GPU Sharing for Image Processing in Embedded Real-Time Systems*

Nathan Otterness[1], Vance Miller[1], Ming Yang[1], James H. Anderson[1], F. Donelson Smith[1], and Shige Wang[2]
[1]Department of Computer Science, University of North Carolina at Chapel Hill
[2]General Motors Research

Abstract

To more efficiently utilize graphics processing units (GPUs) when supporting real-time workloads, it may be beneficial to allow multiple tasks to issue GPU computations without blocking one another. For such an option to be viable, it is necessary to know the extent to which concurrent GPU computations interfere with each other when accessing hardware resources. In this paper, measurement data is presented regarding such interference for several image processing routines motivated by automotive use cases. These measurements were taken on NVIDIA Jetson TK1 and TX1 boards. The presented data suggests that currently available real-time GPU management frameworks should evolve to enable the option of co-scheduling GPU computations.

1 Introduction

Vision-based sensing through cameras is being widely used in automobiles today to support advanced driver assistance systems (ADASs). Common capabilities of current ADASs include forward collision detection with automatic braking, lane departure warnings, and adaptive cruise control. Envisioned capabilities include advanced obstacle-tracking features, sign recognition, and 360-degree sensing.

Such capabilities give rise to workloads that can be challenging to support for three reasons. First, individual tasks may be subject to real-time constraints. Second, such tasks may be computationally intensive. Third, the overall workload must be supported on a hardware platform that operates within an acceptable size, weight, and power (SWaP) envelope and also is not too expensive.[1] In light of these needs, multicore+GPU platforms have been suggested as a promising way forward. Such a platform consists of several general-purpose CPUs augmented with one or more graphics processing units (GPUs) that can accelerate computations typically required in automotive settings.

Prior foundational work: GPUSync. Unfortunately, efficiently utilizing GPUs in contexts where real-time constraints exist requires sifting through many tradeoffs involving how GPUs are allocated at runtime and how GPU computations and related overheads are analyzed when checking real-time schedulability. To enable such tradeoffs to be systematically studied, our research group developed a real-time GPU allocation framework called GPUSync [15]. In GPUSync, the management of GPU-related hardware resources is viewed as a *synchronization* problem and thus real-time multiprocessor locking protocols are used to acquire and release such resources. GPUSync is highly configurable: options exist to control how tasks are scheduled on CPUs, how data is copied to and from GPUs, how GPU-related computations are queued and prioritized, *etc.*

Beyond GPUSync. In recent work, we have been attempting to evolve our work on GPUSync to more directly meet the needs of automotive use cases. The consideration of such use cases has caused the nature of our work to change in two significant ways. First, GPUSync is implemented primarily in LITMUS$^{\mathrm{RT}}$, and the code base is large, approximately 15,000 lines. Automotive manufacturers would likely be highly resistant to allowing such extensive operating system (OS) modifications. Due to this, we have shifted our attention to a simplified variant of GPUSync called GPUSyncLite that implements only a few GPUSync configurations (one currently) and requires only minimal OS modifications (none currently). Second, our prior GPUSync-related experimental work was conducted on an Intel platform that provides 12 CPU cores augmented with eight high-end GPUs. At present, it is hard to imagine such an expensive, energy-hungry platform being used in a production automobile. As a result, we have shifted our attention to less-expensive ARM-based platforms that provide a single less-costly, less-capable GPU.

Efficient GPU utilization through co-scheduling. This shift in hardware platform has created a new dilemma: when using a single, less-capable GPU, any waste of the GPU's capacity becomes untenable. Unfortunately, when using most previously proposed real-time GPU management frameworks [7, 8, 9, 12, 16, 25, 26, 49, 47, 48, 54, 55], including GPUSync, such under-utilization may be common. In particular, these frameworks disallow concurrent GPU execution by different tasks, so a task that under-utilizes the GPU's hardware resources can waste much of its capacity. Other prior work [10, 11] has considered co-scheduling GPU workloads, but in this work, several simplifying assumptions are made that preclude applicability on real-world GPUs. Notably, GPU instructions are assumed to always require only

*Work supported by NSF grants CPS 1239135, CNS 1409175, and CPS 1446631, AFOSR grant FA9550-14-1-0161, ARO grant W911NF-14-1-0499, and funding from General Motors.

[1]In contrast to various "one-off" implementations of autonomous or semi-autonomous features, as seen for example in the Google car [1] and various DARPA challenge vehicles [45], affordability is a serious limitation with respect to production automobiles.

a single clock cycle, and cache misses and memory latency are not considered. Furthermore, this prior work includes no evaluation using real hardware.

To combat GPU under-utilization, we are beginning to investigate a new variant of GPUSyncLite that allows GPU computations issued by different tasks to be concurrently co-scheduled. When considering multi-threaded workloads scheduled on conventional multicore platforms, Jain *et al.* [22] observed that some co-scheduling choices are constructive and some are destructive. This is true in our context as well. In particular, it is *constructive* to co-schedule GPU computations issued by different tasks if the resulting GPU execution times and blocking times (*i.e.*, times spent waiting to access a GPU) yield real-time schedulability improvements. In contrast, such co-scheduling is clearly *destructive* if it causes a large inflation in GPU execution times or blocking times. Any such inflation is a sign that the co-scheduled GPU computations are adversely interfering with each other with respect to the hardware resources they access.

Contributions of this paper. To get a sense of the nature of such interference, we conducted experiments involving several common image-processing routines motivated by automotive use cases. For each of the considered routines, we obtained execution-time data via a measurement process under various co-scheduling scenarios. These measurements were taken on NVIDIA Jetson TK1 and TX1 boards. The obtained data suggests that certain co-scheduling choices are indeed constructive, while others are clearly destructive. The main contribution of this paper lies in presenting this data and discussing its implications as far as the future evolution of real-time GPU management frameworks is concerned.

Organization. In the rest of the paper, we provide needed background on GPUs (Sec. 2), describe the image-processing benchmarks under consideration (Sec. 3), present our experimental data (Sec. 4), and conclude (Sec. 5).

2 Background on GPUs

In this section, we provide a brief introduction to GPU hardware and programming fundamentals.

GPU hardware. GPUs may be either discrete or integrated. *Discrete GPUs* are packaged on adapter cards that plug into a host computer bus. Such a GPU has its own local DRAM memory that is completely independent from the DRAM memory used by the host processor. Discrete GPUs commonly draw between 150 and 250 watts, need active cooling, and occupy substantial space. *Integrated GPUs* are commonly found in system-on-chip (SOC) designs. The SOC typically combines a multicore machine with a GPU and uses DRAM memory that is tightly shared between the GPU and CPU cores. Integrated GPUs commonly draw between 5 and 15 watts, require minimal cooling, and add virtually no space requirements. These attributes make integrated GPUs the *de facto* choice in many embedded computing domains.

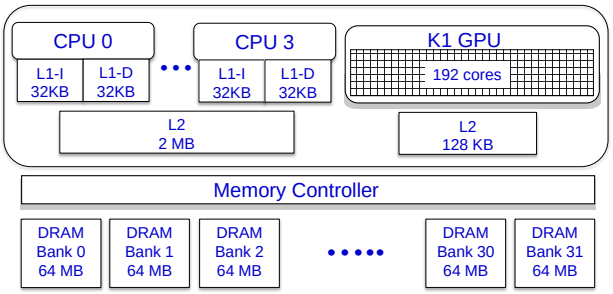

Figure 1: Jetson TK1 architecture.

Several SOC implementations with integrated GPUs capable of running sophisticated image-processing programs are on the market, including options from AMD [5], Intel [21], NXP [41] and NVIDIA [38]. In this work, we are using NVIDIA Jetson TK1 [39] and TX1 [40] boards, which retail for $200 and $600, respectively. These are likely acceptable price points in many automotive settings.

As illustrated in Fig. 1, the TK1 employs an SOC design that incorporates a quad-core 2.32 GHz 32-bit ARM machine and an integrated Kepler GK20a GPU. The CPUs share a 2-MB L2 cache. The GPU has 192 cores and a 128-KB L2 cache and provides up to 365 32-bit GFLOPS. The TK1 is a "big-little" platform in which an additional low power, low performance ARM CPU (not shown in Fig. 1) is provided on chip. The ARM CPUs and the GPU share 2 GB of 930 MHz DRAM memory partitioned into 32 banks.

The TX1 is a higher-end platform with a similar design. It consists of a quad-core 1.91 GHz 64-bit ARM machine, a 2-MB L2 cache shared by all CPUs, 4 GB of 1600 MHz DRAM, and an integrated Maxwell GM20B GPU. The GPU has 256 cores and a 256-KB L2 cache, and provides up to 512 32-bit GFLOPS. The TX1 is also a "big-little" platform.

As Fig. 1 suggests, GPU-using tasks may compete for many hardware resources. These resources include caches, DRAM memory banks, the memory bus and memory controller, and GPU cores. In prior work on real-time multicore computing, issues related to shared-hardware interference have received considerable attention [2, 3, 4, 6, 13, 14, 17, 18, 19, 20, 23, 27, 29, 28, 30, 31, 33, 35, 42, 44, 46, 50, 51, 52, 53]. However, we are aware of no such work that considers hardware interference with respect to GPU computations.

Obviously, concurrent GPU computations by different tasks may directly interfere with each other. Additionally, such computations can also interfere with programs running on CPU cores. For example, on both the TK1 and TX1, requests to load new lines into the GPU's L2 cache require accesses to the DRAM banks and may interfere with accesses by CPU cores. Further, so that GPU programs may be easily ported between discrete and integrated GPUs, CUDA (see below) explicitly treats memory as being either CPU-local (host memory) or GPU-local (device memory) and provides operations for copying data between the two. Such copy op-

erations run concurrently with programs running on both the GPU cores and the CPU cores, potentially creating additional DRAM interference.[2] With integrated GPUs, explicit data copying can be avoided by using the *zero-copy* functions of CUDA (see below).

GPU programming in CUDA. The following is a high-level description of GPU programming in CUDA [37]. A GPU is fundamentally a co-processor that performs operations requested by CPU programs. CUDA programs use a set of C or C++ library routines to request GPU operations that are implemented by a combination of hardware and device-driver software. The typical structure of a CUDA program is as follows: **(i)** allocate GPU-local (device) memory for data; **(ii)** use the GPU to copy data from host memory to GPU device memory; **(iii)** launch a program—called a *kernel*—to run on the GPU cores to compute some function on the data; **(iv)** use the GPU to copy output data from the device memory back to the host memory; **(v)** free the device memory. On integrated GPUs, CUDA provides a zero-copy option where programs can simply pass a pointer to shared memory where data used for a kernel is located—that is, explicit copying from CPU-local memory to GPU-local memory is avoided.

By default, copy operations are synchronous with respect to the CPU program: they do not return until the copy is complete and will not start until any prior kernels have finished. However, kernel launches are always asynchronous, and asynchronous copy operations are also available. These operations require the CPU process to explicitly wait for GPU operations to complete, using a configurable synchronization mechanism. We configured our experiments to block the CPU process while synchronizing.

CUDA operations pertaining to a given GPU are ordered by associating them with a *stream*. By default, there is a single stream for all programs that share a GPU, but multiple streams can be optionally created. Operations in a given stream are executed in FIFO order, but the order of execution across different streams is determined by the GPU scheduling in the device driver. They may execute concurrently (or out of request order with respect to other streams).

Each GPU operation from a CUDA program is represented internally by a command string that is written to a command buffer (queue) managed by the device driver. The driver then schedules these commands for execution on the GPU. Programmers can think of a GPU as being abstractly composed of one or more *copy engines (CEs)* that implement transfers of data between device memory and host memory, and an *execution engine (EE)* that executes GPU kernels. Both the TK1 and TX1 have one CE that moves data both ways.

EEs and CEs operate concurrently. When there are multiple streams, multiple kernels and one or two copy operations can operate concurrently depending on the GPU hardware. When a kernel is scheduled, it may not require all EE resources, in which case the GPU scheduler may co-schedule more than one kernel (from different streams only) to execute concurrently and increase GPU occupancy. Concurrent kernel execution can create more interference in the GPU L2 cache and for DRAM accesses. To the best of our knowledge, complete details of kernel attributes and policies used by NVIDIA to co-schedule kernels are not available.

[2] With discrete GPUs, only the GPU data-copy operations may cause DRAM interference with respect to CPU usage and then typically in the form of DMA operations over a bus.

3 Benchmark Programs

In the study presented herein, we considered both GPU programs and CPU-only benchmark programs.

GPU programs. We chose three CUDA programs as representative of typical image-processing computations, and a fourth to represent a general class of programs that create stress on GPU resources:

- **stereoDisparity (SD)**: Extracts 3D depth information from 2D images taken with a stereo camera. The input consists of left and right 640×533 color images; the output is a 640×533 grayscale image.
- **fastHOG (HOG)**: Detects objects in an image using histograms of oriented gradients. The input is a 640×480 color image; the output is a matrix of bounding-box coordinates and object-detection probabilities.
- **Convbench (CONV)**: Executes convolutional neural-network layers as used in image recognition. The input is a 227×227 color image; the output is a matrix of neural-network parameters.
- **matrixMul (MMUL)**: Multiplies two square matrices of 32-bit floats (16 MB each).

SD and MMUL were taken from CUDA samples distributed by NVIDIA [36], HOG was downloaded from Oxford University [43], and CONV was constructed using code from AlexNet [32], implemented in Caffe [24]. All programs were adapted to run as iterative tasks, with a short random sleep between iterations. Each iteration corresponds to processing one image (SD, HOG, and CONV) or performing one matrix multiplication (MMUL). The programs were instrumented to log total execution time and the time required for performing data copies and executing kernels in every iteration. Even though our experiments were conducted using fixed images as inputs, we still verified that none of the benchmarks exhibited different runtime characteristics based on the content of the input images.

Each CUDA program was executed in a stream of its own, with all memory copies performed asynchronously and placed in the stream along with kernel launches in the intended FIFO order. After each kernel launch or group of memory copies, the CPU execution of each program was blocked while waiting to synchronize with the GPU. Each program was structured to ensure that all memory allocation

and freeing operations were done outside the iteration loop and all memory accesses within each iteration were to pinned memory, as is common practice in real-time systems. Display operations for the visualization of input or output images were removed. Image input data was read from memory buffers as would happen with camera-driven input. Two versions of each program were constructed, one with zero-copy memory and one without.

CPU-only benchmark. We used this program as a CPU-only workload:

- **vectorAdd (VADD)**: Adds two vectors of 32-bit floats (16 MB each).

VADD was based on the CUDA samples [36] and instrumented in an identical fashion as the GPU programs, but launches no GPU kernels.

4 Experiments

We are interested in supporting automotive image-processing workloads on a multicore+GPU platform such as the TK1 or the TX1. We assume that such workloads have soft real-time constraints: missing a deadline (occasionally) does not have catastrophic consequences, as long as an incomplete frame can be dropped and the system as a whole can use redundant or historical data processed by hard real-time components as a fail-safe mechanism. Given this assumption, our tasks can be provisioned by determining their execution times via measurement. Such a provisioning could be based on a task's average-case execution time, its worst-case execution time, or some intermediate value between the two. A measurement-based approach is further justified by the lack of adequate static timing analysis tools for multicore+GPU platforms. Even if such tools did exist, they would probably produce execution-time estimates that are so pessimistic that virtually no interesting workload could be supported.

The issue being considered in this paper is whether allowing GPU co-scheduling might have schedulability benefits. To get a sense of any potential benefits, we conducted experiments on both the TK1 and TX1 in which the various benchmark programs described in Sec. 3 were used as surrogates for real application code. These experiments were designed to assess whether GPU co-scheduling can be constructive from a schedulability point of view. We assessed this by running different combinations of the benchmark programs and recording execution-time data. We call each experiment involving such a combination of programs a *scenario*. In each scenario, execution-time data was recorded for a set amount of time (typically 10–15 minutes) under the default Linux scheduler with the considered programs pinned to separate CPUs. We present our obtained execution-time data by plotting cumulative distribution functions (CDFs), as such functions provide a sense of the best-case, average-case, and worst-case recorded times. We denote a given scenario by simply listing the combination of programs that were run.

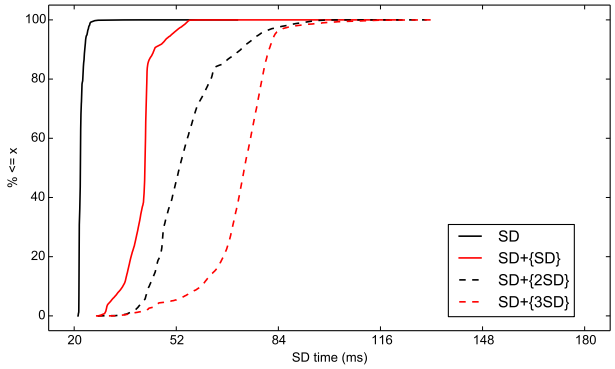

Figure 2: CDF of execution times of SD in scenarios only involving multiple SD instances.

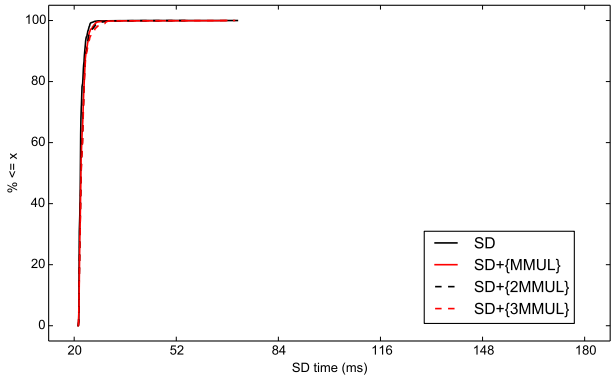

Figure 3: CDF of execution times of SD in scenarios involving MMUL competitors.

For example, in the scenario HOG+{2SD,HOG}, execution-time data was obtained on one CPU for the HOG program in the presence of two instances of SD and another instance of HOG running on the other three CPUs.

In total, we tested 52 scenarios, each both with and without the zero copy feature of CUDA and on both the TK1 and TX1. Unless otherwise noted, the scenarios presented here were measured on the TK1 and did not use the zero-copy feature. Data for all considered scenarios can be found in the appendix.

Typical observed trends. We begin by commenting on general trends seen in our collected data.

Obs. 1. GPU co-scheduling was always constructive in scenarios consisting of multiple instances of a single benchmark.

Fig. 2 supports this observation for the case of the SD benchmark. In this case, GPU co-scheduling is mildly constructive. While SD execution times do increase with more competition, they do not increase to the point of eliminating any benefit due to co-scheduling. In particular, the addition of one competitor yields execution times that are somewhat better than simply doubling the execution time of a single instance, and this trend continues to apply as more competition is introduced.

Obs. 2. GPU co-scheduling was so constructive in some

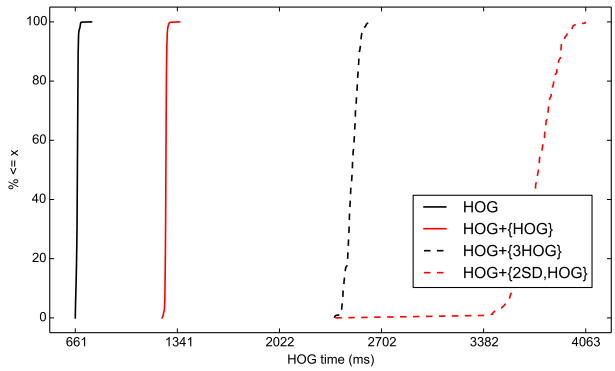

Figure 4: CDF of execution times of HOG in scenarios involving multiple instances of other benchmarks.

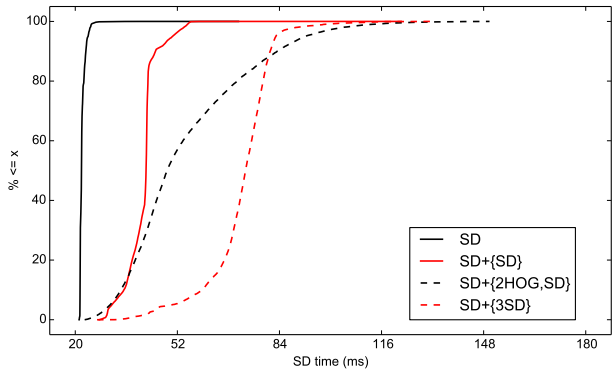

Figure 5: CDF of execution times of SD in scenarios involving multiple instances of other benchmarks.

scenarios that any introduced interference was practically negligible.

Fig. 3 supports this observation. Note that the execution times for SD remain virtually unaffected when instances of MMUL are introduced. This low impact is probably due to MMUL having short kernel execution times (approx. 1ms), which would rarely prevent SD from accessing the GPU.

Obs. 3. In some scenarios, particularly those involving HOG, GPU co-scheduling proved to be rather destructive.

Fig. 4 supports this observation. Note that the most destructive interference occurs when two instances of HOG and two instances of SD are run together, given by the curve for the scenario HOG+{2SD,HOG}. Fig. 5 presents execution-time data for SD that allows us to examine this same scenario from the perspective of SD. In particular, note the curve labeled SD+{2HOG,SD} in Fig. 5.

In our TK1 experiments, the worst-case execution time of SD running in isolation was 71.1ms, and the worst-case execution time of HOG running in isolation was 768.9 ms. However, the *median* execution time of HOG in the HOG+{2SD,HOG} scenario was 3747.0ms. Had the scheduler simply treated the GPU as an exclusive resource when running two instances of HOG and two instances of SD, we could expect HOG's worst-case execution time to be closer to 1680.0ms, which is the sum of each instance's worst-case execution time in isolation. By examining the curve for SD+{2HOG,SD} in Fig. 5, we see that SD in this scenario has a median execution time only approximately 30ms worse than its execution time in isolation. Since a single iteration of HOG performs over 180 kernel invocations of varying sizes, and an iteration of SD performs only one, the plots support the hypothesis that a large portion of the effect on HOG is due to HOG's multiple kernels being interleaved with SD's single kernel at multiple points in each HOG iteration. While one may argue that this scheduling in SD's favor is beneficial in some applications, the significantly increased execution time for HOG may result in an overall net loss in terms of schedulability.

Obs. 4. The TX1 platform exhibited similar trends to those observed on the TK1.

The TX1, with greater resources, unsurprisingly exhibited improved execution times. Most interference patterns, however, applied to both platforms. This is shown in Fig. 7, which shows similar patterns to Fig. 4, and Fig. 8, which is analogous to Fig. 6 (discussed next).

An anomalous result. We conclude this section by discussing an anomalous result that suggests that further study of sources of interference among GPU-using tasks is needed.

Obs. 5. In rare cases, a benchmark program exhibited *better* performance when executing in the presence of a competing workload rather than in isolation.

We were very surprised to find that in some cases, increasing the concurrent workload unintuitively led to slight *improvements* in observed benchmark execution times. We observed such improvements in two sets of scenarios, shown in Figs. 6 and 8, where instances of HOG exhibited execution-time improvements with additional competition. This behavior was noticed in HOG with one or two VADD competitors on the TK1, and with up to 3 VADD competitors on the TX1. The only other scenarios where we observed such behavior involved the CONV benchmark competing against additional CONV instances.

Our current hypothesis is that this behavior is due to DRAM or CPU L2 cache activity. This hypothesis is based on the observation that, in Fig. 6, the VADD benchmark runs solely on the CPU. This fact eliminates GPU contention as the source of the anomaly in Fig. 6, leaving only hardware resources shared by the two benchmarks as potential causes: the CPU, its L2 cache, and the DRAM banks. We still, however, do not have a concrete explanation of this anomalous behavior, and plan to continue investigating it in hopes of identifying specific causes.

5 Conclusion

In order to effectively use GPUs in automotive settings, it is imperative to not waste GPU capacity. Such waste can lead to the necessity of introducing additional hardware, which can have a detrimental impact with respect

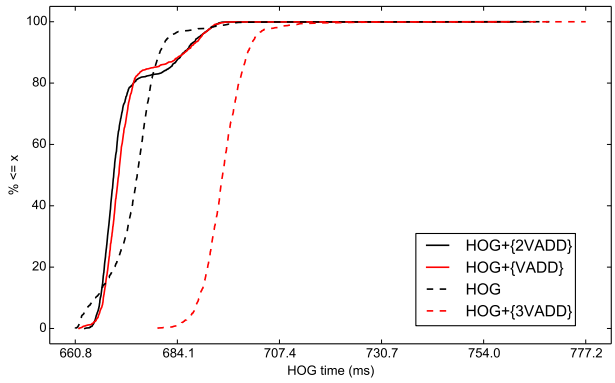

Figure 6: CDF of execution times of HOG in scenarios involving VADD competitors (which are CPU-only).

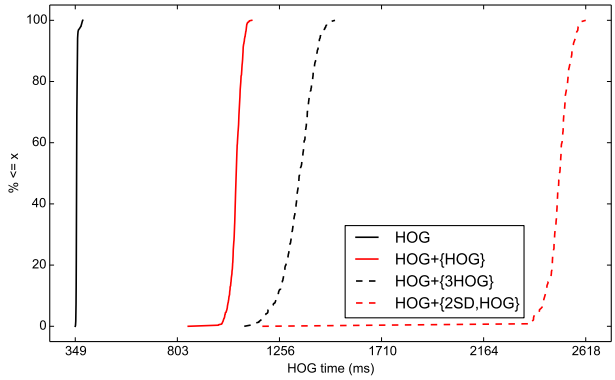

Figure 7: The same scenarios as Fig. 4 running on the TX1.

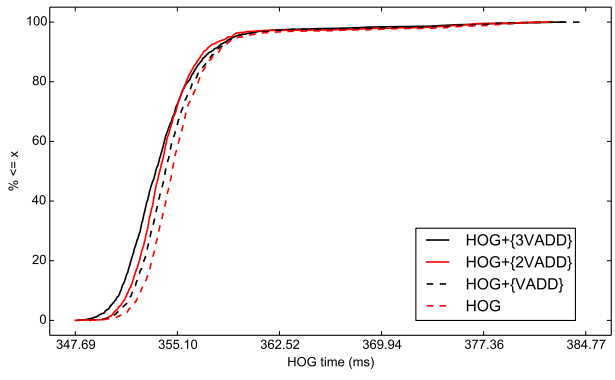

Figure 8: The same scenarios as Fig. 6 running on the TX1.

to SWaP and monetary cost. Unfortunately, most prior GPU management frameworks proposed for real-time systems [7, 8, 9, 12, 16, 25, 26, 49, 47, 48, 54, 55] preclude multiple tasks from executing GPU kernels concurrently. If such a kernel requires only a relatively small fraction of a GPU's processing cores, then much of that GPU's capacity will be wasted. In this paper, we have explored the possibility of allowing multiple kernels to be co-scheduled in the context of image-processing applications. Our results suggest that, in some cases, allowing multiple kernels to be co-scheduled can have a positive impact on real-time schedulability. Allowing such functionality will require new extensions to prior real-time GPU management frameworks.

In future work, we plan to introduce such extensions to the frameworks developed by our group, GPUSync and GPUSyncLite. These extensions will require the use of real-time locking protocols that sometimes allow multiple tasks to hold locks simultaneously. Blocking analysis will be required for these protocols as well. We believe that the needed protocols can be obtained by using ideas found in recently proposed multiprocessor real-time locking protocols for managing replicated resources [34]. Our idea here is to abstractly view a single GPU as a replicated resource and require a task to lock only the replicas it needs. In other future work, we intend to conduct more in-depth experimental studies to try to discern the root sources of interference that cause some kernels to perform poorly when co-scheduled. Additionally, we plan to consider other GPU-based hardware platforms that might be viable in automotive use cases.

References

[1] Google self-driving car project. Online at `https://www.google.com/selfdrivingcar/`, 2016.

[2] A. Alhammad and R. Pellizzoni. Trading cores for memory bandwidth in real-time systems. In *RTAS '16*.

[3] A Alhammad, S. Wasly, and R. Pellizzoni. Memory efficient global scheduling of real-time tasks. In *RTAS '15*.

[4] S. Altmeyer, R. Douma, W. Lunniss, and R.I. Davis. Evaluation of cache partitioning for hard real-time systems. In *ECRTS '14*.

[5] AMD. Amd embedded g-series system-on-chip product brief. Online at `https://www.amd.com/Documents/AMDGSeriesSOCProductBrief.pdf`.

[6] N. Audsley. Memory architecture for NoC-based real-time mixed criticality systems. In *WMC '13*.

[7] J. Aumiller, S. Brandt, S. Kato, and N. Rath. Supporting low-latency CPS using GPUs and direct I/O schemes. In *RTCSA '12*.

[8] C. Basaran and K. Kang. Supporting preemptive task executions and memory copies in GPGPUs. In *ECRTS '12*.

[9] K. Berezovskyi, , L. Santinelli, K. Bletsas, and E. Tovar. WCET measurement-based and extreme value theory characterisation of CUDA kernels. In *RTNS '14*.

[10] K. Berezovskyi, K. Bletsas, and B. Andersson. Makespan computation for GPU threads running on a single streaming multiprocessor. In *ECRTS '12*.

[11] K. Berezovskyi, K. Bletsas, and S. Petters. Faster makespan estimation for GPU threads on a single streaming multiprocessor. In *ETFA '13*.

[12] A. Betts and A. Donaldson. Estimating the WCET of GPU-accelerated

applications using hybrid analysis. In *ECRTS '13*.

[13] M. Campoy, A. Ivars, and J. Mataix. Static use of locking caches in multitask preemptive real-time systems. In *IEEE/IEE Real-Time Embedded Sys. Workshop '01*.

[14] M. Chisholm, B. Ward, N. Kim, and J. Anderson. Cache sharing and isolation tradeoffs in multicore mixed-criticality systems. In *RTSS '15*.

[15] G. Elliott. *Real-Time Scheduling of GPUs, with Applications in Advanced Automotive Systems*. PhD thesis, University of North Carolina at Chapel Hill, 2015.

[16] G. Elliott, B. Ward, and J. Anderson. GPUSync: A framework for real-time GPU management. In *RTSS '13*.

[17] G. Giannopoulou, N. Stoimenov, P. Huang, and L.Thiele. Scheduling of mixed-criticality applications on resource-sharing multicore systems. In *EMSOFT '13*.

[18] M. Hassan and H. Patel. Criticality- and requirement-aware bus arbitration for multi-core mixed criticality systems. In *RTAS '16*.

[19] M. Hassan, H. Patel, and R. Pellizzoni. A framework for scheduling DRAM memory accesses for multi-core mixed-time critical systems. In *RTAS '15*.

[20] J. Herter, P. Backes, F. Haupenthal, and J. Reineke. CAMA: A predictable cache-aware memory allocator. In *ECRTS '11*.

[21] Intel. Intel atom processor series product brief. Online at `http://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/atom-x3-c3000-brief.pdf`.

[22] R. Jain, C. Hughs, and S. Adve. Soft real-time scheduling on simultaneous multithreaded processors. In *RTSS '02*.

[23] J. Jalle, E. Quinones, J. Abella, L. Fossati, M. Zulianello, and P. Cazorla. A dual-criticality memory controller (DCmc) proposal and evaluation of a space case study. In *RTSS '14*.

[24] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *ACMMM '14*.

[25] S. Kato, K. Lakshmanan, A. Kumar, M. Kelkar, Y. Ishikawa, and R. Rajkumar. RGEM: A responsive GPGPU execution model for runtime engines. In *RTSS '11*.

[26] S. Kato, K. Lakshmanan, R. Rajkumar, and Y. Ishikawa. TimeGraph: GPU scheduling for real-time multi-tasking environments. In *USENIX Annual Technical Conference '11*.

[27] H. Kim, D. Broman, E. Lee, M. Zimmer, A. Shrivastava, and J. Oh. A predictable and command-level priority-based DRAM controller for mixed-criticality systems. In *RTAS '15*.

[28] H. Kim, D. de Niz, B. Anderson, M. Klein, O. Mutlu, and R. Rajkumar. Bounding memory interference delay in cots-based multi-core systems. In *RTAS '14*.

[29] H. Kim, A. Kandhalu, and R. Rajkumar. A coordinated approach for practical OS-level cache management in multi-core real-time systems. In *ECRTS '13*.

[30] N. Kim, B. Ward, M. Chisholm, C.-Y. Fu, J. Anderson, and F.D. Smith. Attacking the one-out-of-m multicore problem by combining hardware management with mixed-criticality provisioning. In *RTAS '16*.

[31] Y. Krishnapillai, Z. Wu, and R. Pellizzoni. ROC: A rank-switching, open-row DRAM controller for time-predictable systems. In *ECRTS '14*.

[32] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*.

[33] R. Mancuso, E. Betti, M. Cesati, M. Caccamo, and R. Pellizzoni. Real-time colored lockdown for cache-based multi-core architectures. In *RTAS '13*.

[34] C. Nemitz, K. Yang, M. Yang, P. Ekberg, and J. Anderson. Multiprocessor real-time locking protocols for replicated resources. In *ECRTS '16*.

[35] J. Nowotsch, M. Paulitsch, D. Buhler, H. Theiling, S. Wegener, and M. Schmidt. Multi-core interference-sensitive WCET analysis leveraging runtime resource capacity environment. In *ECRTS '14*.

[36] NVIDIA. Cuda sample programs. Online at `http://docs.nvidia.com/cuda/cuda-samples`.

[37] NVIDIA. Cuda zone. Online at `http://www.nvidia.com/object/cuda_home_new.html`.

[38] NVIDIA. Jetson tx1 system-on-module data sheet. Online at `https://developer.nvidia.com/embedded/downloads`.

[39] NVIDIA. Whitepaper: NVIDIA Tegra K1. Online at `http://www.nvidia.com/content/pdf/tegra_white_papers/tegra-k1-whitepaper.pdf`.

[40] NVIDIA. Whitepaper: NVIDIA Tegra X1. Online at `http://international.download.nvidia.com/pdf/tegra/Tegra-X1-whitepaper-v1.0.pdf`.

[41] NXP. i.mx 6dual/6quad automotive and infotainment applications processors data sheet. Online at `http://cache.freescale.com/files/32bit/doc/data_sheet/IMX6DQAEC.pdf`.

[42] R. Pellizzoni, A. Schranzhofer, J. Chen, M. Caccamo, and L. Thiele. Worst case delay analysis for memory interference in multicore systems. In *DATE '10*.

[43] V. Prisacariu and I. Reid. fastHOG–a real-time GPU implementation of HOG. *Department of Engineering Science*, 2310, 2009.

[44] R. Tabish, R. Mancuso, S. Wasly, A. Alhammad, S. Phatak, R. Pellizzoni, and M. Caccamo. A real-time scratchpad-centric OS for multi-core. In *RTAS '16*.

[45] S. Thrun. Toward robotic cars. *Communications of the ACM*, 53:99–106, 2010.

[46] P. Valsan, H. Yun, and F. Farshchi. Taming non-blocking caches to improve isolation in multicore real-time systems. In *RTAS '16*.

[47] U. Verner, A. Mendelson, and A. Schuster. Batch method for efficient resource sharing in real-time multi-GPU systems. In *ICDCN '14*.

[48] U. Verner, A. Mendelson, and A. Schuster. Scheduling periodic real-time communication in multi-GPU systems. In *ICCCN '14*.

[49] U. Verner, A. Mendelson, and A. Schuster. Scheduling processing of real-time data streams on heterogeneous multi-GPU systems. In *SYSTOR '12*.

[50] B. Ward, J. Herman, C. Kenna, and J. Anderson. Making shared caches more predictable on multicore platforms. In *ECRTS '13*.

[51] M. Xu, S. Mohan, C. Chen, and L. Sha. Analysis and implementation of global preemptive fixed-priority scheduling with dynamic cache allocation. In *RTAS '16*.

[52] H. Yun, R. Mancuso, Z. Wu, and R. Pellizzoni. PALLOC: DRAM bank-aware memory allocator for performance isolation on multicoore platforms. In *RTAS '14*.

[53] H. Yun, G. Yao, R. Pellizzoni, M. Caccamo, and L. Sha. Memory access control in multiprocessor for real-time systems with mixed criticality. In *ECRTS '12*.

[54] J. Zhong and B. He. Kernelet: High-throughput GPU kernel executions with dynamic slicing and scheduling. *TIEEE Transactions on Parallel and Distributed Systems*, 25:15221532, 2014.

[55] H. Zhou, G. Tong, and C. Liu. GPES: A preemptive execution system for GPGPU computing. In *RTAS '15*.

A Tables of Experimental Results

Tables 1, 2, and 3 contain a summary of all experimental data obtained for each of our three primary benchmarks, using standard GPU memory copying mechanisms (as opposed to zero-copy), while running on the TK1. Tables 4, 5, and 6 contain the same scenarios using the zero-copy versions of each benchmark on the TK1.

Likewise for the TX1, tables 7, 8 and 9 contain the data obtained for the copy versions of the benchmarks. Zero-copy times are contained in tables 10, 11 and 12.

	Total time					Memory copy time					Kernel time				
Scenario	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV
CONV	196.5	306.3	212.8	223.2	0.088	114.8	136.3	115.1	116.3	0.039	81.4	166.4	97.5	106.8	0.168
CONV+{2CONV,HOG}	196.7	510.9	314.2	315.4	0.192	117.4	254.5	136.4	142.2	0.140	76.7	338.6	173.1	173.1	0.302
CONV+{2CONV,SD}	236.1	454.1	343.1	343.2	0.103	116.3	222.2	138.4	142.0	0.123	106.7	300.3	202.0	201.1	0.164
CONV+{2CONV}	197.0	454.6	270.4	278.4	0.133	115.8	208.7	125.9	131.6	0.114	77.4	288.4	150.5	146.7	0.264
CONV+{2HOG,CONV}	202.2	557.2	320.5	324.5	0.210	117.7	281.0	141.4	149.4	0.177	77.2	326.7	174.4	174.9	0.299
CONV+{2HOG,SD}	237.9	518.2	321.9	329.6	0.165	118.2	307.9	144.6	151.5	0.176	103.0	338.8	174.0	178.0	0.213
CONV+{2HOG}	204.6	469.7	281.3	289.9	0.196	116.9	280.2	134.9	144.9	0.173	77.0	268.0	142.9	144.8	0.276
CONV+{2MMUL}	248.1	360.8	324.2	310.1	0.100	117.4	144.5	119.4	123.8	0.067	130.0	216.8	205.3	186.2	0.153
CONV+{2SD,CONV}	288.7	473.0	379.3	381.7	0.064	116.5	248.5	139.4	143.7	0.127	160.4	306.6	237.0	237.9	0.097
CONV+{2SD,HOG}	255.1	504.6	354.5	359.4	0.095	116.8	226.2	140.0	143.2	0.119	107.0	347.5	212.8	216.0	0.131
CONV+{2SD}	218.4	399.3	320.2	320.8	0.064	115.6	189.0	132.4	133.9	0.083	77.9	252.0	181.2	186.8	0.095
CONV+{2VADD}	196.8	301.3	212.6	224.9	0.092	115.1	139.3	115.4	116.8	0.040	81.4	165.2	96.9	108.0	0.179
CONV+{3CONV}	193.1	473.3	301.0	303.7	0.076	116.7	199.9	135.9	137.2	0.083	76.2	324.4	161.9	166.4	0.124
CONV+{3HOG}	206.4	576.9	308.9	316.1	0.190	118.6	314.4	142.5	149.9	0.178	78.0	335.0	161.1	166.0	0.260
CONV+{3MMUL}	252.0	367.4	338.4	328.4	0.081	120.1	150.5	123.6	128.8	0.076	129.6	217.5	215.4	199.5	0.128
CONV+{3SD}	208.1	493.1	393.5	391.6	0.074	115.6	205.5	143.5	144.5	0.112	80.0	330.8	243.6	247.0	0.107
CONV+{3VADD}	198.7	334.7	215.0	227.3	0.096	115.2	143.5	116.6	118.0	0.044	82.2	215.4	98.1	109.1	0.184
CONV+{CONV,HOG,SD}	225.2	497.8	326.7	326.4	0.160	116.4	245.6	140.2	144.1	0.139	100.0	323.2	180.0	182.2	0.237
CONV+{CONV,HOG}	196.6	421.8	251.0	261.6	0.184	116.3	233.7	127.7	134.7	0.131	76.7	254.1	114.7	126.7	0.306
CONV+{CONV,SD}	214.4	386.8	268.8	273.8	0.100	115.7	181.0	130.0	130.4	0.073	97.3	223.8	139.5	143.3	0.180
CONV+{CONV}	192.2	356.4	197.7	209.8	0.137	115.7	152.4	119.3	120.1	0.046	76.0	221.4	77.6	89.5	0.307
CONV+{HOG,SD}	226.2	395.0	274.4	280.7	0.100	115.9	252.7	132.5	136.8	0.125	98.8	218.4	140.8	143.8	0.122
CONV+{HOG}	196.9	376.2	231.6	238.8	0.120	116.0	180.7	125.5	131.6	0.110	77.1	223.5	103.1	107.0	0.164
CONV+{MMUL}	229.2	391.9	277.1	278.5	0.100	116.4	141.0	117.5	121.9	0.066	112.8	274.9	159.1	156.5	0.161
CONV+{SD}	206.7	304.9	254.4	253.2	0.047	115.1	154.4	121.9	123.4	0.063	85.1	150.0	126.6	129.7	0.080
CONV+{VADD}	196.8	333.6	212.4	223.8	0.090	115.0	138.7	115.3	116.6	0.039	81.6	216.2	96.8	107.1	0.174

Table 1: Table of runtimes (in ms) of "copy" versions of the CONV benchmark under all scenarios, along with coefficients of variation.

	Total time					Memory copy time					Kernel time				
Scenario	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV
HOG	660.8	768.9	674.9	674.3	0.010	36.0	56.8	36.2	36.9	0.083	624.4	732.2	638.3	637.3	0.009
HOG+{2CONV,HOG}	1651.7	2733.1	2338.9	2343.0	0.072	40.6	163.3	63.1	69.8	0.345	1592.5	2622.8	2286.4	2273.0	0.075
HOG+{2CONV,SD}	886.4	4095.8	3119.9	3172.9	0.145	41.5	189.2	109.1	102.8	0.386	844.7	3963.0	2999.7	3070.0	0.147
HOG+{2CONV}	826.0	1898.5	1599.9	1577.5	0.126	36.2	165.1	64.6	70.6	0.393	789.7	1816.0	1548.7	1506.8	0.129
HOG+{2HOG,CONV}	1580.9	2744.7	2414.6	2415.0	0.052	45.9	188.5	80.7	85.7	0.298	1465.3	2613.9	2313.3	2329.1	0.054
HOG+{2HOG,SD}	1720.9	3138.1	2869.7	2859.6	0.040	42.1	191.8	83.9	86.9	0.309	1662.5	3070.0	2786.3	2772.7	0.041
HOG+{2HOG}	1809.3	1955.7	1842.6	1844.0	0.005	54.7	164.0	72.0	73.3	0.119	1726.3	1791.6	1771.0	1770.4	0.002
HOG+{2MMUL}	670.0	768.9	677.1	684.8	0.025	35.9	56.2	36.4	37.4	0.100	634.0	730.8	640.5	647.4	0.027
HOG+{2SD,CONV}	2226.2	4293.0	3848.1	3836.8	0.054	37.3	209.2	95.1	100.8	0.456	2188.8	4163.2	3746.1	3735.8	0.053
HOG+{2SD,HOG}	2393.9	4062.7	3747.0	3741.4	0.040	36.4	185.5	118.7	114.0	0.247	2357.4	3942.1	3630.9	3627.3	0.040
HOG+{2SD}	967.8	3227.3	2781.8	2778.5	0.066	37.0	123.3	51.1	62.2	0.394	916.4	3126.4	2726.6	2716.2	0.066
HOG+{2VADD}	663.0	764.5	669.6	672.2	0.011	36.1	57.8	36.3	39.2	0.159	626.7	727.2	632.8	633.0	0.005
HOG+{3CONV}	1596.6	3422.3	2419.8	2444.1	0.159	41.6	195.5	81.0	86.1	0.412	1416.4	3294.4	2321.5	2357.9	0.161
HOG+{3HOG}	2388.0	2616.3	2507.3	2506.2	0.016	41.8	150.5	66.1	69.7	0.246	2317.1	2527.6	2440.5	2435.8	0.016
HOG+{3MMUL}	670.6	776.5	699.1	706.1	0.031	36.9	58.9	37.8	39.5	0.127	628.7	737.8	655.5	666.6	0.033
HOG+{3SD}	1850.5	4936.0	4602.7	4559.0	0.055	41.6	203.4	118.6	115.9	0.372	1692.1	4808.4	4479.4	4443.0	0.055
HOG+{3VADD}	679.6	777.2	694.4	694.7	0.008	36.2	59.1	36.7	37.0	0.065	643.0	739.5	657.5	657.6	0.007
HOG+{CONV,HOG,SD}	2597.7	3047.6	2820.4	2830.3	0.024	38.7	161.9	55.9	70.2	0.441	2554.5	2942.6	2756.3	2760.1	0.022
HOG+{CONV,HOG}	1316.9	1959.3	1719.8	1693.2	0.053	54.1	200.0	97.2	100.3	0.228	1252.7	1858.9	1612.5	1592.7	0.055
HOG+{CONV,SD}	1779.2	2227.9	1997.2	1982.7	0.044	38.5	159.9	55.3	72.8	0.482	1729.8	2158.1	1926.6	1909.8	0.042
HOG+{CONV}	898.4	1082.2	980.3	980.3	0.011	36.3	107.6	41.3	43.1	0.210	856.6	1040.1	938.8	937.2	0.013
HOG+{HOG,SD}	1877.0	2367.6	2128.0	2124.6	0.038	37.7	170.0	79.8	82.3	0.313	1808.7	2281.8	2051.3	2042.2	0.042
HOG+{HOG}	1238.8	1355.8	1264.8	1264.8	0.005	68.9	116.5	92.6	92.3	0.056	1162.4	1277.9	1172.0	1172.1	0.004
HOG+{MMUL}	663.1	769.4	674.8	681.3	0.025	36.0	55.4	36.2	38.1	0.128	626.8	730.5	637.9	643.1	0.026
HOG+{SD}	1103.2	1467.6	1326.2	1327.4	0.036	36.8	75.2	48.2	48.0	0.147	1048.1	1415.0	1282.7	1279.3	0.037
HOG+{VADD}	661.7	759.5	670.7	672.8	0.011	36.1	57.1	36.3	38.7	0.151	625.4	722.2	633.9	634.0	0.005

Table 2: Table of runtimes (in ms) of "copy" versions of the HOG benchmark under all scenarios, along with coefficients of variation.

	Total time					Memory copy time					Kernel time				
Scenario	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV
SD	21.2	71.1	21.9	22.2	0.043	1.0	6.8	1.2	1.3	0.119	20.2	68.9	20.6	20.8	0.038
SD+{2CONV,HOG}	21.4	168.7	48.4	54.9	0.454	1.0	128.4	9.2	16.3	1.023	20.2	116.0	34.9	38.5	0.427
SD+{2CONV,SD}	23.1	173.4	54.4	64.2	0.438	1.1	116.1	10.5	18.0	0.984	20.3	123.0	40.7	46.1	0.429
SD+{2CONV}	21.3	136.0	37.7	41.7	0.399	1.0	92.6	6.6	8.6	1.074	20.2	102.1	27.7	33.0	0.408
SD+{2HOG,CONV}	21.4	170.0	46.6	54.5	0.491	1.0	101.4	9.2	15.5	0.994	20.2	139.0	31.7	38.9	0.487
SD+{2HOG,SD}	22.9	149.8	48.8	54.6	0.357	1.0	118.0	18.5	20.0	0.627	20.3	126.5	26.9	34.5	0.454
SD+{2HOG}	21.4	125.4	38.2	44.6	0.431	1.0	91.5	8.1	13.3	0.988	20.2	115.5	25.1	31.2	0.444
SD+{2MMUL}	21.3	55.8	22.1	22.5	0.064	1.0	2.1	1.2	1.2	0.140	20.2	53.6	20.8	21.3	0.064
SD+{2SD,CONV}	26.9	161.1	67.3	71.8	0.329	1.1	118.6	16.4	23.7	0.869	20.4	101.9	43.0	48.0	0.289
SD+{2SD,HOG}	23.2	130.3	61.1	64.5	0.231	1.2	100.1	15.9	18.0	0.624	20.4	108.1	42.1	46.4	0.248
SD+{2SD}	27.5	130.5	53.0	55.3	0.203	1.2	79.3	6.6	10.7	1.002	20.4	81.6	42.8	44.5	0.199
SD+{2VADD}	21.2	88.2	21.8	22.1	0.046	1.0	2.5	1.3	1.3	0.107	20.2	68.2	20.3	20.7	0.037
SD+{3CONV}	21.3	172.0	51.5	55.2	0.432	1.0	115.9	9.3	16.5	1.089	20.2	114.3	35.1	38.5	0.393
SD+{3HOG}	21.5	162.3	46.5	51.0	0.405	1.1	120.1	10.2	15.7	0.880	20.3	141.3	28.8	35.2	0.436
SD+{3MMUL}	21.3	70.5	22.2	22.7	0.082	1.0	2.1	1.1	1.2	0.171	20.2	68.4	20.9	21.4	0.080
SD+{3SD}	26.8	131.7	73.4	71.9	0.143	1.2	90.4	26.9	23.5	0.564	20.5	86.4	42.1	48.3	0.212
SD+{3VADD}	21.2	77.3	22.1	22.4	0.067	0.9	17.4	1.6	1.6	0.223	20.2	68.4	20.3	20.7	0.038
SD+{CONV,HOG,SD}	22.0	158.1	54.5	60.3	0.376	1.1	101.5	16.1	19.1	0.742	20.3	122.3	38.7	41.1	0.413
SD+{CONV,HOG}	21.3	127.3	35.8	42.6	0.470	1.0	90.4	6.9	9.5	1.161	20.2	107.1	28.2	33.0	0.434
SD+{CONV,SD}	23.0	116.9	44.7	51.9	0.350	1.0	93.3	6.7	12.6	1.110	20.3	81.9	40.5	39.1	0.314
SD+{CONV}	21.3	85.5	21.7	29.8	0.438	1.0	18.9	1.3	3.9	1.178	20.2	82.1	20.3	25.8	0.431
SD+{HOG,SD}	21.9	120.9	46.4	50.0	0.316	1.0	83.4	15.8	17.6	0.706	20.3	88.7	25.0	32.3	0.404
SD+{HOG}	21.3	82.9	28.8	33.0	0.349	1.0	16.4	4.2	5.1	0.636	20.2	67.9	22.6	27.9	0.372
SD+{MMUL}	21.3	67.8	22.1	22.5	0.079	1.0	2.8	1.3	1.3	0.123	20.2	66.0	20.7	21.1	0.069
SD+{SD}	27.5	122.2	42.2	41.2	0.117	1.0	18.4	4.9	5.6	0.826	20.3	113.1	40.5	35.5	0.213
SD+{VADD}	21.2	243.1	21.8	22.1	0.088	1.0	6.9	1.3	1.3	0.113	20.2	238.2	20.3	20.7	0.070

Table 3: Table of runtimes (in ms) of "copy" versions of the SD benchmark under all scenarios, along with coefficients of variation.

	Total time					Memory copy time					Kernel time				
Scenario	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV
CONV	202.2	336.6	219.9	224.4	0.068	114.0	134.7	114.3	115.2	0.036	87.9	220.6	105.4	109.0	0.129
CONV+{2CONV,HOG}	204.0	522.6	368.5	363.4	0.170	114.8	181.5	120.6	125.8	0.101	85.9	366.5	244.2	237.5	0.235
CONV+{2CONV,SD}	223.1	458.4	385.5	384.1	0.072	114.9	147.3	119.2	120.2	0.043	107.4	334.3	266.2	263.8	0.101
CONV+{2CONV}	206.7	544.8	352.2	355.8	0.040	115.1	150.4	128.3	129.3	0.037	89.9	415.3	223.4	226.4	0.062
CONV+{2HOG,CONV}	209.7	521.2	322.3	322.9	0.194	114.5	201.3	118.2	125.3	0.130	85.0	347.1	197.2	197.4	0.282
CONV+{2HOG,SD}	231.1	490.7	303.4	310.3	0.146	114.4	195.9	116.8	122.8	0.120	108.1	307.2	184.3	187.3	0.212
CONV+{2HOG}	210.3	457.3	278.2	288.0	0.185	114.4	191.0	117.5	125.2	0.133	85.0	312.0	157.8	162.6	0.281
CONV+{2MMUL}	226.6	395.1	280.7	278.5	0.066	115.4	140.3	115.9	117.2	0.039	110.7	279.4	164.6	161.3	0.110
CONV+{2SD,CONV}	281.7	428.9	351.5	349.0	0.077	114.7	143.7	116.7	117.9	0.043	166.5	309.2	233.3	231.0	0.111
CONV+{2SD,HOG}	269.2	468.6	332.8	338.3	0.093	114.7	166.7	116.1	119.4	0.069	152.4	324.3	214.9	218.7	0.132
CONV+{2SD}	242.2	360.6	289.7	292.9	0.060	114.8	147.5	115.3	116.7	0.042	127.1	242.3	173.7	176.0	0.097
CONV+{2VADD}	202.9	306.7	219.5	224.5	0.069	114.2	138.5	114.5	115.8	0.039	88.3	171.6	104.8	108.5	0.128
CONV+{3CONV}	203.1	461.4	334.8	338.2	0.141	114.9	146.4	118.9	120.7	0.048	86.0	328.2	215.7	217.4	0.210
CONV+{3HOG}	217.3	528.8	309.6	316.3	0.208	114.1	208.8	117.3	126.3	0.165	86.9	348.1	185.5	189.8	0.307
CONV+{3MMUL}	236.9	395.5	282.2	283.7	0.045	116.1	143.2	117.0	118.5	0.043	120.0	278.4	164.9	165.1	0.069
CONV+{3SD}	280.5	450.5	353.9	357.2	0.066	114.8	140.5	115.4	116.6	0.039	149.7	335.2	236.5	240.4	0.095
CONV+{3VADD}	203.6	306.0	220.8	226.7	0.075	114.4	140.2	114.8	116.2	0.041	88.8	172.4	105.7	110.4	0.139
CONV+{CONV,HOG,SD}	226.3	505.8	330.3	329.2	0.148	114.8	174.6	117.6	121.1	0.079	110.0	356.1	209.4	208.0	0.218
CONV+{CONV,HOG}	204.4	455.6	281.2	281.9	0.165	114.8	175.4	118.1	124.1	0.101	85.2	319.6	158.4	157.6	0.256
CONV+{CONV,SD}	220.5	365.6	268.9	272.2	0.090	114.8	142.7	115.9	117.3	0.039	105.1	231.6	151.6	154.9	0.155
CONV+{CONV}	199.9	375.8	207.3	216.3	0.090	115.2	138.0	115.7	116.8	0.037	84.4	255.6	90.3	99.4	0.191
CONV+{HOG,SD}	229.2	426.6	279.3	287.7	0.109	114.8	165.6	116.6	121.5	0.085	110.0	261.0	160.4	166.1	0.161
CONV+{HOG}	207.3	360.2	234.9	245.9	0.128	114.9	167.1	117.2	124.8	0.108	85.3	227.0	117.0	121.0	0.182
CONV+{MMUL}	215.7	392.5	255.4	267.0	0.129	114.8	138.9	115.2	116.4	0.038	100.7	277.0	138.8	150.5	0.228
CONV+{SD}	199.9	289.0	252.9	252.8	0.038	114.5	139.1	115.0	116.3	0.040	84.6	154.9	136.6	136.5	0.062
CONV+{VADD}	202.7	306.3	219.8	225.2	0.069	114.2	137.6	114.4	115.6	0.038	88.3	171.5	105.1	109.5	0.129

Table 4: Table of runtimes (in ms) of "zero-copy" versions of the CONV benchmark under all scenarios, along with coefficients of variation.

	Total time					Memory copy time					Kernel time				
Scenario	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV
HOG	661.6	757.1	674.6	673.5	0.010	36.1	56.5	36.3	36.8	0.073	625.2	720.0	638.0	636.6	0.010
HOG+{2CONV,HOG}	1776.4	2996.5	2558.2	2554.8	0.092	47.1	169.2	66.4	75.6	0.376	1703.7	2913.1	2480.7	2479.0	0.096
HOG+{2CONV,SD}	2153.1	4063.0	2961.0	3036.2	0.122	42.2	205.8	102.8	98.6	0.436	2029.2	3921.5	2862.8	2937.4	0.124
HOG+{2CONV}	1174.1	2201.4	1525.8	1636.2	0.145	36.3	197.2	48.3	74.2	0.528	1137.7	2093.3	1480.3	1561.9	0.147
HOG+{2HOG,CONV}	1912.4	2688.9	2534.1	2511.5	0.027	45.6	317.3	98.5	102.0	0.288	1814.7	2562.9	2420.3	2409.2	0.028
HOG+{2HOG,SD}	2162.7	2800.3	2651.8	2633.6	0.033	47.6	175.5	100.3	98.4	0.254	2026.6	2710.0	2555.0	2535.0	0.037
HOG+{2HOG}	1818.9	1908.0	1845.1	1846.5	0.004	55.7	125.6	71.3	72.6	0.111	1728.1	1788.4	1773.7	1773.5	0.002
HOG+{2MMUL}	669.3	768.1	675.9	677.7	0.011	36.0	56.7	36.1	36.4	0.053	633.1	730.6	639.7	641.2	0.012
HOG+{2SD,CONV}	1707.4	3767.4	3518.1	3498.6	0.054	39.2	204.8	87.7	97.5	0.410	1634.6	3693.0	3426.6	3400.9	0.054
HOG+{2SD,HOG}	1617.5	3398.4	3195.2	3185.2	0.044	38.2	166.7	98.7	100.1	0.218	1534.4	3311.7	3098.9	3085.0	0.046
HOG+{2SD}	1440.2	2674.7	2480.1	2460.3	0.049	36.9	107.0	67.8	66.3	0.253	1370.4	2581.4	2413.9	2393.9	0.049
HOG+{2VADD}	663.7	763.3	670.0	672.6	0.011	36.0	57.9	36.3	39.2	0.161	627.4	726.2	633.1	633.3	0.006
HOG+{3CONV}	1643.5	3554.2	2040.7	2238.9	0.211	43.8	218.1	51.5	75.3	0.522	1597.9	3494.7	1991.1	2163.5	0.213
HOG+{3HOG}	1435.0	2367.3	2125.2	2095.9	0.084	41.5	165.9	67.5	71.8	0.283	1367.5	2311.1	2041.0	2023.8	0.089
HOG+{3MMUL}	663.4	777.3	697.8	701.8	0.018	36.1	59.7	36.8	37.3	0.074	627.2	739.9	660.6	664.5	0.019
HOG+{3SD}	1175.1	4254.9	4011.7	3990.1	0.062	48.9	160.2	109.2	102.2	0.245	1058.4	4140.8	3911.4	3887.8	0.063
HOG+{3VADD}	680.1	783.4	693.8	694.0	0.008	36.1	58.6	36.7	37.0	0.053	643.3	746.1	656.9	656.9	0.008
HOG+{CONV,HOG,SD}	2202.4	3016.6	2833.0	2821.3	0.028	40.3	160.7	59.3	76.6	0.451	2150.3	2976.2	2764.0	2744.5	0.027
HOG+{CONV,HOG}	1480.0	1988.3	1769.4	1761.5	0.041	50.7	181.7	97.8	105.7	0.265	1361.1	1866.6	1673.7	1655.6	0.047
HOG+{CONV,SD}	968.4	2193.6	1986.9	1972.9	0.050	39.9	163.3	54.2	71.5	0.492	844.8	2084.4	1897.4	1901.2	0.051
HOG+{CONV}	922.9	1101.9	1013.9	1016.1	0.014	36.2	116.4	41.3	44.4	0.276	884.4	1060.8	972.4	971.6	0.012
HOG+{HOG,SD}	1777.1	2255.5	2013.7	2011.1	0.040	36.9	139.1	81.5	82.7	0.288	1715.1	2158.5	1938.5	1928.3	0.044
HOG+{HOG}	1238.6	1353.0	1266.0	1266.0	0.006	65.0	164.2	91.9	91.6	0.069	1164.9	1188.6	1174.1	1174.1	0.002
HOG+{MMUL}	663.7	758.9	673.9	674.2	0.011	35.9	56.4	36.1	36.5	0.061	627.6	721.8	637.6	637.6	0.011
HOG+{SD}	910.0	1413.5	1278.8	1279.9	0.034	36.6	70.6	46.2	46.9	0.158	863.7	1369.2	1235.9	1232.9	0.034
HOG+{VADD}	663.0	758.0	671.4	672.4	0.008	36.1	56.3	36.2	37.1	0.092	626.7	720.8	635.0	635.2	0.006

Table 5: Table of runtimes (in ms) of "zero-copy" versions of the HOG benchmark under all scenarios, along with coefficients of variation.

	Total time					Memory copy time					Kernel time				
Scenario	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV
SD	21.1	70.2	21.8	22.0	0.043	0.8	7.7	1.1	1.1	0.124	20.2	68.2	20.6	20.8	0.040
SD+{2CONV,HOG}	21.2	187.1	64.2	65.9	0.492	0.9	126.3	15.0	20.4	0.830	20.2	152.7	39.6	45.4	0.475
SD+{2CONV,SD}	21.4	188.8	68.9	74.3	0.439	0.9	143.2	25.2	29.8	0.728	20.3	126.1	38.3	44.4	0.465
SD+{2CONV}	21.2	140.3	43.0	50.4	0.461	0.9	101.3	10.4	11.6	0.769	20.2	106.1	28.9	38.7	0.454
SD+{2HOG,CONV}	21.3	177.2	56.2	62.1	0.490	0.9	124.0	15.0	20.8	0.839	20.3	139.6	33.5	41.1	0.497
SD+{2HOG,SD}	23.0	169.9	55.5	61.0	0.356	1.2	114.1	22.9	26.6	0.595	20.3	157.4	27.9	34.3	0.413
SD+{2HOG}	21.2	130.3	43.6	49.1	0.428	0.9	94.4	13.2	18.4	0.844	20.2	110.7	25.6	30.7	0.413
SD+{2MMUL}	21.1	98.0	21.8	22.2	0.066	0.8	2.9	0.9	1.0	0.137	20.3	95.6	20.7	21.1	0.054
SD+{2SD,CONV}	24.2	159.4	68.2	73.8	0.330	1.0	122.3	22.1	27.5	0.708	20.3	104.4	42.1	46.2	0.332
SD+{2SD,HOG}	23.6	155.8	63.4	66.8	0.214	2.0	98.0	21.9	25.7	0.524	20.3	102.6	41.5	41.1	0.265
SD+{2SD}	21.6	105.3	54.6	57.5	0.258	0.9	83.8	11.8	18.4	0.819	20.3	64.9	40.9	39.0	0.246
SD+{2VADD}	21.1	81.5	21.4	21.9	0.063	0.8	2.4	1.1	1.1	0.122	20.2	53.1	20.3	20.6	0.035
SD+{3CONV}	21.2	190.5	44.8	61.5	0.712	0.9	125.2	11.5	19.8	1.150	20.2	145.8	28.3	41.6	0.662
SD+{3HOG}	21.4	170.8	55.0	59.0	0.408	1.0	126.1	17.5	23.3	0.758	20.3	133.9	29.6	35.6	0.426
SD+{3MMUL}	21.2	98.0	22.3	22.6	0.073	0.8	13.7	1.2	1.3	0.280	20.3	95.8	20.8	21.2	0.059
SD+{3SD}	27.5	129.4	75.1	74.8	0.102	1.0	88.1	32.7	32.0	0.255	20.3	81.3	41.1	42.7	0.137
SD+{3VADD}	21.1	85.0	22.3	22.5	0.071	0.8	3.5	1.7	1.7	0.234	20.2	68.3	20.3	20.7	0.042
SD+{CONV,HOG,SD}	21.4	167.3	59.5	65.0	0.393	1.0	123.9	22.8	26.6	0.634	20.3	122.3	34.9	38.3	0.446
SD+{CONV,HOG}	21.2	132.4	38.9	44.8	0.450	0.9	90.2	9.7	12.7	0.887	20.2	105.0	24.9	32.0	0.442
SD+{CONV,SD}	21.3	126.4	47.0	55.3	0.407	0.9	85.1	14.2	18.3	0.760	20.2	84.6	34.6	36.9	0.431
SD+{CONV}	21.1	87.5	21.5	31.8	0.508	0.9	26.7	1.1	4.7	1.122	20.2	71.7	20.3	27.0	0.463
SD+{HOG,SD}	21.4	114.7	46.5	51.1	0.333	1.0	85.9	18.9	21.5	0.621	20.3	88.4	23.6	29.5	0.390
SD+{HOG}	21.2	97.6	29.8	34.0	0.341	0.9	17.7	6.2	6.6	0.601	20.2	95.7	22.6	27.3	0.355
SD+{MMUL}	21.1	66.2	21.9	22.2	0.058	0.8	2.1	1.1	1.1	0.126	20.2	53.6	20.7	21.0	0.044
SD+{SD}	21.2	63.4	40.8	39.3	0.200	0.8	26.6	10.1	10.9	0.670	20.2	45.6	20.9	28.3	0.339
SD+{VADD}	21.0	80.5	21.5	21.9	0.065	0.8	2.4	1.1	1.1	0.115	20.2	68.2	20.3	20.7	0.039

Table 6: Table of runtimes (in ms) of "zero-copy" versions of the SD benchmark under all scenarios, along with coefficients of variation.

	Total time					Memory copy time					Kernel time				
Scenario	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV
CONV	130.8	160.8	132.1	133.3	0.038	94.8	121.1	95.3	96.4	0.052	35.7	49.8	36.8	36.8	0.010
CONV+{2CONV,HOG}	133.8	299.8	230.3	225.9	0.134	96.0	142.5	107.6	108.1	0.062	37.7	180.9	120.4	117.7	0.234
CONV+{2CONV,SD}	145.6	268.8	200.1	197.4	0.113	95.6	140.3	105.2	106.2	0.062	40.5	147.7	93.4	91.1	0.212
CONV+{2CONV}	133.4	240.5	138.6	146.0	0.118	95.3	132.9	98.1	99.7	0.060	37.7	125.2	38.4	46.2	0.303
CONV+{2HOG,CONV}	140.2	283.2	207.2	209.2	0.104	96.8	142.7	107.4	108.7	0.062	41.4	164.8	99.2	100.4	0.192
CONV+{2HOG,SD}	181.6	335.5	270.6	269.9	0.060	97.2	150.9	109.1	110.0	0.066	83.9	206.7	160.6	159.7	0.087
CONV+{2HOG}	131.3	223.0	176.8	177.4	0.060	95.6	135.2	101.6	103.0	0.057	35.5	96.5	74.2	74.2	0.106
CONV+{2MMUL}	133.7	195.9	146.3	147.4	0.048	94.4	123.3	95.2	96.4	0.052	36.8	71.6	50.6	50.9	0.090
CONV+{2SD,CONV}	141.7	268.5	203.9	204.5	0.082	96.0	150.2	106.3	107.4	0.059	42.5	134.2	96.3	97.0	0.150
CONV+{2SD,HOG}	189.1	343.2	268.7	267.6	0.064	96.7	172.8	106.0	107.1	0.060	81.0	210.4	161.9	160.4	0.096
CONV+{2SD}	147.1	251.6	199.4	199.9	0.059	97.0	147.4	104.6	105.8	0.066	45.9	121.1	94.4	94.0	0.101
CONV+{2VADD}	130.7	177.6	132.1	133.4	0.040	94.9	122.0	95.4	96.5	0.054	35.6	49.7	36.7	36.7	0.013
CONV+{3CONV}	133.9	286.0	209.4	205.4	0.132	95.8	142.3	106.2	106.8	0.060	37.8	165.5	102.8	98.5	0.252
CONV+{3HOG}	132.5	340.8	264.9	265.2	0.076	96.3	144.2	107.7	108.8	0.056	36.2	204.1	156.3	156.3	0.112
CONV+{3MMUL}	135.6	224.4	167.3	168.6	0.050	99.8	142.2	105.4	107.1	0.062	35.7	87.9	61.3	61.3	0.083
CONV+{3SD}	216.3	319.8	271.9	270.7	0.059	97.0	166.9	110.2	111.0	0.065	100.0	203.2	160.8	159.6	0.088
CONV+{3VADD}	131.1	187.0	132.6	133.9	0.042	95.3	124.6	96.2	97.4	0.054	35.2	39.2	36.3	36.3	0.008
CONV+{CONV,HOG,SD}	153.0	320.5	261.7	259.8	0.076	97.2	146.4	107.4	108.7	0.063	52.2	195.4	153.3	151.0	0.122
CONV+{CONV,HOG}	134.9	261.8	207.9	204.9	0.094	97.2	140.6	103.7	105.1	0.060	37.3	137.6	102.8	99.6	0.172
CONV+{CONV,SD}	146.5	244.4	175.0	178.3	0.094	96.9	138.2	102.4	104.1	0.064	36.9	119.6	71.0	74.1	0.190
CONV+{CONV}	132.6	203.4	134.5	141.3	0.091	95.3	127.3	96.4	98.8	0.063	37.1	84.1	37.7	42.4	0.199
CONV+{HOG,SD}	144.3	251.7	201.4	200.4	0.074	96.3	135.6	103.7	104.7	0.060	46.8	131.0	97.5	95.6	0.135
CONV+{HOG}	134.3	206.6	151.5	152.2	0.061	97.9	143.8	100.3	102.0	0.055	35.4	78.5	49.8	50.1	0.138
CONV+{MMUL}	133.1	174.2	137.8	139.2	0.042	94.9	123.3	95.7	97.0	0.056	36.8	53.5	41.8	42.1	0.047
CONV+{SD}	143.5	209.2	161.3	162.8	0.059	94.8	132.5	98.7	99.8	0.060	45.6	99.8	63.0	63.0	0.122
CONV+{VADD}	130.8	179.9	132.3	133.5	0.041	94.9	121.7	95.4	96.5	0.054	35.8	49.2	36.9	36.9	0.013

Table 7: Table of runtimes (in ms) of "copy" versions of the CONV benchmark under all scenarios, along with coefficients of variation. These measurements were performed on the TX1.

	Total time					Memory copy time					Kernel time				
Scenario	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV
HOG	348.8	382.1	354.7	355.5	0.011	44.1	70.6	44.9	45.5	0.074	303.8	335.3	309.6	309.8	0.007
HOG+{2CONV,HOG}	1033.0	1428.4	1329.7	1313.5	0.049	48.3	98.8	68.7	69.5	0.148	968.6	1355.7	1257.7	1243.9	0.051
HOG+{2CONV,SD}	827.7	1241.1	1098.9	1093.8	0.039	45.4	114.0	68.9	69.7	0.173	782.2	1131.3	1029.8	1024.0	0.042
HOG+{2CONV}	539.9	893.8	658.7	676.1	0.115	44.9	102.0	57.4	59.1	0.179	490.3	835.4	605.1	617.0	0.126
HOG+{2HOG,CONV}	2021.5	2430.8	2313.0	2313.0	0.023	51.5	110.9	68.1	70.0	0.145	1955.5	2358.1	2241.0	2243.0	0.023
HOG+{2HOG,SD}	1201.0	1315.9	1264.6	1264.1	0.015	47.6	105.4	69.4	69.4	0.133	1127.7	1256.5	1194.4	1194.6	0.016
HOG+{2HOG}	1383.8	1825.0	1640.8	1642.3	0.049	50.1	98.7	64.6	64.8	0.093	1325.7	1757.4	1574.9	1577.4	0.051
HOG+{2MMUL}	426.3	603.3	537.6	538.5	0.032	44.6	75.4	45.2	46.3	0.070	381.6	557.7	491.4	492.1	0.035
HOG+{2SD,CONV}	1177.9	1891.3	1678.6	1678.5	0.044	48.7	108.3	71.1	71.5	0.142	1113.3	1812.2	1604.6	1606.9	0.045
HOG+{2SD,HOG}	1180.4	2617.8	2502.9	2494.7	0.039	48.8	98.3	75.4	75.6	0.090	1100.7	2548.3	2428.9	2419.0	0.040
HOG+{2SD}	797.9	1686.8	1410.3	1408.1	0.077	45.9	89.8	64.7	63.9	0.139	741.8	1617.3	1349.5	1344.1	0.079
HOG+{2VADD}	347.7	382.4	353.8	354.5	0.011	44.7	70.8	44.9	45.5	0.070	302.5	336.8	308.7	308.9	0.007
HOG+{3CONV}	676.5	1432.5	1039.2	1043.6	0.166	50.0	120.4	70.1	70.8	0.187	613.6	1366.8	968.3	972.7	0.176
HOG+{3HOG}	1100.0	1503.2	1346.6	1341.3	0.052	59.7	103.5	75.4	75.8	0.086	1022.4	1418.0	1273.2	1265.4	0.056
HOG+{3MMUL}	742.6	928.8	825.1	824.8	0.034	45.3	82.6	50.5	51.8	0.094	685.6	879.3	773.5	772.8	0.036
HOG+{3SD}	1746.4	2294.3	2001.3	2005.4	0.045	49.8	103.7	76.0	75.7	0.118	1669.3	2208.4	1929.0	1929.6	0.047
HOG+{3VADD}	347.8	383.2	353.5	354.2	0.011	45.0	71.8	45.4	46.0	0.069	302.4	316.8	307.9	308.1	0.008
HOG+{CONV,HOG,SD}	1656.1	2115.1	1976.9	1954.6	0.041	48.9	97.2	73.7	73.5	0.108	1570.3	2030.0	1906.8	1881.0	0.043
HOG+{CONV,HOG}	957.7	1320.8	1154.1	1153.8	0.060	45.7	97.6	64.6	65.2	0.127	887.2	1243.4	1093.2	1088.4	0.064
HOG+{CONV,SD}	689.1	1175.5	1030.3	1025.7	0.047	44.6	104.7	60.2	61.9	0.177	644.4	1114.8	963.9	963.7	0.050
HOG+{CONV}	413.3	485.4	440.3	438.4	0.036	45.6	95.2	48.0	49.7	0.097	360.5	427.3	383.7	388.6	0.041
HOG+{HOG,SD}	563.4	1789.7	1586.3	1581.0	0.059	47.8	95.6	61.8	62.7	0.124	512.1	1727.0	1523.1	1518.3	0.061
HOG+{HOG}	851.0	1133.2	1064.1	1064.4	0.024	44.5	85.7	54.3	55.3	0.107	792.9	1074.7	1009.3	1008.9	0.027
HOG+{MMUL}	365.8	429.1	379.7	380.4	0.019	44.5	74.1	45.1	45.8	0.074	318.9	383.0	333.9	334.4	0.018
HOG+{SD}	515.4	614.5	564.5	565.0	0.028	45.3	80.9	49.8	50.4	0.095	463.5	565.1	513.8	514.5	0.030
HOG+{VADD}	348.7	384.8	354.2	354.9	0.011	44.7	71.0	44.9	45.5	0.067	303.7	339.6	309.1	309.4	0.008

Table 8: Table of runtimes (in ms) of "copy" versions of the HOG benchmark under all scenarios, along with coefficients of variation. These measurements were performed on the TX1.

	Total time					Memory copy time					Kernel time				
Scenario	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV
SD	9.5	38.0	10.4	10.4	0.065	0.7	28.6	1.5	1.5	0.252	8.6	27.0	8.9	8.8	0.014
SD+{2CONV,HOG}	9.6	92.6	39.7	41.3	0.417	0.8	37.4	11.4	12.2	0.634	8.6	61.3	27.7	29.0	0.351
SD+{2CONV,SD}	9.7	75.5	36.3	34.8	0.464	0.9	32.0	9.4	10.0	0.608	8.6	54.3	25.7	24.7	0.450
SD+{2CONV}	9.5	61.2	22.2	21.4	0.521	0.8	29.0	4.4	5.4	0.851	8.6	36.4	17.4	15.8	0.437
SD+{2HOG,CONV}	9.5	87.2	31.5	33.9	0.298	0.8	30.3	9.8	10.1	0.368	8.6	58.0	21.4	23.8	0.327
SD+{2HOG,SD}	10.5	70.3	45.2	45.1	0.152	1.8	43.1	15.9	16.0	0.203	8.6	47.3	29.1	29.0	0.183
SD+{2HOG}	9.5	62.0	40.1	39.0	0.215	0.8	25.8	8.0	8.3	0.381	8.6	46.1	31.6	30.6	0.200
SD+{2MMUL}	9.6	64.9	13.7	13.5	0.162	0.8	25.5	2.0	2.4	0.450	8.6	18.6	11.1	10.9	0.170
SD+{2SD,CONV}	9.6	80.4	43.4	44.3	0.297	0.9	34.9	14.3	14.9	0.393	8.6	59.6	29.0	29.3	0.326
SD+{2SD,HOG}	10.2	83.1	49.6	48.3	0.194	1.4	36.2	14.7	14.9	0.308	8.6	56.4	33.9	33.3	0.264
SD+{2SD}	9.7	51.6	26.3	25.4	0.202	0.8	37.4	9.3	9.1	0.296	8.6	25.3	16.5	16.2	0.211
SD+{2VADD}	9.4	73.4	10.4	10.4	0.096	0.7	20.9	1.5	1.5	0.182	8.6	11.6	8.8	8.8	0.008
SD+{3CONV}	9.6	83.4	31.0	31.9	0.558	0.9	31.0	7.9	8.6	0.754	8.6	59.0	22.5	23.2	0.508
SD+{3HOG}	9.4	67.9	46.3	46.2	0.193	0.8	24.9	10.8	11.2	0.341	8.6	50.9	35.2	35.0	0.165
SD+{3MMUL}	9.7	73.0	15.6	15.6	0.213	0.9	24.8	2.1	2.5	0.389	8.6	25.0	12.9	13.0	0.240
SD+{3SD}	10.2	60.2	36.1	35.9	0.191	1.4	28.5	12.8	12.9	0.241	8.6	47.7	22.6	22.9	0.232
SD+{3VADD}	9.4	75.7	10.4	10.4	0.099	0.8	27.0	1.5	1.5	0.251	8.6	11.7	8.8	8.8	0.009
SD+{CONV,HOG,SD}	9.6	84.7	35.3	38.1	0.308	0.9	41.5	14.3	15.4	0.447	8.6	53.6	20.8	22.6	0.348
SD+{CONV,HOG}	9.5	74.5	24.6	28.4	0.417	0.8	21.5	6.5	7.0	0.532	8.6	56.7	17.6	21.2	0.412
SD+{CONV,SD}	9.6	62.3	21.3	23.5	0.461	0.9	25.6	5.6	6.7	0.640	8.6	43.3	15.3	16.7	0.464
SD+{CONV}	9.5	55.9	10.2	13.8	0.444	0.8	33.9	1.5	2.9	0.893	8.6	25.7	8.7	10.8	0.343
SD+{HOG,SD}	9.6	67.3	37.5	36.5	0.161	0.9	41.9	9.4	9.3	0.350	8.6	51.4	27.9	27.2	0.178
SD+{HOG}	9.4	45.8	16.5	16.6	0.226	0.7	29.6	3.0	3.4	0.507	8.6	22.8	13.2	13.1	0.187
SD+{MMUL}	9.4	56.8	12.4	12.5	0.139	0.8	20.6	3.2	2.7	0.378	8.6	17.5	8.9	9.7	0.143
SD+{SD}	9.7	54.3	20.0	18.6	0.329	0.8	14.7	4.9	4.6	0.534	8.6	25.9	13.1	13.8	0.325
SD+{VADD}	9.4	81.2	10.4	10.4	0.118	0.8	28.4	1.5	1.5	0.366	8.6	11.5	8.8	8.8	0.008

Table 9: Table of runtimes (in ms) of "copy" versions of the SD benchmark under all scenarios, along with coefficients of variation. These measurements were performed on the TX1.

	Total time					Memory copy time					Kernel time				
Scenario	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV
CONV	138.2	168.1	138.6	140.2	0.044	104.1	132.8	104.1	105.6	0.056	34.0	58.5	34.4	34.5	0.027
CONV+{2CONV,HOG}	142.1	258.4	176.1	177.4	0.112	104.5	138.8	106.4	108.1	0.060	36.3	124.1	68.6	69.2	0.267
CONV+{2CONV,SD}	141.5	243.3	157.0	165.4	0.119	104.2	136.2	105.4	106.9	0.059	36.3	119.8	50.4	58.4	0.311
CONV+{2CONV}	140.7	228.0	171.2	170.4	0.116	103.8	134.7	104.6	106.0	0.057	33.8	122.1	65.2	64.3	0.304
CONV+{2HOG,CONV}	142.3	251.5	183.9	183.4	0.117	104.7	140.1	106.7	108.5	0.061	35.5	118.8	76.4	74.8	0.277
CONV+{2HOG,SD}	142.9	240.1	185.0	182.4	0.100	105.7	139.9	107.3	109.2	0.062	34.5	103.7	76.6	73.0	0.246
CONV+{2HOG}	142.3	225.7	173.3	170.9	0.098	105.6	140.2	107.5	109.0	0.056	33.8	89.6	64.6	61.8	0.265
CONV+{2MMUL}	137.6	186.7	138.2	139.8	0.044	103.6	132.9	103.7	105.1	0.055	33.8	55.1	34.4	34.5	0.028
CONV+{2SD,CONV}	140.2	227.7	160.2	165.0	0.073	103.8	134.7	104.7	105.9	0.054	33.9	98.2	55.2	58.9	0.172
CONV+{2SD,HOG}	142.9	221.5	177.6	177.2	0.067	105.4	138.5	107.2	108.8	0.056	35.2	92.1	69.5	68.3	0.154
CONV+{2SD}	138.8	192.9	155.4	157.0	0.046	103.6	133.2	103.9	105.3	0.055	34.8	73.1	51.4	51.6	0.076
CONV+{2VADD}	137.5	173.5	137.9	139.6	0.046	103.3	132.8	103.3	104.8	0.057	34.1	58.8	34.5	34.6	0.037
CONV+{3CONV}	141.9	251.1	159.3	167.8	0.144	104.4	137.1	106.3	107.6	0.057	36.3	128.2	51.5	60.1	0.386
CONV+{3HOG}	143.0	253.0	192.3	187.5	0.125	105.8	141.3	107.6	109.4	0.061	34.0	126.3	84.4	77.9	0.301
CONV+{3MMUL}	139.5	187.6	140.4	142.0	0.044	105.2	136.8	105.8	107.3	0.056	33.8	44.1	34.5	34.5	0.011
CONV+{3SD}	140.2	211.1	170.9	171.2	0.045	104.0	142.2	104.2	105.6	0.055	34.6	85.0	65.9	65.5	0.076
CONV+{3VADD}	138.5	193.2	139.6	141.2	0.045	104.1	135.8	104.7	106.2	0.056	34.3	56.0	34.8	34.9	0.034
CONV+{CONV,HOG,SD}	141.7	232.7	169.2	169.4	0.082	104.6	137.9	106.3	107.9	0.058	35.8	104.2	61.9	61.4	0.201
CONV+{CONV,HOG}	142.4	227.7	163.4	164.8	0.093	105.2	138.7	107.1	108.7	0.058	34.0	93.2	55.9	56.0	0.247
CONV+{CONV,SD}	139.9	215.2	155.2	160.3	0.088	103.6	134.1	104.3	105.5	0.055	35.4	92.0	48.9	54.6	0.242
CONV+{CONV}	139.5	200.7	144.1	150.7	0.082	104.0	133.6	104.5	105.9	0.057	34.8	89.4	37.2	44.7	0.254
CONV+{HOG,SD}	141.7	208.0	164.8	162.4	0.071	105.2	137.7	106.9	108.5	0.056	34.3	84.9	56.9	53.8	0.182
CONV+{HOG}	140.4	196.3	154.8	155.2	0.063	103.8	138.3	106.4	107.8	0.057	33.7	75.2	47.6	47.2	0.180
CONV+{MMUL}	137.0	175.0	137.6	139.2	0.043	103.2	139.0	103.3	104.7	0.056	33.6	55.1	34.2	34.3	0.027
CONV+{SD}	137.7	183.0	146.2	146.3	0.044	103.3	132.6	103.5	104.9	0.055	34.0	62.5	42.6	41.4	0.066
CONV+{VADD}	138.4	187.4	139.0	140.6	0.045	104.2	133.4	104.2	105.7	0.055	34.1	56.8	34.6	34.8	0.035

Table 10: Table of runtimes (in ms) of "zero-copy" versions of the CONV benchmark under all scenarios, along with coefficients of variation. These measurements were performed on the TX1.

	Total time					Memory copy time					Kernel time				
Scenario	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV
HOG	233.9	265.8	235.3	236.3	0.017	49.7	78.7	50.3	51.0	0.075	183.5	211.7	184.9	185.2	0.007
HOG+{2CONV,HOG}	507.5	688.8	619.9	615.0	0.051	52.3	102.6	62.2	64.5	0.145	448.8	623.2	562.5	550.3	0.060
HOG+{2CONV,SD}	317.9	550.2	477.0	473.8	0.059	54.0	110.8	65.0	66.5	0.115	256.1	474.1	406.9	407.2	0.070
HOG+{2CONV}	303.5	432.6	374.5	368.9	0.061	51.7	104.2	61.4	62.4	0.136	250.9	348.9	318.2	306.4	0.082
HOG+{2HOG,CONV}	593.9	743.3	679.9	682.0	0.026	50.9	103.3	59.6	63.1	0.171	534.4	659.2	622.8	618.8	0.027
HOG+{2HOG,SD}	597.8	710.0	662.5	662.6	0.012	50.6	91.9	56.3	57.3	0.102	530.2	627.6	606.9	605.2	0.016
HOG+{2HOG}	437.5	601.2	568.2	567.7	0.013	49.5	89.1	50.7	52.1	0.095	373.1	518.7	517.4	515.5	0.015
HOG+{2MMUL}	235.7	273.3	237.2	238.1	0.017	49.4	78.8	49.7	50.3	0.074	186.0	196.0	187.5	187.6	0.004
HOG+{2SD,CONV}	388.9	526.8	448.1	450.9	0.039	53.2	102.5	60.8	62.3	0.118	326.4	445.1	386.7	388.4	0.040
HOG+{2SD,HOG}	529.1	626.4	576.8	578.0	0.023	50.0	92.5	53.7	56.6	0.128	472.5	555.5	523.6	521.2	0.023
HOG+{2SD}	310.2	395.1	344.2	344.3	0.027	50.9	87.6	52.1	54.9	0.096	251.0	315.6	289.2	289.4	0.029
HOG+{2VADD}	234.6	292.6	236.4	237.3	0.018	49.7	79.2	50.3	51.0	0.074	184.3	212.5	185.9	186.1	0.006
HOG+{3CONV}	343.9	620.4	511.1	511.8	0.089	55.3	121.7	72.2	73.6	0.145	288.1	543.6	440.6	438.0	0.103
HOG+{3HOG}	566.3	776.1	735.6	735.4	0.015	50.4	98.1	60.2	62.2	0.119	487.1	692.6	674.7	673.0	0.018
HOG+{3MMUL}	238.0	277.9	239.9	240.8	0.018	50.2	81.8	50.7	51.4	0.078	187.4	210.9	189.1	189.2	0.005
HOG+{3SD}	406.7	525.3	459.9	460.5	0.034	52.3	96.2	57.7	58.9	0.109	342.4	466.1	400.4	401.6	0.038
HOG+{3VADD}	234.9	287.9	239.7	240.5	0.019	50.2	81.0	51.2	51.8	0.071	184.0	199.2	188.3	188.4	0.010
HOG+{CONV,HOG,SD}	475.5	624.3	586.9	581.1	0.028	51.5	98.4	56.3	58.7	0.134	417.7	553.6	529.3	522.3	0.034
HOG+{CONV,HOG}	388.0	535.1	476.9	482.8	0.031	49.9	95.9	56.4	56.9	0.106	326.8	452.1	418.0	425.8	0.036
HOG+{CONV,SD}	298.6	390.4	336.0	338.1	0.047	51.3	99.6	59.3	60.5	0.122	239.2	335.6	271.1	277.6	0.061
HOG+{CONV}	265.9	343.1	289.8	286.8	0.052	50.2	90.0	52.7	54.5	0.094	215.1	268.1	230.7	232.2	0.065
HOG+{HOG,SD}	409.2	505.4	468.1	468.3	0.016	50.2	89.6	52.5	54.3	0.098	352.8	430.9	413.8	413.8	0.018
HOG+{HOG}	305.9	427.6	394.9	394.8	0.011	49.3	82.0	50.5	51.0	0.055	247.7	357.0	344.3	343.6	0.012
HOG+{MMUL}	234.3	267.3	235.9	236.9	0.018	49.5	78.6	49.7	50.4	0.079	184.6	210.0	186.1	186.3	0.007
HOG+{SD}	248.6	310.0	274.2	276.3	0.020	50.0	85.0	50.5	52.1	0.076	194.5	248.1	223.2	224.1	0.019
HOG+{VADD}	234.3	270.1	236.5	237.4	0.018	49.7	83.2	50.3	51.0	0.078	184.1	204.8	186.1	186.3	0.006

Table 11: Table of runtimes (in ms) of "zero-copy" versions of the HOG benchmark under all scenarios, along with coefficients of variation. These measurements were performed on the TX1.

	Total time					Memory copy time					Kernel time				
Scenario	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV	Min	Max	Median	Mean	CV
SD	29.0	57.7	29.1	29.6	0.072	20.8	49.2	20.9	21.4	0.098	8.0	17.4	8.1	8.1	0.014
SD+{2CONV,HOG}	29.6	89.8	43.9	44.0	0.153	21.5	54.1	26.9	27.3	0.108	8.1	34.8	16.8	16.6	0.311
SD+{2CONV,SD}	30.0	73.8	38.6	38.7	0.175	21.6	53.1	24.8	25.1	0.125	8.1	35.4	14.5	13.5	0.363
SD+{2CONV}	29.7	79.9	34.9	37.0	0.198	21.4	53.3	24.1	24.7	0.129	8.1	29.4	8.2	12.2	0.434
SD+{2HOG,CONV}	29.2	91.5	43.9	44.5	0.173	21.0	55.1	26.8	27.0	0.114	8.0	33.5	17.3	17.4	0.338
SD+{2HOG,SD}	29.0	76.7	40.5	40.8	0.154	20.9	50.6	24.8	25.3	0.117	8.1	33.0	15.3	15.4	0.303
SD+{2HOG}	28.9	70.5	40.1	39.8	0.142	20.7	52.2	25.1	25.2	0.108	8.1	25.6	14.6	14.5	0.297
SD+{2MMUL}	28.9	84.6	29.2	29.5	0.078	20.6	46.3	20.9	21.1	0.090	8.1	10.4	8.2	8.2	0.010
SD+{2SD,CONV}	29.6	78.2	35.8	37.0	0.181	21.2	50.2	22.3	23.7	0.122	8.1	32.5	12.4	13.2	0.367
SD+{2SD,HOG}	29.1	68.8	36.6	37.1	0.129	20.8	51.8	23.0	23.7	0.113	8.1	31.7	13.2	13.3	0.251
SD+{2SD}	28.9	60.5	32.7	33.4	0.121	20.8	50.2	21.3	21.8	0.099	8.0	24.3	10.6	11.5	0.290
SD+{2VADD}	29.0	86.0	29.1	29.5	0.075	20.9	46.4	20.9	21.3	0.090	8.1	19.2	8.1	8.1	0.015
SD+{3CONV}	30.7	88.5	42.4	42.8	0.192	22.2	56.0	28.0	27.8	0.127	8.1	36.7	14.9	14.9	0.408
SD+{3HOG}	29.1	82.9	45.6	45.0	0.170	21.0	57.3	26.9	26.9	0.115	8.1	33.5	18.4	18.0	0.328
SD+{3MMUL}	29.1	82.9	29.6	29.9	0.073	20.8	43.1	21.1	21.4	0.084	8.1	11.4	8.3	8.3	0.012
SD+{3SD}	29.2	69.7	35.3	35.2	0.128	20.9	49.5	21.6	22.3	0.104	8.1	30.0	12.6	12.8	0.275
SD+{3VADD}	29.1	85.7	29.3	29.6	0.069	20.9	44.7	21.1	21.4	0.075	8.1	12.7	8.1	8.1	0.010
SD+{CONV,HOG,SD}	29.3	77.6	37.9	39.7	0.169	20.8	55.1	24.3	25.1	0.119	8.1	32.4	13.6	14.5	0.330
SD+{CONV,HOG}	29.1	77.9	37.2	38.4	0.146	20.9	50.6	24.7	24.9	0.103	8.1	27.4	12.7	13.4	0.315
SD+{CONV,SD}	29.2	71.9	32.2	34.4	0.159	21.0	48.9	21.7	22.9	0.111	8.0	26.2	8.7	11.4	0.350
SD+{CONV}	29.1	64.7	29.6	32.4	0.135	20.8	49.5	21.4	22.5	0.107	8.0	18.0	8.1	9.8	0.316
SD+{HOG,SD}	28.9	66.3	34.8	35.4	0.116	20.6	48.6	22.8	23.3	0.106	8.1	24.4	11.8	12.0	0.234
SD+{HOG}	28.7	60.3	34.4	34.3	0.104	20.5	48.3	23.0	23.1	0.101	8.0	16.8	10.9	11.0	0.212
SD+{MMUL}	28.9	74.8	29.2	29.6	0.073	20.7	52.1	20.9	21.2	0.086	8.1	15.4	8.2	8.2	0.013
SD+{SD}	28.8	59.1	29.4	31.3	0.106	20.6	45.5	21.1	21.3	0.085	8.0	16.5	8.1	9.9	0.272
SD+{VADD}	28.9	85.4	29.2	29.5	0.076	20.7	45.7	21.0	21.3	0.087	8.0	10.0	8.1	8.1	0.007

Table 12: Table of runtimes (in ms) of "zero-copy" versions of the SD benchmark under all scenarios, along with coefficients of variation. These measurements were performed on the TX1.