Comparing Two Generations of Embedded GPUs Running a Feature Detection Algorithm

Max Danielsson, Håkan Grahn, and Thomas Sievert
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden
max@autious.net, {hakan.grahn,thomas.sievert}@bth.se

Jim Rasmusson
Sony Mobile Communications AB
SE-221 88 Lund, Sweden
jim.rasmusson@sony.com

***Abstract*—Graphics processing units (GPUs) in embedded mobile platforms are reaching performance levels where they may be useful for computer vision applications. We compare two generations of embedded GPUs for mobile devices when running a state-of-the-art feature detection algorithm, i.e., Harris-Hessian/FREAK. We compare architectural differences, execution time, temperature, and frequency on Sony Xperia Z3 and Sony Xperia XZ mobile devices. Our results indicate that the performance soon is sufficient for real-time feature detection, the GPUs have no temperature problems, and support for large work-groups is important.**

***Index Terms*—Graphics Processing Unit, Mobile Embedded GPU, Computer Vision, Performance Evaluation, Temperature Measurements**

I. Introduction

Today's cellphones have very powerful CPUs and embedded graphics processing units (GPUs) built into them. For example, the Sony Xperia Z3 [17] has a 2.5 GHz quadcore CPU and a 128 core Adreno 330 GPU. This enables performance-demanding applications to migrate from desktop to mobile platforms.

Digital images play a large role in how we communicate with each other. As contemporary cellphones are equipped with high-resolution digital cameras, the need for advanced and powerful image processing capabilities has emerged on mobile phones. One such application domain is computer vision, which includes, e.g., feature detection, object detection and recognition, and pattern matching.

Many feature detection algorithms and feature descriptors have been proposed, e.g., SIFT [11], SURF [4], [3], BRIEF [5], BRISK [10], and ORB [15]. Further, work have been done on developing such algorithms for GPUs, e.g., SIFT on desktop GPUs using CUDA [2], [20]. For mobile GPUs, attempts have been done using OpenGL ES 2.0 [14], [9]. However, evaluation was only done using very small images in [14] (320x240 pixels), while no evaluation was done in [9]. In [6], we presented a novel feature detection/description algorithm targeting mobile embedded devices, called Harris-Hessian/FREAK, based on a Harris-Hessian feature detector [18] and a FREAK feature descriptor [1].

The main questions addressed in this study are: (i) How has embedded GPUs evolved the past two years, from the perspective of running a state-of-the-art feature detection algorithm? (ii) How are the temperature and frequency behavior of the mobile GPUs when running such algorithms?

In this study, we have evaluated two generations of embedded GPUs, i.e., the Adreno 330 (in the Sony Zperia Z3) and the Adreno 530 (in the Sony Xperia XZ), when running a Harris-Hessian/FREAK feature detection algorithm. Our evaluation shows that the performance has increased a factor of ten over two generations, mainly due to more GPU cores and support for larger work-group sizes. Further, the newer GPU was much more performance sensitive to the work-group size. Finally, we have observed that the GPUs can run at their maximum clock frequencies for long periods of time, without any thermal problems or need to reduce the clock frequency.

II. Background and Related Work

Computer vision is a wide field with applications including, e.g., object recognition, image restoration and scene reconstruction. In computer vision, *feature detection* refers to methods of trying to locate arbitrary features that can afterwards be described and compared. These features then need to be described in such a manner that the same feature in a different image can be compared and confirmed to be matching. Typically, areas around the chosen keypoint are sampled and then compiled into a vector, a so called *feature descriptor*.

A. Feature Detection

Scale-Invariant Feature Transform (SIFT) [11] was proposed in 1999, and has become somewhat of an industry standard. It includes both a detector and a descriptor. The detector is based on calculating a Difference of Gaussians (DoG) with several scale spaces.

Partially inspired by SIFT, the Speeded-Up Robust Features (SURF) [4], [3] detector was proposed, which uses integral images and Hessian determinants. SURF and SIFT are often used as base lines in evaluations of other detectors.

The detector chosen for our experiments was proposed by Xie et al. in [18] and is inspired by Mikolajczyk and Schmid [12], particularly their use of a multi-scale Harris operator. However, instead of increasing the scale incrementally, they examined a large set of pictures to determine which scales should be evaluated so that as many features as possible only are discovered in one scale each. Then, weak corners are culled

using the Hessian determinant. As the fundamental operators are the Harris operator and the Hessian determinant, it is called the "Harris-Hessian detector".

B. Feature Description

SIFT, SURF, and many other descriptors use strategies that are variations of histograms of gradients (HOG). The area around each keypoint in an image is divided into a grid with sub-cells. For each sub-cell, a gradient is computed. Then, a histogram of the gradients' rotations and orientations is made for each cell. These histogram then make up the descriptor. SURF, while based on the same principle, uses Haar wavelets instead of gradients. The resulting descriptor vectors of a high dimension (usually $\geqslant$128) which can be compared using, e.g., Euclidean distance.

Calonder et al. proposed a new type of descriptor called Binary Robust Independent Elementary Features (BRIEF) [5]. Instead of using HOGs, BRIEF samples a pair of points at a time around the keypoint, then compares their respective intensities. The result is a number of ones and zeros that are concatenated into a string, i.e., forming a "binary descriptor". They do not propose a single sampling pattern, rather they consider five different ones. The resulting descriptor is nevertheless a binary string. The benefit of binary descriptors is mainly that they are computationally cheap, as well as suitable for comparison using Hamming distance [7], which can be implemented effectively using the XOR operation.

Further work into improving the sampling pattern of a binary descriptor has been made, most notably Oriented FAST and Rotated BRIEF (ORB) [15], Binary Robust Invariant Scalable Keypoints (BRISK) [10], and Fast Retina Keypoint (FREAK) [1].

The descriptor we use in this paper is FREAK [1], where machine learning is used to find a sampling pattern that aims to minimize the number of comparisons needed. FREAK generates a hierarchical descriptor allowing early out comparisons. As FREAK significantly reduces the number of necessary compare operations, it is suitable for mobile platforms with low compute power.

III. Harris-Hessian/FREAK

We use the Harris-Hessian/FREAK algorithm [6], based on a combination of the Harris-Hessian detector [18] and the FREAK binary descriptor [1], as a representative feature detection algorithm targeting mobile devices.

A. The Harris-Hessian Detector

The Harris-Hessian detector was proposed by Xie et al. [18] and is essentially a variation of the Harris-Affine detector combined with a use of the Hessian determinant to cull away "bad" keypoints. The detector consists of two steps: Discovering Harris corners [8] using the Harris-affine-like [12] detector on nine pre-selected scales as well as two additional scales surrounding the most populated one, then culling weak points using a measure derived from the Hessian determinant.

The Harris step finds Harris corners by applying a Gaussian filter at gradually larger σ, then reexamines the scales around the σ where the largest number of corners were found. This σ is said to be the characteristic scale of the image. After all the scales have been explored, the resulting corners make up the scale space, S.

In the Hessian step, the Hessian determinant for each discovered corner in S is evaluated in all scales. If the determinant reaches a local maximum at σ_i compared to the neighboring scales σ_{i-1} and σ_{i+1} and is larger than a threshold T, it qualifies as a keypoint of scale σ_i. Otherwise, it is discarded. The purpose of the Hessian step is to both reduce false detection and confirm the scales of the keypoints.

B. FREAK

FREAK (Fast Retina Keypoint) is a so called "binary" descriptor, since its information is represented as a bit string. Alahi et al. [1] propose a circular sampling pattern of overlapping areas inspired by the human retina. They then—optionally—define 45 pairs using these areas and examines their gradients, to estimate the orientation of the keypoint. With the new orientation, the pattern is rotated accordingly and areas are re-sampled. They use machine learning to establish which pairs of areas result in the highest performance for the descriptor bit string. The sampling pairs are sorted into four cascades with 128 pairs each, starting with coarse (faraway) areas and successively becoming finer and finer. This finally results in a bit string with 512 elements.

IV. Implementation

A more detailed description of our implementation is found in [6], so we only provide a high-level description here. Our implementation is written in standard C99 and OpenCL 1.1 [13], and compiled, built and installed using the Android SDK and NDK toolsets. Additionally, we utilize `stbi_image`[1] and `lodepng`[2] for image decoding/encoding, `ieeehalfprecision`[3] for half-float encoding, and Android Java to create an application wrapper.

All calculations are done in a raster data format, and we maintain the same resolution as the original image. We convert the image to grey scale as the algorithms do not account for color. We normalize and represent scalar pixel values as floating point values in the range of 0.0 to 1.0.

A. Algorithm Overview

The program is executed in a number of steps, see Fig. 1, starting with setting up buffers, loading image data, and decoding it into a raw raster format. The image is transferred to the device before execution of Harris-Hessian and desaturation is performed on the GPU as a separate step.

[1] Sean Barret, http://nothings.org/

[2] Lode Vandevenne, http://lodev.org/lodepng/

[3] Developed by James Tursa.

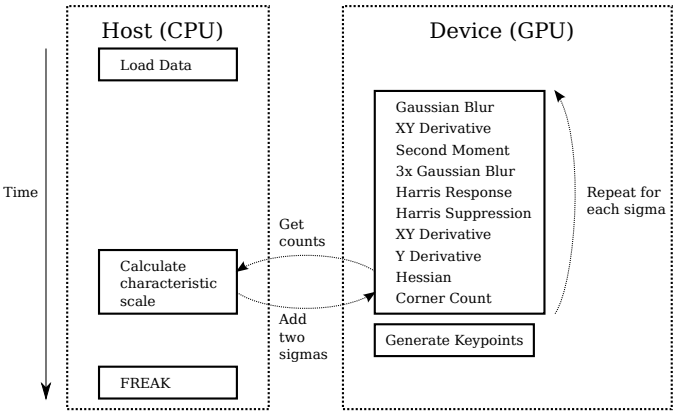

Fig. 1. Visual representation of the algorithm. On the left side is the host CPU with initialization of data, keypoint counts, and execution of FREAK. On the right is the twelve executional kernel calls to perform Harris-Hessian for a given scale and finally the keypoint generation kernel call which gathers the resulting data. Execution order is from top to bottom.

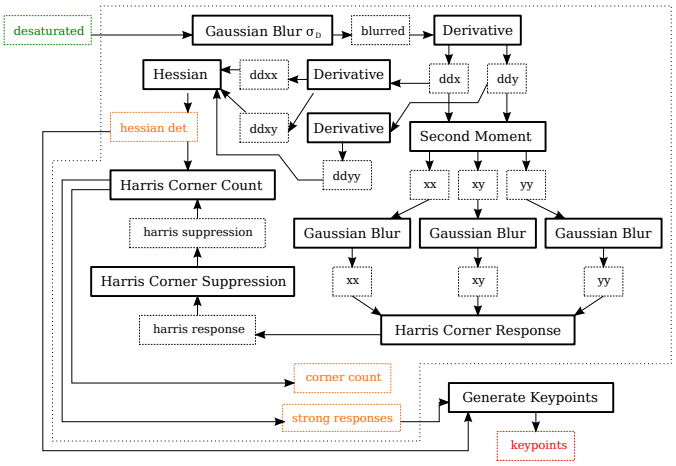

Fig. 2. Data flow in Harris-Hessian. Solid boxes indicate kernel executions and the dotted boxes are buffers or data. Green boxes are input and orange are the resulting output for a given sigma. Red boxes are the results sent to the descriptor. The larger dotted border indicates sigma iteration, anything within this border is executed for each sigma.

B. Harris-Hessian

The implementation is split into two main parts: the Harris-Hessian detector and the FREAK descriptor. Fig. 2 shows an overview of our implementation of the Harris-Hessian detector. Our implementation is targeted for GPU execution, and based on the description in [19]. Harris-Hessian is first executed for the sigmas $0.7, 2, 4, 6, 8, 12, 16, 20, 24$. For each sigma, the number of corners are counted and the sums are transferred to the CPU, which then calculates the characteristic sigma. After the characteristic sigma σ_c is found, we run the Harris-Hessian two more times for $\frac{\sigma_c}{\sqrt{2}}$ and $\sigma_c \cdot \sqrt{2}$.

A majority of GPU execution is spent in the Gaussian blur kernels. A $\sigma = 20$ results in a 121 elements wide filter, i.e., $121 * 2$ global memory accesses per task which is significant compared to all other kernels. Therefore, we use prefetching in the Gaussian kernel, i.e., preloading the global memory into local work-group shared memory. For a work-group (8 by 4 tasks) running the x axis Gaussian kernel, we perform a global to local memory fetch of $(60 + 8 + 60) * 4$ elements and then access the shared local memory from each task.

After running Harris-Hessian, we generate a list of keypoints containing the sigma and coordinates. The keypoints are passed to the FREAK algorithm together with the source image. FREAK then calculates a 512-bit descriptor for each keypoint, which is written to an external file.

C. FREAK

The FREAK implementation runs on the host CPU and is based on the implementation in [1][4]. The main differences in our implementation compared to the origial [1] are: we do not utilize SIMD instructions, we always take rotational or scale invariance into account, and we only use a generated and hard-coded sampling pattern.

[4]Source can be found at https://github.com/kikohs/freak

V. Experimental Methodology

Our experiment and measurements were conducted on a Sony Xperia Z3 [17] and on a Sony Xperia XZ [16]. Table I summarizes the main hardware characteristics of the two phones. The presented execution times are the mean of ten runs. The CPU and GPU temperature and frequency measurements were done using internal probes on the chipset. When running the temperature tests the phone was placed on a table, standing up with the back leaning towards a surface touching a small part of the phone. The room's temperature was around 20 °C.

TABLE I
Hardware characteristics of Sony Xperia Z3 and Sony Xperia XZ.

	Xperia Z3 [17]	Xperia XZ [16]
Release date	Sep./Oct. 2014	Oct. 2016
Chipset	Snapdragon 801	Snapdragon 820
CPU	Krait 400	Kryo
CPU cores	4	4
CPU frequency	2.5 GHz	2.15 GHz
GPU	Adreno 330	Adreno 530
GPU cores	128	256
GPU frequency	450/550/578 MHz	510/624/650 MHz
Main memory	3 GB	3 GB
Flash memory	16 GB	32 GB

As input in our experiments, we use the image shown in Fig. 3. The image content has little effect on Harris-Hessian algorithms. However, it has an impact on FREAK, since different images have different numbers of keypoints and FREAK scales linearly with the number of descriptors. We have not set any limitations on the number of descriptors encoded, which is relevant in a final implementation as it affects both the execution time and the storage requirements for the descriptor.

Fig. 3. Our test image, 800x600 pixels, featuring a series of posters.

VI. Experimental Results

A. Kernel Execution Times

In Fig. 4, we present the execution times for the different GPU kernels running on Xperia Z3 (upper) and Xperia XZ (lower). We present the mean time of ten executions, however, the times varied very little between the runs.

Our main observation in Fig. 4 is that the total time for the GPU kernels was reduced by a factor of ten, i.e., from almost 5700 ms to 550 ms, when moving from the Z3 to the XZ. Clearly, the reason is not only twice as many GPU cores on XZ, see Table I. We identified that one main reason is that the GPU in the XZ supports larger work-group sizes (up to 1024 vs. 256). Therefore, we evaluated how various workgroup sizes impact the performance.

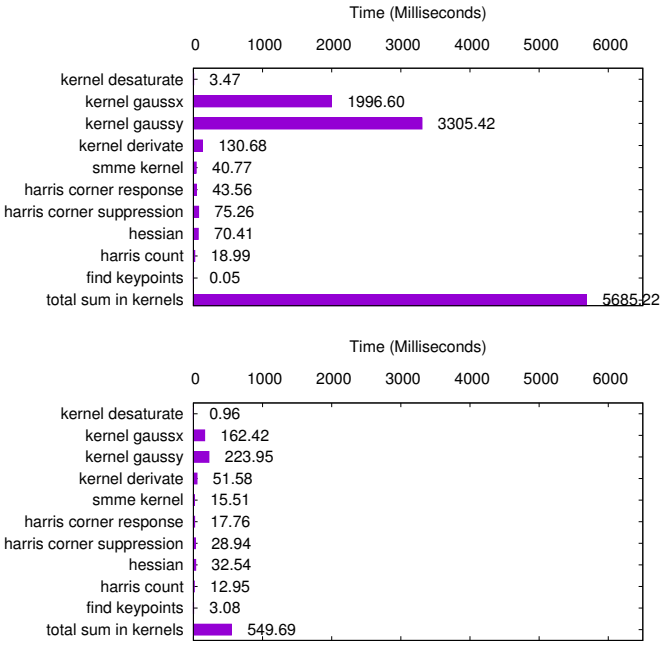

Fig. 4. Execution times (mean of 10 runs) on Xperia Z3 (upper) and Xperia XZ (lower) for the individual kernels.

B. Effect of Various Workgroup Sizes

As we saw in the previous section, the two kernels `gaussx` and `gaussy` contribute most to the execution time. Therefore, we have focused on them when we evaluated the effects of various workgroup sizes. We varied the work-group sizes between 2×2 up to 32×8 on the Xperia Z3 and between 2×2 up to 128×8 and 256×4 on the Xperia XZ when executing the Gaussian blur kernels.

TABLE II
Best and worst execution times (in ms) for different work-group sizes for the GaussX and GaussY kernels, along with the work-group sizes.

GaussX	Best	Worst
Xperia Z3	639 ms (32x8)	4560 ms (2x2)
Xperia XZ	162 ms (128x8)	12833 ms (2x2)
GaussY	**Best**	**Worst**
Xperia Z3	676 ms (8x32)	6234 ms (2x2)
Xperia XZ	224 ms (2x256)	15846 ms (2x2)

The execution times vary significantly, as shown in Figure 5 and Figure 6. In Table II we summarize the best case and worst case on each of the phones. On the Z3 we observed variations of up to a factor of ten, but on the XZ the variation was even larger. The worst kernel execution time was almost 80 times slower than the best on the XZ. Therefore, we conclude that a proper selection of the work-group size has a significant impact on the GPU execution time on the XZ (Adreno 530 GPU).

C. Temperature Effects

The second aspect that we evaluated is the operational temperature of the phones when running a performance demanding computer vision algorithm. Our results in Fig. 7 indicate that neither the Xperia Z3 nor the Xperia XZ have any temperature issues when running the Harris-Hessian/FREAK application.

When the program starts, the phones have been idle for a significant period of time, and we see that they have a temperature of approximately $38\,^{\circ}\mathrm{C}$ (Z3) and $35\,^{\circ}\mathrm{C}$ (XZ). After running the program for roughly 30 minutes, we see that the Z3 has reached a stable temperature zone around $50\,^{\circ}\mathrm{C}$ for the GPU sensors. On the XZ, we have not been able to map the different temperature sensors (tz0-tz20) to specific parts of the chip set, but we can conclude that the XZ has a stable working temperature between $38\,^{\circ}\mathrm{C}$ to $42\,^{\circ}\mathrm{C}$.

On the XZ, we can also observe that after approximately 1800 seconds, the temperature drops approximately $1\,^{\circ}\mathrm{C}$ on the XZ. This can be correlated to the frequency measurements in Fig 9. After approximately 1800 s we observe that the working frequency for CPU3 and CPU4 drop $\approx$ 100 MHz (from $\approx$ 700 MHz to $\approx$ 600 MHz). A general observation from the frequency measurements in Fig. 8 and Fig. 9 is that only 2 CPU cores appear active in both the Z3 and the XZ, while the GPU runs at max frequency for the majority of the program execution. The GPU frequency drops mainly

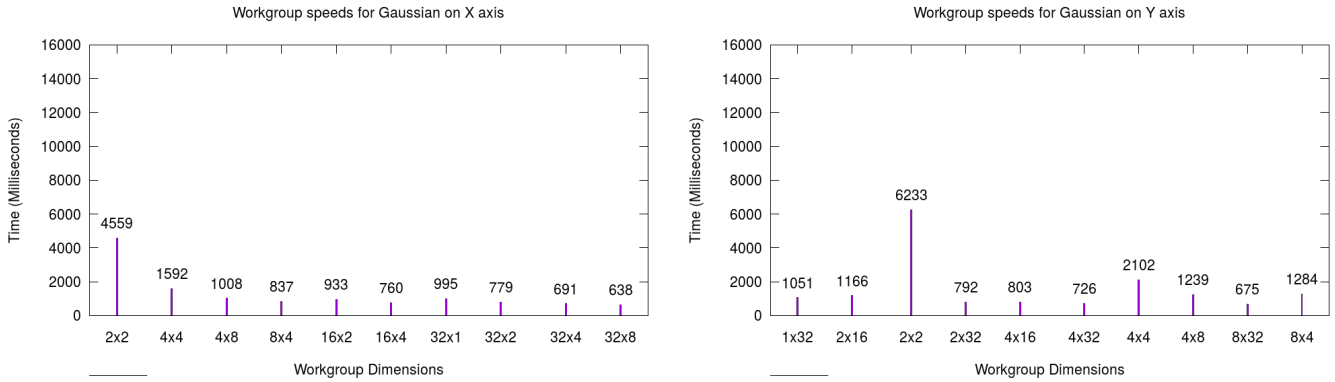

Fig. 5. Execution times on Xperia Z3 for various workgroup sizes for the `GaussX` (left) and `GaussY` (right) kernels.

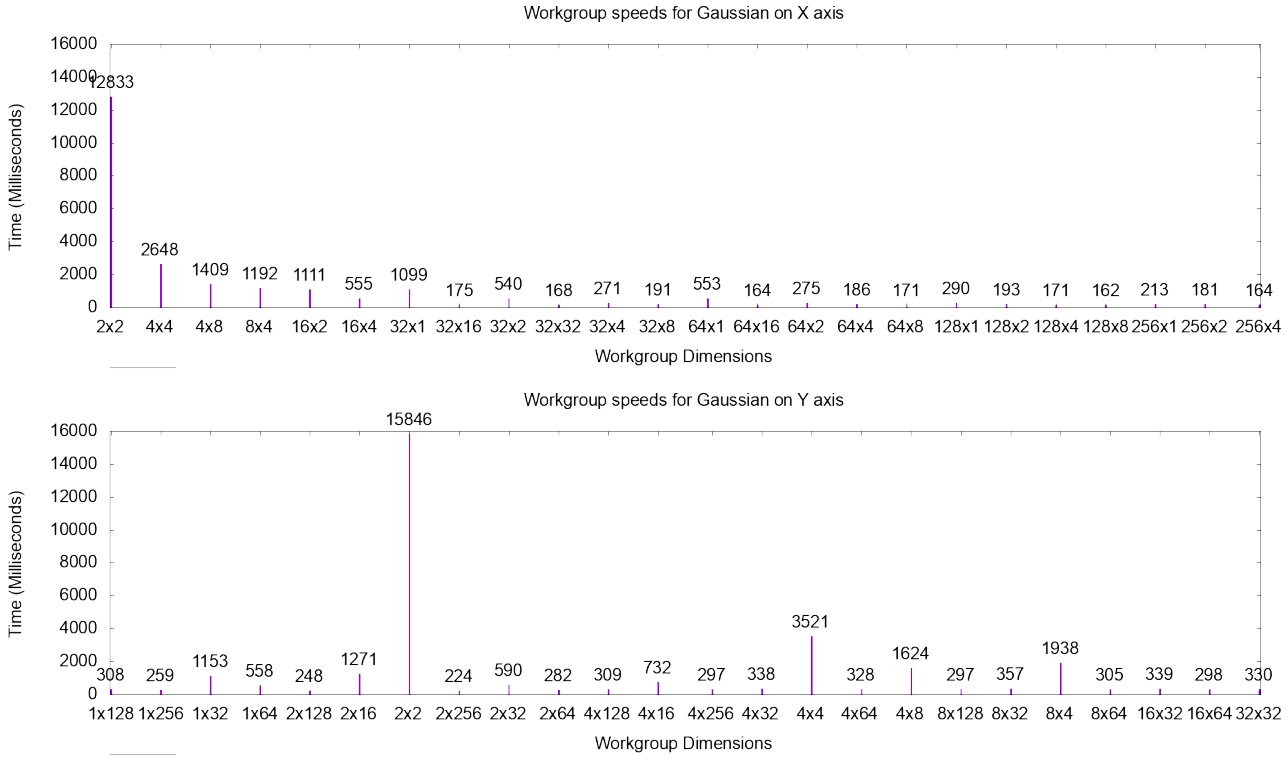

Fig. 6. Execution times on Xperia XZ for various workgroup sizes for the `GaussX` (upper) and `GaussY` (lower) kernels.

when the program is running the FREAK algorithm, which is exclusively on the CPU.

VII. Conclusion

In this paper we have studied two generations of embedded GPUs when running a performance demanding computer vision algorithm. Our study indicates that the performance has increased a factor of ten over two generations, mainly due to more GPU cores and support for larger work-group sizes. Further, the newer GPU was much more performance sensitive to the work-group size.

We have observed that the GPUs can run at their maximum clock frequencies for long periods of time, without any thermal problems or need to reduce the clock frequency. In contrast, the CPU frequencies were decreased to reduce the working temperature.

Acknowledgments

This work was partly funded by the Industrial Excellence Center ”EASE - Embedded Applications Software Engineering”, (http://ease.cs.lth.se), and the ”Scalable resource-efficient systems for big data analytics” project funded by the Knowledge Foundation (grant: 20140032) in Sweden.

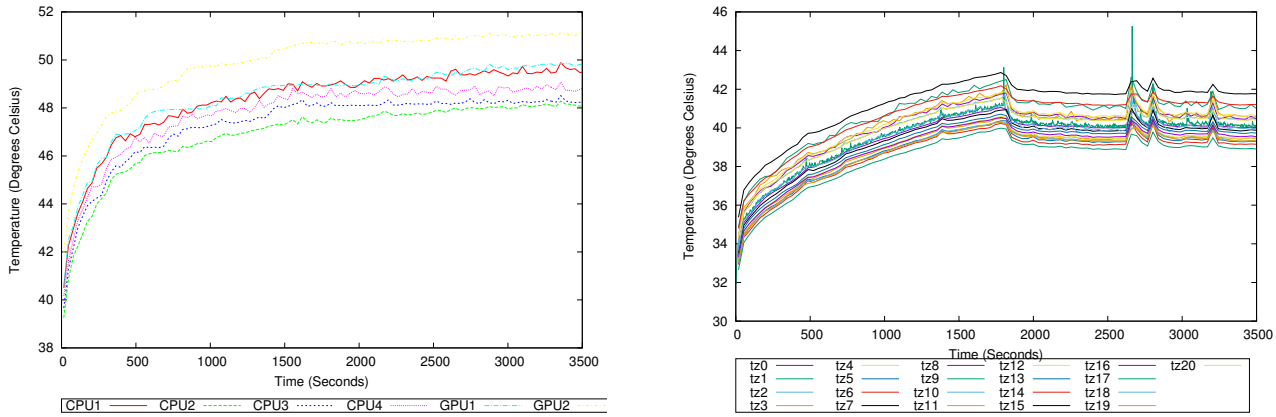

Fig. 7. Temperature during 3500 seconds, from start of idle phone, on Sony Xperia Z3 (left) and Sony Xperia XZ (right).

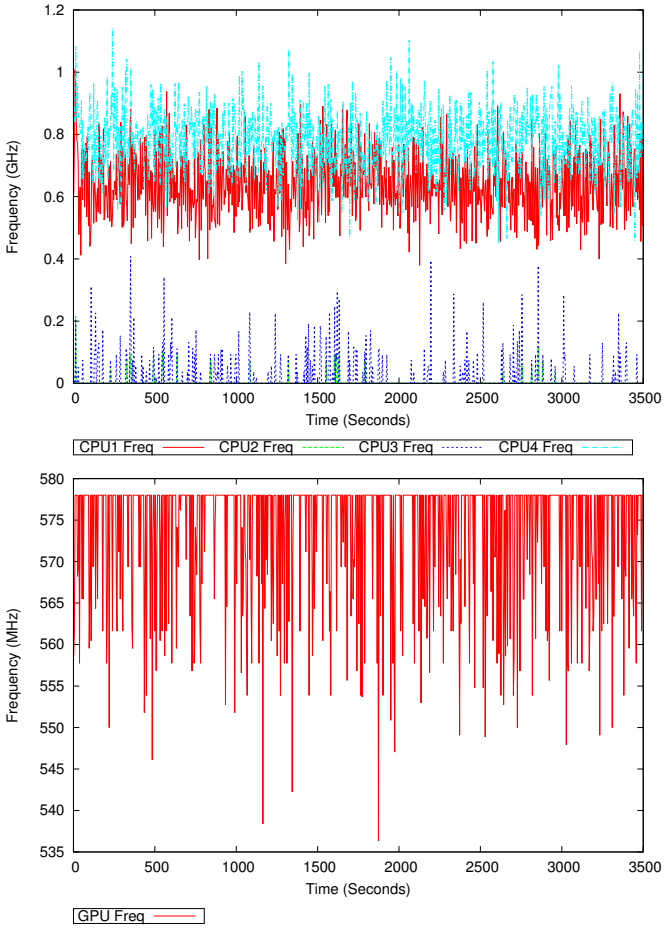

Fig. 8. Clock frequency of the CPUs and the GPU of the Xperia Z3 during the heat stress test. In the graphs we see that CPU 1 and CPU 4 both are active while CPU 2 and 3 appear inactive. The GPU is mainly running at maximum speed with occasional short dips.

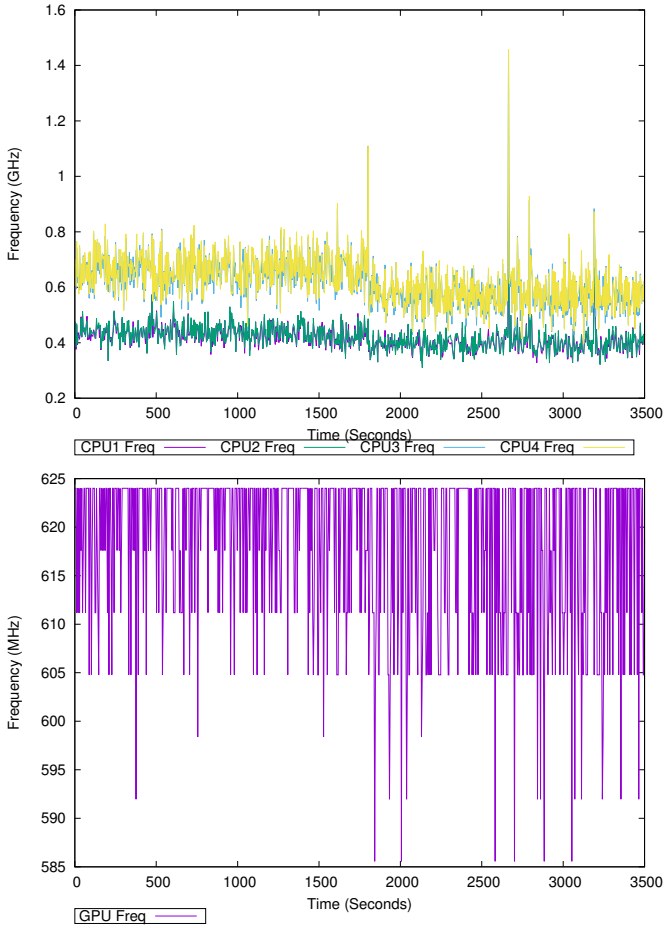

Fig. 9. Clock frequency of the CPUs and the GPU of the Xperia XZ during the heat stress test. In the graphs we see that CPU3 and CPU4 both are active while CPU1 and CPU2 have less work to do. The GPU is mainly running at maximum speed with occasional short dips.

REFERENCES

[1] A. Alahi, R. Ortiz, and P. Vandergheynst. FREAK: Fast Retina Keypoint. In *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 510–517, June 2012.

[2] K. Aniruddha Acharya and R. Venkatesh Babu. Speeding up SIFT using GPU. In *4th Nat'l Conf. on Computer Vision, Pattern Recognition, Image Processing and Graphics*, pages 1–4, Dec 2013.

[3] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding (CVIU)*, 110(3):346–359, 2008.

[4] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded Up

Robust Features. In *Computer Vision – ECCV 2006*, number 3951 in LNCS, pages 404–417. 2006.

[5] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. BRIEF: Binary Robust Independent Elementary Features. In *Computer Vision – ECCV 2010*, number 6314 in LNCS, pages 778–792. 2010.

[6] Max Danielsson, Thomas Sievert, Håkan Grahn, and Jim Rasmusson. Feature detection and description using a Harris-Hessian/FREAK combination on an embedded GPU. In *5th Int'l Conf. on Pattern Recognition Applications and Methods*, pages 517–525, Feb. 2016.

[7] R. W. Hamming. Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 29(2):147–160, April 1950.

[8] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[9] Guy-Richard Kayombya. SIFT Feature Extraction on a Smartphone GPU Using OpenGLES2.0. Master's thesis, Massachusetts Institute of Technology, 2010.

[10] S. Leutenegger, M. Chli, and R.Y. Siegwart. BRISK: Binary Robust Invariant Scalable Keypoints. In *2011 IEEE Int'l Conf. on Computer Vision (ICCV)*, pages 2548–2555, November 2011.

[11] D.G. Lowe. Object recognition from local scale-invariant features. In *Proc. of the Seventh IEEE Int'l Conf. on Computer Vision*, volume 2, pages 1150–1157, 1999.

[12] Krystian Mikolajczyk and Cordelia Schmid. Scale & Affine Invariant Interest Point Detectors. *Int'l J. of Computer Vision*, 60(1):63–86, October 2004.

[13] Aaftab Munshi. The OpenCL Specification Version: 1.1 Document Revision: 44, 2011.

[14] B. Rister, Guohui Wang, M. Wu, and J.R. Cavallaro. A fast and efficient SIFT detector using the mobile GPU. In *2013 IEEE Int'l Conf. on Acoustics, Speech and Signal Processing*, pages 2674–2678, May 2013.

[15] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: An efficient alternative to SIFT or SURF. In *2011 IEEE Int'l Conf. on Computer Vision (ICCV)*, pages 2564–2571, November 2011.

[16] Sony Mobile Communications AB. Sony Xperia XZ.

[17] Sony Mobile Communications AB. Sony Xperia Z3.

[18] Hongtao Xie, Ke Gao, Yongdong Zhang, Jintao Li, and Yizhi Liu. GPU-based fast scale invariant interest point detector. In *2010 IEEE Int'l Conf. on Acoustics Speech and Signal Processing*, pages 2494–2497, March 2010.

[19] Hongtao Xie, Ke Gao, Yongdong Zhang, Sheng Tang, Jintao Li, and Yizhi Liu. Efficient Feature Detection and Effective Post-Verification for Large Scale Near-Duplicate Image Search. *IEEE Transactions on Multimedia*, 13(6):1319–1332, December 2011.

[20] Zhou Yonglong, Mei Kuizhi, Ji Xiang, and Dong Peixiang. Parallelization and Optimization of SIFT on GPU Using CUDA. In *2013 IEEE Int'l Conf. on Embedded High Performance Computing and Communications and 2013 IEEE 10th Int'l Conf. on Ubiquitous Computing (HPCC_EUC)*, pages 1351–1358, Nov. 2013.