Accessing Multiple Sequences Through Set Associative Caches

Peter Sanders

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

Abstract. The cache hierarchy prevalent in todays high performance processors has to be taken into account in order to design algorithms which perform well in practice. We start from the empirical observation that external memory algorithms often turn out to be good algorithms for cached memory. This is not self evident since caches have a fixed and quite restrictive algorithm choosing the content of the cache. We investigate the impact of this restriction for the frequently occurring case of access to multiple sequences. We show that any access pattern to $k = \Theta(M/B^{1+1/a})$ sequential data streams can be efficiently supported on an a-way set associative cache with capacity M and line size B. The bounds are tight up to lower order terms.

Keywords: Set associative cache, external memory algorithm, memory hierarchy, multi merge.

1 Introduction

The mainstream model of computation used by algorithm designers in the last half century [13] assumes a single processor with unit memory access cost. However, the mainstream computers sitting on our desktops have increasingly deviated from this model in the last decade [7–9, 12, 19]. Even without taking disks and tapes into account the memory hierarchy usually has four levels: registers, first-level cache, second-level cache and main memory. We concentrate on the relation between one cache level and the main memory since registers can only be used in a restricted way and multiple cache levels would complicate the analysis. Alos, in many applications, the traffic between two levels forms the main bottleneck. Including all overheads for cache miss, memory latency and translation from logical over virtual to physical memory addresses, a main memory access can be two orders of magnitude slower than a first-level cache hit while the main memory is three to five orders of magnitude larger. Most machines have separate caches for data and code so that we can disregard instruction reads as long as the programs remain reasonably short. A cache of size M can store M/B *cache lines* of size B (we use the size of the data elements of the underlying application as unit). All accesses to the next lower level of memory are done in units of cache lines. An *a-way set associative cache* consists of $M/(aB)$ *cache sets* each of which can store a cache lines. A cache line starting at memory address xB can only be stored in set number $x \bmod M/(aB)$. Caches usually use the following fixed replacement strategy: On a cache miss, the least recently used (LRU) line in the set of x is replaced by the new line. In order to get fast, compact caches with acceptable power dissipation, a is a small constant between one (*direct mapped cache*) and eight.

Although the technological details are likely to change in the future, physical principles imply that fast memories must be small and are likely to be more expensive than slower memories so that we will have to live with memory hierarchies when talking about sequential algorithms for large inputs.

The general approach of this paper is to model one cache level and the main memory by the single disk single processor variant of the external memory model by Vitter and Shriver [22] where M is the size of the internal memory, B is the block transfer size, i.e., we use the word pairs "cache line" and "memory block", "cache" and "internal memory", "main memory" and "external memory" and "I/O" and "cache fault" as synonyms if the context does not indicate otherwise. We include set associative caches into this model by disallowing explicit access to the internal memory, i.e., the cache replacement strategy decides which external memory references can be satisfied from the internal memory. We call this model *cached memory*.

An almost ubiquitous principle behind efficient external memory algorithms is to read or write $k = \mathcal{O}(M/B)$ sequential streams of data [21]. For example, k-way merge sort is based on reading and radix sort, buffer trees [1] or external memory list ranking [18] are based on writing k sequences. Empirically, many of these algorithms also perform well on cached memory. For example, in a study by LaMarca and Ladner [11], k-way merging performs best among algorithms tried and even Sibeyn's quite involved external memory list ranking algorithm [18] performs better than a simple pointer chasing although the latter executes only a fraction of the instructions. We have designed an external memory priority queue based on k-way merging which performs $\mathcal{O}((I/B)\log_{M/B} I/M)$ I/Os for any sequence of operations with I insertions [17]. This algorithm is similar to previous algorithms with the same asymptotic performance [1, 3, 5, 4] yet performs at least a factor of three fewer I/Os. Running in the cache hierarchy of a workstation the algorithm is several times faster than an optimized binary heap implementation which is empirically the best algorithm for small queues [17]. Similarly, an external memory version of a simple parallel algorithm for generating random permutations turns out to be several times faster on a cached memory than the conventional sequential algorithm which executes only half as many instructions [16]. This algorithm is based on writing k sequences to memory.

Unfortunately, most of these algorithms can fail miserably on set associative caches because an adversary can schedule the accesses in such a way that all recent accesses use the same cache set. In Section 2 we show that it is sufficient to randomize the starting addresses of the data streams and to reduce k by a factor $\mathcal{O}(B^{1/a})$ in order to ensure that the expected number of cache misses after N accesses is only a small fraction larger than the N/B first reference misses which have to occur when streaming through N elements. We give upper and lower bounds on the expected number of cache faults which are tight up to lower order terms for the range of inputs which allow efficient operations.

Related Work

Caches are intensively studied in computer architecture and compiler design (e.g. [7]). Evaluations are usually based on simulations. This yields useful quantitative results if traces of meaningful benchmarks are simulated. Simulations also have the advantage that interactions between many complicated features for mitigating cache faults like victim caches, write buffers or out-of-order execution can be modeled. However, each simulation only yields results for one particular combination of architecture, algorithm, implementation, compiler and input. 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 before starting to implement. Simulations are unacceptable for theoreticians who would like to quantify the relative power of different machine models. Furthermore, the external memory algorithms we have in mind, produce so many irregular memory references and data dependencies that the additional architectural optimizations mentioned above cannot completely hide the general structure of the cache defined by the parameters M, B and a.

Simple analytical cache models have long been known [15, 10]. However, in these *independent reference models* the cache lines are assumed to be accessed in random order according to some fixed probability distribution. This assumption is not warranted for accessing sequences and we will see that it can lead to wrong predictions about the impact of the associativity a.

Fricker and Robert [6] have proposed a model for accessing sequences. However, it is limited to one particular access schedule while we allow an adversary to schedule the accesses. Furthermore, their model can only be evaluated numerically for a particular set of parameters and needs at least quadratic time in the number of memory accesses whereas our analysis yields closed form formulas.

On the application side, cache optimizations play an important role in high performance numerical computations but the data access patterns occurring there are often quite regular or at least predictable. Therefore, the basic techniques used in numerical computations are of little help in optimizing irregular and unpredictable access patterns.

External memory algorithms are a well established branch of algorithmics [21, 20]. Our approach to randomize the starting addresses of sequences is similar to the approach used by Barve et al. [2] in order to efficiently use parallel disks for k-way merging. However, we do not want to bound the maximum contention but the fraction of overloaded cache sets. Furthermore, for k-way merging, a clever prefetching algorithm is available whereas we have to live with a fixed replacement strategy. Correspondingly, the analysis techniques are different.

Overview

After defining the problem in Section 2 we derive an upper bound on the number of cache misses due to sequence accesses in Section 3. In Section 4 this bound is refined to take interferences between sequence data and work areas with arbitrary access patterns into account. Section 5 complements this with a lower bound. Finally, Section 5 summarizes the results and compares the bounds numerically for a particular example.

2 Multiple Data Streams

Consider k sequences stored in arrays. These elements are read (or written) sequentially. An adversary is allowed to schedule the accesses to these sequences, i.e, it is allowed to choose N and $s_1, \dots, s_N \in \{1, \dots, k\}$ 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

The analysis is done for starting addresses x_j of the arrays with the property that the values $x_j \bmod M$ are uniformly distributed independent random variables. If the actual code does not use randomization, the analysis will yield average case bounds. The code can also actively randomize the starting addresses. For example, when allocating memory for a sequence of length l, the algorithm can choose a random offset $0 \leq X < M$, allocate a memory block of length $l + X$ and put 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. If the lengths of all sequences are known in advance (e.g., for k-way merge sort), it should be possible to waste even less memory since we additionally have the choice in which order to allocate the sequences. This may be important for large k and large second level caches in order to avoid running out of virtual address space.

3 An Upper Bound

We start with the simplifying assumption that all memory accesses are sequence accesses. In Section 4, we will see that this is often a good approximation.

Theorem 1. *Given an a-way set associative cache with capacity M and cache line size $B < M/a$. Any schedule of N sequential accesses to $k < M\alpha(a)/B$ sequences with randomized starting address*[1] *causes at most*

$$\mathbf{E}[X_a] \leq N\left(\frac{1}{B} + \left(\frac{kB}{M\alpha(a)}\right)^a + \mathcal{O}\left(\frac{k}{M\alpha(a) - kB}\right)\right) + k \qquad (1)$$

cache misses, where $\alpha(a) = \sqrt[a]{a!}/a$.

[1] Closer inspection of the proof shows that it is sufficient if the starting addresses are $(a+1)$-wise independent.

We analyze the different types of cache misses seperately. Whenever a cache line is accessed for the first time we have a *first reference miss* (also called *compulsory miss*).

Lemma 1. *There are at most N/B first reference cache misses.* ∎

Conflict misses arise when more than a cache lines are mapped to the same cache set. We first look at the case that frequent accesses to one sequence fill up multiple entries of a cache set:

Lemma 2. *There are at most $N(k-1)/(M-B+1) \approx Nk/M$ conflict misses with one sequence occupying at least two entries of a cache set.*

Proof. Consider the accesses for a particular sequence b. Let N_b denote the length of sequence b. An access to b can only cause a conflict miss with a sequence b' occupying at least two entries of the accessed cache set, if b' has made at least $M-B+1$ accesses after the last access to b (due to LRU replacement). Since there are at most $N-N_b$ accesses by other sequences overall, sequence b can suffer at most $\lfloor (N-N_b)/(M-B+1) \rfloor$ of the conflict misses under consideration. Summing over all sequences yields the claimed bound. ∎

The most interesting cache misses are those conflict misses where different sequences access the same cache set.

Lemma 3. *Any schedule of N sequential accesses to k sequences with randomized starting address causes at most*

$$\mathbf{E}[X_a] \leq N\frac{B-1}{B}\left(\left(\frac{(k-1)(B-1)}{M\alpha(a)}\right)^a + \frac{k-1}{M\alpha(a)-(k-1)(B-1)}\right) + k \quad (2)$$

cache misses due to conflicts between different sequences, where $\alpha(a) = \sqrt[a]{a!}/a$.

Proof. Let c_{bj} denote the probability that the cache set addressed by sequence b in its j-th access has been accessed by at least a other sequences since the last access of sequence b. By linearity of expectation the expected number of this type of conflict misses can be bounded by

$$\mathbf{E}[X_a] \leq k + \frac{B-1}{B}\sum_b \sum_{j=1}^{N_b} c_{bj}$$

where N_b denotes the total number of accesses to sequence b. The factor $(B-1)/B$ stems from the fact that the first access to each cache line cannot cause a conflict miss since it causes a first reference miss.

Now we focus on a particular sequence (we therefore drop the b indices for now). Let

$$\bar{B} := \{1,\ldots,k\} \setminus \{b\} \quad .$$

Let z_{ij} denote the number of accesses made to sequence $i \in \bar{B}$ between the $(j-1)$-th and j-th access by sequence b and let

$$z_j := \sum_{i\in\bar{B}} z_{ij}$$

denote the total number of possibly conflicting accesses. The probability that sequence i uses the same cache set between these two accesses is

$$p_{ij} = \min\left(1, (z_{ij} + B - 1)\frac{a}{M}\right)$$

if $z_{ij} > 0$ and 0 otherwise. We have

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

since the sequences are shifted independently. We now relax the integrality requirement on z_{ij} and also ignore that p_{ij} is truncated to zero for $z_{ij} = 0$ and solve a constrained maximization problem for the function $\sum_j p_j$ where

$$p_j := \min\Bigl(1, \sum_{I \subseteq \bar{B}, |I|=a} \prod_{i \in I} \frac{a}{M}(z_{ij} + B - 1)\Bigr) \ .$$

First, observe that for fixed z_j, p_j is maximized by choosing all coefficients identical, i.e.,

$$p_j \leq \min\left(1, \binom{k-1}{a}\left(\left(\frac{z_j}{k-1} + B - 1\right)\frac{a}{M}\right)^a\right) \ .$$

A global maximum for this bound is achieved by setting as many of the z_j as possible to a value just large enough to achieve an estimate $p_j = 1$.

Since $\binom{k-1}{a} \leq \frac{(k-1)^a}{a!}$ we get

$$p_j \leq \frac{(k-1)^a}{a!}\left(\frac{\frac{z_j}{k-1} + B - 1}{M}\right)^a = \left(\frac{z_j + (k-1)(B-1)}{M\alpha(a)}\right)^a$$

so $z_j \geq M\alpha(a) - (k-1)(B-1)$ is needed for estimating $p_j = 1$. There are at most $(N - N_b)/(M\alpha(a) - (k-1)(B-1))$ of these terms each contributing 1 to $\sum_j c_{bj}$. The remaining z_j have to be set to 0 now so that we get a contribution of at most

$$N_b\binom{k-1}{a}\left(\frac{(B-1)a}{M}\right)^a \leq N_b\left(\frac{(k-1)(B-1)}{M\alpha(a)}\right)^a$$

for these small terms of $\sum_j p_j$.[2] Summing over all sequences b we get

$$\sum_{b=1}^{k}\sum_{j=1}^{N_b} c_{bj} \leq N\left(\frac{k-1}{M\alpha(a) - (k-1)(B-1)} + \left(\frac{(k-1)(B-1)}{M\alpha(a)}\right)^a\right) \ .$$

■

Theorem 1 is an immediate consequence of lemmata 1, 2 and 3.

[2] Note that this way of estimation is only for technical convenience and gives no hint on actual worst case access schedules: The zero z_j only have a contribution because we ignore the truncation of p_{ij} to zero and the nonzero z_j are counted twice.

4 A Notion of Working Set

Theorem 1 accounts for the most important source of conflict misses. However, in practice our application will have additional frequently accessed data which comes into conflict with the sequence data. In particular, we usually need k sequence pointers and additional data structures of size $\mathcal{O}(k)$ for deciding which sequence is accessed next. We formalize this concept by defining a *working set* of size w to be data touching w/B cache lines such that no two of them are mapped to the same cache set. In particular, this is the case if the working set consists of w words of contiguous aligned memory.

Lemma 4. *Let Y_a denote the number of conflict misses predicted by lemmata 2 and 3 for an a-way associative cache. With a working set of size $w \leq M/a$, the expected number of conflict misses involving both the working set and the k sequences can be bounded by $2\mathbf{E}[Y_{a-1}]wa/M$.*

Proof. An access to stream b conflicts with the working set with probability wa/M. This can lead to a conflict miss if before the last access of stream b, one sequence has made at least $M - B + 1$ accesses or if $a - 1$ other cache lines have accessed this cache set. The number of these events can be bounded using lemmata 2 and 3 respectively. The factor two is a conservative estimate to account for the cases when working set data is evicted from the cache and has to be reloaded later. ∎

Substituting the previously derived bounds we can conclude that Theorem 1 extends to the case where the working set is taken into account.

Corollary 1. *Given an a-way set associative cache with capacity M and cache line size $B < M/a$. For any schedule of N sequential accesses to k sequences with randomized starting address and any number of accesses to a working set of size $\mathcal{O}(k) \leq M/a$, Relation (1) bounds the number of cache faults.*

Only for a working set of size $\Omega(Bk)$, conflict misses involving the working set begin to dominate.

5 A Lower Bound

The bound from Theorem 1 and Corollary 1 on the expected number of cache misses is asymptotically tight. Indeed for the most interesting case of $k = o(M/B)$, when conflict misses are rare, the bound is tight up to lower order terms:

Theorem 2. *There are access schedules to $k < M/(aB)$ sequences such that any strategy to choose the starting addresses of the sequences incurs*

$$\mathbf{E}[X_a] \geq N\left(\frac{1}{B} + \frac{B-1}{B}\left(\frac{(k-a)B}{M\alpha(a)}\right)^a \max\left\{\left(1 - \frac{kaB}{M}\right), \frac{1}{e}\right\}\right) - \mathcal{O}(kM) \tag{3}$$

cache faults on the average, where $\alpha(a) = \sqrt[a]{a!}/a$.

Proof. Suppose that the adversary first "randomly forwinds" the sequences, i.e., it accesses the first X_i elements of sequence i where the X_i are random variables choosen uniformly and independently from $\{1, M\}$.

Even disregarding the conflict misses during forwinding, we have

$$\mathbf{E}[X_a] \geq N/B + P_{\mathrm{miss}}(N - kM)(B-1)/B \geq N/B + P_{\mathrm{miss}}N(B-1)/B - kM$$

where N/B is the number of first reference misses and where P_{miss} is the probability of a conflict miss after forwinding when one of the last $B-1$ elements of a cache line are accessed.

After forwinding, 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. If the adversary subsequently accesses the sequences in a round robin fashion then an access to a sequence implies a cache miss if the $k-1$ previous accesses to the other sequences have accessed the same cache set at least a times. The probability for this event is

$$P_{\mathrm{miss}} := \sum_{i \geq a} \binom{k-1}{i} \left(\frac{aB}{M}\right)^i \left(1 - \frac{aB}{M}\right)^{k-i-1} \tag{4}$$

For the purpose of numerical evaluation this is already all we need since the above tail of a binomial distribution can be efficiently evaluated using a continued fraction development of the incomplete beta function [14, Section 6.4]. For an easy to interpret closed form formula it suffices to work with the first summand:

$$\begin{aligned}
P_{\mathrm{miss}} &> \binom{k-1}{a} \left(\frac{aB}{M}\right)^a \left(1 - \frac{aB}{M}\right)^{k-a-1} \\
&> \frac{(k-a)^a}{a!} \left(\frac{aB}{M}\right)^a \left(1 - \frac{aB}{M}\right)^k \qquad = \left(\frac{(k-a)B}{M\alpha(a)}\right)^a \left(1 - \frac{aB}{M}\right)^k \\
&\geq \left(\frac{(k-a)B}{M\alpha(a)}\right)^a \max\left\{\left(1 - \frac{kaB}{M}\right), \frac{1}{e}\right\}
\end{aligned}$$

The latter estimation uses the relations $(1 - aB/M)^k \geq 1/e$ for $k < M/(aB)$ and $(1 - aB/M)^k \geq (1 - kaB/M)$. ■

6 Discussion

If we set the number of sequences k to $cM/B^{1+1/a}$ for some constant c, we can see from Theorem 1 and Corollary 1 that the probability of a conflict miss is $c^a a^a/(Ba!)+$"lower order terms", i.e., it is of the same order as the fraction of first reference misses. For larger k we get a considerable performance degradation.

Perhaps the most interesting qualitative conclusion is that for large cache lines, the associativity can have a remarkably high impact on cache efficiency. This stands in contrast to the results one would get by blindly applying the independent refenrence model, namely that associativity is of little help. The independent reference model must fail because k long sequences access all cache lines about equally frequently but nevertheless exhibit significant locality if k is not too large.

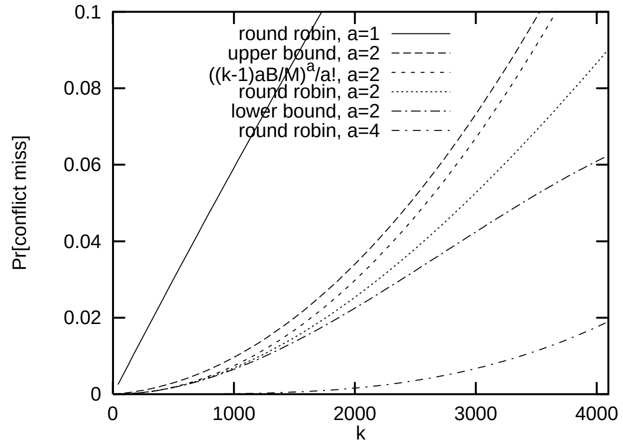

Fig. 1. Conflict miss probabilities for $B = 64$, $M = 2^{20}$. Exact values are computed for the difficult random round robin schedule from Section 5. For $a = 2$ the worst case upper bound from Theorem 1, a simple approximation and the simple lower bound from Corollary 1 are also shown.

We can also draw quantitative conclusions, since for $k = \mathcal{O}(M/B^{1+1/a})$ the derived bounds are tight up to lower order terms. At least for large B this also works out in practice. For example, consider the conflict miss probabilities shown in Fig. 1. The parameters used there represent a possible configuration of the 2-way associative second level cache of the MIPS R10000 processor [12]. Namely, 4 Mbytes divided into cache lines of 256 bytes each. We assume a unit of 4 bytes so that $M = 2^{20}$ and $B = 64$. In this situation, a conflict miss probability of $1/B \approx 1.5\%$ should not be exceeded to match the number of first reference misses. For the range k around 1000 fulfilling this condition, both upper and lower bounds are quite accurate. Even for larger k they clearly separate the behavior of the 2-way associative cache from direct mapped and 4-way associative alternatives. While a direct mapped cache can only support a few hundred sequences efficiently, a 4-way associative version still works quite well for $k = 4000$. Still, a fully associative cache could support 2^{14} sequences without conflict misses.

References

1. 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.
2. R. D. Barve, E. F. Grove, and J. S. Vitter. Simple randomized mergesort on parallel disks. *Parallel Computing*, 23(4):601–631, 1997.
3. Gerth Stølting Brodal and Jyrki Katajainen. Worst-case efficient external-memory priority queues. In *6th Scandinavian Workshop on Algorithm Theory*, number 1432 in LNCS, pages 107–118. Springer Verlag, Berlin, 1998.
4. A. Crauser, P. Ferragina, and U. Meyer. Efficient priority queues in external memory. working paper, October 1997.
5. R. Fadel, K. V. Jakobsen, J. Katajainen, and J. Teuhola. External heaps combined with effective buffering. In *4th Australasian Theory Symposium*, volume 19-2 of *Australian Computer Science Communications*, pages 72–78. Springer, 1997.
6. C. Fricker and P. Robert. An analytical cache model. Technical Report 1496, INRIA, Le Chesnay, 1991.
7. J. L. Hennessy and D. A. Patterson. *Computer Architecture a Quantitative Approach*. Morgan Kaufmann, 1996.
8. Intel Corporation, P.O. Box 5937, Denver, CO, 80217-9808, `http://www.intel.com`. *Intel Archtecture Software Developer's Manual. Volume I: Basic Architecture*, 1997. Ordering Number 243190.
9. J. Keller. The 21264: A superscalar alpha processor with out-of-order execution. In *Microprocessor Forum*, October 1996.
10. A. LaMarca and R. E. Ladner. The influence of caches on the performance of heaps. *ACM Journal of Experimental Algorithmics*, 1(4), 1996.
11. A. LaMarca and R. E. Ladner. The influence of caches on the performance of sorting. In *8th ACM-SIAM Symposium on Discrete Algorithm*, pages 370–379, 1997.
12. MIPS Technologies, Inc. *R10000 Microprocessor User's Manual*, 2.0 edition, 1998. `http://www.mips.com`.
13. J. von Neumann. First draft of a report on the EDVAC. Technical report, University of Pennsylvania, 1945.
14. W. H. Press, S.A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2nd edition, 1992.
15. G. Rao. Performance analysis of cache memories. *Journal of the ACM*, 25(3):378–395, 1978.
16. P. Sanders. Random permutations on distributed, external and hierarchical memory. *Information Processing Letters*, 67(6):305–310, 1998.
17. Peter Sanders. Fast priority queues for cached memory. In *ALENEX '99, Workshop on Algorithm Engineering and Experimentation*, LNCS. Springer, 1999.
18. J. Sibeyn. From parallel to external list ranking. Technical Report MPI-I-97-1-021, Max-Planck Institut für Informatik, 1997.
19. Sun Microsystems. *UltraSPARC-IIi User's Manual*, 1997.
20. D. E. Vengroff. *TPIE User Manual and Reference*, 1995. `http://www.cs.duke.edu/~dev/tpie_home_page.html`.
21. J. S. Vitter. External memory algorithms. In *6th European Symposium on Algorithms*, number 1461 in LNCS, pages 1–25. Springer, 1998.
22. J. S. Vitter and E. A. M. Shriver. Algorithms for parallel memory I: Two level memories. *Algorithmica*, 12(2–3):110–147, 1994.