Real-Time System-Level Implementation of a Telepresence Robot Using an Embedded GPU Platform

Muhammad Teguh Satria*, Swathi Gurumani*, Wang Zheng†, Keng Peng Tee†, Augustine Koh*, Pan Yu†,
Kyle Rupnow*, Deming Chen*
*Advanced Digital Sciences Center, Singapore †Institute for Infocomm Research, Singapore
Email: {*t.satria, swathi.g, augustine.k, k.rupnow, dchen*}*@adsc.com.sg*, {*zhwang, kptee, ypan*}*@i2r.a-star.edu.sg*

***Abstract*—Real-time applications such as telepresence systems present an opportunity to use embedded GPUs for compute acceleration to meet platform goals. In this paper, we develop a prototype of a portable, standalone telepresence robot that performs real-time attention-directed control using an NVIDIA Jetson TK1 embedded platform. We perform platform-specific optimizations to improve thread occupancy, optimize computation workload and improve accuracy of face detection on the embedded GPU and achieve real-time performance of 30 frames per second on the Jetson TK1 and an overall speedup of 10x compared to the ARM CPU version.**

I. Introduction

Telepresence systems extend video-conference systems through the integration of motors to direct camera motion and focus, guided by audiovisual or user-input. Attention-directed telepresence robot systems [1] make remote interaction natural by automating control using audiovisual information. However, this interactive behavior requires use of complex audiovisual algorithms and thus necessitates higher computing capabilities to achieve real-time performance.

Recent generations of embedded platforms have integrated low-power GPUs for general-purpose GPU (GPGPU) compute acceleration [2]–[4]. Despite the similar interfaces and programming environments between desktop-GPUs and embedded GPUs, new development for embedded GPUs and adapting existing GPU implementations onto embedded GPUs remains challenging for several reasons. First, embedded GPGPUs typically have fewer computation cores, reduced shared memory resources, and lower clock frequency than desktop GPUs. Second, though domain-specific libraries such as OpenCV [5] provide pre-designed and optimized GPU implementations, they are generally optimized for specific desktop-class GPU platforms. Importantly, the existing libraries may not consider qualitative performance aspects in algorithms such as face detection where detection accuracy and consistency are just as important as execution latency. Finally, although *best-effort* speedup is sufficient for high-end GPUs, real-time applications must process data at the input data rate (e.g. images from a camera) to meet platform goals. Several applications have been developed for embedded GPGPU architectures [6]–[9], thus emphasizing that such platforms can provide performance benefits with good energy efficiency. Prior works [7], [8] have discussed achieving real-time performance on embedded GPUs, but did not discuss qualitative and quantitative aspects.

In this paper, we develop a prototype of a portable telepresence robot that performs real-time attention-directed control using an NVIDIA Jetson TK1 embedded platform. The prior prototype telepresence system relies on a desktop-based computing platform for running the audiovisual algorithms. We select the compute-intensive vision component for compute-acceleration and develop an efficient embedded GPU implementation of Haar-classifier based face detection. We overcome the challenge of limited resources in embedded GPU by performing both algorithmic and platform-specific implementation optimizations: we improve workload distribution to reduce idle threads and achieve increased thread occupancy; optimize computation workload by utilizing a sufficient image resolution; and optimize the filtering algorithm by using a history based threshold for improved face detection. We also port the other algorithms of the application onto the ARM CPU and demonstrate a standalone system-level implementation of the telepresence robot that meets our design goals. Additionally, we also perform our experiments on a desktop GPU platform, and an embedded GPU prototype board (Kayla) to demonstrate trade-offs among different platforms.

The main contributions of this paper are:

- A GPU implementation of face detection that optimizes computation workload and improves workload distribution to overcome resource limitations in embedded GPUs.
- An embedded GPU implementation of face detection that achieves real-time performance of 30 fps on Jetson TK1 and 10x speedup compared to the ARM CPU.
- A system-level prototype implementation of a portable, standalone attention-directed telepresence robot using the Jetson TK1 embedded platform.

II. Telepresence Robot

A prototype of the prior telepresence robot using laptop computing resources along with a high-level block diagram of the control is shown in Fig. 1. The robotics components consist of a pan-tilt unit with a camera and tablet as a display, a microphone array, control circuitry for the motor and audio interface and desktop-based compute resources. During operation, the tablet display and camera moves to face the person of interest based on face detection information from the camera, and 3D sound localization data from the microphone.

978-3-9815370-6-2/DATE16/

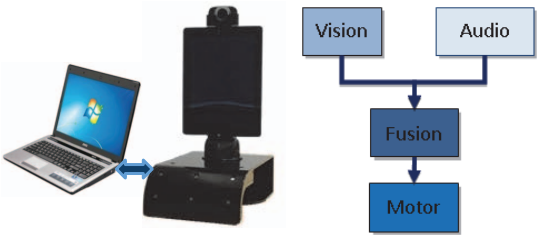

Fig. 1. Prototype and schematic overview of the telepresence robot

The audio module is a sound localization sub-system based on Voice Activity Detector (VAD) [10]. The vision module is a face-detection algorithm based on OpenCV Haar-Based Cascade Classifier [5]. The fusion module is responsible for receiving audio and vision information and computing whether the system attention requires movement of the camera. When the audio and vision information target similar locations, the vision system is given priority due to higher fine-grained accuracy; when they are different, the audio is given priority to focus attention on people currently speaking.

III. GPU-Accelerated Face Detection

The telepresence system as described in Section II was initially developed in a CPU-only implementation on an Intel Core i5 system. The initial prototype's dependence on external computing resources not only limited the portability of the robot, but is neither cost-effective nor power-efficient. The final goal for a commercially viable telepresence robot is execution on an SoC platform either as part of the robotics platform or controlled by the compute resources of the tablet. For prototyping, we choose the NVIDIA Jetson TK1 embedded platform [2] that contains the Tegra K1 SoC with quad-core ARM Cortex A15 CPU and Kepler GPU with 192 cores.

A. Prototyping Platforms

In addition to Jetson TK1, we also use two other platforms to prototype the system: CUDA development on desktop GPUs, and the NVIDIA Kayla platform [11]. The Kayla platform is an ARM development platform for CUDA and OpenCL. It contains an ARM Cortex A9 quad-core and a PCIe CPU-GPU interface to connect desktop-class GPUs. We use GTX770 and GTX650 desktop GPUs; in results we refer to them as 770 and 650 for brevity. We compare results from all three platforms and demonstrate the impact of our optimizations on different architectures. Also, we use a power meter to report power measurements on the target Jetson TK1.

B. CPU/GPU Partitioning

First, to make CPU/GPU decisions, we must perform application profiling to determine the acceleration candidates. After a system-level analysis of the telepresence robot, we identify face detection as the primary acceleration candidate. Thus we map the classifier computation to the GPU and port the audio and fusion components onto the ARM CPU. The existing face-detection implementation on the CPU is based on the Open Source Computer Vision (OpenCV version 2.4.9)

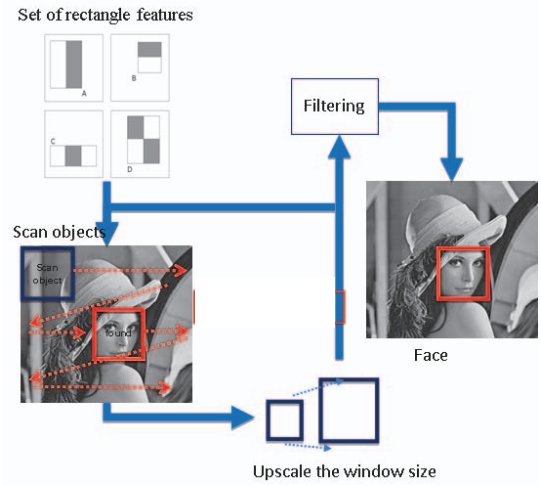

Fig. 2. Face detection algorithm

library implementation of a Haar-Based Cascade Classifier [5]. A Haar-classifier scans the image with a window size (e.g. 20x20) to search for a set of (rectangular) features and saves the regions with high probability of object detection. The scanning process is repeated by improving the window size until the window size becomes larger than the image. Subsequently, the saved regions with high-confidence object detection are filtered to determine if there is a face. This process is illustrated in Fig. 2. The face detection application is required to meet certain specific performance goals.

C. Performance Goals

Qualitative goal: The input image resolution directly corresponds to the maximal distance from the camera at which a face can be detected. In our case, the robot is expected to operate in a typical meeting room; thus, we wish to detect faces up to 3m away from the camera, with high accuracy and stability to prevent rapid modifications in robot guidance.

Quantitative goal: The platform must deliver video at 30 frames per second. Given the qualitative goal of face detection at up to 3m, we need a video resolution of at least 320x240, which corresponds to an average-sized face filling the smallest window size at a distance of 3m.

D. Performance Evaluation

The performance of the CPU version of the face detection algorithm is shown in Table I. We can observe that the desktop CPU (i5) implementation is performance limited for a 320x240 resolution while the ARM implementations are performance limited for both resolutions. But the accuracy is 100% with no detection failures. To verify accuracy, we use a fixed input video file that contains 580 frames. We now begin evaluating GPU acceleration of the application.

E. GPU Implementation and Evaluation

OpenCV also provides a GPU implementation of the Haar-Based Classifier. However, although the GPU implementation is conceptually identical to the CPU implementation, it is

TABLE I
PERFORMANCE OF OPENCV-CPU FACE DETECTION

Accuracy (Number of frames with detection failures in 580 frames)			
	i5 CPU	Kayla ARMv7 rev 9	Jetson ARMv7 rev 3
160x120	0	0	0
320x240	0	0	0
Performance (Frames per Second)			
160x120	49	7	18
320x240	9	1	3

TABLE II
PERFORMANCE OF OPENCV-GPU FACE DETECTION

Accuracy (Number of frames with detection failures in 580 frames)					
	i5+ 770	i5+ 650	Kayla ARM+ 770	Kayla ARM+ 650	Jetson ARM+ TK1
160x120	63	63	63	63	63
320x240	3	3	3	3	3
Performance (Frames per Second)					
160x120	207	185	121	103	42
320x240	113	59	53	38	20

not functionally equivalent. Using the GPU-based OpenCV implementation, the algorithm increased detection failures, particularly for people farther from the camera. Table II presents the performance of OpenCV-GPU face detection on different GPU platforms. Though all the GPU platforms have achieved improved performance compared to the CPU version, the accuracy of the implementation has suffered. In addition to the detection failures, the performance on Jetson TK1 did not meet the quantitative goal of 30 frames per second using this implementation. Further analysis determined that instead of increasing the window size and searching on the same image resolution, the GPU implementation down-scales the image resolution while keeping the window size constant. Although conceptually these are identical, downscaling reduces detection accuracy. By the time the image size is scaled down to the initial search window size, the resolution is too low to detect faces within that window size.

Due to the qualitative differences in the GPU OpenCV Haar-classifier, as a first step, we reimplement the GPU algorithm following the original CPU strategy to meet our qualitative goals in maximal detection distance. Our strategy is to parallelize the search across all window sizes and also concurrently perform searching of all regions in a given window size. Thus all regions for all windows sizes are parallelized as illustrated in Fig. 3. We show four window sizes, where each window size has specific number of regions (the larger the window size, the smaller number of regions as shown for i=3), and each region is handled by a thread. The number of threads (and computations) is larger for the smaller window size as shown for i=0 and thus we can observe that it is computationally expensive to search for smaller sized faces.

The results of this implementation are shown in Table III. The average power consumption is 6.8w (7.6w peak) for 320x240 resolution and 6.6w (6.8w peak) for 160x120 resolution. We achieve 100% accuracy in face detection, the primary goal of this implementation. However, the 320x240 resolution is still performance-limited for GTX650 and Jetson TK1 architectures. Also, on the Kayla platforms, the frame rate is less than desktop frame rate, even with identical GPUs. The difference is attributed to the limitation in Kayla platform bandwidth. With the limited resources in Jetson TK1, the current performance is only 8 frames per second and needs 4x improvement to achieve real-time performance.

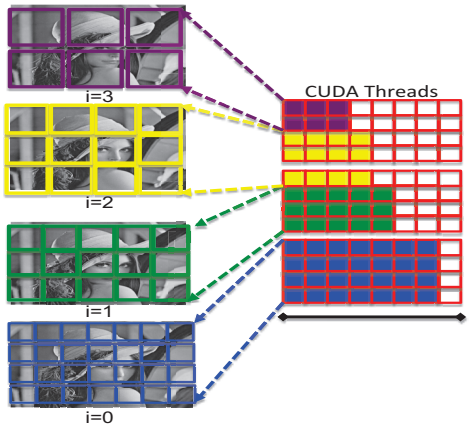

Fig. 3. GPU Implementation matching CPU Accuracy

TABLE III
IMPROVED ACCURACY GPU IMPLEMENTATION

Accuracy (Number of frames with detection failures in 580 frames)					
	i5+ 770	i5+ 650	Kayla ARM+ 770	Kayla ARM+ 650	Jetson ARM+ TK1
160x120	0	0	0	0	0
320x240	0	0	0	0	0
Performance (Frames per Second)					
160x120	159	67	109	62	32
320x240	58	26	33	14	8

IV. PLATFORM-SPECIFIC OPTIMIZATION

We overcome the challenge of limited resources in the embedded GPU by performing both algorithmic and platform-specific optimizations.

1) Improving Achieved Occupancy on Jetson TK1: The performance of desktop GPUs is not necessarily sensitive to achieved occupancy: a measure of the number of parallel threads compared to the allowable maximum parallel threads. However, embedded GPUs have limited resources, and low occupancy also corresponds to under-utilized GPU resources. The GPU profiler highlighted that the achieved occupancy of Jetson TK1 is only 39% (160x120) and 57% (320x240). We analyze and identify that one source of low occupancy is that as the window size increases, there are fewer regions to distribute among threads (as shown in Fig. 3). Furthermore, although the initial workload distribution is simple, it leaves threads unused. We optimize the workload distribution to reduce the number of idle threads. We also optimize the mapping between threadIds and workload data and thus reduce

both unused resources and total number of threadblocks. This increased occupancy to 74% (160x120) and 80% (320x240), for an average improvement of 1.64x.

2) Image Resizing for Computation Reduction: Despite efficient resource usage, computational workload still remains significantly high, limiting performance. The number of regions and total workload is highest for the smallest window size, but this is required to detect faces farther away from the camera and meet our qualitative goal. But as the window size increases beyond a certain size, there is less need to compute at the highest resolution. At the larger window sizes, we are now only looking for faces closer to the camera. Thus, to reduce workload, after reaching a certain window size, we down-scale the input image by half (and convert 320x240 to 160x120 resolution) to reduce workload without reducing accuracy. Unlike the OpenCV-GPU, we do not continue to down-scale the image after every window size and thus exploit the computational benefits of using a smaller resolution only when we are able to detect faces up to 3m.

Although we were able to detect faces at the desired distance, the algorithm was more sensitive to minor variations in rotation and orientation of faces after switching from the fixed input video to a live webcam feed. In a Haar-based classifier, this behavior does not signify total detection failure, just that cumulative weight of detected features is close to the object detection threshold. For each search region, the sum of detected feature weights is compared against a threshold to separate the faces from non-faces.

3) Improved Filtering Algorithm: In order to optimize the qualitative consistency without eliminating achievements in quantitative goals, we optimize the filtering algorithm in the Haar-classifier. We use a detection-history based threshold weighting to help ensure that faces are consistently detected. First, we record all the detected faces in the current frame; then, in the next frame, we compare detected objects to the location of faces in the previous frame. Objects in close proximity to prior detected faces are given increased weight. Thus, faces previously detected are more likely to continue being detected despite minor variations.

The quantitative performance of our final implementation is presented in Table IV. We achieve our implementation goal of 30 frames per second for the 320x240 resolution and detect all faces within 3m distance. Thus we achieve a speedup of 10x compared to the Jetson ARM CPU only version. The average power consumption on the Jetson TK1 is 3.2w (3.3w peak) for both resolutions. The workload reorganization and reduction in threadblocks reduced the power consumption by 50% for both resolutions. We show the telepresence robot connected to the Jetson TK1 running our real-time face detection implementation in Fig. 4. Our work with the telepresence robot makes it portable and power efficient, thus more practical for potential commercial use.

V. Conclusions

We developed a prototype of a portable, standalone telepresence robot that performs real-time attention-directed control using an NVIDIA Jetson TK1 embedded platform. We developed platform-specific optimizations for improving thread occupancy, computation reduction, and accuracy of face detection and thus achieved real-time performance of 30 frames per second for face detection and an overall speedup of 10x speedup compared to the ARM CPU version.

Fig. 4. Telepresence robot running real-time face detection on Jetson TK1

TABLE IV
Final performance of GPU implementation

Performance (Frames per Second)					
	i5+ 770	i5+ 650	Kayla ARM+ 770	Kayla ARM+ 650	Jetson ARM+ TK1
160x120	66	56	75	63	58
320x240	62	40	69	44	30

VI. Acknowledgement

This study is supported in part by the research grant for the Human-Centered Cyber-physical Systems Programme at the Advanced Digital Sciences Center from Singapore's Agency for Science, Technology and Research (A*STAR).

References

[1] R. Yan, K. P. Tee, Y. Chua, Z. Huang, and H. Li, "An attention-directed robot for social telepresence," in *Human-Agent Interaction, The 1st International Conference on*, 2013.

[2] *NVIDIA Jetson TK1 Embedded Development Kit*, http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html.

[3] "Mali graphics hardware," www.arm.com/products/multimedia/mali-graphics-hardware/index.php.

[4] *Vivante Graphics Cores Product Brief*, www.vivantecorp.com/Product_Brief.pdf.

[5] "Open source computer vision opencv," http://opencv.org/.

[6] A. Maghazeh, U. Bordoloi, P. Eles, and Z. Peng, "General purpose computing on low-power embedded gpus: Has it come of age?" in *SAMOS XIII*, July 2013, pp. 1–10.

[7] M. Nitsche and P. De Cristforis, "Real-time on-board image processing using an embedded gpu for monocular vision-based navigation," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, 2012, vol. 7441, pp. 591–598.

[8] S. Yi, I. Yoon, C. Oh, and Y. Yi, "Real-time integrated face detection and recognition on embedded gpgpus," in *Embedded Systems for Real-time Multimedia (ESTIMedia), 2014 IEEE 12th Symposium on*, Oct 2014, pp. 98–107.

[9] G. Wang, Y. Xiong, J. Yun, and J. Cavallaro, "Accelerating computer vision algorithms using opencl framework on the mobile gpu - a case study," in *ICASSP*, May 2013, pp. 2629–2633.

[10] E. Chng, "A microphone array with a 3-dimensional configuration for the I2R social robot," Institute for Infocomm Research, A*STAR, Tech. Rep., 2012.

[11] *NVIDIA Kayla Platform*, https://developer.nvidia.com/content/kayla-platform.