Evaluating the Memory System Behavior of Smartphone Workloads

G. Narancic[1], P. Judd[1], D. Wu[1], I. Atta[1], M. Elnacouzi[1], J. Zebchuk[1],
J. Albericio[1], N. Enright Jerger[1], A. Moshovos[1], K. Kutulakos[2], and S. Gadelrab[3]

[1]*Department of Electrical and Computer Engineering, University of Toronto*
[2]*Department of Computer Science, University of Toronto*
[3]*Qualcomm*
[1]{*patrick.judd, perwudi.wu*}*@mail.utoronto.edu;* {*iatta, michel.elnacouzi, zebchuck, jorge, enright, moshovos*}*@eecg.toronto.edu*
[2]*kyros@cs.toronto.edu*
[3]*sgadelra@qti.qualcomm.com*

***Abstract*—Modern smartphones comprise several processing and input/output units that communicate mostly through main memory. As a result, memory represents a critical performance bottleneck for smartphones. This work[1] introduces a set of emerging workloads for smartphones and characterizes the performance of several memory controller policies and address-mapping schemes for those workloads. The workloads include high-resolution video conferencing, computer vision algorithms such as upper-body detection and feature extraction, computational photography techniques such as high dynamic range imaging, and web browsing. This work also considers combinations of these workloads that represent possible use cases of future smartphones such as detecting and focusing on people or other objects in live video. While some of these workloads have been characterized before, this is the first work that studies address mapping and memory controller scheduling for these workloads. Experimental analysis demonstrates: (1) Most of the workloads are either memory throughput or latency bound straining a conventional smartphone main memory system. (2) The address mapping schemes that balance row locality with concurrency among different banks and ranks are best. (3) The FR-FCFS with write drain memory scheduler performs best, outperforming some more recently proposed schedulers targeted at multi-threaded workloads on general purpose processors. These results suggest that there is potential to improve memory performance and that existing schedulers developed for other platforms ought to be revisited and tuned to match the demands of such smartphone workloads.**

I. Introduction

Smartphones currently represent one of the largest computing device markets with unit shipments projected to rise further [33]. As Figure 1 illustrates, a modern high-end smartphone architecture comprises general purpose and graphics processors, one or more display controllers, a number of sensors such as accelerometers and a touch screen, and other processing units such as video encoders and wireless modems.

Much of the communication among the hardware units happens through the main memory system. As a result, smartphone performance often depends heavily on the main memory system design. While smartphone designs can borrow from conventional computer designs, the physical, energy and power constraints of smartphones are different. Moreover, while a smartphone can be used as a desktop or laptop replacement, typical use cases may be different. The inclusion of several sensors and I/O devices in smartphones enable additional applications while the physical properties and interfaces favor different types of interaction. Smartphones are increasingly used for media-rich applications such as video conferencing, computer vision, and computational photography.

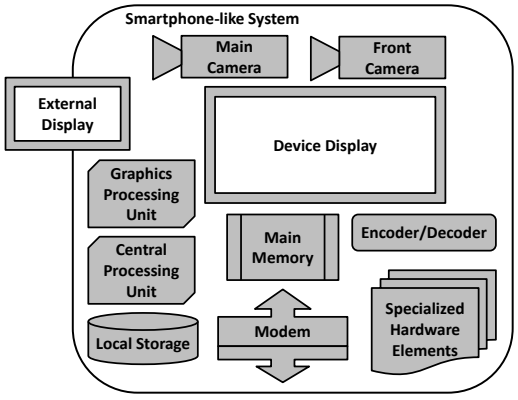


Fig. 1: Smartphone Architecture

Accordingly, the first contribution of this work is that it presents a set of potential smartphone workloads that may strain existing memory systems. The second contribution of this work is that it studies the main memory performance of the presented workloads. The goal of this study is to understand how well existing memory controller policies work for these workloads and to identify opportunities for performance improvement, if any. The workloads include high-definition video conferencing, a number of computer vision and computational photography algorithms, and optical character recognition. While these workloads can be used as standalone applications, we also study combinations representative of potential use cases where the workloads serve as building blocks for more elaborate applications.

This work characterizes the main memory behavior of these workloads considering the impact of the address mapping

[1]This work was supported by NSERC, Qualcomm, the Canadian Foundation for Innovation, and the Ontario Research Fund. Kutulakos was also supported by NSERC DAS, RTI, RGPIN, and GRAND NCE grants.

scheme and of four different memory scheduling policies. Finally, it characterizes the applications identifying those that are memory throughput-, memory latency- or computation-bound. The results demonstrate that: (1) The address mapping schemes that balance spatial locality within DRAM rows and concurrency across banks and ranks perform best; (2) A simple FR-FCFS scheduler with write-drain performs best, narrowly outperforming other more complex schedulers. This suggests that the latter designs should be revisited and tuned according to the demands of smartphone workloads; (3) Some of the applications strain existing memory systems; and (4) there is significant room for improving memory system performance by making better scheduling decisions. These results can serve as motivation for future work in memory controller and memory system design for smartphones.

The rest of this paper is organized as follows. Section II describes the smartphone workloads. Section III and IV describe the experimental methodology and present the experimental analysis, respectively. Section V reviews related work, while Section VI summarizes this work.

II. Smartphone Workloads

This section describes workloads that are representative of typical and future use cases of smartphone devices. These include a video conference workload, a set of computer vision, computational photography workloads, and optical character recognition. As smartphones incorporate multiple, high resolution cameras, such workloads will become increasingly popular either in stand-alone form or as components of more elaborate applications. Accordingly, Section II-A describes a set of workloads that combine some of the aforementioned applications to model additional smartphone use cases.

Video Conference Workload (vcw). The first workload models one side of a two-way video conferencing system. In this video conferencing workload (vcw), each side encodes and transmits a video stream from its camera while at the same time receiving, decoding, and displaying a video stream from the other side. To model future generation smartphones, the workloads uses 1080p video encoded with the H.264 encoding standard [38]. vcw models the memory traffic of five components: camera, encoder, wireless modem, decoder and display.

Figure 2 shows the frame processing where the encoding and decoding proceed concurrently performing the following steps: The camera stores a raw image in memory (E1). The hardware encoder loads the image from memory (E2), encodes it into a video frame in the H.264 format, and stores the result in memory (E3). The modem loads the frame for transmission (E4). The modem receives an encoded video frame and stores it to memory (D1). The decoder loads the frame (D2), decodes it into an image format suitable for the display and writes it back to memory (D3). Finally, the image is loaded and displayed on screen (D4). The above process repeats at a pre-specified frame rate. All communication between the various units is done through memory due to requirements of the multimedia frameworks of commonly used smartphone operating systems; the video processing elements do not have coherent caches. Many different implementations of H.264 encoding/decoding exist. Section III-C details the modeling methodology used.

Scale Invariant Feature Transform (SIFT). The Scale Invariant Feature Transform (SIFT) identifies distinctive regions in an image ("keypoints") and represents them compactly as 128-dimensional feature vectors ("keypoint descriptors") [22]. These descriptors quickly and compactly identify similarities across images and are frequently used in object recognition and image stitching. We adopt the SIFT nearest-neighbor algorithm matching scheme.

Speeded Up Robust Features (SURF). SURF uses basic mathematical approximations and image transforms to simplify computation and is much faster than SIFT [2]. The SURF detector produces 64-dimensional feature vectors reducing computation and matching time. We use SURF with a nearest neighbors matching algorithm.

Face detection. Face detection locates human faces in an image, the first stage in a facial recognition process. We assume that the recognition will be done on the server side on a database of faces, so the smartphone will only be responsible for localizing the face and generating the search key. Face uses the face detection program in libface [17] which implements the Viola-Jones face detector [36] that uses simple rectangular features that can be computed in constant time. Libface uses classifiers trained for specific features, e.g., eyes, nose and mouth, as well as classifiers for whole faces.

High Dynamic Range (HDR). HDR imaging overcomes the limited range of digital cameras by taking a series low-dynamic range photographs (e.g., 8- or 12-bits per color channel) while varying the exposure time. These photos are then merged into a single photo with a higher dynamic range (i.e., higher bit depth). The HDR workload uses `pfscalibrate` which implements the HDR composting technique described by Robertson *et al.* [31].

Image Denoising. Image denoising removes noise from a digital image. A problem of simple denoising algorithms is that they also remove fine structure, detail and texture. Denoise uses the non-local means image denoising algorithm [3], which

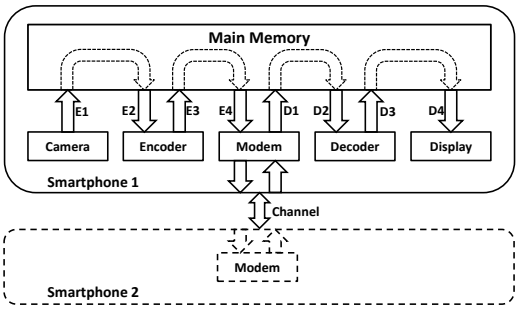


Fig. 2: Frame Processing in a Video Conferencing Workload

preserves fine structure in the image. It finds similar patches in the image and uses the information from these patches to separate the noise from the fine structure.

Upper Body Detection. Upper body detection detects a human's upper body and determines the relative pose of their head and arms. Pose detection is particularly useful because it can infer a person's attitude or action. Upperbody uses the algorithm presented by Eichner *et al.* [8] which models the body as six connected components: the head, the torso, two upper arms, and two lower arms.. It is a fairly unconstrained algorithm in that it allows the arms to be in any position and only requires that the head be above the torso and that the subject be facing directly forward or backward.

Image Segmentation. Image segmentation (segment) divides an image into a set of regions that are relevant for specific tasks (e.g., recognition, foreground-background separation, medical image analysis, image compositing). Segment uses the image segmentation algorithm of Felzenszwalb and Huttenlocher [10] which represents the image as a graph where the nodes are pixels and edges connect adjacent pixels. The edges are weighted with the difference in pixel intensities of the two connected nodes. The algorithm is linear in the number of graph edges and processes edges in order of descending weight. This creates a non-sequential access pattern in the workload.

Web-browsing. The web browsing workload (bbench) runs BBench [11], a web-browsing benchmark over Arora [24], a lightweight internet browser.

A. Combined Workloads

In current and future systems, users are likely to run the above mentioned workloads in various combinations. Here we explore some plausible combinations that are likely to stress our memory system. Cataloguing photos is a common usage scenario for smartphones that uses many image processing algorithms. Segmentation can be used to separate the people in the image from the background. Face detection can identify any people in the image. SURF might be running in the background to identify the location of the photograph; upper body detection might be try to identify a physical activity in a photograph. Hence, we model a combination of face, segment, surf, and upperbody (Combo 1).

Video conferencing may often be accompanied by various computer vision applications. With limited bandwidth, for example, face detection can highlight a region of interest for the video encoder, allowing it to allocate more bits for the face region or it could be used to fetch online information about the person in view. HDR and image denoising are useful for videos in poor lighting conditions. They can improve overall image quality by reducing over- and under-saturation. In addition, a user might be browsing the web, e.g., to access shared documents or looking up related information. We model these scenarios with combinations of VCW with: face, HDR, denoise and bbench; these are respresented by Combos 2-4 in Table I. Table I summarizes the combined workloads.

TABLE I: Workload Summary

SIFT	Feature descriptors using nearest neighbor
SURF	Faster feature detection using approximation and image transformations
face	face detection w/ libface & Viola-Jones face detector
hdr	High Dyanmic Range using pfscalibrate
denoise	Non-local means algorithm
upperbody	Algorithm proposed by Eichner *et al.* [8]
segment	Algorithm by Felzenszwalb and Huttenlocher [10]
Multi-stream workloads	
VCW	camera, encoder (H.264), wireless modem, decoder and display
Combo 1	face + segment + surf + upperbody
Combo 2	bbench + hdr + VCW
Combo 3	bbench + denoise + face + VCW
Combo 4	bbench + face + hdr + VCW

III. Experimental Setup

Smartphone architectures, as Figure 1 shows, are similar to modern personal computers, but commonly include additional units. Such units include one or more cameras, specialized hardware elements such as wireless transceivers, accelerometers, and gyroscopes. Since these components in a smartphone typically communicate and coordinate through main memory, we focus on providing an accurate timing model for main memory and use trace-based models for the rest of elements.

A. Tracing methodology

We use PIN to dynamically instrument the applications to capture their memory access stream during execution [23]. A simple trace that records a sequence of memory accesses with no additional information is not appropriate for studying memory controller performance as it will allow references to proceed unrestricted, resulting in an artificially high concurrency in the memory system. For example, without this information, dependent requests (e.g., chacing pointers) could be issued concurrently, and requests that are separated by several instructions may be issued immediately one after the other. Our trace collection tool analyzes the real data-flow at execution time. We track dependencies at word granularity through registers and memory, and appropriately annotate the resulting traces. To model computational distances we annotate each memory reference with the number of instructions that executed since the last memory reference it depends upon. By scaling the computational distances we mimic the behavior of processing elements with different levels of concurrency.

B. Simulation methodology

We use an in-house simulation engine which models all the cores and PEs. The components of the system are fed with the trace of memory references which also contains dependencies and computational distances. The simulator uses a modified version of DRAMSIM2 which incorporates the policies studied. Section III-D details the simulated memory. Table IIIa

lists the simulated system components and Table IIIc details their configuration. Initially, we consider in-order processing cores that have an IPC equal to 1. Section IV-C considers cores capable of exploiting more instruction level parallelism.

The workloads are executed on conventional processing cores. In order to model such devices, we used 16 KB and 64 KB caches as representative of typical L1 and L2 cache sizes, respectively. As Section III-C explains, we used slightly different cores for VCW due to its high memory demands.

For the computer vision and computational photography workloads, we study two different input sets: low resolution 100×100 pixel images and high resolution 12 MegaPixel (MP) images. Depending on the application, these workloads can be used with the direct input from the camera or with a scaled image. For example, HDR can be used to process large images directly from the camera, while a face detection system for video might scale images prior to processing in order to meet real-time constraints. For bbench, we simulate 2B instructions, and 10B for the rest of workloads.

C. Constructing the Video Conference Workload (VCW)

VCW requires of a more complex tracing methodology. We model it by creating a trace of the memory traffic for each component and synchronizing the rate at which these traces are played back. For the video encoder and decoder, we use PIN [23] to instrument a software implementation of the H.264 standard. We use FFmpeg [1], an open-source video encoder and decoder which uses the libx264 [35] library to perform H.264 encoding. To model on-the-fly encoding, we disable the lookahead and b-frame options that use future information to encode the current frame. We create per-frame traces to record all main memory traffic and to record relevant meta-data (e.g., frame size). We mimic a typical double-buffering scheme for both the encoder and decoder. This is done by post-processing the traces to redirect the input and output frame data structures into different memory locations between subsequent frames.

For the video encoder and decoder we simulate programmable ILP cores. We found that using the aforementioned typical caches sizes resulted in disproportionally large memory traffic as these were not large enough to capture the local processing buffers. Accordingly, we increased the cache size in order to capture much of the local processing buffers. We chose the cache size based on the knee in the hit rate vs. cache size curve. Table IIIa reports these caches size.

To model the camera, modem and display, we create synthetic sequential streams of either read or write requests that scan through the frame buffers used by the encoder and decoder. This appropriately models the memory traffic to stream raw and encoded image data between the devices.

We obey all the timing and sequence constraints to reproduce the sequence of actions shown previously in Figure 2. This synchronizes the actions of the camera, encoder and modem on one end, and the modem, decoder, and display on the other end. We also synchronize the entire process to transmit and receive at the same frame rate. Our camera, modem and display models include parameters for the ratio of local to system clock, the maximum number of in-flight requests (Table IIIc).

TABLE II: Main Memory Parameters

Technology	DDR3 SDRAM (800 MHz)
Capacity	4 GB
Channels	1
Ranks	2 per channel, 16 Micron DDR3 [25] 32 Megabit, $\times 4$ DRAM devices
Banks	8 per rank
Rows	16,384 per bank
Columns	2,048 (groups of 8) 64B cache line per column
Data bus	64 bits
Bandwidth	Max. 11.92 GB/s

D. Memory system

Our goal is to model a system representative of future smartphone platforms which will use better memory technology. We use DDR3, with the parameters shown in Table II, as a proxy for the timing of future LPDDR4[2]. Current LPDDR3 is insufficient for some of the workloads and this is why existing platforms are not capable of supporting them. Moreover, LPDDR3 is optimized for power which is not the target of this work. Based on these parameters, our addressing schemes use one bit to specify the rank, three bits to specify the bank, fourteen bits to specify the row, and eight bits to specify the column. The least significant six bits are ignored since we assume that devices operate at a 64B granularity.

We examine two memory controller configurations, Limited and Large, as Table IIIb shows. Large assumes nearly unlimited resources for the memory scheduler (given the number of devices and the number of requests they can issue), while Limited assumes more realistically constrained resources. Studying two configurations allows us to: (1) Determine whether better scheduling decisions could improve performance, and (2) whether the workloads are memory- or compute-bound. Memory-bound workloads should benefit from the additional resources in the Large configuration, while compute-bound workloads should experience similar behavior for both Large and Limited configurations.

We use a single transaction queue per channel and a single command queue per rank. The write queue holds data from the moment a write request arrives at the controller until the scheduler issues that particular write. For the write drain mode, the write queue keeps write requests until the number of requests exceeds the high watermark and triggers the write drain mode. When in the write drain mode, only write requests are translated and issued until the number of requests falls below the low watermark.

We evaluate four memory scheduling policies: First-Ready First Come, First Served with and without the write drain (FR-FCFS-WD and FR-FCFS, respectively), Thread-Clustering

[2]Our configuration offers a bandwidth of 11.92GB/s while LPDDR4 is expected to provide 12.8GB/s/channel[5]

TABLE III: System Configurations

(a) Simulated system

Cores (x4)	in-order, 16KB+16KB L1s, 64KB L2 (priv.)
Additional devs.	Camera, modem, display, encoder, and decoder
Simulated clock	1600 MHz
Total outstanding memory operations	
Read	8
Write	16 (Limited) 128 (Large)

(b) Schedulers

	Limited	Large
Transaction queue entries	24	512
Command queue entries	8	512
TF scheduler CQ entries	20	1024
Maximum row accesses	32	1024
Write queue parameters		
Entries	16	64
Low Watermark	8	50
High Watermark	12	60

(c) Additional devices configuration

	Camera	Enc.	Modem	Dec.	Disp.
Clock (Mhz)	160	3200	800	3200	160
Cache size (KB)	-	256	-	128	-
Read max. ops.	16	8	16	8	16
Write max. ops.	16	16	16	16	16

(TCM) [20], and Thread-Fair (TF) [9]. FR-FCFS is a simple, greedy scheduling policy which selects the oldest command of all ready-to-issue commands in a given cycle. FR-FCFS treats reads and writes the same; however, writing data requires changing the direction of transfer on the data bus which incurs additional latency. FR-FCFS-WD attempts to avoid excessive switching by placing writes into a write queue and then draining writes only when the occupancy of that queue exceeds a threshold. TCM groups threads into two classes: *latency-sensitive* which issue infrequent memory requests and *bandwidth-sensitive* which issue large numbers of memory requests and have a high degree of memory level parallelism. *Latency-sensitive* threads are given higher priority. We replicate additional settings for the TCM memory scheduler as suggested by Kim *et al.* [20]. TF prioritizes reads over writes; read row hits are given the highest priority. If there are no row hits, the request that is at the head of the transaction buffer is given priority. Otherwise, requests are served oldest first. We implement TF without a transaction queue to allow issuing write hits while not in the write drain mode, or read hits while in write drain mode, as required by the scheduler.

IV. Evaluation

This section presents our experimental results. We initially restrict attention to the VCW workload as it places a high demand on memory and consists of a number of different threads. Section IV-A shows the impact of the address mapping schemes on the VCW performance. Section IV-B studies the performance of our multi-stream workloads under four memory scheduling policies. And finally, the effects of accelerating the computation are studied in Section IV-C.

Scheme	Address bits																			
KBCR	K	Bank	Column										Row							
RCBK	Row										Column						Bank		K	
RCKB	Row										Column					K	Bank			
KRCB	K	Row										Column							Bank	
KBRC	K	Bank	Row												Column					
RBKC	Row									Bank		K	Column							
RKBC	Row									K	Bank		Column							
XOR	Row									K	B	*	Column							
MOP	Row									Column [7:2]				K	Bank		C			

Fig. 3: Address mapping schemes showing which bits are used to specify the row (R), column (C), bank (B) and rank (K).

A. Address Mapping Schemes

The address mapping scheme refers to how logical addresses are assigned to physical memory locations. Functionally, any bit from the logical address can be used to determine the rank, bank, column or row. However, the address mapping can have a significant impact on the timing and scheduling of memory systems servicing realistic memory access patterns.

We consider the nine address mappings shown in Figure 3. We use an acronym for each scheme showing the order, starting from the most significant bit (MSB), in which address bits are used to specify the row (R), column (C), bank (B) and rank (K). The XOR scheme is adapted from the scheme proposed by Lin *et al.* [21] and the "*" filed indicates that the Bank[1:0] bits are calculated as the XOR of the "*" bits and the Row[1:0] bits. We also consider the scheme used as part of the minimalist open-page (MOP) policy proposed by Kaseridis *et al.* [18]. We model 4GB of memory and all requests operate at a granularity of 64 byte cache lines.

Generally, an address mapping scheme should balance rank and bank concurrency with row-buffer hit locality to provide the best overall performance. To reason about these trade offs it is useful to consider how the mapping scheme affects latency with a sequential access stream. In a sequential stream, the lower address bits change more frequently than the higher bits. Using the lowest bits to select the column will result in many accesses to the same rank, bank, and row. The first request in such a scenario will incur the row access latency and subsequent requests to sequential addresses will result in row buffer hits, resulting in lower latency. Conversely, if we use the lower bits for selecting the bank or rank then accesses can proceed concurrently across ranks and banks; however, adjacent accesses will suffer through a row activation latency.

Figure 4 shows the frame rate of VCW for all address mapping schemes, with Limited and Large memory controller configurations. KBCR is the worst performing scheme as it uses the lowest bits to select the row, resulting in frequent

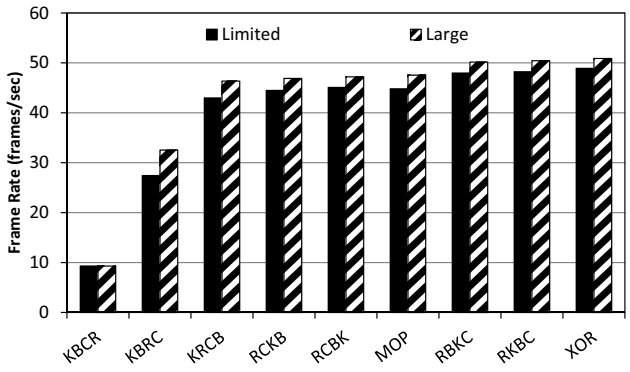


Fig. 4: Frame rate of VCW with different address mapping schemes for Limited and Large configurations (using FR-FCFS as scheduling policy)

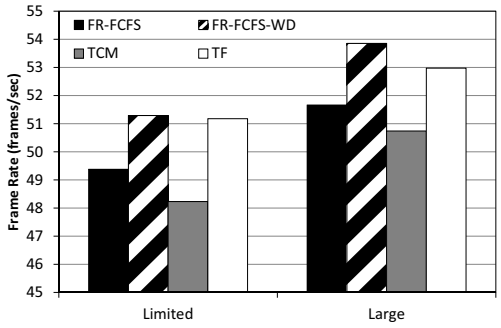
row conflicts. KBRC is the second worst performer since the row uses the second lowest set of bits while bank and rank use the highest bits. As a result, row switching still happens fairly frequently and little parallelism is exploited. The rest of the schemes use the highest bits for row, reducing row conflict frequency. RCBK and RCKB use the lowest bits for bank and rank, causing sequential accesses to be spread across banks and ranks increasing parallelism. However, this also increases the time between same-row accesses, increasing the chance that the row will be closed. RBKC, RKBC and XOR show the best performance because column bits are the lowest making sequential accesses hit in the same row. This shows that exploiting row locality is more beneficial than parallelism for vcw. Although XOR performs best, it is less than 1.5% faster than RKBC. The XOR scheme requires tuning to achieve good performance for different workloads. Accordingly, we use the simpler RKBC scheme for the rest of our analysis.

Comparing Large and Limited, the extra resources provided by Large improve performance by 2.5 frames/sec, suggests that better memory scheduling decisions can improve performance.

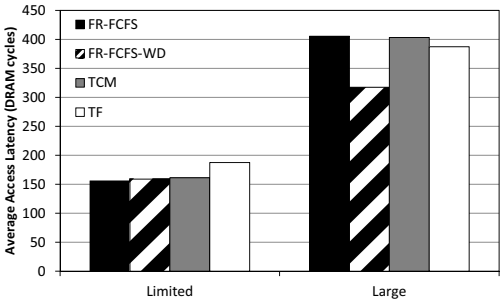
B. Memory Scheduler Policies

This section compares the performance of different memory scheduler policies for our multi-stream workloads. First we focus on VCW and then we annalyze the behavior of our combined workloads.

1) Video Conference Workload: Figure 5a shows the frame rate for all policies, for both Limited and Large configurations. As expected, the Large configuration outperforms the Limited configuration for all policies, which implies that VCW is memory bound. Within either configuration, the best scheduling policy (FR-FCFS-WD) is over 6% faster than the slowest policy (TCM). FR-FCFS-WD and TF both offer good performance, with FR-FCFS-WD being only 0.2% and 1.7% faster for the Limited and Large configurations respectively. TCM and TF offer benefits for modern desktop and server platforms, but the results in Figure 5a indicate that such designs need to be revisited and properly tuned before they can offer similar benefits for smartphone platforms.



(a) Frame rate



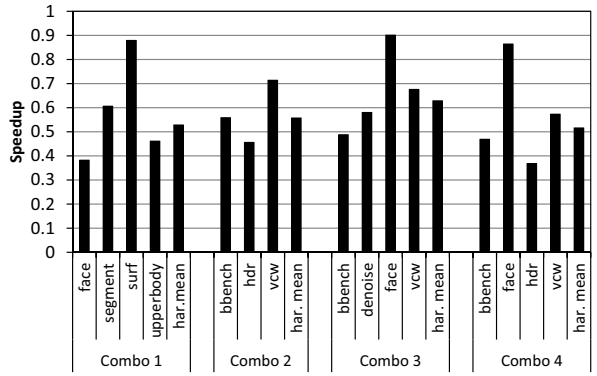
(b) Average latency

Fig. 5: DRAM performance for VCW w/ different memory schedulers.

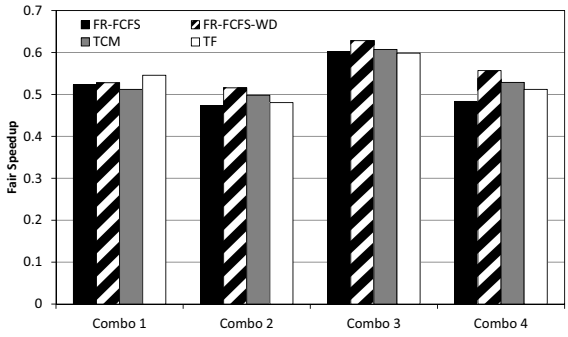
Figure 5b shows the average access latency with the four schedulers. Large results in higher access latency compared to Limited since it delays old requests in favor of new ones that hit in open rows. Given that performance is better with Large we conclude that VCW is memory-throughput bound. Comparing FR-FCFS with FR-FCFS-WD under Limited, we observe that write drain increases latency by 2%, while under Large write drain reduces latency by 22%. This is the result of the larger write buffers in Large which can delay writes longer and avoid frequent contention with reads. Although TF has frame rates almost as fast as FR-FCFS-WD, its average read latency is 18% to 22% higher. This suggests that TF is sacrificing latency to increase utilization and throughput, and since this results in good performance, it provides additional evidence that VCW is memory throughput limited. Further analysis of TF reveals that, on average, it has 2.6% to 8% fewer row buffer hits than FR-FCFS-WD, which likely results in higher power and energy consumption. However, an investigation of power and energy is beyond the scope this work.

2) Combined Workloads: Figure 6a shows the slowdown experienced by individual benchmarks when run in a combined workload. Since the individual applications have very different execution times, the longest running application in each combination suffers much less slowdown than the others. This is explains the results for surf in Combo 1 and face in Combo 3 and Combo 4.

The metric used to study the performance of our combined workloads is the *fair speedup* or *harmonic mean of speedups* [32]. This metric is the harmonic mean of the per-



(a) Normalized speedup of individual benchmarks with FR-FCFS-WD when run as part of combined workload

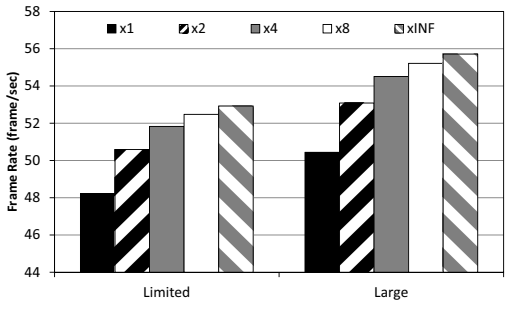


(b) Fair speedup of combined workloads w/ different memory schedulers.

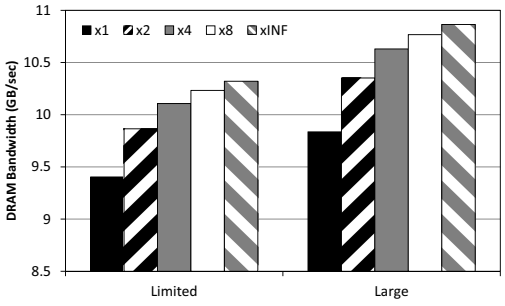
Fig. 6: Combined workloads with Limited configuration and compute ratio of one.

application speedups running the applications in the combo with respect to running the application alone in the system. Figure 6b shows the fair speedup for the combined workloads when using different memory schedulers. On average, FR-FCFS-WD offers the best performance, with a fair speedup 7.5% higher than the worst policy (FR-FCFS), and 3.9% higher than the second best policy (TCM). Note that while TCM has the worst performance for VCW running alone (Section IV-B), it is the second best policy for the combined workloads. But the simpler FR-FCFS-WD still performs better. Again, memory schedulers designed for highly threaded desktop and server systems might need to be re-examined and tuned for smartphone workloads.

For Combo 2, 3 and 4, FR-FCFS-WD and TCM improve the fairness by speeding up the slowest applications in the combinations, hdr and denoise by $28-43\%$. These speedups occur at the expense of small slowdowns in bbench and VCW of $5-16\%$. As this shows, it can be difficult to characterize the performance of different schedulers for such multi-threaded workloads. A user might have priorities and preferences other than simple fairness; future memory scheduler research should examine how tradeoffs between applications can be adjusted to best meet the users needs, instead of focusing on total throughput or fairness.



(a) Frame rate



(b) Bandwidth

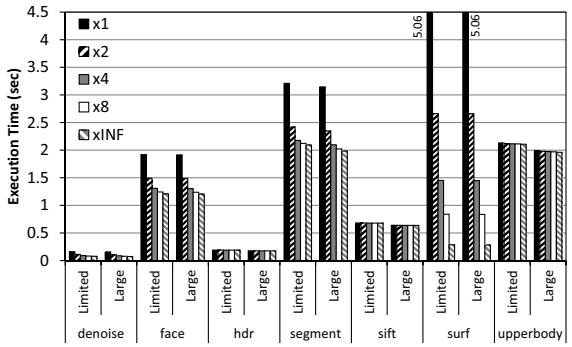
Fig. 7: Performance and bandwidth of VCW with varying computation ratios for Limited and Large configurations.

C. Effect of Faster Compute Engines

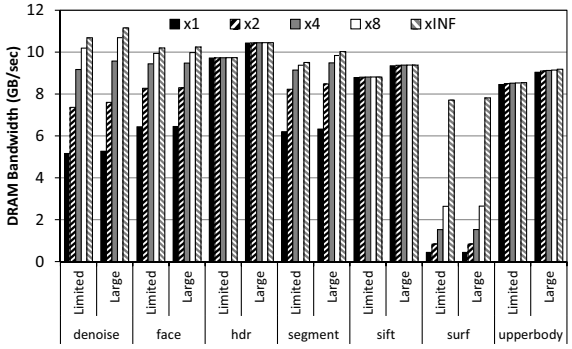
Through either faster or specialized processing units, it may be possible to accelerate the computation part of a workload. This section studies the effect of such acceleration on memory system performance. We model the effects of acceleration using the computational distance which is the number of instructions between dependent memory operations.

Figures 7, 8 and 9 show the effect on execution for all individual workloads when varying the *computation ratio*. The computation ratio models processing engines that can process the computation at a faster rate than one instruction per cycle. Specifically, a computation ratio of N represents a compute engine that can execute N independent instructions per cycle. An ideal, unrealistic accelerator that can perform any computation in zero cycles would have infinite an computation ratio.

1) Video Conference Workload: As Figure 7 shows, using an ideal accelerator improves performance by 10% for VCW while the Large configuration results in performance improvements of 5% on average. These results suggest that parts of VCW are compute bound while others are memory bound. Figure 7b shows that VCW is relatively close to the theoretical bandwidth limit of the system (11.92 GB/s), suggesting that VCW is bound by available memory throughput and not by memory latency. This result highlights the importance of memory controller design; a custom accelerator would be limited by memory performance.



(a) Execution Time



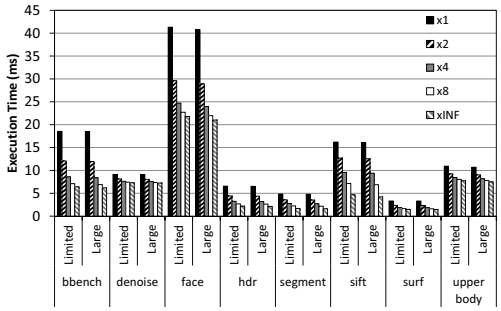
(b) Bandwidth

Fig. 8: Performance and bandwidth of computer vision workloads with **high resolution images** with varying computation ratios for Limited and Large configurations.

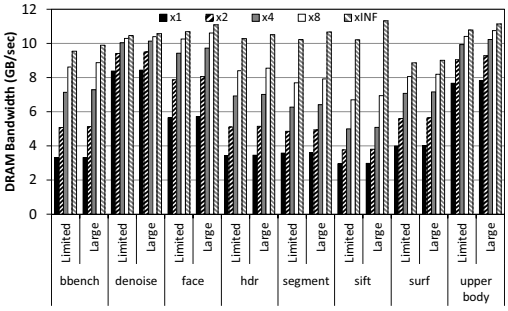
2) Computer Vision Workloads: For the computer vision workloads with high resolution images, Figure 8 shows that there are two classes of workloads When increasing the computation ratio, the performance of hdr, sift and upperbody is not affected. This suggests that these benchmarks are completely memory bound. The other applications exhibit performance improvements with increasing computation ratios. However, in most cases these benefits quickly plateau, suggesting that the workloads are mostly memory bound when they have a computation ratio of eight. However, surf is almost 3× slower with a computation ratio of eight compared to an infinite computation ratio, suggesting that it is still compute bound at this point.

The compute-bound nature of some workloads is further highlighted by looking at their bandwidth consumption. As Figure 8b shows, surf has much lower bandwidth consumption than the other workloads, emphasizing that computation is the bottleneck and not memory. Even with an infinite computation ratio, surf still uses less bandwidth than the other workloads suggesting that its performance has become bound by memory latency rather than memory throughput.

With sufficiently high computation ratios, the other benchmarks all become memory throughput limited since their bandwidth is relatively close to the maximum available bandwidth in the system. For most of these workloads, the increased flexibility available with Large results in higher bandwidth usage and better performance. This encourages work on novel memory controllers that can exploit this potential without using impractically large structures. However, face and surf are the two exceptions that benefit little from Large and are less likely to benefit from memory controller advances.



(a) Execution Time



(b) Bandwidth

Fig. 9: Evaluation of the web browsing and computer vision workloads with **low-resolution images**.

Figure 9 shows that the behavior of the computer vision workloads changes drastically for the low-resolution images; almost all of the workloads become compute bound. It also shows that bbench is initially compute bound. Furthermore, as the computation ratio increases, more benchmarks stay compute bound with a compute ratio of eight, including hdr, segment, and sift.

V. Related Work

This work examines the performance of memory subsystem for smartphone workloads. We focused on policies recently proposed for chip multiprocessors: Thread Clustering [20] and Thread Fair [9], which target chip-multiprocessors. In this section, we discuss other relevant memory controller policies, literature targeting smartphone characterization and benchmarking.

A. Memory Controller Policies

We categorize the memory scheduling policies into two groups based on their primary optimization metric: performance or power/energy. Performance-oriented schedulers either focus on the performance of the memory subsystem or the applications. The memory subsystem performance is often

measured as bandwidth utilization or efficiency. While this value is ought to be high, it does not necessarily reflect the performance of the applications, which might find some memory requests more useful than others. Ishii *et al.* divide thread execution into phases; memory- and non-memory intensive [16]. Non-memory intensive phases are prioritized over memory intensive ones. In addition, while refreshing one rank, write operations are executed on the opposing rank, which aims to reduce the interference of writes and reads. Some schedulers are designed for multi-threaded environments and often target fairness [27], [28] or system throughput [19], [15].

Power and energy consumption are becoming increasingly important in all modern systems. Energy usage is often tightly coupled with improving performance, whereas power considerations are often harmful to the application performance. Memory throttling has been proposed to restrict the maximum number of accesses within a fixed time window [14], [12]. Deng *et al.* propose reducing the frequency of the DIMM modules to limit power [6]. The scheduler detects the system's tolerance to diminished performance, *slack*, and adjusts the frequency accordingly.

B. Benchmarks

The San Diego Vision Benchmark Suite (SD-VBS) includes computer vision workloads [34]. The benchmark suite includes Disparity Map, Feature Tracking, Image Segmentation, SIFT, Robot Localization, SVM, Face Detection, Image Stitching and Texture Synthesis. Their experimental analysis looks at the execution time breakdown of each program in terms of the major kernels and how they scale with image size along with estimating the potential parallelism in each workload.

MEVBench is a Mobile Computer Vision Benchmarking Suite [4]. MEVBench uses some benchmarks from SD-VBS and adds SURF, HOG, FAST, BRIEF, Adaboost, K Nearest Neighbors, Object Recognition and Augmented Reality. For some benchmarks both single and multithreaded applications are used. They use both physical and simulated systems for both mobile and desktop processors. Analysis includes IPC, memory activity, and multithreaded and SSE speedup. The workloads we study partially overlap with the workloads studied in the aforementioned works. MobileBench includes a broad range of real-world smart phone applications [29]. This suite covers general and realistic web browsing by using the publicly available BBench, education web browsing, photo viewing, and video playback. However, our work focuses on memory system behavior and thus complements these existing studies. However, our work focuses on memory system behavior and thus complements these existing studies.

C. Smartphone Characterization

To date, there have been few studies characterizing smartphone workloads. Recent work analyzes branch prediction, cache behavior, TLBs and overall system performance, for a set of audio, video, interactive gaming and web-browsing benchmarks targeting smartphones [11]. We complement their work by focusing on memory system performance and incorporate some different workloads.

Duan *et al.* explore memory energy optimizations in smartphones [7]. They introduce a hybrid memory organization combining RAM and Phase-change Memory (PCM). Huang *et al.* examine overall workload performance, taking into account telecom carriers transmission parameters [13], such as packet retransmission and 3G network delays. Wang *et al.* analyze web-browser behavior on smartphones and correlate it with network round-trip time and OS services [37].

Pandiyan et al. perform energy characterization of the MobileBench benchmark suite by instrumenting the Android framework on Galaxy S III smart phone[29]. Then, by using simulation they study how different microarchitectural aspects affect the performance of applications running in a smart phone. Specifically, they focus on the implications of improving branch prediction, TLB management, and L2 cache performance (by both improving its replacement and prefetching) Qian et al. discover inter-layer inefficiencies by profiling Android platforms through an innovative cross-layer approach [30]. Murmuria et al. present a methodology to measure and model power usage on smartphones. To do so they use the operating system?s power management module [26].

VI. Conclusions

We have developed a set of smartphone workloads and studied how they stress main memory. These workloads include a high definition video conference workload as well as a range of computational photography and computer vision workloads. To model realistic smartphone usage scenarios, we combine various applications to model a photo cataloguing system and various advanced video conference scenarios that incorporate image processing and web browsing activity. Many of these workloads are memory throughput bound; they strain existing memory systems. When processing small images, many of these algorithms become compute limited; our results suggest that more specialized compute engines may provide enough compute performance to make them memory bound.

Our evaluation shows that the address mapping scheme used by memory must balance row buffer locality and concurrency across banks and ranks. Carefully selecting the address mapping scheme can improve performance by as much as 10% over a reasonable but unbalanced mapping scheme. Memory scheduling policies impact performance; recently proposed policies that improve the performance of highly-threaded desktop and server workloads need to be revisited and tuned in order to provide similar benefits for smartphone workloads. For VCW, a simple FR-FCFS-WD policy provides the best performance and reduces average read latency by up to 22% compared to other schedulers. With our combined workloads, FR-FCFS-WD provides the best fair speedup. However, our analysis suggests there is much potential for work that better balances the performance of individual applications with the priorities of the smartphone user.

References

[1] FFmpeg multimedia framework, December 2001.

[2] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[3] A. Buades, B. Coll, and J. M. Morel. A review of image denoising algorithms, with a new one. *Multiscale Modeling and Simulation*, 4:490–530, 2005.

[4] J. Clemons, H. Zhu, S. Savarese, and T. Austin. Mevbench: A mobile computer vision benchmarking suite. In *Workload Characterization (IISWC), 2011 IEEE International Symposium on*, nov. 2011.

[5] D. Skinner (Micron Technology). Lpddr4 moves mobile. In *In JEDEC Mobile Forum Conference, 2013.*

[6] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini. MemScale: active low-power modes for main memory. In *Proc. of the international conference on Architectural Support for Programming Languages and Operating Systems*, 2011.

[7] R. Duan, M. Bi, and C. Gniady. Exploring memory energy optimizations in smartphones. In *Proceedings of the 2011 International Green Computing Conference and Workshops*, IGCC '11, 2011.

[8] M. Eichner, M. Marin-Jimenez, A. Zisserman, and V. Ferrari. 2d articulated human pose estimation and retrieval in (almost) unconstrained still images. *Int. J. Comput. Vision*, 99(2):190–214, Sept. 2012.

[9] K. Fang, N. Iliev, E. Noohi, S. Zhang, and Z. Zhu. Thread-fair memory request reordering. In *3rd JILP Workshop on Computer Architecture Competitions: Memory Scheduling Championship*, MSC, 2012.

[10] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *Int. J. Comput. Vision*, 59(2):167–181, Sept. 2004.

[11] A. Gutierrez, R. G. Dreslinski, T. F. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver. Full-system analysis and characterization of interactive smartphone applications. In *IEEE International Symposium on Workload Characterization*, pages 81–90, 2011.

[12] H. Hanson and K. Rajamani. What computer architects need to know about memory throttling. In *Computer Architecture*, volume 6161 of *Lecture Notes in Computer Science*, pages 233–242, 2012.

[13] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl. Anatomizing application performance differences on smartphones. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, pages 165–178, 2010.

[14] I. Hur and C. Lin. A comprehensive approach to DRAM power management. In *Proc, of International Symposium on High Performance Computer Architecture*, pages 305 –316, feb. 2008.

[15] T. Ikeda, S. Takamaeda-Yamazaki, N. Fujieda, S. Sato, and K. Kise. Request density aware fair memory scheduling. In *3rd JILP Workshop on Computer Architecture Competitions: Memory Scheduling Championship*, MSC, 2012.

[16] Y. Ishii, K. Hosokawa, M. Inaba, and K. Hiraki. High performance memory access scheduling using compute-phase prediction and writeback-refresh overlap. In *3rd JILP Workshop on Computer Architecture Competitions: Memory Scheduling Championship*, MSC, 2012.

[17] A. Jironkin, A. Bhatt, G. Caulier, and M. Wiesweg. libface library, May 2012.

[18] D. Kaseridis, J. Stuecheli, and L. K. John. Minimalist open-page: a DRAM page-mode scheduling policy for the many-core era. In *Proc. of the International Symposium on Microarchitecture*, 2011.

[19] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter. ATLAS: A scalable and high-performance scheduling algorithm for multiple memory controllers. In *HPCA*, pages 1–12, 2010.

[20] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter. Thread cluster memory scheduling: Exploiting differences in memory access behavior. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '43, 2010.

[21] W.-F. Lin, S. Reinhardt, and D. Burger. Reducing DRAM latencies with an integrated memory hierarchy design. In *High-Performance Computer Architecture (HPCA)*, 2001.

[22] D. G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, 1999.

[23] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood. Pin: building customized program analysis tools with dynamic instrumentation. In *The 2005 ACM SIGPLAN conference on Programming language design and implementation*, pages 190–200, 2005.

[24] B. Meyer and J. Wieczorek. Arora lightweight webkit-based web browser, August 2012.

[25] Micron. 1Gb: $\times 4$, $\times 8$, $\times 16$ DDR3 SDRAM features, 2007.

[26] R. Murmuria, J. Medsger, A. Stavrou, and J. Voas. Mobile application and device power usage measurement. *SERE*, 2012.

[27] O. Mutlu and T. Moscibroda. Stall-time fair memory access scheduling for chip multiprocessors. In *Proceedings of the 40th Annual IEEE/ACM Int'l Symposium on Microarchitecture*, MICRO 40, 2007.

[28] O. Mutlu and T. Moscibroda. Parallelism-aware batch scheduling: Enhancing both performance and fairness of shared dram systems. In *Proce. of the International Symposium on Computer Architecture*, pages 63–74, 2008.

[29] D. Pandiyan, S.-Y. Lee, and C.-J. Wu. Performance, energy characterizations and architectural implications of an emerging mobile platform benchmark suite - mobilebench. In *IISWC*, pages 133–142. IEEE, 2013.

[30] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Profiling resource usage for mobile applications: A cross-layer approach. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 321–334, New York, NY, USA, 2011. ACM.

[31] M. A. Robertson, S. Borman, and R. L. Stevenson. Dynamic range improvement through multiple exposures. In *The International Conference on Image Processing*, ICIP99, pages 159–163. IEEE, 1999.

[32] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreaded processor. In *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS IX, pages 234–244, New York, NY, USA, 2000. ACM.

[33] N. Trevet, Khronos Group. SOC Programming Tutorial. In *HOT Chips Conference*. 2012.

[34] S. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. Taylor. Sd-vbs: The san diego vision benchmark suite. In *IEEE Int'l Symposium on Workload Characterization (IISWC)*, 2009.

[35] VideoLAN Organisation. x264 software library, February 2012.

[36] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition*, CVPR'01, pages 511–518, 2001.

[37] Z. Wang, F. X. Lin, L. Zhong, and M. Chishtie. Why are web browsers slow on smartphones? In *Proc. of the 12th Workshop on Mobile Computing Systems and Applications*, HotMobile '11, 2011.

[38] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra. Overview of the H.264/AVC video coding standard. *Circuits and Systems for Video Technology, IEEE Transactions on*, 13(7):560 –576, july 2003.