Real-Time Semi-Global Matching Disparity Estimation on the GPU

Christian Banz, Holger Blume, and Peter Pirsch
Institute of Microelectronic Systems
Leibniz Universität Hannover, Hannover (Germany)
{banz,blume,pirsch}@ims.uni-hannover.de

Abstract

This paper presents the design, implementation and evaluation of new parallelization schemes for performing dense disparity estimation based on non-parametric rank transform and semi-global matching on Graphics Processing Units (GPUs). A detailed analysis of the performance limitating factors (memory throughput, instruction throughput, etc.) for each part of the parallel implementation is performed. Thus, a highly optimized mapping for each parallelization scheme onto the resources of the GPU is obtained. The resulting implementation performs disparity estimation at 27 *frames per second for* 1024×768 *pixel images with* 128 *disparity levels on a Nvidia Tesla C2050 GPU.*

1. Introduction

Robust and dense computation depth information from passive stereo-camera systems is a frequently required, but computationally demanding task in 3D video processing. It is necessary in many (automated) machine vision applications such as *e.g.* driver assistance systems, robotics or applications which require 3D information as real-time feedback. The variety of the latter applications range from keyhole surgery to subtitle placing in 3D movies.

Stereo vision is continuing to be an active topic of research resulting in a growing range of disparity estimation algorithms which have been extensively studied and benchmarked [10] [17] [18]. The semi-global matching (SGM) algorithm by Hirschmüller [9] is among the top-performing algorithms. The combination of rank transform and semi-global matching has been shown to be insensitive to various types of noise and a number of other interferences such as lighting, exposure and gamma differences. Further, this combination is able to deal with large untextured areas and retains edges [10]. This advantageous performance is due to the fact that SGM performs an energy minimization along several 1D paths across the image and, thus, approximates the otherwise two-dimensional NP-complete energy minimization problem.

Nevertheless, high computational loads, extremely high memory bandwidth requirements, and special data access patterns pose challenges for fast implementations. For image sizes providing sufficient details, real-time performance, e.g. camera frame rate, has not been reached on standard processors (CPUs) or even digital signal processors (DSPs). Implementations on SIMD-CPUs have been reported reaching 1.4fps for QVGA resolution [8] and 14fps for 640×320 image pairs (based on a depth adaptive subsampling) [4]. However, it has been shown in [1] and [15] that dense disparity estimation based on SGM can be performed in real-time on dedicated hardware (e.g. FPGA or ASIC). The disadvantage of dedicated hardware solutions is the lack of flexibility regarding changes in the algorithms.

Graphics processing units (GPUs) are increasingly addressing applications with extremely high computational demands [14]. GPUs offer full high-level programmability allowing rapid development at highest performance per price and lowest power consumption per FLOP [20] . Thus, applications previously running only on dedicated hardware become available on high-level programmable off-the-shelf hardware components where overall power consumption is not an issue. GPUs have undergone rapid development providing today drastically more general purpose computation power and memory bandwidth than just a few years ago. With the advances in GPU technology matching parallelization schemes have been developed and evaluated (eg [12]).

Consequently, efforts to use GPUs for SGM-based disparity estimation have already been made. In 2008, Gibson and Marques presented an implementation reaching 8fps for 320×240 pixel (*px*) with 64 px disparity range on a Nvidia G80 GPU [5]. Ernst and Hirschmüller (2008) presented an implementation reaching 4.9 fps for 640×480 images and 128 px disparity range on a GeForce 8800 ULTRA [3]. A implementation on a Nvidia GTX 280 reaching 53 fps for 512×383 at 56 px disparity range was introduced by Haller and Nedevschi [6]. Even though GPU implementations exist, they do no reach the required performance or resolution and of course cannot take into account the recent significant advances in GPU programming models and GPU hardware.

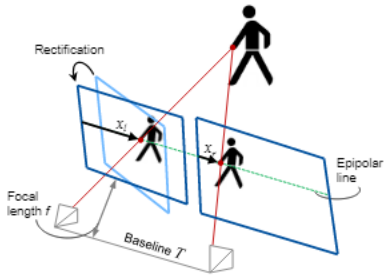

Figure 1. Epipolar geometry of a stereo camera setup with the one-dimensional search space for corresponding pixels. The rectification to achieve a row-aligned search-space is only shown for the left projection plane.

It is therefore necessary to reevaluate existing and develop new parallelization schemes for SGM for current high-end GPUs.

This contribution presents the design and evaluation of new parallelization schemes for the semi-global matching disparity estimation algorithm for high performance GPUs. Based on these approaches, and using the CUDA environment, an implementation on a Nvidia Tesla C2050 GPU is presented. Special focus has been put on a detailed analysis as to which bounds (compute power, memory bandwidth, etc.) limit the performance of the respective part of the algorithm. This allows the formulation of a design paradigm which is used to obtain efficient solutions for each part of the algorithm, thus, maximizing performance.

Section 2 briefly reviews algorithmic background on semi-global matching and Section 3 reviews the most important aspects of GPU hardware. In Section 4 performance bounds and efficiency metrics are discussed. Design, implementation and performance evaluation of the parallelization approaches are presented in Section 5. Finally, conclusions are drawn in Section 6.

2. Semi-Global Matching

Disparity estimation is the task of identifying the projection point of the same 3D real-world point in two or more images taken from distinct viewpoints. The disparity d is the location difference of both projection points in the stereo images (see Fig. 1). The main challenge for disparity estimation algorithms is to identify corresponding projection points, as a very high level of ambiguity exists. Once the disparity d has been obtained, the calculation of the distance z between the 3D point and the baseline of the cameras is straight-forward:

$$z = fT/(x_l - x_r) = fT/d \tag{1}$$

where f and T are the rectified focal length and the baseline of the camera pair, respectively. Due to the underlying epipolar geometry [18] of a pre-rectified stereo image pair, the search space for corresponding pixels is oriented along the epipolar lines.

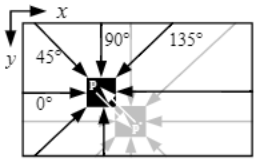

Figure 2. Path orientations using 8 paths for pixel $\mathbf{p}$ shown in black. When moving along the diagonal paths between pixel $\mathbf{p}$ and $\mathbf{p}_2$ the other paths directions are not immediately available resulting in complex memory access patterns.

In most stereo matching methods, a similarity measure between two pixels in the base and match image (or left and right image, respectively) is calculated and those with highest accordance or correlation are assigned as corresponding. In this work the similarity measure is calculated in three steps: matching cost calculation, path cost aggregation, and cost summation. The matching costs $C(\mathbf{p}, d)$ between a pixel $\mathbf{p} = [x, y]^T$ in the base image and its potentially corresponding pixels in the match image are calculated as

$$C(\mathbf{p}, d) = |R_l(x, y) - R_r(x - d, y)| \tag{2}$$

where R is the area-based non-parametric rank-transform [21]. It is defined as the number of pixels $\mathbf{p}'$ in a square $M \times M$ neighborhood $A(\mathbf{p})$ of the center pixel $\mathbf{p}$ with an intensity $I(\mathbf{p}')$ less than $I(\mathbf{p})$.

$$R(\mathbf{p}) = \|\{\mathbf{p}' \in A(\mathbf{p}) \mid I(\mathbf{p}') < I(\mathbf{p})\}\|. \tag{3}$$

In order to deal with non-unique or wrong correspondences due to low texture and ambiguity, the semi-global matching introduces consistency constraints by aggregating matching costs along several independent, one-dimensional paths across the image [9]. An example for eight path orientations, as used in this work, is shown in Fig. 2. The path costs $L_r(\mathbf{p}, d)$ are aggregated along a path $\mathbf{r}$ according to

$$\begin{aligned} L_r(\mathbf{p}, d) = C(\mathbf{p}, d) + \min\big[&L_r(\mathbf{p} - \mathbf{r}, d), \\ &L_r(\mathbf{p} - \mathbf{r}, d - 1) + P_1, \\ &L_r(\mathbf{p} - \mathbf{r}, d + 1) + P_1, \\ &\min_i L_r(\mathbf{p} - \mathbf{r}, i) + P_2\big] - \\ &\min_l L_r(\mathbf{p} - \mathbf{r}, l) \end{aligned} \tag{4}$$

The first term describes the primary matching costs. The second term adds the minimal path costs of the previous pixel $\mathbf{p} - \mathbf{r}$ including a penalty P_1 for disparity changes and P_2 for disparity discontinuities, respectively. Discrimination of small changes $|\Delta d| = 1$ pixel *(px)* and discontinuities $|\Delta d| > 1$ px allows for slanted and curved surfaces on the one hand and preserves disparity discontinuities on the

other hand. In this work P_1 and P_2 are empirically determined constants. The last term prevents constantly increasing path costs. For a detailed discussion refer to [9]. S is the sum of the path costs L_r over all paths

$$S(\mathbf{p}, d) = \sum_{\mathbf{r}} L_r(\mathbf{p}, d). \quad (5)$$

The disparity maps $D_b(p_x, p_y)$ from the perspective of the left camera is calculated by selecting the disparity with the minimal aggregated costs for each pixel (*winner-takes-all*). It is

$$\min_d S(p_x, p_y, d) \quad (6)$$

As a post-processing step a median filter is applied to the disparity map to suppress outliers.

3. Background on the GPU

This work utilizes the Nvidia Cuda environment and, therefore, its corresponding terminology [13]. Nvidia GPUs consists of number of highly decoupled processing cores (streaming multiprocessors) in turn consisting of a set of tightly coupled ALUs (cuda cores) and special functional units. On the Fermi architecture a so-called warp is formed of 32 parallel threads which are executed on a streaming multiprocessor. Several warps form a block of threads. Computation on the cuda cores is highly pipelined. Each streaming multiprocessor (SM) is limited by the available hardware resources, *e.g*. amount of shared memory, number of registers, and the number of different blocks as well as threads it can handle. The function that is executed as a set of parallel threads is called a kernel. Threads of the same block can be synchronized and can share data via the fast, local on-chip shared memory. Blocks (while executing the same kernel) are launched independently from each other, are not synchronizable, and the execution order is nondeterministic. The launch configuration determines how threads are grouped into blocks and influences the distribution of the blocks onto the SMs and, thus, availability of the hardware resources. For efficient memory access it must be aligned and coalesced, which basically means that all threads of a warp must access a set of neighboring addresses. For further details on the Nvidia GPU architectures refer to [13]. A review on GPU-based implementations *e.g*. for video coding and image registration is given in [2] and [20]. More hands-on examples of how to efficiently utilize the hardware resources and optimize for high throughput are given in the Nvidia whitepapers [16] and [7].

The evaluation platform is a Nvidia Tesla C2050 with compute capability 2.0 providing 3 GB DDRRAM global memory with a maximum theoretical bandwidth of 144 GB/s. The streaming multiprocessors run at 1.15 GHz featuring per processor shared memory configurable to 16 kB or 48 kB size at the expense of L1 cache size. The maximum theoretical instruction throughput is 2.0 instructions per clock cycle (per cuda core) due the dual warp scheduler. Compared to previous generations several improvements for general purpose computing have been introduced. These include the dual warp scheduler, branch predication to avoid warp divergence, optional concurrent kernel execution, improved context switching and others. These important new features must be taken into account and actively employed when designing for highest performance image processing on GPUs. Here, the employed tools are Nvidia Toolkit 4.0, Compute Visual Profiler 4.0 and Parallel Nsight 2.0.

4. Performance Bounds and Optimization Techniques

The performance of a kernel is limited by one or more of the following factors:

- **Effective memory bandwidth usage** for the payload data which is *e.g*. reduced by nonaligned, overhead-producing memory access
- **Instruction throughput** consisting of instructions performing arithmetics for the core computation and non-ancillary instructions such as address calculation and loop overhead
- **Latency of the memory interface** occurring *e.g*. when accessing scattered memory locations even if aligned and coalesced, warp-wise access is performed
- **Latency of the arithmetic pipeline** of the cuda cores if arithmetic instructions depend on each other and can only be executed with the result from the previous instruction

Accordingly, kernels can be memory bound, compute bound or latency bound. Kernels that are not limited by any of the three bounds are ill-adapted for GPU implementation and can be classified as bound by their parallelization scheme. Only a metric for memory bandwidth and instruction throughput can be directly profiled with the tools. Efficiency of a kernel in terms of latency hiding and instruction throughput of non-ancillary instructions must be deduced from the measurements and the execution time. Furthermore, the bandwidth usage does not express how efficiently the data is used, *i.e*. the percentage of redundant memory access. Therefore, for high-throughput processing the following principles can be formulated:

- Perform high data reuse to avoid redundant memory access
- Facilitate a high computational instructions to ancillary instructions ratio
- Keep the hardware pipeline filled to hide latency
- Ensure coalesced and aligned memory access

- Facilitate the transfer of coherent chunks of data to minimize the effect of memory latency

The best-performing implementation will consist of a trade-off between these principles which must be weighted according to the algorithm to be implemented. Nevertheless, for devising a parallelization scheme which address all three bounds and employs the five principles the following can be formulated:

An efficient parallelization scheme guarantees inherently aligned and coalesced data access schemes without instruction overhead and includes a combination of parallel and sequential processing with independent arithmetic computation steps. An inner (sequential) loop in the otherwise parallel threads working on a set of data that is kept in shared memory or register facilitates data reuse, increases the instruction ration, and keeps the pipeline filled. Further, coherent access schemes are ensured for the memory interface if results are written out with each loop iteration.

Apart from an inner loop, executing several warps per SM increases pipeline utilization. This can be measured by the occupancy, a metric giving the ratio of executed warps to the maximum manageable warps per streaming multiprocessor. However, it can only be used as a weak metric since it is neither necessary nor a guarantee for efficient use of the hardware pipeline.

5. Parallelization, Implementation and Performance Study

In the following the parallelization principles derived in Sec. 4 are applied to the disparity estimation algorithm. Rank transform and median filter are handled together since they exhibit the same data access schemes. Afterwards, a parallelization for the semi-global matching core will be discussed.

5.1. Rank Transform and Median Filter

The rank transform and median filter are both non-linear, non-separable 2D image transforms. To generate the result of one output pixel, the data of a local $N \times N$-neighborhood from the input image is required (see Eq. (3)). For this class of transforms, it is often suggested to use the GPU texture memory optimized for 2D spatial access. Therefore, texture memory based implementations will be used as a reference.

A new kernel for the non-separable filter compliant with the principles introduced in Sec. 4 is proposed based on the implementation of a separable convolution in [16]. The kernel pre-fetches data of a close two-dimensional spatial locality from global memory into shared memory. Thus, data reuse is maximized because all filter kernels that fully reside in this spatial locality can be processed by a block of threads without additional global memory access. An aligned group of pixels is processed by a two-dimensional

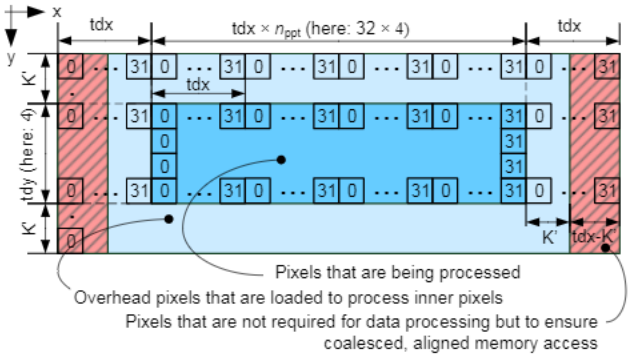

Figure 3. Data fetching and accessing scheme for the 2D filter kernels processing $tdx \cdot n_{\text{ppt}} \times tdy$ kernel windows with a radius of K' where n_{ppt} is the number of pixels processed per thread and the launch configuration is $tdx \times tdy$. Each square represents a pixel and the number inside is the x-dimension thread ID which fetches the pixel from global memory.

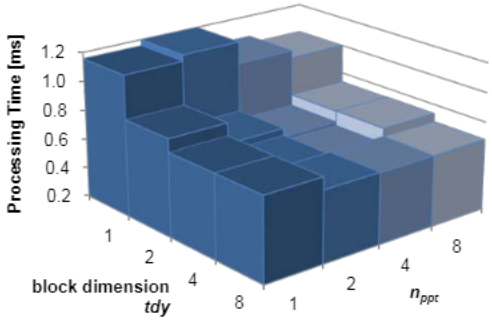

Figure 4. Performance of the 3×3 median filter on 1280×960 images as the parallelization configuration changes. Block width is fixed to $tdx = 32$. Best performance is achieved with $tdx \times tdy = 32 \times 4$ and $n_{\text{ppt}} = 4$.

block of threads first loading the neighboring center pixels of all kernels. Left and right pixels outside the center area are always loaded with the warp width. Even though this causes minimal data to be loaded which is not used by the current block, it ensures inherent coalesced memory access without instruction overhead or warp divergence. An inner loop allows to process several pixels per thread (n_{ppt}) with a stride of the warp width. Adjusting the number of threads per block in x-dimension (tdx), y-dimension (tdy) and (n_{ppt}) allows to navigate between the different optimization principles. Fig. 3 shows the data layout and thread access scheme.

The median filter is always compute bound and performs best with $tdx \times tdy = 32 \times 4$ threads and $n_{\text{ppt}} = 4$. The results of the parameter study for $tdx = 32$ are shown in Fig. 4. Configurations with $n_{\text{ppt}} = 8$ perform slightly worse although redundant memory access is further reduced because of inefficient pipeline utilization. Processing times for a 3×3 median filter (*i.e.* kernel radius $K' = 1$) are given in Fig. 5 resulting in 0.64 ms for the shared memory based ker-

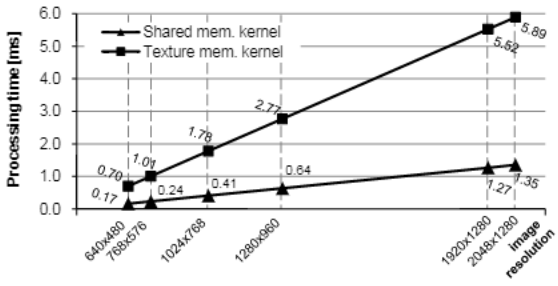

Figure 5. Performance of the 3×3 median filter using the texture memory kernel and the proposed shared memory kernel on a Tesla C2050 GPU for the best-performing parallelization configuration.

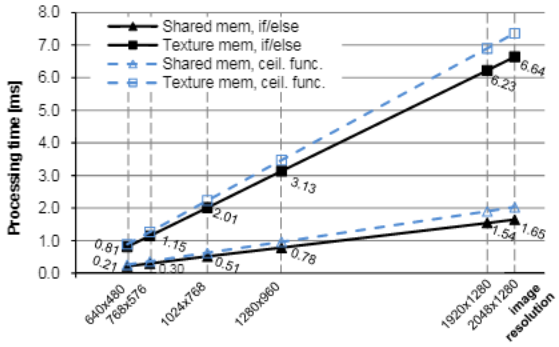

Figure 6. Performance calculating the 9×9 rank transform on a Tesla C2050 GPU using the texture memory kernel and the proposed shared memory kernel. Two calculation descriptions have been evaluated: `if/else` and `ceil. func.`

nel and 2.77 ms for the texture-memory-based kernel which is a speed-up of 4.3 when processing a 1280×960 image.

For the rank transform two core computation descriptions were evaluated. The first is a direct implementation based on Eq. (3) using `if` constructs. The second implementation uses the in-build `__ceil` function to avoid warp divergence. Assuming the input data is scaled to the range of $[0 \dots 1]$ (which is realizable on-the-fly), Eq. (3) can be expressed as

$$R'(\mathbf{p}) = \sum_{\mathbf{p}' \in A(\mathbf{p})} \lceil I(\mathbf{p}') - I(\mathbf{p}) \rceil \qquad (7)$$

For a 9×9 rank transform (*i.e.* $K' = 4$) experiments showed that a block size of $tdx \times tdy = 32 \times 4$ with $n_{\text{ppt}} = 4$ yields best performance for both implementations. Results are given in Fig. 6. On Fermi architecture GPUs the direct if-based implementation always performs slightly better. A speed up of 4.0 is obtained switching from the texture-based kernel (3.13 ms) to the shared memory kernel (0.78 ms) for 1280×960 images. It is noteworthy that on GT200 architecture GPUs the ceil-based computation (4.0 ms using texture memory) significantly outperforms the if/else implementation (62.6 ms). This example emphasizes the need to consider new architectural features when designing new implementations for current GPUs.

5.2. Semi-Global Matching

For every pixel location $\mathbf{p}$, calculation of the matching cost $C(\mathbf{p}, d)$ according to Eq. (2) results in a vector with one entry for each disparity level d. Thus, the spatial directions (x and y) and the disparity range span a three-dimensional space. The matching cost (*MC*) calculation for every point in this space can be performed independently allowing for parallelization in all three dimensions.

A straightforward parallelization is to assign each thread with the calculation of one entry in the 3D cost space of $C(\mathbf{p}, d)$. This kernel (`mc_unaligned`) reaches 16.3 ms and 48.6 GB/s which is far from the bandwidth limit due to inefficient, often misaligned memory access, lack of data reuse, and little latency hiding possibilities (see Sec. 4). This kernel is latency bound which can only be eliminated by a new parallelization scheme.

The new kernel (`mc_proposed`) processes all disparity levels of a group of tdx neighboring pixels synchronously in tdx threads. The disparity dimension itself is further separated into tdy groups each processing d_{range}/tdy disparity levels with an inner loop in the kernel. By adjusting tdy thread parallelism is substituted with inner loop complexity. Pixels from the base image are read aligned and coalesced over the tdx threads. The required pixels from the right image are loaded in groups of tdx aligned, coalesced pixels into the shared memory where they can be accessed and reused by all threads. The parallelization scheme is shown in Fig. 7. Choosing tdx as a multiple of the warp size (*i.e.* 32) results in always aligned memory access. This kernel adheres the optimization approach of Sec. 4 by providing inherently aligned memory access, high data reuse, and efficient use of the arithmetic pipeline. With an obtained performance of 6.1 ms and 126.2 GB/s this is a memory bound kernel providing a speed-up of factor 2.7.

Data reduction provides the only means for further speed-up. Based on a 9×9 rank kernel the maximum matching cost value is $C_{\max} = 80 < 2^8$ for which 8 bit are sufficient. Since performing arithmetic in non-native GPU data types (*i.e.* other than 32-bit integer and float) is slow, input images and computation are based on 32-bit integer and type conversion to `uchar` is performed just before writing out the result. To maintain coalesced, aligned data access over all threads, cost values are only written to memory every four iterations using the in-built packed `uchar4` data type. This produces a folded data layout of the cost space which can be ignored if all further kernels use the same parallelization scheme and, thus, inherently the same data layout. A performance of 1.8 ms and 111.2 GB/s is achieved providing further speed-up of 3.4 and a combined speed-up

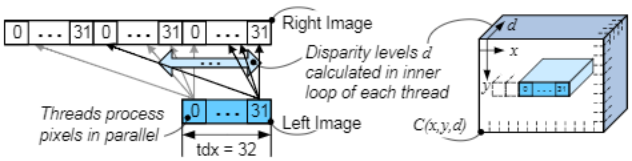

Figure 7. Memory access scheme for calculating the matching costs for tdx pixel parallelly in a tdx-thread wide warp and $tdy = 1$. The location of the results in the 3D matching cost space is shown. Again the numbers in the squares represent the thread ID that fetches the according pixels from global memory.

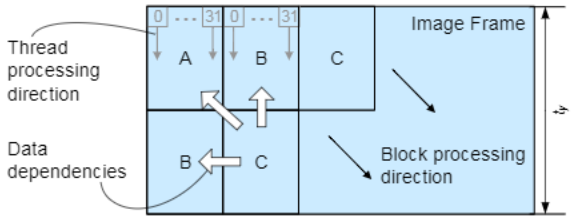

Figure 8. Image tiling for the 45° path and $t_y = 2$ allowing divergent processing direction and path direction while tiles with the same letter can to be processed in parallel. The processing direction ensures coalesced and aligned memory access.

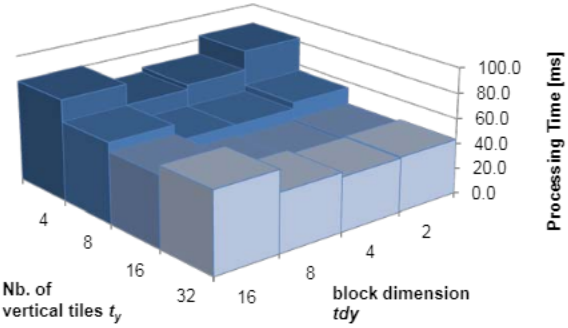

Figure 9. Performance of the concurrent path cost calculation for 8 paths of the SGM for 1280×960 images and 128 disparity levels as the parallelization configuration changes. Block width and tile width are both fixed to $tdx = 32$. Best performance is achieved with $tdx \times tdy = 32 \times 4$ (*i.e.* each inner loop processes 32 disparity levels) and $t_y = 16$.

of 9.2. A performance summary is given in Table 1.

The path costs (PC) calculation according to Eq. (4) is performed by individually traversing along each of the eight path directions updating the matching cost values and resulting in a new 3D cost space for each path direction. PC calculation must be done sequentially along the respective path direction (*e.g.* from left to right) because the previous pixel's path costs must be known. This limits parallelization by one dimension compared to the MC calculation. For a single path it is $L_{r,\max} \leq C_{\max} + P_2 \leq 2^8$ also allowing the use of character precision data storage. The parallel minimum search over the disparity levels has been implemented similar to the parallel reduction scheme from [7]. Although the MCs are common to all PC directions and it seems obvious to separate MC and PC calculation, it is faster to integrate MC and PC calculation and recalculate the MCs on-the-fly for each PC direction. This drastically reduces pressure on the performance-limiting memory bandwidth since the MC data is never transfered via the external memory but can be kept locally in shared memory.

Therefore, each path cost direction is processed in its own kernel using the highly efficient memory access scheme from the MC kernel. All eight path directions are executed concurrently using the CUDA concurrent kernel execution on Fermi architectures. This effectively compensates for the fact that parallelization of one path is limited. Due to the coalesced memory access obligation only a group of horizontally neighboring pixels can be efficiently accessed in memory. The path costs kernels must be modified according to their path direction in order to maintain efficient memory access. For the non-diagonal paths, computation is performed sequentially along the path direction.

For each diagonal path direction, processing is separated into rectangular tiles. Within each tile the processing direction is along the image columns, *i.e.* misaligned to the path direction, but ensuring aligned memory access. Path costs at the tile borders are transfered via an aligned scratch space in global memory causing a negligible memory overhead depending on the tiling granularity (t_x and t_y). Tiles not sharing data dependencies can be processed in parallel as independent thread blocks. This is similar to the intrablock encoding scheme for video streams proposed in [11]. An example of the parallel processing order is shown in Fig. 8. Since block synchronization does not exist on GPUs, correct execution order is established by sequentially launching a kernel for each diagonal tile front causing some minor time overhead.

Overall, two major parameters influence the parallelization configuration and, thus, the mapping onto the GPU hardware. Adjusting these parameters allows to navigate between the performance optimization principles detailed in Sec. 4. The first parameter (tdy) trades thread parallelism against sequential computation in the inner loop for all kernels. The second parameter (t_y) trades the number of parallelly processable blocks versus launch overhead and memory overhead for the four diagonal paths. Since all kernels are executed concurrently, interactions between these parameters must be observed. Fig. 9 shows the result of the parameter study. Choosing $tdy = 4$ and $t_y = 16$ results in best performance (39.8 ms and 39.7 GB/s) for a 1280×960 image. If the concurrent kernel execution is not used, performance is approximately halved (75.7 ms and 20.9 GB/s). Both kernel sets, concurrent and sequential, are latency bound.

Summation of the eight path cost spaces (5) and winner-takes-all disparity selection (6) can be performed indepen-

Kernel	Time [ms]	Bandwidth [GB/s]	B
MC unaligned	16.32	48.6	P
MC proposed (`float`)	6.12	126.2	M
MC proposed (`uchar4`)	1.80	107.3	L
MC+PC 8 path dir. (sequen.)	75.68	20.9	L
MC+PC 8 path dir. (concur.)	39.81	39.7	L
Sum, WTA left disp. map	15.09	117.4	M

Table 1. Performance results of the optimized kernels (MC: matching costs, PC: path costs, WTA: winner-takes-all) with optimal launch configuration for computing the semi-global matching algorithm for images with 1280×960 pixels and 128 disparity levels. *B* is the performance bound abbreviated as M: memory bandwidth, C: compute, and L: latency.

Image Size	$d_{\text{range}} = 64$	128	256
640×480	9.7 ms	16.0 ms	29.0 ms
1024×768	21.5 ms	35.9 ms	67.1 ms
1280×960	32.9 ms	56.2 ms	105.7 ms

Table 2. Performance results for the entire disparity estimation algorithm using rank transform, semi-global matching and median filtering on a Nvidia Tesla C2050 GPU. Results are k-mean values over multiple runs and images.

dently for each pixel allowing for the same parallelization scheme as for the MC calculation. This kernel (`sum_wta`) requires 15.1 ms and is memory bound with 117.4 GB/s. An overview of all processing steps for disparity estimation as partitioned for the GPU implementation is given in Fig. 10.

5.3. Overall Performance and Evaluation

The processing time for the complete disparity estimation including rank transform, semi-global matching for 8 paths, disparity map generation and median filtering on a Tesla C2050 Fermi architecture GPU is summarized in Table 2. Overall, a 1280×960 image with 128 disparity levels requires 56.2 ms. The resulting disparity map is shown in Fig. 11.

The processing times do not include data transfer between host and GPU because it can be effectively hidden using concurrent data transfer when processing image streams. When processing 1280×960 image sets ca. 5 ms additional transfer time is required.

A summary of different implementations targeting GPUs and other hardware architectures is given in Table 3. Even though all chosen references use the original SGM algorithm, different matching costs algorithms have been employed. Performing an quantitative performance comparison based on normalized throughput is a complex problem because the throughput depends on the type of employed

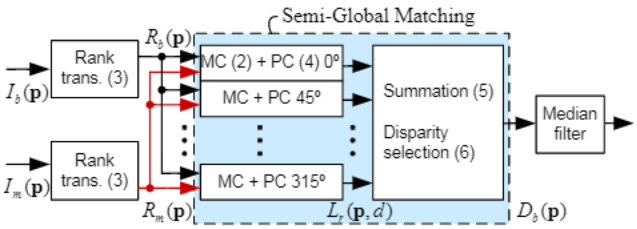

Figure 10. Partitioning of the disparity estimation algorithm into separate kernels as implemented on the GPU. Bracketed numbers refer to the corresponding equations.

Figure 11. (a) Left input image of the cones image pair [19] and (b) gray-scale representation of the disparity map computed with the presented implementation on the GPU.

parallelization and not only on the performed arithmetic operations. However, all GPU implementations significantly outperform the CPU implementation.

The proposed implementation outperforms all other GPU implementations in terms of processing time. Of course different GPUs with significantly different capabilities have been used. Nevertheless, the good performance is not only due to using a high-end Fermi architecture GPU but also of highly optimized kernels employing the parallelization schemes introduced above. The efficiency of the implementation has been shown by detailed analysis in the previous section.

Even the implementations using specialized and partially dedicated hardware resources (FPGA and customized DSP) are outperformed in terms of processing time. Power consumption of GPUs, however, is significantly higher by orders of magnitude. It's importance can only be rated considering the target application field (*e.g.* embedded systems vs. mainstream computing systems).

6. Conclusions

The performance results and evaluation show that the proposed parallelization schemes for the semi-global matching disparity estimation algorithm enable real-time processing with more than 25 fps of images with a resolution of 1024×768 and 128 disparity levels on Nvidia Fermi architecture GPUs. This allows application in fields where high image resolution is mandatory. The detailed analy-

Implementation	Architecture	Algorithm (# paths)	Image Size	Proc. Time
Proposed here	Tesla C2050	RT+SGM(8)+MF	640×480×128	16 ms
Haller and Nedevschi [6]	GTX 280	CT+SGM(4)	512×383×56	19 ms
Ernst and Hirschmüller [3]	GeForce 8800 Ultra	HMI+SGM(8)	640×480×128	175 ms
Gibson and Marques [5]	Quadro FX5600	BT+SGM(8)	450×375×64	170 ms
Banz *et al.* [1]	Virtex-5 FPGA	RT+SGM(4)+MF	640×480×128	31 – 13 ms
Paya Vaya *et al.* [15]	Custom DSP	RT+SGM(4)	640×480×64	33 ms
Hirschmüller [8]	SMD Opteron CPU	HMI+SGM(16)+MF	450×375×64	1800 ms

Table 3. Summary of different SGM implementations. The different matching cost functions are rank transform (RT), census transform (CT), mutual information (MI) and Birchfeld-Tomasi (BT). (MF) denotes the median filter.

sis of the GPUs performance bounds provided valuable insights which enabled the design of highly optimized kernels. Moreover, the performance bound analysis and parallelization principles can be applied to design new parallelization schemes for other image processing algorithms. The speed-up in the processing time of more than 4 over existing GPU implementations shows that with advances in GPU technology it is crucial to devise according parallelization schemes to fully exploit the performance optimizations possible.

Future work includes the implementation of integrity checks such as the left/right-check and subpixel calculation without increasing pressure on the latency performance bound.

References

[1] C. Banz, S. Hesselbarth, H. Flatt, H. Blume, and P. Pirsch. Real-time stereo vision system using semi-global matching disparity estimation: Architecture and fpga-implementation. In *IEEE Intl. Conf. Embedded Computer Systems: Architectures, Modeling and Simulation*, pages 93–101, 2010. 1, 8

[2] N.-M. Cheung, X. Fan, O. C. Au, and M.-C. Kung. Video coding on multicore graphics processors. *IEEE Signal Processing Magazine*, 27(2):79–89, 2010. 3

[3] I. Ernst and H. Hirschmüller. Mutual information based semi-global stereo matching on the GPU. *Advances in Visual Computing*, 5358:228–239, 2008. 1, 8

[4] S. K. Gehrig and C. Rabe. Real-time semi-global matching on the CPU. In *Proc. IEEE Comp. Soc. Conf. Computer Vision and Pattern Recognition Workshops*, pages 85–92, 2010. 1

[5] J. Gibson and O. Marques. Stereo depth with a unified architecture GPU. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition Workshops*, pages 1–6, 2008. 1, 8

[6] I. Haller and S. Nedevschi. GPU optimization of the SGM stereo algorithm. In *Proc. IEEE Int Intelligent Comp. Communication and Processing*, pages 197–202, 2010. 1, 8

[7] Harris. Optimizing parallel reduction in CUDA. Whitepaper included in Nvidia Cuda SDK 4.0, 2007. 3, 6

[8] H. Hirschmüller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *Proc. IEEE Comp. Soc. Conf. Computer Vision and Pattern Recognition*, volume 2, pages 807–814, 2005. 1, 8

[9] H. Hirschmüller. Stereo Processing by Semiglobal Matching and Mutual Information. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 30(2):328–341, 2008. 1, 2, 3

[10] H. Hirschmüller and D. Scharstein. Evaluation of Cost Functions for Stereo Matching. *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 1–8, 2007. 1

[11] M. C. Kung, O. C. Au, P. H. W. Wong, and C. H. Liu. Block based parallel motion estimation using programmable graphics hardware. In *Proc. Intl. Conf. Audio, Language and Image Processing*, pages 599–603, 2008. 6

[12] D. Lin, X. Huang, Q. Nguyen, J. Blackburn, C. Rodrigues, T. Huang, M. N. Do, S. J. Patel, and W.-M. W. Hwu. The parallelization of video processing. *IEEE Signal Processing Magazine*, 26(6):103–112, 2009. 1

[13] Nvidia. *Nvidia CUDA C Programming Guide*, 2011. 3

[14] J. Owens. GPU computing: Heterogeneous computing for future systems. In *Proc. IEEE Intl. Symp. Parallel & Distributed Processing*, 2009. 1

[15] G. Payá-Vayá, J. Martín-Langerwerf, C. Banz, F. Giesemann, P. Pirsch, and H. Blume. VLIW architecture optimization for an efficient computation of stereoscopic video applications. In *IEEE Intl. Conf. Green Circuits and Systems*, pages 457 –462, 2010. 1, 8

[16] V. Podlozhnyuk. Image convolution with CUDA. Whitepaper included in Nvidia Cuda SDK 4.0, 2007. 3, 4

[17] D. Scharstein and R. Szeliski. The Middlebury Stereo Pages. `http://vision.middlebury.edu/stereo/`. 1

[18] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Intl. Journal of Computer Vision*, 47(1):7–42, 2002. 1, 2

[19] D. Scharstein and R. Szeliski. High-accuracy stereo depth maps using structured light. *Proc. IEEE Conf. Comp. Vision and Pattern Recognition*, 1:I–195–I–202 vol.1, 2003. 7

[20] R. Shams, P. Sadeghi, R. Kennedy, and R. Hartley. A survey of medical image registration on multicore and the GPU. *IEEE Signal Processing Magazine*, 27(2):50–60, 2010. 1, 3

[21] R. Zabih and J. Woodfill. Non-parametric local transforms for computing visual correspondence. In *European Conference on Computer Vision*, pages 151–158, 1994. 2