GPU-enhanced Multimodal Dense Matching

Nicolai Behmann*, Max Mehltretter†, Sebastian P. Kleinschmidt‡,
Bernardo Wagner‡, Christian Heipke† and Holger Blume*
*Institute of Microelectronic Systems, Leibniz Universität Hannover
{behmann, blume}@ims.uni-hannover.de
†Institute of Photogrammetry and GeoInformation, Leibniz Universität Hannover
{mehltretter, heipke}@ipi.uni-hannover.de
‡Institute of Systems Engineering - Real Time Systems Group, Leibniz Universität Hannover
{kleinschmidt, wagner}@rts.uni-hannover.de

***Abstract*—Multiple modalities for stereo matching are beneficial for robust path estimation and actioning of autonomous robots in harsh environments, e.g. in the presence of smoke and dust. In order to combine the information resulting from the different modalities, a dense stereo matching approach based on semi-global matching and a combined cost function using cross-based support regions and phase congruency shows a good performance. However, these computationally complex algorithmic steps set high requirements for the mobile processing platform and prohibit a real-time execution at limited power budget on mobile platforms. Therefore, this paper explores the usage of graphic processors for the parallelization and acceleration of the aforementioned algorithm. The resulting implementation performs the computation of phase congruency and cross-based support regions at 68 and 5 frames per second for [960 x 560] pixel images on a Nvidia Quadro P5000 and Tegra X2 GPU respectively.**

***Index Terms*—multimodal, stereo, matching, GPU, cross-based support regions, phase congruency**

I. Introduction

The combination of RGB and thermal imaging has already proven to be useful in the field of robotics, especially in search and rescue scenarios. In presence of dust, smoke or difficult lighting situations, thermal imaging has shown to be a robust supplement to traditional RGB imaging, since it still provides usable images even then. However, as shown in [1], RGB imaging provides more image features in ordinary situations compared to the number of detectable features in the more uniformly looking thermal images. Furthermore, the features extracted from thermal images suffer from high ambiguities. Therefore, the use of both imaging modalities combines the advantages of feature count, matching quality as well as robustness.

Using this sensor combination, [2] has shown that visual odometry based on multimodal imaging can drastically improve the robustness against partial, unimodal sensor failure as it can occur in situations including smoke or challenging illumination. Furthermore, [3] and [4] utilize this combination for dense stereo matching. Since in [2] the scale of the resulting trajectory is estimated based on different assumptions, its accuracy depends on the validity of these hypotheses. A combination with the approach of [3] or [4] could therefore increase the robustness of this step. Unfortunately, due to the algorithmic complexity of the solution, the multimodal dense stereo matching approach is not suited for use in mobile robotic applications, when only a CPU is available.

Consequently, in this paper we present a GPU-enhanced implementation, to make the idea of [4] usable for a wider range of applications. The computationally most complex algorithmic parts of cross-based support region construction and phase congruency map computation represent $\approx 70\%$ of the total runtime. Using different GPU-oriented optimization techniques regarding parallelization and optimized memory accesses, the runtime is drastically improved.

The remaining article is structured as follows: Section II reviews existing implementations of algorithmic components on graphic processors. In Section III the multimodal dense stereo matching algorithm is presented, which is optimized and migrated to the GPU in Section IV. The performance is then evaluated in Section V. Finally, Section VI concludes this paper.

II. Related Work

The previously named computationally expensive parts of the proposed multimodal dense stereo matching algorithm have been partly accelerated in the following related publications on GPUs. A performance of corresponding $3.33\,ms$ for stereo images with a resolution of $[450 \times 375]$ pixels for cross-based support region computation has been achieved on a Nvidia Geforce8800 GTX GPU using shared memory [5]. In [6] the support region computation and aggregation consumes $\approx\ 70\,\%$ of the total runtime; it could be improved from $15\,s$ (Core2Duo 2.20GHz CPU) to $0.095\,s$ (NVIDIA GeForce GTX 480). In [7] a significant speed-up using a NVIDIA GeForce 7900 GTX GPU was reached in comparison to the 3.2 GHz CPU. For a stereo scene with resolution $[384 \times 288]$ pixels and 16 disparity levels, the total speed is about $17\,fps$ (numbers for support-region construction were not provided). Phase congruency is used in different medical applications and is accelerated on a GPU in [8]. A frame-rate of $15\,fps$ was reached for frames with a resolution of $[720 \times 576]$ pixels. However, these implementations partly benefit from approximate support regions, different support construction

978-1-5386-5541-2/18/$31.00 ©2018 IEEE

paradigms or missing moment calculations. Therefore, they are unsuitable for the mentioned multimodal dense stereo matching approach.

An optimized CPU implementation of semi-global matching is presented in [9]. Various GPU implementations have been presented on a GeForce 8800 Ultra [10] ($0.0057\,fps$ at a resolution of $[640\times480]$ pixels and 128 disparity levels), a Quadro FX5600 [11] and a GTX 280 [12], [13]. On a Tesla C2050, a high performance implementation with $63\,fps$ at a resolution of $[640\times480]$ pixels and 128 disparity levels is demonstrated in [14]. This CUDA-based implementation is used within the proposed multimodal stereo matching algorithm for the SGM acceleration.

III. Multimodal Dense Stereo Matching

To densely reconstruct a 3D scene from images that depict it in different manners, is a challenging task in the field of photogrammetry; especially, if the images are as different as those taken with a thermal and a RGB camera. To nevertheless be able to accurately estimate the depth for most of the pixels, a robust image matching approach is crucial. For this purpose, an approach based on a combined cost function operating on phase congruency maps [15] and optimized by semi-global matching [16] was proposed in [4] and is schematically shown in Figure 1. In the following part of this section, a brief review on the main elements of this approach is given. For more details, the interested reader is referred to the original publications.

A. Combined Cost Function

The core element of the underlying approach is a combined cost function which consists of four elements: A modified Census transformation (MC), Zero-mean Normalized Cross-Correlation (ZNCC), Normalized Sum of Squared Differences (NSSD) and a triangle-based depth prediction approach. The combination itself is realized as a weighted sum of the aforementioned metrics, using a confidence-based weighting scheme. For this purpose, the confidence in each metric is determined for every pixel separately based on the corresponding cost function. Consequently, the metric with the highest confidence score dominates the cost function for the current pixel. In this way, the different strengths of the individual approaches can be exploited and the overall performance improved. Additionally, all metrics are applied to the images transformed by phase congruency [15], in order to increase the robustness against the differences introduced by the various modalities.

B. Phase Congruency

Phase congruency is a concept used to transform images into a representation which is invariant to differences in illumination and contrast and was originally proposed in [15]. The basic idea is to analyze the phase information of an image in a local context within the frequency domain. Since it is not possible to accurately determine spatial position and frequency simultaneously due to the Time-Frequency Uncertainty Principle, a bank of Log-Gabor filters is utilized to solve this task approximately. The response of a single filter with scale n and orientation θ consists of an even and an odd symmetric component and results from the convolution of the corresponding filter part with the input image at position x:

$$[e_{n\theta}, o_{n\theta}] = [I(x) * M^e_{n\theta}, I(x) * M^o_{n\theta}]\,. \tag{1}$$

The phase congruency itself is then approximated by the sum of all employed filters, divided by the amplitude of the response $A_{n\theta}(x)$. Additionally, ϵ is introduced to prevent the denominator from becoming zero.

$$PC(x) = \frac{\sum_\theta^\Theta \sqrt{(\sum_n^N (e_{n\theta}))^2 + (\sum_n^N (o_{n\theta}))^2}}{\sum_\theta^\Theta \sum_n^N (A_{n\theta}(x)) + \epsilon}\,, \tag{2}$$

$$A_{n\theta}(x) = \sqrt{(I(x) * M^e_{n\theta})^2 + (I(x) * M^\theta_{n\theta})^2}\,. \tag{3}$$

In addition to the result of the phase congruency itself, the corresponding moments of the transformation can be employed as well:

$$M = \frac{1}{2}\left(c + a + \sqrt{b^2 + (a-c)^2}\right)\,, \tag{4}$$

$$m = \frac{1}{2}\left(c + a - \sqrt{b^2 + (a-c)^2}\right)\,. \tag{5}$$

Following the approach proposed in [17], they can be used to detect edges and corners robustly. As can be seen in Figure 1, this additional information serves as input for the subsequent steps of triangle-based depth prediction (corners) and semi-global matching (edges). In order to determine this information, the values of the maximum moment M and the minimum moment m are compared to the thresholds ϵ_E and ϵ_C: If $M > \epsilon_E$ in a certain point, this point is labeled as 'edge' and if in addition $m > \epsilon_C$ the point is labeled as 'corner'. The corresponding coefficients are defined as:

$$a = \sum_\theta^\Theta \left(PC(\theta)cos(\theta)\right)^2\,, \tag{6}$$

$$b = 2\sum_\theta^\Theta \left(PC(\theta)cos(\theta)\cdot PC(\theta)sin(\theta)\right)\,, \tag{7}$$

$$c = \sum_\theta^\Theta \left(PC(\theta)sin(\theta)\right)^2\,, \tag{8}$$

where $PC(\theta)$ denotes the phase congruency that only takes into account orientation θ, including all scales.

C. Cross-based Support Regions

Referring to [18], stereo matching algorithms basically consist of four elements: Matching cost computation, cost aggregation within a support region, optimization and disparity refinement. While for many local stereo matching approaches cost aggregation is a major element, for global approaches it only plays a minor role, since disparity estimates are obtained within the optimization step. Nevertheless, many metrics for

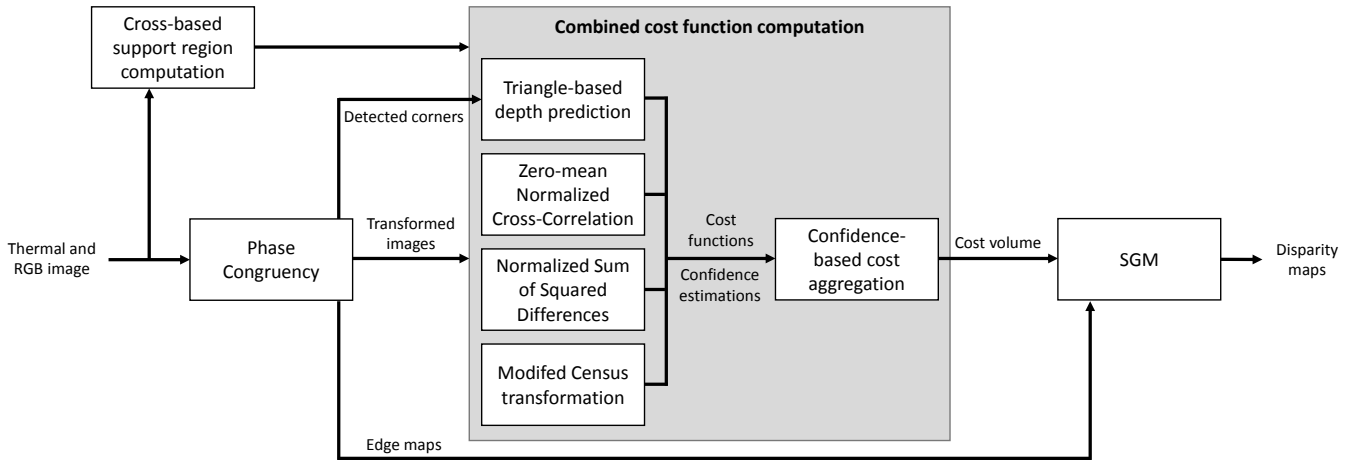
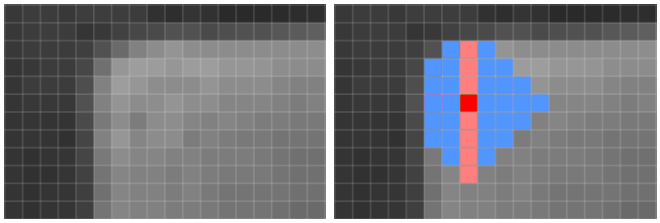


Fig. 1: Overview of the multimodal dense image matching approach proposed in [4].



(a) (b)

Fig. 2: Concept of cross-based support regions: Image (a) shows a pixel raster of an object boundary represented by gray values. In image (b), a possible cross-based support region with a maximum spatial distance of 4 px is visualized. The pixel of interest is shown in red, the vertical arms in pink and the horizontal arms in blue.

matching cost computation take information of the local neighborhood into account. Consequently, a support region around the pixel of interest is still required. The simplest and most common approach is to use a rectangular region with fixed size centered on the pixel of interest. While this approach allows easy and fast computation, it introduces some error: Near depth discontinuities, also pixels with different disparities are taken into account. These disparities have a direct influence on the matching result, but should be considered as outliers. In order to minimize the number of such outliers, the shape and size of a support region should be adapted near depth discontinuities.

Proposed by [19], cross-based support regions are one way to achieve this goal. For this purpose, a two-stage process is performed: First, two vertical arms (shown in pink in Figure 2b) are constructed, one upwards regarding to the pixel of interest and one downwards. The length of these arms is limited by thresholds for the spatial distance and the difference of gray values between the pixel of interest and the currently observed one. Starting from the pixel of interest, the upward arm is enlarged until one of these values exceeds the corresponding threshold. The same is done for the downward arm. Subsequently, for every pixel on the vertical arms, a left and a right horizontal arm is constructed (shown in blue in Figure 2b). This is done analogously to the construction of the vertical arms. In contrast to the original approach, the distance to the center pixel is not measured with the L1 but with the L2 norm. This leads to support regions with a more circular shape, as can be seen in Figure 2b. The critical part of this approach is to find suitable thresholds. Especially, a good choice for the maximum gray value difference can be challenging, since the arms should ignore noise and small deviations but not cross object boundaries.

IV. Optimization

As proven in the original publication [4], the presented method provides state-of-the-art results in terms of accuracy. However, some of the steps are computationally intensive and considerably slow down the process. In this context, three components are particularly noticeable: The computation of the cross-based support regions (39 %), the image transformation via phase congruency (32 %) and the optimization step carried out by semi-global matching (5 %). The percentages indicate the proportion of each component in the total runtime of the process shown in Figure 1 based on the original implementation. While there are already several optimization approaches for semi-global matching listed in Section II, this does not apply to the first two components mentioned. Consequently, this work concentrates on the optimization of these components.

In general, acceleration on a GPU is achieved by exploiting the following principles:

- **Work distribution** by dividing the computational load across parallel threads. Equally sized work loads and a minimum of conditional execution within the kernel reduces synchronization inefficiencies.
- **Maximized memory bandwidth** by strategically choosing the best memory hierarchy from registers, thread-

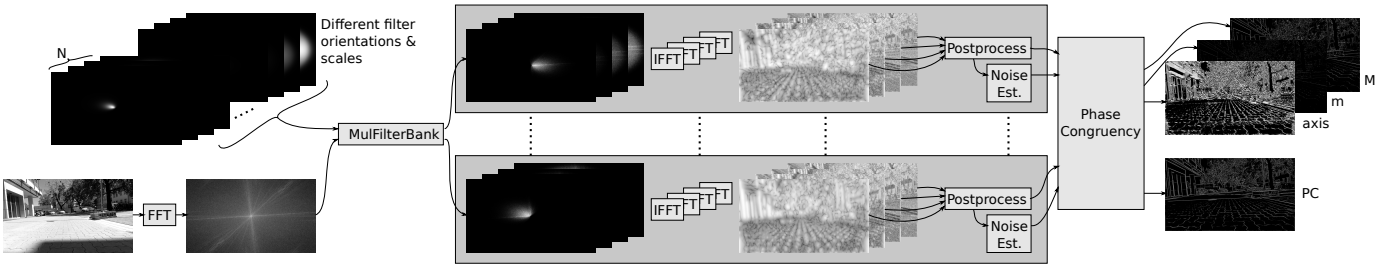


Fig. 3: Algorithmic pipeline for phase congruency computation. Following the FFT of the source image, the corresponding spectrum is multiplied with all filters of the Log-Gabor filter bank at once in the frequency domain. Afterwards, for each orientation θ , the responses are compressed in the Postprocessing kernel after the inverse FFT. Following the noise estimation from the lowest scale $n = 0$ of each orientation θ, the phase congruency map PC and minimum m and maximum M moments are calculated.

block shared memory, texture memory and global memory and thereby minimizing arithmetic pipeline stalls.

- **Fused computation** through the combination of multiple computing steps in one kernel and thereby reducing the amount of memory accesses. Additionally, SIMD techniques can be used in order to minimize the amount of memory accesses and the execution of vector operations in a single instruction.

In the following subsections, a detailed overview of the optimization techniques for the particular functions is given. The implementation for the Nvidia GPUs is done in the *Compute Unified Device Architecture* (CUDA) programming language and thereby scalable for different power budgets.

A. Phase Congruency

The computation of the phase congruency map and the corresponding moments (see Section III) is approximated with a Log-Gabor filter bank. The processing pipeline employed for this is explained in the following and also shown in Figure 3. Each filter of the bank is specified by its azimuth orientation Θ and scale N and precomputed prior to execution of the phase congruency computation. The filter responses are stored in the GPU global device memory. Subsequent to a Fast Fourier Transformation (FFT) of the source image, the filters are multiplied with the spectrum of the source image within the frequency domain. For the FFT and inverse FFT computation, the highly optimized *cufft* library from Nvidia is used. By multiplying all filters at once, the source pixels only need to be loaded once, thereby reducing the number of memory accesses. Additionally, each thread block computes a 2-dimensional image tile of the source spectrum with all filters. Using intrinsic SIMD instructions of the GPU, the instruction throughput within this step is maximized. On this occasion, two neighboring pixels with complex 32-bit floating point numbers are processed in parallel. Furthermore, loading $128\,bit$ values at once reduces the number of memory accesses as well.

Following the computation of the Log-Gabor filter bank, each kernel processes only one orientation θ with N scales. Thereby, the reduction across scales can be optimized and no temporal storage is needed. The *Postprocess* kernel first shifts the DC component of the inverse FFT and scales its values with the FFT size. With the responses still in local registers, the amplitude $A_{n,\theta}$ and the local energy is computed for every pixel. Again, SIMD operations are utilized for efficient instruction throughput and data loading. To compute the local energy, the mean value for each pixel across all scales N is needed and therefore calculated in a first iteration. In order to avoid the necessity to load the source values from global memory twice, they are stored in the local memory during the FFT postprocessing step. From there they can be accessed for later reuse. Lastly, the median value of the amplitude with scale $n = 0$ within the space domain is needed to estimate the noise for each orientation θ. For the purpose of efficient computation, an optimized Radix sort algorithm [20] is used.

For the final computation of the coefficients a, b, c of the minimum and maximum moments m and M, the transformations corresponding to the orientations θ are compressed in the last kernel. Local SIMD accumulators are used to minimize the external memory bandwidth.

B. Cross-based support regions

For the computation of the cross-based support regions, a vertical arm for each pixel is constructed first, which is then extended with horizontal pixels. The maximum size of a support region depends both on a maximum spatial distance K as the $L2$-norm to the current pixel and a maximum difference in the gray values to the corresponding base point of the current arm.

As no dependencies exist between neighboring support regions, each source pixel can be processed in parallel. Therefore, each thread is assigned to one pixel of the source image and calculates the cross-based support region within one kernel. Different sizes of the support region within the same thread block can lead to different execution times and thereby synchronization overhead. However, as the pixels are relatively close to each other, neighboring support regions have a similar shape and the runtime difference is negligible in comparison to the overhead for additional load and stores in case of separate construction of vertical and horizontal arms.

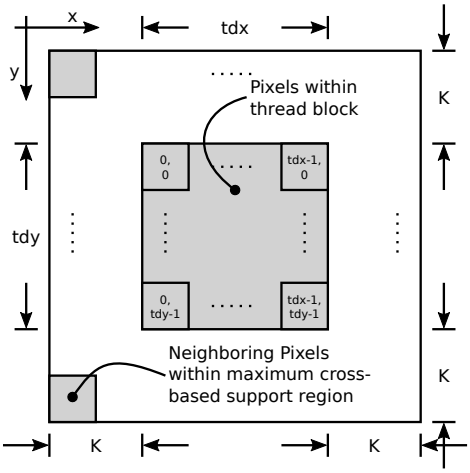


Fig. 4: K neighboring pixels at the borders of the current thread block of the image are loaded to shared memory and used for computation of cross-based support region. The block uses $tdx \times tdy$ threads.

An initial benchmark of the cross-based support region kernel using global memory accesses shows that the high amount of memory accesses leads to a high stall rate of the threads, thereby minimizes computing efficiency. Using cached texture memory, instead of the global memory, the memory accesses are optimized at the expense of texture binding and unbinding. Additionally, neighboring threads, e.g. within the same thread block, share the majority of source pixels, needed for support region construction. Therefore, as another method, shared memory is introduced to speed-up accesses to the relevant source image regions. Figure 4 illustrates the usage of shared memory for the cross-based region computation.

Beside the pixel of each thread, pixels around the thread block borders and inside a radius of the maximum spatial distance K are loaded. Within the kernel, first each thread loads its current pixel and then synchronizes with the other threads in the same block. Threads on the border additionally load the pixels of their respective border column or row. The computing imbalance is maximized between central threads, loading only one pixel, and corner pixels, responsible for $(K+1)^2$ pixels before the synchronization. Subsequently, all threads construct their support region in parallel and benefit from high-bandwidth access to the source pixels in shared memory. The size of the shared memory is limited to $49152\ Byte$ per thread block, which is not critical for the maximum thread block size of 1024 threads and a typical threshold of $K = 5$ ($1764\ Byte$).

V. Evaluation

For the purpose of evaluation, two different GPUs from Nvidia for different fields of application are used. A 256-core mobile GPU variant of the Nvidia Tegra X2 processor utilizing $8\,GB\ 128\,bit$ LPDDR4 with a theoretical bandwidth of $59,7\,GB/s$ is evaluated for energy-limited scenarios with a power consumption of less than $15\,W$. For high-performance mobile platforms, the Quadro P5000 GPU with 2560 CUDA-cores, $16\,GB$ GDDR5X at a $256\,bit$ memory interface and $288\ GB/s$ theoretical bandwidth at $180\,W$ power consumption is evaluated. Both platforms share the underlying Pascal architecture. The evaluation is performed on an image with size $[960 \times 540]$ pixels and the runtime is averaged over 100 runs.

TABLE I: Execution time and percentage share of total duration for the different kernels of the phase congruency computation.

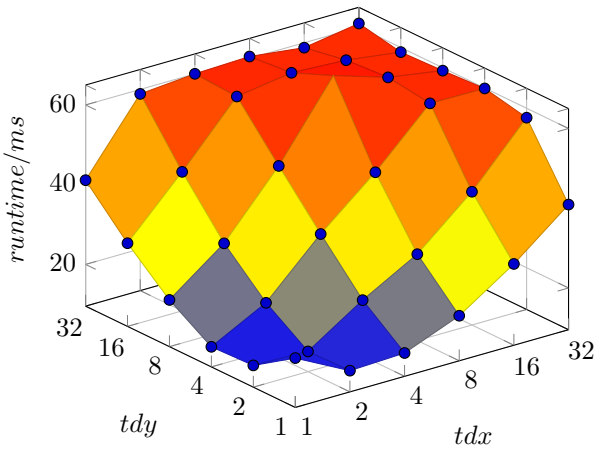
Kernel	Runtime [ms] Quadro	Percentage Quadro	Runtime [ms] Tegra	Percentage Tegra
FFT and IFFT	2.77	42 %	40.9	53 %
Multiply Filter Bank	0.67	10 %	6.4	8 %
Postprocess	0.69	11 %	7.0	9 %
Noise Estimation	2.20	33 %	17.9	23 %
Phase Congruency	0.27	4 %	5.5	7 %
Total	6.60	100 %	77.7	100 %

A. Phase Congruency

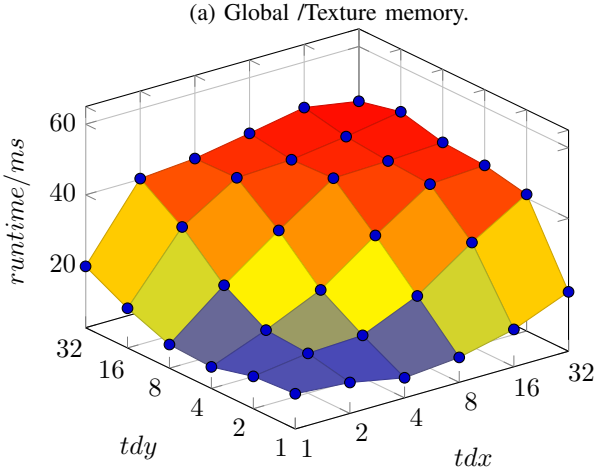
The portions of the runtime of the phase congrueny computation are presented in Table I for the Nvidia Quadro P5000 and Tegra X2 GPU. On both platforms the Fast Fourier Transformation consumes the majority of the GPU computing resources of $\approx 40\,\%$, followed by the noise estimation algorithm with $33\,\%$ and $23\,\%$ on the high-performance Quadro and energy-efficient Tegra GPU, respectively. The latter processing step can be approximated with a constant noise estimation for a whole sequence or the same threshold for all orientations θ with only little differences in the phase congruency map. For the computation of the phase congruency, a thread group size of $tdx = tdy = 16$ is found as the optimal configuration. Comparing the runtime of the Quadro GPU with the Tegra GPU, the former high-performance GPU is $11\times$ faster than the mobile GPU, which also offers only $10\,\%$ of the CUDA cores in comparison to the Quadro. This indicates, that the computation is compute bound and not limited by the available memory bandwidth.

B. Cross-based support region

Beside the parallelization across multiple threads, the cross-based support region computation is accelerated mainly by using optimized or local memories, see Section IV-B. In case of shared memory, an increase in the number of threads per block ($[tdx \times tdy]$) leads to higher reuse of data in the shared memory. However, in the stage of loading these values, more threads in the central part stall, while border threads copy data and therefore reduce computational efficiency. In order to find



(a) Global /Texture memory.



(b) Shared memory.

Fig. 5: Execution time of the cross-based region implementation using different memories and numbers of threads tdx, tdy in a thread block for an image of size $[960 \times 540]$ pixels.

the optimum block size, first the execution time is evaluated as illustrated in Figure 5. The usage of texture memory in comparison to global memory does not result in a noticeable speed-up, as the required areas are already cached. The results are plotted in Figure 5a. A large improvement can be seen for shared memory (Figure 5b), with a minimum runtime for small block sizes of $[4 \times 1]$ of $8\,ms$ or $144\,ms$ on the Nvidia Quadro or Tegra respectively. The big runtime difference of a factor of $18\times$ between the Nvidia Tegra and Quadro GPU, which has $10\,\%$ of the computing resources of the latter, highlights the fact, that the computation of the cross-based support regions is highly memory bandwidth limited. The memory bandwidth of the LPDDR4 DRAM is much smaller in comparison to the GDDR5X memory and therefore, the threads of the mobile GPU are not as busy as the Quadro GPU.

VI. Conclusion

The implementation of the phase congruency and cross-based support region computation on the Nvidia Quadro P5000 and Tegra X2 GPU reach a framerate of 68 and 5 frames per second respectively on images with a resolution of $[960 \times 560]$ pixels, which is sufficient for the given use-case of mobile platforms in harsh environments, as smoke. The detailed analysis of the GPUs performance bounds provided valuable insights which enabled the design of highly optimized kernels and data flow.

References

[1] S. P. Kleinschmidt and B. Wagner, "Probabilistic Fusion and Analysis of Multimodal Image Features," *18th International Conference on Advanced Robotics*, pp. 498–504, 2017.

[2] ——, "Visual Multimodal Odometry: Robust Visual Odometry in Harsh Environments," *IEEE International Symposium on Safety, Security, and Rescue Robotics*, 2018, (In press).

[3] M. Mehltretter, , and C. Heipke, "Illumination Invariant Dense Image Matching based on Sparse Features," in *38. Wiss.-Tech. Jahrestagung, DGPF Publikationen der DGPF, Band 27*, 2018, pp. 584–596.

[4] M. Mehltretter, S. P. Kleinschmidt, B. Wagner, and C. Heipke, "Multimodal Dense Stereo Matching," in *German Conference on Pattern Recognition*, 2018, (In press).

[5] K. Zhang, J. Lu, G. Lafruit, R. Lauwereins, and L. V. Gool, "Real-time accurate stereo with bitwise fast voting on cuda," in *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, Sept 2009, pp. 794–800.

[6] X. Mei, X. Sun, M. Zhou, S. Jiao, H. Wang, and X. Zhang, "On building an accurate stereo matching system on graphics hardware," in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, Nov 2011, pp. 467–474.

[7] J. Lu, K. Zhang, G. Lafruit, and F. Catthoor, "Real-time stereo matching: A cross-based local approach," in *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, April 2009, pp. 733–736.

[8] L. Struyf, S. De Beugher, D. Van Uytsel, F. Kanters, and T. Goedem, "The battle of the giants: A case study of gpu vs fpga optimisation for real-time image processing," vol. 1. VISIGRAPP; Lisbon, 2014, pp. 112–119. [Online]. Available: https://lirias.kuleuven.be/retrieve/259821

[9] S. Gehrig and C. Rabe, "Real-time semi-global matching on the CPU," *Proc. IEEE Conf. Computer Vision and Pattern Recognition Workshop*, pp. 85–92, 2010.

[10] I. Ernst and H. Hirschmüller, "Mutual information based semi-global stereo matching on the GPU," *Proc. Intl. Symp. Visual Computing*, vol. 5358, pp. 228–239, 2008.

[11] J. Gibson and O. Marques, "Stereo depth with a unified architecture GPU," *Proc. IEEE Conf. Computer Vision and Pattern Recognition Workshop*, pp. 1–6, 2008.

[12] I. Haller and S. Nedevschi, "GPU optimization of the SGM stereo algorithm," *Proc. IEEE Intl. Conf. Intelligent Computer Communication and Processing*, pp. 197–202, 2010.

[13] C. Pantilie and S. Nedevschi, "SORT-SGM: Subpixel optimized real-time semiglobal matching for intelligent vehicles," *IEEE Trans. Vehicular Technology*, vol. 61, no. 3, pp. 1032–1042, 2012.

[14] C. Banz, H. Blume, and P. Pirsch, "Real-time semi-global matching disparity estimation on the GPU," *Proc. IEEE Intl. Conf. Computer Vision Workshops*, pp. 514–521, 2011.

[15] P. Kovesi, "Image Features from Phase Congruency," *Videre: Journal of Computer Vision Research*, vol. 1, no. 3, pp. 1–26, 1999.

[16] H. Hirschmuller, "Stereo Processing by Semiglobal Matching and Mutual Information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 328–341, 2008.

[17] P. Kovesi, "Phase Congruency detects Corners and Edges," in *The Australian Pattern Recognition Society Conference: DICTA*, vol. 2003, 2003.

[18] D. Scharstein and R. Szeliski, "A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms," *International Journal of Computer Vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[19] K. Zhang, J. Lu, and G. Lafruit, "Cross-based Local Stereo Matching using Orthogonal Integral Images," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 19, no. 7, pp. 1073–1079, 2009.

[20] N. Satish, M. Harris, and M. Garland, "Designing efficient sorting algorithms for manycore gpus," in *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, ser. IPDPS '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–10. [Online]. Available: https://doi.org/10.1109/IPDPS.2009.5161005