Space Debris Detection on the HPDP, A Coarse-Grained Reconfigurable Array Architecture for Space

D. Suárez[a, b], J. Weidendorfer[a], T. Helfers[b], D. Bretz[b], J. Utzmann[c]

[a]Technische Universität München, Boltzmannstraße 3, 85748 Garching, Germany
[b]Airbus Defence and Space GmbH, Robert-Koch-Straße 1, 85521 Ottobrunn, Germany
[c]Airbus Defence and Space GmbH, Claude-Dornier-Straße, 88090 Immenstaad, Germany

diego.suarez@airbus.com

Abstract

Stream processing, widely used in communications and digital signal processing applications, requires high-throughput data processing that is achieved in most cases using ASIC designs. Lack of programmability is an issue especially in space applications, which use on-board components with long life-cycles requiring applications updates. To this end, the HPDP architecture integrates an array of coarse-grained reconfigurable elements to provide both flexible and efficient computational power suitable for stream-based data processing applications in space.

In this work the capabilities of the HPDP architecture are demonstrated with the implementation of a real-time image processing algorithm for space debris detection in a space-based space surveillance system. The implementation challenges and alternatives are described making trade-offs to improve performance at the expense of negligible degradation of detection accuracy. The proposed implementation uses over 99% of the available computational resources. Performance estimations based on simulations show that the HPDP can amply match the application requirements.

I. INTRODUCTION

A hardware architecture supporting parallelism, such as pipelining and data-flow parallelism is of high importance in stream-processing applications, in which conventional processors do not deliver the required performance efficiently. An Application-Specific Integrated Circuit (ASIC) achieves low power consumption with the best performance, but lacks of any reconfiguration capabilities needed especially in space applications where the on-board hardware has long life-cycles and might require application upgrades. On the other hand, a Field-Programmable Gate Array (FPGA) allows reconfigurable hardware design at gate level, offering more flexibility than an ASIC at expenses of higher power consumption, more silicon and at a relatively reduced maximum clock frequency, but capable of achieving better computational performance than processors in stream-based applications [2]. However, fine granularity reduce performance in an FPGA because of the complexity of the programmable connections used to build logic blocks [3].

As a result, architectures are evolving towards hardware with reconfigurable capabilities that integrates modules that can be configured to efficiently perform frequently used operations. The eXtreme Processing Platform (XPP) is the core of the High Performance Data Processors (HPDP) architecture [4]. The XPP allows runtime reconfiguration of a network of coarse-grained computation and storage elements. The algorithm's data-flow graph is implemented in configurations, in which each node is mapped to fundamental machine operations executed by a configurable Arithmetic Logic Unit (ALU) [5].

The present work aims to determine the effectiveness, portability and performance of an image processing algorithm in the HPDP architecture. Space debris is a major issue for operational satellites and spacecraft. A Space Based Space Surveillance (SBSS) mission using an optical telescope has been proposed [1] in order to detect and track such debris. The required frame rate for the instrument calls for an efficient on-board image processing implementation in order to keep payload data volume within limits. Such on-board data reduction can be implemented by detecting features of interest (debris, stars) while omitting the remaining image content (noise, space background).

The main objective of porting the algorithm to the HPDP architecture is to fulfil the requirement of real-time detection of space debris. Portability analysis covers use of hardware resources among different implementation alternatives, its parallelisation capabilities, throughput, memory usage (size and required bandwidth) and errors derived from rounding and data truncation.

The paper is structured as follows. The first section introduces the HPDP architecture with its constitutive elements. Next, the theory behind the boundary tensor algorithm as a feature detection method is explained. Then, the implementation of the algorithm in the HPDP is described. In the following section, the cycle-accurate simulation results are presented to measure the throughput of the algorithm running on the HPDP, estimate the performance in the expected hardware, and quantify the detection error. Finally, the objectives are evaluated and conclusions are given.

II. THE XPP AS THE CORE OF THE HPDP

The XPP is a runtime-reconfigurable data processing architecture, that combines a coarse-grained reconfigurable data-flow array with sequential processors. This allows mapping regular control-flow algorithms that operates over a

stream of data and achieve high throughput. Control-flow dominated tasks can be executed in the programmable processors [5].

The XPP Core consists of three types of Processing Array Elements (PAE): arithmetic logic unit PAE (ALU-PAE), random access memory with I/O PAE (RAM-PAE) and the Function PAE (FNC-PAE). ALU-PAE and RAM-PAE objects are arranged in a rectangular array, called the XPP Data-flow Array [6].

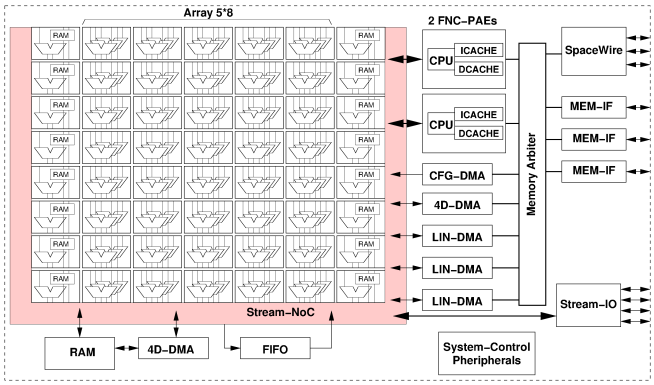


Figure 1: Overview of the HPDP architecture [4]

For the implementation of the feature detection algorithm the XPP-III 40.16.2 core is used, consisting of 40 ALU- PAE objects arranged in a 5x8 array, 16 RAM-PAE and two FNC-PAE. For the HPDP project the XPP core has been selected by Airbus DS due to the availability as HDL source code among others. This enables the implementation on the STM65nm semiconductor technology, using a radiation hardened library. The elements in the library are designed such that radiation effects such as bit flips in storage elements and transients on control signals lines are very much limited. This makes the resulting HPDP chip suitable to operate in all earth orbits and be- yond. The development of this chip is currently on-going, first prototypes are expected in the second half 2016.

III. Algorithm for Space Debris Detection

The objective of the used algorithm is to detect linear streaks formed by space debris trails. A linear feature is defined as a neighbourhood of image pixels with an intensity distribution forming a patter fitting in a line with some width and length, and with a high enough signal-to-noise ratio (SNR) to be detected.

The boundary tensor [7][8] combined with thresholding is used as the detection algorithm to obtain a binary image containing the detected objects.

The boundary tensor is constructed combining the results of applying a set of polar separable filters to the input image. It has been demonstrated that an adequate linear combination of the results of applying a set of polar filters to an image, produces a single peak of energy when an edge is found, regardless of the type of symmetry in the feature: step edges that exhibit even symmetry or roof edges that has odd symmetry [7]. Filtering is performed in the spatial domain, saving computational efforts compared with filters working in the frequency domain where Fourier transformations are required. For this purpose, a set of even and odd filters are used and the filtering operation is implemented as a set of 1-D Convolutions along the columns and rows of the image, generating a set of odd and even responses. Their energies are combined to obtain the boundary tensor. Seven filter kernels are used, which are calculated from the Gaussian function and successive derivatives.

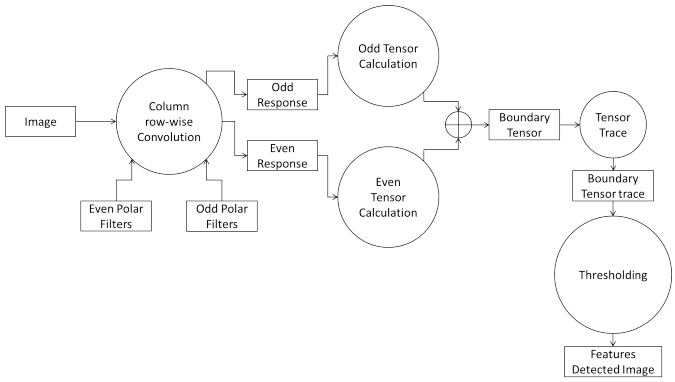


Figure 2: Boundary tensor and thresholding data-flow graph for space debris detection

IV. Porting the Data-Flow Graph to the XPP Array

Convolution is the basic operation of most signal processing algorithms. For the boundary tensor algorithm seven row-wise convolutions and seven subsequent column-wise convolutions are used to calculate the even and odd responses. The convolution process accounts for 80% of the data processing required for the whole boundary tensor algorithm. Thus, its implementation has a high impact in the final performance. Four types of operations are required to complete the convolution stage as illustrated in Figure 3.

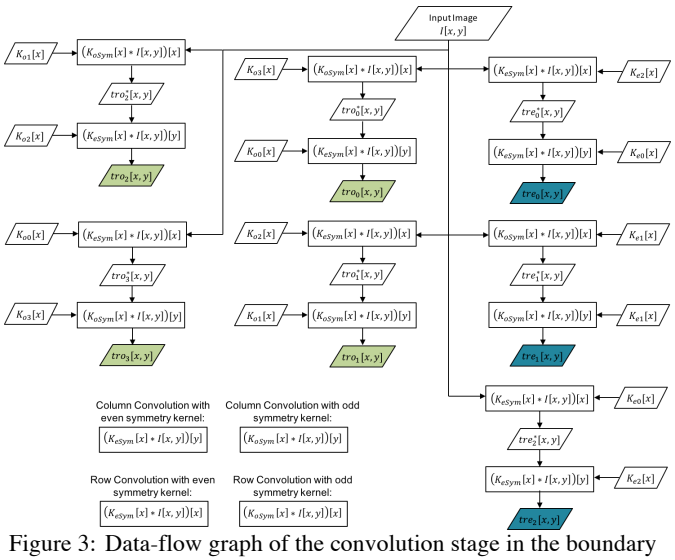


Figure 3: Data-flow graph of the convolution stage in the boundary tensor algorithm for feature detection

A. 1-D Convolution Implementation

The reference design of boundary tensor [8] requires, in first instance, floating-point arithmetic. However, hardware for signal processing often uses fix-point arithmetic because floating-point support needs more hardware resources. This in turn increases power consumption. Furthermore, issues may arise in time-constrained applications since operations could take an unpredictable amount of time [9]. Therefore, convolution is implemented using fix-point arithmetic in this work. Kernels with radius r = 3 are used.

1) Bit-Width for Data Representation in XPP computations

The XPP array does not have enough computational elements to calculate several convolution sets in one configuration. And it has neither enough internal memory elements to perform convolution rounds with different kernels, without having to stream-out intermediate results to the system's memory. Therefore, the bit-width value representation used in the XPP computations has great influence in the volume of data exchanged between the XPP array and the system memory and, in consequence, impact in the performance. The input pixels are unsigned 16 bit values (`uint16`), signed arithmetic is required due to the negative elements of some kernels, and that the 4-Dimensional Direct Memory Access (4D-DMA) can transfer data at a maximum of 64 bits/cycle. A trade-off between accuracy and performance is possible. If the full-resolution input pixels are used for computation, two 16 bits data buses from the XPP array are required to hold computation values. This means that a pixel is represented by an `int32` value and the 4D-DMA is only capable to transfer 2 pixels/cycle. However, if the least significant bit (LSB) of the input pixels is truncated, all computations fit into 16 bits, therefore 4 pixels/cycle can be streamed to the XPP array. Additionally, the `int16` implementation requires the transfer of half the data volume than the `int32`, at expenses of inducing an error in the detection result. This LSB truncation approach is used and detection error is analysed.

2) Overflow consideration

Kernels that are derived from the Gaussian function are normalised, which means that the sum of the absolute value of the kernel elements is equal to one. In addition, for kernels obtained from the successive derivatives of the Gaussian function, it can be demonstrated that the sum of the absolute values is a positive number less than one.

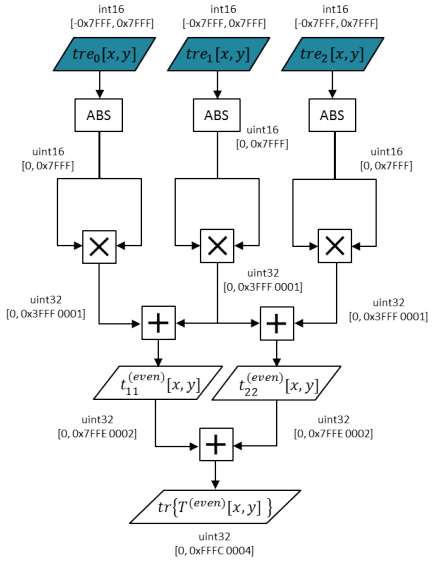
3) Resource Optimisation based on Kernel symmetry

To convolve a full row (or column), a convolution is executed over all its pixels. At the end of this process, each kernel element is assumed to be multiplied with all pixels in the row (column). This rule applies to all pixels except the ones near the borders, i.e. the first and last r pixels in the row or column. These are not multiplied by all kernel elements, but only by r of them. In first instance, it is possible to assume that for each pixel convolution 2r+1 multiplications must be done and 2r additions must be calculated.

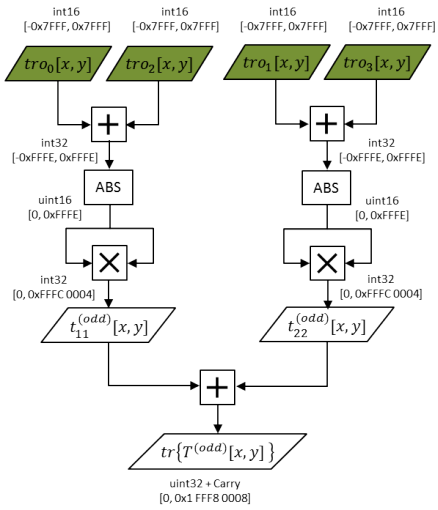
Symmetry in a kernel is advantageous for the implementation, because it reduces the number of necessary multiplications between kernel elements and pixels. In the case that the kernels show even symmetry, the values at each side of the vertical axis are a reflection of the other side. As a result, only r+1 multiplications are necessary. For kernels with odd symmetry, the central element is always zero and the elements at one side of the vertical axis have the same magnitude with opposite sign than the values at the other side. This means that using this kind of kernel, only r multiplications are needed per pixel convolution.

B. Boundary tensor trace calculation.

Boundary tensor calculation is performed only once at the end of the algorithm and its complete implementation fits in a single XPP array configuration. For this reason, there are no intermediate values that must be temporarily stored in the system memory to be streamed-back to the XPP array for further processing. As illustrated in Figure 4 calculations are carried out using the given bit-width.



a) Even tensor calculation



b) Odd tensor calculation

Figure 4: data-flow graph of the even tensor calculation, with data type and value ranges

III. RESULTS

In this section the performance of the feature detection algorithm executed in the HPDP is evaluated. The HPDP chip is not yet available. The following runtime estimates are derived from a cycle-accurate simulation of the XPP array and the expected clock frequencies as given in the following section.

C. XPP Array Throughput

For determining the throughput of the complete implementation, each pipeline in every of the six configurations (i.e. row and column-wise convolution with even and odd symmetry, transpose and boundary tensor calculation) is executed in the HPDP simulator. The maximum average throughput is 3.98 Bytes/cycle, which is achieved by the configurations computing convolution with odd symmetry kernels.

For an XPP array working with a 200 MHz clock, after the data flow in the pipelines has been balanced, a maximum of 796 MBytes/s will be flowing into the XPP array for processing and, at the same bit rate, results will be generated. Thus, for a single memory port, the minimum bandwidth to provide and store-back a continuous data stream to the XPP array is 1592 MBytes/s. However, this requirement is not met by the assumed HPDP hardware specification which integrates two 64-bit wide memory ports: one with an internal 4 MBytes SRAM operating at 100 MHz (i.e. 800 MBytes/s) and another with an external DRAM attached running at 50 MHz (i.e. 400 MBytes/s). So the maximum theoretical bit transfer of the SRAM is nearly half the bit rate at which the XPP array is consuming data and generating results for the implemented algorithm.

1) Sub-image Processing

To achieve the best performance for the given specifications, the SRAM should be used for all memory transactions required for the convolution and boundary tensor calculation. This implementation requires eight image buffers for complete execution. One stores the input image, and the other seven hold the row-convolution results. Because the transposition operation reads the input image column-wise and writes the result row by row, it is not possible to use the same origin and destination buffer for this operation, otherwise loss of data will occur. Splitting the 2048x2048 pixels input image in 16 parts, produces sub-images that can be processed one at a time using eight 512 KBytes sub-image buffers stored in SRAM. DRAM is used to store the input and result image.

2) Estimated Computation Time

The performance of the algorithm on the specified HPDP hardware is determined by the memory speed. Based on the number of write and read operations needed for the complete algorithm, an estimation of the execution time of the feature detection algorithm is computed. The algorithm completion time for the expected HPDP hardware is 734 ms using sub-image processing with SRAM, compliant with the maximum one second requirement for processing a 2048x2048 pixels image.

3) Detection Accuracy

For each detected streak in the binary image obtained from the HPDP simulation, there are approximately 10% less detected pixels compared with the reference implementation, as shown in Figure 5 for an input image containing a streak with an SNR of 7.19 dB. The error is negligible since the detection information per object can then be used to store full streak pixel values in order to not lose accuracy with respect to the position and brightness in a further processing step on-ground.

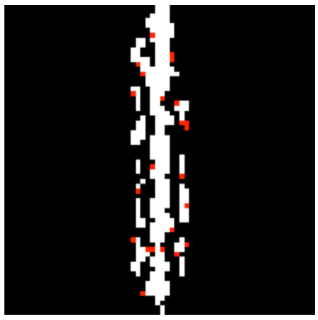


Figure 5: Comparison between reference and HPDP implementation. Detection values present in the reference implementation but not in the HPDP results are highlighted in red, and represent 10% of miss-detected pixels.

V. CONCLUSIONS

In this paper, we showed that the boundary tensor algorithm can be mapped to a data-flow graph and a simple control flow is only required for filter kernel update, border replication and pipeline cleaning tasks. Thus, the XPP array is appropriate for its implementation, reaching in average 4.7 GOp/s, for 16-bit fixed-point multiplication-addition operations. The model used for the convolution implementation makes possible to implement pipeline parallelism, because the image input stream is multiplied first by all kernel elements and the adder module receives the required multiplication results as they are produced. Moreover, convolution is appropriate for task parallelism in XPP array, since four consecutive pixel streams are received, and four pipelines can compute the convolution of four pixels simultaneously, without data dependencies. The utilisation of 99% of XPP array computation elements (e.g. ALU-PAE), and the use of the maximum transfer mode of the 4D-DMA, shows that this implementation is taking advantage of all the capabilities of the architecture.

Additionally, it has been determined that for a noise-less detection, the SNR of the feature must be greater than 7.19 dB. This specifies the capabilities of the implemented algorithm and shall be used as a detection-effectivity benchmark for comparison with other detection algorithms.

In terms of scalability, the XPP array configuration (use of array objects and connections) implementing this algorithm is

independent from the dimensions of the input image. The size of the kernel has direct impact in the required operations and as consequence more XPP array resources are needed if the kernel radius is increased. This is determined by the deviation of the Gaussian function that generates the filters. The deviation value has an impact on the geometry of the features that can be detected.

Finally, the LSB truncation is an effective alternative to meet the real-time requirement because the gain in performance is greater (twice as fast) than the error caused in the detection, evidenced by a loss of only 10\% of high-detection pixels. Integer arithmetic keeps the hardware implementation at the lowest level of complexity, using less resources, reducing power consumption and assuring computation time determinism, with negligible error.

To summarize, our experience from implementing the given algorithm shows that the coarse-grained reconfigurable array approach successfully can achieve typical requirements in space. A key feature is the fast re-configurability, which not only makes programmability possible in the first place, but also allows even complex data flows to be implemented in multiple configurations with modest hardware resources and still high data streaming throughput.

REFERENCES

[1] Utzmann, J., Wagner, A., Silha, J., Schildknecht, T., Willemsen, P., Teston, F., Flohrer, T. (2014, October). Space- Based Space Surveillance and Tracking Demonstrator: Mission and System Design. 65th International Astronautical Congress, Toronto, Canada.

[2] Bailey, D. (2011, June). Design for Embedded Image Processing on FPGAs John Wiley & Sons.

[3] Bobda, C. (2007). Introduction to Reconfigurable Computing: Architectures, Algorithms, and Applications. Springer Netherlands.

[4] Syed, M., Acher, G., Helfers, T. (2013, May). A High Performance Reliable Dataflow Based Processor for Space Applications. Proceedings of the ACM International Conference on Computing Frontiers.

[5] Schüler, E., Weinhardt, M. (2009). XPP-III: Reconfigurable Processor Core. In: A. R. Nikolaos Voros and M. Hübner, Eds. Dynamic System Reconfiguration in Heterogeneous Platforms: The MORPHEUS Approach, Chap. 6, Springer Netherlands.

[6] PACT XPP Technologies AG. (2006). XPP-III Processor Overview White Paper. Germany.

[7] Köthe, U. (2003, October). Integrated edge and junction detection with the boundary tensor. Ninth IEEE International Conference on Computer Vision.

[8] VIGRA Homepage, Heidelberg Collaboratory for Image Processing. http://ukoethe. github.io/vigra/

[9] Owen, M. (2007). Practical Signal Processing. Cambridge University Press.