Real Time Image Processing based on Reconfigurable Hardware Acceleration

Steffen Klupsch, Markus Ernst
research center caesar
center of advanced european studies and research
Friedensplatz 16, 53111 Bonn

Sorin A. Huss
Integrierte Schaltungen und Systeme
Fachbereich Informatik
Technische Universität Darmstadt
Alexanderstr. 10, 64283 Darmstadt

M. Rumpf, R. Strzodka
Institut für Mathematik
Gerhard-Mercator-Universitaet Duisburg
Lotharstr. 65, 47048 Duisburg

Abstract

This paper is concerned with a substantial speed up of image processing methods on 2D and 3D images making use of modern FPGA (Field Programmable Gate Array) technology. The applications of this class of methods ranges from 2D and 3D image denoising and restoration, segmentation, morphological shape recovery and matching to vector field visualization and simulation. The described demonstrator is based on level set methods, but the proposed workflow allows to exchange the underlying mathematical methods easily.

The FPGA based hardware implementation profits especially from the high parallelism in the algorithm and the moderate number precision required to preserve the qualitative effects of the mathematical models. Furthermore, different variants can be supported on the same hardware by uploading a new programming onto the FPGA. This will enable the use of these flexible image processing methods in applications where real time performance is indispensable.

1 Mathematical Background

The quality and the size of image data (especially 3D medical data) is constantly increasing. Fast and optimally interactive postprocessing of these images is a major concern. E. g., segmentation on them, morphing of different images, sequence analysis and measurement are difficult tasks to be performed. Especially for segmentation and for morphing purposes *level set methods* [3] play an important role. They can be regarded as continuous region growing algorithms dependent on a variety of boundary criteria and have proved to be effective and robust (Fig. 1).

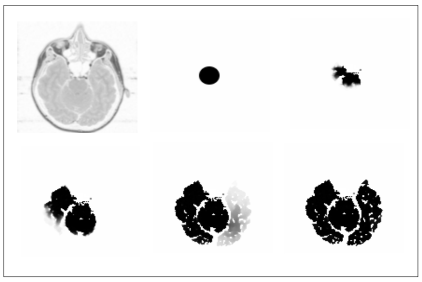


Figure 1: Segmentation of a human brain.

The motion of an interface $\Gamma(t)$ in time t starting from an initial position $\Gamma(0) = \Gamma_0$ can be formulated based on an implicit description of the interface $\Gamma(t) = \{x \,|\phi(t,x) = 0\,\}$.

If $f(t, x)$ is the speed of the interface at time t and position x we obtain the partial differential equation

$$\partial_t \phi(t, x) + f(t, x)\|\nabla\phi\| = 0$$

In the case of image segmentation f is a force which pushes the interface towards the boundary of a segment region in an image. Usually f equals one in homogeniety regions of the image, where as f tends to zero close to the segment boundary.

The discretization of the level set model is performed with finite differences on an uniform quadrilateral or octahedral grid. The grid elements are enumerated with a d-dimensional ($d = 2, 3$) multi-index α and denote the element centers by the d-dimensional vector x^α. The element vector containing values of a numerical approximation Φ of the continuous function ϕ associated with the elements centers will be marked by a bar: $\bar{\Phi} = (\bar{\Phi}_\alpha)_\alpha = (\phi(x^\alpha))_\alpha$.

Our numerical solution of the level set equation is based on an explicit Euler scheme in time and an upwinding flux approximation in space. Depending on the order of the time scheme update formulas have to be evaluated to obtain the next timestep solution.

Picking up the upwinding methodology [3] used in finite volume computations we approximate $H(u) := f\|u\|$ by the numerical flux $g : \mathbb{R}^{2\times2} \to \mathbb{R}$ and thus obtain the upwind level set scheme as

$$\Phi^{k+1}_\alpha = \Phi^k_\alpha - \tau_k \cdot g(D^-_\alpha \Phi^k, D^+_\alpha \Phi^k) \tag{1}$$

$$D^+_\alpha \Phi^k := \left(\frac{\Phi^k_{\alpha+e_i} - \Phi^k_\alpha}{\left|x^{\alpha+e_i}_i - x^\alpha_i\right|} \right)_{i=1\ldots d} \qquad D^-_\alpha \Phi^k := \left(\frac{\Phi^k_\alpha - \Phi^k_{\alpha-e_i}}{\left|x^\alpha_i - x^{\alpha-e_i}_i\right|} \right)_{i=1\ldots d}$$

τ_k is the current timestep width. The discrete solution corresponding to the vector $\bar{\Phi}^k$ is expected to approximate $\phi(\sum_{j=0}^{k-1} \tau_j, .)$. To satisfy the natural boundary condition we set $D^\pm_\alpha \bar{\Phi}^k := 0$ for all α enumerating border elements.

For g we can also choose between different orders of approximation. The simple Enquist-Osher flux [1] is suitable for convex H:

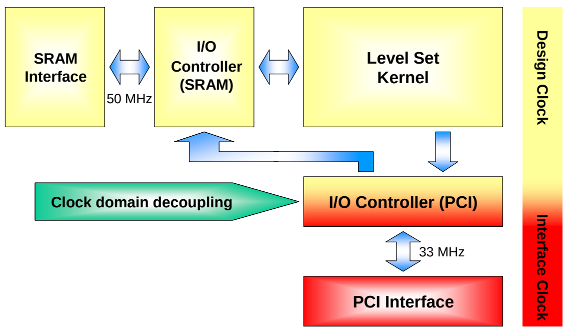
$$\begin{aligned} g(U, V) &:= H(0) + \int_0^U H'(s)^\oplus ds + \int_0^V H'(s)^\ominus ds \\ A^\oplus &:= \max(A, 0), \qquad A^\ominus := \min(A, 0) \\ \rightsquigarrow g(\bar{U}_\alpha, \bar{V}_\alpha) &= \bar{F}^\oplus_\alpha \sqrt{\|\bar{U}^\oplus_\alpha\|^2 + \|\bar{V}^\ominus_\alpha\|^2} + \bar{F}^\ominus_\alpha \sqrt{\|\bar{U}^\ominus_\alpha\|^2 + \|\bar{V}^\oplus_\alpha\|^2} \end{aligned}$$

For the implementation 2D data on an equidistant grid is considered, with the grid specific diameter h, and the first order time discretization with the Enquist-Osher flux is applied to it.

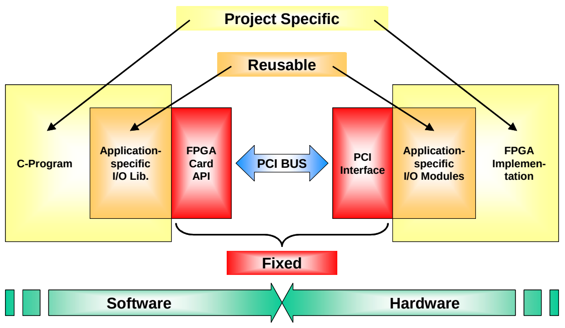
2 Design Methodology

A characteristic of image processing methods is the above described multiple iterative processing of data sets. Due to the possible restriction on the number precision it is possible to work on integer data sets with a restricted range of values, i. e., an application specific word length. Furthermore, it is possible to incorporate parallel execution of the update formulas.

General purpose CPUs of todays computers can be used to implement such algorithms, but the performance of pure software implementations is not sufficient. All operations have to be mapped



(a)



(b)

Figure 2: Conceptional hardware and software design of the hardware accelerated system.

to a processor with a fixed word length (e.g., Intel Pentium, 32 bit) - the choice of the word length is not based on the requirements for the concrete processing task. The range of available operations is limited to common basic operations. Complex operations have to be realized by a large sequence of simple operations. Finally the CPU is burdened with additional tasks, such as operating system requests, user interaction, etc., which is a major drawback in the context of real-time processing. As a result it is difficult to meet hard real time requirements.

The proposed approach to solve the shortcomings of genuine software implementations is to use application specific hardware acceleration. A standard PCI card product is used as accelerator board [4]. This card is equipped with a reconfigurable logic device (FPGA) from Xilinx Inc. [5], in which the functionality can be implemented. The used card is equipped with a XC4085XLA-FPGA with a complexity of max. 180.000 system gates. Furthermore, the card comes with a programmable clock generator, 2MB static RAM and external interfaces. The integration into a target system is accomplished via the PCI interface (Fig. 2(a)).

A generic concept for system partitioning of hardware accelerators based on PCI cards is shown in Fig. 2(b). The system is centered around the *fixed* PCI interface with reusable, but application-specific, low level I/O access functions. The system functionality is divided into a software part (running on the CPU) and a hardware part (implemented within the FPGA). The high performance PCI interface allows a fast and efficient data exchange between the FPGA card and the software running on the host computer. Accuracy and word length of operations can be optimized and arbitrary complex operations can be built to allow *single step* calculations. The FPGA approach promotes the usage of pipelined and parallelized algorithms through a concurrent execution paradigm.

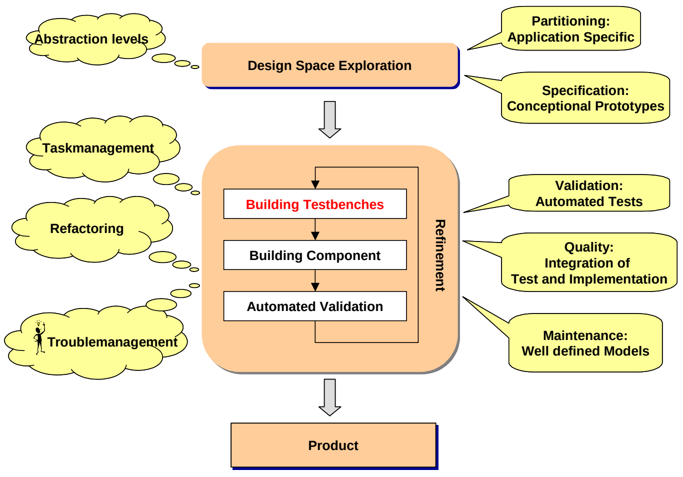

Abstraction levels
Design Space Exploration
Partitioning: Application Specific
Specification: Conceptional Prototypes
Taskmanagement
Refactoring
Troublemanagement
Building Testbenches
Building Component
Automated Validation
Refinement
Validation: Automated Tests
Quality: Integration of Test and Implementation
Maintenance: Well defined Models
Product

Figure 3: Quality optimized workflow.

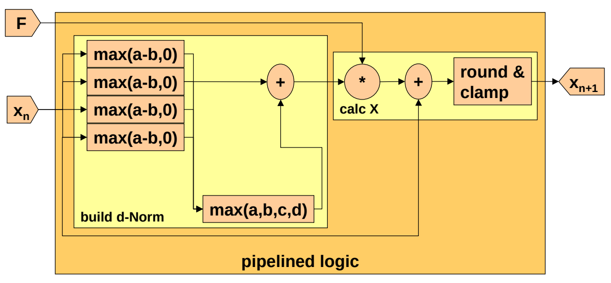
Nowadays, the design entry for most digital circuits is based on behavioral models in hardware description languages (HDL), such as VHDL or Verilog. These descriptions are processed by synthesis tools to derive a netlist of basic logic elements, which can be fed into place and route tools. With the development of especially large FPGAs and the high time-to-market pressure for consumer products, the need for fast Algorithm-to-Chip workflows has proved to be very lucrative. As a matter of fact the time needed during development can be more important than the optimization of the synthesized circuit with regard to chip size or performance. As a consequence many attempts are made to incorporate higher abstraction levels into synthesis tools - which is exploited for our design, but not without shifting the optimization goal (Fig. 3): By using parametrised high level descriptions, we gain the possibility to do detailed design space explorations. The design complexity is reduced by using small abstract behavioral descriptions. By extensive hardware/software co-evaluation we add validation of the synthesis results without leaving the abstraction level [2]. The time saved is then used for selective optimization of critical design subcircuits.

3 Design Optimizations

Image processing algorithms as described in Sec. 1 consist of a complex sequence of primitive operations, which have to be performed on each nodal value. By combining the complete sequence of primitive operations into a compound operation it is possible to reduce the loss of performance caused by the synchronous design approach (Fig. 4).

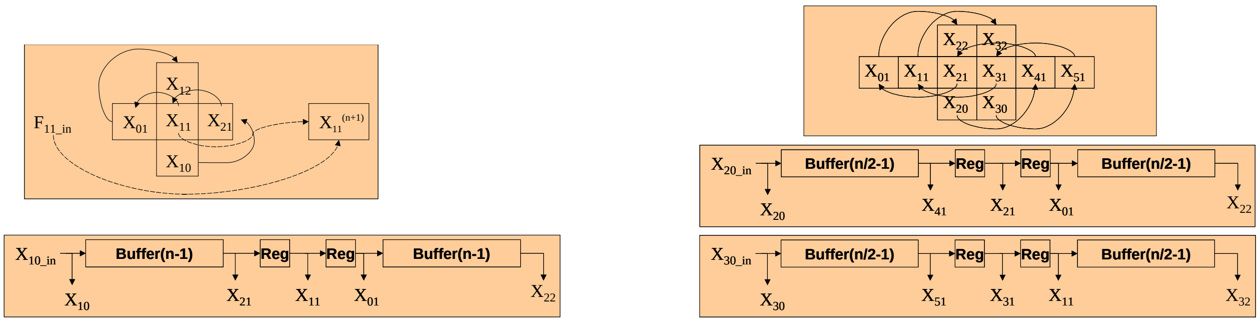
This design approach, which is common to CPU designs as well as for FPGA designs, is based on the assumption that all arithmetic operations will converge well in advance to the clock tick, which will cause the results to be postprocessed. Therefore, the maximum clock speed of such systems is defined by the slowest combinatorial path. In a CPU this leads to ’waiting time’ for many operations. Furthermore, there is no need for command fetching in FPGA designs, which solves another problem of CPU-based algorithms. Additionally, it is possible to do arbitrary parallel data processing in a FPGA, so that several nodal values can be updated simultaneously.

The input data rate for CPU-based and FPGA-based applications is determined by the bandwidth of the available memory interface. A 256^2 image results in 64k words, resulting in 768 kBit data at 12 bit resolution. A CPU with a 16 bit wide data access would need 128 kByte to store the original image data, without taking the memory for intermediate results into account. Even though many FPGA architectures can store data in special RAM blocks within the FPGA chip, we need to store most of the image data in external RAM. The access speed of the RAM and the bit width are limiting



- **High accuracy through adaptive result ranges**
- **Arbitrary pipeline stages possible**
- **Exchangeable norm calculus**

Figure 4: Node update implementation.



(a) (b)

Figure 5: Cache structures for a single calculation unit (a) and for doubled calculation units (b).

the input data rate. To reduce the impact of this bottleneck, it is a major design task to buffer reusable data in appropriate structures within the FPGA in order to minimize the amount of RAM accesses (Fig. 5(b)). This involves algorithm analysis and sophisticated scheduling, first order optimizations, like reuse of data by calculating sequels of adjacent nodal values, are obvious and reduce the amount of data needed for the level set application from 6 to 4. By doing further optimizations it is possible to reduce the amount of transferred data per node update to the optimum of 2 data values, so that both data arrays (input data $\bar{\Phi}$ and the velocity vector $\bar{F}$) will be accessed only once which is shown in Fig. 5(a) for a single processing instance. Fig. 5(b) shows the cache implementation with doubled data throughput for parallel update of two adjacent nodes. Further parallelisation is not reasonable due to restricted external data rates.

Due to the flexible architecture of FPGAs, it is possible to guarantee that numerical stiffness within the compound operation will not impair errors caused by the quantization of the image. This is achieved by delaying rounding to the final nodal value update. Opposed to the fixed word length operations of CPUs, we support growing intermediate bit widths as needed. E.g., the result of the addition of 12 bit data is 13 bit width, the result of the multiplication of two 12 bit data will be 24 bit long. In the end, this allows to reduce the resolution of the input image data, since error propagation is reduced.

4 Experimental Results

The demonstrator built can be loaded with a large picture, which is stored on the FPGA card's SRAM. Access to the SRAM is limited by the standard RAM interface, therefore the calculation speed of the design is determined be the amount of RAM elements needed per node update. The FPGA card uses 36 bit wide external asynchronous SRAM with 13 ns delay. Adding the FPGA's I/O delays, buffered SRAM access with up to 50 MHz was expected to be realizable. Since synchonous 50 MHz designs on the XC4085XLA with a CLB usage of over 80 % are highly implausible we decided to split the design in a small high speed SRAM component and a large computation engine which works with doubled latency. Fig. 6 shows the SRAM controller design. The SRAM72 component was optimized by means of manual placement and further enhanced by adding dual edge clock support which allows to realize pipelined SRAM access, which exploits the internal SRAM latencies. This results in a 32% increase of the data throughput resulting in a constant unidirectional

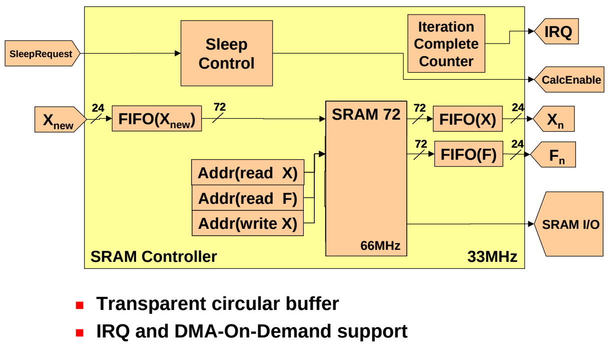


Figure 6: SRAM controller implementation.

72 Bit*33MHz data rate. There are two parallel node update engines on the demonstrator. Each is based on a highly pipelined architecture which was generated by Synopsys Synthesis tools with automated delay balancing (Fig. 8).

The node update engines work on 12 Bit data values, which motivates the SRAM controller design. The SRAM controller does on-the-fly multiplexing and demultiplexing of the 2*12 bit data sets which are delivered at each clock cycle eventhough the SRAM72 component writes or reads 72 bit data vectors at each clock cycle. The design is optimal in respect to performance, as a matter of fact it is not possible to get better data throughput without fundamental changes of the FPGA card.

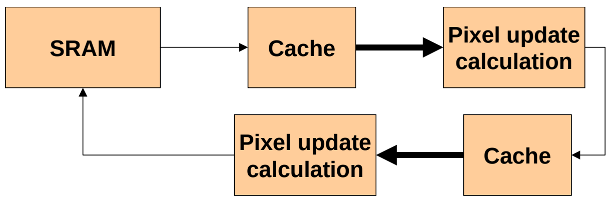


Figure 7: Chaining processing elements.

We are caching all reusable data within the FPGA by using internal buffer components as illustrated in Fig. 5(b). These buffer components can be chained which allows for calculation of multiple iterations without intermediate storage of data in the external RAM (Fig. 7). The XC4085XLA allows for a two iteration design. Therefore, the final frame could be calculated, which was experimentally validated: The final design processes approximatly 2000 frames per second which corresponds to approximately $2,2 \cdot 10^9$ integer operations per second implemented on an FPGA which is available since 1998.

Pipeline Stages	LUT Area Count	FF Count	Best Case Rate [MHz]
1	558	51	16,52
2	558	181	26,42
3	565	229	34,58
4	558	280	37,38
5	558	358	40,51
6	558	407	40,52
7	558	498	40,55
8	526	557	40,57

Figure 8: Synthesis results for a single node update unit.

Bibliography

[1] B. Engquist and S. Osher. Stable and entropy-satisfying approximations for transonic flow calculations. *Math. Comp.*, Vol. 34(149):45–75, 1980.

[2] S. Klupsch. Design, Integration and Validation of Heterogeneous Systems. In *2nd IEEE International Symposium on Quality Electronic Design (ISQED 2001)*, San Jose (CA), Mar. 2001. International Society for Quality Electronic Design (ISQED Org.).

[3] J. A. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, Cambridge, 2nd edition, Nov. 1998. ISBN 0-521-64204-3.

[4] Silicon Software. microEnable Users Guide, 1999.

[5] Xilinx. Programmable Logic Data Book, 1999.