The Hardness of Cache Conscious Data Placement
(Extended Abstract)

Erez Petrank
Dept. of Computer Science
Technion
Haifa 32000, Israel
E-mail: erez@cs.technion.ac.il.

Dror Rawitz
Dept. of Computer Science
Technion
Haifa 32000, Israel
E-mail: rawitz@cs.technion.ac.il.

Abstract

The growing gap between the speed of memory access and cache access has made cache misses an influential factor in program efficiency. Much effort has been spent recently on reducing the number of cache misses during program run. This effort includes wise rearranging of program code, cache-conscious data placement, and algorithmic modifications that improve the program cache behavior. In this work we investigate the complexity of finding the optimal placement of objects (or code) in the memory, in the sense that this placement reduces the cache misses to the minimum. We show that this problem is one of the toughest amongst the interesting algorithmic problems in computer science. In particular, suppose one is given a sequence of memory accesses and one has to place the data in the memory so as to minimize the number of cache misses for this sequence. We show that if P $\neq$ NP, then one cannot efficiently approximate the optimal solution even up to a very liberal approximation ratio. Thus, this problem joins the small family of extremely inapproximable optimization problems. The other two famous members in this family are minimum coloring and maximum clique.

Keywords: Cache conscious data placement, cache conscious code rearrangement, memory management, computational complexity, hardness of approximation.

1 Introduction

1.1 Background and Previous Work

Much effort has recently been put into improving the cache performance of programs, i.e., reduce the number of cache misses. One avenue for improving cache behavior is to rearrange the code in memory so that cache misses during code fetch reduces (see for example [23, 16, 14]) a second avenue is to place data in the memory wisely using the memory manager (see for example [10, 7, 9, 8]), and finally, algorithms may be modified so that they access the memory in a way that reduces cache misses (see for example [6, 20, 22, 18, 19]).

In this work we concentrate on the first two avenues: cache conscious data and cache conscious code placement. The study of cache-conscious objects or code placement in the memory faces two major obstacles. The first obstacle is in obtaining the input data for the placement algorithm. A wise data placement requires knowing the sequence of accesses to the relevant objects. However, before a program is run, it is not clear what lines of codes will be run and in which order, so it is difficult to suggest the right arrangement of the code in the memory. Similarly, before a program is run, it is hard to tell what the sequence of reads and writes of the program will be. So how can the memory manager decide wisely where to put an object that is now created? The second obstacle is the algorithm itself. Suppose one got the exact series of accesses to the memory, be it accesses to the program code or accesses to obtain the data. Can one do a good placement job given full knowledge of the future?

Looking at previous work, both problems are solved via smart heuristics. In order to obtain the data, Calder et al. [7] use profiling. The assumption is that programs tend to have similar behavior even with varying inputs. Chilimbi and Larus [10] on the other hand, use a different heuristic assumption. They build on the tendency of program to behave consistently over time. Thus, they gather information on the access behavior of the program during one period of time and then use it to improve the next period, assuming the behavior remains the same. Algorithms for code rearranging usually use smart static analysis to determine the code flow.

In order to determine how to place data (or code) in the memory, given the information gathered so far, previous work use heuristic algorithms. Several such works manage to improve program performance significantly. The question arises, is there an efficient algorithm that does better? Is there an efficient algorithm that does best?

1.2 This Work

In this work we show that unless P = NP there is no efficient optimal algorithm for data placement or code rearrangement that minimizes the number of cache misses. Furthermore, it is not even possible to get close. Actually, in worst case, all efficient algorithms output a placement that is very far from the optimal solution. This result holds in a very liberal sense. It holds even if one has completely overcome the difficulty of obtaining the sequence of future accesses, and is just trying to output a mapping of objects to the memory that does well for this specific given sequence. Let A be an efficient algorithm that gets as input a sequence of memory accesses and outputs a mapping f of the objects in the sequence to the memory. Let f^* be the optimal mapping that achieves the minimal number of cache misses for this sequence. Let $Misses(f)$ be the number of caches misses when the mapping f is used. We show that unless P = NP, then for any $\varepsilon > 0$ there exists a sequence for

which $\text{Misses}(f)/\text{Misses}(f^*) = \Omega(n^{1-\varepsilon})$, where n is the length of the sequence. Thus, in worst case, the algorithm does extremely bad compared to the optimal mapping.

Our result holds for direct mapping and also for t-way mapping for any $t > 1$[1]. It holds for any cache that has more than 3 cache blocks (all realistic caches do). It holds for any reasonable cache replacement algorithm[2]. It holds even in the (special) case that all objects have the same length, i.e., the difficulty does not stem from variable sized objects. It holds also if the placement of the objects is not required to be consecutive. Namely, the objects can be put anywhere in the memory. Finally, we stress again that this result holds even if one has learned the future accesses completely and knows exactly what the sequence of accesses is going to be.

A corollary of our theorem is that these problems remain intractable even if one only tries to estimate the number of cache misses in a optimal placement, without outputting an actual placement. In particular, for every $\varepsilon > 0$, if P $\neq$ NP, there is no efficient algorithm that outputs an estimate that is within a factor of $n^{\frac{1}{2}-\varepsilon}$ from the optimal value.

Similar results on the hardness of approximation, i.e., hardness to within an $n^{1-\varepsilon}$ ratio, are known to only a handful of interesting problems in computer science. The most famous of these are Min-Coloring and Max-Clique.

Next, we move to the other obstacle in cache conscious data placement: gathering the information. Most previous algorithms do not make use of the full sequence since it is too expensive. Thus, data is first processed to keep only pairwise information. For example, Calder et al. [7] keep information on how many cache misses would be caused if a pair of objects o_i and o_j are placed in memory locations that are mapped to the same cache line. They do not keep further information that allows telling what would happen if three objects o_i, o_j, o_k are placed in this manner. Chilimbi and Larus [10] gather information on the temporal affinity for each pair of objects. No information is kept on triplets. Another question that arises is how good we can do with pairwise information. Do we lose a lot of information in the process of reducing the full information to pairwise information?

We show that in the worst case the lost information is critical. Namely, after the sequence is processed to keep only pairwise information, even an algorithm that has no time limitation cannot come close to the optimal solution. Clearly, with the full information at hand, an exponential time algorithm can find the best placement. It simply tries all possible placements[3]. But when we keep only information on pairs of objects, an exponential algorithm (and also doubly exponential algorithm, or any unlimited algorithm) will not be able to find a solution that is within a factor of $k-3$ from the optimal solution, where k is the number of blocks in the cache.

[1] Of-course, our result does not apply to fully associative caches since in this case the placement in the memory does not make a difference. Fully associative caches are not used in practically all modern architectures.

[2] Note that if the replacement policy is extremely bad, then even an optimal mapping may become terribly expensive. In that case, outputting a random placement may do as bad as the optimum.

[3] As explained in Section 4.4 the optimal placement does not exploit the infinite memory. Searching for a placement that places all objects in a memory of size polynomial in the number of objects is enough.

1.3 Implications

Our results bear two important practical implications. First, we must use heuristics. There is no hope to find one (efficient) algorithm that does well for all possible benchmarks (unless P = NP). One should study how normal programs behave and do well with respect to them. The second important implication is that it is not possible to evaluate the potential of reducing cache misses via such methods. Sometimes, researchers are interested in the potential benefits of a method. The potential is used to decide whether a project is worth the effort and later it is used to evaluate how much of the potential benefit is actually obtained by the strategy chosen. In this case, there is no hope to get even close to evaluating the potential reduction in cache misses (unless P = NP), and thus we cannot tell how much of the possible benefit our strategy obtains. We can only compare the cache conscious algorithm to the normal algorithm and check how much of an improvement is obtained.

The implication of the second result (about keeping information for pairs) is that if one wishes to spend an exponential time to find the optimal arrangement in the memory, say, because one has extremely short sequences that are reused many times, then one should be careful about the way data is processed. Keeping information in a pairwise manner is not good in this case: even if one has infinite time resources, pairwise information is not enough to even get close to an optimal solution.

Finally, researchers in the theory community may find it interesting that such a strong result can be obtained without the use of Probabilistic Checkable Proofs [3, 2]. Thus, this result could have been proven in the 70's. As far as we know, there is only a handful of (interesting) problems for which such strong results exist.

1.4 Background in Hardness of Approximations

A lot of research in theoretical computer science has been devoted to showing that some problems are hard to approximate. However, until the beginning of the 90's most of the results were fairly weak. The breakthrough was provided in a series of papers [11, 3, 2] (see [1] for more details). They contained a new characterization of NP in terms of *probabilistically checkable proofs* (PCP), and with it several strong results on the hardness of approximating interesting optimization problems. For example, using this characterization Arora et al. [2] showed that for some $\varepsilon > 0$, there is no $(1-\varepsilon)$-approximation algorithm for MAX-3SAT, unless P = NP. many other interesting problems were shown to be hard to approximate to within a constant. Also, hardness of approximation to within a logarithmic factor was then shown for SET-COVER. Finally, a couple of problems were shown to be hard to within a polynomial factor. The two most interesting ones were the maximum clique problem [5, 15] and the minimum chromatic number problem [5, 12]. For more background on approximation algorithms and their hardness, the reader is referred to [4].

1.5 Techniques

We provide several reductions. The reductions are all from the k-colorability problem, known to be NP-Complete for $k \geq 3$. The input to each of the reductions is a graph and

the output is a sequence of memory accesses. Our typical reduction has one very strong property. In particular, we show that if the input graph has a k-coloring, then the objects in output sequence can be mapped to memory so that the number of cache misses is extremely small. However, if the input graph does not have a k-coloring, then any mapping of the objects in the output sequence to the memory causes a huge number of cache misses. This is true even if the input graph can be colored with just $k+1$ colors. We conclude that if one can determine quite vaguely what the number of cache misses of a given sequence would be for the optimal mapping, then the same algorithm can be used to tell if a graph is k-colorable.

1.6 A Related Work

Hollander has discussed a problem called the *k-positioning* problem in his dissertation [17]. In this problem the memory is finite (of size greater than the number of objects), and any memory block can be assigned to any cache block. This problem is equivalent to our problem for direct mapping. Hollander showed that the decision version of the k-positioning problem is NP-complete for any $k \geq 2$ and MAX-SNP-hard for $k = 2$. Thus, when the cache has two blocks the problem cannot be approximated within a factor of $1+\varepsilon$ for some $\varepsilon > 0$ unless P = NP. Our results are much stronger. First, we deal with realistic caches that have more than two blocks. Moreover, even if one is interested in caches of two blocks Hollander's result indicates that it is hard to approximate the problem to within some small constant, it does not rule out approximation by a factor of, say, 1.1. This means that there may exist an algorithm that outputs data placement that has at most 10% more misses than the optimum. Such an algorithm might have served as an excellent approximation in practice. Our results rule out the possibility of outputting a placement that is within a factor of $O(n^{1-\varepsilon})$ from the optimum for any $\varepsilon > 0$. An approximation to within such a ratio cannot be considered useful in practice.

1.7 Organization

In Section 2 we present basic concepts in caching systems. In Section 3 we define the problems considered in the paper. In Section 4 we present and prove our main results on the hardness of approximating the minimum cache misses. In Section 5 we define pairwise information algorithms and prove our result on using pairwise information. We conclude in Section 6.

2 Caching Systems

Today's caching systems are extremely complex using coherence protocols, buffering, policies of dealing with read and write misses, etc. We will show that data placement is difficult even for the simplest system. A cache system includes a cache with k cache blocks, a.k.a. cache lines. The memory is normally much bigger than the cache, so it consists of many more blocks. When an access to the memory is made, there is a mapping function that determines where in the cache the memory block may be. If the block is found (valid) in the cache we have a *cache hit*. If it is not, there is a cache miss and the block is read from the memory into the cache block. The time it takes to bring the block from the memory to the cache is much larger than the time it takes to read a cache line into the CPU. Thus, the number of cache misses is considered an important factor in the running time of a program.

Most systems employ a direct mapping system or a t-way associative mapping. In a direct mapping, each memory block is mapped exactly to a single cache block. Of-course, many memory blocks are mapped to any single cache block. Usually, the mapping is just the memory block number modulo k. In a t-way mapping, we have $k \cdot t$ cache blocks. The blocks are grouped into k sets of t blocks each. A memory block is mapped first modulo k to determine its set, and then to one of t cache lines of the k'th set. When an access to a memory block is made, the system calculates the set that corresponds to that memory block, and then either the memory line is available from one of the t cache lines of the set, or there is a cache miss and the block is read from the memory to one of the t corresponding cache blocks.

In today's technology cache blocks normally consists of 32, 64 or 128 bytes. Cache sizes start at 8KB for L1 caches and may reach several Mega Bytes for L2 caches in server machines.

3 The Problems

In this section we define the problems we consider in the paper. We start with the simplest: cache conscious placement for direct mapping cache systems.

Direct Mapping

Let us now try to formalize the problem in a theoretical framework. We denote the set of memory objects by $\mathcal{O} = \{o_1, \ldots, o_m\}$. We assume all objects have identical size. We denote a sequence of object accesses by $\sigma = (\sigma_1, \ldots, \sigma_n)$ where $\sigma_i \in \mathcal{O}$ for $i \in \{1, \ldots, n\}$. We assume the size of the memory is not bounded, i.e., infinite, and we start by assuming direct mapping to a cache of k lines (a.k.a. blocks). An address n in the memory is assigned by the direct mapping to block number (n mod k) in the cache[4]. The goal is to find a memory placement of the objects in $\mathcal{O}$ that minimizes the number of cache misses which occur when accessing the data objects according to the sequence σ. Note that the placement of the objects in the memory completely determines the number of cache misses for the sequence σ. Once the objects have been placed, the direct mapping allows no freedom of choice during the run. Suppose that the objects in $\mathcal{O}$ are placed in the memory by the mapping f. Denote by Misses$((\mathcal{O}, \sigma), f)$ the number of cache misses occuring while accessing the memory according to a sequence σ of objects from $\mathcal{O}$. We formalize the above in the following manner:

Minimum k-cache misses

Instance: A set of memory objects $\mathcal{O} = \{o_1, \ldots, o_m\}$, and a sequence of accesses $\sigma = (\sigma_1, \ldots, \sigma_n)$ such that $\sigma_i \in \mathcal{O}$ for all $i \in \{1, \ldots, n\}$.

Solution: A placement of the objects to the memory, i.e., a mapping $f : \mathcal{O} \to \mathbb{N}$.

[4]Our result holds for any other reasonable direct mapping. We require that the mapping be efficiently computable, efficiently invertible. and that memory blocks are evenly mapped into the k cache lines. In fact, an even distribution is not necessary, but far from even distribution may hurt.

Measure: Misses$((\mathcal{O}, \sigma), f)$.

A simplification we make, is that we think of object sizes as being equal to each other and also equal to the size of the cache block. Note that this is a special case of the general problem, and even this special case turns out hard. Our results can easily be extended to show hardness also for cases in which the objects are forced to being smaller or larger than the cache blocks.

Another simplification we use is that the memory is not limited. Intuitively, the fact that actual memories are finite does not help in finding a good placement. We show this formally in Section 4.4.

The t-way Mapping and Sensible Replacement Protocols

A natural extension of the k-cache misses problem is the *t-way k-cache misses* problem. In this problem we are given a t-way cache: each of the k cache sets contains t cache blocks. This means that, in any given moment, there can be t objects in each cache set (when using our simplified object sizes). Here, the number of misses is determined by the mapping of objects to the memory and by the policy, called *the replacement protocol*, used to determine which of the t blocks in the set is replaced when a miss occurs.

Our results for t-way caches require that the replacement policy is not extremely bad. If it is, we may get that all mapping of objects to memory behave so bad, so that there is no difference between an optimal mapping and any other mapping. We think of a replacement policy as extremely bad, if recently used objects keep being replaced, whereas unused objects are kept in the cache.

Consider the following scenario. We are given a sequence of memory accesses ending with many repetitions of a subsequence $\tilde{\sigma}$ that accesses a subset A of the objects, and all objects in A are mapped to the same cache set. Clearly, if the number of objects in A is greater than t many cache misses will occur. However, what should we expect when there are t or less objects in A? A bad replacement protocol may keep old memory objects in the cache and replace the objects in A even though they repeat many times in this subsequence. This sort of behavior is very expensive in terms of cache misses and does not happen in practical replacement protocols. We expect a reasonable replacement protocol to place all the objects in A in the cache and prevent more cache misses. Therefore, a reasonable requirement from a replacement protocol of a t-way cache is to satisfy the following rule: if a (possibly very long) sequence iteratively accesses q objects which are mapped to the same cache set, and $q \leq t$, then the number of cache misses should not exceed q by much. We formalize this by the following.

Definition 1 *A replacement protocol for a t-way cache is called* C-sensible *if the following holds. Let σ be a sequence of q distinct objects $\sigma = o_{i_1} o_{i_2} \ldots o_{i_q}$ and let these objects be placed in the memory via any placement that does not cause more than t of these objects to be mapped to the same cache set. Then, for any possible sequence σ' and any integer ℓ, when we run the sequence σ' and σ^ℓ consecutively we get at most $q + C$ misses on the accesses of σ^ℓ.*

In asymptotic analysis, the exact value of the constant C becomes immaterial. Therefore, we will sometimes (when appropriate) ignore the constant C and call the replacement protocol *sensible*.

Clearly, replacement protocols such as LRU and FIFO are 0-sensible replacement protocols. A pseudo-LRU algorithm is used in the Intel Pentium platform. The replacement protocol works as follows. Each set is divided into two pairs. Each pair has a flag indicating which block in this pair was accessed last. Another flag is used to determine which pair contains the most recently used block. The replacement protocol replaces the least recently used block from the least recently used pair (note that this may not be the least recently used block). This replacement protocol is 0-sensible as well.

t-way Caches

Denote by Misses$_t((\mathcal{O}, \sigma), f)$ the number of cache misses occuring while accessing the memory according to a sequence σ of objects from $\mathcal{O}$ when using a t-way cache (the replacement protocol should be clear from the context). The cache misses minimization problem in this case can be formulated as follows:

Minimum t-way k-cache misses

Instance: A set of memory objects $\mathcal{O} = \{o_1, \ldots, o_m\}$, and a sequence of accesses $\sigma = (\sigma_1, \ldots, \sigma_n)$ such that $\sigma_i \in \mathcal{O}$ for all $i \in \{1, \ldots, n\}$.

Solution: A placement of the objects to the memory, i.e., a mapping $f : \mathcal{O} \to \mathbb{N}$.

Measure: Misses$_t((\mathcal{O}, \sigma), f)$.

We define the following for the t-way k-cache misses problem. The corresponding definitions for the k-cache misses problem can be understood in a strait-forward manner. Given a set of objects $\mathcal{O}$ and a sequence σ, a placement f^* is called *optimal* if every placement f satisfies Misses$_t((\mathcal{O}, \sigma), f^*) \leq$ Misses$_t((\mathcal{O}, \sigma), f)$. A placement f is called *r-approximate* if Misses$_t((\mathcal{O}, \sigma), f) \leq r \cdot$ Misses$_t((\mathcal{O}, \sigma), f^*)$, where f^* is an optimal placement. A polynomial time algorithm is called an *r-approximation algorithm* if it returns an r-approximate placement.

3.1 Graph colorability

We use the graph k-colorability problem in our reductions.

Graph k-colorability (kCOL)

Instance: An undirected graph $G = (V, E)$.

Problem: Is G k-colorable, i.e., does there exist a function $c : V \to \{0, \ldots, k-1\}$ such that $c(u) \neq c(v)$ whenever $(u, v) \in E$?

This problem is known to be NP-complete for any $k \geq 3$ (see, e.g., [13]).

We call a function $c : V \to \{0, \ldots, k-1\}$ a k-coloring. Given a graph G and a k-coloring c we call an edge $e = (u, v)$ monochromatic if $c(u) = c(v)$. We denote by Mono(G, c) the number of monochromatic edges in G. In particular, if c is a k-coloring of G, then Mono$(G, c) = 0$.

4 Hardness Results

In this section we prove our results on the difficulty of approximating minimum k-cache misses and minimum t-way

k-cache misses. In both problems we show that, if P $\neq$ NP, for every $k \geq 3$ the problem cannot be approximated within $O(n^{1-\varepsilon})$ for every $\varepsilon > 0$. We show this by constructing reductions from graph k-colorability. The reductions get as input a graph G and output an instance of the k-cache misses or the t-way k-cache misses problem such that if G is k-colorable then the optimal number of cache misses is extremely low, and otherwise it is very high. Such reductions imply that an approximation algorithm for the k-cache misses or the t-way k-cache misses problem can be used to distinguish between graphs which are k-colorable and graphs which are not. Moreover, even a vague evaluation of the optimal solution provides a way to determine whether a given graph is k-colorable.

4.1 Notation and Terminology

We use the following notation. Given two sequences $\sigma = (\sigma_1, \ldots, \sigma_n)$ and $\sigma' = (\sigma'_1, \ldots, \sigma'_{n'})$, we denote by $\sigma \bigodot \sigma'$ the sequence $(\sigma_1, \ldots, \sigma_n, \sigma'_1, \ldots, \sigma'_{n'})$ (i.e., $\bigodot$ denotes concatenation). σ^t stands for $\bigodot_{i=1}^{t} \sigma$.

4.2 Minimum k-cache Misses

We start by constructing an instance of the k-cache problem for any possible graph. This construction will later be used in the proof of Theorem 1. In the construction we use a parameter ℓ which is a constant to be determined later. Given a graph $G = (V, E)$ we construct an instance of the k-cache misses problem. This instance contains a memory object for each vertex in V, and a sequence of accesses in which each edge is represented by a sub-sequence. The subsequence corresponding to an edge $(u, v) \in E$ consists of many repetitions of the two objects which correspond to the two vertices u and v. Formally, given a graph $G = (V, E)$ we define $H_\ell(G) = (\mathcal{O}_G, \sigma_G)$, where $\mathcal{O}_G = \{o_1, \ldots, o_{|V|}\}$, and

$$\sigma_G = \bigodot_{(v_i, v_j) \in E} (o_i, o_j)^{|E|^\ell} .$$

Since ℓ is a constant $H_\ell(G)$ can be constructed in polynomial time, and it is of size $\Theta(|E|^{\ell+1})$.

Next we relate placements f to colorings c in such a way that the number of misses in the sequence σ_G with the placement f will be strongly connected to the number of monochromatic edges in G when colored by the corresponding coloring c. Given a coloring c of V we define a corresponding family of corresponding placements F_c. A placement $f : \mathcal{O}_G \to \mathbb{N}$ is in F_G if it places o_i in a memory block that is mapped to a cache line which is indexed by $c(v_i)$ (e.g., $f(o_i) = i \cdot k + c(v_i)$). Note that for every placement f there exists a coloring c for which $f \in F_c$ (for c that colors $v_i \in V$ according to the cache line assigned to the memory address in which o_i is placed). The next lemma shows a connection between a coloring c of V and a corresponding placement $f \in F_c$. Recall that Mono(G, c) is the number of monochromatic edges in G when colored by c.

Lemma 1 *Let c be a k-coloring of V. Then, for any $f \in F_c$*

$$\mathit{Misses}(H_\ell(G), f) = \Theta(|E|^\ell \, \mathit{Mono}(G, c)) + O(|E|) .$$

Proof: Consider a mapping $f \in F_c$. Now, examine an edge $(v_i, v_j) \in E$. If the edge (v_1, v_2) is monochromatic then $f(o_i)$ and $f(o_j)$ are mapped to the same cache line. Therefore, the sub-sequence $(o_i, o_j)^{|E|^\ell}$ that corresponds to a monochromatic edge contributes at least $2|E|^\ell - 1$ misses. On the other hand, if (v_1, v_2) is not monochromatic then $f(o_i)$ and $f(o_j)$ are not mapped to the same cache line. Thus, a non-monochromatic edge causes no more than two misses. ∎

Next, we show that if G is k-colorable then the optimal number of cache misses is low, and otherwise it is very high.

Proposition 1 *If G is k-colorable then $Opt(H_\ell(G)) = O(|E|)$, otherwise $Opt(H_\ell(G)) = \Omega(|E|^\ell)$.*

Proof: If G is k-colorable then there exists a coloring c for which there are no monochromatic edges. Thus, by Lemma 1, Misses$(H_\ell(G), f) = O(|E|)$ for $f \in F_c$. On the other hand, if G is not k-colorable then a coloring for which there are no monochromatic edges does not exist, which means that Mono$(G, c) > 0$ for any coloring c of G. As mentioned before, for any placement f there exists a coloring c such that $f \in F_c$. Therefore, Lemma 1 implies that Misses$(H_\ell(G), f) = \Omega(|E|^\ell)$ for every placement f. ∎

Theorem 1 *For every $k \geq 3$ and $0 < \varepsilon \leq 1$ the k-cache misses problem cannot be approximated within a factor of $O(n^{1-\varepsilon})$, unless $P = NP$.*

Proof: Given a graph G we fix $\ell = \frac{3}{\varepsilon} - 1$ (note that it is a constant as promised), and construct $H_\ell(G)$. By Proposition 1 if $G \in k$COL then Opt$(H_\ell(G)) = O(|E|)$, while if $(G, k) \notin$ COL then Opt$(H_\ell(G)) = \Omega(|E|^\ell)$. Thus, distinguishing between instances I for which Opt$(I) = O(|E|)$, and instances I for which Opt$(I) = \Omega(|E|^\ell)$ is NP-hard. Suppose there exists an algorithm that outputs an $O(|E|^{\ell-2})$-approximate placement. Then, we can use it to determine if Opt$(I) = O(|E|)$. If there exists a placement f such that Misses$(H_\ell(G), f) = O(|E|)$ then the algorithm will output a placement g such that Misses$(H_\ell(G), g) = O(|E|^{\ell-1})$. However, if such a placement does not exist, the algorithm will output a placement g for which Misses$(H_\ell(G), g) = \Omega(|E|^\ell)$. This implies that there cannot exist an $O(|E|^{\ell-2})$-approximation algorithm for the minimum cache misses problem, unless P $\neq$ NP. It remains to relate $|E|^{\ell-2}$ to the instance size, i.e., to $|H_\ell|$. Note that:

$$\begin{aligned} |E|^{\ell-2} &= \Theta(|H_\ell(G)|^{\frac{\ell-2}{\ell+1}}) \\ &= \Theta(|H_\ell(G)|^{\frac{3/\varepsilon-3}{3/\varepsilon}}) \\ &= \Theta(|H_\ell(G)|^{1-\varepsilon}) \end{aligned}$$

and we are done. ∎

Corollary 2 *For every $k \geq 3$ and $0 < \varepsilon \leq \frac{1}{2}$ the optimum of the k-cache misses problem cannot be efficiently approximated within a factor of $O(n^{\frac{1}{2}-\varepsilon})$, unless $P = NP$.*

Proof: Given a graph G we fix $\ell = \frac{3}{2\varepsilon} - 1$, and construct $H_\ell(G)$. The gap shown in Theorem 1 implies that if we

can efficiently approximate the value of the optimal solution within a factor of $O(|E|^{(\ell-2)/2})$ then we can decide in polynomial time whether G is k-colorable or not. Thus, it is NP-hard to approximate the value of the optimal solution within a factor of $O(|E|^{(\ell-2)/2})$, and

$$\begin{aligned} |E|^{(\ell-2)/2} &= \Theta(|H_\ell(G)|^{\frac{\ell-2}{2(\ell+1)}}) \\ &= \Theta(|H_\ell(G)|^{\frac{3/2\varepsilon-3}{3/\varepsilon}}) \\ &= \Theta(|H_\ell(G)|^{\frac{1}{2}-\varepsilon}) \ . \end{aligned}$$

The corollary follows. ■

4.3 Minimum t-way k-cache Misses

We now turn to showing that data placement is difficult even for systems that do not employ direct mapping. We start with constructing an instance of the t-way k-cache misses problem (i.e., a sequence of accesses) for any given a graph $G = (V, E)$. As in the case of the k-cache misses problem this instance contains a memory object for each vertex in V, and each edge $(u, v) \in E$ is represented by some sub-sequence which consists of many repetitions of the two objects which correspond to the two vertices u and v. However, as opposed to the direct mapping case, the instance contains more (dummy) memory objects and an additional sub-sequence whose purpose is to force the replacement policy to imitate a direct mapping cache. Due to this property we can establish a connection between the number of monochromatic edges in a given coloring and the number of cache misses in a corresponding mapping for any sensible replacement protocol.

Given a graph $G = (V, E)$, we construct the instance $H_\ell^t(G) = (\mathcal{O}_G^t, \sigma_G^t)$, where

$$\mathcal{O}_G^t = \left\{o_1, \ldots, o_{|V|}\right\} \cup \{d_{i,j} \ : \ i \in [k],\ j \in [t+1]\}$$

where $[r] \triangleq \{0, \ldots, r-1\}$, and

$$\sigma_G^t = \sigma_{G,1}^t \bigodot \sigma_{G,2}^t$$

where

$$\begin{aligned} \sigma_{G,1}^t &= \bigodot_{(v_i,v_j)\in E} \left(\left(\bigodot_{i=0}^{k-1} \bigodot_{j=0}^{t-2} (d_{i,j}) \right) \bigodot (o_i, o_j) \right)^{|E|^\ell} \\ \sigma_{G,2}^t &= \bigodot_{j=0}^{t} \left(\bigodot_{i=0}^{k-1} \bigodot_{j' \neq j} (d_{i,j'}) \right)^{|E|^{\ell+2}} \end{aligned}$$

Next, as in the k-cache case we relate mappings to colorings in such a way that the number of misses in σ_G^t due to a mapping will be strongly related to the number of monochromatic edges in G when colored by the corresponding coloring. Given a coloring c of V we define a corresponding family of placements F_c^t. A placement $f : \mathcal{O}_G^t \to \mathbb{N}$ is in F_c^t if it places o_i in a memory block that is mapped to a cache set which is indexed by $c(v_i)$, and distributes the objects in $D_j \triangleq \{d_{i,j} : i \in [k]\}$ into the k different cache sets, i.e., places, for all $j \in [t+1]$, exactly one object from the set D_j into each cache set. Note that for every placement f which places exactly one object from D_j into each cache set for all $j \in [t+1]$ there exists a coloring c for which $f \in F_c$ (the one which colors $v_i \in V$ according to the cache set assigned to the memory address in which o_i is placed). The next lemma shows a connection between a coloring c of V and a corresponding placement $f \in F_c^t$.

Lemma 2 *Let c be a k-coloring of V. If the replacement protocol is C-sensible, then for any $f \in F_c^t$*

$$Misses_t(H_\ell^t(G), f) = \Theta(|E|^\ell \ Mono(G, c)) + O(|E|) \ .$$

Proof: Consider a mapping $f \in F_c^t$. When using a cache with an C-sensible replacement protocol and the placement f the number of misses in the second part of σ_G^t (i.e., in $\sigma_{G,2}^t$) is at most $t \cdot C = O(1)$.

Now let us check $\sigma_{G,1}^t$. Consider an edge $(v_i, v_j) \in E$. If (v_i, v_j) is monochromatic then $f(o_i)$ and $f(o_j)$ are mapped to the same cache set. In this case, each repetition of the sub-sequence $\tilde{\sigma} = (\bigodot_{i=0}^{k-1} \bigodot_{j=0}^{t-2} (d_{i,j})) \bigodot (o_i, o_j)$ causes at least one cache miss (and up to $t+1$ misses) because there are $t+1$ objects in this sub-sequence which are mapped to the same cache set. Since $\tilde{\sigma}$ is repeated $|E|^\ell$ times there are $\Theta(|E|^\ell)$ misses due to the sub-sequence corresponding to the edge (v_i, v_j). On the other hand, if the edge (v_i, v_j) is not monochromatic then $f(o_i)$ and $f(o_j)$ are not mapped into the same cache set. In this case, there are exactly t objects whose addresses are mapped to the i'th set for any $i \in [k]$. Thus, when using a sensible replacement protocol, the sub-sequence $\tilde{\sigma}^{|E|^\ell}$, which corresponds to the non-monochromatic edge (v_i, v_j), causes $O(1)$ misses.

The lemma follows from the fact that there are $\text{Mono}(G, c)$ monochromatic edges, and up to $|E|$ non-monochromatic edges. ■

Proposition 2 *If G is k-colorable then $Opt(H_\ell^t(G)) = O(|E|)$, otherwise $Opt(H_\ell^t(G)) = \Omega(|E|^\ell)$.*

Proof: Consider an optimal placement f. There are two possibilities, either, for all $j \in [t+1]$, f splits each set $S_j \triangleq \{d_{i,j'} : i \in [k], j' \in [t+1]\} \setminus D_j$ evenly in the cache, i.e., for each cache set C_r ($r \in [k]$) f places exactly t objects from S_j in addresses corresponding to C_r, or f does not have this property. If f does not place the $d_{i,j}$'s as described above then for some j more than t objects from S_j are placed in addresses corresponding to some cache set. In this case, the resulting number of misses in $\sigma_{G,2}^t$ will be $\Omega(|E|^{\ell+2})$. On the other hand, if f places the $d_{i,j}$'s in this way the number of misses caused by $\sigma_{G,2}^t$ will be no greater than $O(tk) = O(1)$ (k and t are constants). By Lemma 2, the overall number of misses will be $O(|E|^{\ell+1}) = o(|E|^{\ell+2})$. We conclude that an optimal placement must have the above property, i.e., it must place exactly t objects from S_j in addresses corresponding to each cache set for every $j \in [t+1]$.

Now, we show that in order to satisfy the above requirement, an optimal placement must place exactly one object from $D_j = \{d_{i,j} : i \in [k]\}$ in an address corresponding to each cache set for any $j \in [t+1]$. Consider two indices $j_1, j_2 \in [t+1]$ such that $j_1 \neq j_2$. We know that an optimal placement f must place exactly t objects from S_{j_r} in addressed corresponding to each cache set for $r = 1, 2$. By definition $S_{j_2} = S_{j_1} \cup D_{j_1} \setminus D_{j_2}$. Thus, the surplus in

any cache set after adding the objects in D_{j_1} to S_{j_1} must be eliminated by the removal of D_{j_2}. Thus, the number of objects from the sets D_{j_1} and D_{j_2} which are placed by f in each cache set must be the same. This argument works for all $j_1, j_2 \in [t+1]$ such that $j_1 \neq j_2$, and, therefore, the number of objects from the set D_j which are placed by an optimal placement f in each cache set must be the same for any $j \in [t+1]$. We claim that the only way to do this while placing exactly t objects from S_j in each cache set for every $j \in [t+1]$ is by placing exactly one object from D_j in each cache set for any $j \in [t+1]$. Assume the contrary. Then, there exist a cache set i in which f places either no objects from the D_j's or more than one object from each of the D_j's. Pick an index $j \in [t+1]$ and examine the set S_j. In the first case no objects are placed in the i'th set, and in the second at least $2t$ objects are placed in the i'th set. Both in contradiction to the requirement that an optimal solution places exactly t objects from S_j in each cache set for every $j \in [t+1]$. Therefore, an optimal solution must place exactly one object from D_j in each cache set for any $j \in [t+1]$.

The rest of the proof is the similar to the proof of Proposition 1 (where Lemma 2 replaces Lemma 1). Recall that if a placement f places exactly one object from D_j in each cache set for any $j \in [t+1]$, then there exist a k-coloring c such that $f \in F_c^t$. If G is k-colorable then there exists a coloring c for which there are no monochromatic edges. Thus, by Lemma 2, $\text{Misses}_t(H_\ell^t(G), f) = O(|E|)$ for any $f \in F_c^t$. On the other hand, if G is not k-colorable then a coloring for which there are no monochromatic edges does not exist. Therefore, Lemma 2 implies that $\text{Misses}(H_\ell^t(G), f) = \Omega(|E|^\ell)$ an optimal placement f. ∎

Theorem 3 *For every $k \geq 3$ and $0 < \varepsilon \leq 1$ the t-way k-cache misses problem cannot be approximated within a factor of $O(n^{1-\varepsilon})$, unless $P = NP$.*

Proof: Given a graph G we fix $\ell = \frac{4}{\varepsilon} - 2$, and construct $H_\ell^t(G)$. By Proposition 2 distinguishing between instances I for which $\text{Opt}(I) = O(|E|)$, and instances I for which $\text{Opt}(I) = \Omega(|E|^\ell)$ is NP-hard. This implies that there cannot exist an $O(|E|^{\ell-2})$-approximation algorithm for the minimum cache misses problem, unless P $\neq$ NP. It remains to relate $|E|^{\ell-2}$ to the instance size, i.e., to $\left|H_\ell^t\right|$. Note that

$$\begin{aligned} |E|^{\ell-2} &= \Theta(\left|H_\ell^t(G)\right|^{\frac{\ell-2}{\ell+2}}) \\ &= \Theta(|H_\ell(G)|^{\frac{4/\varepsilon-4}{4/\varepsilon}}) \\ &= \Theta(|H_\ell(G)|^{1-\varepsilon}) \end{aligned}$$

and we are done. ∎

Corollary 4 *For every $k \geq 3$ and $0 < \varepsilon \leq \frac{1}{2}$ the optimum of the k-cache misses problem cannot be efficiently approximated within a factor of $O(n^{\frac{1}{2}-\varepsilon})$, unless $P = NP$.*

Proof: Given a graph G we fix $\ell = \frac{4}{2\varepsilon} - 2$, and construct $H_\ell^t(G)$. The gap shown in Theorem 3 implies that if we can efficiently approximate the value of the optimal solution within a factor of $O(|E|^{(\ell-2)/2})$ then we can decide in polynomial time whether G is k-colorable or not. Thus, it is NP-hard to approximate the value of the optimal solution within a factor of $O(|E|^{(\ell-2)/2})$, and

$$\begin{aligned} |E|^{(\ell-2)/2} &= \Theta(|H_\ell(G)|^{\frac{\ell-2}{2(\ell+2)}}) \\ &= \Theta(|H_\ell(G)|^{\frac{4/2\varepsilon-4}{4/\varepsilon}}) \\ &= \Theta(|H_\ell(G)|^{\frac{1}{2}-\varepsilon}) \ . \end{aligned}$$

The corollary follows. ∎

Note that the proof of Lemma 2 and consequently Theorem 3 and Corollary 4 hold for any sensible replacement protocol, and in particular, for LRU, FIFO, and pseudo-LRU.

4.4 Finite Size Memory

It seems intuitively correct that an infinite memory makes the placement problem easier to solve, and our hardness results stronger. In this section we formalize this intuition by showing that our hardness results are valid even if the memory is finite. We start by integrating the memory size into the definition of the problem, and continue by showing that the modified problem is as hard to approximate as its infinite counterpart. For simplicity we apply this modification to the simplest model, i.e., direct mapping with the mapping being the modulus function. A similar argument holds also for the t-way mapping, and for any other mapping that maps the memory to the cache in a reasonably balanced manner.

Consider the following scenario. We are given a set of memory objects $\mathcal{O} = \{o_1, \ldots, o_m\}$, an access sequence σ, and a memory of size $M \geq |\mathcal{O}|$. Our goal is to find a placement $f : \mathcal{O} \to [M]$ which minimizes the number of cache misses. The problem is formalized as follows.

Minimum k-cache M-memory misses

Instance: A set of memory objects $\mathcal{O} = \{o_1, \ldots, o_m\}$, a sequence of accesses $\sigma = (\sigma_1, \ldots, \sigma_n)$ such that $\sigma_i \in \mathcal{O}$ for all $i \in \{1, \ldots, n\}$, and an integer $M \geq m$.

Solution: A placement of the objects to the memory, i.e., a mapping $f : \mathcal{O} \to [M]$.

Measure: $\text{Misses}((\mathcal{O}, \sigma), f)$.

Theorem 5 *For any $k \geq 3$ and $0 < \varepsilon \leq 1$ the k-cache M-memory misses problem cannot be approximated within a factor of $O(n^{1-\varepsilon})$, unless $P = NP$.*

Proof: We use the same proof as in Theorem 1, except that on a given input graph G, the output of the reduction is $(H_\ell(G), m \cdot k)$. namely, we add a bound on the memory size being $M = m \cdot k$, where m is the number of objects and k is the number of blocks in the cache. To see that this works well, we have to show that when G is k-colorable, the optimal placement has a small number of cache misses and when G is not k-colorable any placement causes many cache misses. In the latter case the analysis does not change. The limitation on the placement (i.e., the size of the memory) may only increase the cache misses for the optimal solution, and this does not foil the analysis. In the first case, where G is k-colorable, we need to show that the optimal placement does have the low number of cache misses as in the original proof of the theorem. However, since there are m objects,

the optimal placement may try to map at most m objects to any specific cache block. Doing this is possible also when the memory has $m \cdot k$ cache blocks. To map j objects to cache block number i (for $j \leq m$), the placement puts them in memory blocks $i, i+k, \ldots, i+j \cdot k$. Thus, the optimal placement is possible also when the memory is restricted, and we are done with the proof of Theorem 5. ■

Corollary 6 *For every $k \geq 3$ and $0 < \varepsilon \leq \frac{1}{2}$ the optimum of the k-cache M-memory misses problem cannot be approximated within a factor of $O(n^{\frac{1}{2}-\varepsilon})$, unless $P = NP$.*

Proof: Same argument as in Corollary 2. ■

5 Pairwise Information

In this section we show that keeping only pairwise information on a given sequence of memory accesses causes a loss of critical information, even if one is allowed to process the information with no computational limitations. Our intentions is to model the actual procedure by which data placement algorithms have kept the data [10, 7]. Since the full sequence of accesses is hard to predict, and, even when given, it is long and requires a lengthy processing, previous works used pairwise information. In particular, for each pair of objects – how many cache misses would result from their being placed to two locations in the memory that are mapped to the same cache line.

We show that the value of a pairwise optimal solution may be far from the value of an optimal solution to the k-cache misses problem for every $k \geq 2$. We do this by constructing an instance with the following property: the placement that achieves the minimum cache misses for pairs is is a factor of k away from an optimal placement.

We demonstrate the difficulty for direct mapping. We believe that these reductions can be extended to t-way caches as well.

5.1 The Problem

Let us begin with formalizing the above notion of using pairwise information. Consider a instance $(\mathcal{O}, \sigma)$ of the k-cache misses problem, where $\mathcal{O} = \{o_1, \ldots, o_m\}$. Given two objects $o_i, o_j \in \mathcal{O}$, let $\sigma_{i,j}$ be the sequence resulting from eliminating all other elements, i.e., all elements o_r such that $r \neq i, j$ from σ. We call the sequence $\sigma_{i,j}$ the *pairwise sequence* of o_i and o_j. Consider the following scenario: we are given $\binom{m}{2}$ pairwise sequences $\{\sigma_{i,j} : 1 \leq i < j \leq m\}$, and we try to find one global good mapping that minimizes the number of misses in all pairwise sequences, that is, we try to find a mapping f that minimizes $\text{Pairs}((\mathcal{O}, \sigma), f) = \sum_{i<j} \text{Misses}((\{o_i, o_j\}, \sigma_{i,j}), f)$. Formally, we consider the following problem:

Minimum k-cache pairwise misses

Instance: A set of memory objects $\mathcal{O} = \{o_1, \ldots, o_m\}$, and a sequence of accesses $\sigma = (\sigma_1, \ldots, \sigma_n)$ such that $\sigma_i \in \mathcal{O}$ for all $i \in \{1, \ldots, n\}$.

Solution: A placement of the objects to the memory, i.e., a mapping $f : \mathcal{O} \to \mathbb{N}$.

Measure: $\text{Pairs}((\mathcal{O}, \sigma), f)$.

Note that instances of the k-cache misses problem and instances of the k-cache pairwise misses problem have the same format. The difference is in the target optimization function.

Given a placement f let $\text{Size}_{\mathcal{O},f}(i)$ be the number of objects from $\mathcal{O}$ which are mapped by f to the i'th cache line. Using the convention that memory blocks are mapped to cache blocks with the modulo function, we get, $\text{Size}_{\mathcal{O},f}(i) \triangleq |\{o_j : f(o_j) \mod k = i\}|$. We call a mapping f *balanced* if $\text{Size}_f(i) = \text{Size}_f(j)$ for every $i, j \in [k]$, otherwise it is called *unbalanced*[5].

5.2 The Construction

We construct an instance of the k-cache misses problem for which the optimum to the k-cache pairwise misses problem is a factor of k away from the optimum of the k-cache misses problem. The instance contains k^2 memory objects, and an access sequence which treats them as k sets of k objects. Let $\mathcal{O} = \{o_{ij} : i \in [k], j \in [k]\}$, and let $\sigma = \sigma_1 \bigodot \sigma_2$, where

$$\sigma_1 = \bigodot_{i=0}^{k-1} \left(\bigodot_{r=0}^{k-1} o_{ir} \right)^{3\ell}$$

$$\sigma_2 = \bigodot_{0 \leq i < j \leq k-1} \left(\bigodot_{r=0}^{k-1} \bigodot_{s=0}^{k-1} (o_{ir}, o_{js})^{2\ell} \right)$$

and ℓ is a large number (to be determined later).

The intuition behind this construction is the following. The goal is to make the optimal pairwise placement be much different from the real optimal placement. This sequence makes sure that the pairwise optimal solution, i.e., for the k-cache pairwise misses problem, places the objects $o_{i1}, \ldots, o_{ik}$ in addresses corresponding to different cache sets for any i. The real optimal placement does the opposite: for each i, it places the objects $o_{i1}, \ldots, o_{ik}$ in addresses corresponding to the same cache line.

The goal of the first sequence, σ_1, is to mislead the pairwise target function. This sequence seems extremely expensive to the pairwise target function because there are $\binom{k}{2}$ pairs, each causing many cache misses. However, when the sequence is really run, these costs are not disjoint and need not be counted $\binom{k}{2}$ times. After blinding the pairwise target function in this manner, we make it pay for its mistake using the second sequence σ_2. In the eyes of the pairwise target function the cost of σ_2 seems small comparing to the cost of σ_1. This is of-course not true when the full sequence is checked. The optimal placement pays very little when running σ_2 and this is how the gap between the two placements is obtained. We go on and formalize this intuition.

In what follows it is important to remember that ℓ is a large number, much larger than k. The access sequence σ consists of inner sub-sequences (containing k or 2 memory objects) which appear either 3ℓ or 2ℓ times. In σ_1 an inner sub-sequence is of the form $\bigodot_{r=0}^{k-1} o_{ir} = (o_{i1}, \ldots, o_{ik})$ and in σ_2 it is of the form (o_{ir}, o_{js}). Consider such an inner sub-sequence $\tilde{\sigma}$. Given a mapping f we can divided the cache misses occuring during the repetitions of $\tilde{\sigma}$ into two

[5] In our analysis, the number of objects m is always an integer multiplication of the number of sets in the cache, k. A natural generalization of this definition allows, when m is not a multiplication of k, a difference of at most 1 between the maximum and the minimum values of $\text{Size}_{\mathcal{O},f}(i)$ over all i's.

types. The first type are misses occuring during the first time the sub-sequence $\tilde{\sigma}$ is accessed, and the second type are the misses that occur in the many other repetitions of $\tilde{\sigma}$. We consider this difference because we want to separate discussing the required cost of bringing some objects to the cache the first time they are accessed from the (not always required) cost of repeatedly bringing them to the cache again and again because the placement was not good. If $\tilde{\sigma}$ contains two or more objects that are mapped to addresses which correspond to the same cache line a miss occur in each repetition. Otherwise, no misses of the second type occur. The sequence σ_1 contains k inner sub-sequences of size k, and σ_2 contains $k^2\binom{k}{2}$ inner sub-sequences of size 2. Therefore, there can be up to $k^2(1+2\binom{k}{2}) \leq k^4$ misses due to the first type of misses for every possible mapping. In the analysis, misses of the second type will be the dominant.

Our plan is as follows. We start with showing that the optimal placement has a low cost. Next, we show that the placement that gets the pairwise target function to the minimum has a specific structure. Finally, we show that with this structure, the optimal pairwise placement must cause a lot of cache misses.

We fix $\ell = k^3$, and we start with an upper bound on the value of an optimal mapping for the k-cache misses problem.

Lemma 3 *Let f be an optimal placement for the k-cache misses problem. Then, Misses$((\mathcal{O},\sigma),f) \leq 4\ell k^2$.*

Proof: Let g be the following placement function: the objects in the set $\mathcal{O}_i \triangleq \{o_{ij} : j \in [k]\}$ are placed in memory locations corresponding to the i'th line of the cache. For this placement we get Misses$((\mathcal{O},\sigma_1),g) = 3\ell k^2$, and Misses$((\mathcal{O},\sigma_1),g) \leq k^4$. Thus, Misses$((\mathcal{O},\sigma),g) \leq 3\ell k^2 + k^4 \leq 4\ell k^2$. Since f is the optimal placement, the cost it implies is bounded by the cost of using g. ∎

The cache misses counted by Pairs$((\mathcal{O},\sigma),f)$ can be divided into two cases. Consider two memory objects $o_{i_1j_1}$, $o_{i_2j_2} \in \mathcal{O}$, and denote by $\sigma_{(i_1j_1),(i_2j_2)}$ the pairwise sequence of two objects, i.e, the projection of the sequence σ on these two objects. If $i_1 = i_2$ then $\sigma_{(i_1j_1),(i_2j_2)}$ consists of a prefix $(o_{i_1j_1}, o_{i_2j_2})^{3\ell}$ that is projected from σ_1 and a suffix that contains $(k-1)k$ copies of $o_{i_1j_1}^{2\ell}$ and $(k-1)k$ copies of $o_{i_2j_2}^{2\ell}$ projected from σ_2. On the other hand, if $i_1 \neq i_2$ then $\sigma_{(i_1j_1),(i_2j_2)}$ consists of a prefix $o_{i_1j_1}^{3\ell} \bigodot o_{i_2j_2}^{3\ell}$ originating from σ_1, and three contributions from σ_2: a sub-sequence $(o_{i_1j_1}, o_{i_2j_2})^{2\ell}$, $k(k-1)-1$ copies of $o_{i_1j_1}^{2\ell}$, and $k(k-1)-1$ copies of $o_{i_2j_2}^{2\ell}$.

The number of misses resulting from placing the two objects in addresses corresponding to the same cache line can be divided into two types. The first type of misses are those that occur due to the sub-sequence which contains either 2ℓ or 3ℓ repetitions of $(o_{i_1j_1}, o_{i_2j_2})$, and the second type are the rest. If we access the same object many time, it is as if we have accessed it only once, as far as the cache is concerned. Therefore, there are no more than $2k(k-1)$ pairwise misses of the second type (in both cases).

We show that a pairwise optimal solution must be balanced.

Lemma 4 *A placement f which minimizes Pairs$((\mathcal{O},\sigma),f)$ is balanced.*

Proof: Let g be an unbalanced placement. We show how to construct a placement f from g such that Pairs$((\mathcal{O},\sigma),f) <$ Pairs$((\mathcal{O},\sigma),g)$. Thus, an unbalanced placement cannot be optimal with respect to the target function Pairs. We start by observing that because g is unbalanced there exist two cache sets i_1, i_2 for which Size$_{\mathcal{O},g}(i_1) \geq k+1$ and Size$_{\mathcal{O},g}(i_2) \leq k-1$. Namely, the number of objects that g maps the cache set i_1 is larger by at least 2 from the number of objects it maps to the cache set i_2. Furthermore, there exists an index i for which g maps more objects from $\mathcal{O}_i = \{o_{ij} : j \in [k]\}$ to set i_1 than to set i_2. Denote by t_1 and t_2 the number of objects from $\mathcal{O}_i$ which are mapped to sets i_1 and i_2, respectively. Also, define $\Delta_r = \text{Size}(\mathcal{O},g)(i_r) - t_r$ for $i \in \{1,2\}$. Δ_1 and Δ_2 are the number of objects not from $\mathcal{O}_i$ that are mapped to the sets i_1 and i_2, respectively. Clearly,

$$(t_1+\Delta_1) - (t_2+\Delta_2) = \text{Size}_{\mathcal{O},g}(i_1) - \text{Size}_{\mathcal{O},g}(i_2) \geq 2 \ . \tag{1}$$

Let f be a placement which places objects as g does with the exception that one of the objects, which is mapped by g to set i_1, is placed by f in set i_2. We begin by ignoring the first type of misses (i.e., the misses that occur in the first repetition) and consider the misses that occur repeatedly. We will later claim that misses of the first type do not interfere too much in the calculations. We start by computing the difference in pairwise misses between g and f. Note that the difference originates only from misses in the i_1 and i_2 cache lines. Namely, due to the fact that t_1 is reduced by 1 and t_2 is increased by 1. We separate the calculation to σ_1 and σ_2. For σ_1 the modified cost is due to the interaction between the t_1 (or t_2) objects in the set $\mathcal{O}_i$.

$$\begin{aligned}\text{Pairs}((\mathcal{O},\sigma_1),g) - \text{Pairs}((\mathcal{O},\sigma_1),f) = \\ 6\ell(\tbinom{t_1}{2} + \tbinom{t_2}{2} - \tbinom{t_1-1}{2} - \tbinom{t_2+1}{2}) \ = \\ 6\ell(t_1 - t_2 - 1)\end{aligned}$$

For σ_2 the main cost is the interaction of the moved object with the Δ_1 (or Δ_2) objects that are not in $\mathcal{O}_i$.

$$\text{Pairs}((\mathcal{O},\sigma_2),g) - \text{Pairs}((\mathcal{O},\sigma_2),f) = 4\ell(\Delta_1 - \Delta_2) \ .$$

We show that Pairs$((\mathcal{O},\sigma),g) >$ Pairs$((\mathcal{O},\sigma),f)$ in each of three possible cases:

1. $t_1 - t_2 = 1$.
   In this case, by Equation 1, $\Delta_1 - \Delta_2 \geq 1$.
   Thus, Pairs$((\mathcal{O},\sigma_1),g) =$ Pairs$((\mathcal{O},\sigma_1),f)$,
   and Pairs$((\mathcal{O},\sigma_2),g) >$ Pairs$((\mathcal{O},\sigma_k^2),f)$.

2. $t_1 - t_2 \geq 2$ and $\Delta_1 - \Delta_2 \geq 0$.
   Pairs$((\mathcal{O},\sigma_1),g) >$ Pairs$((\mathcal{O},\sigma_1),f)$,
   and Pairs$((\mathcal{O},\sigma_2),g) \geq$ Pairs$((\mathcal{O},\sigma_2),f)$.

3. $t_1 - t_2 \geq 2$ and $\Delta_1 - \Delta_2 < 0$.
   In this case, using Equation 1,

   $$\text{Pairs}((\mathcal{O},\sigma_1),g) - \text{Pairs}((\mathcal{O},\sigma_1),f) \geq 6\ell\cdot(2 - \Delta_1 + \Delta_2),$$

   whereas,

   $$\text{Pairs}((\mathcal{O},\sigma_2),g) - \text{Pairs}((\mathcal{O},\sigma_2),f) \geq 4\ell(\Delta_1 - \Delta_2).$$

   Thus,

   $$\begin{aligned}\text{Pairs}((\mathcal{O},\sigma),g) - \text{Pairs}((\mathcal{O},\sigma),f) \geq \\ 6\ell(2 - \Delta_1 + \Delta_2) + 4\ell(\Delta_1 - \Delta_2).\end{aligned}$$

   Using the fact that $2 - \Delta_1 + \Delta_2 > 0$, we get that

   $$\text{Pairs}((\mathcal{O},\sigma),g) - \text{Pairs}((\mathcal{O},\sigma),f) \geq 4\ell \cdot 2 > 0.$$

We conclude that, $\text{Pairs}((\mathcal{O},\sigma),g) - \text{Pairs}((\mathcal{O},\sigma),f) \geq \ell$. Now we must take into account the change in the first type of pairwise misses. As said earlier, there can be up to $2k(k-1)$ misses of the second type. Thus, in the worst case, $\text{Pairs}((\mathcal{O},\sigma),g) - \text{Pairs}((\mathcal{O},\sigma),f) \geq k^3 - 2k(k-1) > 0$. This means that f is a better placement than g with respect to the k-cache pairwise problem. It follows that for every unbalanced placement there exist another better placement. Therefore, an optimal placement must be balanced. ■

Next, we present a lower bound on the value of a pairwise optimal solution.

Lemma 5 *Let f be a placement that minimizes $Pairs((\mathcal{O},\sigma),f')$. Then,*

$$Misses((\mathcal{O},\sigma),f) \geq 2\ell k(k-1) \ .$$

Proof: Let g be a balanced placement. Consider two objects $o, o' \in \mathcal{O}$ which are placed in the same cache set by g. If $o, o' \in \mathcal{O}_i$ for some $i \in [k]$, then their contribution to $\text{Pairs}((\mathcal{O},\sigma),g)$ is at least 6ℓ misses, otherwise they contribute no more than $4\ell + 2k(k-1)$ misses. $4\ell + 2k(k-1) < 6\ell$, thus, the best pairwise balanced placement f would be a placement which places the objects from $\mathcal{O}_i$ in different cache sets for any $i \in [k]$ (e.g., the one that places $o_{1j}, \ldots, o_{kj}$ in the same cache set for any j). Thus, $\text{Pairs}((\mathcal{O},\sigma),f) \geq 4\ell k\binom{k}{2}$. However, recall that we want to find out the number of "real" misses that occur when using the placement f. To do that examine σ_2. As mentioned above, f places the objects from $\mathcal{O}_i$ in different cache sets for any $i \in [k]$. Thus, each object from $\mathcal{O}_i$ shares a cache set with objects that are not members of $\mathcal{O}_i$ for any $i \in [k]$. In other word this means that all the $\mathcal{O}_i$'s are represented in every cache line L_r ($r \in [k]$). Consider a cache line L_r. There are k objects that are mapped to addresses corresponding to L_r, such that every pair appears in 2ℓ times in σ_2. Thus, the contribution of L_r to $\text{Misses}((\mathcal{O},\sigma),f)$ is $\binom{k}{2}4\ell$ misses. This is true for all cache lines, and, therefore, $\text{Misses}((\mathcal{O},\sigma),f) \geq 4\ell k\binom{k}{2} = 2\ell k^2(k-1)$. ■

Finally, we can compare the values of an optimal mapping and an optimal pairwise mapping.

Theorem 7 *For every $k \geq 2$ there exist an instance for which the number of misses due to an optimal pairwise mapping is within a factor of $\frac{1}{2}(k-1)$ from an optimal mapping.*

Proof: Let f and g be an optimal placement and an optimal pairwise placement, respectively. By Lemma 3, $\text{Misses}((\mathcal{O},\sigma),f) \leq 4\ell k^2$. On the other hand, by Lemmas 4 and 5 $\text{Pairs}((\mathcal{O},\sigma),g) \geq 2\ell k^2(k-1)$. The theorem follows. ■

This result can be improved to a factor of $k-3$ in the following manner.

Theorem 8 *For every $k \geq 2$ there exist an instance for which the number of misses due to an optimal pairwise mapping is within a factor of $k-3$ from an optimal mapping.*

Proof: Replace the 3ℓ in σ_1 by $(k-2)\ell$, and the 2ℓ in σ_2 by $(k-3)\ell$. Let f and g be an optimal placement and an optimal pairwise placement, respectively. By slightly changing Lemma 3 we can show that $\text{Misses}((\mathcal{O},\sigma),f) \leq (k-2)\ell k^2 + k^4$. Using the fact that $\ell = k^3$, we get that

$$\text{Misses}((\mathcal{O},\sigma),f) \leq \ell k(k-1)^2.$$

We can also modify Lemmas 4 and 5 in order to prove that $\text{Pairs}((\mathcal{O},\sigma),g) \geq (k-3)\ell k^2(k-1)$. The theorem follows from the fact that $\frac{k(k-3)}{k-1} \geq k-3$. ■

6 Conclusion

We have studied an important challenge in todays computing environment: how to place data and code in the memory so that cache misses are reduced to minimum. We have shown that in the worst case, an efficient algorithm cannot even come close to achieving this goal (not even just estimate the number of misses for the optimal placement), unless P = NP. Thus, this problem joins the small family of extremely inapproximable optimization problems.

We have also studies the question of how much information is lost when one keeps only pairwise relations between the objects. We have shown that in the worst case critical information is lost in this manner and it is not possible to get close to the optimal memory arrangement even if one has no time limits.

The implication of the first result is that we must use heuristics. There is no hope to find one (efficient) algorithm that does well for all possible benchmarks (unless P = NP). One should study how normal programs behave and do well with respect to them. Another important implication is that it is not possible to evaluate the potential of reducing cache misses via such methods. Thus, one cannot tell, before implementing a cache conscious data (or code) placement algorithm, what the potential benefit is. Also, one cannot evaluate how much of the potential benefit his implemented algorithm actually obtains. The measure of such algorithms should be their improvement over existing non-cache-conscious algorithms on given benchmarks.

The implication of the second result (about keeping information for pairs) is that if one wishes to spend a long time to find the optimal arrangement in the memory via a full search, then one should be careful about the way data is processed. Keeping information in a pairwise manner is not good in this case: even if one has infinite time resources, pairwise information is not enough to even get close to the optimal solution.

7 Acknowledgments

We thank Seffi Naor for helpful discussions.

References

[1] S. Arora and C. Lund. Hardness of approximations. In *Approximation Algorithms for NP-Hard Problem*, chapter 10, pages 399–446.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998.

[3] S. Arora and S. Safra. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998.

[4] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation; Combinatorial optimization problems and their approximability properties*. Springer Verlag, 1999.

[5] M. Bellare, O. Goldreich, and M. Sudan. Free bits, PCPs, and nonapproximability—towards tight results. *SIAM Journal on Computing*, 27(3):804–915, 1998.

[6] H.-J. Boehm. Reducing garbage collector cache misses. In T. Hosking, editor, *ISMM 2000 Proceedings of the Second International Symposium on Memory Management*, volume 36(1) of *SIGPLAN*, Minneapolis, MN, Oct. 2000. ACM Press.

[7] B. Calder, C. Krintz, S. John, and T. Austin. Cache-conscious data placement. In *Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, CA, Oct. 1998.

[8] T. M. Chilimbi, B. Davidson, and J. R. Larus. Cache-conscious structure definition. In PLDI [21], pages 13–24.

[9] T. M. Chilimbi, M. D. Hill, and J. R. Larus. Cache-conscious structure layout. In PLDI [21], pages 1–12.

[10] T. M. Chilimbi and J. R. Larus. Using generational garbage collection to implement cache-conscious data placement. In R. Jones, editor, *ISMM'98 Proceedings of the First International Symposium on Memory Management*, volume 34(3) of *SIGPLAN*, pages 37–48, Vancouver, Oct. 1998. ACM Press. ISMM is the successor to the IWMM series of workshops.

[11] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *32th IEEE Symposium on Foundations of Computer Science*, pages 2–12, 1991.

[12] U. Feige and J. Kilian. Zero knowledge and the chromatic number. *Journal of Computer and System Sciences*, 57(2):187–199, 1998.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[14] N. Gloy and M. D. Smith. Procedure placement using temporal-ordering information. *ACM Transactions on Programming Languages and Systems*, 21(5):977–1027, 1999.

[15] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. In *37th IEEE Symposium on Foundations of Computer Science*, pages 627–636, 1996.

[16] E. A. Henis, G. Haber, M. Klausner, and A. Warshavsky. Fdpr - a post-link optimization tool for large subsystems. *Feedback Directed Optimizations 2 Workshop*, 1999.

[17] Y. Holander. *Search algorithms for cache memory*. PhD thesis, Technion, 1995.

[18] Desktop performance and optimization for Intel$^{(R)}$ Pentium$^{(R)}$ 4 processor. `http://developer.intel.com/design/pentium4/papers/249438.htm`, 2001.

[19] Intel$^{(R)}$ Pentium$^{(R)}$ 4 processor optimization, reference manual. `http://developer.intel.com/design/pentium4/manuals/248966.htm`.

[20] K. S. McKinley, S. Carr, and C.-W. Tseng. Improving data locality with loop transformations. *ACM Transactions on Programming Languages and Systems*, 18(4):424–453, 1996.

[21] *Proceedings of SIGPLAN'99 Conference on Programming Languages Design and Implementation*, SIGPLAN, Atlanta, May 1999. ACM Press.

[22] G. Rivera and C.-W. Tseng. Eliminating conflict misses for high performance architectures. In *Proceedings of the International Conference on Supercomputing (ICS-98)*, pages 353–360, New York, 1998. ACM press.

[23] W. J. Schmidt, R. R. Roediger, C. S. Mestad, B. Mendelson, I. Shavit-Lottem, and V. Bortnikov-Sitnitsky. Profile-directed restructuring of operating system code. *IBM Systems Journal*, 37(2):270–305, 1998.