Competitive Cache Replacement Strategies for Shared Cache Environments

Anil Kumar Katti
Department of Computer Science
University of Texas at Austin
Austin, TX 78712, USA
Email: akatti@cs.utexas.edu

Vijaya Ramachandran
Department of Computer Science
University of Texas at Austin
Austin, TX 78712, USA
Email: vlr@cs.utexas.edu

***Abstract*—We investigate cache replacement algorithms (CRAs) at a cache shared by several processes under different multicore environments. For a single shared cache, our main result is the first CRA,** GLOBAL-MAXIMA**, for fixed interleaving under shared full knowledge [1], where any data can be accessed by any process, and each process has full knowledge about its future request sequence. We establish that** GLOBAL-MAXIMA **has competitive ratio within a constant factor of optimal. This answers the major open question in [1]. We also present** RR-PROC-MARK**, a CRA for the disjoint full knowledge case, which is very simple and efficient, and achieves a better competitive ratio than the algorithms in [2], [1]; it is in fact optimal except when the number of processes sharing the cache is small.**

We then consider a cache hierarchy, both for a single process and when shared by several processes. We present CRAs for three types of caching models commonly used at a higher-level cache: inclusive, exclusive, and partially-inclusive, and we establish that several of our CRAs have optimal competitive ratio. Our results for a cache hierarchy are new even in the traditional no knowledge case and even for a single process.

***Keywords*-caching; cache replacement; shared cache; cache hierarchy; multicore; interleaved request sequence; full knowledge; inclusion property;**

I. INTRODUCTION

Caching and paging are among the earliest problems addressed through competitive analysis [3], and they have been widely studied in this context since then. In this paper we study cache replacement algorithms (CRAs) under competitive analysis in a parallel setting when several processes (or cores) share a cache, or a cache hierarchy. This type of caching environment is widely used in multicores.

In the parallel setting, an optimal offline strategy (OPT) can have an advantage over an online strategy through two mechanisms: knowledge of the future (as in the sequential setting) and knowledge of the interleaving of cache requests from the different cores. Two types of interleaving of requests have been studied in the literature. Barve et al. [1] and Cao et al. [2] considered *fixed interleaving* of requests from the individual request sequences at the shared cache. The main assumption in this case is that the interleaving of requests reaching the shared cache is same for all CRAs, and adversarial in nature for the competitive analysis. Hassidim [6] and Lopez-Ortiz and Salinger [7] considered a different type of interleaving, which we call *free interleaving* at the shared cache. Here, the interleaving of requests reaching the shared cache is assumed to depend on the cache replacement decisions taken by the CRA, and different CRAs could see different interleaving of requests at the shared cache.

This work was supported in part by NSF grant CCF-0830737.

Orthogonal to fixed and free interleaving, we consider CRAs in two 'knowledge' models — *no knowledge* (NK) and *full knowledge* (FK). The NK model is the standard classical caching model, where processes are assumed to have no knowledge about future requests in the online setting. In the FK model [1], [2], processes have full knowledge about their individual request sequences in the online case, and the optimal offline algorithm (OPT) has full knowledge about both the future request sequences and the interleaving of these request sequences.

We now summarize the known results for these models.

No knowledge (NK) model: This is the standard caching model, and is well understood in both fixed and free interleaving.

- **Fixed interleaving:** Caching at a single shared cache in the NK model reduces to the sequential caching model [1], with LRU being an optimal online deterministic CRA [3] and PARTITION being an optimal online randomized CRA [4], [5].
- **Free interleaving:** Two variants of this model were considered in [6], [7]. In the model in [6], the CRA could determine the schedule of requests reaching the shared cache, and it was shown that even with resource augmentation (where the online CRA has a bigger cache than OPT), many of the well known CRAs such as LRU and FIFO have competitive ratio $\Omega(b/p)$ (in terms of makespan), where b is the ratio of cache miss time to cache hit time and p is the number of processes. Since any CRA can achieve a competitive ratio of b even if it misses on every request, these CRAs perform poorly when the number of processes is a constant. The model considered in [7] forces the CRA at a shared cache to serve requests as soon as they arrive. Using a slight modification of the lower bound sequence used

Table I

Overview of results for CRAs in no knowledge and full knowledge models					
Model	**Interleaving**	**Single Shared Cache**		**Shared Cache Hierarchy**	
		$p = 1$	$p > 1$	$p = 1$	$p > 1$
No Knowledge	**Fixed**	[3], [4], [5]	[3], [4], [5]	**This paper (Table III*)**	
	Free		[6], [7]	**Extension from [6] (Section III-C)**	
Full Knowledge	**Fixed**	**Online same as OPT**	[1], [2], **This paper (Table II*)**	**Online same as OPT**	**This paper (Table II*)**
	Free		**Online same as OPT (Section II-D)**		**Online same as OPT (Section II-D)**
*Refer to the table for more details.					

in [6], it can be seen that the competitive ratio of online algorithms is $\Omega(b/p)$ (in terms of makespan) even in this model. Also, [7] studies the performance of CRAs under different partitioning strategies for the cache.

Full knowledge (FK) model: This model was first considered in the application-controlled caching context by [2] and [1]. We discuss the practical relevance of this model in the multicore caching context later in this section.

- **Fixed interleaving:** This was studied in [1], [2]. In this setting, each individual process has full knowledge of its future request sequence (the FK model), and the power of the offline algorithm arises from its knowledge of the interleaving, which is fixed for all the CRAs and assumed to be adversarial for the sake of competitive analysis. We further classify the FK model into the disjoint memory full knowledge (DFK) model studied in [1], where processes request disjoint sets of memory blocks, and the shared memory full knowledge (SFK) model, left as an open problem in [1], where different processes may access the same memory block. Our main results are for the SFK model. We also present a new and improved CRA in the DFK model.
- **Free interleaving:** The FK model has not been considered before under free interleaving, and we study it in this paper. We use the free interleaving model in [6] mentioned earlier since the lower bounds in that paper can be readily adapted to the model in [7] by slightly changing the lower bound sequences; further both the lower bound sequences and the analysis are simpler with the former model. Online CRAs are as powerful as offline CRAs since both of them have full knowledge about the individual request sequences and can control the interleaving of requests reaching the shared cache by controlling the schedule of each core. Hassidim [6] proved that the optimal offline CRA in this model is NP complete under the assumption that $k = p/3$ (where k is the size of the shared cache and p is the number of cores). We generalize this result to arbitrary k and p.

A. *Relevance to current practice*

The models that we consider in this paper are related to current practice in several ways.

Shared memory full knowledge: In a number of parallel shared memory applications, cores work together to solve a given problem and hence share memory blocks. In many of these computations (e.g., matrix multiplication, FFT, I-GEP [8], [9]), each process has full knowledge about the future request sequence of its current task. Another scenario in which the SFK model relates well to the real world is in modern multicore processor systems equipped with prefetchers [10], [11]. These units predict the future request sequence in an online fashion for most computations. Both of these examples motivate us to consider the SFK model.

Fixed and free interleaving: Under fixed interleaving we assume that the interleaving of requests reaching the shared cache is the same for all CRAs and adversarial in nature for competitive analysis. However, in modern multicore systems, requests from different cores reach the shared cache in parallel. When a cache miss is incurred, only the core with the cache miss gets delayed. All other cores can continue requesting memory blocks. Thus, the interleaving of request sequences as seen at the shared cache depends on the eviction decisions taken by the CRA. This observation was modeled as free interleaving in [6]. On the other hand, even though the interleaving depends on the eviction decisions taken by the CRA , it does not necessarily depend on just the delays due to cache misses. This is because modern operating systems interrupt processes for a number of other reasons (e.g., serving system calls, scheduling, etc.) and the interleaving of requests is significantly affected by these interrupts. Thus, it can be argued that adversarial fixed interleaving is a suitable way to model these interrupts for competitive analysis. This was also stated as the motivation behind fixed and adversarial interleaving in [2].

B. *Related work*

We measure the performance of online CRAs using classical competitive analysis [3]. Let $cost(\text{ALG}, \sigma)$ denote the number of cache misses incurred by a CRA ALG on

Table II

Competitive ratio of CRAs at a single shared cache under (k,k)-paging (our results are in bold font and in boxes) (Section II)						
Model	**Deterministic Algorithms**			**Randomized Algorithms**		
	Lower Bound	**Upper Bound**	**CRA**	**Lower Bound**	**Upper Bound**	**CRA**
NK	k [3]	k [3]	LRU	H_k [5]	H_k [4]	PARTITION
DFK	$p+1$ [1]	$2(p+1)$ [1]	[2]	H_{p-1} [1]	$2(H_{p-1}+1)$ [1]	[1]
		$\boxed{\mathbf{\max(10, p+1)}}$ **[Thm II.14]**	**RR-PROC-MARK**			
SFK	$\boxed{\mathbf{\frac{p}{2}\log\frac{4}{3}\frac{k+1}{p}}}$ **[Thm II.1]**	$\boxed{\mathbf{2(p\ln(ek/p)+1)}}$ **[Thm II.3]**	**GLOBAL-MAXIMA**	$\boxed{\mathbf{\frac{1}{2}\log(k+1)}}$ **[Thm II.2]**	H_k [4]	PARTITION

Table III

Multi level shared cache hierarchy with l levels of same-type cache (all results are new) (Section III)

Table A: Equivalent cache size for the top level cache L_l

CRA	**INCL**	**EXCL**	**PRT-INCL**
ONLINE $\boldsymbol{size = k}$	$\boldsymbol{k_l}$	$\boldsymbol{\sum_{i=1}^{l} k_i}$	$\boldsymbol{\sum_{i=1}^{l} k_i - l + 1}$
OPT $\boldsymbol{size = h}$	$\boldsymbol{\sum_{i=1}^{l} h_i}$		

k_i and h_i are size of L_i for ONLINE and OPT resp., $1 \le i \le l$.

Table B: Competitive ratio of CRAs at top level cache L_l

Model	$h = c \cdot k$ & $c < 1^*$ (Table IIIA has the h and k values)	
	Lower Bound	**Upper Bound**
NK	$\boldsymbol{\frac{k}{k-h+1}}$	$\boldsymbol{\frac{k}{k-h+1}}$
DFK SFK	$\boldsymbol{\Omega(1)}$	$\boldsymbol{\mathcal{O}(1)}$**-optimal**

*See Table II for $c = 1$.

a sequence σ. Recall that the competitive ratio of ALG is bounded by $\mathcal{R}(\text{ALG})$ if there exists a constant c' such that for all sequences σ, $cost(\text{ALG}, \sigma) \le \mathcal{R}(\text{ALG}) \cdot cost(\text{OPT}, \sigma) + c'$. ALG is c-optimal if $\mathcal{R}(\text{ALG})$ is at most c times the smallest competitive ratio that any online algorithm can achieve.

The competitive ratio is measured in terms of makespan in [6] since makespan is a more suitable cost measure in the free interleaving model.

Caching at a single shared cache in the DFK model was formalized as 'application-controlled caching' in [2], who proposed a CRA which picks the process that owns the least recently used (LRU) page and asks it to make a suitable eviction. An upper bound of $2(p+1)$ on the competitive ratio of this algorithm was established by [1], who also gave a lower bound of $p+1$ on the competitive ratio of deterministic CRAs in this model. In the randomized setting, [1] presented a simple randomized algorithm with competitive ratio $2(H_{p-1}+1)$ in the DFK model, and established a lower bound of H_{p-1} on the competitive ratio. The case when processes share memory blocks (SFK) was left as an open problem in [1]. We are not aware of any prior work on CRAs for a cache hierarchy.

C. *Our results*

Our main results are for fixed interleaving, and we have two classes of results, one for a single shared cache, and the other for a hierarchy of shared caches.

1. **Single Shared Cache** (Section II.) For the SFK model, we give a lower bound of $\frac{p}{2}\log\frac{4(k+1)}{3p}$ on the competitive ratio, and we present GLOBAL-MAXIMA, a deterministic CRA which is $O(1)$-optimal in this model. We also give a lower bound on the competitive ratio of randomized algorithms in the SFK model, which establishes that PARTITION is $O(1)$-optimal for this model. These are the first nontrivial results for the SFK model. In the DFK model, we introduce the notion of a *process marking algorithms*, and we present RR-PROC-MARK, a deterministic algorithm with a competitive ratio of at most $\max(10, p+1)$. Along with being 1-optimal for $p \ge 9$, RR-PROC-MARK is also computationally more efficient than the algorithm in [2], [1], and it is a fair algorithm in the sense that every process is asked to make almost the same number of evictions. It also has a good competitive ratio under the 'full access cost model'. Table II gives the known and new results for a single shared cache.

2. **Shared Cache Hierarchy** (Section III.) We formalize the definitions of inclusive (INCL), exclusive (EXCL) and partially-inclusive (PRT-INCL) caches based on their descriptions in the literature. We obtain upper bounds on the competitive ratio of deterministic CRAs for the top level cache L_l for all three types of caches using the notion of *cache equivalence*, where we establish that each type is

equivalent to a standard single-level cache whose size is a certain function of the sizes of L_i for $1 \leq i \leq l$ (see Table IIIA). For OPT we define L-FITF, which is an extension of FITF, the well-known greedy OPT [12] used for the single-level cache in the sequential setting. These results are new even for a single process in the NK setting.

Road map. In Section II, we investigate CRAs for the FK model under fixed interleaving. We also extend results from [6] for FK model under free interleaving. In Section III, we consider CRAs for a hierarchy of caches. We first define 3 types of caches commonly used in practice – INCL, EXCL and PRT-INCL and compare CRAs using competitive analysis under fixed interleaving.

II. Cache replacement algorithms for a single shared cache

We consider a single cache shared by p processes. In this section we mainly investigate CRAs for the FK model under fixed interleaving. In subsection II-A, we consider the SFK model which was left as an open question in [1] and present deterministic and randomized CRAs which are $O(1)$-optimal. In subsection II-B, we present a new and improved deterministic CRA for the DFK model, which has several features that might make it good choice in practice. In subsection II-D, we consider FK model under free interleaving and generalize the NP-completeness result in [6].

A. *Caching in the SFK model*

In the SFK model, any block can be accessed by any of the p processes. Each process has full knowledge about its individual request sequence. Further, the interleaving of these requests is assumed to be fixed and adversarial for the sake of competitive analysis. In this subsection, we consider CRAs in the (k, k)-paging setting.

For deterministic algorithms, we establish a lower bound of $\frac{p}{2}\log\frac{4(k+1)}{3p}$ on the competitive ratio, and we present GLOBAL-MAXIMA, a CRA which is 11-optimal. For randomized algorithms, we establish a lower bound of $\frac{1}{2}\log(k+1)$ on the competitive ratio; this proves that PARTITION [4] is $\mathcal{O}(1)$-optimal.

GLOBAL-MAXIMA is a *global* algorithm that pools the individual future knowledge of the p processes in order to determine which block to evict. If we confine ourselves to *local* algorithms where future knowledge cannot be shared between processes, we can show that the competitive ratio must be at least k.

*1) **Lower bounds on the competitive ratio in the SFK model**:* To establish our lower bounds we fix the individual request sequences for all of the p processes and then build different adversarial interleaving for the deterministic and randomized lower bounds.

Individual request sequence: We group the p processes into $p/2$ pairs such that the ith pair consists of process P_{2i-1} and P_{2i}. We use $k+1$ distinct memory block requests which are evenly distributed among $p/2$ pairs. Hence, every pair requests $\frac{2(k+1)}{p}$ distinct memory blocks. Process P_{2i-1} requests the blocks requested by P_{2i} in the reverse order. The blocks requested by P_j and P_{j+1} where $j = 2i-1$:
$P_j : (j-1)\frac{k+1}{p}+1, (j-1)\frac{k+1}{p}+2, \cdots, (j+1)\frac{k+1}{p}$.
$P_{j+1} : (j+1)\frac{k+1}{p}, (j+1)\frac{k+1}{p}-1, \cdots, (j-1)\frac{k+1}{p}+1$.
We establish lower bounds on the competitive ratio of *marking* CRAs. We can invoke the notion of *repeated requests to already requested blocks* [13] to generalize these bounds to non-marking CRAs.

Adversarial interleaving: Assume the shared cache of both the online CRA and OPT initially contain blocks $1, 2, \cdots, k$. Then, the interleaving is defined as follows:

1. **Deterministic:** The first phase starts with process P_p requesting $k+1$ (resulting in a cache miss). Let the deterministic marking CRA ALG evict q from the cache. Observe that q can be requested by two processes, say P_{2i-1} and P_{2i}. The adversary makes one of these two processes request q such that minimum number of blocks are marked before q is requested again. This results in another cache miss and the pattern repeats. Theorem II.1 establishes a lower bound of $\frac{p}{2}\log\frac{4(k+1)}{3p}$ on the competitive ratio.

We establish a stronger lower bound (of k) on the competitive ratio of deterministic local CRAs (where future knowledge cannot be shared between processes). The first phase of the interleaving in this case starts with process P_p requesting $k+1$ (resulting in a cache miss). Let the deterministic local marking CRA ALG make an eviction decision based on the local knowledge of a process, say P_{2i-1}. We assume that the unmarked block that is requested farthest in the future request sequence of P_{2i-1}, say q, is chosen for eviction. If not, the request sequence can be modified correspondingly. The adversary makes P_{2i} request the block q. This results in another cache miss and the pattern repeats. Observe that ALG incurs cache miss on every request and OPT on the other hand incurs one cache miss for every k cache misses incurred by ALG by evicting the block that is requested farthest in the interleaving. This results in a lower bound of k on the competitive ratio of deterministic local algorithms.

2. **Randomized:** We use the fact that a lower bound on the expected cost of a deterministic CRA ALG on a probabilistic interleaving of the individual request sequences is also a lower bound on the expected cost of a randomized marking CRA on any deterministic interleaving (von Neumann minimax principle as described by Yao [14]). Each phase consists of a number of stages. Initially all blocks are unmarked. During a given stage, for each pair of processes, one of the 2 processes is picked with equal probability $(1/2)$ and asked to request half of the unmarked blocks in its request sequence. At the end of the stage, all blocks requested during the

current stage are marked. Theorem II.2 proves the claimed lower bound.

The above defined pattern continues till all the blocks in the cache are marked. At that point, the adversary makes all the p processes request the remaining blocks in their request sequences in order to start a new phase.

Theorem II.1. *The competitive ratio of any deterministic CRA under (k,k)-paging in the SFK model is at least $\frac{p}{2}\log\frac{4(k+1)}{3p}$.*

Proof: Let U be the set of unmarked blocks in the request sequences of P_{2i-1} and P_{2i} at the start of the current phase. Let u_j be the number of unmarked blocks left in U just before the jth block in U is evicted by the deterministic CRA ALG. In order to bound the number of cache misses incurred by ALG on the blocks in U before all the blocks in U are marked, we bound u_j in terms of u_{j-1} for an arbitrary j. Let q be the $j-1$st block in U that is evicted by ALG. Let the number of unmarked blocks requested before the first request to q at the moment q was evicted in the request sequence of P_{2i-1} and P_{2i} be $d_{2i-1}(q)$ and $d_{2i}(q)$, respectively. Adversary requests q such that at most $\min(d_{2i-1}(q), d_{2i}(q))+1$ unmarked blocks (including q) in U are marked by the time the request to q is served. After the request to q is served, the adversary does not request any other block belonging to the request sequences of P_{2i-1} and P_{2i} till the jth element in U is evicted. Hence u_j is equal to the number of elements left unmarked in the the request sequences of P_{2i-1} and P_{2i} after the request to q is served. Hence, $u_j = u_{j-1} - \min(d_{2i-1}(q), d_{2i}(q)) + 1$. Observe that the sequence of unmarked blocks in the request sequence of P_{2i-1} always remains an exact reverse of the sequence of unmarked blocks in the request sequence of P_{2i}. Hence at the point q was evicted, for each unmarked block x in $U : \min(d_{2i-1}(x), d_{2i}(x)) \leq \lceil u_{j-1}/2 \rceil - 1$. Which implies, $\min(d_{2i-1}(q), d_{2i}(q)) \leq \lceil u_{j-1}/2 \rceil - 1$. Hence,

$$\begin{aligned} u_j &= u_{j-1} - \min(d_{2i-1}(q), d_{2i}(q)) + 1 \\ &\geq u_{j-1} - \lceil u_{j-1}/2 \rceil \geq \lfloor u_{j-1}/2 \rfloor \end{aligned}$$

Further, observe that $u_1 = 2(k+1)/p$. Using the above equation and the bound on u_1, we can prove by induction on j that $u_j \geq \frac{k+1}{2^{j-2}p} - \sum_{l=0}^{j-2} \frac{1}{2^l}$. The phase ends for the ith pair of processes when there is just one unmarked block left. Solving for j in $\frac{k+1}{2^{j-2}p} - \sum_{l=0}^{j-2} \frac{1}{2^l} \leq 1$, ALG incurs at least $\log\frac{4(k+1)}{3p}$ cache misses on blocks in U before all blocks in U get marked. Since there are $p/2$ such pairs, ALG incurs at least $\frac{p}{2}\log\frac{4(k+1)}{3p}$ cache misses in the entire phase. On the other hand, OPT incurs just 1 cache miss by evicting the block that is requested farthest in the interleaved request sequence when it incurs a cache miss during this phase. Since every phase proceeds in a similar fashion, the lower bound on the competitive ratio of deterministic marking CRAs is $\frac{p}{2}\log\frac{4(k+1)}{3p}$. Using the notion of repeated requests to already requested blocks, we obtain the desired lower bound of $\frac{p}{2}\log\frac{4(k+1)}{3p}$ for all deterministic CRAs. □

Theorem II.2. *The competitive ratio of any randomized CRA in the (k,k)-paging SFK model is at least $\frac{1}{2}\log(k+1)$.*

Proof: In each stage, an unmarked block is requested with probability $1/2$. Since one of these unmarked blocks does not exist in the cache, the expected number of cache misses incurred by a deterministic CRA ALG is $1/2$ during any given stage. Since the number of stages is at least $\log(k+1)$, the expected total number of cache misses incurred by ALG in the entire phase is at least $\frac{1}{2}\log(k+1)$. Recall that OPT incurs exactly one cache miss on any deterministic interleaving consisting of $k+1$ distinct blocks. Hence, the competitive ratio of any randomized CRA is at least $\frac{1}{2}\log(k+1)$ in the SFK model. □

*2) **Deterministic global cache replacement algorithm: GLOBAL-MAXIMA**:* We present a deterministic global marking CRA called GLOBAL-MAXIMA which makes an eviction based on the *global distance* of unmarked blocks in the cache, defined as follows.

Local and global distance functions: Let U be the set of all unmarked blocks in the cache at any point in time. The local distance of an unmarked block x with respect to the process P_i is given by $d_i(x) =$ number of distinct blocks belonging to U in the request sequence of process P_i before the first request to x. If x never occurs in the request sequence of process P_i, $d_i(x) = |U| - 1$. The global distance of an unmarked block x is given by $d(x) = \min_{i=1}^{p} d_i(x)$.

GLOBAL-MAXIMA. GLOBAL-MAXIMA is an SFK marking algorithm which, upon a cache miss, evicts an unmarked block q with the maximum global distance from the cache. By evicting a block with the maximum global distance, GLOBAL-MAXIMA ensures that maximum number of unmarked blocks get marked before q gets requested again.

*3) **Analysis of GLOBAL-MAXIMA**:* We obtain an upper bound on the number of cache misses incurred by GLOBAL-MAXIMA in a given marking phase as a function of the number of distinct requests to clean blocks l during the phase, and the number of processes p. Recall that a clean block for a phase is a block that did not reside in the cache at the beginning of the phase. We will make use of the following 2 bounds while establishing the upper bound on the cost of GLOBAL-MAXIMA. The first bound directly follow from the definitions of local and global distance functions. The second bound can be proved using the pigeonhole principle.

- **Bound 1:** For all $x \in U : 0 \leq d_i(x) \leq k-1$ (for all $1 \leq i \leq p$) and $0 \leq d(x) \leq k-1$.
- **Bound 2:** For every eviction candidate q in our algorithm, $\frac{|U|}{p} - 1 \leq d(q) \leq k-1$.

Before presenting the analysis, we also review a few terms from [1].

- **Clean and non-clean blocks:** A block q is said to be a clean block if it did not reside in the cache at the start of the phase. All the blocks that reside in the cache at the start of a phase are called non-clean blocks.
- **Hole and hole movement:** A hole is said to be created at a non-clean block q if it gets evicted during the current phase. If the missing block q is requested again during this phase, another unmarked block q' is evicted. We say that the hole h *moves* from q to q'.

Theorem II.3. *The competitive ratio of* GLOBAL-MAXIMA *in the (k,k)-paging SFK model is at most $2(p\ln(ek/p)+1)$.*

Proof: Requests to l clean blocks result in l cache misses and hence creation of l holes. We now bound the number of cache misses due to repeated requests to one such hole, h, during the current phase. Let u_0 be the number of unmarked blocks in the cache when h is created (due to a request to a clean block). Note that, $u_0 \leq k$. Let u_j represent the number of unmarked blocks in the cache when the non-clean block associated with h is requested for the jth time.

Using Bound 2, it is easy to see that at least $\frac{u_{j-1}}{p}$ blocks are marked between two consecutive requests to the non-clean blocks associated with h. Hence, $u_{j-1} - u_j \geq \frac{u_{j-1}}{p}$. This implies that $u_j \leq u_0 \cdot (1-1/p)^j \leq k \cdot \frac{1}{e^{\frac{j}{p}}}$. With at most $p\ln(k/p)$ requests to h, the number of unmarked blocks in the cache reduces to at most p. The total number of cache misses due to h during this phase is at most $p\ln(k/p)+p = p\ln(ek/p)$. Since every cache miss (hole) is treated in a similar fashion, the total number of cache misses due to l holes is bounded by $l + lp\ln(ek/p)$ cache misses. Recall that OPT incurs at least $l/2$ cache misses on this phase in an amortized sense. Hence, the competitive ratio of GLOBAL-MAXIMA in the SFK model is at most $2(p\ln(ek/p)+1)$. □

B. Caching in the DFK model

In the DFK model, studied in [1], p processes share a cache and access disjoint sets of memory blocks. Thus each memory block *belongs* to a unique process. Each process has full knowledge about its individual request sequence.

We present RR-PROC-MARK, a simple deterministic CRA which attains a competitive ratio of at most $\max(10, p+1)$, which is optimal when $p \geq 9$ [1]. RR-PROC-MARK is an example of a *process marking algorithm*, a class of CRA that we next define.

Definition II.4. *A process marking algorithm proceeds in marking phases. A phase starts with all blocks and processes unmarked. A block is marked when a request to it is served. A process gets marked when all of the blocks in cache that belong to it are marked. Upon a cache miss, an unmarked process, if available, is asked to make an eviction. This process evicts one of the unmarked blocks that belongs to it from the cache. If no unmarked process is available, a new phase is started with all processes and blocks unmarked. An eviction decision taken by a process is considered* good *if the process gets marked before the evicted block is requested again during the current phase.*

The randomized DFK algorithm in [1] is an example of a process marking algorithm (with processes always making good evictions). The LRU based deterministic CRA presented in [2] is an example of a implicit process marking algorithm (the marking scheme is implicitly followed by LRU). We have formalized this notion into a general framework for DFK in this paper. The proofs of the following two lemmas are in the first author's Masters Thesis [15].

Lemma II.5. *(Proof in [15]) Once a process gets marked during the phase of a process marking algorithm, it remains marked till the end of the phase.*

Lemma II.6. *(Proof in [15]) If the processes picked by a process marking algorithm always make good evictions, the number of holes associated with every unmarked process is non-decreasing and the number of holes associated with every marked process is non-increasing.*

The following lemma follows from the proof of the lower bound on the cost of OPT in [5].

Lemma II.7. *[5] Consider a marking phase of a process marking algorithm with l clean block requests. The cost of an optimal offline algorithm (*OPT*) for this phase is at least $\frac{l}{2}$ in an amortized sense.*

*1) **Deterministic process marking algorithm: RR-PROC-MARK**:* Upon a cache miss, RR-PROC-MARK picks an unmarked process (also referred to as a victim process) using the *round robin scheme* and asks it to make an eviction; this process evicts an unmarked block from the cache that belongs to it and is requested farthest in its future request sequence (we refer to this choice as MARK-FITF). Observe that MARK-FITF always ensures good evictions.

Analysis of RR-PROC-MARK.

Our analysis obtains an upper bound on the number of cache misses that occur under RR-PROC-MARK in a given marking phase as a function of l, the number of distinct requests to *clean blocks* during the phase, and p, the number of processes. For the upper bound we consider the two cases $l < p$ and $l \geq p$ separately.

Upper bound on the cost of RR-PROC-MARK when $l < p$: In order to bound the cost of RR-PROC-MARK during the phase when $l < p$, we divide the current phase into stages, where the jth stage starts with the request that results in an unmarked process being associated with j holes for the first time. Each hole is attributed to a distinct clean block request, hence there are at most l holes present at any point. Using Lemma II.6 we can conclude that there are at most l stages.

Observe that Lemma II.5 together with the fact that RR-PROC-MARK always makes good evictions gives an upper

bound of $l \cdot p$ on the number of caches misses incurred. We will establish a stronger bound in Lemma II.10, which uses the following lemma in its proof.

Lemma II.8. *[15] The number of holes associated with an unmarked process is $j-1$ just before the start of stage j.*

The following is a corollary for Lemma II.8. This corollary continues to hold for the case when $l \geq p$ because both the definition of the stage and proof for the lemma does not depend on the $l < p$ inequality.

Corollary II.9. *The difference between the maximum number of holes associated with an unmarked process and the minimum number of holes associated with an unmarked process is at most 1 at any point during the phase.*

Lemma II.10. *The number of cache misses in stage 1 is at most p and the number of cache misses in stage j (m_j for $j \geq 2$) is at most $\min(\lfloor \frac{\lambda_{j-1}}{j-1} \rfloor, \lceil \frac{l}{j-1} - 1 \rceil)$, where λ_j is the number of clean blocks requested in stages 1 through j.*

Proof: (Sketch, full proof is in [15].) From Lemma II.8, it is not difficult to see that there are at most $\lfloor \frac{\lambda_{j-1}}{j-1} \rfloor$ unmarked processes at the start of the jth stage (for $j \geq 2$), since each such process must hold at least $j-1$ holes. Further, when $\lambda_{j-1} = l$ (i.e., we have a total of l holes in the system just before the start of stage j), we establish a slightly tighter bound. If none of the l holes are associated with marked processes, an unmarked process should start this stage by requesting an associated hole. At that point, it gets marked and the number of holes associated with at most $\frac{l}{j-1} - 1$ unmarked processes can increase by 1. Hence the number of cache misses in this case is at most $\frac{l}{j-1} - 1 (\leq \lceil \frac{l}{j-1} - 1 \rceil)$. If some (say $x \geq 1$) of these holes are associated with marked processes, the number of unmarked processes is at most $\frac{l-x}{j-1} (\leq \lceil \frac{l}{j-1} - 1 \rceil)$. Hence, the number of cache misses during stage j (m_j) is at most $\min(\lfloor \frac{\lambda_{j-1}}{j-1} \rfloor, \lceil \frac{l}{j-1} - 1 \rceil)$. □

Lemma II.11. *Consider a phase of* RR-PROC-MARK *with l clean block requests. The cost of* RR-PROC-MARK *is at most $\frac{l}{2} \cdot \max(5, p+1)$ for this phase when $l < p$.*

Proof: The number of cache misses incurred by RR-PROC-MARK ($cost$(RR-PROC-MARK)) during this phase is at most $p + \sum_{j=2}^{l} \min(\lfloor \frac{\lambda_{j-1}}{j-1} \rfloor, \lceil \frac{l}{j-1} - 1 \rceil) \leq p + l \cdot H_{l-1}$

For $l \geq 6$, $cost$(RR-PROC-MARK) $\leq \frac{l}{2}(p+1)$ and a case-by-case analysis for $l < 6$ shows that $cost$(RR-PROC-MARK) $\leq \frac{l}{2} \cdot \max(5, p+1)$. □

Upper bound on the cost of RR-PROC-MARK **when $l \geq p$:** In order to bound the cost of RR-PROC-MARK during the phase when $l \geq p$, we again divide the phase into stages, but now stage j starts with the request that results in $j-1$st process getting marked and ends just before the request that results the in jth process getting marked (or the last request in the current phase). We will have at most $p+1$ stages. Without loss of generality, we assume that P_j gets marked due to the first request in $j+1$st stage.

Lemma II.12. *The number of holes associated with P_i just before it gets marked is at most $\frac{\lambda_{i-1}}{u_i} + 1$, where λ_i is the number of clean blocks requested in stages 1 through i and u_i is the number of unmarked processes just before the start of stage i.*

Proof: Using Corollary II.9, it is not difficult to see that the maximum number of holes associated with any unmarked processes at the start of stage j is at most $\lfloor \frac{\lambda_{i-1}}{u_i} \rfloor \leq \frac{\lambda_{i-1}}{u_i} + 1$. □

Lemma II.13. *Consider a phase of* RR-PROC-MARK *with l clean block requests. The cost of* RR-PROC-MARK *is at most $l \cdot (H_p + 2) \leq \frac{l}{2} \cdot \max(10, p+1)$ for this phase when $l \geq p$.*

Proof: Similar to the case when $l < p$ we have, $cost$(RR-PROC-MARK) $\leq \sum_{j=0}^{p} l_j + \sum_{i=1}^{p} n_i$, where, n_i is the number of cache misses due to requests to holes by process P_i. From Lemma II.12, $cost$(RR-PROC-MARK) $\leq \sum_{j=0}^{p} l_j + \sum_{i=1}^{p} \frac{\lambda_{i-1}}{u_i} + 1$. Further, $u_i = p - i + 1$ since exactly one process gets marked in every stage. Hence, $cost$(RR-PROC-MARK) $\leq \sum_{j=0}^{p} l_j + \sum_{i=1}^{p} \frac{l}{p-i+1} + 1 \leq l \cdot (1 + p/l + H_p)$. Given $l \geq p$, the cost of RR-PROC-MARK is at most $l \cdot (H_p + 2) \leq \frac{l}{2} \cdot \max(10, p+1)$. □

Since OPT incurs at least $l/2$ cache misses per phase with l clean page requests in an amortized sense [5], [16], we have the following theorem:

Theorem II.14. *With p processes, the competitive ratio of* RR-PROC-MARK *is at most $\max(10, p+1)$ under (k, k) paging.*

*2) **Additional features of RR-PROC-MARK**:* In the previous subsection, we proved that RR-PROC-MARK is optimal in the disjoint memory full knowledge model under fixed interleaving except when the number of processes is very small. We now discuss some of its additional features.

Efficient implementation: Apart from being optimal in most cases, RR-PROC-MARK is also an extremely simple algorithm to implement when compared to the LRU based algorithm proposed in [2] and analyzed in [1]. Keeping track of the process which owns the least recently used (LRU) block requires more bits and more computation than the corresponding resources in RR-PROC-MARK.

Fairness: RR-PROC-MARK is a fair algorithm in the sense that all the processes are asked to make an almost equal number of evictions. On certain adversarial sequences, the LRU based algorithm proposed in [2] could ask a subset of processes to make a large number of evictions, and this could have an adverse effect on their performance.

Performance under full access cost: The *full access cost model* was proposed in [17] as a more realistic cost model for sequential caching. In contrast to the classical

competitive analysis, in which we measure the ratio of the total number of cache misses incurred by the online algorithm to the total number of cache misses incurred by the offline algorithm, in the full access cost model, we measure the ratio of the total *access cost* incurred by the online algorithm to the total access cost incurred by the offline algorithm to serve the entire request sequence. The access cost of a block request is the time taken to serve the request.

In this cost model, we show that RR-PROC-MARK has an exponentially better competitive ratio in comparison to the LRU based algorithm presented in [2]. We let t_{hit} and t_{miss} represent the time needed to serve a request that results in a cache hit and a cache miss, respectively. Let n be the total number of requests in the given request sequence σ and m be the total number of cache misses incurred by ALG on this sequence. The access cost of ALG on σ represented by $acost(\text{ALG})$ is given by:

$$\begin{aligned} acost(\text{ALG}) &= (n-m)\cdot t_{hit} + m \cdot t_{miss} \\ &= n \cdot t_{hit} + m \cdot (t_{miss} - t_{hit}) \end{aligned}$$

In order to keep the equations simple, we let $t_{miss} = b$ and $t_{hit} = 1$. It was shown in [17] that the competitive ratio of any marking algorithm in the full access cost model is at most $1 + \frac{(k-1)b}{k+b}$ in sequential NK setting (recall that k is the cache size and b is the delay caused by a cache miss). We show that the competitive ratio of RR-PROC-MARK in the full access cost model is at most $2(H_p + 1)$.

Theorem II.15. *The competitive ratio of* RR-PROC-MARK *in the full access cost model is at most* $2(H_p + 2)$.

Proof: Consider an arbitrary phase of RR-PROC-MARK with l clean block requests and p processes. From Lemmas II.11 and II.13, we have the following for the number of cache misses incurred by RR-PROC-MARK:

$$cost(\text{RR-PROC-MARK}) \leq \begin{cases} p + lH_{l-1} & \text{if } l < p \\ l + p + lH_p & \text{if } l \geq p \end{cases}$$

Let n be the total number of requests during the current phase and let $s = b - 1$. We have the following for the full access cost of RR-PROC-MARK:

$$acost(\text{RR-PROC-MARK}) \leq \begin{cases} n + (p + lH_{l-1})s & \text{if } l < p \\ n + (l + p + lH_p)s & \text{if } l \geq p \end{cases}$$

On the other hand, OPT incurs at least $l/2$ cache misses during this phase in an amortized sense. The cost of OPT in the full access cost model is at least $n + (l/2)s$ per phase in an amortized sense. The competitive ratio of RR-PROC-MARK in the full access cost model is:

$$\frac{acost(\text{RR-PROC-MARK})}{acost(\text{OPT})} \leq \begin{cases} \frac{n+(p+lH_{l-1})s}{n+(l/2)s} & \text{if } l < p \\ \frac{n+(l+p+lH_p)s}{n+(l/2)s} & \text{if } l \geq p \end{cases}$$

Note that the total number of requests (n) in the considered phase of RR-PROC-MARK is at least k (since the phase ends only after all of the k blocks in the cache are marked).

$$\frac{acost(\text{RR-PROC-MARK})}{acost(\text{OPT})} \leq 1 + \begin{cases} \frac{(p+l(H_{l-1}-1/2))s}{k+(ls/2)} & \text{if } l < p \\ \frac{(H_p+3/2)ls}{k+(ls/2)} & \text{if } l \geq p \end{cases}$$

Observe that the above expression monotonically increases with l and achieve maximum value at the maximum value of l. Recall that the number of clean block requests during a marking phase is at most k. Hence, the maximum value of l in the above two cases is $p-1$ and k, respectively. Also, usually k is orders of magnitude greater than p.

Simplifying the above equations we obtain the desired result that the competitive ratio of RR-PROC-MARK in the full access cost model is at most $2(H_p + 1)$. □

In contrast, it can be shown that the competitive ratio of the LRU based CRA presented in [2] is $\Omega(p)$. The reason for the exponentially better performance of RR-PROC-MARK in the full access cost model is that it performs much better when the number of clean block requests per phase is large. When the number of clean block requests per phase is small, the total cost is small, and hence the total time needed for computation remains relatively small for RR-PROC-MARK.

C. (h, k)-paging in DFK and SFK models

In (h, k)-paging, the online CRA is given a cache of size k and the offline CRA is given cache of size $h < k$. LRU is known to be 1-optimal in the NK model under (h, k)-paging with competitive ratio $\frac{k}{k-h+1}$ [3]. When $h = c \cdot k$, for a constant $c < 1$, LRU also has constant competitive ratio. Since LRU can be used under the DFK and SFK by ignoring the future knowledge, this gives an $O(1)$-competitive CRA for both DFK and SFK under (h, k)-paging, for $h = c \cdot k$ and $c < 1$.

D. *Caching in the FK model under free interleaving*

An online CRA in this model is as powerful as the offline CRA. It was shown in Section 3.2 of [6] that the optimal offline algorithm at a single shared cache is NP complete in this model by reducing a slightly modified 3-partition problem to the multicore caching problem when $k = p/3$, where k is the size of the cache and p is the number of processes (cores). In the real world, k is at least as large as p and usually orders of magnitude larger than p. This NPC result can be extended to general k and p by making core i repeatedly request y_i rounds of requests with each round consisting of a series of $3k/p$ distinct block requests instead of just one distinct block (as in [6]). In order to minimize the total makespan, each core now requires $3k/p$ locations. It is easy to see that the rest of the NPC proof follows in this case as well.

III. CACHE REPLACEMENT ALGORITHMS FOR A HIERARCHY OF CACHES

Multiple levels of caches are the norm in multicores, and the types of higher-level caches that are widely used in practice are *inclusive*, *exclusive* and *partially-inclusive*. For instance, the L_2 cache is partially-inclusive in the Intel-Nehalem and exclusive in the AMD-Shanghai processor; the L_3 is inclusive in the former and partially-inclusive in the latter [18], [19], [20]. We aim to compare these caches using competitive analysis. Our competitive analysis is focussed on the cache misses at the highest level in the shared cache hierarchy since the cache miss delay is maximum at this level.

For concreteness we initially consider a 2-level shared cache hierarchy. We are not aware of competitive analysis of CRAs in this setting even in the sequential (uni-processor) case under NK, the classical no-knowledge case. We then generalize these results to CRAs for multicore in the NK and FK models under fixed interleaving at a 2 level shared cache, and also generalize these results to multiple levels. For free interleaving, we show in Section III-C that the negative results in [6] continue to hold for the NK model at a 2-level shared cache.

Road map. In Section III-A, we define the 3 types of caches for a 2-level cache. In Section III-B, we introduce the notion of *cache equivalence*, which relates the performance of a CRA at a single-level cache with CRAs at each of the three types of 2-level caches; this allows us to establish that suitable variants of LRU have optimal competitive ratio for INCL, EXCL and PRT-INCL L_2 caches with the corresponding cache equivalence sizes. These results are obtained for single core with a 2-level cache hierarchy. We then generalize these results to a shared cache hierarchy, and to multiple levels of shared caches in section III-C.

A. *Inclusive, exclusive and partially-inclusive caches*

We now define the three commonly used types of 2 level caches based on their descriptions in [21], [22], [19], [20]. Here, for a block β in cache L_i, we use the term *evicted* to denote that β is moved from L_i to main memory, and we use the term *removed* to denote that β is moved out of L_i either to a higher cache or to main memory, the decision being left to the CRA. A block β in cache L_i is *swapped* with a block β' in the other cache if β' is moved into L_i and β is moved into the other cache.

1. **Inclusive cache (INCL):** The L_2 cache is forced to include the contents of the L_1 cache. When the requested block is in L_2 but not in L_1, the block is read into L_1 after the CRA is asked to evict a block from L_1. When the requested block is not in L_1 and L_2, the block is read into both L_1 and L_2. In order to accommodate the requested block, the CRA is asked to evict a block from L_2. It is then asked to evict a block from L_1, such that L_2 still includes all the blocks in L_1.

2. **Exclusive cache (EXCL):** The L_2 cache is forced to exclude the contents of the L_1 cache. When the requested block is in L_2 but not in L_1, the block is read into L_1 after the CRA is asked to swap the requested block with a block in L_1. When the requested block is not in L_1 and L_2, the block is directly read into L_1 after the CRA is asked to remove a block from L_1. The CRA either stores the block removed from L_1 in L_2 after evicting another block from L_2 or evicts the block removed from L_1 (in which case L_2 remains unchanged).

3. **Partially-Inclusive cache (PRT-INCL):** The L_2 cache is not forced to either include or exclude the contents of the L_1 cache. When the requested block is in L_2 but not in L_1, the block is read into L_1 after the CRA evicts a block from L_1. When the requested block is not in L_1 and L_2, the block is read into both L_1 and L_2. In order to accommodate the requested block, the CRA is asked to evict a block each from L_1 and L_2.

B. *2-level cache hierarchy for a single core*

We present the key concept of *cache equivalence*, which allows us to equate the performance of a CRA at a 2 level cache with a corresponding CRA at a single-level cache of suitable size. This, together with our results on 2-FITF in the next section allows us to establish upper and lower bounds on the competitive ratio of CRAs at a 2 level cache with INCL, EXCL and PRT-INCL L_2 caches. ALG-TYPE is said to incur a cache miss at L_2 if the requested block data item does not exist in both L_1 and L_2.

Cache equivalence: A CRA ALG-TYPE at a 2 level cache (L_1 of size k_1 and L_2 of size k_2) is said to be *equivalent* to a CRA ALG at a single level cache L of size $k = f(k_1, k_2)$, if the number of cache misses incurred by ALG-TYPE at L_2 is same as the number of cache misses incurred by ALG at L on any request sequence provided the 2-level cache initially contains the initial contents of the single level cache.

Lemma III.1. *(cache equivalence) Consider a 2 level cache with L_1 and L_2 of size k_1 and k_2, respectively. Let* ALG *be a deterministic CRA at a single level cache. There exists a CRA,*

1. ALG-INCL *at the 2 level cache with INCL L_2 which is equivalent to* ALG *at a single level cache of size k_2.*

2. ALG-EXCL *at the 2 level cache with EXCL L_2 which is equivalent to* ALG *at a single level cache of size $k_1 + k_2$.*

3. ALG-PRT-INCL *at the 2 level cache with PRT-INCL L_2 which is equivalent to* ALG *at a single level cache of size $k_1 + k_2 - 1$.*

Proof: We construct ALG-INCL, ALG-EXCL and ALG-PRT-INCL for a 2 level cache with INCL, EXCL and PRT-INCL L_2 based on the definition of ALG at a single level cache of size k_2, k_1+k_2 and k_1+k_2-1, respectively. Further, we shall prove the cache equivalence based on the definition of ALG-INCL, ALG-EXCL and ALG-PRT-INCL CRAs.

Definition of ALG-INCL*:*

- When the requested block β is in L_1: ALG-INCL serves the request.
- When β is in L_2 but not in L_1: it is read into L_1 after evicting a block from L_1 that ALG would have evicted from a cache of size k_1 and the same contents as L_1.
- When β is not in L_1 and L_2: a block from L_2 that ALG would have evicted from a cache of size k_2 and the same contents as L_2 is located and evicted. Let this block be γ. If γ is in L_1, it is evicted from L_1 as well. If γ is not in L_1, the block in L_1 that ALG would have evicted from a cache of size k_1 and the same contents as L_1 is evicted. β is read into both L_2 and L_1.

Definition of ALG-EXCL:

- When the requested block β is in L_1: ALG-EXCL serves the request.
- When β is in L_2 but not in L_1: β is swapped with a block from L_1 that ALG would have evicted from a cache of size k_1 and the same contents as L_1.
- When β is not in L_1 and L_2: a block from L_1 and L_2 together that ALG would have evicted from a cache of size k_1+k_2 and same contents as L_1 and L_2 is located. Let this block be γ. If γ is in L_1, it is evicted from L_1. If γ is not in L_1, it is evicted from L_2 and a block in L_1 that ALG would have evicted from a cache of size k_1 and the same contents as L_1 is moved to L_2. β is directly read into L_1.

Definition of ALG-PRT-INCL:

- When the requested block β is in L_1: ALG-PRT-INCL serves the request.
- Let η be a block that exists in both L_1 and L_2 (observe that there is at least one such block since the most recently read block is read into both L_2 and L_1).
- When β is in L_2 but not in L_1: it is read into L_1 after evicting η from L_1.
- When β is not in L_1 and L_2: a block from L_1 and L_2 together that ALG would have evicted from a cache of size k_1+k_2-1 and same contents as L_1 and L_2 is located. Let this block be γ. If γ is in L_1, it is evicted from L_1 and η is evicted from L_2. If γ is not in L_1, it is evicted from L_2 and η is evicted from L_1. β is read into both L_2 and L_1.

We prove by induction on the number of requests that ALG-INCL, ALG-EXCL for ALG-PRT-INCL take the same decision as ALG at a single level cache of size k_2, k_1+k_2 and k_1+k_2-1, respectively on each block request. We assume that the initial contents of the single level cache is same as that of the 2 level cache in each of the three cases. Consider the first request. It could either be a cache hit or a cache miss for ALG at L.

Cache hit: If the first request results in a cache hit for ALG, it results in a cache hit at either L_1 or L_2 for ALG-INCL, ALG-EXCL for ALG-PRT-INCL. Hence the content of the 2 level cache remains unchanged.

Cache miss: If the first request results in a cache miss for ALG, it results in a cache miss at both L_1 and L_2 for ALG-INCL, ALG-EXCL for ALG-PRT-INCL. By the definition of these algorithms, the block that gets evicted from the 2 level cache is same as the block that is evicted by ALG from L. Hence, the contents of single level cache is same as that of the 2 level cache after serving the first request. By induction the result holds for each request in the request sequence, and the lemma follows. □

*1) **Optimal offline CRA at a 2 level cache hierarchy**:* An optimal offline CRA (OPT) is assumed to have full knowledge about the request sequence reaching all levels of caches and for fair comparison of different types of caches, we assume that OPT has no constraint imposed by the cache type. Denote by h_i the size of L_i cache for OPT. Our optimal offline CRA for this model is an extension of FITF (referred to as 2-FITF). We shall show that 2-FITF is optimal in terms of the number of cache misses at the L_2 cache. The algorithm is defined as follows:

2-FITF: When the requested block is in L_2 but not in L_1: the block is swapped with a block in L_1 that is requested farthest in the future request sequence. When the requested block is not in L_1 and L_2: the block is directly read into L_1 after removing a block (γ) from L_1 that is requested farthest in the future request sequence. Let δ be a block in L_2 that is requested farthest in the future request sequence. If δ is requested after γ, δ is evicted and γ occupies its position in L_2. If γ is requested after δ, γ is evicted.

The following theorem is proved by establishing that on every cache miss, 2-FITF at the L_2 cache takes the same decision as FITF at a single shared cache of size h_1+h_2.

Theorem III.2. *[15]* 2-FITF *at a 2 level cache, where the L_1 cache has size h_1 and the L_2 cache has size h_2, is equivalent to* FITF *at a single level cache of size h_1+h_2.*

Theorem III.3. *Let L_1 and L_2 have sizes k_1 and k_2, respectively. Let ρ be the optimal competitive ratio at the L_2 cache. Then, ρ is same as that of a single level cache of size k_2 for an INCL cache, size k_1+k_2 for an EXCL cache, and size k_1+k_2-1 for a PRT-INCL cache.*

Proof: For any sequential deterministic CRA, at most k_2, k_1+k_2 and k_1+k_2-1 distinct cache blocks exists in L_1 and L_2 when the L_2 cache is INCL, EXCL and PRT-INCL, respectively. Hence, using adversarial sequences consisting of k_2+1, k_1+k_2+1 and k_1+k_2 distinct cache block requests, lower bound on the competitive ratio of sequential deterministic CRA at the 2 level cache can be shown to be no less than that of sequential deterministic CRA at a single level cache of size k_2, k_1+k_2 and k_1+k_2-1, respectively.

Further, the upper bound on the competitive ratio of ALG-INCL, ALG-EXCL and ALG-PRT-INCL at a 2 level cache with

INCL, EXCL and PRT-INCL L_2 caches can be shown to be equal to that of ALG at a single level cache of size k_2, $k_1 + k_2$ and $k_1 + k_2 - 1$, respectively, using cache equivalence of ALG-INCL, ALG-EXCL, ALG-PRT-INCL and OPT (Lemma III.1 and Theorem III.2). □

Theorems III.2 and III.3 lead to the following corollary. Similar results hold in the FK model.

Corollary III.4. LRU-INCL, LRU-EXCL *and* LRU-PRT-INCL *(as defined in the Lemma III.1) are 1-optimal at a 2 level cache hierarchy with INCL, EXCL and PRT-INCL L_2 caches, respectively.*

C. CRAs for a shared cache hierarchy

We extend results for sequential cache hierarchy to shared cache hierarchy. In the following discussion, we use TYPE to represent the type of the L_2 cache — INCL, EXCL and PRT-INCL. First we extend our results to a simple 2 level shared cache hierarchy.

CRAs at a 2 level shared cache hierarchy under fixed interleaving: Our results for sequential cache hierarchy can be easily generalized to parallel deterministic CRAs in NK and FK models under fixed interleaving at a 2 level shared cache hierarchy. The main observation is that Lemma III.1 continues to hold for parallel deterministic CRAs in the NK and FK models under fixed interleaving. In section II, we proved that RR-PROC-MARK and GLOBAL-MAXIMA are optimal (up to a constant factor) in DFK and SFK under fixed interleaving (when considered under (k, k)-paging). Modified versions of these algorithms (RR-PROC-MARK-TYPE and GLOBAL-MAXIMA-TYPE) continue to be optimal (up to a constant factor) at a 2 level cache for suitable values of k_1, k_2, h_1 and h_2.

CRAs at a 2 level shared cache hierarchy under free interleaving: We extend results from [6] to a 2 level shared cache hierarchy. Let the cost of serving a block request be 1 for L_1, b_1 for L_2, and b_2 for the main memory. We let k_i, the size of L_i for online CRAs be larger than h_i, that of the optimal offline CRA (resource augmentation). For our lower bound we use a sequence that is similar to the one in the lower bound proof in [6]. Our sequence, however, depends on the effective cache size. Consider a 2 level cache hierarchy with an INCL L_2 cache and let LRU-INCL be the CRA at this 2 level cache. Our adversarial sequence has each core requesting h_1+h_2 distinct cache blocks in parallel b_2+1 times. If $k_2 < p(h_1 + h_2)$, each of these requests results in a cache miss at L_2 for LRU-INCL and hence the total time taken to serve these requests is $(b_2 + 1)b_2(h_1 + h_2)$. On the other hand, OPT serves one core after another. Each core takes $b_2(h_1+h_2)+((h_1-1)+(h_2-h_1+1)b_1)b_2$ time to serve all its requests. Hence the total time is $(b_2(h_1+h_2)+((h_1-1)+(h_2-h_1+1)b_1)b_2)p$. The total time taken by OPT to serve requests from all p cores is $O((h_1+h_1)b_1b_2p)$. Hence, the competitive ratio of LRU-INCL in terms of make span is $\Omega(b_2/b_1p)$ when $h_2 < k_2 < p(h_1 + h_2)$. For a constant number of cores, the lower bound on the competitive ratio of LRU-INCL is $\Omega(b_2/b_1)$.

CRAs at a multi level shared cache hierarchy: Consider an l-level shared cache hierarchy, where the ith level cache L_i has size k_i. Typically, $k_1 < k_2 < \cdots < k_l$, with L_1 closest to the core and L_l closest to the main memory. Practical caching systems often use different types of caches at different levels. However, the number of different configurations increases exponentially with l, and hence we focus on multiple levels of same-type cache in this paper.

Definition III.5. *The L_i cache is said to be an inclusive cache if the contents of the L_i cache includes the contents all the lower level caches $(L_1, L_2, .. L_{i-1})$ and each of the lower level cache is inclusive as well. Similarly, the L_i cache is said to be an exclusive cache if the contents of the L_i cache excludes all the lower level caches and each of the lower level cache is exclusive as well. The L_i cache is non inclusive if neither of these constraints are enforced but, when a block request incurs a cache miss at L_i, it is read into L_i and all the lower level caches as well $(L_1, L_2, \ldots L_{i-1})$.*

Theorem III.6. *Consider a l level same-type cache hierarchy. Let the size of L_i be k_i for the online CRAs and h_i for the optimal offline CRA for $1 \le i \le l$. Let $s(x_l) = \sum_{i=1}^{l} x_i$. In the NK model under fixed interleaving,* LRU-TYPE *is 1-optimal with competitive ratio:*

1. $\frac{k_l}{k_l - s(h_l)+1}$ *for INCL L_l.*
2. $\frac{s(k_l)}{s(k_l)-s(h_l)+1}$ *for EXCL L_l.*
3. $\frac{s(k_l)-l+1}{s(k_l)-s(h_l)-l+2}$ *for PRT-INCL L_l.*

Proof: The effective size of L_l for LRU-TYPE can be obtained by extending the proof for effective size of L_2 (Lemma III.1) using induction on the number of levels. The effective size of L_l for LRU-TYPE is $k_l, s(k_l)$ and $s(k_l)-l+1$ for INCL, EXCL and PRT-INCL L_l, respectively. Also, note that the optimal offline algorithm for l levels of shared caches is an extension of 2-FITF, denoted by L-FITF. L-FITF tries to maintain exclusive set of cache blocks in each level of cache and retain the blocks which will be requested in the recent future at the lowest level caches. Using induction on the number of levels, one can easily extend Theorem III.2 to prove that L-FITF is optimal for l levels of caches and the effective cache size for L-FITF is $s(h_l)$. The upper bound competitive ratio of LRU-TYPE follows from these observations. Further, using sequences similar to the ones defined in Theorem III.3, we can prove that LRU-TYPE is in fact optimal. □

IV. DISCUSSION

For a single level shared cache, our main contributions were for the FK model under fixed interleaving. In the shared memory FK model under fixed interleaving, we gave new lower bounds on the competitive ratio of deterministic and

randomized CRAs. We also presented a deterministic CRA (GLOBAL-MAXIMA) which is $O(1)$-optimal in this model, and we established that PARTITION [4], [5] is $O(1)$-optimal for randomized CRAs. As discussed in Section I, the SFK model relates well to a number of parallel shared memory applications. Obtaining a strongly competitive (1-optimal) CRA is a topic for further research.

In the disjoint memory FK model, we presented a deterministic CRA (RR-PROC-MARK) with $\max(10, p + 1)$ competitive ratio. A lower bound of $p+1$ was established on the competitive ratio of deterministic CRAs in the disjoint memory FK model in [1]. We also noted that RR-PROC-MARK is fairer and more efficient to implement than the LRU based algorithm presented in [2].

For a multi-level cache hierarchy, our results have established that LRU-EXCL has the best competitive ratio at the highest-level cache L_k, followed by LRU-PRT-INCL, and that LRU-INCL, which is optimal for INCL, is the worst with respect to competitive ratio for CRAs. This validates experimental findings on the superiority of EXCL [23], [24].

PRT-INCL is considered to be simpler and more efficient to implement than EXCL caches in part because the swap operation in EXCL can be more expensive than eviction or removal. It also appears that PRT-INCL caches are common in practical systems: both Intel Nehalem and AMD Shanghai have a PRT-INCL cache. It is noteworthy that our results have established that the competitive ratio of CRAs at PRT-INCL caches is very close to that for EXCL caches.

We have confined our attention to the highest-level cache L_k, since the cache miss cost is largest at that level. However, it is not difficult to see that LRU-INCL and LRU-EXCL have the desirable property that they support the optimal LRU at the lower level caches. However, LRU-PRT-INCL does not share this property. Obtaining tight bounds for a CRA at all cache levels for PRT-INCL is a topic for further investigation, as is the topic of extending these results to a hierarchy of private and shared caches (a tree-like cache hierarchy).

REFERENCES

[1] R. D. Barve, E. F. Grove, and J. S. Vitter, "Application-controlled paging for a shared cache," *SICOMP*, vol. 29(4), 1995.

[2] P. Cao, E. W. Felten, and K. Lee, "Application-controlled file caching policies," *USENIX*, pp. 171–182, 1994.

[3] D. D. Sleator and R. E. Tarjan, "Amortized efficiency of list update and paging rules," *CACM*, vol. 28(2), pp. 202–208, 1985.

[4] L. A. McGeoch and D. D. Sleator, "A strongly competitive randomized paging algorithm," *Algorithmica*, vol. 6, pp. 816–825, 1991.

[5] A. Fiat, R. Karp, M. Luby, L. McGeoch, D. Sleator, and N. Young, "On competitive algorithms for paging problems," *Journal of Algorithms*, vol. 12, pp. 685–699, 1991.

[6] A. Hassidim, "Cache replacement policies for multicore processors," *ICS*, 2010.

[7] A. Lopez-Ortiz and A. Salinger, "Brief announcement: Paging for multicore processors," *SPAA*, pp. 137–138, 2011.

[8] R. Chowdhury and V. Ramachandran, "The cache-oblivious Gaussian elimination paradigm: Theoretical framework, parallelization and experimental evaluation." *Theory of Computing Systems*, vol. 47(1), pp. 878 – 919, 2010.

[9] R. Chowdhury, F. Silvestri, B. Blakeley, and V. Ramachandran, "Oblivious algorithms for multicores and network of processors," *IEEE IPDPS*, 2010.

[10] D. Callahan, K. Kennedy, and A. Porterfield, "Software prefetching," *ASPLOS IV*, pp. 40–52, 1991.

[11] T. C. Mowry, M. S. Lam, and A. Gupta, "Design and evaluation of a compiler algorithm for prefetching," *ASPLOS V*, pp. 62–73, 1992.

[12] A. L. Belady, "A study of replacement algorithms for virtual storage computers," *IBM Sys Jour*, vol. 5, pp. 78–101, 1966.

[13] A. Borodin, S. Irani, P. Raghavan, and B. Schieber, "Competitive paging with locality of reference," *JCSS*, vol. 50, pp. 244–258, 1995.

[14] A. C.-C. Yao., "Probabilistic computations: Towards a unified measure of complexity," *FOCS*, pp. 222–227, 1977.

[15] A. Katti, "Competitive cache replacement strategies for a shared cache," *Masters thesis, UT Austin*, 2011.

[16] N. Young, "Competitive paging and dual-guided on-line weighted caching and matching algorithms," *Dissertation, Princeton University*, 1991.

[17] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

[18] D. Hackenberg, D. Molka, and W. E. Nagel, "Comparing cache architectures and coherency protocols on x86-64 multicore smp systems." *MICRO*, pp. 413–422, 2009.

[19] INTEL, "White paper intel xeon processor 3500 and 5500 series intel microarchitecture." 2009.

[20] AMD, "http://blogs.amd.com/developer/2008/11/13/larger-l3-cache-in-shanghai-part-i," 2009.

[21] J. L. Baer and W. H. Wang, "On the inclusion properties for multi-level cache hierarchies," *ISCA*, pp. 73–80, 1988.

[22] N. P. Jouppi and S. J. E. Wilton, "Tradeoffs in two-level on-chip caching," *ISCA*, pp. 34–45, 1994.

[23] M. Zahran, K. Albayraktaroglu, and M. Franklin, "Non-inclusion property in multi-level caches revisited," *Intl Jour of Computers and Their Applications 2007*, June 2007.

[24] J. L. Baer and W. H. Wang, "Retrospective: On the inclusion properties for multi-level cache hierarchies," *ISCA*, pp. 59–60, 1998.