A Scalable High-Performance Hardware Architecture for Real-Time Stereo Vision by Semi-Global Matching

Jaco Hofmann, Jens Korinth, and Andreas Koch

Embedded Systems and Applications Group (ESA)
Technische Universität Darmstadt, Germany
{jah,jk,ahk}@esa.tu-darmstadt.de

Abstract

Perceiving distance from two camera images, a task called stereo vision, is fundamental for many applications in robotics or automation. However, algorithms that compute this information at high accuracy have a high computational complexity. One such algorithm, Semi Global Matching (SGM), performs well in many stereo vision benchmarks, while maintaining a manageable computational complexity. Nevertheless, CPU and GPU implementations of this algorithm often fail to achieve real-time processing of camera images, especially in power-constrained embedded environments. This work presents a novel architecture to calculate disparities through SGM. The proposed architecture is highly scalable and applicable for low-power embedded as well as high-performance multi-camera high-resolution applications.

1. Introduction

Allowing computers to perceive their environment is still one of the most challenging tasks in computer vision. Especially stereo vision, the perception of depth using two cameras, is important for many areas such as robotics and autonomous driving. Stereo vision uses two cameras that are located some distance apart horizontally, but are on the same level vertically. Pixels in the images captured by the two cameras are thus displaced only in the horizontal direction, with the maximum pixel offset (traditionally called *disparity*) limited by the distance of the two cameras. The computed disparity for the pixel can then be used to derive depth information from the stereo images using triangulation (pixels closer to the cameras have larger disparities).

The algorithm at the center of this work is called Semi-Global Matching (SGM), which is one of the fastest algorithms also scoring well on accuracy in stereo vision benchmarks such as KITTI [1] and Middlebury [2]. This has led to its widespread use in many practical applications.

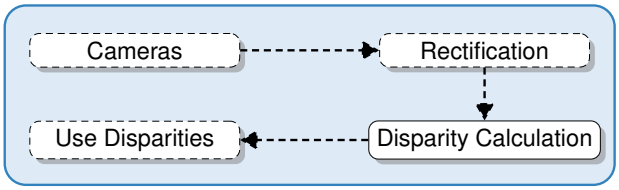

Figure 1: Typical stereo vision system. This paper focuses on the disparity calculation step.

This work proposes a novel hardware architecture to compute the SGM algorithm on FPGAs. The parametrized architecture is highly scalable, easily allowing implementations on small low-power devices (e.g., in autonomous robots) as well as for large high-performance chips (e.g., in stationary use-cases for processing multiple high-resolution video streams). As shown in Fig. 1, we focus on the disparity computation, rectification and the actual camera interfaces will not be discussed

2. Semi-Global Matching

The SGM algorithm was introduced by Hirschmüller in 2005 [3]. It provides good accuracy at manageable computational effort and is robust with regard to choices for configuration parameters [4].

We will use the function $C(\mathbf{p}, d) \in \mathbb{N}_0$ to denote the cost of matching a pixel $\mathbf{p} = (x, y)$ at coordinates (x, y) in the base image at an assumed disparity (offset) of d at coordinates $(x - d, y)$ in the matching image. As suggested by [5], the differences in the counts of pixels darker than the center pixel (a parametric rank transform) can be used to realize C (see Eq. 4). Section 4 discusses these equations in more detail. To determine the actual best match, these costs are calculated for all potential disparities $d < D_{\text{max}}$, where

D_{max} is the upper limit of the potential disparity (due to the physical mounting distance of the two cameras). At first approximation, the match with the lowest cost is assumed to indicate the true disparity $\arg\min_{d<D_{\text{max}}} C(\mathbf{p}, d)$ between base and match images for an individual pixel $\mathbf{p}$ (but see below for further constraints).

To achieve better matching accuracy, (semi) global approaches such as SGM compute these costs of potential matches not just between individual (or neighborhoods) of pixels, but along multi-pixel *paths* stretching across the entire image. The cost of matching along an entire path, described by the relative offset of path elements $\mathbf{r} = (\Delta x, \Delta y)$, for an assumed disparity d is denoted as $L_{\mathbf{r}}(\mathbf{p}, d)$. These paths are distributed evenly over the image (see Fig. 2 for examples) for a global view of the matches. Typically, at least eight evenly distributed paths are used (Fig. 2.b), but 16 are suggested for optimal coverage. The number and arrangement of paths has a direct impact not only on the matching accuracy, but also on the computational effort and, in our case, on the actual architecture of the SGM hardware accelerator.

We will aim for a compromise between performance and accuracy. As shown in [6], a reduction from eight down to the four paths $0°$ ($\mathbf{r} = (1, 0)$), $45°$ ($\mathbf{r} = (1, 1)$), $90°$ ($\mathbf{r} = (0, 1)$) and $135°$ ($\mathbf{r} = (-1, 1)$) (Fig. 2.a) results in an accuracy loss of only 1.7% (increase in count of mislabeled disparities) in the Middlebury benchmark, but allows a highly efficient hardware architecture computing the $L_{\mathbf{r}}$ for all of these paths in parallel.

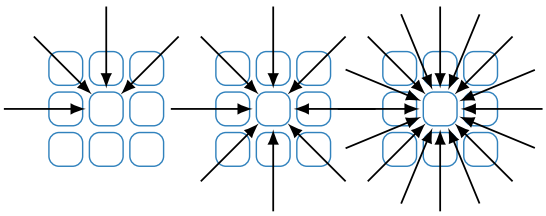

Figure 2: (a) Four, (b) Eight, and (c) 16 directions used in Semi-Global Block Matching.

The raw cost $L'_{\mathbf{r}}(\mathbf{p}, d)$ for matching the pixels $\mathbf{p}$ along a path $\mathbf{r}$ for an assumed disparity of d is calculated using the formula

$$L'_{\mathbf{r}}(\mathbf{p}, d) = C(\mathbf{p}, d) + \min \begin{cases} L'_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, d) & \textbf{.a} \\ L'_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, d-1) + P_1 & \textbf{.b} \\ L'_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, d+1) + P_1 & \textbf{.c} \\ \min_i L'_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, i) + P_2 & \textbf{.d} \end{cases} \quad (1)$$

These raw path costs are calculated for all of the selected paths, for all potential disparities d up to the limit D_{max}. For each pixel, both the local cost C as well as a semi-global component is evaluated. The latter considers four characteristics observed in real-world images, of which the minimum is added to the local cost: The first characteristic is the cost of the *prior* pixel along the path (1.**a**), the second and third components penalize small disparity changes of $|\Delta d| = 1$ by P_1 (1.**b** and **.c**), while the final term (1.**d**) penalizes larger disparity changes (so-called *discontinuities*) by P_2. P_1 is usually determined off-line experimentally by analyzing input images typical for the actual stereo vision use-case. P_2, on the other hand, is adjusted dynamically at run-time: As disparity discontinuities are often also represented as pixel *intensity* changes, computing $P_2 = \frac{P'_2}{|I_{\mathbf{p}} - I_{\mathbf{p}-\mathbf{r}}|}$ compensates for different pixel intensities $I_{\mathbf{p}}$ and $I_{\mathbf{p}-\mathbf{r}}$ along the path $\mathbf{r}$. As for P_1, P'_2 is a constant determined experimentally based on representative sample images off-line. For further discussion of the path cost calculation, please refer to the original work by Hirschmüller [3]. To determine the (semi) global matching cost, the path costs are summed up across all paths. However, for a hardware implementation, it is worthwhile to consider a slightly changed formulation.

In hardware, a key characteristic from both the performance as well as area usage perspectives is the *word width* (in bits) of arithmetic operators and data types. Since the paths will run across the *entire* image, they can be quite long (depending on the camera resolution), and summing their costs can result in large values that need wide words for computation and storage. We can counteract this by subtracting from the raw path costs for a pixel $L'_{\mathbf{r}}(\mathbf{p}, d)$ the *minimum* of the path costs for all assumed disparities d for the *prior* pixel $\mathbf{p} - \mathbf{r}$ along the path $\mathbf{r}$. The effect of encoding only the differences between prior and current pixels leads to a reduction of the magnitude of the values, which require correspondingly narrower data words for storage and computation. Thus, the hardware-optimized path cost computation becomes

$$L_{\mathbf{r}}(\mathbf{p}, d) = C(\mathbf{p}, d) + \min \begin{cases} L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, d) \\ L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, d-1) + P_1 \\ L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, d+1) + P_1 \\ \min_i L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, i) + P_2 \end{cases} - \min_j L_{\mathbf{r}}(\mathbf{p} - \mathbf{r}, j). \quad (2)$$

The paths costs $L_{\mathbf{r}}$ along all paths $\mathbf{r}$ are summed up as $S(\mathbf{p}, d) = \sum_{\mathbf{r}} L_{\mathbf{r}}(\mathbf{p}, d)$. The disparity d with the minimal matching cost $\arg\min_d S(\mathbf{p}, d)$ is considered the winning disparity for the pixel $\mathbf{p}$. These winning disparities are output by the accelerator for each pixel, as input for later computing the actual depth (Z-axis position, not discussed here).

As indicated above, in practice additional constraints need to be imposed to clean-up outliers and mark invalid disparities: The result of $\arg\min_d S(\mathbf{p}, d)$ might be multi-element set, meaning that the minimal matching cost for a pixel $\mathbf{p}$ occurs for *different* potential disparities d. With such a non-unique cost, the algorithm cannot decide on a single winning disparity, and instead registers the disparity for this pixel as "invalid".

Additionally, a so-called *left/right check* is performed, which compares the results of the algorithm when running it with swapped roles of base and match images. This check can also be implemented efficiently (avoiding recalculating all disparities for the former match image now used as base) by re-using the previously computed $S(\mathbf{p}, d)$ along an epipolar line as $\arg\min_d S((x(\mathbf{p}) + d, y(\mathbf{p})), d)$ to select the winning disparity d for the second image. The left/right check sets the disparity to "invalid" if the corresponding disparities of the original and role-swapped passes differ by more than one. This step eliminates phantom disparities resulting from occluded surfaces that are visible in one image, but hidden in the other.

Finally, the disparity map is post-processed using a basic 3×3 median filter to suppress outliers.

3. Related Work on High-Performance SGM Implementations

Several implementations of SGM exist for a wide variety of use-cases. However, they often have unsatisfactory performance or high power requirements.

Even when exploiting current vector extensions (SIMD) such as Intel AVX2 on a fast i7-4960HQ processor, a recent software implementation [7] achieved only 16 frames-per-second (fps) for VGA image pairs and $D_{\text{max}} = 128$. This processor is rated to draw 47 W under load.

The less power-hungry software solution described in [8] targets the P4080 embedded eight-core processor. Clocked at 1.2 GHz, the implementation achieves 0.5 fps on VGA images, but limits the search space to $D_{\text{max}} = 64$. The P4080 is rated to draw less than 30 W even when fully loaded.

A different trade-off was used for a heterogeneous embedded system in [9]: By combining a small Core2Duo system with an embedded OMAP3530 ARM processor and a Xilinx Spartan 6 FPGA, it achieved 14.6 fps, but has a latency of 250 ms for processing a single image of 1024×508 pixels and $D_{\text{max}} = 128$. This latency might be too high for certain real-time use-cases.

The use of GPUs leads to results similar to that of CPUs. [10] describes an implementation achieving 11.7 fps on an NVidia GTX480 GPU for VGA images with $D_{\text{max}} = 64$. However, this GPU is rated to draw 250…300 W of power when loaded.

FPGA implementations do significantly better, both with regard to performance as well as power efficiency: In [11] an architecture is proposed that is capable of processing 1024×768 pixels with $D_{\text{max}} = 96$ at 31.79 fps on a Altera EP4SGX230 device. Another approach is followed in [12], where high-level synthesis from C to hardware is used to explore different strategies targeting the Xilinx Zynq Z7020 system-on-chip on a ZedBoard. They achieve 30 fps at VGA resolution, but have a tight search limit of $D_{\text{max}} = 16$. A combination of FPGA and CPU processing is used in [13] to achieve 60 fps for images of 752×480 pixels on a Xilinx Artix 7 FPGA. While exact power numbers are not given, in many cases FPGAs draw less than 10 W for computing purposes when the high-speed serial transceivers are not used.

4. Architecture

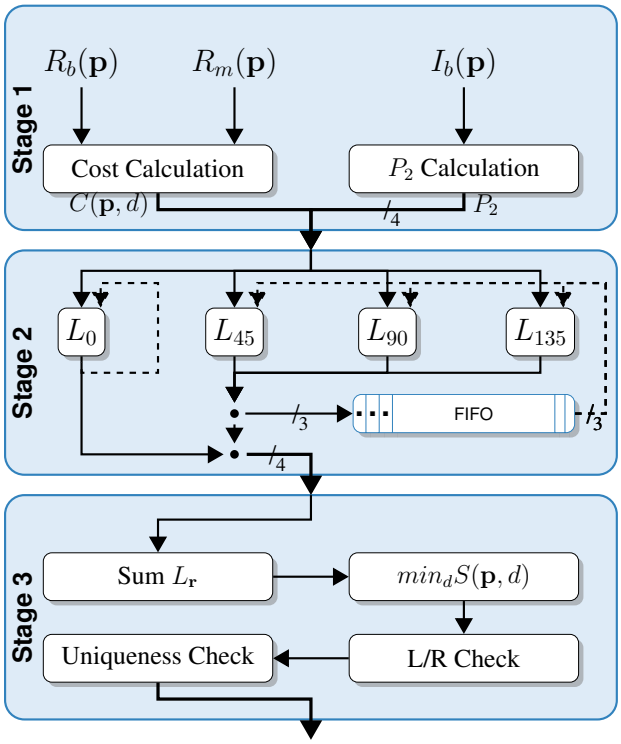

Figure 3: Base architecture for SGM. Stage 1 produces the per-pixel costs and P_2 penalty values. Stage 2 calculates the path costs, using the outputs of the previous stage as well as prior paths costs stored in a stage-internal buffer. Stage 3 computes the disparities from the path costs.

The architecture proposed here is inspired by prior work by Banz et al., presented in [6]. However, not only has our architecture been enhanced by introducing an extra level of fine-grained parallelism (e.g., parallel disparity computation and sorting), it has also been implemented using a state-of-the-art latency insensitive design style in a next-

generation hardware description language. As a result, it is significantly more scalable, easier to extend, yet also much faster than the original work (even when compensating for the differences in FPGA target technologies).

The algorithm described in Section 2 is mapped to the structure shown in Fig. 3. The modules in Stage 1 have the base and/or matching image as inputs. The cost module implements the computation of the per-pixel (neighborhood) cost (e.g., using Census or Mutual Information) from both input images. This module outputs the cost $C(\mathbf{p}, d)$ for all pixels and all disparities up to D_{max} as a stream starting from $d = 0$ up to $d = D_{max} - 1$. Secondly, Stage 1 computes the P_2 values (intensity-based penalty for discontinuities), which are generated from the base image for every pixel and the four path directions used. Thus, it outputs a stream of four P_2 values per pixel. Only Stage 1 actually reads image data, the next stage operates operates only on streams of per-pixel costs and P_2 values.

From these streams, Stage 2 computes the disparities along the four paths in parallel, using a separate module for each path $\mathbf{r}$. Since this computation requires not just the *current* per-pixel cost $C(\mathbf{p}, d)$, but also prior costs $L_\mathbf{r}(p, d)$ from *earlier* locations along the paths, the stage needs internal feedback, using buffers to delay the earlier values appropriately (storing them as three-element vectors). In summary, the Stage 2 modules require $C(\mathbf{p}, d)$, which is constant for all paths, P_2 which is constant for each pixel, $\min_i L_\mathbf{r}(\mathbf{p}-\mathbf{r}, i)$ (for the P_2-penalized L term and the magnitude reduction), which is constant for all disparities of a pixel, and finally $L_\mathbf{r}(\mathbf{p}, d-1)$, $L_\mathbf{r}(\mathbf{p}, d)$ and $L_\mathbf{r}(\mathbf{p}, d+1)$ (for the L terms penalized by P_1), all of which are unique for all disparities and all pixels and all paths. The stage produces streams of four-element vectors containing the costs for the four paths for each pixel, with the path costs for the different potential disparities $0 \leq d < D_{\max} - 1$ just being streamed-out in order.

Stage 3 operates on these streams to compute $S(\mathbf{p}, d)$ for each potential disparity and determine $\min_d S(\mathbf{p}, d)$. The latter is then checked for uniqueness and also undergoes the left/right check re-using the previously computed costs (see Section 2 for details). The output of the module is a stream of the final winning disparities, cleaned-up by a 3×3 median filter, or the "invalid" markers for pixels for which no winning disparity could be determined.

This base architecture can already perform the complete SGM computation, and, with some care in the implementation, can be fully pipelined. However, it performs only the computation of the path costs $L_\mathbf{r}$ in parallel. This can be improved both at the fined-grained as well as at the coarse-grained levels.

Fine-grained parallelization One key contribution of our work beyond [6] is the computation of per-pixel and per-path costs for n potential disparity values in parallel (see Fig. 4). Thus, Stage 1 now needs to compute n per-pixel costs, which are then streamed to Stage 2 as n-element vectors. Note that the P_2 computation is unaffected, it is still performed only once per path for each pixel. Considerable extension is required in Stage 2, which will now be replicated n-times to accept the n-element vectors from Stage 1 and process them in parallel. Two areas require special care: The intra-stage buffers are becoming larger (now having to hold $n + 2$-element vectors) with a more complex forwarding network, also the calculation of the minimal $L_\mathbf{r}$ values has to happen in parallel (achieved by a fast comparator tree). Stage 3 is similarly sped-up, replacing the sequential summing of paths for each potential disparity value with parallel adder trees. The critical path determining the performance of this solution is the parallel computation of $\min_i L_{0^\circ}(\mathbf{p} - \mathbf{r}, i)$ across all potential disparity values i, as its results are immediately needed to compute the costs for extending the path to the next pixel. Note that the other $L_\mathbf{r}$ units' outputs are buffered and required only in the next row, but L_{0° has a self-loop.

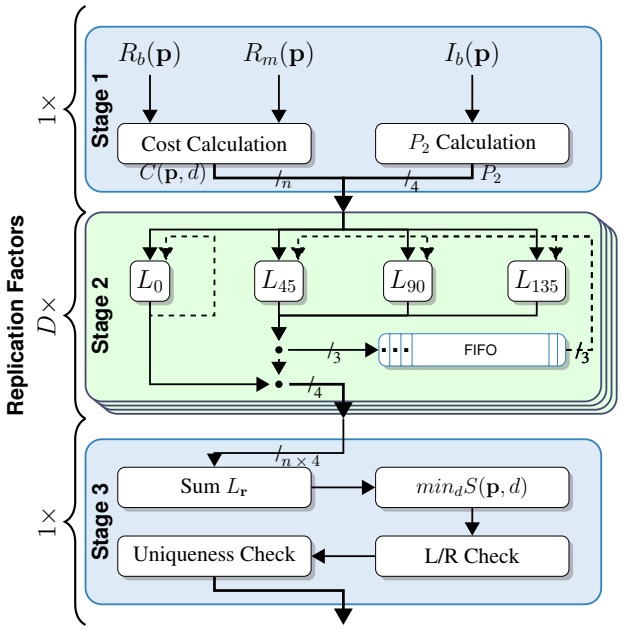

Figure 4: Extending the base architecture with fine-grained parallelism: Parallel cost computations (per-pixel, per-path) for multiple potential disparity values

Coarse-grained parallelization The base architecture can also be parallelized on a coarse-grained level (Fig. 5). This requires m multiple instances of the entire base architecture (called a *Row Processor* in this context). Each Row Processor is responsible for processing one line of the input images, leading to an image stripe m rows in height being

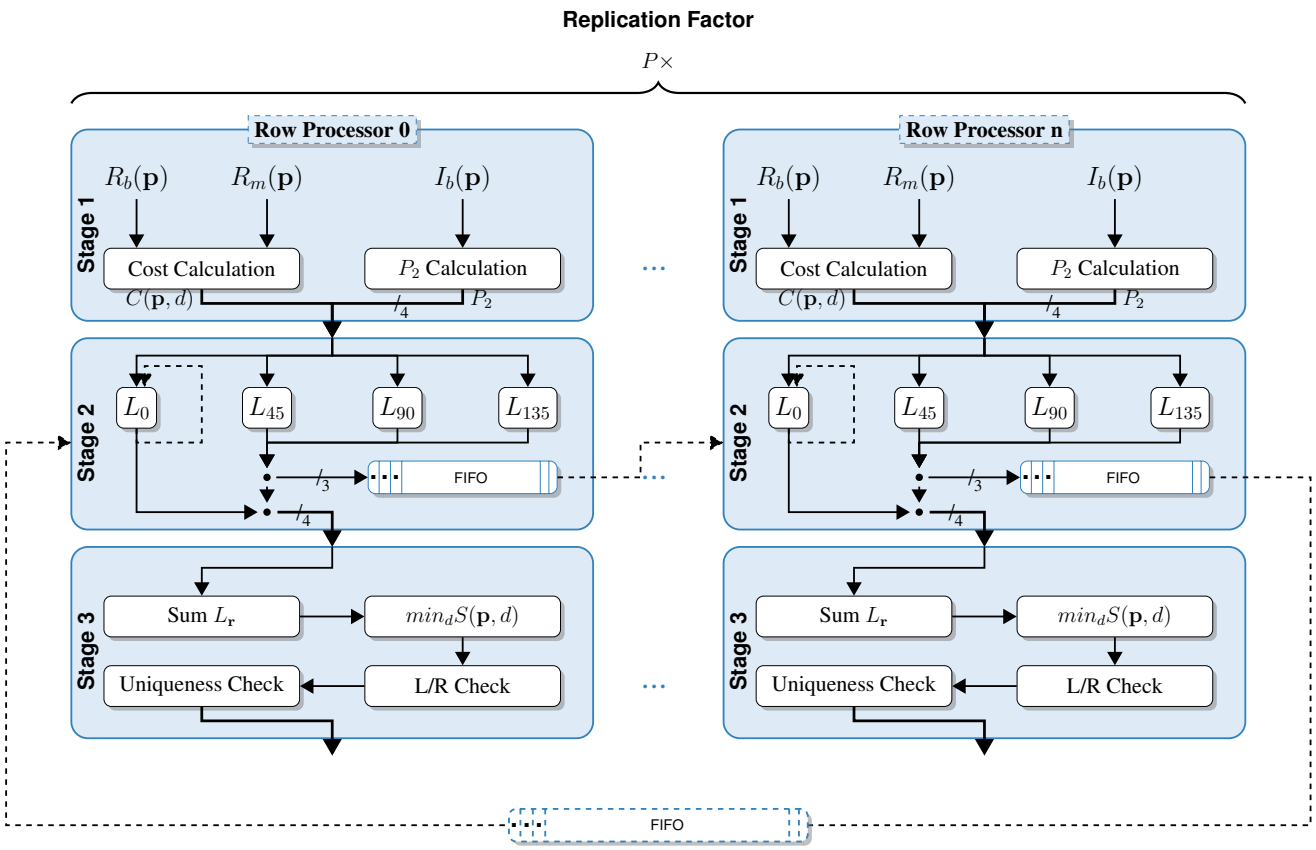

Figure 5: Coarse-grained parallelization: Processing multiple rows in parallel

processed in parallel. Buffers between the Row Processors forward the intermediate data from the $L_\mathbf{r}$ computations in Stage 2 to Stage 2 in the next Row Processor, where they will be consumed by the corresponding $L_\mathbf{r}$ units. Note that this forwarding occurs only for paths at $45°, 135°, 90°$ angles, as the path at $0°$ does not require data from the prior row (see Fig. 2.a). These forwarding buffers also provide the wrap-around to the first Row Processor, as there will be fewer Row Processors than image rows, and the first Row Processor (which will be processing the first line of the next stripe down) depends on the last Row Processor (which was processing the final line of the previous stripe up). The performance of this approach depends on how quickly the inter-row buffers can be filled with data, as Row Processors waiting for data from their predecessors will remain idle. Similarly, a Row Processor will stall if it cannot deposit its output due to the corresponding buffer to next row being full. This coarse-grained approach was already suggested in [6]. By treating it as a wavefront array (systolic array with handshake instead of lock-step data propagation), it lends itself ideally to an implementation in the new latency-insensitive hardware design style we apply.

Both parallelization techniques can be combined to mitigate their respective weaknesses (critical path length in the fine-grained approach, Row Processors stalling in the coarse-grained one). In Section 5, automatic design space exploration will be used to derive the optimal composition of the two approaches.

Selected architectural details As discussed in Section 2, we will be using the non-parametric rank transform variant of Census. The rank transform consists of counting the *number* of pixels in a neighborhood that are of lower intensity than the center pixel:

$$R(\mathbf{p}) = \left\| \mathbf{p}' \in N_s(\mathbf{p}) | I(\mathbf{p}') < I(\mathbf{p}) \right\|, \qquad (3)$$

Here, $R(\mathbf{p})$ is the rank transform of pixel $\mathbf{p}$, $N_s(\mathbf{p})$ the neighborhood around pixel $\mathbf{p}$ encompassing all pixels with a distance less or equal to $(s-1)/2$ from the center, and $I(\mathbf{p})$ the intensity of pixel $\mathbf{p}$. The neighborhood, also called a kernel, is square with a total size of s^2 pixels. Each rank transformed value is encoded in $\lceil \log_2 (s^2) \rceil$ bit.

The calculation of $R(\mathbf{p})$, which serves as input to the cost calculation stage, is done in parallel for both images. The Row Processors are fed with streams of $R_b(\mathbf{p})$ for the

base image and $R_m(\mathbf{p})$ of the matching image, as well as a stream for the P_2 penalty values. If more than one Row Processor is present in the system, round-robin is used to distribute the inputs one row at a time (leading to processing occurring in the downward moving stripe). All Row Processors require input FIFOs which can store a full row to enable operation as a wavefront array.

The actual per-pixel matching cost computation based on the rank transform R is thus

$$C(\mathbf{p}, d) = \|R(\mathbf{p}) - R((x(\mathbf{p}) - d, y(\mathbf{p})))\| \,. \qquad (4)$$

The wrap-around buffer from the last to the first Row Processor (shown in Fig. 5) is larger than the normal inter-Row Processor buffers, as it has to hold all $D_{\max}$ intermediate results for every pixel in the last row.

In a design with multiple Row Processors, the calculated disparities are buffered in FIFOs until they are retrieved by the output module. The output module then merges the outputs of the Row Processors using the same round-robin scheme as the input module. The 3×3 median filter is then applied to the merged stream remove outliers in the calculated disparities.

As shown by Hirschmüller in [3] the maximum value of any L is always less than $C_{\max} + P_2$. This limits the word widths required for data storage and arithmetic operators in the hardware implementation.

5. Evaluation

We evaluate our approach at three levels: First, we consider the accuracy, then the simulated target-independent performance of the hardware architecture in terms of clock cycles, and finally the wall-clock performance on three actual FPGA platforms, encompassing embedded system and data center use.

5.1. Accuracy

Since our core algorithm is the same as that of [6], we achieve the same disparity computation accuracy. Using the Middlebury[2], [14] benchmark, the algorithm produces on average $8.4\,\%$ disparities exceeding an error threshold of one pixel.

5.2. Platform-independent performance

Cycle-accurate simulation of the architecture (described in highly parametrized Bluespec SystemVerilog, BSV) was used to determine the run-time characteristics of sample implementations. Comparison to the actual hardware implementations (Section 5.3) shows, that these simulations are actually representative of final performance.

The core is evaluated at three image resolutions: 640×480 pixels (VGA), 1280×720 pixels (720p) and 1920×1080 pixels (1080p). VGA resolution images are evaluated at $D_{\max} = 64$, the higher resolutions have $D_{max} = 128$. For all resolutions, the number of clock cycles needed to complete a single frame are determined through simulation.

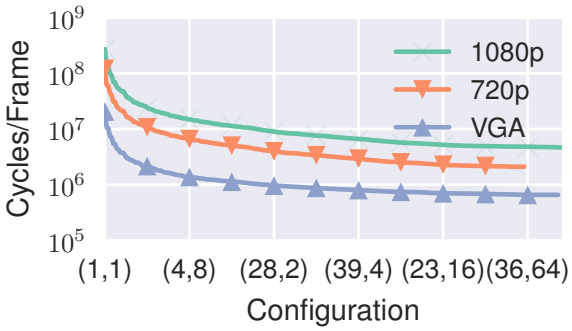

Figure 6: Cycles needed to process a single disparity map for varying degrees of parallelism.

We have examined different compositions of coarse- and fine-grained parallelism. An implementation is described by the pair (**#p**, **#d**), which indicates the use of **#p** Row Processors, each computing **#d** assumed disparities in parallel. For each of the resolutions, we have used automatic design space exploration to generate 250 implementation alternatives, shown on the X-axis in order of increasing area or performance. Due to space constraints, only a subset of the alternatives could be labeled here with (#p,#d).

As shown in Fig. 6 for VGA images, the architecture scales well with increasing the number of Row Processors, down to a lower bound of 654644 cycles, at which point a single pixel is calculated in 2.11 cycles and a single disparity requires 0.033 cycles. This design is limited by the speed the input buffers can be filled, which could be increased even more by also applying fine-grained parallelization techniques to the Stage 1 computations (see Section 6).

At an assumed clock rate of $200\,\mathrm{MHz}$ (which is realistic, see Section 5.3) the architecture would reach up to 306 fps as shown in Fig. 7. Such large and fast systems could be used to calculate the disparities over multiple cameras to produce a surround-view of a scene.

More interesting for low-power applications is the *minimal* frequency necessary to achieve 30 frames per second (a typical requirement for real-time processing). As shown in Fig. 8, the architecture is able to fulfill this requirement at a frequency as low as $30\,\mathrm{MHz}$ for VGA images.

5.3. Performance on real FPGA platforms

The true performance of our approach can be measured only when actually mapping an implementation of the architecture to a real hardware platform, of which

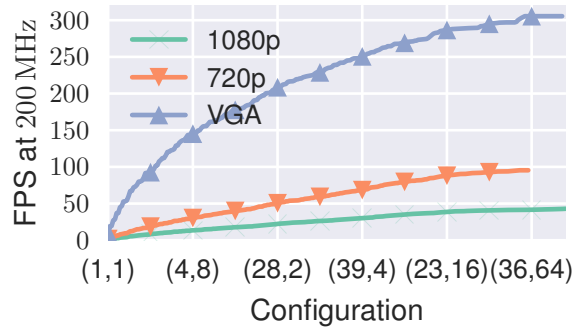

Figure 7: Frames per second achieved by the proposed architecture at a clock frequency of 200 MHz.

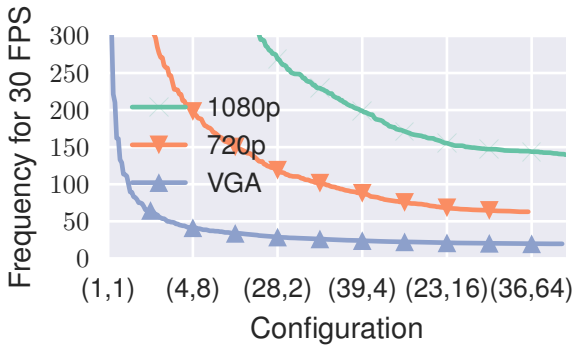

Figure 8: Frequency needed to achieve 30 frames per second on the proposed architecture for varying degrees of parallelism.

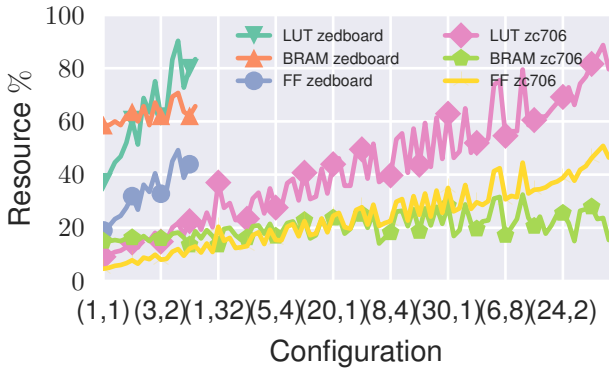

Figure 9: Hardware synthesis results for accelerators at VGA resolution for Zedboard and ZC706 platforms

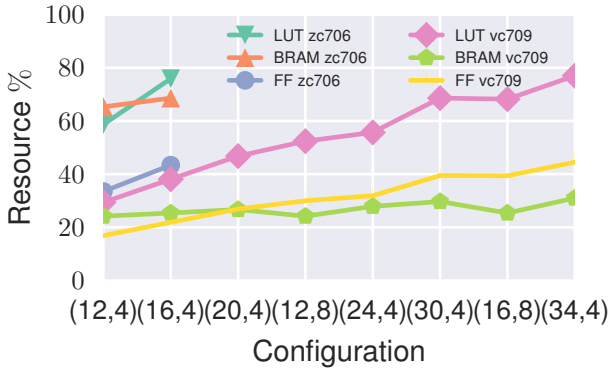

Figure 10: Hardware synthesis results for accelerators at 720p resolution for ZC706 and VC709 platforms

we consider three cases: ZedBoard (ZC7Z020T), Xilinx ZC706 (ZC7Z045) and VC709 (ZC7VX690T) development boards. The first two use Zynq-7000 reconfigurable system-on-chip devices, the last a large Virtex-7 FPGA.

On the tools side, Bluespec 2015.09 beta2 was use to compile the BSV descriptions into synthesizable RTL-style Verilog. Threadpool Composer (TPC) 2016.03 [15] was used to assemble the hardware accelerators into full system-on-chips (e.g., adding memory and control interfaces), and perform automatic design space exploration. The entire hardware system was then mapped to the FPGA devices, using Xilinx Vivado 2015.2 for logic synthesis, placement and routing.

Fig. 9 shows the resource requirements of the core at VGA resolution for different degrees of parallelism. Only the smaller two platforms are considered here, as the large VC709 is severely underutilized at VGA resolution (it easily holds even the largest sensible VGA accelerator). The smaller Zedboard is capable of running a core with #p=8 row processors and #d=1 parallel disparities as the largest core, achieving 33 VGA fps. However, design space exploration by TPC has discovered that an SGM core parametrized as (#p=5,#d=2) actually performs better (40 fps) at the 100 MHz clock frequency used on the Zedboard, and is also smaller. The ZC706 is capable of housing much larger cores: TPC exploration suggests core configurations of (21,1), achieving 140 VGA fps at 210 MHz, and (13,4) achieving 134 fps at 150 MHz.

The increase in image resolution and disparity range to 720p resolution and $D_{\max} = 128$ grows the search space by six times. However, our implementations still manage to process images in real time. on the two larger platforms (the Zedboard is too limited for the higher resolutions). The ZC706, running at 140 MHz, is capable of supporting a (16,4) configuration which yields 32 fps. The larger VC709 handles up to 45 fps at 130 MHz using a (34,4) configuration.

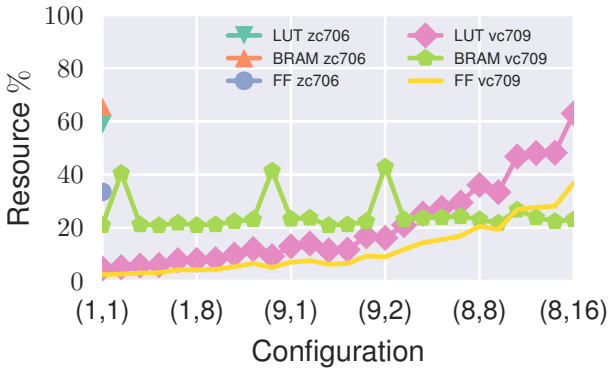

Figure 11: Hardware synthesis results for accelerators at 1080p resolution for ZC706 and VC709 platforms

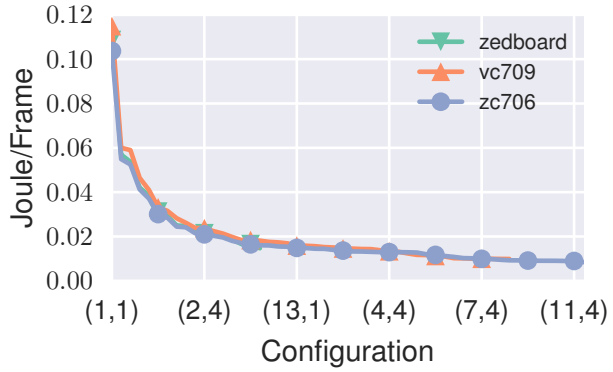

Figure 12: Energy consumed by different configurations of VGA-resolution SGM accelerators at maximum fps

The largest images we tested are in 1080p resolution with $D_{\text{max}} = 128$, requiring yet another 2.25x increase in search space over 720p. Despite this large search space, the VC709 is again capable of real-time performance at 30 frames per second in a (20,4) configuration at 130 MHz. The smaller ZC706 tops out with a (12,4) configuration running at 140 MHz, yielding 12 fps.

In addition to performance and area, the energy consumption is an important characteristic when evaluating hardware accelerators targeting low-power use-cases. As can be seen in Fig. 12, the architecture requires as low as 8.404 mJ to process a single frame. Slowing the clock frequency to reach a target of 30 fps results in similar energy requirements per frame as shown in Fig. 13. The lower clock frequencies are counteracted by the longer active time per frame.

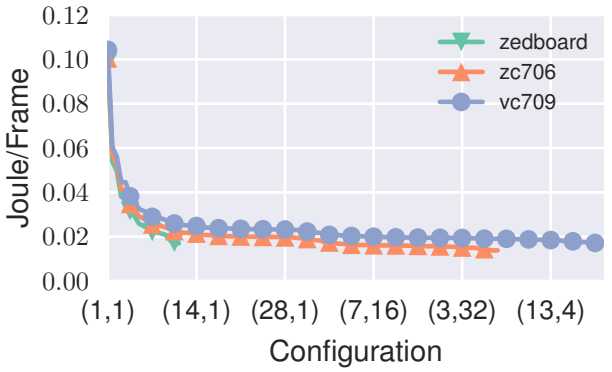

Figure 13: Energy consumed by different configurations of VGA-resolution accelerators when achieving 30 fps

6. Conclusion and Future Work

The proposed architecture to compute semi-global matching on FPGAs performs well over a wide range of scenarios. Low-power VGA configurations run at 30 fps with a clock as low as 74 MHz even on small FPGAs such as that used on the Xilinx ZedBoard. For higher performance needs, the architecture offers multiple levels of parallelization, and can be tuned by TPC in an automatic design space exploration to discover optimal configurations.

Our introduction of fine-grained parallelism into Stage 2 allows a much better adaptation of the accelerators to the needs of the individual use-case, as just increasing the number of Row Processors (as done in [6]) does not always result in the most efficient implementation.

Areas for future work include extending the use of fine-grained parallelism to Stage 1 of the architecture, namely the per-pixel cost computation (including the rank transform) and the P_2 calculation, as well as reducing the number of stalls in the Row Processor wavefront array by improvements in the buffering scheme.

Acknowledgment This work was partially funded by the EU in the FP7 research project *REPARA* (ICT-609666). The authors would also like to thank Xilinx, Inc. for supporting their work by hardware and software donations.

References

[1] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[2] D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, vol. 1, Jun. 2003, pp. 195–202. DOI: 10.1109/CVPR.2003.1211354.

[3] H. Hirschmüller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2, Jun. 2005, 807–814 vol. 2. DOI: 10.1109/CVPR.2005.56.

[4] ——, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 328–341, 2008, ISSN: 0162-8828. DOI: http://doi.ieeecomputersociety.org/10.1109/TPAMI.2007.1166.

[5] R. Zabih and J. Woodfill, "Computer vision — eccv '94: Third european conference on computer vision stockholm, sweden, may 2–6 1994 proceedings, volume ii," in, J.-O. Eklundh, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, ch. Non-parametric local transforms for computing visual correspondence, pp. 151–158, ISBN: 9783540484004. DOI: 10.1007/BFb0028345. [Online]. Available: http://dx.doi.org/10.1007/BFb0028345.

[6] 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 *Embedded Computer Systems (SAMOS), 2010 International Conference on*, Jul. 2010, pp. 93–101. DOI: 10.1109/ICSAMOS.2010.5642077.

[7] R. Spangenberg, T. Langner, S. Adfeldt, and R. Rojas, "Large scale semi-global matching on the cpu," in *Intelligent Vehicles Symposium Proceedings, 2014 IEEE*, Jun. 2014, pp. 195–201. DOI: 10.1109/IVS.2014.6856419.

[8] O. J. Arndt, D. Becker, C. Banz, and H. Blume, "Parallel implementation of real-time semi-global matching on embedded multi-core architectures," in *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII), 2013 International Conference on*, Jul. 2013, pp. 56–63. DOI: 10.1109/SAMOS.2013.6621106.

[9] K. Schmid and H. Hirschmüller, "Stereo vision and imu based real-time ego-motion and depth image computation on a handheld device," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, May 2013, pp. 4671–4678. DOI: 10.1109/ICRA.2013.6631242.

[10] M. Michael, J. Salmen, J. Stallkamp, and M. Schlipsing, "Real-time stereo vision: Optimizing semi-global matching," in *Intelligent Vehicles Symposium (IV), 2013 IEEE*, Jun. 2013, pp. 1197–1202. DOI: 10.1109/IVS.2013.6629629.

[11] W. Wang, J. Yan, N. Xu, Y. Wang, and F.-H. Hsu, "Real-time high-quality stereo vision system in fpga," in *Field-Programmable Technology (FPT), 2013 International Conference on*, Dec. 2013, pp. 358–361. DOI: 10.1109/FPT.2013.6718387.

[12] A. Qamar, C. Passerone, L. Lavagno, and F. Gregoretti, "Design space exploration of a stereo vision system using high-level synthesis," in *Mediterranean Electrotechnical Conference (MELECON), 2014 17th IEEE*, Apr. 2014, pp. 500–504. DOI: 10.1109/MELCON.2014.6820585.

[13] D. Honegger, H. Oleynikova, and M. Pollefeys, "Real-time and low latency embedded computer vision hardware based on a combination of fpga and mobile cpu," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, Sep. 2014, pp. 4930–4935. DOI: 10.1109/IROS.2014.6943263.

[14] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Int. J. Comput. Vision*, vol. 47, no. 1-3, pp. 7–42, Apr. 2002, ISSN: 0920-5691. DOI: 10.1023/A:1014573219977. [Online]. Available: http://dx.doi.org/10.1023/A:1014573219977.

[15] J. Korinth, D. d. l. Chevallerie, and A. Koch, "An open-source tool flow for the composition of reconfigurable hardware thread pool architectures," in *Field-Programmable Custom Computing Machines (FCCM), 2015 IEEE 23rd Annual International Symposium on*, May 2015, pp. 195–198. DOI: 10.1109/FCCM.2015.22.