I J C T A, 9(13) 2016, pp. 5927-5935
© International Science Press

GPU Computing for Computer Vision: A Survey

P. Subhasree[1*], P. Karthikeyan[**] and R. Senthilnathan[**]

ABSTRACT

Graphics processing unit (GPU) is a special form of processor architecture meant for parallel processes. Computer Vision being a field which handles large data and intensive algorithms naturally seems to be candidate choice for GPU implementation on a GPU. This paper reports the review of literature related to utilization GPU for general purpose, non-graphics based application. Fundamentals related to GPU architecture and software tools available for general purpose computing in GPU are studied from the perspective of a computer vision engineer. The significant benefits of GPU implementation of computer vision algorithms and various limitations and challenges involved are described in good detail. The paper reviews the various programming models and software libraries for general purpose GPU (GPGPU) computing along with computing vision specific libraries. The most significant contribution in the paper is the presentation of performance metrics and the various aspects of evaluation that may be used to assess the GPU acceleration of computer vision algorithms. Publications reporting the GPU acceleration of popular algorithms are presented highlighting the need for GPU implementation, hardware details, performance metrics and variables considered for evaluation and the intended applications are reported.

Keywords: Graphics Processing Unit, GPGPU, GPU Programming Tools, Computer Vision, Performance Metrics

I. INTRODUCTION

It is an era of parallel processing that computing with a single processor no more gives the desired, fast or efficient outcomes. It is already proven to be impossible to build up a processor with higher speedup according to Moore's law and this gave the opportunity and necessity to get into the world of parallel processing [1]. Parallel architectures with greater processing capabilities are now available and the only necessity is to develop algorithms and skeletons that would effectively utilize the processing elements of the parallel architectures [2]. Many applications with complex algorithms, numerous logical operations can be realized using the parallel architectures. Computer vision is one such field with so many applications like specified above. Like a biological vision system, the computer vision algorithms also take in large volume of data into memory, processes the data which is complex, sometimes does real time processing of the data. This makes computer vision algorithms as candidates for parallel processing.

Among many upcoming parallel architectures, GPUs are powerful which are available in the form of commodity graphic cards that are inexpensive. The GPU was developed by NVIDIA Corporation in the year 2000 [1]. The graphics card has many processing elements with large memory bandwidth and basically is a single instruction multiple data (SIMD) architecture [1]. GPU has multiple processors, in which each processor runs a different thread, but all the threads of a multiprocessor execute same instruction on multiple data [2]. Graphics cards which were originally developed for computer graphics are now used for many applications. The use of GPUs lead to a new domain in computing hardware called GPGPU or what is commonly referred to as GPU-accelerated computing. GPU-accelerated computing is the use of a GPU together with a CPU to accelerate scientific, analytics, engineering, consumer, and enterprise applications

[*] Department of Production Technology, MIT Campus, Anna University Chennai, India
[**] Department of Mechatronics Engineering, SRM University, Kattankulathur, India
[1]*subhasreesenthilnathan@gmail.com*

[1]. GPGPU gained popularity after 2001 with the advent of shaders and floating point support on GPUs [1]. Initial GPGPU applications required tedious reformations of the desired computing problems to be converted to the framework of computer graphics. Moreover hardware vendor dependency is another major challenge. In recent trends significant developments have been made for vendor independent GPGPU pipelines that can exploit the speed of GPU without involving explicit conversion of the application data to graphics primitives.

Many articles were found in the literatures which review the utilization of GPU for computer vision and related tasks. Most articles were based on a small class of algorithms, if not, on a specific algorithm. The evolution of CPU-GPU architecture for general purpose computing from the perspective of hardware sharing and collaborative execution schemes is presented [1]. Many non-computer graphics based application such as image processing, data mining etc., were considered to demonstrate the collaborative processing architectures of CPU-GPU systems. Review of computer vision and image processing algorithms are presented [2]. This paper is a similar attempt to present the survey details of utilization of GPUs for computer vision and related tasks. The paper starts with preamble content for GPU followed by the various tools available for GPGPU. The next section of the paper presents the suitability, difficulties and tools related to GPU architecture. One of the main novelties in the review is the way the detail of survey is presented. The GPU implementation results presented in the literature is carefully classified based on the algorithm type and the application where they were tested. Followed by this the various popular computer vision algorithms subjected to GPU acceleration are tabulated presenting the challenges, hardware details and metrics.

II. GPU ARCHITECTURE

A GPU is a super-specialized parallelized microprocessor which is meant to offload and accelerate 2-D or 3-D rendering from CPU. GPUs are available for a variety of hardware platform ranging from PC architecture to super computers. A detailed summary and history of GPU architecture is presented [3].

(A) CPU vs GPU Architecture

Conventionally CPUs were sped up to handle computationally intensive tasks mainly by increasing the clock speed of the CPU. This trend reached saturation due to the adverse effects of increasing clock speeds such as energy-consumption and heat dissipation issues. One of the main strategy adopted was using more than one processor in two possible alternatives were realizable. Multi-core architecture is a typical example for CPU architecture, which was designed to maximize the speed of execution of sequential programs. The CPU cores are basically out-order multiple instruction issue processor [4]. The other alternative for CPU is the massively parallel processors to which GPU architecture belongs. The philosophy of GPU is to utilize hundreds of small in-order processors which are best suited for applications that naturally possess parallelization possibility. Multi-core CPUs adopt a complex control logic to make instructions from a single thread to run in parallel with compromising the appearance of sequential execution. Fig. 1 illustrates the architectural difference between CPU and GPU.

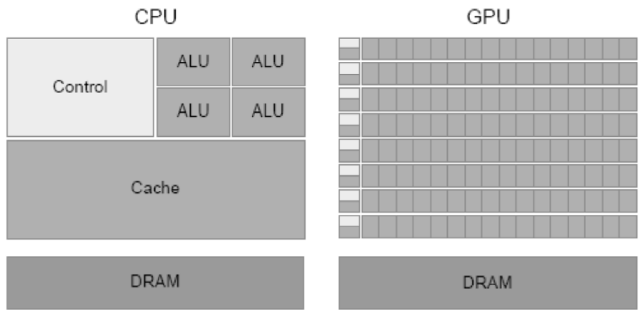

Figure 1: Comparison of CPU and GPU architectures

(B) Benefits of GPU

The main advantage of GPUs is the possibility of parallelization of certain algorithms eased by numerous processing cores. Algorithm that offer scope for parallelization can run more than 100 times faster as compared to a CPU. GPUS generally offers more floating point units than CPU. Modern GPUs are energy efficient and relatively cheap compared to common CU counterparts.

(C) Limitations of GPU

GPUs though has vast computational resources in terms of processors has its own limitations and disadvantages in certain context of application. GPU implementation of an algorithm requires the structure of the algorithm to be modified to make it suitable for the GPU architecture [1]. This is generally a tedious process as GPU programming is significantly different from a conventional CPU programming. Algorithms which offer less scope for parallelization when attempted to be accelerated will result in additional over heads contributed by frequent CPU-GPU communication [1]. Generally GPU acceleration is suitable only for algorithms with less inter-task dependencies are very less which eases parallelization. GPUs as an additional hardware in a system adds to cost, power consumption and heat dissipations and hence must be justified by the demands of the application [1].

(D) Software Tools for GPGPU

Programming GPU for general purpose applications have taken a giant leap with the advent of compute unified device architecture (CUDA) from NVIDIA Corporation and open computing language (OpenCL) from Khronos Group. Many diverse application such as image processing [5][6][7], computer vision, Monte Carlo simulation [8] etc. OpenMP is one of the popular API for shared memory multi-processor programming which supports a variety of programming languages and operating systems. Similar to OpenMP, development of high-level programming models for GPU based heterogeneous systems have been made. Most recently, models such as hybrid multicore parallel programming (HMPP) [9], open accelerators (OpenACC) [10], PGI accelerator [11], hiCUDA [12], OpenMPC [13], have been developed and reported. The programming complexity and time performance obtained through HMPP and OpenACC directives have been evaluated and compared with multicore CPU, OpenMP and CUDA version in [14]. CUDA, HMPP, OpenCL, and OpenACC have been used individually to accelerate financial applications in [15]. Evaluation of OpenACC, OpenCL, and CUDA with respect of performance, developer’s productivity, and portability was reported in [16]. Detailed evaluation of existing directive based models was presented in [17]. A comparative evaluation of OpenMP, OpenACC and CUDA implementations of parallel computation of aerial target reflection of background infrared radiation was presented in [18]. This section is completely based on [18].

III. GPU FOR COMPUTER VISION ALGORITHMS

(A) Suitability of GPU for Computer Vision

Computer vision systems operate on digital images which are large two dimensional data which would greatly benefit from parallel processing. Applications such as visual tracking, optical flow, and vision guided robots require very high processing rates which will be aided by fast computation. One of the basic means of achieving fast computation is parallel execution. One of the main reasons for the suitability of the computer vision algorithms is that most algorithms operate the same mathematical function across all the pixels. This creates a scope for SIMD type of execution model. The computer vision algorithms are usually complex in their mathematical background. Most of the mathematical functions in the computer vision algorithms require several floating point and logical operations which often exceed the real-time capabilities of the CPU architecture, resulting in very less time for higher-level tasks.

(B) Hurdles for GPU Implementation of Computer Vision

One of the main hurdles in GPU implementation of computer vision algorithms is the diversified nature of the algorithms which poses a problem in terms of generalization so as to have a unified framework for GPU implementation. Computer vision algorithms require large memory buffers which often require to be frequently accessed during the course of execution of the algorithms. The nature of access is random and this makes parallelization ineffective since the latency involved in the memory access subdues the benefits of improved time performance achieved due to parallelization. Most computer vision algorithms involve interdependency on data generated from various stages of the algorithm thus direct parallelization of the entire set of procedures is not straight forward. Most of the implementation in fact adopts parallelization at the subset level. Many computer vision algorithms involve divergent branching which makes parallelization difficult.

(C) Open Source Software Tools for Computer Vision Specific GPU Acceleration

One of the most powerful open source computer vision libraryis the OpenCV library where in GPU acceleration GpuCV [19] is an opensource GPU-Accelerated framework for computer vision which has provisions for hardware management, data synchronization, OpenCV compatibility and CUDA programs. MinGPU [20] is a similar library where tools required for GPU acceleration of computer vision algorithms are provided. OpenVIDIA [21] is another popular API which is specifically developed for GPU acceleration of computer vision and image processing algorithms. NVIDIA offers a library called NVIDIA Performance Primitives (NPP) which contains GPU-accelerated image, video, and signal processing functions. The website GPU4VISION offers many resources like codes of popular implementations and their corresponding documentation.

(D) GPU Acceleration in Commercial Software Products

HALCON from Mvtec, one of the most popular machine vision library provides an efficient automatic acceleration by optimal usage of GPU onboard the computer. The GPU acceleration in HALCON is based on OpenCL standard. Image Processing Toolbox and Computer Vision System Toolbox from Mathworks which run under the MATLAB platform offers automatic GPU acceleration for most functions. For improved acceleration the Parallel Computing Toolbox support for parallelizing algorithms on NVIDIA GPUs without using CUDA, directly from MATLAB. Other software packages like Vision Development Module from National Instruments and Matrox Imaging Library from Matrox Imaging offer provision in their library which enables exploitation of the power of GPUs for computer vision tasks.

IV. GPU IMPLEMENTATION OF POPULAR COMPUTER VISION ALGORITHMS AND PERFORMANCE CHARACTERIZATION

GPU acceleration has been reported in the context of image processing as well as computer vision. Many computer vision algorithms subjected to GPU acceleration were presented in the open literature. Some of the popular algorithms include feature extraction descriptors like Scale Invariant Fast Transform (SIFT) and Speeded-Up Robust Features (SURF), corner detectors such as Harris corner detector etc. Many scene reconstruction techniques such as stereo vision, optical flow are also subjected to GPU acceleration and their speedup is reported in literature. This paper lists 9 different articles in Table I where the algorithm subjected to GPU implementation is identified for its category. Since GPU implementation may not be possible for the entire set of procedures of an algorithm, the GPU mapped portions as presented in the original article is listed. Any algorithm subjected to GPU acceleration must have a suitable application where it may be used. Hence the suggested applications in the original article are presented in the table. One of the important aspect of GPU acceleration is the details of hardware and the programming APIs used in the implementation. The hardware details must not only be about GPU but also about CPU since in any

Table I
Summary of Few Popular Computer Vision Algorithms Subjected to GPU Implementation

Reference	*Computer Vision Algorithm*	*Algorithm Type*	*GPU Mapped Portions*	*Need for GPU Impleme-ntation*	*Application Demonstrated*	*GPU Hardware Details*	*Programming Model and Libraries Used*	*Parameters Considered for Comparison*	*Performance Metricused for Evaluation*	*Speed Up In terms of No. of Folds (×)*
[24]	PMVS	Optical Flow Feedback for Scene Reconstruction	Crude Mesh, Synthesize image, Mesh Adjustment	Improvement of Real-time Scene Construction	Monocular 3D Reconstruction	CPU: Intel Core i3-2100 GPU: NVIDIA GeForce GT740	CUDA 5.5	Image Resolution, CPU-GPU Implemen-tation	Average Execution Time	Crude Mesh – 19×Synthesize image – 88 × Mesh adjustment – 26 ×
[25]	Anisotropic Huber-L^1	Optical Flow	Almost Completely Mapped	No. of operations permemory access	Flow Field Estimation	CPU: Intel Core i7 4770 GPU: NVIDIA GTX-780 & GT-640	Open CL 1.2	Image Resolution, Bit Precision, CPU-GPU Implementation	Per frame execution time	200-300 ×
[26]	Optimized Fast feature matching	Feature Matching	Build Pyramid computation, Key points matching	Optimization of Time Performance	Stereo Vision	CPU: Intel Core i3 GPU: NVIDIA's Ge-Force 310	CUDA	Image Sequence Considered for Matching & Compared with Lowe's SIFT	Average Execution Time	10 ×
[27]	SRM Feature Extraction	Feature Extraction	Residual computation & co-occurrence matrix calculation	Increase speed of extraction	Steganalysis	CPU: Intel i5-2310 GPU: Radeon HD 6850	Open CL	Image Resolution	Average Execution Time	25-55 ×
[28]	Generalized Fourier Descriptor	Feature Descriptor	Fast Fourier Transform	Compute intensive, Big volume of data, Complex Floating point operations	Real time Object recognition, Classification of Images	CPU: Intel Core i3-2350M GPU: GVIDIA GeForce GT 525M	CUDA (CUFFT), Open CV	Image Resolution, CPU-GPU Implementation	Average Execution Time	18-142 ×

contd. table 1

Reference	*Computer Vision Algorithm*	*Algorithm Type*	*GPU Mapped Portions*	*Need for GPU Impleme-ntation*	*Application Demonstrated*	*GPU Hardware Details*	*Programming Model and Libraries Used*	*Parameters Considered for Comparison*	*Performance Metricused for Evaluation*	*Speed Up In terms of No. of Folds (×)*
[29]	Erosion and Dilation	Binary Morphological Operation	Almost Completely Mapped	For Real-time and High performance operation Optimization on usage of shared, texture and register memory	Object recognition, feature extraction, video processing and automated surveillance systems	CPU: Intel Core i3-2100 GPU: NVIDIA Geforce GTX 480, Geforce GTX 580 FPGA: Xilinx Virtex-5 (FX130T) Xilinx Virtex-6 XC6VLX 240T	CUDA	Parameters related to device utilization, Size of structuring element, Compared with related works, CPU-GPU-FPGA Implementation	Throughput in Mbps, Device utilization	Global memory optimization - 1.87 ×, texture memory optimization - 1.17 × , shared memory optimization - 4.12 × and register optimization - 1.08 ×
[30]	Feature Group Matching Algorithm (FGM)	Feature Extraction and Matching	FGM: Group creation-Calculation of distances, calculation of minimum distance	High Computational time, Computational complexity	Simple Image to Image Feature Matching	CPU: Intel Core i7 3630QM GPU: Nvidia GT650M	CUDA 5.5, Open CV, SIFTGPU	Number of Keypoints per image	Average Execution Time	5-35 ×
[31]	Vertical/ Horizontal Sobel edge detectors & Feature Matching	Stereo Vision	Computing disparity map, Region of Interest Detection	Usage of High Depth and Image Resolution	Pedestrian detection system	CPU:Intel Core 2 Quad GPU: Nvidia GTX260M	Open CV	Different Image Sequences	True Positive Rate and False Positive Rate	No reporting of speed-up
[32]	Harris Corner Detection	Corner Detector	Computation of Integral Images	Mobile Platform Implementation	Corner Detection	CPU:Intel Core i7 CedarTrail Atomchipset, a dual-core D2550GPU: Nvidia Ge Force GTX480 & GTS450	CUDPP	Image Resolution, Different CPUs and GPUs	Frames/Second, Performance/ Watt	12 ×

GPGPU task the challenge is about how to distribute the tasks between CPU and GPU. The GPU acceleration must be quantified through a formal evaluation procedure for which many metrics have been proposed in the literature. Significant metrics for performance evaluation of image processing algorithms on GPUs is presented in [6]. A quantitative performance analysis model for GPU architectures was presented in [22] which not only considers the algorithm but takes the analysis to the level of analyzing the instruction execution at the GPU architecture level. There are various aspects based on which the performance of a GPU based computation may be evaluated. Some of the most common aspects of evaluation are listed in the Table II. The most important understanding as mentioned earlier is the understanding of the possibility of parallelization possibility in an algorithm. Many articles have reported measures of parallelism, some of which are listed in the table. In most of the performance evaluation of GPU-accelerated algorithms, the execution time is the main metric which may be observed in many manifestations as listed in the table. All other metrics meant for basically devised to understand the contributing factors for any computational latency. Some of the factors which is desirable to monitored in a GPU implementation of a computer vision algorithm includes memory access, data transfer related latencies, latencies due to resource utilization, among others. The metrics are generally chosen based on the observers interest in capturing the contribution of the possible factors.

TABLE II
Performance Metrics and Evaluation Aspect

Reference	*Metric*	*Targeted Aspect*
-	Average Execution Time	Execution time
[6]	Parallel Fraction	Execution time (Maximum speed up compared to sequential computation)
[6]	Ratio of floating point computation to global memory access	Memory Access, Resource utilization (Exploiting the resources during memory latency)
[6]	Per-pixel floating point instructions	Resource utilization (Floating point computational resources of the GPU can be exploited)
[6]	Per-pixel memory access	Memory access, Speed up (exploiting memory bandwidth)
[6]	Branching Diversity	Degree of thread parallelism (in-turn Speed up)
[6]	Task Dependency	Speed up (Due to less parallelism)
[22]	Bottleneck component	Execution Time(Ranking of processes taking more time)
[22]	Instruction/Memory Throughput	Execution time
[22]	Computational density	Instruction throughput (will be less due to less resource utilization)
[22]	Coalescing Efficiency	Memory access, Speed up
[22]	Bank Conflicts Penality	Instruction throughput (Due to serial execution caused by serial memory access)
[22]	No. of Warps per Streaming Multi-processor	Memory access
[23]	Memory Latency	Memory access, Resource utilization
[23]	Pipeline Latency	Memory access, Resource utilization
[23]	Floating point Operations/second	Resource utilization

V. CONCLUDING REMARKS

GPU acceleration for general purpose applications has now got numerous tools which have made the programming of GPUs easy though there is a significant development to make it easier is in progress. Moreover the recent developments in combining CPU and GP into one processing element may open up new avenues and tremendous opportunities for general purpose computing. Computer vision can greatly benefit from GPU acceleration but requires a thorough knowledge on the possibility of parallelization of the algorithm under consideration. Commercial computer vision libraries though offer GPU acceleration, are not as dynamic as programming a GPU from scratch. This offers enormous research scope for research into the implementation aspects of popular existing computer vision algorithms.

REFERENCES

[1] M. Arora, “The Architecture and Evolution of CPU-GPU Systems for General Purpose Computing,” pp. 1–12, 2012.

[2] J. Fung and S. Mann, “Using graphics devices in reverse: GPU-based Image Processing and Computer Vision,” *Multimedia and Expo, 2008 IEEE International Conference on*. pp. 9–12, 2008.

[3] C. Mcclanahan, “History and Evolution of GPU Architecture,” pp. 1–7, 2010.

[4] “Juan G´ omez Luna C´ ordoba, Febrero de 2012,” 2012.

[5] D. Castaño-Díez, D. Moser, A. Schoenegger, S. Pruggnaller, and A. S. Frangakis, “Performance evaluation of image processing algorithms on the GPU.,” *J. Struct. Biol.*, vol. 164, no. 1, pp. 153–60, Oct. 2008.

[6] I. K. Park, N. Singhal, M. H. Lee, S. Cho, C. W. Kim, P. In Kyu, N. Singhal, L. Man Hee, C. Sungdae, and C. W. Kim, “Design and Performance Evaluation of Image Processing Algorithms on GPUs,” *Parallel Distrib. Syst. IEEE Trans.*, vol. 22, no. 1, pp. 91–104, 2011.

[7] A. Paz and A. Plaza, “GPU implementation of target and anomaly detection algorithms for remotely sensed hyperspectral image analysis,” *Proc. SPIE*, vol. 7810. p. 78100R–78100R–10, 2010.

[8] T. Preis, P. Virnau, W. Paul, and J. J. Schneider, “GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model,” *J. Comput. Phys.*, vol. 228, no. 12, pp. 4468–4477, Jul. 2009.

[9] R. Dolbeau, S. Bihan, F. Bodin, and C. Entreprise, “HMPP: A Hybrid Multi-core Parallel Programming Environment.” 2007.

[10] S. Wienke, P. Springer, C. Terboven, and D. Mey, “Euro-Par 2012 Parallel Processing: 18th International Conference, Euro-Par 2012, Rhodes Island, Greece, August 27-31, 2012. Proceedings,” C. Kaklamanis, T. Papatheodorou, and P. G. Spirakis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 859–870.

[11] M. Wolfe, “Implementing the PGI Accelerator Model,” in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, 2010, pp. 43–50.

[12] T. D. Han and T. S. Abdelrahman, “hiCUDA: A High-level Directive-based Language for GPU Programming,” in *Proceedings of 2Nd Workshop on General Purpose Processing on Graphics Processing Units*, 2009, pp. 52–61.

[13] S. Lee and R. Eigenmann, “OpenMPC: Extended OpenMP Programming and Tuning for GPUs,” in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1–11.

[14] O. Hernandez, W. Ding, B. Chapman, C. Kartsaklis, R. Sankaran, and R. Graham, “Facing the Multicore - Challenge II: Aspects of New Paradigms and Technologies in Parallel Computing,” R. Keller, D. Kramer, and J.-P. Weiss, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 96–107.

[15] S. Grauer-Gray, W. Killian, R. Searles, and J. Cavazos, “Accelerating Financial Applications on the GPU,” in *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units*, 2013, pp. 127–136.

[16] J. A. Herdman, W. P. Gaudin, S. McIntosh-Smith, M. Boulton, D. A. Beckingsale, A. C. Mallinson, and S. A. Jarvis, “Accelerating Hydrocodes with OpenACC, OpenCL and CUDA,” in *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:*, 2012, pp. 465–471.

[17] S. Lee and J. S. Vetter, “Early Evaluation of Directive-based GPU Programming Models for Productive Exascale Computing,” in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 23:1–23:11.

[18] X. Guo, J. Wu, Z. Wu, S. Member, and B. Huang, “Parallel Computation of Aerial Target Reflection of Background Infrared Radiation/ : Performance Comparison of OpenMP, OpenACC , and CUDA Implementations,” pp. 1–10, 2016.

[19] Y. Allusse, P. Horain, A. Agarwal, and C. Saipriyadarshan, “GpuCV: A GPU-Accelerated Framework for Image Processing and Computer Vision,” *Adv. Vis. Comput.*, vol. 5359, no. 1, pp. 430–439, 2008.

[20] P. Babenko and M. Shah, “MinGPU: a minimum GPU library for computer vision,” *J. Real-Time Image Process.*, vol. 3, no. 4, pp. 255–268, 2008.

[21] J. Fung, S. Mann, and C. Aimone, “OpenVIDIA: Parallel GPU Computer Vision,” *Image Process.*, pp. 849–852, 2005.

[22] Y. Zhang and J. D. Owens, “A Quantitative Performance Analysis Model for GPU Architectures.pdf,” pp. 382–393, 2002.

[23] V. Volkov and J. Demmel, “Benchmarking g GPUs to Tune Dense Linear Algebra,” *High Perform. Comput. Networking, Storage Anal.*, no. November, pp. 1–11, 2008.

[24] Y. Ailin, L. Xiuzhi, J. Songmin, and Q. Baoling, “Monocular three dimensional dense surface reconstruction by optical flow feedback,” in *Information and Automation, 2015 IEEE International Conference on*, 2015, pp. 504–509.

[25] D. Buyukaydin and T. Akgun, “GPU Implementation of an Anisotropic Huber- L 1 Dense Optical Flow Algorithm Using OpenCL,” *Samos*, no. 2, pp. 1–6, 2015.

[26] K. Sharma, “High performance GPU based optimized feature matching for computer vision applications,” *Opt. - Int. J. Light Electron Opt.*, vol. 127, no. 3, pp. 1–7, 2015.

[27] K. Chen, C. Lin, S. Zhong, and L. Guo, “A Parallel SRM Feature Extraction Algorithm for Steganalysis Based on GPU Architecture,” *2014 Sixth Int. Symp. Parallel Archit. Algorithms Program.*, pp. 178–182, 2014.

[28] B. Haythem, H. Mohamed, C. Marwa, S. Fatma, and A. Mohamed, “Fast Generalized Fourier Descriptor for object recognition of image using CUDA,” in *Computer Applications & Research (WSCAR), 2014 World Symposium on*, 2014, pp. 1–5.

[29] T. Li, Y. Dou, J. Jiang, and J. Gao, “Efficient parallel implementation of morphological operation on GPU and FPGA,” *Proc. 2014 IEEE Int. Conf. Secur. Pattern Anal. Cybern. SPAC 2014*, pp. 430–435, 2014.

[30] M. Marinelli, A. Mancini, and P. Zingaretti, “GPU acceleration of feature extraction and matching algorithms,” in *Mechatronic and Embedded Systems and Applications (MESA), 2014 IEEE/ASME 10th International Conference on*, 2014, pp. 1–6.

[31] B. Nam, S. Kang, and H. Hong, “Pedestrian Detection System based on Stereo Vision for Mobile Robot,” *Adv. Imaging*.

[32] A. Glenis, “Performance and energy characterization of high-performance low-cost cornerness detection on GPUs and multicores.”