or if $b > a$ and $a \geq 1$. If $c = 1$ and $a \geq b$, then $\mathcal{A}$ is $O(\log_b N)$ away from optimal on an input of size N in the cache-adaptive model. Optimality is defined as in [6], or equivalently as given by the notion of* efficiently cache adaptive*, defined below.*

Paper goal: closing the logarithmic gap. The above theorem uses a very structured memory profile in the case of $c = 1$ and $a \geq b$ to tease out the worst possible performance of $(a, b, 1)$-regular algorithms. We explore the smoothing of these profiles when $a > b$ in this paper. We leave the case of $a = b$ for future work because we prioritize the broader class of algorithms described by $a > b$.

Progress bounds in the cache-adaptive model. When an (a, b, c)-regular algorithm is run on a square profile, the ***progress*** of a box is the number of base-case subproblems performed (at least partly) within the box. Define the ***potential $\rho(|\square|)$*** of a box of size $|\square|$ to be the maximum possible progress that a size $|\square|$ box could ever make starting at any memory access of any execution of $\mathcal{A}$ on a problem of arbitrary size.

Lemma 1. *The potential of a box $\square$ for an (a, b, c)-regular algorithm $\mathcal{A}$ with $a > b$ and $c = 1$ is $\rho(|\square|) = \Theta(|\square|^{\log_b a})$.*

Proof. A square $\square$ can complete any subproblem A whose size in blocks is sufficiently small in $\Theta(|\square|)$. This allows $\square$ to complete $\Omega(a^{\log_b |\square|}) = \Omega(|\square|^{\log_b a})$ base-case subproblems, which proves $\rho(|\square|) \geq \Omega(|\square|^{\log_b a})$.

On the other hand, a square $\square$ is unable to complete in full any subproblem A whose size in blocks is sufficiently large in $\Theta(|\square|)$. It follows that $\square$ can complete base cases from at most two such subproblems A (the one that $\mathcal{A}$ begins $\square$ in and the one that $\mathcal{A}$ ends $\square$ in). This limits the number of base cases completed to $\rho(|\square|) \leq O(|\square|^{\log_b a})$. □

Intuitively, the potential of a box $\square$ is essentially the same as the number of base-case recursive leaves in a problem of size $|\square|$.

Optimality in the cache-adaptive model. The progress of each square is upper bounded by its potential. An execution of the algorithm $\mathcal{A}$ on a problem of size n blocks and on a square profile M is ***efficiently cache-adaptive*** if the sequence of squares $(\square_1, \square_2, \ldots, \square_j)$ given to the algorithm (with the final square rounded down in size to be only the portion of the square actually used)

satisfies

$$\sum_{i=1}^{j} \rho(|\square_i|) \leq O(n^{\log_b a}). \quad (1)$$

The right-hand side of the expression represents the total amount of progress that must be made by any (a, b, c)-regular algorithm on a problem of size n in order to complete. In summary, an execution is efficiently cache-adaptive on the profile if every square in the profile makes progress asymptotically equal to its maximum possible potential.

An algorithm $\mathcal{A}$ (rather than just a single execution) is ***efficiently cache-adaptive*** (or ***cache-adaptive*** for short) if every execution of $\mathcal{A}$ is efficiently cache-adaptive on every infinite square-profile consisting of squares that are all of sizes sufficiently large in $O(1)$.

By Lemma 1, Inequality 1 can be written as $\sum_{i=1}^{j} |\square_i|^{\log_b a} \leq O(n^{\log_b a})$. Since all squares $\square_i$ completed by the algorithm are of size $O(n)$, an equivalent requirement is

$$\sum_{i=1}^{j} \min(n, |\square_i|)^{\log_b a} \leq O(n^{\log_b a}). \quad (2)$$

This requirement has the advantage that the size of the final square $\square_j$ cannot affect the veracity of the condition. Consequently, when using this version of the condition, we may feel free to *not* round down the size of the final square $\square_j$ in the profile M.

The definition of efficiently cache-adaptive is easily adapted to use an arbitrary progress function and an arbitrary algorithm $\mathcal{A}$ that need not be (a, b, c)-regular.[4]

Cache-adaptivity over distributions of profiles. We now define what it means for an algorithm to be cache-adaptive in expectation over a distribution of memory profiles. This allows us to perform smoothed and average-case analyses in subsequent sections.

Definition 3 (Efficiently cache-adaptive in expectation)**.** *Let $\mathcal{M}$ be a distribution over (infinite) square memory profiles. Let M be a square-profile $(\square_1, \square_2, \ldots)$ drawn from the distribution $\mathcal{M}$, and define $\mathcal{S}_n$ to be the number of squares in the profile required by an (a, b, c)-regular $\mathcal{A}$ to complete on any problem of size n. We say that $\mathcal{A}$ is **(efficiently) cache-adaptive in expectation on $\mathcal{M}$** if for all problem sizes n,*

$$\mathbb{E}\left[\sum_{i=1}^{\mathcal{S}_n} \min(n, |\square_i|)^{\log_b a}\right] = O(n^{\log_b a}).$$

The bulk of this paper is devoted to investigating which memory-profile distributions cause $(a, b, 1)$-regular algorithms to be cache-adaptive in expectation.

[4]There is an alternative progress function based on operations. Consider the progress function in which each square makes progress equal to the number of memory accesses it completes. This generalizes our definition to non(a, b, c)-regular algorithms and, for many natural (a, b, c)-regular algorithms, this yields the same definition of cache-adaptivity as the above progress definition. However, the memory-access-based definition of progress can differ from our definition if some large scans are very non-homogeneous, i.e. if they contain portions in which a single small box can complete a large number of memory accesses. We use the sub-problem-based definition so that our results can apply to all (a, b, c)-regular algorithms.

A useful lemma. We conclude the section by presenting a useful lemma, known as the ***No-Catch-up Lemma***, that is implicitly present in [6]. The lemma will be used as a primitive throughout the paper, and for completeness, a full proof is given in the full version of the paper. Intuitively, the No-Catch-up Lemma states that delaying the start time of an algorithm can never help it finish earlier than it would have without the start time delay.

Lemma 2. *Let $\sigma = (r_1, r_2, r_3, \ldots)$ be a sequence of memory references, and let $S = (\square_1, \square_2, \ldots \square_k)$ be a sequence of squares. Suppose that if $\square_1$ starts at r_i, then $\square_k$ finishes at r_j. Then, for all $i' < i$, if $\square_1$ starts at $r_{i'}$, then for some $j' \leq j$, $\square_k$ finishes at $r_{j'}$.*

3 WHAT BAD MEMORY PROFILES LOOK LIKE

We begin by explaining how an (a, b, c)-regular algorithm can fail to be adaptive in the worst case, and why there is reason to hope that the worst cases are brittle.

*MM-Scan: a canonical non-adaptive algorithm. Consider a divide-and-conquer matrix-multiplication algorithm MM-Scan that computes eight subresults and then merges them together using a linear scan.MM-Scan is an $(8, 4, 1)$-regular cache-oblivious algorithm and its recurrence relation is $T(N) = 8T(N/4) + \Theta(N/B)$. Its I/O complexity is $O(N^{3/2}/\sqrt{M}B)$, which is optimal for an algorithm that performs all the elementary multiplications of a naïve nested-loop matrix multiply [28, 29].

However, since MM-Scan has $c = 1$ in its recurrence, it is not adaptive: there are bad memory profiles that cause it to run slowly despite giving MM-Scan ample aggregate resources[5]. This section gives intuition for what these bad profiles look like.

A worst-case profile for MM-Scan. Here's how to make a bad profile for MM-Scan [5]. The intuition is to give the algorithm lots of memory when it cannot benefit from it, i.e., when it is doing scans, and give it a paucity of memory when it could most use it, i.e., during subproblems.

Concretely, during a scan of size N, which takes N/B I/Os, set the memory to the fixed size N/B. Repeat recursively. Thus, a bad profile for MM-Scan on a problem of size N consists of eight recursive bad profiles for $N/4$ followed by a "square" of size N/B I/Os by N/B blocks of cache; see Figure 1.[6] This recursion continues down to squares of size $\Theta(B)$ blocks[7]. MM-Scan's I/O cost with this worst-case profile is exactly the same as if the memory stayed constant at its smallest possible value. MM-Scan can perform exactly one multiply of $\Theta(\sqrt{N} \times \sqrt{N})$ matrices on this profile. MM-InPlace, on the other hand, can perform $\Omega(\log \frac{N}{B})$ multiplies on this profile [5]. This proves that MM-Scan is not optimal in the cache-adaptive model.

[5]There is an alternate form of the algorithm, MM-InPlace, that immediately adds the results of elementary multiplications into the output matrix as they are computed. Since it needs no linear scan to merge results from sub-problems, it is an (8, 4, 0)-regular algorithm. Consequently, its I/O complexity in the DAM model is also $O(N^{3/2}/\sqrt{M}B)$, but it *is* optimally cache-adaptive.

[6]In the cache-adaptive model, it's enough to analyze cache-oblivious algorithms only on ***square profiles***, defined as follows [5]. Whenever the RAM size changes to have the capacity for x blocks, it stays constant x I/Os before it can change again. This paper focuses exclusively on cache-oblivious algorithms, so we use square profiles throughout.

[7]We stop at $\Theta(B)$ blocks due to the ***tall-cache requirement*** of MM-Scan [28]

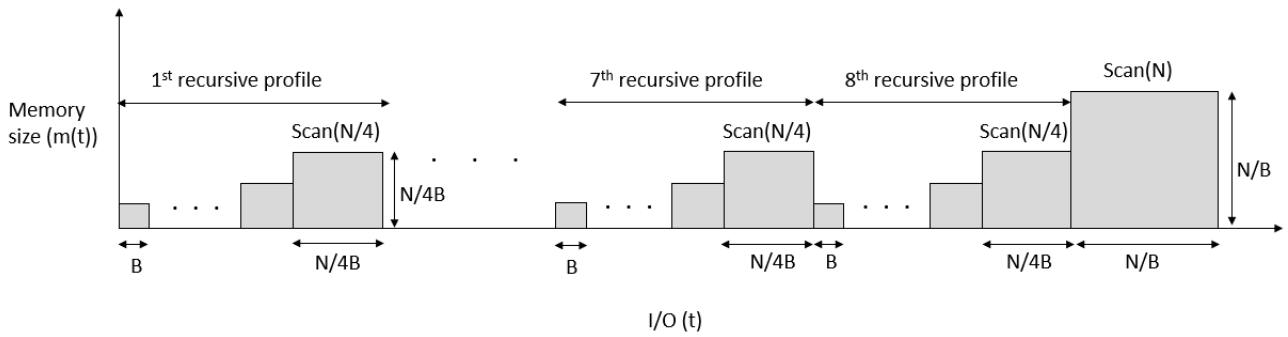

Figure 1: A bad profile for MM-SCAN as defined recursively.

This worst-case profile exactly tracks the execution of MM-Scan. From the perspective of the algorithm, the memory profile *does the wrong thing at every time step*; whenever MM-Scan cannot use more memory, it gets the maximum amount possible, and whenever it can use more memory, that memory gets taken away. This bad example for matrix multiplication generalizes to any $(a, b, 1)$-regular algorithm. When $c < 1$, this construction is ineffective—the scans are simply too small to waste a significant amount of resources.

The MM-Scan example reveals a fascinating combinatorial aspect of divide-and-conquer algorithms. At some points of the execution, the I/O performance is sensitive to the size of memory and sometimes it is almost entirely insensitive. These changes in memory sensitivity make cache-adaptive analysis nontrivial.

4 TECHNICAL OVERVIEW

Cache-Adaptivity on Randomly Shuffled Profiles

The main technical result of the paper is that random shuffling of adversarially constructed box-profiles makes (a, b, c)-regular algorithms where $a > b$ and $c = 1$ cache-adaptive in expectation. In the full version of the paper we prove the following:

Theorem 3 (Informal)**.** *Consider an (a, b, c)-regular algorithm $\mathcal{A}$, where $a > b$ ($b > 1$) are constants in $\mathbb{N}$ and $c = 1$. Let Σ be a probability distribution over box sizes, and consider a sequence of boxes $(\square_1, \square_2, \square_3, \ldots)$ with sizes drawn independently from the distribution. Then $\mathcal{A}$ is cache-adaptive (in expectation) on the random sequence of boxes $(\square_1, \square_2, \square_3, \ldots)$.*

For simplicity, we discuss here the case where the block size $B = 1$, and $\mathcal{A}$ has the same values of a, b, c as the not-in-place naïve matrix-multiplication algorithm,with $a = 8$ and $b = 4$ and $c = 1$. Moreover, we assume that all box sizes and problem sizes are powers of 4. Doing so ensures $|a - b| \geq \Omega(1)$, allows us to simplify many of the expressions in intermediate calculations, and frees us from tracking factors of B and its added complexity[8]. Additionally, we consider a simplified model of caching: any box of size s that begins in a subproblem of size s or smaller completes to the end of the problem of size s containing it (and goes no further); and any box of size s that begins in the scan of a problem of size greater than s continues either for the rest of the scan or for s accesses in the scan, whichever is shorter. As a final simplification, we assume that each scan in each problem of some size s consists of exactly s memory accesses. In fact, we show in the full version of the paper that these simplifications may be made without loss of generality for arbitrary (a, b, c)-regular algorithms.

Let $\square$ denote a single box drawn from the distribution Σ. The proof of Theorem 3 begins by applying the Martingale Optional Stopping Theorem to combinatorially reinterpret what it means for $\mathcal{A}$ to be cache-adaptive in expectation on the random sequence $(\square_1, \square_2, \ldots)$. In particular, if $f(n)$ is the expected number of boxes needed for $\mathcal{A}$ to complete a problem of size n, then cache-adaptivity-in-expectation reduces to:

$$f(n) \leq \frac{O(8^{\log_4 n})}{m_n} = \frac{O(n^{\log_4 8})}{m_n} = \frac{O(n^{3/2})}{m_n}, \tag{3}$$

where $m_n = \mathbb{E}\left[\min(n, |\square|)^{3/2}\right]$ is the ***average n-bounded potential*** of a box.

At the heart of the proof of Theorem 3 is a somewhat unintuitive combinatorial argument for expressing $f(n)$, the expected number of boxes needed to complete a problem of size n, in terms of $f(n/4)$.

Lemma 3 (Stated for the simplified assumptions)**.** *Define $p = \Pr[|\square| \geq n] \cdot f(n/4)$. Then, the expected number of squares to complete the subproblems in a problem of size n is exactly $\sum_{i=1}^{8}(1-p)^{i-1} f(n/4)$, and the expected number of additional squares needed to complete the final scan is $(1 - \Theta(p)) \cdot \frac{\Theta(n)}{\mathbb{E}[\min(|\square|, n)]}$.*

Proof Sketch. When executing a problem of size n, the first subproblem requires $f(n/4)$ boxes to complete, on average. Define q to be the probability the boxes used to complete the first subproblem include a box of size n or larger. Then with probability q, no additional boxes are required[9] to complete the rest of the problem of size n. We will show that $q = p$ later in this proof. Otherwise, an average of $f(n/4)$ additional boxes are needed to complete the next subproblem of size $n/4$. Again, with probability q, one of these boxes completes the rest of the problem in its entirety. Similarly,

[8] At a high level, the cache-line analysis works exactly as one would expect for a nice, recursive algorithm. However, actually getting the probabilistic analysis correct adds some complication and is resolved through several of the simplification steps in the full version of the paper. As intuition (a, b, c)-regular algorithms will recurse down to sizes small enough to fit inside cache lines getting the requisite cache locality.

[9] This is due to the aforementioned simplified caching model.

the probability that the i-th subproblem is completed by a large box from a previous subproblem is $(1-q)^{i-1}$. Thus the expected number of boxes needed to complete all 8 subproblems is

$$\sum_{i=1}^{8}(1-q)^{i-1}f(n/4). \tag{4}$$

Remarkably, the probability q can also be expressed in terms of $f(n/4)$. Define S to be the random variable for the number of boxes used to complete the first subproblem of size $n/4$; define $\ell \le O(n^{3/2})$ to be an upper bound for S; and define X to be the random variable for the number of the boxes in the subsequence $\square_1, \ldots, \square_S$ that are of size n or greater. The expectation of X can be expressed as

$$\mathbb{E}[X] = \sum_{i=1}^{\ell} \Pr[S \ge i] \cdot \Pr[|\square_i| \ge n \mid S \ge i].$$

Since $|\square_i|$ is independent of $|\square_1|, \ldots, |\square_{i-1}|$, we have that $\Pr[|\square_i| \ge n \mid S \ge i] = \Pr[|\square_i| \ge n]$. Thus

$$\begin{aligned}\mathbb{E}[X] &= \Pr[|\square| \ge n] \cdot \sum_{i=1}^{\ell} \Pr[S \ge i] \\ &= \Pr[|\square| \ge n] \cdot \mathbb{E}[S] = \Pr[|\square| \ge n] \cdot f(n/4).\end{aligned}$$

Notice, however, that at most one of the boxes $\square_1, \ldots, \square_S$ can have size n or larger (since such a box will immediately complete the subproblem). Thus X is an indicator variable, meaning that $q = \Pr[X \ge 1] = \mathbb{E}[X] = \Pr[|\square| \ge n] \cdot f(n/4) = p$. So $q = p$, as promised. Expanding Equation 4, we get that the expected number of boxes needed to complete the 8 subproblems is, as desired, at most

$$\sum_{i=1}^{8} \Big(1 - \Pr[|\square| \ge n] \cdot f(n/4)\Big)^{i-1} f(n/4) \tag{5}$$

Next we consider the boxes needed to complete the final scan. Suppose the scan were to be run on its own. Let K denote the number of boxes needed to complete it, and let $\square'_1, \ldots, \square'_K$ denote those squares.

Rather than consider $\mathbb{E}[K]$ directly, we instead consider $\mathbb{E}[K] \cdot \mathbb{E}[\min(|\square|, n)]$. Through a combinatorial reinterpretation, we have

$$\begin{aligned}E[K] \cdot \mathbb{E}[\min(|\square|, n)] &= \mathbb{E}[\min(|\square|, n)] \cdot \sum_{i=1}^{\ell} \Pr[K \ge i] \\ &= \sum_{i=1}^{\ell} \Pr[K \ge i] \cdot \mathbb{E}[\min(|\square'_i|, n) \mid K \ge i] \\ &= \mathbb{E}\left[\sum_{i=1}^{K} \min(|\square'_i|, n)\right].\end{aligned}$$

The quantity in the final expectation has the remarkable property that it is *deterministically* between n and $2n-1$. Thus the same can be said for its expectation, implying that $\mathbb{E}[K] \cdot \mathbb{E}[\min(|\square|, n)] = \Theta(n)$.

Recall that K is the expected number of boxes to complete the scan on its own. In our problem, the scan is at the end of a problem, and thus with probability $1-(1-p)^8 = \Theta(p)$, the scan is completed by a large box from one of the preceding subproblems. Hence the expected number of additional boxes to complete the scan is $(1-\Theta(p)) \cdot \frac{\Theta(n)}{\mathbb{E}[\min(|\square|, n)]}$.

□

Lemma 3 suggests a natural inductive approach to proving Theorem 3. Rather than explicitly showing that $f(n) \le \frac{O(n^{3/2})}{m_n}$, one could instead prove the result by induction, arguing for each n that

$$\frac{f(n)}{f(n/4)} \le \frac{n^{3/2}/m_n}{(n/4)^{3/2}/m_{n/4}} = 8 \cdot \frac{m_{n/4}}{m_n}. \tag{6}$$

One can construct example box-size distributions Σ showing that the Equation 6 does not always hold, however. In particular, the scan at the end of a subproblem of size n can make $f(n)$ sufficiently larger than $f(n/4)$ that Equation 6 is violated. To get around this problem, one could attempt to instead prove that

$$\frac{f'(n)}{f(n/4)} \le 8 \cdot \frac{m_{n/4}}{m_n}, \tag{7}$$

where $f'(n)$ is the expected number of boxes needed to complete a problem of size n, without performing the final scan at the end. Unlike Equation 6, Equation 7 does not inductively imply a bound of the form $f(n) \le \frac{O(n^{3/2})}{m_n}$, which is necessary for cache-adaptivity in expectation. If one additionally proves that

$$\prod_{4^k \le n} \frac{f(4^k)}{f'(4^k)} \le O(1), \tag{8}$$

then Equation 8 could be used to "fill in the holes in the induction" in order to complete a proof of cache-adaptivity. Equation 8 is somewhat unintuitive in the sense that individual terms in the product can actually be a positive constant greater than 1. The inequality states that, even though the scans in an individual subproblem size could have a significant impact on $f(n)$, the aggregate effect over all sizes is no more than constant.

To make this semi-inductive proof structure feasible, one additional insight is necessary. Rather than proving Equation 7 for all values n, one can instead restrict oneself only to values n such that

$$f(n) \ge C \cdot \frac{n^{3/2}}{m_n}, \tag{9}$$

where C is an arbitrarily large constant of our choice. In particular, if n_0 is the largest power of 4 less than our problem-size such that $f(n_0) < C \cdot \frac{n^{3/2}}{m_n}$, then we can use cache-adaptivity within problems of size n_0 as a base case, and then prove Equation 7 only for problem-sizes between n_0 and n. Similarly, we can restrict the product in Equation 8 to ignore problem sizes of size smaller than n_0.

When Equation 9 holds, Equation 7 can be interpreted as a negative feedback loop, saying that as we look at problem sizes $n = 1, 4, 16, \ldots$, whenever $f(n)$ becomes large enough to be on the cusp of violating cache-adaptivity, there exists downward pressure (in the form of Equation 7) that prevents it from continuing to grow in an unmitigated fashion.

The full proof of Theorem 3 takes the structure outlined above. At its core are the combinatorial arguments used in Lemma 3, which allow us to recast $f(n)$ and $f'(n)$ in terms of $f(n/4)$ and $f'(n/4)$. When applied in the correct manner, these arguments can be used

to show Equation 7 (assuming Equation 9) with only a few additional ideas. The proof of Equation 8 ends up being somewhat more sophisticated, using combinatorial ideas from Lemma 3 in order to expand each of the terms $f(4^k)/f'(4^k)$, and then relying on a series of subtle cancellation arguments in order to bound the end product by a constant.

Robustness of Worst-Case Profiles

We consider three natural forms of smoothing on worst-case profiles. Remarkably, the worst-case nature of the profiles persists in all three cases. The canonical worst-case profile is highly structured, giving the algorithm a larger cache precisely when the algorithm does not need it. It is tempting to conclude that bad profiles must remain closely synchronized with the progression of the algorithm. By exploiting self-symmetry within worst-case profiles as well as the power of the No-Catchup Lemma, our results establish that this is not the case. The No-Catchup Lemma, in particular, allows us to capture the idea of an algorithm resynchronizing with a profile, even after the profile has been perturbed.

We begin by defining a canonical (a, b, c)-regular algorithm $\mathcal{A}_n$ on problems of size n, and a corresponding worst-case profile $M_{a,b}$. The profile $M_{a,b}$ completes each scan of size k in $\mathcal{A}$ in a single box of size k, thereby ensuring that every box makes its minimum possible progress. The profile $M_{a,b}$ is the ***limit profile*** of the sequence of profiles $M_{a,b}(n)$ for $n = 1, b, b^2, \ldots$, constructed recursively by defining $M_{a,b}(n)$ by concatenating together a copies of $M_{a,b}(n/b)$ and then placing a box of size n at the end. The algorithm $\mathcal{A}_n$ requires the entirety of the profile $M_{a,b}(n)$ to complete. One can check inductively that $M_{a,b}(n)$ has total potential $n^{\log_b a} \cdot \log n$, thereby making $M_{a,b}$ a worst-case profile.

Box-size perturbations. Consider any probability distribution $\mathcal{P}$ over $[0, t]$ for $t \leq \sqrt{n}$ such that for X drawn at random from $\mathcal{P}$, $\mathbb{E}[X] = \Theta(t)$. Let $X_1, X_2, \ldots$ be drawn iid from $\mathcal{P}$ and define $\mathcal{M}$ to be the distribution over square profiles obtained by replacing each box $\square_i$ in $\mathcal{M}$ with a box of size $|\square_i| \cdot X_i$. In the full version of the paper we show that the highly perturbed square profiles in $\mathcal{M}$ still remain worst-case in expectation.

The proof takes two parts. We begin by defining T to be the smallest power of b greater than t, and considering the profile $T \cdot M_{a,b}$ obtained by multiplying each box's size by T. Exploiting self-symmetry in the definition of $M_{a,b}$, we are able to reinterpret $T \cdot M_{a,b}$ as the profile $M_{a,b}$ in which all boxes of size smaller than T have been *removed*. Recall that $M_{a,b}(n)$ denotes the prefix of $M_{a,b}$ on which $\mathcal{A}_n$ completes. Using the fact that $T \leq \sqrt{n}$, we prove that the boxes of size smaller than T contain at most a constant fraction of the potential in the prefix $M_{a,b}(n)$. On the other hand, by iterative applications of the No-Catchup Lemma, the removal of the boxes cannot facilitate $\mathcal{A}$ to finish earlier in the profile. Combining these facts, we establish that $T \cdot M_{a,b}$ remains worst-case.

To obtain an element of $\mathcal{M}$ from $T \cdot M_{a,b}$, one simply multiplies the size of each box $\square_i$ in $T \cdot M_{a,b}$ by X_i/T, where T is drawn from the distribution $\mathcal{P}$. Using that $\mathbb{E}[X_i/T] = \Theta(1)$ and that $n^{\log_b a}$ is a convex function, Jensen's inequality tells us that the expected potential of the new box of size $\frac{|\square_i| \cdot X_i}{T}$ is at least a constant fraction of the original potential. Since the perturbations preserve the expected potentials of the boxes in $T \cdot M_{a,b}$ up to a constant factor, we can prove that the resulting profile is worst-case in expectation by demonstrating that the perturbations do not result in $\mathcal{A}_n$ finishing earlier in $T \cdot M_{a,b}$ then it would have otherwise. Since each perturbation can only reduce the size of a box in $T \cdot M_{a,b}$, this can be shown by iterative applications of the No-Catchup Lemma.

Start time perturbations. We consider what happens if the memory profile $M_{a,b}(n)$ is cyclic-shifted by a random amount. This corresponds to executing $\mathcal{A}_{a,b}(n)$ starting at a random start-time in the cyclic version of $M_{a,b}(n)$. Again, the resulting distribution of profiles remains worst-case in expectation.

The key insight in the proof is that $M_{a,b}(n)$ can be expressed as the concatenation of two profiles $A = (\square_1, \ldots, \square_x)$ and $B = (\square'_1, \ldots, \square'_y)$ such that

$$\sum_{i=1}^{x} |\square_i| \geq \Omega\left(\sum_{i=1}^{y} |\square'_i|\right), \tag{10}$$

$$\sum_{i=1}^{x} |\square_i|^{\log_b a} \leq O\left(\sum_{i=1}^{y} |\square'_i|^{\log_b a}\right). \tag{11}$$

Equation 10 establishes that with at least constant probability, a random selected start-time in the profile $M_{a,b}(n)$ falls in the prefix A. By a slight variant on the No-Catchup Lemma, if $\mathcal{A}$ is executed starting at that random start-time, it is guaranteed to use all of the boxes in the suffix B. By Equation 11, however, these boxes contain a constant fraction of the potential from the original worst-case profile $M_{a,b}(n)$. Thus, with constant probability the algorithm $\mathcal{A}$ runs at a random start-time that results in a profile that is still worst-case. This ensures that the randomly shifted profile will be worst-case in expectation.

Box-order perturbations. The bad profile, $M_{a,b}$, is constructed recursively by making a copies of $M_{a,b}(n/b)$ followed by a box of size n. The box comes at the end, intuitively, because all $(a, b, 1)$-regular algorithms with upfront scans in each subproblem can converted to an equivalent $(a, b, 1)$-regular algorithm, where the scans in all subproblems are at the end, preceded by a single linear scan.

We consider a relaxation of the construction of $M_{a,b}$. When constructing $M_{a,b}(n)$ recursively, rather than always placing a box of size n after the *final* instance of $M_{a,b}(n/b)$, we instead allow ourselves to place the box of size n after *any* of the a recursive instances of $M_{a,b}(n/b)$ (each of which may no longer be identical to the others due to the non-determinedness of the new recursive construction).

Although at first glance moving the largest box in the profile seems to closely resemble the random shuffling considered in the full version of the paper, we prove that the resulting distribution over square profiles again remains worst-case in expectation. In fact, we can prove a stronger statement: for memory profile M drawn from the resulting distribution $\mathcal{M}$ of square profiles, M is a worst-case profile with probability *one*.

To prove this, we begin by constructing what we call a ***universal worst-case profile*** $U_{a,b}$. The prefixes $U_{a,b}(n)$ of the profile $U_{a,b}$ are recursively constructed in the same manner as $M_{a,b}$, except with the following twist: rather than concatenating together a copies

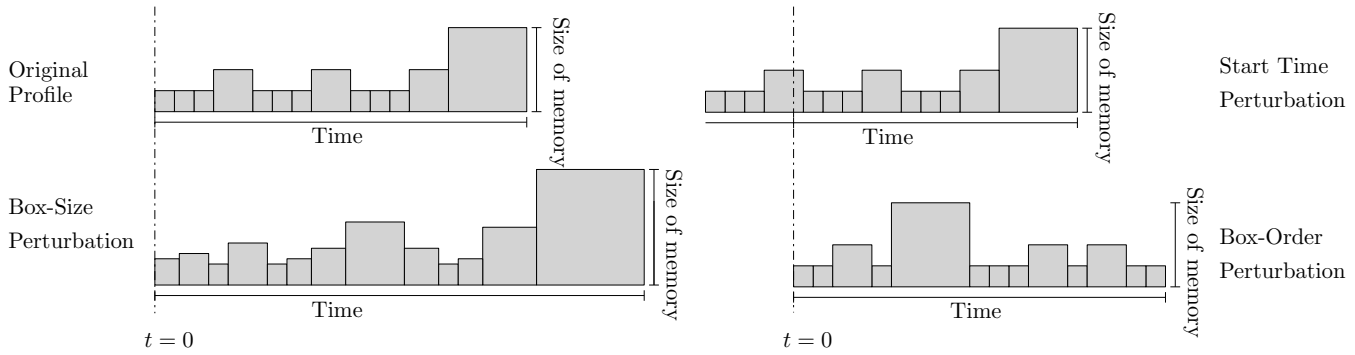

Figure 2: Four square profiles providing an example of the three smoothing transformations explored in the full version of the paper.The first profile is an example of an original profile. The next three profiles represent respectively: *box-size perturbations*, *start time perturbations* and, *box-order perturbations* of the Original Profile.

of $U_{a,b}(n/b)$ with a single box of size n at the end, we instead concatenate together a copies of $U_{a,b}(n/b)$ with a box of size n at the end of each copy. Exploiting self-symmetry in the construction of $M_{a,b}$, we show that $U_{a,b}$ is also a worst-case profile.

Each square profile in the distribution $\mathcal{M}$ can be obtained from $U_{a,b}$ by removing a $\frac{a-1}{a}$ fraction of the boxes from $U_{a,b}$. By iterative applications of the No-Catchup Lemma, such removals cannot facilitate the algorithm $\mathcal{A}_{a,b}$ to finish earlier in the profile than it would have otherwise. On the other hand, because an $\frac{1}{a}$-fraction of the boxes of each size remain after the removals, the total potential in each prefix $U_{a,b}(n)$ of $U_{a,b}$ is affected only by a constant factor by the removals. Thus all square profiles from the distribution $\mathcal{M}$ is guaranteed to still be worst-case.

5 RELATED WORK

Modeling the performance of algorithms in the real-world is an active area of study and has broad implications both theoretically and for performance engineering. In order to apply our algorithms to real-world systems, it is important to find the right model in which the theoretical efficiency of our algorithms closely matches their practical efficiency.

The ***disk access model (DAM)*** was formulated [1, 33] to account for multi-level memory hierarchies (present in real systems) where the size of memory available for computation and the speed of computation differs in each level. The DAM [1] models a 2-level memory hierarchy with a large (infinite sized) but slow disk, and a small (bounded by M) but fast cache. The drawback of DAM is that efficient algorithms developed in this model require knowledge of cache size. The ***ideal-cache model*** [28, 51] was proposed to counteract this drawback by building an automatic paging algorithm into the model and providing *no knowledge* of the cache size to algorithms. Thus, ***cache-oblivious algorithms*** [21, 36] are independent of the memory parameter and can be applied to complex multi-level architectures where the size of each memory-level is unknown. There exists a plethora of previous work on the performance analysis and implementations of cache-oblivious algorithms (on single-core and multi-core machines) [7, 8, 10, 12, 15, 16, 19, 27, 28, 38, 61, 62]. Among other limits [4, 11], one critical limit of the cache-oblivious model is that it does not account for *changing cache-size*. In fact, preliminary experimental studies have shown that two cache-oblivious algorithms (with the same I/O-complexity) might in fact perform vastly differently under a changing cache [40].

Changing cache size can stem from a variety of reasons. For example, shared caches in multi-core environment may allocate different portions of the cache to different processes at any time (and this allocation could be independent of the memory needed by each process). There has been substantial work on paging in shared-cache environments. For example, Peserico formulated alternative models for page replacement [50] provided a fluctuating cache. However, Peserico's page-replacement model differs from the cache-adaptive model because in his model, the cache-size changes at specific locations in the page-request sequence as opposed to being temporally related to each individual I/O. Other page replacement policies have applied to multicore shared-cache environments [35] where several processes share the same cache [2, 32, 34, 41, 42, 63] leading to situations where page sizes can vary [42] and where an application can adjust the cache size itself [34, 63].

Theoretical [2, 3] and empirical studies [47, 65, 66] have been done in the past to study partial aspects of adaptivity to memory fluctuations [13, 31, 44, 45, 48, 64, 66]. Barve and Vitter [2, 3] were the first to generalize the DAM model to account for changing cache size. In their model, they provide optimal algorithms for sorting, matrix multiplication, LU decomposition, FFT, and permutation but stops just short of a generalized technique for finding algorithms that are optimal under memory fluctuations [2, 3]. In their model, the cache is guaranteed to stay at size M for M/B I/Os. In this way, their model is very similar to our notion of square profiles.

The cache-adaptive model [6] introduced the notion of a ***memory profile***. The memory profile provides the cache size at each time step (defined as an I/O-operation), and at each time step the cache can increase by 1 block or decrease by an arbitrary amount. Bender et al. [5] went on to show that *any* optimal (in the DAM) (a, b, c)-regular algorithm where $a > b$ and $c < 1$ is ***cache-adaptive*** or *optimal* under this model. However, disappointingly, they showed that (a, b, c)-regular algorithms where $c = 1$ can be up to a log-factor away from optimal [5]. This leads to the the question of whether

non-adaptive (a, b, c)-regular algorithms can be turned into cache-adaptive algorithms via some procedure. Lincoln et al. [40] took the first step in this direction by introducing a ***scan-hiding*** procedure for turning certain non-adaptive (a, b, c)-regular algorithms into cache-adaptive ones. Although scan-hiding takes polynomial time, it introduces too much overhead and also does not apply to all (a, b, c)-regular algorithms where $a > b$ and $c = 1$.

This paper takes another important step in this direction by showing that (a, b, c)-regular algorithms where $a > b$ and $c = 1$ *are cache-adaptive in expectation*. Whereas previous papers analyzed all algorithms in the *worst-case*, we believe that this is, in fact, unnecessary and does not accurately depict real-world architectures. We introduce the notion of average case cache-adaptivity in what we hope to be a more accurate picture of shared-cache multi-core systems.

Acknowledgments

We gratefully acknowledge support from NSF grants: CCF-1439084, CCF-1533644, CCF-1725543, CSR-1763680, CCF-1716252, CCF-1617618, CNS-1938709, CNS-1553510.

This research is partially supported by an NSF GRFP fellowship and Fannie & John Hertz Foundation fellowship. This research is also supported by the United States Air Force Research Laboratory and was accomplished under Cooperative Agreement Number FA8750-19-2-1000.[10]

6 CONCLUSION

This paper presents the first beyond-worst-case analysis of (a, b, c)-regular cache-adaptive algorithms. The main positive result in this paper gives hope for cache-adaptivity: even though the worst-case profile from previous work [5, 6] is robust under random perturbations and shuffling, many (a, b, c)-regular algorithms become cache-adaptive in expectation on profiles generated from any distribution. Notably, to our knowledge, all currently known subcubic matrix multiplication algorithms (such as Strassen's [55], Vassilevska Williams' [59], Coppersmith-Winograd's [20], and Le Gall's [39]) were a logarithmic factor away from adaptive under worst-case analysis, but are adaptive in expectation on random profiles via smoothed analysis. Our results provide guidance for analyzing cache-adaptive algorithms on profiles beyond the adversarially constructed worst-case profile.

Cache fluctuations are a fact of life on modern hardware, but many open questions remain. In this paper, we randomized memory profiles for deterministic (a, b, c)-regular algorithms. Could randomized algorithms also overcome worst-case profiles and result in cache-adaptivity? On the empirical side, which patterns of memory fluctuations occur in the real world? Further exploration of beyond-worst-case analysis may help model practical memory patterns more accurately.

REFERENCES

[1] Alok Aggarwal and S. Vitter, Jeffrey. 1988. The input/output complexity of sorting and related problems. *Commun. ACM* 31, 9 (Sept. 1988), 1116–1127.

[2] Rakesh Barve and Jeffrey S. Vitter. 1998. *External memory algorithms with dynamically changing memory allocations.* Technical Report. Duke University.

[3] Rakesh D. Barve and Jeffrey Scott Vitter. 1999. A Theoretical Framework for Memory-Adaptive Algorithms. In *Proc. 40th Annual Symposium on the Foundations of Computer Science (FOCS)*. 273–284.

[4] Michael A. Bender, Gerth Stølting Brodal, Rolf Fagerberg, Dongdong Ge, Simai He, Haodong Hu, John Iacono, and Alejandro López-Ortiz. 2011. The Cost of Cache-Oblivious Searching. *Algorithmica* 61, 2 (2011), 463–505.

[5] Michael A. Bender, Erik D. Demaine, Roozbeh Ebrahimi, Jeremy T. Fineman, Rob Johnson, Andrea Lincoln, Jayson Lynch, and Samuel McCauley. 2016. Cache-Adaptive Analysis. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 135–144. https://doi.org/10.1145/2935764.2935798

[6] Michael A. Bender, Roozbeh Ebrahimi, Jeremy T. Fineman, Golnaz Ghasemiesfeh, Rob Johnson, and Samuel McCauley. 2014. Cache-adaptive Algorithms. In *Proc. 25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)* (Portland, Oregon). 958–971.

[7] Michael A. Bender, Martin Farach-Colton, Jeremy T. Fineman, Yonatan R. Fogel, Bradley C. Kuszmaul, and Jelani Nelson. 2007. Cache-oblivious Streaming B-trees. In *Proc. 19th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 81–92.

[8] Michael A. Bender, Martin Farach-Colton, and Bradley C. Kuszmaul. 2006. Cache-oblivious string B-trees. In *Proc. 25th Annual ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*. 233–242.

[9] Andreas Björklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. 2014. Listing triangles. In *International Colloquium on Automata, Languages, and Programming*. Springer, 223–234.

[10] Guy E Blelloch, Rezaul A Chowdhury, Phillip B Gibbons, Vijaya Ramachandran, Shimin Chen, and Michael Kozuch. 2008. Provably good multicore cache performance for divide-and-conquer algorithms. In *Proc. 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. Society for Industrial and Applied Mathematics, 501–510.

[11] Gerth Stølting Brodal and Rolf Fagerberg. 2003. On the limits of cache-obliviousness. In *Proc. 35th Annual ACM Symposium on Theory of Computing (STOC)* (San Diego, CA, USA). New York, NY, USA, 307–315.

[12] Gerth Stølting Brodal, Rolf Fagerberg, and Kristoffer Vinther. 2007. Engineering a cache-oblivious sorting algorithm. *ACM Journal of Experimental Algorithmics* 12 (2007).

[13] Kurt P Brown, Michael James Carey, and Miron Livny. 1993. Managing memory to meet multiclass workload response time goals. In *Proc. 19th International Conference on Very Large Data Bases (VLDB)*. Institute of Electrical & Electronics Engineers (IEEE), 328–328.

[14] Jichuan Chang and Gurindar S Sohi. 2006. *Cooperative caching for chip multiprocessors.* Vol. 34.

[15] Rezaul Chowdhury, Muhibur Rasheed, Donald Keidel, Maysam Moussalem, Arthur Olson, Michel Sanner, and Chandrajit Bajaj. 2013. Protein-Protein Docking with F2Dock 2.0 and GB-Rerank. *PLoS ONE* 8, 3 (2013).

[16] Rezaul Alam Chowdhury, Hai-Son Le, and Vijaya Ramachandran. 2010. Cache-Oblivious Dynamic Programming for Bioinformatics. *IEEE/ACM Trans. Comput. Biology Bioinform.* 7, 3 (2010), 495–510.

[17] Rezaul Alam Chowdhury and Vijaya Ramachandran. 2006. Cache-oblivious dynamic programming. In *Proc. 17th annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 591–600.

[18] Rezaul Alam Chowdhury and Vijaya Ramachandran. 2010. The cache-oblivious gaussian elimination paradigm: theoretical framework, parallelization and experimental evaluation. *Theory of Computing Systems* 47, 4 (2010), 878–919.

[19] R Cole and V Ramachandran. 2010. Efficient resource oblivious scheduling of multicore algorithms. manuscript.

[20] Don Coppersmith and Shmuel Winograd. 1990. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation* 9, 3 (1990), 251–280.

[21] Erik D. Demaine. 2002. Cache-Oblivious Algorithms and Data Structures. (2002). Lecture Notes from the EEF Summer School on Massive Data Sets.

[22] Erik D. Demaine, Andrea Lincoln, Quanquan C. Liu, Jayson Lynch, and Virginia Vassilevska Williams. 2018. Fine-grained I/O Complexity via Reductions: New Lower Bounds, Faster Algorithms, and a Time Hierarchy. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*. 34:1–34:23. https://doi.org/10.4230/LIPIcs.ITCS.2018.34

[23] Peter J Denning. 1968. Thrashing: Its causes and prevention. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*. 915–922.

[24] Peter J. Denning. 1980. Working sets past and present. *IEEE Transactions on Software Engineering* 1 (1980), 64–84.

[25] Dave Dice, Virendra J. Marathe, and Nir Shavit. 2014. Brief Announcement: Persistent Unfairness Arising from Cache Residency Imbalance. In *26th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '14, Prague, Czech Republic - June 23 - 25, 2014*. 82–83. https://doi.org/10.1145/2612669.2612703

[26] David Durfee, John Peebles, Richard Peng, and Anup B. Rao. 2017. Determinant-Preserving Sparsification of SDDM Matrices with Applications to Counting and Sampling Spanning Trees. In *FOCS*. IEEE Computer Society, 926–937.

[27] Matteo Frigo and Steven G Johnson. 2005. The design and implementation of FFTW3. *Proc. IEEE* 93, 2 (2005), 216–231.

[28] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. 1999. Cache-Oblivious Algorithms. In *Proc. 40th Annual Symposium on the Foundations of Computer Science (FOCS)*. 285–298.

[29] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. 2012. Cache-Oblivious Algorithms. *ACM Transactions on Algorithms* 8, 1 (2012), 4.

[30] Matteo Frigo and Volker Strumpen. 2005. Cache-oblivious stencil computations. Citeseer.

[31] Goetz Graefe. 2013. A New Memory-Adaptive External Merge Sort. Private communication.

[32] Avinatan Hassidim. 2010. Cache Replacement Policies for Multicore Processors. In *Proc. 1st Annual Symposium on Innovations in Computer Science (ICS)*. 501–509.

[33] Jia-Wei Hong and H. T. Kung. 1981. I/O complexity: The red-blue pebble game. In *Proc. 13th Annual ACM Symposium on the Theory of Computation (STOC)*. 326–333.

[34] Sandy Irani. 1997. Page Replacement with Multi-Size Pages and Applications to Web Caching. In *Proc. 29th Annual ACM Symposium on the Theory of Computing (STOC)*. 701–710.

[35] Anil Kumar Katti and Vijaya Ramachandran. 2012. Competitive Cache Replacement Strategies for Shared Cache Environments. In *Proc. 26th International Parallel and Distributed Processing Symposium (IPDPS) (IPDPS '12)*. 215–226.

[36] P. Kumar. 2003. Cache Oblivious Algorithms. (2003), 193–212. http://link.springer.de/link/service/series/0558/tocs/t2625.htm

[37] Rasmus Kyng and Sushant Sachdeva. 2016. Approximate Gaussian Elimination for Laplacians - Fast, Sparse, and Simple. In *FOCS*. IEEE Computer Society, 573–582.

[38] R.E. Ladner, R. Fortna, and B.-H. Nguyen. 2002. A Comparison of Cache Aware and Cache Oblivious Static Search Trees Using Program Instrumentation. *Experimental Algorithmics* (2002), 78–92.

[39] François Le Gall. 2014. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation*. 296–303.

[40] Andrea Lincoln, Quanquan C. Liu, Jayson Lynch, and Helen Xu. 2018. Cache-Adaptive Exploration: Experimental Results and Scan-Hiding for Adaptivity. In *Proc. 30th on Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 213–222. https://doi.org/10.1145/3210377.3210382

[41] Alejandro López-Ortiz and Alejandro Salinger. 2012. Minimizing Cache Usage in Paging. In *Proc. 10th Workshop on Approximation and Online Algorithms (WAOA)*.

[42] Alejandro López-Ortiz and Alejandro Salinger. 2012. Paging for multi-core shared caches. In *Proc. Innovations in Theoretical Computer Science (ITCS)*. 113–127.

[43] Paul Menage. [n.d.]. CGROUPS. https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt

[44] Richard Tran Mills. 2004. *Dynamic adaptation to CPU and memory load in scientific applications.* Ph.D. Dissertation. The College of William and Mary.

[45] Richard T Mills, Andreas Stathopoulos, and Dimitrios S Nikolopoulos. 2004. Adapting to memory pressure from within scientific applications on multiprogrammed COWs. In *Proc. 8th International Parallel and Distributed Processing Symposium (IPDPS)*. 71.

[46] Khang T Nguyen. [n.d.]. Introduction to Cache Allocation Technology in the Intel Xeon Processor E5 v4 Family. https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology

[47] HweeHwa Pang, Michael J. Carey, and Miron Livny. 1993. Memory-Adaptive External Sorting. In *Proc. 19th International Conference on Very Large Data Bases (VLDB)*. Morgan Kaufmann, 618–629.

[48] HweeHwa Pang, Michael J Carey, and Miron Livny. 1993. Partially Preemptible Hash Joins. In *Proc. 5th ACM SIGMOD International Conference on Management of Data (COMAD)*. 59.

[49] J-S Park, Michael Penner, and Viktor K Prasanna. 2004. Optimizing graph algorithms for improved cache performance. *IEEE Transactions on Parallel and Distributed Systems* 15, 9 (2004), 769–782.

[50] Enoch Peserico. 2013. Paging with dynamic memory capacity. *CoRR* abs/1304.6007 (2013).

[51] H. Prokop. 1999. *Cache Oblivious Algorithms.* Master's thesis. Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

[52] Aaron Schild, Satish Rao, and Nikhil Srivastava. 2018. Localization of Electrical Flows. In *SODA*. SIAM, 1577–1584.

[53] Raimund Seidel. 1995. On the all-pairs-shortest-path problem in unweighted undirected graphs. *Journal of computer and system sciences* 51, 3 (1995), 400–403.

[54] Avi Shoshan and Uri Zwick. 1999. All pairs shortest paths in undirected graphs with integer weights. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*. 605–614.

[55] Volker Strassen. 1969. Gaussian elimination is not optimal. *Numerische mathematik* 13, 4 (1969), 354–356.

[56] Yuan Tang, Rezaul Alam Chowdhury, Bradley C. Kuszmaul, Chi-Keung Luk, and Charles E. Leiserson. 2011. The pochoir stencil compiler. In *SPAA*. 117–128.

[57] Jue Wang, Xiangyu Dong, Yuan Xie, and Norman P Jouppi. 2014. Endurance-aware cache line management for non-volatile caches. *ACM Transactions on Architecture and Code Optimization (TACO)* 11, 1 (2014), 4.

[58] David Williams. 1991. *Probability with martingales*. Cambridge University Press.

[59] Virginia Vassilevska Williams. 2012. Multiplying matrices faster than Coppersmith-Winograd. In *In Proc. 44th ACM Symposium on Theory of Computation*. Citeseer.

[60] Virginia Vassilevska Williams and Ryan Williams. 2010. Subcubic equivalences between path, matrix and triangle problems. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*. 645–654.

[61] Sung-Eui Yoon, Peter Lindstrom, Valerio Pascucci, and Dinesh Manocha. 2005. Cache-oblivious mesh layouts. 24, 3 (2005), 886–893.

[62] Kamen Yotov, Tom Roeder, Keshav Pingali, John Gunnels, and Fred Gustavson. 2007. An experimental comparison of cache-oblivious and cache-conscious programs. In *Proc. 19th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*. 93–104.

[63] Neal E. Young. 2002. On-Line File Caching. *Algorithmica* 33, 3 (2002), 371–383.

[64] Hansjörg Zeller and Jim Gray. 1990. An adaptive hash join algorithm for multiuser environments. In *Proc. 16th International Conference on Very Large Data Bases (VLDB)*. 186–197.

[65] Weiye Zhang and Per-Äke Larson. 1996. A memory-adaptive sort (MASORT) for database systems. In *Proc. 6th International Conference of the Centre for Advanced Studies on Collaborative research (CASCON)* (Toronto, Ontario, Canada). IBM Press, 41–.

[66] Weiye Zhang and Per-Äke Larson. 1997. Dynamic Memory Adjustment for External Mergesort. In *Proc. 23rd International Conference on Very Large Data Bases (VLDB)*. Morgan Kaufmann Publishers Inc., 376–385.

[67] Uri Zwick. 1998. All pairs shortest paths in weighted directed graphs-exact and almost exact algorithms. In *Proceedings 39th Annual Symposium on Foundations of Computer Science (Cat. No. 98CB36280)*. 310–319.

Appendix

A CACHE-ADAPTIVITY OF RANDOMLY SHUFFLED PROFILES

Consider an (a, b, c)-regular algorithm, where $a > b$ are constants in $\mathbb{N}$ and $0 < c \leq 1$. Let Σ be a probability distribution over box sizes, and consider a sequence of boxes $(\square_1, \square_2, \square_3, \ldots)$ with sizes drawn independently from the distribution Σ. (Note that each $|\square_i|$ is a random variable.) The goal of this section will be to prove that for any such algorithm, and for any distribution of box sizes Σ, the algorithm will be cache-adaptive (in expectation) on the random sequence of boxes $(\square_1, \square_2, \ldots)$.

Remark 3. *Theorem 1 does not hold in the case of $a = b$. Consider, for example, the case of $a = b = 2$ and $c = 1$ (e.g., mergesort) with block size $B = 1$. Moreover, suppose each scan in each problem of size s occurs at the end of the problem, and consists of exactly s distinct block accesses, one after another. Finally, suppose Σ contains only one box-size $\sqrt{n}$.*

Then the potential of a box is $O(\sqrt{n})$, since each subproblem of size $\Theta(\sqrt{n})$ contains $\Theta(\sqrt{n})$ recursive leaves. In order for cache-adaptivity to be achieved, it follows that $\mathcal{S}_n$ (which is deterministically determined since there is only one box-size) must be $O(\sqrt{n})$, that way $\mathcal{S}_n \cdot \Theta(\sqrt{n})$ will be within a constant factor of the total progress n to be made for the full problem. However, every scan of size $s \geq 2\sqrt{n}$ requires $\Omega(s/\sqrt{n})$ boxes devoted entirely to that scan in order to be completed. Since the sum of the sizes of the scans at each level of recursion is n, it follows that the sum of the sizes of all scans of size at least $2\sqrt{n}$ is $\Theta(n \log n)$. Hence the total number of boxes required is at least

$$\mathcal{S}_n \geq \Omega(n \log n) \cdot \frac{1}{\sqrt{n}} = \Omega(\sqrt{n} \log n),$$

meaning the algorithm is not cache-adaptive in expectation.

Before continuing, we introduce some notation. Throughout the section, let $\square$ denote a single box whose size is drawn from the distribution Σ. As a convention, we will use $\mathcal{S}_n$ to denote the index of the final box $\square_{\mathcal{S}_n}$ required for the algorithm to complete on an input of size n (in blocks). (If different inputs to problems of size n require different numbers of boxes, then we define $\mathcal{S}_n$ to be the maximum over all possible inputs.) The quantity $\mathcal{S}_n$ is sometimes also referred to as the ***stopping time***.

For each problem-size n, define the ***work function*** $W_n = n^{\log_b a}$, the number of base-case recursive leaves in a problem of size n. We define the ***n-bounded potential*** $m_n(s)$ of a box of size s to be $\min(W_n, W_s)$, corresponding (up to a constant factor) with the size of the largest subproblem that the box can complete within a problem of size n. We will also sometimes use m_n to denote $\mathbb{E}[m_n(|\square|)]$.

With this notation, cache-adaptivity in expectation reduces to the statement

$$\mathbb{E}\left[\sum_{i=1}^{\mathcal{S}_n} m_n(|\square_i|)\right] \leq O(W_n).$$

The remainder of the section is devoted to proving Theorem 1. In Appendix A.1, we reduce the proof of Theorem 1 to a simpler, more specialized version of the theorem. The specialized version of the theorem is then proven in Appendix A.2.

A.1 A Simplified Problem

In this section, we present a series of simplifications to the problem of proving Theorem 1. Each simplification builds on the prior ones.

Recall that the algorithm $\mathcal{A}$ is cache-adaptive in expectation if, for all problem sizes n,

$$\mathbb{E}\left[\sum_{i=1}^{\mathcal{S}_n} m_n(|\square_i|)\right] \leq O(W_n). \tag{12}$$

We begin with a lemma that provides a simpler formulation of the quantity on the left side of Equation 12.

Lemma 4. *For any box-size distribution Σ, and any (a, b, c)-regular algorithm $\mathcal{A}$,*

$$\mathbb{E}\left[\sum_{i=1}^{\mathcal{S}_n} m_n(|\square_i|)\right] = \mathbb{E}[\mathcal{S}_n] \cdot m_n.$$

The lemma follows immediately from a specialized variant of the Martingale Optional Stopping Theorem [58]. We include the proofs of Lemma 4 and Theorem 4 in Appendix D.

Theorem 4 (Martingale Optional Stopping Theorem [58])**.** *Let $X_1, X_2, \ldots$ be iid random variables, and let γ be a function such that $\gamma(X_i)$ has finite mean μ. Consider an arbitrary process that runs in steps, and at each step i is given the value of X_i. Suppose that the process terminates after no more than C steps for some value C. Let S be the random variable denoting the number of steps that the process runs. Then,*

$$\mathbb{E}\left[\sum_{i=1}^{S} \gamma(X_i)\right] = \mathbb{E}[S] \cdot \mu.$$

This brings us to our first simplification:

Simplification 1. *To prove cache-adaptivity on a problem of size n, it suffices to show that*

$$\mathbb{E}[\mathcal{S}_n] \leq O\left(\frac{W_n}{m_n}\right).$$

Our second simplification ▷, which is a consequence of Lemma 11 in Appendix F, has to do with the structure of scans within subproblems:

Simplification 2. *We may restrict ourselves to (a, b, c)-regular algorithms in which scans occur exclusively at the end of subproblems (rather than before or between recursions), with the exception of the largest subproblem, which may also perform scan work at the beginning of the algorithm.*

We will refer to the scan work at the beginning of the algorithm, due to the largest problem, as the ***upfront scan***. As a convention, when we refer to the ***scan*** in a given subproblem, we will by default always be referring to the scan at the end of the subproblem, and not including the upfront scan, even if the subproblem we are discussing is the full problem on which the algorithm is running.

The next observation has to do with what we call the hard-stopping rule, which can be applied to any (a, b, c)-regular algorithm $\mathcal{A}$ satisfying the property from Simplification 2 (that scans appear at the ends of subproblems, with the exception of an upfront scan). Let $q \in O(1)$ be the smallest positive power of b (depending on the(a, b, c)-regular algorithm $\mathcal{A}$) such that any subproblem of any size s (ignoring any upfront scan) accesses at most $q \cdot s$ distinct blocks, such that the scan at the end of any subproblem of any size s can be completed by any sequence of boxes (with sizes sufficiently large in $\Omega(1)$) whose sizes sum to at least $q \cdot s$, and such that the upfront scan in any problem of any size n can also be completed by any such sequence of boxes. Notice that this implies that any box of size qs or larger can complete any subproblem of size s (ignoring the upfront scan). The ***hard-stopping rule*** is a modification to the manner in which the algorithm $\mathcal{A}$ is executed on a problem of size n:

- The upfront scan is simulated as being of length exactly $q \cdot n$. The upfront scan is complete after l boxes for the smallest l such that $\sum_{i=1}^{l} |\square_i| \geq q \cdot n$. The l-th box is not allowed to continue past the upfront scan (i.e., the box is not allowed to perform any additional memory accesses after the scan), and the $(l + 1)$-th box begins immediately after the upfront scan.
- Once the upfront scan is complete, whenever a box of some size t is generated within a subproblem of size $\frac{t}{q}$ or smaller, the box continues to the end of the largest subproblem of size $\frac{t}{q}$ or smaller that contains the box. The box stops at the end of that subproblem (i.e., the next subproblem begins using the next box).
- Finally, scans at the end of subproblems of each size s are simulated to always be of length $q \cdot s$. In particular, if the scan is not completed by some box of size qs or larger (possibly generated within the scan), then the scan requires a sequence of boxes whose sizes sum to $q \cdot s$ or larger in order for the scan to complete. The final of these boxes stops at the end of the scan, and the next box generated then starts at the beginning of whatever follows the scan.

As permitted by Theorem 1, we will assume that boxes have sizes large enough in $\Omega(1)$ that whenever their sizes sum to $q \cdot s$, they can complete any scan in a problem of size s (or an upfront scan in a problem of size s). We will also assume, in general, that every box in the distribution Σ has size large enough to complete any base-case problem while following the hard-stopping rule. (That is, if the largest size a base-case problem can be is n_0, then every box has size at least $q \cdot n_0$.). Thus every time a box is drawn, its behavior is guaranteed to be covered by one of the cases described above. Note that these assumptions on box sizes, which we will call the ***base-box-size assumption***, is permitted by the statement of Theorem 1, which allows us to require box sizes be at least arbitrarily large constants.

The hard-stopping rule has a number of appealing properties. The number of squares needed to complete a given algorithm on a problem of size n becomes a function of n only (and is unaffected by the algorithm input). Moreover, boxes always either complete some subproblem in its entirety, or finish in the same scan in which they began.

Note that the hard-stopping rule, in general, only inhibits the work completed in a box. In particular, any box of size s is always capable of completing any subproblem of size s/q or smaller, and any sequence of boxes (with sizes sufficiently large in $\Omega(1)$) whose sizes sum to $q \cdot s$ or larger are always capable to completing a scan of size s. Thus the hard-stopping rule is a modification of the execution of the algorithm such that boxes are sometimes asked to complete before they would normally need to.

Intuitively, when an execution follows the hard-stopping rule, the number of boxes for the algorithm to complete cannot decrease. This can be formalized by repeated applications of the No-Catchup Lemma (Lemma 2). In particular, each time that a box terminates early, rather than continuing, the No-Catchup Lemma tells us that the number of additional boxes needed to complete the problem (without following the hard-stopping rule) cannot decrease.

We use $\mathcal{S}_n^h$ as the random variable denoting the number of boxes needed to complete a problem of size n, given that the execution is following the hard-stopping rule. The key observation is that $\mathcal{S}_n^h \geq \mathcal{S}_n$. This brings us to our fifth simplification:

Simplification 3. *In order to prove that $\mathbb{E}[\mathcal{S}_n] \leq O(W_n/m_n)$, it suffices to prove that $\mathbb{E}[\mathcal{S}_n^h] \leq O(W_n/m_n)$, while making the base-box-size assumption.*

In our next simplification, we exploit the hard-stopping rule in order to remove upfront scans from the problem. In particular, consider an (a, b, c)-regular algorithm $\mathcal{A}$ on a problem of size n with scans only at the ends of subproblems, and one upfront scan at the beginning of the full problem. (Moreover, suppose $\mathcal{A}$ satisfies the base-box-size assumption when executed with the hard-stopping rule.) When bounding $\mathbb{E}[\mathcal{S}_n]$, we may assume without loss of generality that, in $\mathcal{A}$, the final scan at the end of the full problem contains as a suffix a copy of the upfront scan (since appending the upfront scan to the final scan can only increase $\mathbb{E}[\mathcal{S}_n]$). Moreover, rather then bounding $\mathbb{E}[\mathcal{S}_n]$, Simplification 3 allows us to instead focus on bounding $\mathbb{E}[\mathcal{S}_n^h]$ for $\mathcal{A}$. Let $\mathcal{B}$ be the same algorithm except with the scan at the front of $\mathcal{A}$ removed. (Notice that the value q used for $\mathcal{B}$ when following the hard-stopping rule will be the same as the value used for $\mathcal{A}$.) Any sequence of boxes which would complete B while following the hard-stopping rule would also complete the scan at the upfront of $\mathcal{A}$. (Indeed, even the boxes that complete the final scan in $\mathcal{B}$ suffice to complete the upfront scan of $\mathcal{A}$.) Thus if $\mathcal{S}_n^h$ denotes the number of boxes to complete $\mathcal{B}$ while following the hard-stopping rule, then the expected number of boxes to complete the upfront scan in $\mathcal{A}$ (while following the hard-stopping rule) is at most $\mathbb{E}[\mathcal{S}_n^h]$. The expected total number of boxes to complete $\mathcal{A}$ (while following the hard stopping rule) is therefore at most $2\mathbb{E}[\mathcal{S}_n^h]$. This brings us to our next simplification, which extends Simplification 3:

Simplification 4. *In order to prove that $\mathbb{E}[\mathcal{S}_n] \leq O(W_n/m_n)$ (i.e., prove Theorem 1), it suffices to consider only algorithms in which all scans (including those in the largest subproblem) occur exclusively at the end of their respective subproblems, and to then prove that $\mathbb{E}[\mathcal{S}_n^h] \leq O(W_n/m_n)$, while making the base-box-size assumption.*

Our next simplification is that we may restrict ourselves to problem sizes n which are powers of b. In particular, for any algorithm $\mathcal{A}$ run on a problem of size n that is not a power of b, we can define

$r < b$ such that $n \cdot r$ is a power of b, and then define $\mathcal{B}$ to be the same algorithm except with the problem size of $\mathcal{B}$ formally defined to be r times the corresponding problem size in $\mathcal{A}$.[11] The value of $\mathcal{S}_n^h$ for $\mathcal{A}$ is the same as the values of $\mathcal{S}_{nr}^h$ for $\mathcal{B}$, and the values of m_n and W_n for $\mathcal{A}$ are within a constant factor of the values of m_{nr}, and W_{nr} for $\mathcal{B}$. Thus if we prove that $\mathbb{E}[\mathcal{S}_{nr}^h] \le O(W_{nr}/m_{nr})$ for $\mathcal{B}$, then we will have proven that $\mathbb{E}[\mathcal{S}_n^h] \le O(W_n/m_n)$ for $\mathcal{A}$.

Simplification 5. *We may further assume without loss of generality that all problem sizes are powers of b.*

It will be convenient for the value q used in the hard-stopping rule to always be $q = 1$. Next we show that this may be assumed without loss of generality. Consider a (a, b, c)-regular algorithm $\mathcal{A}$, with scans only at the ends of subproblems, which is executed on a problem of size n using the hard-stopping rule with some value $q = t$ and using some sequence of box-sizes $(S_1, S_2, \ldots)$ satisfying the base-box-size assumption.

Then consider an (a, b, c)-regular algorithm $\mathcal{A}'$ which is executed on a problem of size nt using the hard-stopping rule with $q = 1$, and with base-case problem-size one. (Note that because q is always a power of b, multiplying n by q does not change whether n is a power of b.) The key insight is that, due to the hard-stopping rule, a given box will interact with problems of size s in the execution of $\mathcal{A}$, in precisely the same way that the same box interacts with problems of size $s \cdot t$ in the the execution of $\mathcal{A}'$. Hence the number of boxes used to complete the second execution will be precisely equal to the number used to complete the first.

Now let Σ be a distribution of box sizes satisfying the base-box-size assumption for $\mathcal{A}$. Suppose that $\mathcal{S}_{nt}^h \le O(W_{nt}/m_{nt})$ for algorithm $\mathcal{A}'$ on distribution Σ. Since $t \in O(1)$, and since $\mathcal{S}_{nt}^h$ for $\mathcal{A}'$ equals $\mathcal{S}_n^h$ for $\mathcal{A}$, it follows that $\mathcal{S}_n^h \le O(W_n/m_n)$ for algorithm $\mathcal{A}$ on distribution Σ. Hence we have the following simplification:

Simplification 6. *We may further assume without loss of generality that $q = 1$. Additionally, we may assume that the base-case problem size is 1.*

For the rest of the section, since $q = 1$, we will adapt the convention of saying that the ***size of a scan*** is simply the same as the size of the subproblem it is in (i.e., the size of a scan in a subproblem of size l is simply l).

Simplifications 5 and 6 combine to give us one final simplification: that without loss of generality, all box sizes are powers of b. In particular, since the problem sizes are powers of b, and since $q = 1$, the hard-stopping rule does not distinguish between boxes of different sizes in the range $[b^r, b^{r+1})$. Rounding each box size down to the nearest power of b changes m_n by at most a constant factor and does not change $\mathbb{E}[\mathcal{S}_n^h]$ or W_n. Thus we get our final simplification:

Simplification 7. *We may further assume without loss of generality that all box sizes are powers of b.*

In light of Simplifications 6 and 7, the hard-stopping rule has a very simple interpretation: any box of size s which is started in a problem of size s or smaller continues to the end of whatever problem of size s it's in; and a box of size s started in the scan of size $l > s$ finishes within the same scan, and completes the scan if and only if the sum of the sizes of the boxes started within the scan sum to l or greater.

Combining these simplifications, we get that in order to prove Theorem 1, it suffices to prove the following specialized version:

Theorem 5. *Let $a > b$ be constants in $\mathbb{N}$ and consider $c \in [0, 1]$. Consider an (a, b, c)-regular algorithm $\mathcal{A}$, in which all scans occur exclusively at the ends of subproblems, and suppose $\mathcal{A}$ is executed with the hard-stopping rule using $q = 1$. Then for any problem size n that is a power of b, and for any box-size distribution Σ in which all boxes have sizes that are powers of b,*

$$\mathbb{E}[\mathcal{S}_n^h] \le O(W_n/m_n).$$

The remainder of the section is devoted to proving Theorem 5.

A.2 Proof of Theorem 5

We begin by introducing some additional notation. Let $f(n)$ denote $\mathbb{E}[\mathcal{S}_n^h]$, the expected number of boxes needed to complete a problem of size n (while following the hard-stopping rule). Note that $f(n)$ is well defined since the number of boxes needed to complete a problem while following the hard-stopping rule is a function of n only. In order to prove cache-adaptivity for problems of size n, we wish to prove that

$$f(n) \le O(W_n/m_n). \tag{13}$$

(Recall that W_n is the work function, satisfying $W_n = n^{\log_b a}$, and m_n is the expectation of the n-bounded potential function, satisfying $m_n = \mathbb{E}[\min(W_n, W_{|\square|})]$, where $\square$ is a box of size drawn from the distribution Σ.)

Intuitively, Equation 13 says that the ***typical progress*** of a box is m_n, allowing for roughly W_n/m_n boxes to complete the problem, on average. Formally, we say that the ***typical progress*** of a box is given by

$$\phi(n) = W_n/f(n),$$

the amount of work in a problem of size n divided by the average number of boxes needed. In order to prove cache-adaptivity in expectation, we therefore wish to show that

$$\phi(n) \ge \Omega(m_n),$$

for all problem sizes n.

A natural approach might be to prove this by induction on n. (Recall that the eligible problem sizes n are the powers of b.) In particular, it would suffice to prove the relationship

$$\frac{\phi(n)}{\phi(n/b)} \ge \frac{m_n}{m_{n/b}}. \tag{14}$$

Unfortunately, Equation 14 does not always hold, as shown in the following example.

Example 1. *Suppose that the box sizes $|\square|$ take some value l deterministically. Then the expected number of boxes $f(l)$ to complete a problem of size l will be one. But the expected number of boxes $f(b \cdot l)$ to complete a problem of size $b \cdot l$ will be $a + b$. In particular, a boxes*

[11] Note that $\mathcal{B}$'s base case for recursion happens on problems r times as large as in $\mathcal{A}$. This is acceptable since $r \le O(1)$. Also note that if a distribution Σ of box-sizes for $\mathcal{A}$ satisfies the base-box-size assumption, then so will the same distribution for $\mathcal{B}$, since the base-case problems in both algorithms use the same block-access patterns.

are used to solve the a subproblems, and b boxes are used for the scan. This means that $\phi(l \cdot b)/\phi(l)$ will be

$$\frac{\phi(l \cdot b)}{\phi(l)} = \frac{W_{lb}/f(lb)}{W_l/f(l)} = \frac{W_{lb}}{W_l} \cdot \frac{f(l)}{f(lb)} = a \cdot \frac{1}{a+b} < 1.$$

Note that in the final inequality we use the useful fact that $\frac{W_{lb}}{W_l} = \frac{(lb)^{\log_b a}}{l^{\log_b a}} = a$.

On the other hand, m_l and m_{lb} will both be W_l (since all boxes are size l), meaning that $\frac{m_{lb}}{m_l} = 1$, and thereby violating Equation 14. The reason for the violation is intuitively that the long scan in the problem of size $b \cdot l$ causes the typical progress of a box to shrink (when compared to a problem of size b), while the average bounded potential is left unchanged.

The next lemma shows that Example 1 is extremal in the sense that one will always have $\frac{\phi(l \cdot b)}{\phi(l)} \geq \frac{a}{a+b}$.

Lemma 5. *For all problem sizes l, $\frac{f(l \cdot b)}{f(l)} \leq a + b$. Consequently, $\frac{\phi(l \cdot b)}{\phi(l)} = \frac{W_{lb}/f(lb)}{W_l/f(l)} \geq \frac{a}{a+b}$.*

Proof. Note that the expected number of boxes needed to complete a subproblems of size l is at most $a \cdot f(l)$. Moreover, a scan of size $b \cdot l$ can require at most b times as many boxes to complete (in expectation) as does a scan of size l. Since a subproblem of size l contains a scan of size l, it follows that a scan of size $b \cdot l$ requires at most $b \cdot f(l)$ boxes to complete (in expectation).[12] In total, we get that $f(l \cdot b) \leq af(l) + bf(l)$, as desired. □

To resolve the issue in Example 1, one can separate the analysis of each scan from the analysis of the subproblems preceding the scan. Define $f'(n)$ to be the expected number of boxes needed to complete a problems of size n/b one after another. (i.e., $f'(n)$ is the expected number of boxes to complete a problem of size n, while ignoring the final scan.) Similarly, define $\phi'(n) = W_n/f'(n)$. Rather than proving Equation 14, which Example 1 proves false, one could instead attempt to show that

$$\frac{\phi'(n)}{\phi(n/b)} \geq \frac{m_n}{m_{n/b}}. \tag{15}$$

Then, rather than individually considering $\phi(n)/\phi'(n)$ for each problem size n, one could instead bound the total impact of scans across all problem sizes on the typical progress function, proving that

$$\prod_{b^t \leq n} \phi(b^t)/\phi'(b^t) \geq \Omega(1). \tag{16}$$

In particular, the goal would be to prove that, although the scans at any particular level of recursion can bring down the typical progress of a box by as much as a constant factor, in aggregate the scans across all levels of recursion bring the typical progress of a box down by no more than a constant factor.

Combined, Equation 15 and Equation 16 would suffice for proving cache adaptivity in expectation. (In particular, when combined, they imply that $\phi(n) \geq \Omega(m_n)$.) The approach we take differs from this in one additional important way. Rather than proving Equation 15 for all n (that are powers of b), we instead restrict ourselves to values of n for which $\phi(n/b)$ is on the cusp of being too small for cache adaptivity (meaning $\phi(n/b)/m_{n/b}$ is a sufficiently small constant). For these values, Equation 15 can be viewed as a sort of negative feedback loop, showing that whenever $\phi(n)/m_n$ starts to become small, there is upward pressure so that $\phi'(n \cdot b)/m_{n \cdot b}$ does not become even smaller. Formally, we will prove the following propositions, the proofs of which we will present in the coming subsections.

Proposition 1. *Consider a problem size $l > b$, and suppose l/b satisfies $\phi(l/b) < \frac{m_{l/b}}{4a}$. Then,*

$$\frac{\phi'(l)}{\phi(l/b)} \geq \frac{m_l}{m_{l/b}}.$$

Proposition 2. *Consider the largest problem size $l \leq n$ for which $\phi(l) \geq \frac{m_l}{4a}$. Then,*

$$\prod_{l < b^t \leq n} \phi(b^t)/\phi'(b^t) \geq \Omega(1).$$

In fact, we will not need Proposition 2 in its full strength. Rather, we will use the nearly equivalent fact that

$$\prod_{b \cdot l < b^t \leq n} \phi(b^t)/\phi'(b^t) \geq \Omega(1),$$

which is implied by Proposition 2 and the observation that $\phi(b \cdot l)/\phi'(b \cdot l) \leq 1$. Assuming the propositions, Theorem 5 can be proven as follows.

Proof of Theorem 5. Suppose we wish to prove cache adaptivity on problems of size n. Consider the largest subproblem size l for which $\phi(l) \geq \frac{m_l}{4a}$. (Note that $l = 1$ necessarily satisfies this property since $\phi(1) = W_1/f(1) = 1$ and $\frac{m_1}{4a} = \frac{1}{4a}$.) If $l = n$, then $\phi(n) \geq \Omega(m_n)$, which proves cache adaptivity. On the other hand, if $l < n$, then we can express $\phi(n)$ as

$$\phi(n) = \phi(b \cdot l) \cdot \prod_{b \cdot l < b^t \leq n} \frac{\phi'(b^t)}{\phi(b^{t-1})} \cdot \frac{\phi(b^t)}{\phi'(b^t)}.$$

By Proposition 1 (which holds for all problem sizes greater than $b \cdot l$) and Proposition 2, this becomes

$$\begin{aligned}\phi(n) &\geq \Omega\left(\phi(b \cdot l) \cdot \prod_{b \cdot l < b^t \leq n} \frac{m_{b^t}}{m_{b^{t-1}}}\right) \\ &= \Omega\left(\phi(b \cdot l) \cdot \frac{m_n}{m_{b \cdot l}}\right).\end{aligned}$$

Recall that our goal is to establish that the typical progress $\phi(n)$ is at least $\Omega(m_n)$, the average n-bounded potential of a box. To complete the proof, it therefore suffices to show that $\phi(b \cdot l) \geq \Omega(m_{bl})$. Recall that, by the definition of l, $\phi(l) = \frac{W_l}{f(l)} \geq \frac{m_l}{4a}$, meaning that $f(l) \leq 4a \cdot W_l/m_l$. Since $W_{bl} = a \cdot W_l$, and since $m_{bl} \leq a \cdot m_l$ (in particular, $\min(W_{|\square|}, W_l) \leq a \cdot \min(W_{|\square|}, W_{bl})$), it follows that $f(l) \leq 4a \cdot W_{lb}/m_{lb}$. By Lemma 5, we then get that $f(l \cdot b) \leq (a+b) \cdot 4a \cdot W_{bl}/m_{bl}$, and thus $\phi(bl) \geq \frac{m_{bl}}{4a \cdot (a+b)} \geq \Omega(m_{bl})$, as desired. □

[12] Here we are implicitly using the fact that for any sequence of boxes, the hard-stopping rule allows to complete a problem of size l, the hard-stopping rule would have also allowed them to complete a single scan of size l.

Name	Symbol	Expansion
work function	W_n	$n^{\log_a b}$
n-bounded potential	m_n	$\mathbb{E}[\min(W_n, W_{\lvert\square\rvert})]$
expected stopping time	$f(n)$	–
expected stopping time ignoring final scan	$f'(n)$	–
typical progress	$\phi(n)$	$W_n/f(n)$
typical progress ignoring final scan	$\phi'(n)$	$W_n/f'(n)$

Figure 3: A reference table of the functions used to analyze cache-adaptivity in expectation.

The remainder of the section is devoted to proving Propositions 1 and 2. For reference by the reader, Figure 3 gives a table of the functions defined in this section that continue to be used in the proofs of the propositions.

A.2.1 Proof of Proposition 1. We begin by comparing the average l-bounded potential m_l to the average l/b-bounded potential $m_{l/b}$, assuming $\phi(l/b) < \frac{m_{l/b}}{4a}$.

Lemma 6. *Suppose l/b satisfies $\phi(l/b) < \frac{m_{l/b}}{4a}$. Then,*

$$\frac{m_l}{m_{l/b}} < 1 + \frac{\Pr[|\square| \geq l] \cdot f(l/b)}{4}.$$

Proof. The only box sizes s for which the l-bounded potential differs from the l/b-bounded potential are the sizes $s \geq l$. In particular,

$$m_l - m_{l/b} = \Pr[|\square| \geq l] \cdot (W_l - W_{l/b}) \leq \Pr[|\square| \geq l] \cdot W_l.$$

Hence

$$\frac{m_l - m_{l/b}}{m_{l/b}} \leq \frac{\Pr[|\square| \geq l] \cdot W_l}{m_{l/b}}.$$

Because $\phi(l/b) = W_{l/b}/f(l/b) < \frac{m_{l/b}}{4a}$, it follows that

$$\begin{aligned}\frac{m_l - m_{l/b}}{m_{l/b}} &< \frac{\Pr[|\square| \geq l] \cdot W_l}{4aW_{l/b}/f(l/b)} \\ &= \frac{\Pr[|\square| \geq l] \cdot aW_{l/b}}{4aW_{l/b}/f(l/b)} \\ &= \frac{\Pr[|\square| \geq l] \cdot f(l/b)}{4}.\end{aligned}$$

Adding one to both sides,

$$\frac{m_l}{m_{l/b}} < 1 + \frac{\Pr[|\square| \geq l] \cdot f(l/b)}{4}.$$

□

Next we compare the typical progress $\phi'(l)$ of a box in a problem of size l (ignoring the scan) to the typical progress $\phi(l/b)$ of a box in a problem of size l/b. Note that

$$\begin{aligned}\phi'(l)/\phi(l/b) &= \frac{W_l/f'(l)}{W_{l/b}/f(l/b)} \\ &= \frac{af(l/b)}{f'(l)}.\end{aligned}$$

Thus in order to prove a lower bound for $\phi'(l)/\phi(l/b)$ (i.e., to show that the typical progress has increased between problem sizes), it suffices to compare $f(l/b)$ and $f'(l)$.

Lemma 7. *Consider any problem size $l \geq b$. Then,*

$$f'(l) \leq a \cdot f(l/b) \cdot (1 - f(l/b) \cdot \Pr[|\square| \geq l]/2).$$

Recall that $f'(l)$ is the expected number of boxes needed to complete a consecutive problems of size l/b, while $f(l/b)$ is the expected number of boxes needed to complete a single such problem.

Proof. Consider what happens when the algorithm $\mathcal{A}$ is run on a problem of size l/b (while following the hard-stopping rule as in Theorem 5). Define p to be the probability that a box of size l or greater is generated during the problem. By the Martingale Optional Stopping Theorem (Theorem 4), the expected number of such boxes generated is $f(l/b) \cdot \Pr[|\square| \geq l]$. (In this application of Theorem 4, the variables X_i are defined to be $X_i = |\square_i|$, and $\gamma(X_i)$ is the indicator variable $\mathbb{I}(|\square_i| \geq l)$.) Note that no more than one such box can be generated, however, since any box of size l or greater will complete the problem of size l/b. Thus the probability p of such a box being generated is equal to the expected number of such boxes, and

$$p = f(l/b) \cdot \Pr[|\square| \geq l].$$

With this in mind, we consider $f'(l)$, the expected number of boxes needed to complete a problems of size l/b. The first of the a subproblems will require $f(l/b)$ boxes to complete, in expectation. With probability p, one of these boxes will be of size at least l, and will thus complete the entire computation. Otherwise, the second of the a subproblems will then require $f(l/b)$ boxes to complete, in expectation. Again, with probability p, one of these boxes will be of size at least l, and thus complete the entire computation. Continuing like this, the probability that the i-th subproblem is handled by a large box from a previous subproblem is $1 - (1-p)^{i-1}$, and thus the expected number of additional boxes needed to handle the i-th

subproblem is $(1-p)^{i-1} \cdot f(l/b)$. Summing over the subproblems,

$$\begin{aligned} f'(l) &= \sum_{i=1}^{a} (1-p)^{i-1} \cdot f(l/b) \\ &\le f(l/b) + \sum_{i=2}^{a} (1-p)^{i-1} \cdot f(l/b) \\ &\le a \cdot f(l/b) \cdot (1-p/2). \end{aligned}$$

Expanding p,

$$f'(l) \le a \cdot f(l/b) \cdot (1 - f(l/b) \cdot \Pr[|\square| \ge l]/2),$$

as desired. □

Combining Lemmas 6 and 7, we can now complete the proof of Proposition 1.

Proof of Proposition 1. Recall that $\frac{\phi'(l)}{\phi(l/b)}$ expands to

$$\frac{\phi'(l)}{\phi(l/b)} = \frac{af(l/b)}{f'(l)}.$$

By Lemma 7, it follows that

$$\frac{\phi'(l)}{\phi(l/b)} \ge \frac{1}{(1 - f(l/b) \cdot \Pr[|\square| \ge l]/2)} \ge 1 + f(l/b) \cdot \Pr[|\square| \ge l]/2.$$

Comparing this to Lemma 6, we see that

$$\frac{\phi'(l)}{\phi(l/a)} \ge \frac{m_l}{m_{l/b}},$$

as desired. □

A.2.2 Proof of Proposition 2. Proposition 2 is essentially about bounding the impact of scans on typical progresses. We begin by considering the number of boxes needed to perform a scan.

Define $Q(b^t)$ to be the expected number of boxes needed to run a scan of size b^t on its own (while following the hard-stopping rule). A box of a given size s could potentially make progress as much as $\min(b^t, s)$ through the scan. Thus one might intuitively expect $Q(b^t)$ to be roughly $\frac{b^t}{\mathbb{E}[\min(b^t, |\square|)]}$. The following lemma proves this up to a constant factor.

Lemma 8.

$$\frac{b^t}{\mathbb{E}[\min(b^t, |\square|)]} \le Q(b^t) < \frac{2b^t}{\mathbb{E}[\min(b^t, |\square|)]}.$$

Proof. Consider the sequence of box sizes $A_1, A_2, \ldots, A_S$ needed to complete the scan. (Here S is a random variable.) For the box A_S to complete the scan, the progress across the entire sequence must be

$$\sum_{i=1}^{S} \min(b^t, |A_i|) \ge b^t.$$

Moreover, since each box A_i except the final box A_S makes progress exactly $|A_i| = \min(b^t, |A_i|)$ in the scan,

$$\sum_{i=1}^{S} \min(b^t, |A_i|) \le b^t + \min(b^t, |A_S|) < 2b^t.$$

Since the quantity $\sum_{i=1}^{s} \min(b^t, |A_i|)$ is deterministically between b^t and $2b^t - 1$, its expectation must also be between b^t and $2b^t - 1$, satisfying

$$b^t \le \mathbb{E}\left[\sum_{i=1}^{S} \min(b^t, |A_i|)\right] < 2b^t.$$

On the other hand, the Martingale Optional Stopping Theorem (Theorem 4) tells us that

$$\mathbb{E}\left[\sum_{i=1}^{S} \min(b^t, |A_i|)\right] = \mathbb{E}[S] \cdot \mathbb{E}[\min(b^t, |\square|)] = Q(b^t) \cdot \mathbb{E}[\min(b^t, |\square|)].$$

Hence,

$$\frac{b^t}{\mathbb{E}[\min(b^t, |\square|)]} \le Q(b^t) < \frac{2b^t}{\mathbb{E}[\min(b^t, |\square|)]},$$

as desired. □

We now complete the proof of Proposition 2

Proof of Proposition 2. In order to prove that

$$\prod_{l < b^t \le n} \phi(b^t)/\phi'(b^t) \ge \Omega(1),$$

it suffices to prove that

$$\prod_{l < b^t \le n} f(b^t)/f'(b^t) \le O(1), \tag{17}$$

since $\phi(b^t)/\phi'(b^t) = f'(b^t)/f(b^t)$. We an restate Equation 17 as

$$\prod_{l < b^t \le n} \left(1 + \frac{f(b^t) - f'(b^t)}{f'(b^t)}\right) \le O(1).$$

Consider the scan at the end of a problem of size b^t. Let p the probability that a box of size at least b^t is generated during one of the a subproblems of size b^{t-1}. Then with probability p, the box generated during the a subproblems will complete the scan. On the other hand, with probability $(1-p)$, the scan will require additional boxes. It follows by Lemma 8 that

$$f(b^t) - f'(b^t) = (1-p) \cdot Q(b^t) < (1-p) \cdot \frac{2b^t}{\mathbb{E}[\min(b^t, |\square|)]}.$$

To prove the proposition, it therefore suffices to show that

$$\prod_{l < b^t \le n} \left(1 + (1-p) \cdot \frac{2b^t}{\mathbb{E}[\min(b^t, |\square|)] \cdot f'(b^t)}\right) \le O(1).$$

Equivalently, we wish to prove that

$$\sum_{l < b^t \le n} \ln\left(1 + (1-p) \cdot \frac{2b^t}{\mathbb{E}[\min(b^t, |\square|)] \cdot f'(b^t)}\right) \le O(1).$$

Since $\ln(1+x) \le x$ for all $x \ge 0$, we may instead prove that

$$\sum_{l < b^t \le n} \left((1-p) \cdot \frac{2b^t}{\mathbb{E}[\min(b^t, |\square|)] \cdot f'(b^t)}\right) \le O(1). \tag{18}$$

Let use take a moment to solve for p. Recall that within a problem of size b^t, p is the probability that a box of size at least b^t is generated during any one of the a subproblems of size b^{t-1}. By the Martingale Optional Stopping Theorem (Theorem 4), the expected number of such boxes generated is given by $\Pr[|\square| \ge b^t] \cdot f'(b^t)$. Moreover, at most one such box can be generated (since it will then

complete the problem of size b^t). Hence the number of such boxes is an indicator variable, and the probability p of such a box being generated is also $\Pr[|\square| \geq b^t] \cdot f'(b^t)$. Expanding p in Equation 18, the sum we wish to bound becomes

$$\sum_{l<b^t\leq n}\left((1-\Pr[|\square|\geq b^t]\cdot f'(b^t))\frac{2b^t}{\mathbb{E}[\min(b^t,|\square|)]\cdot f'(b^t)}\right)$$
$$=\sum_{l<b^t\leq n}\left(\frac{2b^t}{\mathbb{E}[\min(b^t,|\square|)]\cdot f'(b^t)}-\frac{2b^t\cdot\Pr[|\square|\geq b^t]}{\mathbb{E}[\min(b^t,|\square|)]}\right).$$

Next we focus on $f'(b^t)$. By the fact that $b^t > l$ and by the definition of l in the statement of the proposition, we know that $\phi(b^t) \leq \frac{m_{b^t}}{4a}$. Expanding $\phi(b^t)$, this means that $W_{b^t}/f(b^t) \leq \frac{m_{b^t}}{4a}$, and thus that

$$f(b^t) \geq \frac{4a\cdot W_{b^t}}{m_{b^t}} = \frac{4a^{t+1}}{m_{b^t}}. \tag{19}$$

In order to transform this into a statement about $f'(b^t)$, notice that $f'(b^t) \geq f(b^t)/2$. In particular, $f'(b^t)$ counts the number of boxes needed to complete a subproblems of size b^{t-1}. This includes a scans of size b^{t-1}. The expected number of boxes needed to complete these scans alone is at least the number of boxes needed to complete a scan of size $a\cdot b^{t-1} > b^t$. Since the hard-stopping rule guarantees that any sequence of boxes which completes the a subproblems could have also completed the a scans alone (without the additional portions of the subproblems), it follows that a subproblems of size b^{t-1} require, in expectation, at least as many boxes as a scan of length b^t, meaning that $f'(b^t) \geq f(b^t) - f'(b^t)$, and thus that $f'(b^t) \geq f(b^t)/2$. Combining this with Equation 19, we get that

$$f'(b^t) \geq \frac{2a^{t+1}}{m_{b^t}}.$$

Plugging this into our sum, it suffices to prove the bound

$$\sum_{l<b^t\leq n}\left(\frac{b^t\cdot m_{b^t}}{\mathbb{E}[\min(b^t,|\square|)]\cdot a^{t+1}}-\frac{2b^t\cdot\Pr[|\square|\geq b^t]}{\mathbb{E}[\min(b^t,|\square|)]}\right)\leq O(1).$$

Define $r_{b^u} = \Pr[|\square| = b^u]$ to be the probability of a box taking size b^u. Our sum expands to

$$\sum_{l<b^t\leq n}\left(\frac{b^t\cdot\left(\sum_{u<t} r_{b^u} W_{b^u}+\Pr[|\square|\geq b^t]\cdot W_{b^t}\right)}{\mathbb{E}[\min(b^t,|\square|)]\cdot a^{t+1}}-\frac{2b^t\cdot\Pr[|\square|\geq b^t]}{\mathbb{E}[\min(b^t,|\square|)]}\right)$$
$$=\sum_{l<b^t\leq n}\left(\frac{b^t\cdot\left(\sum_{u<t} r_{b^u} a^u+\Pr[|\square|\geq b^t]\cdot a^t\right)}{\mathbb{E}[\min(b^t,|\square|)]\cdot a^{t+1}}-\frac{2b^t\cdot\Pr[|\square|\geq b^t]}{\mathbb{E}[\min(b^t,|\square|)]}\right)$$
$$=\sum_{l<b^t\leq n}\left(\frac{b^t\cdot\sum_{u<t} r_{b^u} a^u}{\mathbb{E}[\min(b^t,|\square|)]\cdot a^{t+1}}+\frac{b^t\cdot\Pr[|\square|\geq b^t]}{\mathbb{E}[\min(b^t,|\square|)]\cdot a}-\frac{2b^t\cdot\Pr[|\square|\geq b^t]}{\mathbb{E}[\min(b^t,|\square|)]}\right).$$

Using half of the third sum to dominate the second sum, this is at most

$$\sum_{l<b^t\leq n}\left(\frac{b^t\cdot\sum_{u<t} r_{b^u} a^u}{\mathbb{E}[\min(b^t,|\square|)]\cdot a^{t+1}}-\frac{b^t\cdot\Pr[|\square|\geq b^t]}{\mathbb{E}[\min(b^t,|\square|)]}\right). \tag{20}$$

Focusing on the positive summands,

$$\sum_{l<b^t\leq n}\frac{b^t\cdot\sum_{u<t} r_{b^u} a^u}{\mathbb{E}[\min(b^t,|\square|)]\cdot a^{t+1}}$$
$$=\sum_{b^u<n}\ \sum_{b^u,l<b^t\leq n}\frac{b^t\cdot r_{b^u} a^u}{\mathbb{E}[\min(b^t,|\square|)]\cdot a^{t+1}}$$
$$=\sum_{b^u\leq l}\ \sum_{l<b^t\leq n}\frac{b^t\cdot r_{b^u} a^u}{\mathbb{E}[\min(b^t,|\square|)]\cdot a^{t+1}}+\sum_{l<b^u\leq n}\ \sum_{b^u<b^t\leq n}\frac{b^t\cdot r_{b^u} a^u}{\mathbb{E}[\min(b^t,|\square|)]\cdot a^{t+1}}$$
$$\leq\sum_{b^u\leq l}\ \sum_{l<b^t\leq n}\frac{b^t\cdot r_{b^u} a^u}{\mathbb{E}[\min(b^t,|\square|)]\cdot a^{t+1}}+\sum_{l<b^u\leq n}\ \sum_{b^u<b^t\leq n}\frac{b^t\cdot r_{b^u} a^u}{\mathbb{E}[\min(b^u,|\square|)]\cdot a^{t+1}}$$
$$=\sum_{b^u\leq l}\ \sum_{l<b^t\leq n}\frac{b^t\cdot r_{b^u} a^u}{\mathbb{E}[\min(b^t,|\square|)]\cdot a^{t+1}}+\sum_{l<b^u\leq n}\frac{b^u\cdot r_{b^u} a^u}{\mathbb{E}[\min(b^u,|\square|)]\cdot a^{u+1}}\cdot\sum_{b^u<b^t\leq n}(b/a)^{t-u}$$
$$<\sum_{b^u\leq l}\ \sum_{l<b^t\leq n}\frac{b^t\cdot r_{b^u} a^u}{\mathbb{E}[\min(b^t,|\square|)]\cdot a^{t+1}}+\sum_{l<b^u\leq n}\frac{b^u\cdot r_{b^u} a^u}{\mathbb{E}[\min(b^u,|\square|)]\cdot a^{u+1}}\cdot\sum_{s\geq 0}((a-1)/a)^s$$
$$=\sum_{b^u\leq l}\ \sum_{l<b^t\leq n}\frac{b^t\cdot r_{b^u} a^u}{\mathbb{E}[\min(b^t,|\square|)]\cdot a^{t+1}}+\sum_{l<b^u\leq n}\frac{b^u\cdot r_{b^u} a^u}{\mathbb{E}[\min(b^u,|\square|)]\cdot a^{u+1}}\cdot a$$
$$=\sum_{b^u\leq l}\ \sum_{l<b^t\leq n}\frac{b^t\cdot r_{b^u} a^u}{\mathbb{E}[\min(b^t,|\square|)]\cdot a^{t+1}}+\sum_{l<b^u\leq n}\frac{b^u\cdot r_{b^u}}{\mathbb{E}[\min(b^u,|\square|)]}.$$

Adding back in the negative terms from Equation 20, we get that Equation 20 is at most

$$\sum_{b^u\leq l}\ \sum_{l<b^t\leq n}\frac{b^t\cdot r_{b^u} a^u}{\mathbb{E}[\min(b^t,|\square|)]\cdot a^{t+1}}+\sum_{l<b^u\leq n}\frac{b^u\cdot r_{b^u}}{\mathbb{E}[\min(b^u,|\square|)]}-\sum_{l<b^t\leq n}\frac{b^t\cdot\Pr[|\square|\geq b^t]}{\mathbb{E}[\min(b^t,|\square|)]}.$$

Since $\Pr[|\square| \geq b^t] \geq r_{b^t}$, the third sum dominates the second sum, and thus we are left with at most

$$\sum_{b^u\leq l}\ \sum_{l<b^t\leq n}\frac{b^t\cdot r_{b^u} a^u}{\mathbb{E}[\min(b^t,|\square|)]\cdot a^{t+1}}.$$

Since $u < t$, we have $r_{b^u} b^u \leq \mathbb{E}[\min(b^t, |\square|)]$, and thus our expression is at most

$$\sum_{b^u\leq l}\ \sum_{l<b^t\leq n}\frac{b^{t-u}\cdot a^u}{a^{t+1}}$$
$$\leq\sum_{b^u\leq l}\frac{1}{a}\cdot\left(\frac{b}{a}\right)^{-u}\cdot\sum_{l<b^t}\left(\frac{b}{a}\right)^t.$$

If we define k such that $l = b^k$, then the sum can be rewritten as

$$\sum_{u\leq k}\frac{1}{a}\cdot\left(\frac{b}{a}\right)^{k-u}\cdot\sum_{t>0}\left(\frac{b}{a}\right)^t$$
$$<\sum_{u\leq k}\frac{1}{a}\cdot\left(\frac{b}{a}\right)^{k-u}\cdot\sum_{t\geq 0}\left(\frac{a-1}{a}\right)^t$$
$$=\sum_{u\leq k}\frac{1}{a}\cdot\left(\frac{b}{a}\right)^{k-u}\cdot a$$
$$=\sum_{u\leq k}\left(\frac{b}{a}\right)^{k-u}=O(1),$$

which completes the proof. □

B ROBUSTNESS OF WORST-CASE PROFILES

In this section, we consider three ways to smooth worst-case profiles: box-size perturbations, start time perturbations and, box-order perturbations. In all three cases the smoothed profiles will remain examples of profiles on which any $(a, b, 1)$-regular algorithm (where $a > b$), $\mathcal{A}_{a,b}$, is not adaptive. We show that the canonical worst-case profiles are robust to these perturbations and not brittle, by exploiting self-symmetry within worst-case profiles and the power of the No-Catchup Lemma. This is surprising because the canonical worst-case profile seems fragile—it gives $\mathcal{A}_{a,b}$ memory only when the algorithm can't use it, and gives as much memory as possible at those times.

Throughout the section, we examine a specific (a, b, c)-regular algorithm with constants $a, b \in \mathbb{N}$ satisfying $a > b$ and $c = 1$. We define the ***canonical* $(a, b, 1)$*-regular algorithm*** $\mathcal{A}_{a,b}(n)$ on problem-sizes n that are powers of b as follows: If $n > 1$, then the algorithm $\mathcal{A}_{a,b}(n)$ first recursively performs a subproblems of size n/b. Then (regardless of whether $n > 1$), the algorithm $\mathcal{A}_{a,b}(n)$ accesses each of the blocks $1, 2, \ldots, n$, one after another.

For n a power of b, the ***canonical worst-case profile*** $M_{a,b}(n)$ is the profile constructed so that each scan in the the algorithm $\mathcal{A}_{a,b}$ will be covered by a box exactly the size of that scan (i.e., the number of block-accesses in the scan). In particular, $M_{a,b}(n)$ consists of a single box of size 1 when $n = 1$, and otherwise recursively consists of a instances of $M_{a,b}(n/b)$ followed by a box of size n. We define $M_{a,b}$ to be the infinite profile containing $M_{a,b}(n)$ as a prefix for each n that is a power of b.

We say that a square profile M is ***worst-case*** for the algorithm $\mathcal{A}_{a,b}(n)$ if the sequence of boxes used to complete $\mathcal{A}_{a,b}(n)$, $M = \square_1, \ldots, \square_k$, satisfies

$$\sum_{i=1}^{k} \min(n, |\square_i|)^{\log_b a} \geq \Omega(\log n) \cdot n^{\log_b a}.$$

In particular, such a profile M ensures that the algorithm $\mathcal{A}$ is at least an $\Omega(\log n)$ factor off from cache-adaptive, which by the results of [5] make M a worst-case profile (up to a constant factor). Similarly, we say that a probability distribution $\mathcal{M}$ on square profiles is ***worst-case (in expectation)*** for the algorithm $\mathcal{A}_{a,b}(n)$ if for M randomly selected from $\mathcal{M}$, the sequence of boxes $(\square_1, \ldots, \square_k)$ used to complete M (note that now k is a random variable) satisfies

$$\mathbb{E}\left[\sum_{i=1}^{k} \min(n, |\square_i|^{\log_b a})\right] = \Omega(\log n) \cdot n^{\log_b a}.$$

When all the box-sizes considered are trivially of size n or smaller, as will often be the case in this section, we omit the minimum in the above sum.

Bender et al. [5] showed that $M_{a,b}$ is a worst-case profile for $\mathcal{A}_{a,b}(n)$ for all n. Notice, in particular, that an execution of $\mathcal{A}_{a,b}(n)$ on $M_{a,b}$ will use exactly the boxes in the prefix $M_{a,b}(n)$ of $M_{a,b}$, with each scan using a box of precisely the same size as the scan; and by induction on n, the sum of $|\square|^{\log_b a}$ over the boxes in $M_{a,b}(n)$ is $\log_b n \cdot n^{\log_b a}$.

In this section, we consider the robustness of the worst-case memory profile $M_{a,b}$ to three types of smoothing:

- **Box-size Perturbations:** In Appendix B.1 we consider what happens if each box in $M_{a,b}$ has its size randomly perturbed (i.e., multiplied by a value in $[0, 1]$ drawn from a distribution $\mathcal{P}$ that has constant expectation). We show that the resulting distribution $\mathcal{M}$ remains worst-case in expectation.
- **Start Time Perturbations:** In Appendix B.2, we consider what happens if the memory profile $M_{a,b}(n)$ is cyclic-shifted by a random quantity. Shifting the memory profile is equivalent to executing algorithm $\mathcal{A}_{a,b}(n)$ starting at a random start-time in the cyclic version of $M_{a,b}(n)$. Again, the resulting distribution of profiles remains worst-case in expectation.
- **Box-order Perturbations:** In Appendix B.3, we consider a relaxation of the construction of $M_{a,b}(n)$ in which rather than always placing a box of size n after the *final* instance of $M_{a,b}(n/b)$, we instead allow ourselves to place the box of size n after *any* of the a recursive instances of $M_{a,b}(n/b)$. We prove that the resulting distribution over box sizes again remains worst-case in expectation. In fact, for a square profile M drawn from the distribution at random, we find that M remains worst-case with probability *one*. This is a large contrast to random shuffling considered in Appendix A, where the random shuffle causes the algorithms to be adaptive in expectation.

B.1 Box-size Perturbations

In this section, we consider the distribution over square profiles generated by randomly perturbing the sizes of boxes within the profile $M_{a,b}$ and show that algorithm $\mathcal{A}_{a,b}$ is not cache-adaptive on the modified profile (for all problem sizes n).

Theorem 6. *Let n be the size of an input to the algorithm $\mathcal{A}_{a,b}$, $t \in [1, \sqrt{n}]$ be an arbitrary value, and let $\mathcal{P}$ be a probability distribution on $[0, t]$. Suppose that the expected value of a random variable drawn from $\mathcal{P}$ is at least $\Omega(t)$.*

Let $X_1, X_2, \ldots$ be iid random variables drawn from $\mathcal{P}$. Let $M'_{a,b}$ be constructed by replacing each box $\square_i$ in $M_{a,b}$ with a box of size $X_i \cdot |\square_i|$. Then $M'_{a,b}$ is worst-case for $\mathcal{A}_{a,b}(n)$ in expectation.

We begin by presenting two lemmas about algorithm performance on modified box sizes.

Lemma 9. *Let $M = (\square_1, \square_2, \ldots)$ be an arbitrary square profile. Suppose that when $\mathcal{A}_{a,b}(n)$ is executed on M, it completes at box $\square_k$. Now define $M' = (\square'_1, \square'_2, \ldots)$ to be a square profile such that $|\square'_i| \leq |\square_i|$ for all i. (For convenience, we will even allow $|\square'_i| = 0$.) Then when $\mathcal{A}_{a,b}(n)$ is executed on M', it completes at some box $\square'_{k'}$ satisfying $k' \geq k$.*

Proof. This follows by repeated applications of the No-Catchup Lemma (Lemma 2). Suppose that M and M' differ only in their i-th box, with $|\square'_i| < |\square_i|$. Then the first $(i-1)$ boxes of each of M and M' will finish at the same point within $\mathcal{A}_{a,b}(n)$ (i.e., will finish after the same block access). The i-th box of M' will then finish at the same point or earlier within $\mathcal{A}_{a,b}(n)$ than does the i-th box of M. By the No-Catchup Lemma, it follows that the k-th box of M' will also finish at the same point within $\mathcal{A}_{a,b}(n)$ or earlier than does the k-th box of M. Since $\mathcal{A}_{a,b}(n)$ requires k boxes to complete

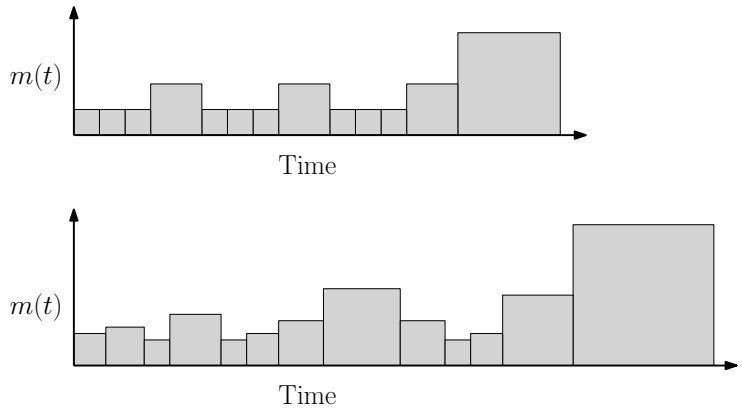

Figure 4: An example of randomly perturbing box sizes. The top profile is $M_{a,b}$ when $a = 3$ and $b = 2$. The bottom profile is an example of a random multiplication of the box sizes in $M_{a,b}$.

on profile M, $\mathcal{A}_{a,b}(n)$ will also require at least k boxes to complete on profile M'.

The above reasoning assumes that M and M' differ in only a single box. By k repeated applications of the argument, we may instead allow M and M' to differ in all of their first k boxes. This implies the full lemma. □

Lemma 10. *Let $\alpha \cdot M_{a,b}$ denote the memory profile obtained by multiplying the size of each box in $M_{a,b}$ by α. If $\alpha \le \sqrt{n}$ is a power of b, then $\alpha \cdot M_{a,b}$ is still a worst-case profile for $\mathcal{A}_{a,b}(n)$.*

Proof. The proof of the lemma takes advantage of the self-symmetry implicitly present within $M_{a,b}$. In particular, notice that $\alpha \cdot M_{a,b}$ can be obtained from $M_{a,b}$ by *removing* every box in $M_{a,b}$ of size smaller than α.

Define $M'_{a,b}(n)$ to be the profile obtained by removing every box from $M_{a,b}(n)$ of size smaller than α. By Lemma 9, the algorithm $\mathcal{A}_{a,b}(n)$ will require (at least) all of the boxes in $M'_{a,b}(n)$ to complete, since it requires all of the boxes in $M_{a,b}(n)$ to complete.

Since $\alpha \cdot M_{a,b}$ contains $M'_{a,b}(n)$ as a prefix, in order to prove that $\alpha \cdot M_{a,b}$ is a worst-case profile, it suffices to show that the sum of $|\square|^{\log_b a}$ over the boxes in $M'_{a,b}(n)$ is $\Omega(\log n \cdot n^{\log_b a})$. That is, if $M'_{a,b}(n) = (\square_1, \ldots, \square_k)$, then we wish to show that

$$\sum_{i=1}^{k} |\square_i|^{\log_b a} = \Omega(\log n \cdot n^{\log_b a}).$$

Notice that for each box-size n/b^j such that $b^j \le \sqrt{n}$, the profile $M'_{a,b}(n)$ contains a^j instances of a box of size n/b^j. (In particular, the recursive construction of $M_{a,b}$ includes a^j subproblems of size n/b^j.) Thus

$$\begin{aligned}
&\sum_{i=1}^{k} |\square_i|^{\log_b a} \\
&\ge \sum_{b^j=1}^{b^j=\sqrt{n}} a^j \cdot \left(\frac{n}{b^j}\right)^{\log_b a} \\
&= \sum_{b^j=1}^{b^j=\sqrt{n}} a^j \cdot \frac{n^{\log_b a}}{a^j} \\
&= \Theta(\log n) \cdot n^{\log_b a},
\end{aligned}$$

as desired. □

Combining Lemmas 9 and 10, we can now complete the proof of Theorem 6.

Proof of Theorem 6. Let T be the smallest power of b greater than t. By Lemma 10, the square profile $T \cdot M_{a,b}$ is worst-case for $\mathcal{A}_{a,b}(n)$.

Suppose that $\mathcal{A}_{a,b}(n)$ uses k boxes to complete on $T \cdot M_{a,b}$. Since $T \cdot M_{a,b}$ has the property that its i-th box is of size at least as large as the i-th box of $M'_{a,b}$, Lemma 9 tells us that $\mathcal{A}_{a,b}$ also requires at least k boxes to complete on $M'_{a,b}$.[13] Since $T \cdot M_{a,b}$ is a worst-case profile, in order to complete the proof, it therefore suffices to show that

$$\mathbb{E}\left[\sum_{i=1}^{k} (X_i \cdot |\square_i|)^{\log_b a}\right] \ge \Omega\left(\sum_{i=1}^{k} (T \cdot |\square_i|)^{\log_b a}\right).$$

By linearity of expectation, it suffices to prove that $\mathbb{E}[X_i^{\log_b a}] \ge T^{\log_b a}$. Since the function $f(x) = x^{\log_b a}$ is convex (because $a > b$), Jensen's inequality tells us that

$$\mathbb{E}[X_i^{\log_b a}] \ge \mathbb{E}[X_i]^{\log_b a} \ge \Omega(T^{\log_b a}),$$

as desired. □

[13] Note that if a box in $M_{a,b}$ has its size multiplied by zero, resulting in an empty box in $M'_{a,b}$, we still consider that box when talking about the i-th box of $M'_{a,b}$.

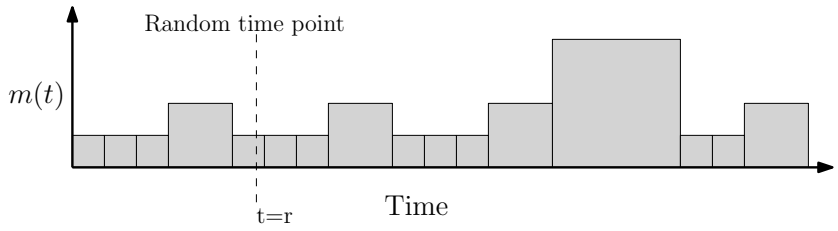

Figure 5: An example of picking a random start time with the $M^{\circ}_{a,b}$ profile where $a = 3$ and $b = 2$. If we start the algorithm at time $t = r$ instead of $t = 0$ we have created a cyclic shift.

B.2 Start Time Perturbations

In this section, we consider another natural form of smoothing, in which the algorithm $\mathcal{A}_{a,b}(n)$ begins at a random start-time within a cyclic version of the profile $M_{a,b}(n)$. Random start times simulate jobs starting at arbitrary times while a system is running.

Define the profile $M^{\circ}_{a,b}(n) = M_{a,b}(n) \circ M_{a,b}(n) \circ M_{a,b}(n) \circ \cdots$ to be the infinite profile consisting of duplicates of the profile $M_{a,b}(n)$. We will use $(\square_1, \square_2, \ldots)$ to denote the boxes in $M^{\circ}_{a,b}(n)$.

We will now generate truncated profiles simulating random start times by removing boxes from the beginning of $M^{\circ}_{a,b}(n)$. Let k be the number of boxes in $M_{a,b}(n)$ and $t = \sum_{i=1}^{k} |\square_i|$ denote the sum of the sizes of the boxes in $M_{a,b}(n)$. Define a distribution $\mathcal{M}$ over infinite profiles such that $M \in \mathcal{M}$ is constructed as follows: first select a random $r \in \{0, 1, \ldots, t-1\}$; then identify the first box $\square_j$ such that $\sum_{i=1}^{j} |\square_i| \geq r$; finally, construct M by removing each of the boxes $\square_1, \ldots, \square_{j-1}$, and replacing the box $\square_j$ with a box of size $\left(\sum_{i=1}^{j} |\square_i|\right) - r$.

The purpose of this section is to prove the following theorem:

Theorem 7. *Suppose $n > 1$ is a power of b. The distribution $\mathcal{M}$, in which a random start-time is selected within the cyclic profile $M^{\circ}_{a,b}(n)$, is worst-case in expectation for $\mathcal{A}_{a,b}(n)$.*

Proof. Recall that the profile $M_{a,b}(n)$ can be expressed as a copies of the profile $M_{a,b}(n/b)$, along with a box of size n. Let A denote the prefix of $M_{a,b}(n)$ consisting of the first $a-1$ copies of the profile $M_{a,b}(n/b)$, and let B denote the final copy of the profile $M_{a,b}(n/b)$ along with the box of size n.

Let $(\square_1, \ldots, \square_x)$ denote the boxes in A and $(\square'_1, \ldots, \square'_y)$ denote the boxes in B. We claim that

$$\sum_{i=1}^{x} |\square_i| \geq \Omega\left(\sum_{i=1}^{y} |\square'_i|\right), \tag{21}$$

and that

$$\sum_{i=1}^{x} |\square_i|^{\log_b a} \leq O\left(\sum_{i=1}^{y} |\square'_i|^{\log_b a}\right). \tag{22}$$

Before proving Equation 21 and Equation 22, we first use them to complete the proof of the theorem. When constructing a random profile M in $\mathcal{M}$, Equation 21 tells us that with probability $\Omega(1)$, r will satisfy $r \leq \sum_{i=1}^{x} |\square_i|$. When this occurs, the profile M can be obtained from the profile $M^{\circ}_{a,b}(n)$ by eliminating and shrinking boxes in the subsequence $(\square_1, \ldots, \square_x)$ and not modifying any other boxes. If we identify each of the boxes $\square_{x+1}, \square_{x+2}, \ldots$ in $M^{\circ}_{a,b}(n)$ with their counterparts in M, then by Lemma 9, when $\mathcal{A}_{a,b}(n)$ is executed on M it will still use all of the boxes $\square_{x+1}, \ldots, \square_{x+y}$. Recall that $M_{a,b}(n)$ has the property that

$$\sum_{i=1}^{x+y} |\square_i|^{\log_b a} = \log n \cdot n^{\log_b a}.$$

By Equation 22, it follows that

$$\sum_{i=x+1}^{x+y} |\square_i|^{\log_b a} \geq \Omega\left(\log n \cdot n^{\log_b a}\right). \tag{23}$$

Thus when we condition on $r \leq \sum_{i=1}^{x} |\square_i|$, the box-profile M is guaranteed to be worst-case. Since this occurs with probability $\Omega(1)$, it follows that $\mathcal{M}$ is worst-case in expectation.

It remains to prove Equation 21 and Equation 22. Since A consists of $a-1$ copies of $M_{a,b}(n/b)$ and B contains a single copy of $M_{a,b}(n/b)$ followed by a box of size n, it follows that

$$n + \sum_{i=1}^{x} |\square_i| \geq \left(\sum_{i=1}^{y} |\square'_i|\right), \tag{24}$$

and that

$$\sum_{i=1}^{x} |\square_i|^{\log_b a} \leq (a-1) \cdot \sum_{i=1}^{y} |\square'_i|^{\log_b a}. \tag{25}$$

Since $a \geq 2$, Equation 25 implies Equation 22, as desired. Since the box sequence $A = (\square_1, \ldots, \square_x)$ contains an instance of $M_{a,b}(n/b)$, it must contain at least one box of size n/b. Thus

$$(b+1) \cdot \sum_{i=1}^{x} |\square_i| \geq n + \sum_{i=1}^{x} |\square_i|.$$

Combining this with Equation 24, we get Equation 21, as desired. □

B.3 Box-order Perturbations

In this section, we consider smoothing via shuffling positions of boxes within the profile $M_{a,b}$. In particular, for n a power of b, we define $\mathcal{T}_{a,b}(n)$ to be the set of profiles constructed as follows: When $n = 1$, $\mathcal{T}_{a,b}(n)$ contains a single profile consisting of a box of size one. When $n > 1$, $\mathcal{T}_{a,b}(n)$ consists of all profiles M that can be constructed by selecting sub-profiles $X_1, \ldots, X_a \in \mathcal{T}_{a,b}(n/b)$, inserting a box of size n after one of the profiles X_i, and then concatenating the sub-profiles together. That is,

$$M = X_1 \circ X_2 \circ \cdots \circ X_i \circ \square \circ X_{i+1} \circ \cdots \circ X_a,$$

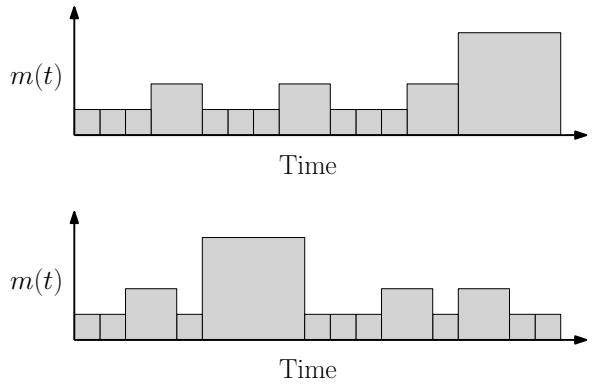

Figure 6: An example of re-ordering an $M_{a,b}$ profile. In this case $a = 3$ and $b = 2$. The top profile is $M_{a,b}$, the bottom profile is M.

for some $i \in \{1, \ldots, a\}$, and where $\square$ is a box of size n. We depict an example of a re-ordered profile, M, in Figure 6.

The profiles M in the set $\mathcal{T}_{a,b}(n)$ can be thought of as relaxations of the profile $M_{a,b}(n)$. In particular, they are the profiles obtained by allowing the recursive construction of $M_{a,b}(n)$ to, at each step in the recursion, place a scan after an arbitrary subprofile rather than always after the final subprofile.

We define the set $\mathcal{T}_{a,b}$ of infinite box-profiles to contain all profiles M that for all n (that are powers of b) contain as a prefix an element of $\mathcal{T}_{a,b}(n)$.

The main purpose of this section is to prove that all profiles $M \in \mathcal{T}_{a,b}$ are worst-case profiles for the algorithm $\mathcal{A}_{a,b}$. That, is, the recursive construction of the worst-case profile $M_{a,b}$ is robust to random shuffling of large boxes within the recursive structure.

Theorem 8. *All profiles $M \in \mathcal{T}_{a,b}$ are worst-case profiles for the algorithm $\mathcal{A}_{a,b}$.*

Proof. For n a power of b, define the ***Universal Worst-Case Profile*** $U_{a,b}(n)$ as follows. When $n = 1$, $U_{a,b}(n)$ consists of a single box of size 1 repeated a times. When $n > 1$, we construct $U_{a,b}(n)$ by concatenating together a copies of $U_{a,b}(n/b)$, and inserting a box of size n after each of them. That is,

$$U_{a,b}(n) = \left(U_{a,b}(n/b) \cdot \square\right)^a,$$

where $\square$ is a box of size n, and the multiplication operator is defined to perform concatenation.

Define the infinite-square profile $U_{a,b}$ to be the unique infinite-square profile that contains each $U_{a,b}(n)$ as a prefix. We claim that $U_{a,b}$ is a worst-case profile for $\mathcal{A}_{a,b}$. In fact, a stronger statement is true: The profile $U_{a,b}(n)$ is *the same* as the profile $M_{a,b}(n \cdot b)$, except without the final box of size $n \cdot b$ that appears at the end of the latter. This statement follows immediately by induction on $\log_b n$. It follows that $U_{a,b} = M_{a,b}$, and that $U_{a,b}$ is a worst-case profile.

We now demonstrate how to construct each profile $M \in \mathcal{T}_{a,b}(n)$ by removing boxes from $U_{a,b}(n)$. In particular, we claim that each profile $M \in \mathcal{T}_{a,b}(n)$ can be obtained from the universal worst-case profile $U_{a,b}(n)$ by removing exactly an $\frac{a-1}{a}$ fraction of the boxes in each size-class from $U_{a,b}(n)$. When $n = 1$, this is immediate, since $U_{a,b}(n)$ consists of a boxes of size one, and the only profile in $\mathcal{T}_{a,b}(1)$ consists of a single box of size one. When $n > 1$ is a power of b, the claim follows by induction, using as an inductive hypothesis that $U_{a,b}(n/b)$ can be transformed into any element of $\mathcal{T}_{a,b}(n/b)$ by removing a $\frac{a-1}{a}$ fraction of the boxes in each size-class. Recall, in particular, that each profile $M \in \mathcal{T}_{a,b}(n)$ is obtained by selecting a profiles $X_1, \ldots, X_a \in \mathcal{T}_{a,b}(n/b)$, and concatenating them together with a single box of size n after one of them. The universal profile $U_{a,b}(n)$, on the other hand, is obtained by pasting together a copies of $U_{a,b}(n/b)$ with a box of size n after each of them. By removing all but of one of the boxes of size n from $U_{a,b}(n)$ (which corresponds with removing a $\frac{a-1}{a}$ fraction of the boxes of size n), and then applying the inductive hypothesis to each of the copies of $U_{a,b}(n/b)$ in order to transform it into X_i, it follows that each $M \in \mathcal{T}_{a,b}(n)$ can be constructed by removing from $U_{a,b}(n)$ some choice of $\frac{a-1}{a}$ fraction of the boxes in each size-class.

Since $U_{a,b}(n)$ is a prefix of $M_{a,b}(n \cdot b)$, when $\mathcal{A}_{a,b}(n \cdot b)$ is executed on profile $U_{a,b}$, it must use all the boxes in $U_{a,b}(n)$. For each box-size s, let t_s denote the number of boxes of size s in $U_{a,b}(n)$. By the claim in the preceding paragraph, and by Lemma 9, when $\mathcal{A}_{a,b}(n \cdot b)$ is executed on any element $M \in \mathcal{T}_{a,b}$, it must use at least t_s/a boxes of each size s. If $(\square'_1, \square'_2, \ldots)$ denotes the profile M, and k is the number of boxes that $\mathcal{A}_{a,b}(n \cdot b)$ uses when executed on M, it follows that

$$\sum_{i=1}^{k} |\square'_i|^{\log_b a} = \sum_s \frac{t_s}{a} \cdot s^{\log_b a} \geq \Omega\left(\sum_s t_s \cdot s^{\log_b a}\right). \tag{26}$$

Since $U_{a,b}(n)$ is the same as $M_{a,b}(n \cdot b)$, except with the final box of size n removed, $|\square|^{\log_b a}$ over the boxes in $U_{a,b}(n)$ is $\Omega(\log n \cdot n^{\log_b a})$. Thus the right-hand side of Equation 26 is $\Omega(\log n \cdot n^{\log_b a})$. It follows that the profile $M \in \mathcal{T}_{a,b}$ is a worst-case profile, as desired. $\square$

C PSEUDOCODE FOR MM-SCAN

Algorithm 1 Cache-oblivious matrix multiply of two $n \times n$ matrices (each of size $N = n^2$) with $\Theta(1+N/B)$ linear scan [28]. In this pseudocode, A_{BR} refers to the Bottom Right quadrant of a matrix, A_{TL} the Top Left, etc.

function MM-Scan(n,A,B)
  if $N = 1$ **then**
    return $A \times B$
  else
    $X_{TL} \leftarrow$ MM-Scan$(n/2, A_{TL}, B_{TL})$
    $X_{TR} \leftarrow$ MM-Scan$(n/2, A_{TL}, B_{TR})$
    $X_{BL} \leftarrow$ MM-Scan$(n/2, A_{BL}, B_{TL})$
    $X_{BR} \leftarrow$ MM-Scan$(n/2, A_{BL}, B_{TR})$
    $Y_{TL} \leftarrow$ MM-Scan$(n/2, A_{TR}, B_{BL})$
    $Y_{TR} \leftarrow$ MM-Scan$(n/2, A_{TR}, B_{BR})$
    $Y_{BL} \leftarrow$ MM-Scan$(n/2, A_{BR}, B_{BL})$
    $Y_{BR} \leftarrow$ MM-Scan$(n/2, A_{BR}, B_{BR})$

    $C \leftarrow X + Y$ ▷ Linear scan
    return C

D PROOF OF THEOREM 4 AND LEMMA 4

Theorem 4 (Martingale Optional Stopping Theorem [58]). *Let $X_1, X_2, \ldots$ be iid random variables, and let γ be a function such that $\gamma(X_i)$ has finite mean μ. Consider an arbitrary process that runs in steps, and at each step i is given the value of X_i. Suppose that the process terminates after no more than C steps for some value C. Let S be the random variable denoting the number of steps that the process runs. Then,*

$$\mathbb{E}\left[\sum_{i=1}^{S} \gamma(X_i)\right] = \mathbb{E}[S] \cdot \mu.$$

Proof. Expanding $\mathbb{E}[\sum_{i=1}^{S} \gamma(X_i)]$ gives

$$\mathbb{E}\left[\sum_{i=1}^{S} \gamma(X_i)\right] = \sum_{i=1}^{C} \Pr[S \geq i] \cdot \mathbb{E}[\gamma(X_i) \mid S \geq i].$$

The key observation is that $\mathbb{E}[\gamma(X_i) \mid S \geq i] = \mu$, since the decision of whether $S \geq i$ is a function only of $X_1, \ldots, X_{i-1}$, and is therefore independent of X_i. Thus

$$\mathbb{E}\left[\sum_{i=1}^{S} \gamma(X_i)\right] = \mu \cdot \sum_{i=1}^{C} \Pr[S \geq i] = \mu \cdot \mathbb{E}[S].$$

□

Lemma 4. *For any box-size distribution Σ, and any (a, b, c)-regular algorithm $\mathcal{A}$,*

$$\mathbb{E}\left[\sum_{i=1}^{\mathcal{S}_n} m_n(|\square_i|)\right] = \mathbb{E}[\mathcal{S}_n] \cdot m_n.$$

Proof. Consider the random process that selects boxes $\square_1, \ldots, \square_{\mathcal{S}_n}$, each of size independently drawn from a distribution Σ, until the algorithm $\mathcal{A}$ is able to use the box to complete on *any* problem of size n. Then since $\mathcal{S}_n$ is bounded above by a function of n (i.e., $\mathcal{S}_n \leq O(n^{\log_b a})$), Theorem 4 tells us that

$$\mathbb{E}\left[\sum_{i=1}^{\mathcal{S}_n} m_n(|\square_i|)\right] = \mathbb{E}[\mathcal{S}_n] \cdot m_n.$$

□

E PROOF OF THE NO-CATCH-UP LEMMA

Lemma 2. *Let $\sigma = (r_1, r_2, r_3, \ldots)$ be a sequence of memory references, and let $S = (\square_1, \square_2, \ldots \square_k)$ be a sequence of squares. Suppose that if $\square_1$ starts at r_i, then $\square_k$ finishes at r_j. Then, for all $i' < i$, if $\square_1$ starts at $r_{i'}$, then for some $j' \leq j$, $\square_k$ finishes at $r_{j'}$.*

Proof. We prove this via induction on k, the number of squares in the sequence. We first prove the base case of $k = 1$.

In the base case when $k = 1$, if $\mathcal{A}$ starts $\square_1$ at access r_i and finishes $\square_1$ at access r_j, then the number of *distinct* blocks in the sequence $r_i, \ldots, r_{j+1}$ is $|\square_1| + 1$. This is because a block of size $|\square_1|$ lasts for $|\square_1|$ cache misses and $|\square_1| + 1$ distinct blocks are needed to generate $|\square_1|$ cache misses. If $\mathcal{A}$ instead starts $\square_1$ on access $r_{i'}$ for some $i' < i$, then the number of distinct blocks in the sequence $r_{i'}, \ldots, r_i, \ldots, r_{j+1}$ will also be at least $|\square_1| + 1$. Thus, $\square_1$ cannot now finish at any $r_{j'}$ satisfying $j' \geq j + 1$.

For our inductive step, we assume that the lemma holds for all $k \leq l$. We now prove the lemma holds for $k = l + 1$. Given $\square_1, \ldots, \square_l$ and a starting point r_i, let r_q be the access at which $\square_l$ finishes when $\mathcal{A}$ starts $\square_1$ at access r_i. By our inductive hypothesis, if $\mathcal{A}$ starts $\square_1$ at access $r_{i'}$ where $i' < i$, then $\mathcal{A}$ must finish $\square_l$ at access $r_{q'}$ where $q' \leq q$. Applying our proof of the base case (when $k = 1$), the memory access r_j at which $\square_{l+1}$ will finish if it starts at r_{q+1}, must come at or after the memory access $r_{j'}$ at which $\square_{l+1}$ will finish if it starts at $r_{q'+1}$. This completes the proof of the theorem. □

F STANDARDIZING $(a, b, c = 1)$-REGULAR ALGORITHMS

Lemma 11. *Let $\mathcal{A}$ be an $(a, b, c = 1)$-regular algorithm, where $b < a \in O(1)$. Then there is an $(a, b, c = 1)$-regular algorithm $\mathcal{A}'$ which has the same access pattern as $\mathcal{A}$ but which can be written as (1) a single scan consisting of at most $O(n)$ block accesses followed by (2) an $(a, b, c = 1)$-regular algorithm $\mathcal{B}$ in which in which each subproblem has its scan entirely at the end of the subproblem (rather than between or before sub-calls to smaller subproblems).*

The proof of Lemma 11 uses a variant of the "scan hiding" technique from [40].

Proof. For each subproblem in $\mathcal{A}$, we break the scan into $a + 1$ ***scan pieces***, where the first scan piece is the portion of the scan that occurs before any recursion, the second scan piece is the portion of the scan that occurs between the first and second recursive subcall, and so on.

Consider an execution of $\mathcal{A}$ on an input of size n. Call a non-base-case subproblem S in $\mathcal{A}$ a ***prefix subproblem*** if S is either the entire problem of size n, or is the subproblem resulting from the first recursive call of another prefix subproblem. Call a scan piece a ***prefix scan piece*** if it appears at the beginning of a prefix subproblem, before any recursive calls are made within the prefix subproblem.

Notice that in the execution of $\mathcal{A}$, the prefix scan pieces are performed before any other part of the computation. We define $\mathcal{A}'$ to begin by performing the prefix scan pieces together as a single large scan. For each subproblem size, there can be at most one prefix subproblem of that size. Thus the sum of the sizes of the prefix scan pieces is at most

$$O\left(\sum_{i=1}^{\log_b n} b^i\right) \leq O(n).$$

Moreover, since there are only $O(\log n)$ such pieces, their concatenation will still satisfy the property that a sufficiently large cache of constant size can complete them in $O(n)$ accesses.

The algorithm $\mathcal{A}'$ must then perform the portions of $\mathcal{A}$ that are not prefix scan pieces. Next we reinterpret these portions as an (a, b, c)-regular algorithm $\mathcal{B}$ in which scans occur only at the ends of subproblems. In a subproblem S of $\mathcal{A}$, we call a scan piece ***unassigned*** if it is not a prefix scan piece and occurs before the final recursive subcall in the subproblem.

For each unassigned scan piece in $\mathcal{A}$, we "assign ownership" for the scan piece to whichever subproblem finishes the latest out of the subproblems that finish before the scan piece. (Note that such a subproblem will exist because the scan-piece is not a prefix scan piece.) We then define $\mathcal{B}$ to be the algorithm with the same access pattern as $\mathcal{A}$ (without the prefix scan pieces), except that in the execution of $\mathcal{B}$ each subproblem (including base-case subproblems) includes any later scan pieces to which the subproblem has been assigned ownership. (Note, in particular, that each subproblem in $\mathcal{A}$ appears immediately before all scan-pieces to which it has been assigned ownership, with no other memory accesses in-between.)

Let us consider the sum of the sizes of the scan-pieces assigned to any given subproblem S of some size m. Note that any subproblem S' of size greater than $b \cdot m$ cannot assign ownership of any of its scan pieces to S; in particular, the subproblem of size $b \cdot m$ containing S must complete before any scan pieces in any such subproblem S' can occur, thereby preventing the scan pieces from being assigned to S. Moreover, for each sub-problem-size $k \leq b \cdot m$ there can be at most one subproblem of size k that assigns ownership of any of its scan pieces to S. Thus the total combined size of the scan pieces assigned to S can be at most

$$O\left(\sum_{i=1}^{\log_b(b \cdot m)} b^i\right) \leq O(m).$$

This ensures that the scans in each subproblem of size m in $\mathcal{B}$ access $O(m)$ distinct blocks, and more importantly, can be completed by a constant-size cache in time $O(m)$, meaning that algorithm $\mathcal{B}$ is, in fact, an (a, b, c)-regular algorithm. Since all of the scans in $\mathcal{B}$ occur only at the ends of subproblems, the proof is complete. □

G TRIANGLE PROFILES

Previous work showed that only considering square profiles is sufficient [6] for determining cache-adaptivity of an algorithm. In this section, we show the same result but with right triangles. A *triangle profile* can be described as a square profile where the cache is cleared between each square, hence producing a "triangular" profile composed of many adjacent right triangles.

Throughout this section, we use $\mathcal{A}$ to refer to some particular (a, b, c)-regular algorithm where $a, b \in \mathbb{N}$ and a, b, c are constants. Recall Lemma 1 relating potential to (a, b, c)-regular algorithms; since the algorithm $\mathcal{A}$ in question is (a, b, c)-regular, we let $\rho(|\square_i|) = \Theta(|\square_i|^{\log_b a})$. Finally, we let $W_n = \Theta(n^{\log_b a})$ be the total amount of progress $\mathcal{A}$ *must make* on a problem of size n in order to complete. Throughout this section, we treat both W_n and $\rho(|\square_i|)$ as fixed polynomials in n and $|\square_i|$, respectively, provided constants a, b, and c.

Intuitively, we show that triangle profiles are sufficient in proving the optimality (or non-optimality) of algorithms by noting that a box of X cache lines that lasts for X time steps fits under a triangle that starts with 0 cache lines, ends at $2X$ cache lines, and lasts for $2X$ time steps[14]. This logic accounts for the outer triangle in Fig. 7. We also note that a triangle of height and width X fits inside a box of size X, accounting for the inner triangle in Fig. 7. This intuition tells us that any square profile can be upper and lower bounded by a triangle profile up to a factor of 2 (or 1/2).

We now formalize this intuition. First, take a square profile and consider two related triangular profiles, the lower triangular profile and the upper triangular profile.

Definition 4. *A triangle, $\triangle$, of size X lasts for X IOs and on the i^{th} IO the size of the cache is i cache lines.*

The size of a triangle is represented as $|\triangle| = X$.

Definition 5. *A triangular profile, $M(t)$, is formed by a sequence of k triangles $\triangle_1 \circ \triangle_2 \circ \ \ldots \circ \triangle_k$ of sizes $\triangle_1 = X_1$.*

Definition 6. *Given a square profile $M(t) = \square_1 \circ \ \ldots \circ \square_k$ we will define the lower and upper triangular profiles.*

The lower triangular profile of $M(t)$ is $M_{LT}(t)$ where $\triangle_1 \circ \triangle_2 \circ \ \ldots \circ \triangle_k$ and $|\triangle_i| = |\square_i|$.

The upper triangular profile of $M(t)$ is $M_{UT}(t)$ where $\triangle'_1 \circ \triangle'_2 \circ \ \ldots \circ \triangle'_k$ and $|\triangle'_i| = 2|\square_i|$.

We give an example of an upper and lower triangular profile in Figure 8. We will similarly need to bound a triangular profile by square profiles.

Definition 7. *Given a triangular profile $T(t)$ $\triangle_1 \circ \triangle_2 \circ \ \ldots \circ \triangle_k$ we will define the lower square profile.*

The upper square profile of $T(t)$ is $T_{US}(t)$ where $\square_1 \circ \square_2 \circ \ \ldots \circ \square_k$ and $|\square_i| = |\triangle_i|$.

The lower square profile of $T(t)$ is $T_{LS}(t)$ where $\square'_1 \circ \square'_2 \circ \ \ldots \circ \square'_k$ and $|\square'_i| = |\triangle_i|/2$.

Definition 8. *Let $\rho(\triangle)$ be the maximum possible progress that $\mathcal{A}$ can make on a triangle of size $|\triangle|$.*

[14]Recall that *time steps* are counted in terms of block read-ins from disk to cache.

Lemma 12. *Consider $|\triangle_1| = x$, $|\triangle_2| = 2x$ and $|\square_1| = x$.*

Then the progress for our :

$$\rho(\triangle_1) = \Theta\left(x^{\log_b(a)}\right) \tag{27}$$

$$\rho(\triangle_2) = \Theta\left(x^{\log_b(a)}\right) \tag{28}$$

$$\rho(\square_1) = \Theta\left(x^{\log_b(a)}\right) \tag{29}$$

Proof. Note that we can solve a problem of size x using $\triangle_1$. Thus, we can make $x^{\log_b(a)}$ progress. By Lemma 1 we have that $\rho(\square_1) = \Theta\left(x^{\log_b(a)}\right)$. A triangle of size x fits inside a square of size x and by memory monotonicity we have that $\rho(\triangle_1) = O\left(x^{\log_b(a)}\right)$. Thus, $\rho(\triangle_1) = \theta\left(x^{\log_b(a)}\right)$.

By this we have that $\rho(\triangle_2) = \theta\left((2x)^{\log_b(a)}\right)$. Simplified $\rho(\triangle_2) = \theta\left(x^{\log_b(a)}\right)$.

□

Corollary 1. *Let $M(t)$ be a square profile. Then the progress for our (a, b, c)-regular algorithm is:*

$$\rho(M_{LT}(t)) = \Theta\left(\rho(M(t))\right)$$

and

$$\rho(M_{UT}(t)) = \Theta\left(\rho(M(t))\right).$$

Proof. $\rho(M(t))$ is the sum of the progress in the squares that make up the profile. $\rho(M_{UT}(t))$ and $\rho(M_{LT}(t))$ are the sums of the progress in the triangles that make up the profile. For any particular square, $|\square_i| = x$, in $M(t)$ there is a one to one correspondence with a triangle $|\triangle_i| = x$ in profile $M_{LT}(t)$ and a triangle $|\triangle'_i| = 2x$ in profile $M_{UT}(t)$. By Lemma 12 these all have the same progress up to constant factors. Thus, the sum of the progress of the boxes and triangles that make up these profiles will be within constant factors of each other. □

Lemma 13. *If $\mathcal{A}$ completes on $M(t)$ then it completes on $M_{UT}(t)$.*

If $\mathcal{A}$ doesn't complete on $M(t)$ then it also doesn't complete on $M_{LT}(t)$.

Proof. **For the first statement:** We prove this by induction. Let $\mathcal{A}$ be the sequence of accesses $a_1, a_2, \ldots, a_y$.
Let $M(t) = \square_1 \circ \ \ldots \circ \square_k$. Let $M_{UT}(t)$ where $\triangle'_1 \circ \triangle'_2 \circ \ \ldots \circ \triangle'_k$ and $|\triangle'_i| = 2|\square_i|$.

Assume $\mathcal{A}$ would complete access a_j by the end of $\square_i$ and $\mathcal{A}$ would complete access a_ℓ by the end of $\triangle'_i$. Further assume $i \leq \ell$. Let $a_{j'}$ be the access that $\mathcal{A}$ finishes by the end of $\square_{i+1}$. Let $a_{\ell'}$ be the access that $\mathcal{A}$ finishes by the end of $\triangle'_{i+1}$. $\square_{i+1}$ starts with at most $|\square_{i+1}|$ cache lines in memory and can bring in at most $|\square_{i+1}|$ new cache lines. $\triangle'_{i+1}$ starts with zero cache lines in memory, and can bring in $2|\square_{i+1}|$ cache lines into memory. If $j' > \ell'$ then during $\square_{i+1}$ $\mathcal{A}$ completes more accesses in x cache misses with at most x cache lines in memory at the start. However, $\triangle'_{i+1}$ can do any computation over $2x$ IOs, thus $\mathcal{A}$ run on $\triangle'_{i+1}$ can compute everything that $\mathcal{A}$ run on $\square_{i+1}$ computes. This is a contradiction. Thus, if $j \leq \ell$ then $j' \leq \ell'$.

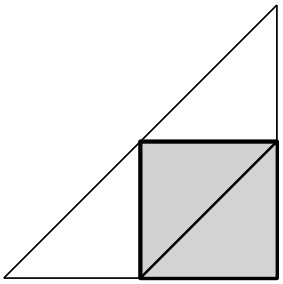

Figure 7: A triangle contained in a box and a triangle containing a box. The box is shaded in light gray.

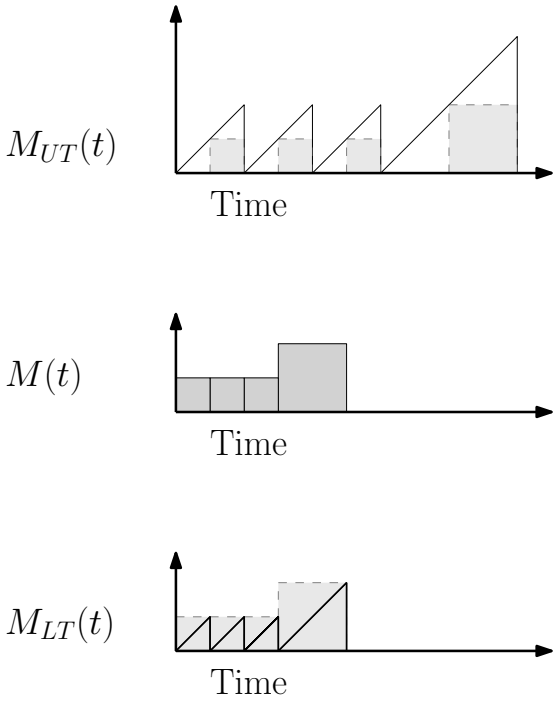

Figure 8: An example of a lower triangular profile and an upper triangular profile. The boxes from the profile $M(t)$ are presented with dashed lines on the corresponding upper and lower triangular profiles.

Base case: Before the start of the first boxes $j = \ell = 0$. Thus, by the end of $\square_1$ if $\mathcal{A}$ reaches a_j and $\mathcal{A}$ gets to access a_ℓ by the end of $\measuredangle_1'$ then $j \leq \ell$.

For the second statement: Proof by contradiction, if $\mathcal{A}$ completes on $M_{LT}(t)$ then by memory monotonicity it must also complete on $M(t)$. However, by assumption it does not, thus $\mathcal{A}$ does not complete on $M_{LT}(t)$. □

Lemma 14. *If $\mathcal{A}$ is adaptive on square profile $M(t)$ then it is adaptive on $M_{UT}(t)$ as well.*

Proof. If $\mathcal{A}$ completes on $M(t)$ then it completes on $M_{UT}(t)$ and $\rho(M_{UT}(t)) = \Theta(\rho(M(t)))$.

Thus, if $\mathcal{A}$ is adaptive on $M(t)$ it continues to be non-adaptive. □

Lemma 15. *If $\mathcal{A}$ is adaptive on triangular profile $M_{LT}(t)$ then it is adaptive on $M(t)$ as well.*

Proof. If $\mathcal{A}$ completes on $M_{LT}(t)$ then it completes on $M(t)$ and $\rho(M(t)) = \Theta(\rho(M_{LT}(t)))$.

Thus, if $\mathcal{A}$ is adaptive on $M_{LT}(t)$ it continues to be non-adaptive. □

The following theorem will allow us to use triangular profiles when proving non-adaptivity and non-adaptivity in expectation.

Theorem 9. *Let $\square_n$ be a box of size n.*

If $\mathcal{A}$ is non-adaptive on triangular profile $T(t)$ then it is also non-adaptive on the square profile $T_{LS}(t)$.

Proof. If $\mathcal{A}$ is non adaptive on $T(t)$ then

$$\rho\left(T(t)\right) = \omega\left(W\right).$$

Furthermore, $T(t)$ is the upper triangular profile of $T_{LS}(t)$ and thus

$$\rho\left(T_{LS}(t)\right) = \omega\left(W\right).$$

Let us define i and j such that $\mathcal{A}$ completes on the i^{th} triangle of $T(t)$ and $\mathcal{A}$ completes on the j^{th} square of $T_{LS}(t)$. Then, because $T(t)$ is the upper triangular profile of $T_{LS}(t)$ we can use Lemma 13 to say that the finishing point of $\mathcal{A}$ is later in $T_{LS}(t)$ than in $T(t)$. So, we have that $j \geq i$.

When $\mathcal{A}$ completes it makes W_n progress, but the available potential progress in the boxes of $T_{LS}(t)$ that it uses will be $\omega\left(W_n\right)$. The algorithm will continue to be non-adaptive on this related profile.

A distribution over triangular profiles can be connected to a distribution over square profiles by converting each triangular profile into $M_{LT}(t)$, a square profile. □

The following theorem will allow us to use triangular profiles when proving results about algorithms being adaptive in expectation.

Theorem 10. *Let D be a distribution over triangular profiles $T(t)$.*

If $\mathcal{A}$ is adaptive in expectation over a distributions D $\mathcal{A}$ is adaptive over a distribution D', where D' is formed by taking every triangular profile $T(t) \in D$ and replacing it with $T_{US}(t)$.

Proof. Note that the lower triangular profile of $T_{US}(t)$ is $T(t)$. So, we can apply Theorem 9 to $T(t)$ and $T_{US}(t)$. Every adaptive profile $T(t)$ corresponds to an adaptive $T_{US}(t)$. Additionally, every non-adaptive profile $T(t)$ corresponds to an non-adaptive $T_{US}(t)$. However, in addition to this, the optimal progress over every profile $T(t)$ and $T_{US}(t)$ are the same up to constant factors. Thus, given an algorithm run on $T(t)$ and the same algorithm run on $T_{US}(t)$ the contribution to the expected optimal progress is within constant factors. □