Alcom-FT Technical Report Series
ALCOMFT-TR-02-30

Scanning Multiple Sequences Via Cache Memory*

Kurt Mehlhorn Peter Sanders
Max-Planck-Institut für Informatik,
Im Stadtwald, 66123 Saarbrücken, Germany.
E-mail: mehlhorn,sanders@mpi-sb.mpg.de
WWW: http://www.mpi-sb.mpg.de/~mehlhorn,sanders

February 8, 2002

Abstract

We consider the simple problem of scanning multiple sequences. There are k sequences of total length N which are to be scanned concurrently. One pointer into each sequence is maintained and an adversary specifies which pointer is to be advanced. The concept of scanning multiple sequence is ubiquitous in algorithms designed for hierarchical memory.

In the external memory model of computation with block size B, a memory consisting of m blocks, and at most m sequences the problem is trivially solved with N/B memory misses by reserving one block of memory for each sequence.

However, in a cache memory with associativity a, every access may lead to a cache fault if $k > a$. For a direct mapped cache ($a = 1$) two sequences suffice. We show that by randomizing the starting addresses of the sequences the number of cache misses can be kept to $O(N/B)$ provided that $k = O(m/B^{1/a})$, i.e., the number of sequences that can be supported is decreased by a factor $B^{1/a}$. We also show a corresponding lower bound.

Our result leads to a general method for converting sequence based algorithms designed for the external memory model of computation to cache memory even for caches with small associativity.

Keywords: Cache memory, set associative cache, external memory algorithm, memory hierarchy, multi merge, data distribution.

1 Introduction

Modern computers have a hierarchical memory: registers, first-level cache, second-level cache, main memory, disks, and tapes. Levels closer to the processor (= lower levels) are faster and smaller than the levels farther away from the processor. Data transfer between different levels occurs in chunks which are called cache blocks or pages depending on the level of the hierarchy. One essential difference between the cache memory and higher levels of the hierarchy is that the placement of data in the cache is highly restricted by a hardware feature called *associativity of the cache*.

Formally, a *cache memory* is defined as follows. A cache consists of a certain number m of *cache blocks*. Each cache block has size B and hence the total size of the cache is $M = m \cdot B$.

*Partially supported by EU ESPRIT LTR Project N. 20244 (ALCOM-IT). A preliminary version of this paper was presented at ICALP 99 [San99a].

Size may be measured in bytes; a more natural unit is the size of the data elements of the underlying application and this is the view which we adopt in this paper.[1] Data transport from and to the next higher level of memory is done in blocks. We call a block of the next higher level of memory a *memory block*. A memory block can usually be only placed at a small number of cache blocks. More precisely, a cache is organized into $s = m/a$ cache sets and each cache set consists of a cache blocks; the constant a is called the *associativity* of the cache. A memory block starting at memory address xB (in short hand: memory block x) can only be cached in the set numbered $x \bmod s$. A cache with $a = 1$ is called *direct mapped* and a cache with $a = m$ is called *fully associative*. For reasons of speed, cost, and power dissipation, actual caches typically have an associativity between one and eight. An alternative but equivalent view of cache memory is as follows: Memory is partitioned into s sets of memory blocks and for each set, a fully associative cache of size a is available.

An access to a memory block x that is not residing in cache memory is called a *cache fault* or *cache miss*. Caches usually use the LRU or the FIFO replacement strategy: On a cache miss, the least recently used (LRU) block or the block used first (FIFO) in the set of x is replaced by the new block.

Actual hardware caches may have features which our model does not directly handle, (e.g., [HP96, Chapter 5]). Nevertheless, it can often be adapted. The most important issue are multiple levels of cache. Our analysis can be applied to each pair of levels separately. A similar situation arises for *pseudo-associative caches* which have fast and slow cache hits. For fast hits it behaves like an a-way associative cache and for slow hits it behaves like an a'-way associative cache with $a' > a$. Often, $a = 1$ and $a' = 2$. Again, our analysis can be used to estimate the number of slow and fast hits separately. Another example are *victim caches*. A victim cache is a small (typically, less than 32 blocks) cache with high associativity that holds the last cache blocks evicted from the cache. However, if the number of sequences k exceeds the capacity of the victim cache, it has little impact on the number of cache misses. For similar reasons, a small number of *write buffers* are of little help if a program writes to many sequences.

In this paper we study the effect of the degree of associativity on the simple problem of concurrent traversal of multiple sequences, a problem ubiquitous in algorithms designed for hierarchical memory: k sequences of total length N are to be scanned concurrently. In each iteration, an adversary selects the sequence whose pointer is to be advanced. The adversary is oblivious, i.e., it does not know the choice of our random numbers. We show:

- If the number of sequences is larger than a, the worst case number of cache misses is N.
- If the starting addresses of the sequences are randomized and the number of sequences is $k = \mathcal{O}(m/B^{1/a})$, the number of cache misses is $\Theta(N/B)$. If $k \ll s$, our upper and lower bounds on the number of cache faults essentially agree.
- Our results imply a general method for converting sequence-based algorithms designed for the external memory model of computation to cache-aware algorithms: Restrict the number of sequences concurrently traversed to $O(m/B^{1/a})$ and randomize starting addresses.

There are also results how caches with limited associativity can emulate external memory with an optimal replacement policy. Frigo et al. [FLPR99] give a randomized scheme. Sen

[1] We make the simplifying assumption that B is integral.

and Chatterjee [SC00] give a deterministic scheme that requires the application program to provide the optimal access strategy. Although both strategies need some constant factor more memory ([SC00] only for $a > 1$) than a perfect cache would need, this factor is independent of B. Hence, these results allow less cache faults for large B and applications that need $\omega M/B^{1/a}$ fast memory. We believe that this does not make our results less interesting because our approach leaves programs unchanged except for the memory allocation whereas direct emulation techniques require software controlled access to every memory block that is moved between slow and fast memory. Currrently this seems only practical for algorithms like matrix multiplication that have simple predictable memory access patterns. Another advantage of our results is that we give upper bounds on the number of cache faults that are very small for applications that only need $o(M/B^{1/a})$ cache memory.

We state our results in more detail in Section 2. Section 3 puts our results into context. Sections 4 and 5 contain the proofs of our upper and lower bounds, respectively, Section 6 generalizes the upper bound to situations where additional data is stored in addition to the sequence data, Section 7 discusses the space requirement of randomized placement, and Section 8 gives applications.

2 Problem Definition and Statement of Results

We define our problem formally and state our results. Consider k sequences stored in arrays. The sequences are read (or written) sequentially. An adversary is allowed to schedule the accesses to the sequences, i.e., is allowed to choose N and $s_1, \ldots, s_N \in \{1, \ldots, k\}$ in the following code:

for $t := 1$ **to** N **do**
    work on the current element of sequence s_t;
    advance sequence s_t to the next element;

We use N_i to denote the number of elements scanned in sequence i. Then $N = \sum_i N_i$.

The concurrent traversal of multiple sequences is trivially realized in the external memory model of computation as long as the number k of sequences is at most the number M/B of memory blocks. One simply reserves one block of memory for each sequence. Observe that this strategy does not follow the LRU or the FIFO replacement strategy, as it always replaces a block by a block of the same sequence, and that it uses full associativity, as all elements of a sequence are stored in the same block of main memory. The total number of I/Os is equal to the number of memory blocks occupied by the sequences (at least $\sum_i \lceil N_i/B \rceil$ and at most k more) and this is clearly optimal.

The behavior of multiple sequence access in cache memory can be dreadful. Assume that the sequences are scheduled in round-robin fashion and that the i-th block of all sequences is mapped to the same cache set for all i, $1 \le i \le k$. This will for example be the case, if all sequences have the same length $c \cdot M/a$ for some integer c and if the sequences are stored consecutively. In this situation and if the number of sequences is more than a, each access will lead to a cache fault. The total number of cache faults is equal to N. We performed an experiment to confirm this analysis. We allocated an array A of $N = 2^{24}$ ints and divided it into k pieces of length $L = N/k$ each. We then scanned the pieces in round-robin fashion (p and $p0$ are pointers into the array A) and summed the elements of all sequences.

k	1	2	4	8	16	32	64	128	256	512	1024
T	0.52	4.03	3.99	4.02	4.04	4.01	5.6	5.58	5.6	5.53	5.55

Table 1: Execution times in seconds for scanning k sequences of total length $N = 2^{24}$ in round-robin fashion. The program was compiled with g++, optimization flag -O4, and run on a SUN Sparc Ultra. The machine has a direct mapped cache, i.e., $a = 1$.

```
for (p0 = A;  p0 < A + L;  p0++) {
   p = p0;
   do {
     sum += *p;
     p += L;
   } while (p < A + N);
 }
```

Table 1 shows the observed running times for different values of k. We see two jumps in the execution time. The running time increases almost eight-fold when k grows from 1 to 2, and it grows by a factor of about $5.6/4.0 = 1.4$ when the number of sequences exceeds the size of the translation lookaside buffer (= the cache used for translating addresses). The latter effect is not part of our analysis.

We propose a randomized strategy to overcome this dreadful behavior. More precisely, we propose to choose the starting addresses x_j of the arrays such that the values $x_j \bmod M$ are uniformly distributed independent random variables. The idea of placing objects at random positions in memory has been successfully used before, e.g., in a multi-disk merge-sort algorithm [BGV97] and in PRAM-simulation [MV84, Ran91]. Our strategy may be implemented as follows: When allocating memory for a sequence of length l, the algorithm chooses a random offset X with $0 \leq X < M$, allocates a memory block of length $l + X$, and puts the sequence at the end of this block. Note that in a system with virtual memory, this wastes only one page of physical memory since the beginning of the block is never accessed. We come back to this point in Section 7.

We show three upper bounds for the behavior of multiple sequence access under the randomized placement policy. We start with a simple and almost tight upper bound for direct mapped caches ($a = 1$) and then give a general bound which is close to the lower bound for $k \ll s$ and works well for the practically important cases $a = 2$ and $a = 4$. We continue with a better bound for large a and moderate k which is asymptotically tight in the sense that its predictions about how large k can be made such that the number of cache faults is still $\mathcal{O}(N/B)$ is only a constant factor away from the predictions of the lower bound.

The lower bound is based on the following simple two-phase access pattern: In the first phase it "randomly winds forward" the sequences, i.e., it accesses the first X_i elements of sequence i where the X_i are chosen uniformly and independently from $\{0, M-1\}$. In the second phase it accesses the sequences in round-robin fashion.

Theorem 1 *Given an a-way set associative cache with m cache blocks, $s = m/a$ cache sets, cache block size B, and LRU or FIFO replacement strategy. Let U_a denote the expected number of cache misses in any schedule of N sequential accesses to k sequences with randomized* *starting addresses*[2].

$$
\begin{aligned}
U_1 &\le k + \frac{N}{B}\left(1 + (B-1)\frac{k}{m}\right) && (1)\\
U_a &\le k + \frac{N}{B}\left(1 + (B-1)\left(\frac{k\alpha}{m}\right)^a + \frac{1}{m/(k\alpha)-1} + \frac{k-1}{s-1}\right) \quad \textit{for } k \le \frac{m}{\alpha} && (2)\\
U_a &\le k + \frac{N}{B}\left(1 + (B-1)\left(\frac{k\beta}{m}\right)^a + \frac{1}{m/(k\beta)-1}\right) \quad \textit{for } k \le \frac{m}{2\beta} && (3)\\
U_a &\ge \frac{N}{B}\left(1 + (B-1)P_{\text{tail}}\left(k-1, \frac{1}{s}, a\right)\right) - kM && (4)\\
&\ge \frac{N}{B}\left(1 + (B-1)\left(\frac{(k-a)\alpha}{m}\right)^a\left(1-\frac{1}{s}\right)^k\right) - kM && (5)\\
U_1 &\ge \frac{N}{B}\left(1 + (B-1)\frac{k-1}{m+k-1}\right) && (6)
\end{aligned}
$$

where $\alpha = \alpha(a) = a/a!^{1/a}$,[3] $P_{\text{tail}}(n,p,a) = \sum_{i\ge a}\binom{n}{i}p^i(1-p)^{n-i}$ *is the cumulative binomial probability and* $\beta := 1 + \alpha(\lceil ax\rceil)$ *where* $x = x(a) = \inf\{0 < z < 1 \,:\, z + z/\alpha(\lceil az\rceil) = 1\}$.[4]

We will prove Theorem 1 in Sections 4 and 5. In the remainder of this section we will discuss the bounds and give numerical examples. All our bounds are of the form $p \cdot N + C$, where C does not depend on N. We call p the *cache miss probability.* We first discuss C and then turn to p. We have not investigated under what circumstances similar bounds on the total number of chache faults can be guaranteed with high probability. In our model, such bounds are likely to depend on k, M, B, and a but not on N since an adversary could enforce repeated occurences of the same cache fault patterns.

The term k in the upper bounds accounts for the possible extra block per sequence due to randomization. The term $-kM$ in the lower bound accounts for the fact that the adversary in our lower bound advances all sequences by a random amount in the range 0 to $M-1$. We are not able to count cache faults in this set-up phase.

We turn to the cache miss probability next. With $r = k/m$, the ratio between the number of sequences and the number of cache blocks, our bounds for the cache miss probability essentially become

[2] It is sufficient if the starting addresses are $(a+1)$-wise independent.

[3] We have $1 \le \alpha < e$, $\alpha(1) = 1$, $\alpha(2) \approx 1.41$, $\alpha(3) \approx 1.66$, $\alpha(4) \approx 1.81$, $\alpha(8) \approx 2.13$, and $\alpha(\infty) = e \approx 2.71$

[4] We have $\beta(1) = 2$, $\beta(2) = 1 + \alpha(2) \approx 2.41$, $\beta(4) = 1 + \alpha(3) \approx 2.66$, and $\beta(\infty) = 1 + e \approx 3.71$. The function $f(x) = x + x/\alpha(\lceil x \cdot a\rceil)$ is piece-wise linear with discontinuities at $x = i/a$ for integer i. The function value decreases at discontinuities and $f(0+) = 0$ and $f(1) = 1 + 1/\alpha(\lceil a\rceil) > 1$. This implies that $x(a)$ exists. Let $j(a) = \lceil x(a)a\rceil$. Then $x(a) = \alpha(j(a))/(1+\alpha(j(a)))$ and hence $\beta(a) = \alpha(j(a))/x(a) = 1 + \alpha(j(a))$. Thus $2 \le \beta(a) \le 1 + \alpha(a)$.

$$
\begin{aligned}
p_1 &\leq \frac{1}{B}(1 + (B-1) \cdot r) && (7)\\
p_1 &\geq \frac{1}{B}(1 + (B-1) \cdot \frac{r}{1+r}) && (8)\\
p_a &\leq \frac{1}{B}(1 + (B-1)(r\alpha)^a + r\alpha + ar) \quad \text{for } r \leq \frac{1}{\alpha} && (9)\\
p_a &\leq \frac{1}{B}(1 + (B-1)(r\beta)^a + r\beta \qquad\qquad \text{for } r \leq \frac{1}{2\beta} && (10)\\
p_a &\geq \frac{1}{B}(1 + (B-1)(r\alpha)^a \left(1 - \frac{1}{s}\right)^k && (11)
\end{aligned}
$$

Every strategy has to incur at least one cache fault per memory block, the so-called first reference miss. The term $1/B$ which occurs in all bounds accounts for the first reference misses. The remaining terms bound the number of conflict misses. A conflict miss arises when a cache block is evicted before all elements in the block are scanned. The number of conflict misses can be made small by restricting the number of sequences. Observe that the upper bounds converge to $1/B$ for r going to zero. More generally, inequality (3) tells us that the number of misses is $O(N/B)$ if $r \leq 1/(2\beta)$ and $(B-1)(r\beta)^a = O(1)$. Both conditions are satisfied if $k \leq m/\max(B^{1/a}, 2\beta)$.

The upper and lower bounds for p_1 basically agree; the upper bound lies only by a factor $1 + r$ above the lower bound.

For $a \geq 2$ the situation is more complicated. For small r, (10) is better than (9) since its derivative at $r = 0$ is smaller, for large r, (9) is better than (10) since $\alpha < \beta$. The upper bounds are close to the lower bound as long $(1 - 1/s)^k \approx 1$ and $(\alpha + a)r \ll 1$. Both conditions are satisfied if $k \ll s$.

We next give a numerical example based on the properties of the MIPS R1000 [MIP98]. This processor has a 2-way associative second level cache with cache blocks of 256 bytes. Assuming data units of 4 bytes this amounts to $B = 64$. A possible size for the cache memory is 4 Mbytes or $M = 2^{20}$; we get $m = M/B = 2^{14}$. Consider the case of $k = 512$ sequences. For a direct mapped cache ($a = 1$), inequality (1) bounds the number of conflict misses to $(63 \cdot 512/2^{14})N/B \approx 2N/B$, i.e, twice the number of first reference misses. The upper bounds holds for any access pattern. Inequality (6) gives a lower bound of $(63 \cdot 511/(2^{14} + 512))N/B \approx 2N/B$ conflict misses for some access pattern. Upper and lower bound are essentially the same. For a two-way associative cache ($a = 2$) inequality (2) bounds the number of conflict misses to $((63/2) \cdot 16^{-2} + 1/(16\sqrt{2} - 1) + 1/16)N/B \approx (1/8 + 1/22 + 1/16)N/B \approx 0.23N/B$, i.e., less than one fourth the number of first reference misses. Thus a two-way associate cache does significantly better than a direct mapped cache. Inequality (5) gives a lower bound of $63(511\sqrt{2}/2^{14})^2 \cdot (1 - 2^{-13})^{512} N/B \approx 0.12N/B$ conflict misses; direct evaluation of inequality (4) gives a lower bound of $0.22N/B$ conflict misses.

Figure 1 plots our bounds for the cache miss probability for $a = 1$, $a = 2$, and $a = 4$ and $M = 2^{20}$ and $B = 2^{64}$. In the upper bounds the minimum of inequalities (2) and (3) is taken. The bounds clearly separate the behavior of direct mapped, 2-way associative and 4-way associative caches. Moreover, for $a = 1$ and $a = 2$ the lower and upper bounds are quite close together in particular in the interesting range of small miss probabilities.

Our bounds can also be used to predict the number of sequences that can be supported with a certain cache miss probability. For example, for $a = 1$ cache miss probability $2/B$

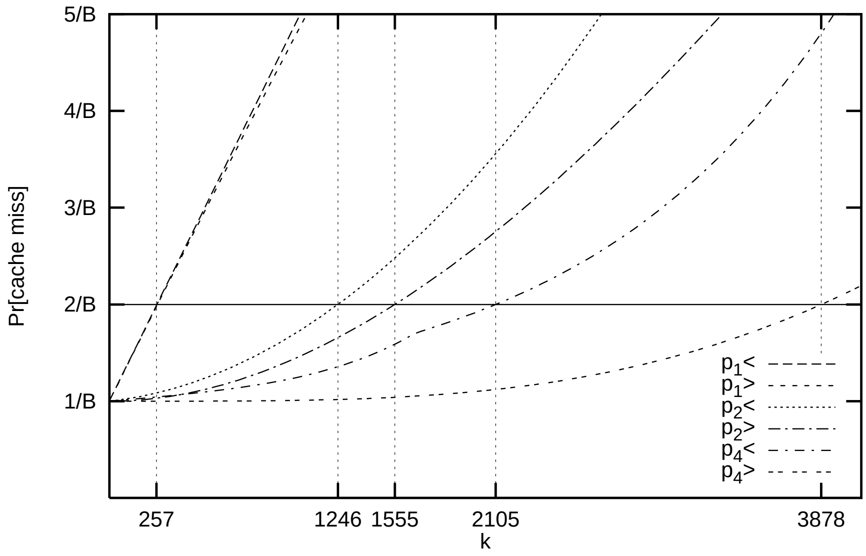

Figure 1: Cache miss probabilities for $B = 64$, $M = 2^{20}$. The upper and lower bound on the cache miss probability p_a for an a-way associative cache is denoted $p_a>$ and $p_a<$, respectively. Note that the curve labels are sorted by miss probability. The bounds clearly separate different degrees of associativity. Observe the little bump in $p_4<$; for $k \leq 1600$ inequality (3) gives the better bound and for $k > 1600$ inequality (2) gives the better bound.

is guaranteed for up to 252 sequences and can be reached with 263 sequences, for $a = 2$ is guaranteed for up to 1246 sequences and can be reached with 1255 sequences, and for $a = 4$ is guaranteed for up to 2105 sequences and can be be reached with 3878 sequences.

3 Related Work

The influence of associativity has been intensively studied in computer architecture and compiler design (e.g. [HP96]) and in numerical software. In the former areas, experiments (simulations or actual execution) are the method of choice, as at this point of time only experiments are able to model the complex interactions between advanced features for mitigating cache faults like victim caches, write buffers or out-of-order execution. Unfortunately, simulations only yield results for one particular combination of architecture, algorithm, implementation, compiler and input and hence the insights from simulations are hard to generalize. This is undesirable for algorithm designers who would like to estimate the performance of a family of algorithms for many systems and all possible inputs. Simulations are also unable to quantify the relative power of different machine models.

Algorithms of numerical analysis, e.g., matrix multiplication, frequently exhibit regular and predictable access patterns to memory and hence it is often possible to find a memory layout that is guaranteed to work well even in caches with small associativity. A detailed study of matrix multiplication can be found in [ERS98].

Early studies of the influence of associativity use the *independent reference model* [Rao78] where it is assumed that cache blocks are accessed according to some fixed probability distribution (possibly combined with a simple deterministic access pattern [LFM99]). The independent reference model is not able to distinguish different degrees of associativity. Direct mapped caches work as well as caches of higher associativity. Sequence traversal does not fall under the independent reference model.

The paper which comes closest to ours is by Fricker and Robert [FR91]. They proposed a model for accessing sequences. However, their model is limited to one particular access schedule while our model allows an adversary to schedule the accesses. Furthermore, their model can only be evaluated numerically for a particular set of parameters. The time for the evaluation grows at least quadratically in the number of memory accesses, whereas our analysis yields closed form and nearly matching upper and lower bounds.

In order to give the reader a basis for comparing our results with the emulation techniques by [FLPR99, SC00], we outline the emulation approach of [SC00] applied to sequence access for direct mapped cache. We maintain an array $\mathcal{B}$ of size mB in main memory in addition to the memory containing the sequences (which we call sequence memory in the sequel). Assume that the pointer into the i-th sequence is at address p_i and let $d_i = \lfloor p_i/B \rfloor$. Then addresses d_iB, d_iB+1, ... , d_iB+B-1 of the i-th sequence are stored in the i-th block of $\mathcal{B}$ and all reading and writing to sequence data is done in the buffer. When p_i reaches d_iB+B, the i-th block of the buffer is written back to sequence memory and filled with the next block of the i-th sequence. The accesses to $\mathcal{B}$ incur no cache faults, because the buffer maps without conflicts into cache memory. In this way the number of cache faults will be $2N/B$ and the instruction count is increased by $\Theta(N)$. The additional instructions arise for copying the data into and from the buffer and for checking each access to a sequence element whether it crosses a block boundary.

Interestingly, higher associativity makes this emulation approach more difficult, although with some additional overhead the problems can be solved. The emulation approach of Frigo et al. has additional overhead for emulating a fully associative LRU cache but has the advantage that decisions about the cache content are made automatically.

4 The Proof of the Upper Bounds

We prove Inequalities (1)–(3). The access to the first element of any sequence is always a cache miss. For any sequence b and any j, $2 \le j \le N_b$, let x_{bj} be the probability that we have a cache miss when accessing the j-th element of sequence b where N_b is the number of accesses made to sequence b. The expected number of cache misses is therefore given by

$$U_a = k + \sum_{b=1}^{k} \sum_{j=2}^{N_b} x_{bj}.$$

The j-th element is either the first element in its memory block (probability $1/B$) or it is not (probability $(B-1)/B$). In the first case, we always have a cache miss (a so-called *first reference miss*). In the second case, we have a cache miss (a so-called *conflict miss*) if there were at least a accesses to sequences different from b between the $(j-1)$-th and the j-th access to b that went to the cache set addressed by sequence b in its j-th access. We use c_{bj} to denote the probability of this event. Then

$$x_{bj} = \frac{1}{B} + \frac{B-1}{B} \cdot c_{bj}$$

and hence the expected number of cache faults is bounded by

$$U_a \le k + \frac{N}{B} + \frac{B-1}{B} \sum_{b=1}^{k} \sum_{j=2}^{N_b} c_{bj}.$$

The hard part is now to analyze $\sum_{j=2}^{N_b} c_{bj}$ for a particular sequence b. Let $\bar{B} := \{1, \ldots, k\} \setminus \{b\}$. Let n_{ij} be the number of accesses to sequence i between the $(j-1)$-th and the j-th access to sequence b and let $n_j = \sum_{i \in \bar{B}} n_{ij}$ and note that $\sum_j n_j \le N - N_b$.

We look at three cases. Section 4.1 covers the easiest case $a = 1$, i.e., direct mapped caches. This case introduces some of the main ideas for the analysis and also yields a slightly better bound than the case of general a. We derive inequality (2) in Section 4.2 and inequality (3) in Section 4.3.

4.1 Direct Mapped Caches (Inequality (1))

Let p_{ij} denote the probability that between the $(j-1)$-th and the j-th access to sequence b, sequence i uses the same cache set as sequence b. We have: $p_{ij} = 0$ if $n_{ij} < 1$, $p_{ij} = 1/s$ for $n_{ij} = 1$ since the access goes to a random cache set, and p_{ij} grows linearly with n_{ij} for larger values until all s cache sets are affected by the recent accesses to sequence i. Thus,

$$p_{ij} = \min(1, \frac{n_{ij} + B - 1}{sB}) \text{ if } n_{ij} > 0, \text{ 0 otherwise} \tag{12}$$

and hence

$$c_{bj} \;\le\; \sum_{i \in \bar{B}} p_{ij} \;\le\; \sum_{i \in \bar{B}} \frac{n_{ij} + B - 1}{sB} \;=\; \frac{(B-1)(k-1) + n_j}{sB} \quad .$$

By summing over all sequences we get

$$\begin{aligned}
\sum_{b=1}^{k} \sum_{j=2}^{N_b} c_{bj} \;&\le\; \sum_{b=1}^{k} \frac{(N_b - 1)(B-1)(k-1) + (N - N_b)}{sB} \\
&=\; \frac{(N-k)(B-1)(k-1) + N(k-1)}{sB} \\
&=\; \frac{k-1}{sB}(NB - k(B-1)) \;\le\; \frac{Nk}{s} \;=\; \frac{Nk}{m} \quad .
\end{aligned}$$

Multiplication with $(B-1)/B$ completes the proof of Inequality (1).

4.2 Inequality (2)

We partition the set of conflict misses into two sets: The first set contains all misses where some sequence occupies at least two entries of the cache set under consideration and the second set contains all other conflict misses. The first set is the subject of the next lemma.

Lemma 1 *There are at most $\frac{N}{B} \cdot \frac{k-1}{s-1}$ conflict misses in accesses to cache sets in which some other sequence occupies at least two entries of some cache set.*

Proof: Consider an access to a particular sequence b. If it causes a conflict miss and some other sequence b' is occupying at least two entries of the accessed cache set at the time of the miss then b' has made at least $sB - B + 1$ accesses after the last access to b (due to LRU or FIFO replacement). Since there are at most $N - N_b$ accesses by other sequences overall, sequence b can suffer at most $\lfloor (N - N_b)/(sB - B + 1) \rfloor$ of the conflict misses under consideration. Summing over all sequences yields

$$\frac{N(k-1)}{sB - B + 1} \leq \frac{N}{B} \cdot \frac{k-1}{s-1}$$

misses. ■

To bound the remaining conflict misses, we bound c_{bj} for those cases where no other sequence made more than $sB - B + 1$ accesses. To observe a cache miss, there must then be a set $I \subseteq \bar{B}$ of at least a other sequences hitting the same cache set as sequence b. Since the sequences are shifted independently we can estimate

$$c_{bj} \leq \min\Bigl(1, \sum_{I \subseteq \bar{B}, |I| = a} \prod_{i \in I} p_{ij}\Bigr)$$

where p_{ij} is defined as in Equation (12).

To bound $p_j := \sum_{I \subseteq \bar{B}, |I| = a} \prod_{i \in I} p_{ij} = \sum_{I \subseteq \bar{B}, |I| = a} \prod_{i \in I} (B - 1 + n_{ij})/(sB)$, we view the latter expression as a function of the n_{ij} and ask for its maximum subject to the constraints that $n_j = \sum_{i \in \bar{B}} n_{ij}$ is fixed and $0 \leq n_{ij} \leq sB - B$. With $u_i = (B - 1 + n_{ij})/(sB)$ and $h = a$ we conclude from Lemma 2 below that the maximum is attained if all n_{ij} have the same value $n_j/(k-1)$. We obtain

$$\begin{aligned} p_j &\leq \sum_{I \subseteq \bar{B}, |I| = a} \left(\left(\frac{n_j}{k-1} + B - 1 \right) \frac{1}{sB} \right)^a = \binom{k-1}{a} \left(\left(\frac{n_j}{k-1} + B - 1 \right) \frac{1}{sB} \right)^a \\ &\leq \frac{(k-1)^a}{a!} \left(\left(\frac{n_j}{k-1} + B - 1 \right) \frac{1}{sB} \right)^a = \left(\frac{n_j + (k-1)(B-1)}{M/\alpha} \right)^a . \end{aligned}$$

We next bound $\sum_{j=2}^{N_b} c_{bj} = \sum_{j=2}^{N_b} \min(1, p_j)$ by applying Lemma 3 below with $A = N_b - 1 \leq N_b$, $C = (k-1)(B-1)$, $D = M/\alpha$, and $E = N - N_b$ and obtain

$$\begin{aligned} \sum_{j=2}^{N_b} c_{bj} &\leq \frac{N - N_b}{M/\alpha - (k-1)(B-1)} + N_b \left(\frac{(k-1)(B-1)}{M/\alpha} \right)^a \\ &\leq \frac{N - N_b}{M/\alpha - (k-1)(B-1)} + N_b \left(\frac{k\alpha}{m} \right)^a . \end{aligned}$$

By summing over all sequences we get

$$\begin{aligned} \frac{B-1}{B} \sum_{b=1}^{k} \sum_{j=2}^{N_b} c_{bj} &\leq \frac{B-1}{B} \left(\frac{N(k-1)}{M/\alpha - (k-1)(B-1)} + N \left(\frac{k\alpha}{m} \right)^a \right) \\ &= \frac{N}{B} \left(\frac{(k-1)(B-1)}{M/\alpha - (k-1)(B-1)} + (B-1) \left(\frac{k\alpha}{m} \right)^a \right) \\ &\leq \frac{N}{B} \left(\frac{1}{m/(k\alpha) - 1} + (B-1) \left(\frac{k\alpha}{m} \right)^a \right) . \end{aligned} \tag{13}$$

Summing this bound, the $k + N/B$ first reference misses, and the conflict misses accounted for in Lemma 1 completes the proof of Inequality (2) except for the proof of our two technical lemmas.

Lemma 2 *Subject to the constraints $u_i \geq 0$ and $\sum_{i \in \bar{B}} u_i = U$, the function $f : (u_i)_{i \in \bar{B}} \mapsto \sum_{I \subseteq \bar{B}, |I|=h} \prod_{i \in I} u_i$ is maximized for $u_i = U/(k-1)$ for all $i \in \bar{B}$.*

Proof: The claim is obvious for $h = 1$ or $k - 1 = |\bar{B}| = 1$. So assume $h \geq 2$ and $k \geq 3$. Observe first that f is continuous and that the domain of definition is compact. Thus the function attains its maximum. We now show indirectly that all arguments are equal at the maximum. Assume w.l.o.g. that $\{1, 2\} \subseteq \bar{B}$ and let $B' = \bar{B} \setminus \{1, 2\}$. Each term in f either contains none of u_1 or u_2, exactly one, or both. We may therefore rewrite f as

$$\sum_{I \subseteq B', |I|=h} \prod_{i \in I} u_i + \sum_{I \subseteq B', |I|=h-1} (u_1 + u_2) \prod_{i \in I} u_i + \sum_{I \subseteq B', |I|=h-2} (u_1 \cdot u_2) \prod_{i \in I} u_i.$$

For fixed values of u_i, $i \in \bar{B} \setminus \{1, 2\}$ and fixed value of $u_1 + u_2$, the maximum is attained if $u_1 = u_2$. We conclude that all components of the maximizing argument are equal. ■

Lemma 3 *Let $A \in \mathbb{N}$ and consider positive parameters C, D, E with $D > C$. Then*

$$f(\mathbf{z}) := \sum_{j=1}^{A} \min\left(1, \left(\frac{z_j + C}{D}\right)^h\right) \leq \frac{E}{D - C} + A\left(\frac{C}{D}\right)^h,$$

where f is understood as a mapping from $\mathbb{R}^A_{\geq 0}$ to $\mathbb{R}$ under the constraint $\sum_{j=1}^{A} z_j = E$.

Proof: Clearly, $f(\mathbf{z}) \leq A$ and hence the claim is trivial if $E/(D - C) > A$. So assume that $E \leq A(D-C)$. In the maximizing argument, there will be no z_j larger than $D - C$ and hence it suffices to bound $f(\mathbf{z}) = \sum_{j=1}^{A} ((z_j + C)/D)^h$ subject to the constraints $0 \leq z_j \leq D - C$ and $\sum_{j=1}^{A} z_j = E$. For convenience, define $g(x) := ((x + C)/D)^h$.

We claim that in a maximizing argument of f at most one z_j has a value strictly between 0 and $D - C$. This follows from the observation that, since g is concave, we have $g(x) + g(y) < g(x - \epsilon) + g(y + \epsilon)$ for any two x and y with $0 < x \leq y < (D - C)$ and $\epsilon = \min(x, D - C - y)$.

The maximizing argument of f therefore has $\lfloor A/(D - C) \rfloor$ variables equal to $D - C$, one variable equal to $\delta(D - C)$ where $\delta = A/(D - C) - \lfloor A/(D - C) \rfloor$, and the remaining $A - \lfloor A/(D - C) \rfloor - 1$ variables equal to zero. Thus

$$f(\mathbf{z}) \leq \left\lfloor \frac{A}{D - C} \right\rfloor + g(\delta(D - C)) + \left(A - \left\lfloor \frac{A}{D - C} \right\rfloor - 1\right)\left(\frac{C}{D}\right)^h.$$

Next observe that $g(\delta(D-C)) \leq (1-\delta)g(0) + \delta g(D-C) = (1-\delta)(C/D)^h + \delta$ by the concavity of g. Thus $f(\mathbf{z}) \leq A/(D-C) + A(C/D)^h$. ■

4.3 Inequality (3)

In the proof of Inequality (2) we distinguished between two kind of accesses: cache misses where some other sequence occupied at least two entries of the accessed cache set and cache misses where at least a other sequences accessed the same cache block. Lemma 1 dealt with the former case of faults. Its proof does not exploit random placement and gives a bound that increases with a. In order to get a better bound we have to study more carefully under what circumstances sequences occupy more than one block of a cache set.

Lemma 4 *Let $d_{ij} = \lfloor (n_{ij}+B-1)/(sB) \rfloor$, and let $z_{ij} = n_{ij} - d_{ij} \cdot sB$. Then $-(B-1) \le z_{ij} < sB-(B-1)$ and either d_{ij} or $d_{ij}+1$ of the accesses to sequence i go to the same cache set as the j-th access to b. The probability for the second event is bounded by $p_{ij} = (z_{ij}+B-1)/(sB)$.*

Proof: We have $d_{ij} \ge 0$ and hence the claim is certainly true for $n_{ij} = 0$. So assume $n_{ij} > 0$. If $n_{ij} = 1$, sequence i uses one memory block. For each increment of n_{ij}, the expected number of memory blocks occupied by the n_{ij} elements of sequence i grows by $1/B$, since the probability that the additional element starts a new memory block is $1/B$. The expected number of memory blocks occupied by the n_{ij} elements is therefore equal to $1 + (n_{ij} - 1)/B$. For a fixed cache set, these memory blocks use an expected number of $(1 + (n_{ij} - 1)/B)/s$ cache blocks in the set. The actual number of cache blocks used is either the floor or the ceiling of this number. The latter case occurs with probability $(1+(n_{ij}-1)/B)/s-d_{ij} = (B-1+n_{ij}-d_{ij}\cdot sB)/(sB) = p_{ij}$. ∎

Let $d_j = \sum_{i\in\bar{B}} d_{ij}$. Between the $j-1$-th and the j-th access to sequence b at least d_j cache blocks of the cache set containing the j-th access to b are replaced. If a are to replaced, there must be a subset $I \subseteq \bar{B}$ of cardinality $h_j := a - d_j$ such that all sequences $i \in I$ use $d_{ij} + 1$ cache blocks in the cache set. We want to bound the probability c_{bj} of this event. We use different arguments for small and large values of h_j and therefore group the indices j according to the value h_j. For h, $0 \le h \le a$, let $J^{(h)}$ be the set of j with $h_j = h$, let $N_b^{(h)} = \left|J^{(h)}\right|$, let $N^{(h)} = \sum_{j\in J^{(h)}} n_j$, and let $Z^{(h)} = \sum_{j\in J^{(h)}} z_j$. In the proof of Inequality (2) we only distinguished the sets $J^{(a)}$ and $\cup_{h<a} J^{(h)}$ and used Lemma 1 to deal with the latter set.

Consider first the case that $h \le x\cdot a$ where $x = x(a)$ is defined as in the statement of Theorem 1; we refine the argument used in the proof of Lemma 1. For any $j \in J_h$ we have $d_j \ge (1-x)a$ and hence (using $d_{ij} = \lfloor (n_{ij}+B-1)/(sB)\rfloor \le (n_{ij}+B)/(sB)$)

$$n_j = \sum_{i\in\bar{B}} n_{ij} \ge \sum_{i\in\bar{B}} (d_{ij}\cdot sB - B) \ge (1-x)\cdot a \cdot sB - kB = M/\beta - kB.$$

Thus

$$\sum_{j\in J^{(h)}} c_{bj} \le \left|J^{(h)}\right| \le \sum_{j\in J_h} \frac{n_j}{M/\beta - kB}.$$

We next consider $h \ge x\cdot a$ and hence $h \ge \lceil x\cdot a\rceil$. Consider any $j \in J_h$. There must be a subset $I \subseteq \bar{B}$ of cardinality h_j such that all sequences $i \in I$ use $d_{ij}+1$ cache blocks in the cache set under consideration. We want to bound

$$c_{bj} \le \min(1, p_j) \text{ where } p_j := \sum_{I\subseteq\bar{B},|I|=h_j} \prod_{i\in I} p_{ij} = \sum_{I\subseteq\bar{B},|I|=h_j} \prod_{i\in I} \frac{z_{ij}+B-1}{sB} .$$

Maximizing p_j using Lemma 2 (with $z_j = \sum_{i\in\bar{B}} z_{ij}$, $-(B-1) \le z_{ij} < sB-(B-1)$, and $u_i = (z_{ij}+B-1)/(sB)$) we get $z_{ij} = z_j/(k-1)$. We obtain

$$\begin{aligned} p_j &\le \sum_{I\subseteq\bar{B},|I|=h_j} \left(\left(\frac{z_j}{k-1}+B-1\right)\frac{1}{sB}\right)^{h_j} = \binom{k-1}{h_j}\left(\left(\frac{z_j}{k-1}+B-1\right)\frac{a}{M}\right)^{h_j} \\ &\le \frac{(k-1)^{h_j}}{h_j!}\left(\left(\frac{z_j}{k-1}+B-1\right)\frac{a}{M}\right)^{h_j} = \left(\frac{(z_j+(k-1)(B-1))a}{M(h_j!)^{1/h_j}}\right)^{h_j} \le \left(\frac{z_j+kB}{M/\beta}\right)^{h_j}, \end{aligned}$$

where the last inequality uses $h_j \geq \lceil x \cdot a \rceil$ and hence

$$\frac{a}{(h_j!)^{1/h_j}} \leq \frac{a}{\lceil xa \rceil !^{1/\lceil xa \rceil}} \leq \frac{\lceil xa \rceil}{x \cdot \lceil xa \rceil !^{1/\lceil xa \rceil}} = \frac{\alpha(\lceil xa \rceil)}{x} = 1 + \alpha(\lceil xa \rceil) = \beta \ .$$

We can now estimate $\sum_{j \in J^{(h)}} \min(1, p_j)$. We apply Lemma 3 with $A = N_b$, $C = kB$, $D = M/\beta$, and $E = Z^{(h)}$ and obtain

$$\sum_{j \in J^{(h)}} \min(1, p_j) \leq \frac{Z^{(h)}}{M/\beta - kB} + N_b^{(h)} \left(\frac{k\beta}{m}\right)^h .$$

Observe next that $Z_j + (a - h_j)sB = N_j$ for all j. Summation over $j \in J^{(h)}$ yields $Z^{(h)} + N_b^{(h)} \cdot (a - h) \cdot sB = N^{(h)}$. We may therefore rewrite the right hand side as

$$\frac{N^{(h)} - N_b^{(h)}(a-h)sB}{M/\beta - kB} + N_b^{(h)} \left(\frac{k\beta}{m}\right)^h = \frac{N^{(h)}}{M/\beta - kB} + N_b^{(h)} \left[\left(\frac{k\beta}{m}\right)^h - \frac{(a-h)s}{m/\beta - k} \right] .$$

Consider the expression in the square bracket. We claim that this expression is non-positive for $h < a$. Since $k \leq m/(2\beta)$, the first term is bounded by $(1/2)^h$. The second term is at least $(a-h)s/(m/\beta) = (a-h)\beta/a$. Substituting $h = a - d$ it therefore suffices to show $(1/2)^{a-d} \leq d\beta/a$ or $a/2^a \leq \beta d/2^d$ for $d \geq 1$. For $d \geq 2$, the inequality holds since $\beta \geq 2$ and since $x \mapsto x/2^x$ is decreasing for $x \geq 2$. For $d = 1$ the inequality holds, since $a/2^a \leq 1 \leq \beta(a)/2$ for all a.

We conclude that

$$\sum_{j \in J^{(h)}} c_{bj} \leq \sum_{j \in J^{(h)}} \min(1, p_j) \leq \frac{N^{(h)}}{M/\beta - kB} + N_b^{(h)} \left(\frac{k\beta}{m}\right)^a$$

for all h. Summing the bound over all h yields

$$\sum_{j=2}^{N_b} c_{bj} \leq \frac{N - N_b}{M/\beta - kB} + N_b \left(\frac{k\beta}{m}\right)^a .$$

By summing over all sequences we obtain

$$\frac{B-1}{B} \sum_{b=1}^{k} \sum_{j=2}^{N_b} c_{bj} = \frac{N}{B} \left(\frac{1}{m/\beta - 1} + (B-1) \left(\frac{k\beta}{m}\right)^a \right) .$$

This completes the proof of Equation (3).

5 The Lower Bounds (Inequalities (4), (5), and (6))

We prove Inequalities (4), (5), and (6). Recall the two-phase access strategy. In the first phase the adversary "randomly winds forward" the sequences, i.e., it accesses the first X_i elements of sequence i where the X_i are chosen uniformly and independently from $\{0, M-1\}$. In the second phase it accesses the sequences in round-robin fashion.

After winding forward, we have a first reference miss with probability $1/B$. Furthermore, the cache sets currently accessed by the sequences are independent and uniformly distributed over the entire possible range regardless which starting addresses have been used and hence each access to the last $B-1$ elements of a memory block has the same probability P_{miss} of a cache fault. We have a cache miss if the $k-1$ preceding accesses to the other sequences have accessed the same cache set at least a times. The probability for this event is

$$P_{\text{miss}} := \sum_{i \geq a} \binom{k-1}{i} \left(\frac{1}{s}\right)^i \left(1 - \frac{1}{s}\right)^{k-1-i} .$$

We conclude that the expected number of cache misses is at least

$$\frac{N}{B} + \frac{(N-kM)(B-1)}{B} P_{\text{miss}} \geq \frac{N}{B}(1 + (B-1)P_{\text{miss}}) - kM.$$

This proves Inequality (4). For Inequality (5) we estimate P_{miss} by the first term of the sum. We have:

$$P_{\text{miss}} \;>\; \binom{k-1}{a} s^{-a} \left(1 - \frac{1}{s}\right)^{k-1-a} > \frac{(k-a)^a}{a!} s^{-a} \left(1 - \frac{1}{s}\right)^k = \left(\frac{(k-a)\alpha}{m}\right)^a \left(1 - \frac{1}{s}\right)^k .$$

For Inequality (6) we use $P_{\text{miss}} = 1 - \sum_{i<a} \binom{k-1}{i} \left(\frac{1}{s}\right)^i \left(1 - \frac{1}{s}\right)^{k-1-i}$. For $a = 1$ this amounts to $P_{\text{miss}} = 1 - (1 - 1/m)^{k-1}$. Next observe that

$$(1 - 1/m)^{k-1} = e^{(k-1)\ln(1-1/m)} \leq e^{-(k-1)/m} \leq 1/(1 + (k-1)/m),$$

where the first inequality follows from $\ln(1+x) \leq x$ for all x and the second inequality follows from $e^{-x}(1+x) \leq 1$ for $x > 0$ (observe that the line through the point $(-x, e^{-x})$ with slope e^{-x} intersects the y-axis below the value 1.). Thus $P_{\text{miss}} \geq 1 - 1/(1 + (k-1)/m) = (k-1)/(m+k-1)$.

6 A Generalization

In the preceding sections we studied the concept of scanning multiple sequences in its pure form. An application of the concept will also access additional data; there is need for sequence pointers and for data structures which decide which sequence is to be accessed next. In this section we will show that a large class of algorithms based on the concept runs well in cache memory.

We consider algorithms that scan k sequences of total length N and use additional data occupying w conflict-free memory blocks, i.e., no two are mapped to the same cache set. This will be the case if the additional data has size at most sB and is stored in contiguous words of memory. We refer to the additional data as the *working set*.

Theorem 2 *Let U_a denote our bound on the number of cache misses for an a-way associative cache defined in Theorem 1 and define $U_0 = N$. With the working set occupying w conflict-free memory blocks, the expected number of cache misses arising in the N accesses to the sequence data and any number of accesses to the working set, is bounded by $w + (1 - w/s)U_a + 2(w/s)U_{a-1}$.*

Proof: We first show how to charge the cache faults caused by accesses to data in the working set. The first fault of each memory block in the working set is charged to the memory block and each further fault is charged to the sequence access which caused the eviction of the block from the cache memory. In this way w faults are charged to the blocks in the working set and all faults of blocks in the working set are charged to some access.

Consider next the accesses to sequences data. Such an access goes either to a cache set not used by the working set (probability $1 - w/s$) or it goes to a cache set also used by the working set (probability w/s). For the former case of accesses the analysis of Theorem 1 applies and hence the number of cache faults in such accesses is bounded by $(1 - w/s)U_a$. For the second kind of accesses, we make two observations: we certainly upper bound the number of cache faults caused by sequence accesses if we restrict the size of a cache set to $a - 1$ and each access is charged one further fault (namely the access of the working set data which it might have evicted). Thus, the total number of charges to accesses of the second kind is bounded by $2(w/s)U_{a-1}$. ∎

The preceding theorem is easily extended to additional data of larger size. For example, if the additional data occupies l contiguous memory blocks, the expected number of cache faults is bounded by $l + (1 - w/s)U_{a-r} + 2(w/s)U_{a-r-1}$, where $w = l \bmod s$ and $r = \lfloor w/s \rfloor$.

7 Space Requirement

Our randomized placement strategy wastes virtual address space. Although the actual space used for the k sequences is at most $N + 2kB'$, where B' is the size of a virtual memory page, the virtual address space used might be as large as $N + kM$. For large k and large M this might be a problem. Consider the numerical example discussed at the end of Section 2: With $M = 2^{24}$ bytes of cache memory, a 2-way associative cache easily supports $k = 2^{10}$ sequences. The naive memory allocation scheme wastes $kM = 2^{34}$ virtual memory. On 32 bit systems this is impossible and even a 64 bit operating system might only work if these 16 Gbytes of space are available on the swapping disk. Fortunately, there are better allocation strategies.

7.1 Offline Allocation

We start with the case where all the sequence lengths $L_1, \ldots, L_k$ are known in advance. We choose offsets $X_i \in [0 \mathinner{..} M - 1]$, $i \in \{1, \ldots, M\}$ uniformly at random. Now we are still free to choose the order in which the sequences are allocated in memory. The task to find the allocation order with minimum memory requirement is a special case of the asymmetric travelling salesman problem. The problem turns out to be equivalent to the problem of *drum scheduling* where the task is to retrieve k blocks from a rotating storage device without seek delays. There is an $\mathcal{O}(k \log k)$ time optimal algorithm for this problem [Ful72] and a simple greedy algorithm that yields memory allocations that are optimal within an additive offset of M [SF73]. This algorithm works as follows: Assume that memory starts at address x. We look for the offset X_i that minimizes $(X_i - x) \bmod M$ and allocate it first. Then we set $x = X_i + L_i \bmod M$ and remove sequence i from the set of sequences to be allocated. The process of allocating a closest sequence is repeated until all sequences are allocated.

Unfortunately, even an optimal allocation might use a lot of memory in the worst case. However, since the offsets are random this is very unlikely. Sanders and Vöcking [SV01] have analyzed this *random arc allocation problem*. Let $\alpha = \frac{1}{kM} \sum_{i=1}^{k} (L_i \bmod M)$ denote the average sequence length in units of cache sizes. The expected memory overhead in units of cache sizes is

$$\mathbf{E}[R] = \mathcal{O}(\sqrt{k\alpha(1-\alpha)\log\frac{1}{\alpha(1-\alpha)}}) = \mathcal{O}(\sqrt{k}) \ .$$

Experiments indicate that the constant factor involved is at most 1.65. For our numerical example, we would get memory overhead at most 38MB rather than 16GB.

7.2 Online Allocation

A scheme that works even if the sequence lengths are not known in advance maintains a data structure with all free intervals of memory. When a sequence of size L_i and random offset X_i is to be allocated, the data structure is searched for a free interval which can accommodate the sequence with this offset. Then, only the space actually needed for the sequence is allocated. We give an analysis of this scheme for the case that all sequences have the same length L. We assume that memory starts at logical address 0. An allocation strategy finds values d_i, $i \in \{1, \ldots, k\}$ such that sequence i starts at address $d_i M + X_i$, $d_i \in \mathbb{N}$ such that no two sequences overlap in memory. We propose *first fit* online allocation where d_i is minimized for each sequence based on the previous allocation. We first show that first fit finds allocations that are at most a factor two away from an optimal offline allocation.

Theorem 3 *First fit is 2-competitive if all allocated sequences have size L.*

Proof: The claim is trivially true for $L \geq M$ because the algorithm will produce unused pieces of memory of size below M. So assume $L \leq M$. Let $d_i = \max_{1 \leq j \leq k} d_j$ for a first fit allocation. The memory consumption of first fit is $f = d_i M + X_i + L$. Now consider the memory locations $l_j = jM + X_i$ and $r_j = jM + X_i + L - 1$ for $0 \leq j < d_i$. For each such j there must be a sequence occupying memory location l_j or memory location r_j. (Assume the contrary. Since all sequences have length L there cannot be any occupied memory cell in the interval $[jM + X_i, jM + X_i + L - 1]$ and hence first fit would have choosen a smaller d_i.)

Hence there is an offset x ($x = X_i$ or $x = X_i + L - 1 \bmod M$) where at least $\lceil d_i/2 \rceil + 1$ sequences require a memory cell. Now consider the case where $d_i = 2r - 1$ is odd. Then even an optimal offline allocation needs $f^* = M \lceil d_i/2 \rceil M + X_i + 1 = rM + X_i + 1$ memory. For $L \leq M$ we have

$$\frac{f}{f^*} = \frac{(2r-1)M + X_i + L}{rM + X_i + 1} = \frac{2rM + X_i - (M - L)}{rM + X_i + 1} \leq 2 \ .$$

For even d_i we distinguish two subcases. If we can choose $x = X_i + L - 1$ we have

$$\frac{f}{f^*} = \frac{d_i M + X_i + L}{M d_i/2 + X_i + L} \leq 2 \ .$$

Otherwise, there must be at least $d_i/2 + 1$ sequences overlapping offset $x = X_i$ and we have

$$\frac{f}{f^*} = \frac{d_i M + X_i + L}{(d_i/2 + 1)M + X_i + 1} \leq 2 \ .$$

■

Theorem 3 allows us to translate the performance guarantee for offline allocation from Section 7.1. The expected memory consumption is bounded by

$$2N + \mathcal{O}(M\sqrt{k}) \ .$$

8 Applications

There is a large and growing body of work on algorithms designed for hierarchical memory. They run under the names external memory algorithms or cache-aware algorithms or cache-oblivious algorithms. A recent survey of external memory algorithms was given by Vitter [Vit98]. Many algorithms designed for hierarchical memory use the concept of scanning multiple sequences. In almost all cases the analysis of the algorithms assumes full associativity and user-controlled block replacement.

User-controlled block replacement was addressed by Frigo et al [FLPR99]. They point out that a cache of size $2M$ with the LRU-replacement strategy produces at most twice as many cache faults as a cache of size M and user-controlled replacement strategy. The result holds for fully associative caches.

Our results imply that external memory algorithms based on sequence traversal will run with essentially the same number of memory faults, in a cache with small associativity provided

- the number of sequences is sufficiently small, e.g., $k \leq m/\max(B^{1/a}, \min(a, 2\beta))$,
- the starting addresses of the sequences are randomized, and
- the data manipulated in addition to the sequence data is small.

Specifically, the following algorithms will run well in cache memories of small associativity: k-way merge sort [NBC+94, LL97], radix sort, the *cache oblivious*[5] algorithms Funnelsort and Distribution sort described in [FLPR99], buffer trees [Arg95], external memory list ranking [Sib97], cache-aware generation of permutations [San98], flash sort [Neu98, RR99], the cache-aware priority queue of [San99b], all solutions to the class of batched geometric problems mentioned in the survey [Vit98], the general techniques for obtaining external algorithms from parallel algorithms [CGG+95, SK97, DDH97].

Some of the papers mentioned above, e.g. [NBC+94, LL97, San98, San99b], report the experimental fact that their algorithms run well in cache memory. The experiments were performed on random data. For some of these applications it should be easy to prove this observation.

References

[Arg95] L. Arge. The buffer tree: A new technique for optimal I/O-algorithms. In *4th Workshop on Algorithms and Data Structures*, number 955 in LNCS, pages 334–345. Springer, 1995.

[BGV97] R. D. Barve, E. F. Grove, and J. S. Vitter. Simple randomized mergesort on parallel disks. *Parallel Computing*, 23(4):601–631, 1997.

[CGG+95] Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamasia, D. E. Vengroff, and J. S. Vitter. External memory graph algorithms. In *6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 139–149, 1995.

[5]Implementing cache oblivious algorithms using our approach makes them cache aware because random offsets are choosen modulo the cache size. But we can at least obtain algorithms that are cache efficient on all levels of the cache hierarchy by choosing a random offset modulo the least common multiple of all cache sizes. We view it as likely that one can get completely cache oblivious algorithms by choosing offsets modulo the input size N. We would like to thank an anonymous referee for this observation.

[DDH97] F. Dehne, W. Dittrich, and D. Hutchinson. Efficient external memory algorithms by simulating coarse-grained parallel algorithms. In *ACM Symposium on Parallel Architectures and Algorithms*, pages 106–115, Newport, RI, 1997.

[ERS98] N. Eiron, M. Rodeh, and I. Steinwarts. Matrix multiplication: A case study of algorithm engineering. In K. Mehlhorn, editor, *2nd Workshop on Algorithm Engineering*, number MPI-I-98-1-019, ISSN: 0946-011X in Research Reports MPII, pages 98–109, 1998.

[FLPR99] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. In *40th Symposium on Foundations of Computer Science*, 1999.

[FR91] C. Fricker and P. Robert. An analytical cache model. Technical Report 1496, INRIA, Le Chesnay, 1991.

[Ful72] S. H. Fuller. An optimal drum scheduling algorithm. *IEEE Trans. on Computers*, 21(11):1153, November 1972.

[HP96] J. L. Hennessy and D. A. Patterson. *Computer Architecture a Quantitative Approach*. Morgan Kaufmann, 1996.

[LFM99] R. E. Ladner, J. D. Fix, and A. La Marca. Cache performance analysis of traversals and random access. In *10th Symposium on Discrete Algorithms*, 1999.

[LL97] A. LaMarca and R. E. Ladner. The influence of caches on the performance of sorting. In *8th Symposium on Discrete Algorithm*, pages 370–379, 1997.

[MIP98] MIPS Technologies, Inc. *R10000 Microprocessor User's Manual*, 2.0 edition, 1998. `http://www.mips.com`.

[MV84] K. Mehlhorn and U. Vishkin. Granularity of memory in parallel computation. *Acta Informatica*, 21:339–374, 1984.

[NBC+94] C. Nyberg, T. Barclay, Z. Cvetanovic, J. Gray, and D. Lomet. Alphasort: A RISC machine sort. In *SIGMOD*, pages 233–242, 1994.

[Neu98] K. S. Neubert. The flashsort1 algorithm. *Dr. Dobb's Journal*, pages 123–125, February 1998.

[Ran91] A.G. Ranade. How to emulate shared memory. *Journal of Computer and System Sciences*, 42(3):307–326, june 1991.

[Rao78] G. Rao. Performance analysis of cache memories. *Journal of the ACM*, 25(3):378–395, 1978.

[RR99] N. Rahman and R. Raman. Analysing cache effects in distributed sorting. In *3rd Workshop on Algorithm Engineering*, LNCS, 1999.

[San98] P. Sanders. Random permutations on distributed, external and hierarchical memory. *Information Processing Letters*, 67(6):305–310, 1998.

[San99a] P. Sanders. Accessing multiple sequences through set associative caches. In *ICALP*, number 1644 in LNCS, pages 655–664, 1999.

[San99b] P. Sanders. Fast priority queues for cached memory. In *ALENEX '99, Workshop on Algorithm Engineering and Experimentation*, number 1619 in LNCS, pages 312–327. Springer, 1999.

[SC00] S. Sen and S. Chatterjee. Towards a theory of cache-efficient algorithms. *SODA*, pages 829–838, 2000.

[SF73] H. S. Stone and S. F. Fuller. On the near-optimality of the shortest-access-time-first drum scheduling discipline. *Communications of the ACM*, 16(6), June 1973. Also published in/as: Technical Note No.12, DSL.

[Sib97] J. Sibeyn. From parallel to external list ranking. Technical Report MPI-I-97-1-021, Max-Planck Institut für Informatik, 1997.

[SK97] J. Sibeyn and M. Kaufmann. BSP-like external-memory computation. In *3rd Italian Conference on Algorithms and Complexity*, pages 229–240, 1997.

[SV01] P. Sanders and B. Vöcking. Random arc allocation and applications to disks, drums and drams. submitted for publication, 2001.

[Vit98] J. S. Vitter. External memory algorithms. In *6th European Symposium on Algorithms*, number 1461 in LNCS, pages 1–25. Springer, 1998.