

A Design Methodology for Hardware Acceleration of Adaptive Filter Algorithms in Image Processing*

Hritam Dutta, Frank Hannig, and Jürgen Teich
Department of Computer Science 12,
University of Erlangen-Nuremberg, Germany.

Benno Heigl and Heinz Hornegger
Siemens AG, Medical Solutions (AX),
Forchheim, Germany.

Abstract

Massively parallel processor array architectures can be used as hardware accelerators for a plenty of dataflow dominant applications. Bilateral filtering is an example of a state-of-the-art algorithm in medical imaging, which falls in the class of 2D adaptive filter algorithms. In this paper, we propose a semi-automatic mapping methodology for the generation of hardware accelerators for such a generic class of adaptive filtering applications in image processing. The final architecture deliver similar synthesis results as a hand-tuned design.

1 Introduction

Modern medical image processing has come a long way since the first hazy X-ray images produced by Dr. Roentgen in 1895. The state-of-the-art Computerized Tomography (CT), Magnetic Resonance (MR), Angiography equipments produced by medical imaging companies testify the colossal progress. In this paper, we study applications from angiography which deal with interactive catheter guidance (passing to all kinds of vessels) and minimal-invasive vessel treatments (stenosis, aneurism, embolization, etc.). The major problem still facing medical images is the poor signal to noise ratio (SNR) due to limited dosage and exposure for health reasons [4]. Limited dosage especially is an issue if fluoroscopic images are acquired. Fluoro images are not used for diagnostic purpose but for guiding a catheter, placing a device (e.g. a stent) and other interactive interventional procedures taking up to several hours. Therefore, the use of image processing algorithms to reduce the present noise along with preservation of visual structures is a major field of research. Usually, a sequence of algorithms within an *imaging pipeline* is considered to tackle the different types of noise. Important among them are 2D filtering algorithms. Basic pre-processing image filters such as mean or other linear filters fail to preserve

the visually important structures as edges and lines. Bilateral filtering as introduced by Tomasi and Manduchi is the state-of-the-art adaptive filtering algorithm which offers much better edge-preserving smoothing [13]. The second major challenge facing the medical imaging community is *real-time* medical image pre-processing with futuristic data rates of up to 120 Mpixel/sec and latency constraint of 100 ms (clinically relevant in order to get sufficient feedback for interactivity). Therefore, for real-time processing *hardware acceleration* is unavoidable. There are numerous well known software/hardware alternatives from which to choose, including digital signal processors (DSPs), custom application-specific integrated circuits (ASICs), application specific instruction processors (ASIPs) including graphic processors and field programmable gate arrays (FPGAs) and also lesser known ones such as coarse-grained processor arrays (CGAs). These alternatives offer varying degrees of performance benefits that must be weighed against other factors including cost, power consumption, and design time. The current imaging systems mostly use multi-DSP solutions for pre-processing of the images. However, there is a growing trend of combining multi-DSP based platforms with FPGAs to keep in pace with the next generation algorithms and technical advancements in equipments. Today, FPGAs offer millions of programmable gates and hundreds of hardwired multipliers for custom implementation of next generation number crunching dataflow algorithms. The most of the custom implementation of such algorithms are realized as processor array architectures which provide an optimal platform for the parallel execution. However, there is a lack of tools which map algorithm descriptions (given as *for* or *while* loops in C language) onto a hardware target (e.g. FPGA) subject to performance constraints in latency, area, or power. The current state-of-the-art HLS tools in the industry and academia are PICO Express from Synfora [10], Catapult C from Mentor Graphics [7], MMalpha [2], and PARO [8]. These tools convert algorithm descriptions to RTL along with analysis of micro-architecture implementation possibilities. Some of these tools are based on the polytope model [5] which is an intuitive methodol-

*Supported in part by the German Science Foundation (DFG) in project under contract TE 163/13-1 and Siemens AG, Medical Solutions.

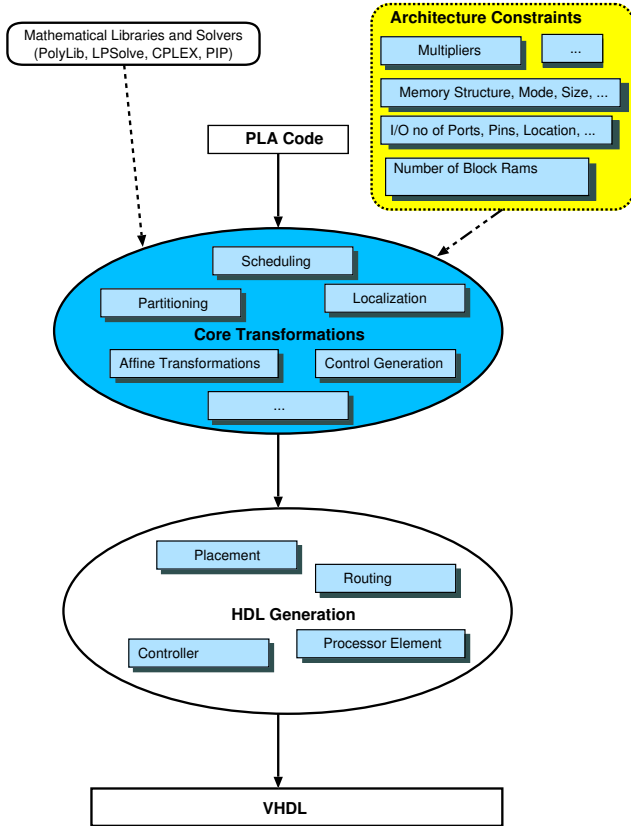


Figure 1. PARO design flow.

ogy for loop parallelization and mapping of loop nests onto massively parallel architectures.

The major contributions of this paper are an efficient and methodic realization of a computational intensive algorithm from medical imaging as hardware accelerator on an FPGA. But first in Section 2, we present the basic concepts in our mapping methodology for generation of hardware accelerators. Afterwards in Section 3, a concise description of the bilateral filter algorithm in the context of a reference imaging pipeline is given. Subsequently in Section 4, an efficient architecture realization of a bilateral filter is derived. Finally, in Section 5 and 6, the results and the conclusion of our mapping methodology and implementation are presented.

2 Background

In this section we give an overview of our mapping methodology for generating synthesizable descriptions of massively parallel processor arrays from nested loop programs. The design flow of our approach is depicted in Fig. 1. The input to the compiler is a class of algorithmic descriptions (of loop nests). This class of algorithms is formally defined as *Piecewise Linear Algorithm* (PLA)

[11]. A set of transformations is applied to the algorithmic descriptions like *localization*, *partitioning*, ..., *space-time mapping* (see Fig. 1) in order to obtain a parallel architecture description. Localization is a transformation which converts affine data dependencies into uniform data dependencies by propagation of variables from one index point to neighboring index points [11]. The transformation enables maximum data reuse within the processor array, and henceforth minimizes the amount of external I/O communication (with peripheral memory). Partitioning is a transformation which enables generation of array architectures under resource constraints. Space-time mapping defines the placement of processing elements (PEs) and scheduling of the iterations. The algorithmic description obtained after space-time mapping can be translated into a hardware description using a code generator shown by the white ellipse in Fig. 1. The hardware generation is responsible for both data path and control path generation. The successive refinement of algorithmic descriptions for hardware generation has also been done for applications from the field of neural networks, digital filtering [9], and image analysis [6], etc. In our case study, we derived not only a new efficient architecture but also incorporated some of the hand tuned architecture optimizations in the design flow.

3 Imaging Pipeline

The aim of the medical imaging community is to reduce the noise associated with the images as much as possible through the use of better algorithms. Meanwhile, due to different sources of errors, there is no single algorithm for noise removal but a sequence of algorithms usually called *imaging pipeline*. The algorithms in the imaging pipeline are responsible for the following functions.

- Conditioning: Conditioning algorithms handle defects produced due to sensor-introduced pixel errors, e.g., defect pixel interpolation (DPI), etc.
- Pre-Processing: This step involves the application of a series of algorithms for image enhancement through noise removal, e.g., filtering.

A reference imaging pipeline (see Fig. 2) constituted of conditioning and pre-processing steps is selected for our study with respect to algorithm, architecture and implementation complexity. The algorithm defect pixel interpolation (DPI) and gain-offset (G-O) correction constitute the conditioning phase of the imaging pipeline. The DPI algorithm corrects the value of defect pixels (stored in defect map) by replacing a pixel value with an average pixel value of the neighborhood (3×3 or 5×5 window). G-O correction is used to correct each pixel as they are associated with particular noise due to dark current and their own peculiar sensitivities. The G-O correction is done by multiplying the incom-

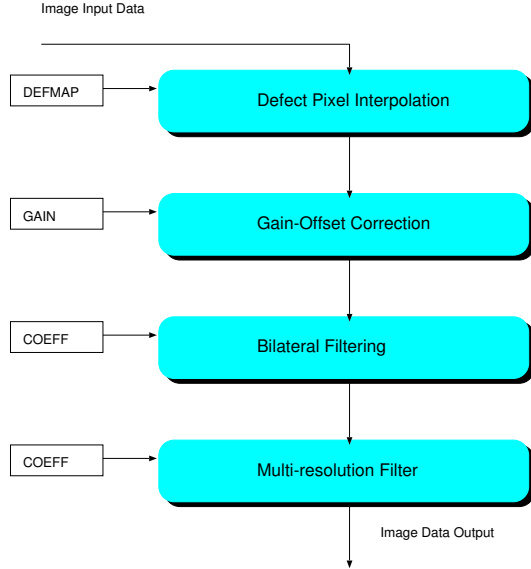


Figure 2. A reference imaging pipeline containing four algorithms. The blocks denote the particular parameters required by the corresponding algorithm.

ing image with the values in *gain map* and then offsetting it with the values in an *offset map*. The maps are obtained during the calibration step. The algorithms bilateral filter and multi-resolution filter constitute the pre-processing phase of the imaging pipeline. The bilateral filter is a 2D adaptive filter which removes the low frequency noise and preserves vessels and other structures. If in an adaptive filter the kernel size is large (say 9×9), then although low frequency noise is removed, however, structures may not be preserved. Also if the kernel size is small then the structures in the image are preserved better, but, with reduced removal of low frequency noise. This dilemma is solved by undertaking a multi-resolution representation where the adaptive filter works on *Gaussian* (the images with subsequent reduced resolutions ($g_0(1024 \times 1024)$, $g_1(512 \times 512)$, $g_2(256 \times 256)$, ...)) and *Laplacian* (difference of images at different resolutions) pyramids. The basic concept is that on going to a lower resolution, it is easier to filter low frequency noise and preserve the structures.

To use the complete parallelism available in the algorithms one needs to ascertain the required number of operations per cycle, number of memory accesses, and latency. A close study of the imaging pipeline (see Fig. 3) reveals that the algorithmic requirements of the conditioning algorithms are comparatively lower than for the pre-processing algorithms. The requirements of the conditioning algorithms can easily be met by software implementations. Whereas for a 5×5 mask, an efficient parallel implementation of a bi-

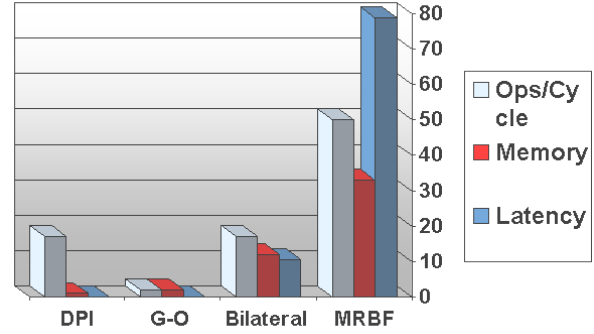


Figure 3. The requirements of the different algorithms in imaging pipeline with respect to number of required operations, memory access, and minimum latency (in terms of 100 cycles).

lateral filter requires 25 multipliers, 24 adders, and 25 parallel memory accesses for processing one pixel per clock cycle. Furthermore, the multi-resolution algorithm uses a gradient adaptive filter [4] (similar to bilateral filter) at different resolutions. In order to accelerate these pre-processing algorithms one needs parallel implementations for this class of adaptive filter algorithms. Besides, conception of dedicated architectures for a generic class of adaptive filter algorithms is subjected to constraints and algorithmic change calls for an automatic mapping methodology. Therefore, we take the bilateral filter as running example throughout the paper to exemplify our mapping methodology. In the following section we describe bilateral filtering in detail.

3.1 Bilateral Filtering

Filtering is perhaps the most fundamental operation of image processing and computer vision. The term *filtering* means that the value of the filtered image at a given location is a function of the values of the input image in a small neighborhood of the same location. E.g., Gaussian low-pass filtering computes for each pixel a weighted average of pixel values in a neighborhood where the weights decrease as a Gaussian function of distance from the center pixel. The problem with Gaussian filtering is that low spatial variation of the nearby pixels fails at edges, where we have abrupt changes in the pixel values. So, these edges are consequently blurred by low pass filtering. To prevent the blurring of edges, the local image variation needs to be measured at each pixel, and the pixel values are averaged with mask coefficients whose values depend also on the local variation. This is the main idea behind *adaptive filtering*. A bilateral filter does filtering in the range (i.e., gray values or RGB values) and spatial domain

$$y(i, j) = \sum_{m=-\frac{P-1}{2}}^{\frac{P-1}{2}} \sum_{n=-\frac{P-1}{2}}^{\frac{P-1}{2}} u(i-m, j-n) \cdot c(m, n) \cdot s(u(i, j), u(i-m, j-n)) \quad (1)$$

$$\text{where } c(m, n) = e^{-1/2(m^2+n^2)/\sigma_d^2} \text{ and } s(u(i, j), u(i-m, j-n)) = e^{-1/2(u(i, j)-u(i-m, j-n))^2/\sigma_r^2} \quad (2)$$

of an image whereas traditional filters do filtering only in the spatial domain. In the discrete domain, the bilateral filter is described by Eq. (1), where $y(i, j)$ is the processed output pixel, $u(i-m, j-n)$ describes the neighborhood of input pixels required for the computation. The convolution is carried out with closeness mask, c and similarity mask, s . The closeness mask corresponds to a Gaussian filtering mask as it enforces closeness by weighting pixel values with coefficients that fall off with the distance as an exponential function (see Eq. (3)). The coefficients of the similarity mask depend on the difference of center pixel and corresponding neighborhood pixel. E.g., the similarity mask as shown in Fig. 4 can be 0,1,1,0,1,1,0,1,1 substituting the "?" with zeros corresponding to maximum pixel value difference and 1 corresponding to 0 pixel difference as input pixel neighborhood (as 64-64=0). The following algorithm represents the functioning of a bilateral filter.

ALGORITHM: Bilateral Filter

The window operator or the mask is given by $a(m, n)$.
 FORALL $i = \frac{P-1}{2}, \dots, M - (\frac{P-1}{2})$ of the image
 FORALL $j = \frac{P-1}{2}, \dots, M - (\frac{P-1}{2})$ of the image
 FORALL $m = \frac{P-1}{2}, \dots, -\frac{P-1}{2}$ of the mask
 FORALL $n = \frac{P-1}{2}, \dots, -\frac{P-1}{2}$ of the mask
 $s = u(i, j) - u(i-m, j-n)$
 $a(m, n) = LUT(s)$ (LUT contains the c and the s masks)
 $prod(i, j) += a(m, n) \cdot u(i-m, j-n)$
 $sum_w(i, j) += a(m, n)$
 END FOR
 END FOR
 $y(i, j) = \frac{prod(i, j)}{sum_w(i, j)}$
 END FOR
 END FOR

The values in the mask are determined by the *geometric spread*, σ_d and *photometric spread*, σ_r . The mask coefficients conveniently stored in a LUT as exponential functions as they are difficult to calculate at run-time. $prod$ contains the intermediate weighted sum, whereas sum_w contains sum of mask coefficients.

3.2 Boundary Conditions

The pixels on the border strip do not have enough neighboring pixels to process the filter algorithms. For example, in case of a 3×3 filter operation, the pixels on the left, right, top and bottom strip of width 1 pixel (denoted

by x) cannot be processed (see Fig. 4). The following approaches summarize the way the boundary pixels can be handled. a) The boundary pixel values have the same values after the filtering step as before, b) *Zero padding*: Additional strips of pixels are pasted around the boundary, the pixels in the new strip have zero or maximum gray value, and c) *Symmetric extension*: An additional strip of pixels are pasted around the boundary. However, the value of pixels in the strip are mirrored values of pixels across the boundary. The same boundary handling method is also carried out for DWT based encoders [1].

4 Design Methodology

In this section, the PARO methodology is applied for synthesis of a massively parallel architecture from algorithmic description. Starting point of the methodology is the PLA description which explicitly represents the full data parallelism in the algorithm. The variables in the PLA description of the nested loop algorithm with different number of indices needs to be *embedded* into an index space with common dimensions. This allows the scheduling and placement of the variables in the polytope model. Therefore, variables u , a and y_{out} , i.e., inputs, weights and output image are embedded in the common 4D index space. The following PLA is equivalent to the algorithm in Section 3.1.

ALGORITHM: Bilateral Filter (PLA)

$$\begin{aligned} u[i, j, m, n] &= u[i-m+T, j-n+T, 0, 0] \\ a[i, j, m, n] &= F(u[i, j, m, n] - u[i, j, T, T]) \\ z[i, j, m, n] &= a[i, j, m, n] \cdot u[i, j, m, n] \\ y_1[i, j, m, n] &= z[i, j, m, n] \text{ if } m = 0 \wedge n = 0 \\ y_1[i, j, m, n] &= y_1[i, j, m, n-1] + z[i, j, m, n] \text{ if } n > 0 \\ y[i, j, m, n] &= y_1[i, j, m-1, n] + y_1[i, j, m, n] \text{ if } m > 0 \wedge n = P-1 \\ m_1[i, j, m, n] &= a[i, j, m, n] \text{ if } m = 0 \wedge n = 0 \\ m_1[i, j, m, n] &= m_1[i, j, m-1, n] + a[i, j, m, n] \text{ if } m > 0 \\ m[i, j, m, n] &= m[i, j, m, n-1] + m_1[i, j, m, n] \text{ if } n > 0 \wedge m = P-1 \\ y_{out}[i, j, m, n] &= \frac{y[i, j, m, n]}{mask[i, j, m, n]} \text{ if } m = P-1 \wedge n = P-1 \end{aligned}$$

The index space is given by

$$\mathcal{I} = \{(i, j, m, n)^T \mid 0 \leq i, j \leq M-1 \wedge 0 \leq m, n \leq P-1\}.$$

The weights $a[i, j, m, n]$ is an exponential function of the difference of the pixel value ($u[i, j, m, n]$) with the center pixel ($u[i, j, T, T]$). The possible values of the exponential function are stored in a look-up table defined by F . The variables $z[i, j, m, n]$ builds the product of the mask coefficients ($a[i, j, m, n]$) and the corresponding pixel

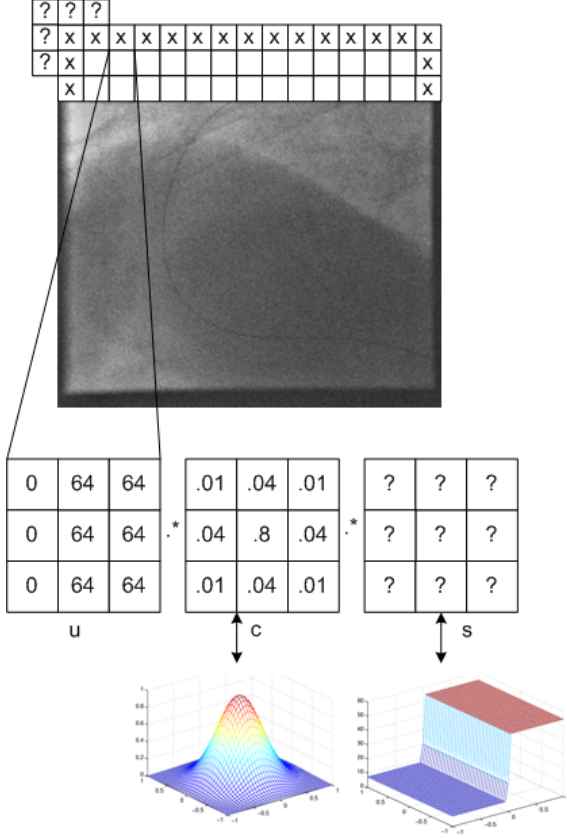


Figure 4. x denotes all the border pixels which cannot be processed without additional boundary conditions defining the "?". The bilateral filter computes the mask product of the image neighborhood with a closeness mask, c and a similarity mask, s .

$(u[i, j, m, n])$. The variables $y_1[i, j, m, n]$ and $y[i, j, m, n]$ store the intermediate and final weighted pixel sum. Similarly, $m_1[i, j, m, n]$ and $m[i, j, m, n]$ store the intermediate and final sum of the mask coefficients. Finally, $y_{out}[i, j, m, n]$ store the output pixel value.

Localization enables maximum data reuse within the processor array and henceforth minimizes the amount of external I/O communication (with a line buffer as will be seen later). After the transformation *Localization*, the equivalent equations for input pixel, u and center pixel, u_c , are obtained as follows:

$$\begin{aligned}
 u[i, j, m, n] &= u[i-1, j, m-1, n] && \text{if } i > 0 \wedge m > 0 \\
 u[i, j, m, n] &= u[i, j-1, m, n-1] && \text{if } j > 0 \wedge n > 0 \\
 u[i, j, m, n] &= U_{i-m+T, j-n+T} && \text{if } m = 0 \\
 u_c[i, j, m, n] &= u_c[i, j, m-1, n] && \text{if } m > 0 \\
 u_c[i, j, m, n] &= u_c[i, j, m, n-1] && \text{if } m = 0 \wedge n > 0 \\
 u_c[i, j, m, n] &= U_{i-P/2, j-P/2} && \text{if } m = 0 \wedge n > 0
 \end{aligned}$$

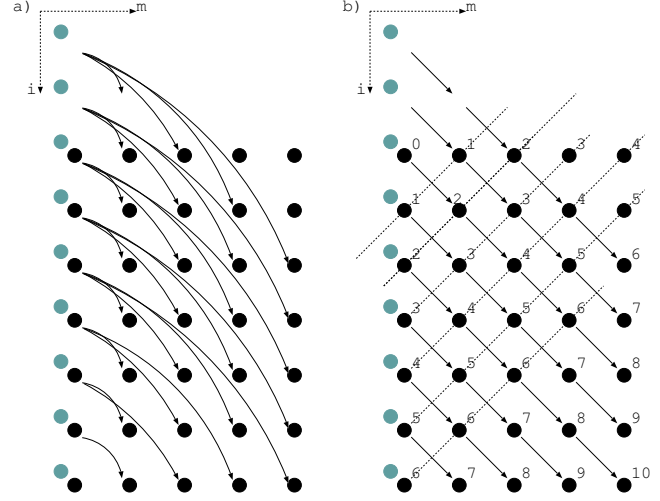


Figure 5. a) Projected dependence graph of bilateral filter considering only the dependence of input pixels, u b) Localized dependence graph. The dotted lines show the isotemporal hyperplanes defining the schedule.

Although our dependence graph is a 4D graph, the idea of localization of u can be illustrated as in Fig. 5(a) and (b) by neglecting index variables j and n . By means of localization the broadcast of input data as shown in Fig. 5(a) is converted to propagation of data by introduction of equivalent uniform data dependencies (see Fig. 5(b)). The computations of the above PLA which has only uniform dependencies may be represented by a dependence graph (DG). However, the four dimensional nature of the iteration makes it difficult to visualize the DG. However, the DGs can be represented in a reduced form, the so called *reduced dependence graphs* (RDGs). All the dependence vectors are associated with corresponding edges of the graph. The RDG of the algorithm can be seen in Fig. 6.

Space-time mapping is a linear transformation that assigns index vectors (i, j, m, n) to processors (p_1, p_2) and time t . In other words, it determines on which processor and at what time an iteration is to be executed. The reason for using linear allocation and scheduling functions is local and regular data flow required between the processing elements (PEs). We chose the following transformation to describe the processor array structure and mapping:

$$\begin{pmatrix} p_1 \\ p_2 \\ t \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & L & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} i \\ j \\ m \\ n \end{pmatrix}$$

L is the number of pixels in a single image line. The interpretation is that $p_1 = m$ and $p_2 = n$ means all iterations (i, j) corresponding to same coefficient position (same m

and n) takes place on the same PE. Furthermore the set of operations defined at index points $\lambda \cdot (i, j, m, n)^T = const$, where $\lambda = (1 \ L \ 1 \ 1)$ denotes all iteration scheduled at the same time step. In our approach, the scheduling vector λ is obtained by the formulation of a latency minimization problem as a mixed integer linear program (MILP) similar as in [12, 3]. Assuming constant j and n , the dotted line in Fig. 5(b) denotes the *isotemporal hyperplanes* (i.e. all the points on the dotted line are executed at same time step). The following description is obtained after space-time mapping:

ALGORITHM: PLA after space-time mapping

$$\begin{aligned}
 u[p_1, p_2, t] &= u[p_1 - 1, p_2, t - 2] \\
 u[p_1, p_2, t] &= u[p_1, p_2 - 1, t - (L + 1)] \\
 u_c[p_1, p_2, t] &= u_c[p_1 - 1, p_2, t - 1] \text{ if } p_1 > 0 \\
 u_c[p_1, p_2, t] &= u_c[p_1, p_2 - 1, t - 1] \text{ if } p_2 > 0 \\
 a[p_1, p_2, t] &= F(u[p_1, p_2, t] - u_c[p_1, p_2, t]) \\
 z[p_1, p_2, t] &= a[p_1, p_2, t] \cdot u[p_1, p_2, t] \\
 y_1[p_1, p_2, t] &= z[p_1, p_2, t] \text{ if } p_1 = 0 \\
 y_1[p_1, p_2, t] &= y_1[p_1 - 1, p_2, t - 1] + z[p_1, p_2, t] \text{ if } p_1 > 0 \wedge p_2 = 0 \\
 y[p_1, p_2, t] &= y[p_1, p_2 - 1, t - 1] + y_1[p_1, p_2, t] \text{ if } p_1 = 0 \wedge p_2 > 0 \\
 m_1[p_1, p_2, t] &= a[p_1, p_2, t] \text{ if } p_1 = 0 \\
 m_1[p_1, p_2, t] &= m_1[p_1 - 1, p_2, t - 1] + a[p_1, p_2, t] \text{ if } p_1 > 0 \wedge p_2 = 0 \\
 m[p_1, p_2, t] &= m[p_1, p_2 - 1, t - 1] + m_1[p_1, p_2, t] \text{ if } p_1 = 0 \wedge p_2 > 0
 \end{aligned}$$

The above program description can be parsed to obtain the final architecture. The delays can be modeled either as shift registers ($t - 2$ represents a shift register of size 2). Furthermore, the difference in processor index in the same equation indicates that the data is obtained from the neighboring processor. In the following section, we discuss the obtained architecture.

4.1 Architecture

The core of the obtained design is a 2D processor array (PA) of $P \times P$ PEs. The value $P \times P$ corresponds to the size of the filter mask. The PA architecture is shown for $P = 3$ in Fig. 7. Corresponding to the RDG, each PE contains one fixed point, full precision multiply and accumulate unit (MAC). The subtract unit calculates the difference of the center pixel and the neighbor pixel in order to calculate the mask coefficients. The mask coefficient is an exponential function of the difference. Instead of calculating the exponential function which is expensive in hardware, a LUT is used to store the pre-calculated function values corresponding to the difference. For an 8-bit gray scale image the LUT contains maximum of 256 values. The size of the LUT depends on the *photometric spread* parameter, σ_d . Therefore, the function values close to zero need not be stored. The ADD units are used to calculate the intermediate pixel sum and the mask sum. The LUT values are configured in the Block RAM within the PE. The border PEs get their input samples from the line buffers. It may be

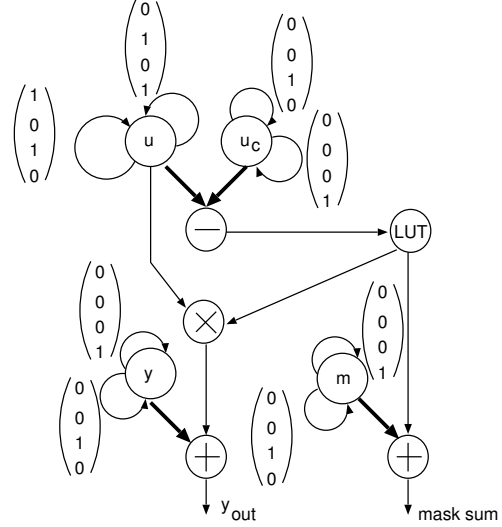


Figure 6. Reduced Dependence Graph for bi-lateral filter. The vectors denote the data dependencies between iterations.

noted that the size of the line buffers is $L + 1$ which can be inferred from the algorithm description obtained after the space-time mapping. Similarly the number of line buffers ($=P - 1$) can be inferred from the conditionals in the program description. The equations are valid in the domain corresponding the if conditionals. On intersection of the processor-based if conditional one finds the set of PEs executing the same operation. Hence, all the PEs do not have the same set of operations (e.g. border PEs have an extra addition unit as shown in Fig. 7). The pipelined divider is used to divide the pixel sum with the mask sum to obtain the final output pixel. The pipelined divider has a latency depending on the size of divisor and dividend and delivers the output pixel. The latency (i.e. the time from reading the first input pixel to obtain the output pixel) of the shown processor array architecture with a 3×3 filter mask for an image of 512×512 is 1060 cycles. The factors deciding the total latency are latency of the buffer memory, latency of the processor array (e.g., the time the output pixel takes to wander from PE(0,0) to PE(2,2)), and the latency of the pipelined divider. Control generation is required in order not to calculate the border pixels.

5 Results

The above derived PA architecture was implemented on a Xilinx Virtex XC2V6000 based board. Table 1 gives the resource usage of the processor array architecture without the input and output system. Of the 144 Block RAMs, only 9 and 2 BRAMs were needed by the processor array and the pixel pipeline respectively. However, the whole design

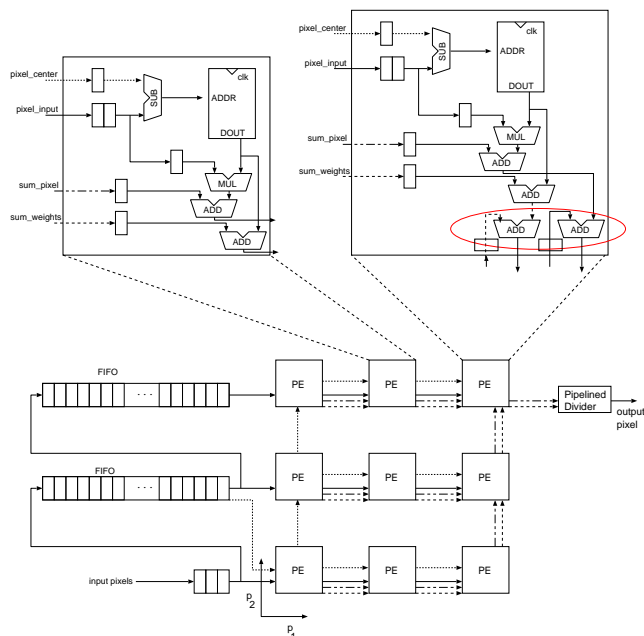


Figure 7. Highly parallel processor array architecture for bilateral filtering. The blocks show the internal architecture of the processing elements. The border PEs, i.e., on $p_2 = 2$ have two additional adders (shown within the ellipse).

included buffering of image data therefore leading to high consumption of buffer memory. This recommends usage of on-board SRAMs as intermediate video memory. Secondly, for this FPGA (which is considered relatively large by any standards), the resources would not be enough to handle bilateral filtering for a mask bigger than 11×11 . In case of bigger masks, the architecture design must be further partitioned. This aspect was not studied as filter masks not bigger than 7×7 were considered in the imaging pipeline. A maximal clock speed of 87.65 Mhz was obtained during the synthesis of the processor array design.

6 Conclusion

In this paper, we introduced a methodological derivation of a massively parallel architecture for a bilateral filter algorithm. The fine tuning of memory access is formalized by the PARO methodology. Our future work entails inclusion of other high level transformations. For example, although the processor array can attain maximal clock rate of 100Mhz, the input data frequency for current medical imaging applications is only 20-30 Mhz. In Fig. 7, for the input rate of 25Mhz, a processor array clocked at 75Mhz can implement the algorithm with only 3 PEs and by switching the input data with the help of a multiplexer. This optimization

Table 1. Resource Utilization of the PA.

Slices	1447 out of 33792	4%
BRAMs	11 out of 144	6%
MULT18X18s	9 out of 144	6%

can be done by applying a different *partitioning* transformation. All the proposed transformations and VHDL code generation concepts are currently implemented in our design system [8].

References

- [1] C. Chakrabarti. A DWT-Based Encoder Architecture for Symmetrically Extended Images. In *Proc. of the International Symposium on Circuits and Systems*, 1999.
- [2] S. Derrien and T. Risset. Interfacing compiled FPGA programs: the MMAAlpha approach. In *PDPTA*, 2000.
- [3] F. Hannig and J. Teich. Resource Constrained and Speculative Scheduling of an Algorithm Class with Run-Time Dependent Conditionals. In *Proceedings IEEE 15th International Conference on Application-specific Systems, Architectures and Processors (ASAP 2004)*, Galveston, TX, U.S.A., Sept. 2004.
- [4] D. Kunz, K. Eck, H. Fillbrandt, and T. Aach. A Nonlinear Multi-resolution Gradient-Adaptive Filter for Medical Images. *SPIE Medical Imaging, SPIE*, 5032:732–742, Feb. 2003.
- [5] C. Lengauer. Loop Parallelization in the Polytope Model. In E. Best, editor, *CONCUR'93, Lecture Notes in Computer Science 715*, pages 398–416. Springer-Verlag, 1993.
- [6] E. Mémin and T. Risset. VLSI Design Methodology for Edge-preserving Image Reconstruction. *Real Time Imaging*, 7(1):109–126, February 2001.
- [7] Mentor Graphics. www.mentor.com.
- [8] PARO Design System Project. www12.cs.fau.de/research/paro.
- [9] H. Ruckdeschel, H. Dutta, F. Hannig, and J. Teich. Automatic FIR Filter Generation for FPGAs. In T. Hämmäläinen, A. Pimentel, J. Takala, and S. Vassiliadis, editors, *Embedded Computer Systems: Architectures, Modeling, and Simulation, 5th International Workshop, SAMOS 2005, Proceedings*, volume 3553 of *Lecture Notes in Computer Science (LNCS)*, pages 51–61, Island of Samos, Greece, July 2005. Springer.
- [10] Synfora, Inc. www.synfora.com.
- [11] J. Teich. *A Compiler for Application-Specific Processor Arrays*. PhD thesis, Institut für Mikroelektronik, Universität des Saarlandes, Saarbrücken, Germany, September 1993.
- [12] J. Teich, L. Thiele, and L. Zhang. Scheduling of Partitioned Regular Algorithms on Processor Arrays with Constrained Resources. *Journal of VLSI Signal Processing*, 17(1):5–20, Sept. 1997.
- [13] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision*, page 839, Washington, DC, USA, 1998. IEEE Computer Society.