RANSAC Acceleration on FPGA Multi-Processor with Overlay Extension

Danielle Tchuinkou
CSCE Department, University of Arkansas
dtchuink@uark.edu

Joel Mandebi
CSCE Department, University of Arkansas
jmmandeb@uark.edu

Christophe Bobda
CSCE Department, University of Arkansas
cbobda@uark.edu

ABSTRACT

Random Sample Consensus (RANSAC) is commonly used in 3D vision to compute mathematical models needed for structure extraction from points clouds. Although the algorithm is simple, its execution is computationally expensive, particularly in embedded systems with restricted resources. We propose an FPGA implementation of RANSAC that uses multiple embedded processors, each extended with an overlay to accelerate repetitive and time consuming instructions. The proposed architecture shows a performance improvement of 82% compared to a software only solution. While the resource overhead is higher, it amounts for only 25% of the resource of an Altera Cyclone IV FPGA.

Keywords

Multi-Processor SoC, RANSAC, Overlay, Point Cloud.

1. INTRODUCTION

Visual scene understanding is one of the most efficient, non invasive methods to assess the environment around us. Cameras are used in this process to capture and analyze a scene without requirement to objects and person in the scene. Decreasing costs and size of processors, along with their increasing performance, have led to embedded smart cameras that can process information in real-time in-situ, and thus addressing privacy concerns in sensitive applications areas such as nursing homes[1] and manufacturing[2]. Being able to analyze and detect objects through images and videos is a major topic in research.

RANSAC[3] is a robust estimation algorithm to estimate various mathematical models. In[4] the authors present an FPGA implementation of image based ellipse estimation for an embedded eye tracking system. They use RANSAC to estimate the ellipse shape of pupil contour in eye. While most camera systems rely on 2D image processing, 3D image processing has proven more accurate and more reliable in critical applications. 3D image processing extends 2D imaging by a third coordinate, the depth, which makes it easy to compute distance to objects and size of objects within the scene. While the advantages of 3D image processing towards their 2D counterpart are far reaching, the computational challenges of 3D processing have limited their deployment, in particular in embedded systems. In view of the great interest in 3D images, much work has been done on 3D reconstruction with FPGA. In [5], the authors describe an FPGA implementation of one important iterative kernel called Expectation Maximization, which is the major point of computation for the 3D reconstruction algorithm. However, Gauthier and al.[6] present an FPGA implementation of the Iterative Closest Point and the Volumetric Integration algorithms for the 3D reconstruction. Their design helped them to achieve a low-power 3D reconstruction system which operates in real-time. Acquiring point clouds is already possible with embedded systems, but much work remains for the visual scene.

In this paper, we present a hardware acceleration for structure recognition from 3D points clouds representing a certain scene. They can then be fed to an on board processing engine for extraction of structures such as walls, doors, and tables along with their size and distance away from the recording point. The core of the 3D structure extraction is based on the RANSAC method. The flexibility of the algorithm allows for use in various applications through a simple update or change of the model [7].

The processing in real-time requires a viable computational platform capable of balancing the memory bandwidth requirement and the processing power. Acceleration of RANSAC in embedded systems has been limited to porting the parallel implementation of to multi processors on FPGA using softcore, without hardware acceleration [8].

In this work, we present a multi-core architecture with an overlay extension on each core. The overlays are used to accelerate the computation of the deviation error. While overlays are slower than pure hardware implementations, they provide enough computational power for our need, along with the flexibility required to adapt to new models at runtime without the need for hardware synthesis. Implementations of our system with points cloud of various sizes shows a performance improvement of 82% toward a sequential software-only solution and 62% toward a parallel implementation without hardware acceleration.

After discussing the details of the RANSAC algorithm as well as its parallel implementation, our multi-core architecture with overlay extension will be presented along with the overall computation flow. The implementation details and results will then be provided, followed by a conclusion and remarks.

This work was presented in part at the international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2017) Bochum, DE, June 7-9, 2017.

2. RANSAC OVERVIEW

RANSAC is an iterative algorithm having the goal of estimating parameters of a mathematical model from a set of data with a significant amounts of noise. A model can be a plane, a sphere or a cylinder. The set of data used by RANSAC is usually called a *points cloud*. A points cloud is a set of three dimensional points defined by (x, y, z) coordinates. In this context, all the points belonging to the model are called *inliers*, while the others are *outliers*. In order to find the best parameters, the algorithm needs to iterate a certain number of times. An overview of RANSAC execution flow [9] is shown in Figure 1. At stage 1, the program

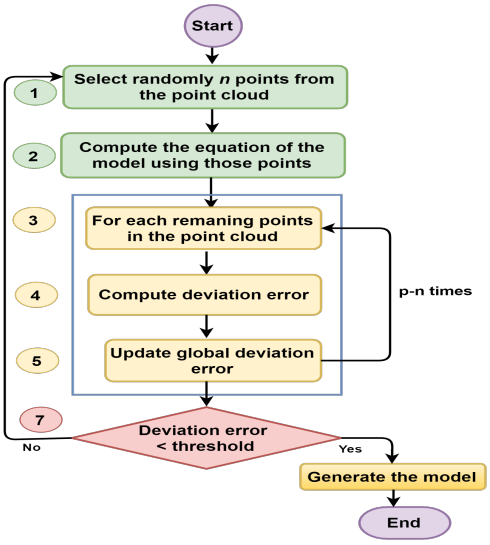


Figure 1: RANSAC Sequential Processing

chooses randomly a number of points n from the whole set of p points. The number n depends on the model chosen. For instance, if the model is a plane then three points are going to be used. Having chosen the minimum number of points, the algorithm computes the equation representing the model. For the $p-n$ remaining points, we compute the distance to the model(*the deviation error*). If the point is close enough to the model with a certain tolerance, it is labeled as an *inlier*, otherwise it is an *outlier*. We then update the global deviation error to have an overview of the distribution of the points through the model. In case of a global deviation error smaller than the threshold, we compute the parameters of the model using the inliers to find a better model, otherwise we continue iterating for a certain number of times. To get the best possible model, we need to find the right minimum number of iterations. This is determined by a probability p to find a correct model [9]. So, we need to iterate *Iterations* number of times (See equation 1).

$$Iterations = \frac{\log(1-p)}{\log 1 - W^n} \quad (1)$$

In equation 1, W is the probability that a point belongs to the model. Since the number of iterations depends on the probability for a point to belong to the model, RANSAC is a non-deterministic algorithm because it produces a reasonable result with a certain probability, which also depends on the number of iterations. The higher the number of iterations, the higher the chances are to get a better model. The sequential execution of this algorithm is computationally intensive because of the typically high number of iterations.

3. OUR PARALLEL RANSAC IMPLEMENTATION

As shown in Figure 1, the models found are different from iteration to iteration. In addition, operations going from stage 1 to 5 are independent from one iteration to another. Those two observations make the whole RANSAC execution suitable for parallelization. Distributing iterations into many parallel processors can significantly decrease the global execution time without impacting the quality of the results.

3.1 Proposed Architecture

As explained in section 2, a parallel version of RANSAC is possible due to the independent nature of operations at each iteration. Thereby, we can distribute iterations into many processors. Among all those processors, one would be chosen as a **master** (see Figure 2). Its role is to allow data access to other processors called **workers** and collect the equation of the best model they produced at the end, to chose the one having the smallest deviation error. With k workers, each worker will iterate $\frac{I}{k}$, where I is the number total of iterations. To avoid concurrent accesses to a shared memory and the overhead of a memory coherency protocol, we assign a private memory to each processor to store their data. In addition, the size of points clouds might reach hundreds of megabytes or more. Therefore, loading a points cloud into those memories in conjunction with programs to be executed becomes a non-viable option. Having a shared memory capable of storing points cloud becomes a necessity. However, to ensure the integrity of the data, only the master can write in that memory.

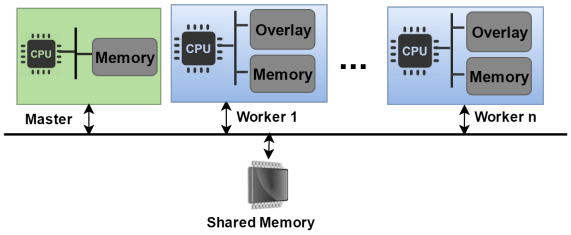


Figure 2: RANSAC Parallel Architecture

Once the workers have access to the points cloud, we first select the minimum number of points to compute the equation of the model. Because every worker has exactly the same point cloud, we want to ensure that the points selected randomly by a worker will not be used by another one. To this end, we keep a list called ***used list*** of every combination of points chosen so far. After selecting the points, we verify if they have already been used in *used list*. If yes, we select other points. Otherwise, we register this combination in the current *used list*, and notify the other workers that this combination of points is not longer available and compute the equation of the model. We next determine the inliers, the outliers and the global deviation error of the model. If the the deviation error is the best so far, we send it back to the master. The master compares the received deviation error to its own. If it is smaller, it send it back

to the other workers. This guarantees that the master will always receive a smaller deviation error. The workers iterate until they reach the maximum number of iterations, or when they find a model with a deviation error smaller than the defined threshold. At the end, the master chooses the best model received from the workers. However, we notice that for every model found, obtaining the deviation error is a repetitive process. For example, if you want to find a plane from a given point cloud, the equation of this plane in the three dimensional system is expressed by equation (2)

$$ax + by + cz + d = 0 \quad (2)$$

where a, b, c and d are the parameters of the model and x, y and z are independent variables from the model like the values of a point. In that case, the deviation error is given by equation (3).

$$deviation_{error} = |ax + by + cz + d| \quad (3)$$

RANSAC was used in the implementation of a real-time affine geometry estimation on FPGA[10] for real-time video processing. Since software processors may not produce good results for larger data sets, only the most computationally intensive tasks in RANSAC iterations will be accelerated in hardware. The authors found that the calculation of the error accounted for more than 80% of the overall time.
In order to restrain computation times, we use a *hardware accelerator* to calculate deviation errors. The hardware accelerator is a custom overlay architecture offering a set of processing elements to carry out calculations (see section 3.2). At each iteration, once having the equation of the model, the worker sends the value (x, y, z) of each remaining points to the *hardware accelerator*, one after the other. The overlay then computes the deviation error for each point and sends back the result to the worker. The overlay also offers us the possibility of not having to reconfigure the hardware in case the model that we wish to detect had to change. In this case, it will only be necessary to update the placement of the calculation of the new error on the overlay.

With the previous analysis, the architecture needed to perform the parallel implementation of RANSAC requires n independent processors. One of them will be the *master*, and the remaining $(n-1)$ ones the *workers*. The workers execute RANSAC in parallel while communicating by messages to having the best possible model. The overall architecture will use at least 3 processors. In order to speed up computations, each worker is coupled to a hardware accelerator in charge of computing deviation errors. The next section will give an insight on the overlay architecture.

3.2 Overlay Extension

In order to accelerate computations, some operations are directly implemented in the hardware. To offer a software-like programmability, we used an overlay to deploy on top of an FPGA. Our 2D-Torus overlay (See Figure 3) implements a Network-on-Chip (NoC) architecture that allows a flexible and unrestricted placement of tasks at runtime, and communications between tasks with a low latency. The size of the architecture can easily be increased or decreased to fit the needs of an application. The main components are processing elements (PE) and routers (R). PEs are gathered in group of four around routers. Routers are used to convey data packets either to PEs (configurations or operands for ALUs) or to a processor (result of computations). Each PE is mainly composed of an ALU that executes operations on 32 bit operands and a configuration register purposed for telling the PE where to take input operands from, where to forward results and which operation the ALU has to execute. Routers also contain some FIFOs to avoid collisions between packets received and implement a XY-routing algorithm[11] that is deadlock free [12]. The NoC infrastructure of the overlay offers direct communication links between neighbor PEs, which allows direct data exchanges without going through routers. This ability spares tightly coupled computations from routing overheads. The architecture used then maximizes local communications among operations belonging to a task through direct communications between PEs, and global exchanges among distant tasks through routers.

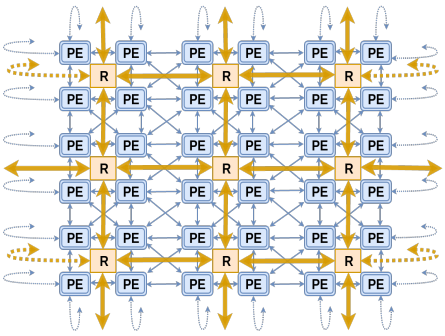


Figure 3: Overlay Architecture

To be able to ease the writing of applications to control all computing resources, we implemented a custom C/C++ mapping library offering routines to allow communication among processors and with the overlay. Since processors communicate by messages, the SoC-MPI[13] library provides functions to allow inter-processor communication. We have extended that library in order to send computations to the overlay. So, in addition to the functions provided by SoC-MPI to allow inter-processor exchanges, table 1 summarizes major functionalities enabling to access the overlay.

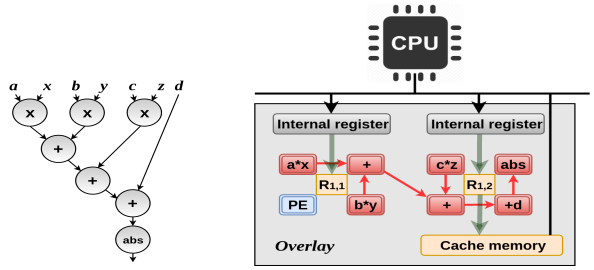
Table 1: Additional Functionalities of Our Library

Function	Description
send_configuration	Sends a configuration packet to the overlay
send_operandA	Sends the first operand of an operation
send_operandB	Sends the second operand of an operation
read_resultRegister	Reads the content of a result register of the overlay

4. EVALUATION

In order to evaluate our proposed implementation, we designed and implemented a System on Chip (SoC) on an Altera Cyclone IV E FPGA and used *planes* as model. Our implementation uses 3 Nios II/f processors (one master and the 2 workers). The limited amount of resources on the FPGA forces us to have an off-chip memory as shared memory where the points cloud will be stored. Workers also have

an overlay to getting the deviation error faster. Acceleration of the computation of deviation errors is done through a mapping of operations of equation (3) to PEs. Figure 4a shows the corresponding data flow graph (DFG). We use an instance of our overlay having two routers and eight PEs. Figure 4b shows the mapping of operation along with their placement on the overlay. Constants a, b, c and d are fixed



(a) DFG Representation

(b) Deviation Error Placement

Figure 4: Hardware Implementation of the Deviation Error

at configuration time. Only x, y and z values change from one point to the other. To increase the throughput of computations, some registers in the overlay keep the next x, y and z to use. Upon completing the calculation of a deviation error, the overlay sends an interrupt to the processor it is attached to. While the processor reads the result from a cache, the overlay loads the new operands from its internal registers. Finally, when the overlay starts working out the new deviation error, using another interrupt, it asks its processor to send the next x, y and z.

4.1 Experimental Results

We compare the proposed architecture to a sequential execution of RANSAC on a classical Nios II/f processor and to a parallel implementation of RANSAC. The tests were carried out with 3 points clouds with sizes 5000, 10000 and 50000 points. We used this data on three different implementations of RANSAC. The results are presented in the table below.

Table 2: Execution Times and Resources Usage

Number of points	Embedded CPU NiosII	Parallel RANSAC NiosII	Parallel RANSAC NiosII Overlay
	Execution Times		
5000 points	13.44 ms	8.62 ms	1.81 ms
10000 points	26.89 ms	14.59 ms	3.50 ms
50000 points	85.89 ms	44.09 ms	15.52 ms
Resource Usage			
Logic Cells	3%	8%	25%
LUT-Only LCs	1386	3970	13806
Register-Only LCs	413	1049	4609

Our implementation reaches a speedup of 82% from a sequential implementation and a speedup of 62.27% from a parallel implementation without hardware acceleration. Although the amount of resources used is higher than the other implementations, it amounts only for 25% of the Cyclone IV resource.

5. CONCLUSION

We presented a RANSAC implementation on a multi-core architecture with an overlay extension on each core. The overlay is used as accelerator to compute specific operations. Our multi-core implementation of RANSAC sharply decreases execution times compared to sequential and parallel implementations without hardware acceleration. Our future work will aim for optimizing the architecture in order to reduce the amount of communications among processors. The more processors exchanging messages, the higher the communication overhead. This trend is particularly amplified with the augmentation of the number of processors in the system. Another goal is to push more tasks to the overlay to keep decreasing computational times.

6. REFERENCES

[1] J. Kutchka, D. Tchuinkou, J. Mandebi, E. Nghonda, and C. Bobda, "Automatic assessment of environmental hazards for fall prevention using smart-cameras," in *2016 IEEE First International Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, June 2016, pp. 24–29.

[2] Toyota, "Lexus ls 460: Techniken zur unfallvermeidung," 2009. [Online]. Available: http://www.toyotasbb.de/toyota_news/lexus_ls_460_techniken_zur_unfallvermeidung.html

[3] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[4] J. Chen, J. Cong, M. Yan, and Y. Zou, "Fpga-accelerated 3d reconstruction using compressive sensing," in *Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays*. ACM, 2012, pp. 163–166.

[5] K. Dohi, Y. Hatanaka, K. Negi, Y. Shibata, and K. Oguri, "Deep-pipelined fpga implementation of ellipse estimation for eye tracking," in *Field Programmable Logic and Applications (FPL), 2012 22nd International Conference on*. IEEE, 2012, pp. 458–463.

[6] Q. Gautier, A. Shearer, J. Matai, D. Richmond, P. Meng, and R. Kastner, "Real-time 3d reconstruction for fpgas: A case study for evaluating the performance, area, and programmability trade-offs of the altera opencl sdk," in *Field-Programmable Technology (FPT), 2014 International Conference on*. IEEE, 2014, pp. 326–329.

[7] Y. Guo, M. Bennamoun, F. Sohel, M. Lu, and J. Wan, "3d object recognition in cluttered scenes with local surface features: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2270–2287, 2014.

[8] M. Fularz, M. Kraft, A. Schmidt, and A. Kasiński, "Fpga implementation of the robust essential matrix estimation with ransac and the 8-point and the

5-point method," in *Facing the Multicore-Challenge II*. Springer, 2012, pp. 60–71.

[9] A. Hidalgo-Paniagua, M. A. Vega-Rodríguez, N. Pavón, and J. Ferruz, "A comparative study of parallel ransac implementations in 3d space," *International Journal of Parallel Programming*, vol. 43, no. 5, pp. 703–720, 2015.

[10] J. W. Tang, N. Shaikh-Husin, and U. U. Sheikh, "Fpga implementation of ransac algorithm for real-time image geometry estimation," in *Research and Development (SCOReD), 2013 IEEE Student Conference on*. IEEE, 2013, pp. 290–294.

[11] R. M. P. Shubhangi D Chawade, Mahendra A Gaikwad, "Review of xy routing algorithm for network-on-chip architecture," *International Journal of Computer Applications*, vol. 43, no. 21, pp. 0975–8887, April 2012.

[12] N. H. H. Muhammad Athar Javec Sethi, Fawnizu Azmadi Hussin, "Survey of network on chip architectures," *Sci.Int(Lahore)*, vol. 27, no. 5, pp. 4133–4144, 2015.

[13] P. Mahr, C. Lörchner, H. Ishebabi, and C. Bobda, "Soc-mpi: A flexible message passing library for multiprocessor systems-on-chips," *2008 International Conference on Reconfigurable Computing and FPGAs*, pp. 187–192, 2008.