Design and Analysis of a Multi-core PDF Estimation Algorithm on FPGAs

Karthik Nagarajan
Department of Electrical and Computer Engineering
University of Florida, Gainesville, FL 32611
nagkart@ufl.edu

Madhan Sivakumar
Department of Electrical and Computer Engineering
University of Florida, Gainesville, FL 32611
madhanuf@ufl.edu

***Abstract* —** *FPGAs (Field Programmable Gate Arrays) and the field of "High Performance Computing" have been applied to computationally intensive problems in various domains mainly addressing speedup issues. However, there is still a significant need of in-depth research and proof of success with real applications for proposing them as solutions for a more general class of problems. Along this line, this work involves the design, development and analysis of a multi-core Probability Density Function (PDF) estimation algorithm using Gaussian kernels on FPGAs. Speedup and performance metrics were obtained and discussions addressing scalability issues suggest that FPGAs are a good choice for a wide class of applications that are primarily based on density function estimations.*

Keywords – FPGA, Rapid PDF estimation, Gaussian kernel, Parallel architecture designs, High performance computing

I. INTRODUCTION

MPI (Message Passing Interface) and UPC (Unified Parallel C) are two popular parallel programming models widely used by programmers for achieving high performance. This work addresses the fundamental issue of achieving speedups through parallelism at the hardware level. FPGAs lie between GPPs (General Purpose Processors) and ASICs (Application-Specific Integrated Circuits) in the spectrum of processing elements in that they are highly flexible and have potentials for high performance. FPGAs, though extremely flexible and known for ad hoc designs, run at a much slower clock frequency when compared to traditional processors. Speedups and hence efficient deployment of cores on FPGAs depend on the extent to which the cores can be parallelized. In general, they could potentially find applications in any field involving algorithms that can make use of extensive parallelism. Along this line, owing to their architectural benefits, application cores like FFTs (Fast Fourier Transforms) [1], convolution [2], LUD (Lower Upper triangular matrix Decomposition) [3] and other BLAS (Basic Linear Algebra Subprograms) algorithms have been effectively ported to FPGAs. However, there is still a significant need of in-depth research and proof of success with real applications for proposing them as solutions for a more general class of problems. Decisions regarding bit precision, design architectures, memory access and storage depend on the FPGA's resource availability and the slice utilization of the core design.

Non-parametric PDF estimation is found to be computationally intensive and is an important core in pattern recognition/machine learning algorithms, finger print recognition, bayesian classification, feature extraction [4], bioinformatics [5], networking [6], stock market prediction [7,8] and image processing [9]. Due to the computational complexity, researchers are investigating ways of solving problems (with acceptable error rates) under a reduced number of dimensions. Efficient development of the core on FPGAs will have a broader impact in the academic community in that researchers would still be able to work on high dimensional problems and yet produce results under a reasonable timeframe.

Significant work has been carried out that discuss the feasibility of porting some of the above mentioned applications on FPGAs. In [10], the authors present a MATLAB to VHDL application mapper. Though the paper is centered on the mapper development, the authors use a Gaussian kernel estimator as one of the case studies. It closely relates to this work in that we propose to employ Gaussian kernels to estimate density functions.

An object recognition algorithm is implemented by traversing data in a parallel fashion in [11]. The paper stands as an example to showcase how a data parallel programming model can be utilized to solve a difficult problem. From an application engineer's perspective, the paper neither addresses error or performance analysis on the algorithm side nor the hardware side. Object recognition based on probability densities give residual information addressing issues like noise, error rate and accuracies. These metrics, though not very appealing to the RC (Reconfigurable Computing) and FPGA community, are very important for researchers in the application domain. As a motivation to this work, the authors in [12], [13] develop signal processing applications on FPGA's for potential speedups. They make a strong note on the implications of hardware-software co-design as well and also the possibilities of having multiple architectures (with tradeoffs in solution accuracy and resource utilization).

Non-parametric density estimation, their importance in solving practical problems and theory is discussed in section II. The design methodology and architecture are presented in section III followed by discussions on results and performance analysis in section IV. We conclude the work in Section V by listing insights gained and scope for future work.

II. BACKGROUND

A. Non-parametric PDF Estimation

The common parametric forms of density functions rarely fit the densities encountered in practice. In particular, all of the classical parameter densities are unimodal (having a single local maximum), whereas many practical problems involve multimodal densities. Furthermore, one's hopes are rarely fulfilled that a high-dimensional density might be represented as the product of one-dimensional densities. This work targets the estimation of nonparametric PDFs using the Parzen window technique. The computational complexity of the algorithm is of order $O(Nn^d)$ where N = Total number of data points, n = Number of discrete points in the PDF and d = Number of dimensions. The Parzen window technique is a generalized nonparametric approach to estimating probability density functions in a d dimensional space. The probability that a point x falls in region R is given by,

$$p_n(x) = \frac{k_n/n}{V_n} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{V_n}\varphi\left(\frac{x-x_i}{h_n}\right) \quad (1)$$

$$k_n = \sum_{i=1}^{n}\varphi\left(\frac{x-x_i}{h_n}\right) \quad (2)$$

where $V_n = h_n^d$, h_n is the dimension of the hypercube, x is the point at which the PDF is estimated (n in number) and x_i is the input data sample (N in number)

The kernel function $\varphi(u)$ could be as simple as a histogram function, a more general rectangular kernel or the widely favored Gaussian kernel. The first two cases fall under the class of naïve estimators. In the first case, the data range is divided into a set of successive and non-overlapping intervals (bins), and the frequencies of occurrence in the bins are plotted against the discretised data range (the rectangular kernel case is operationally similar except that we could have overlapping intervals). In either case, the bin size should be chosen such that a sufficient number of observations fall into each bin. In principle, the bin size may vary across the data range. Both the techniques are easy to implement computationally. However, the resulting PDF estimate depends on the bin size as well as the origin of the discretised data range, and is discontinuous at the bin boundaries.

B. Gaussian Kernels

Though naive estimators yield discontinuous results, the construction can be easily generalized to get continuous PDF estimates by employing different kernel functions. A sample illustration of the improvements we can achieve using a Gaussian kernel is shown in Fig. 1.

This work investigates the possibilities of extracting parallelism and optimizing the design of an estimator under the Gaussian kernel case as defined in equation (3).

$$\varphi(x, x_i) = \frac{1}{\sigma\sqrt{2\pi}}e^{\frac{-(x-x_i)^2}{2\sigma^2}} \quad (3)$$

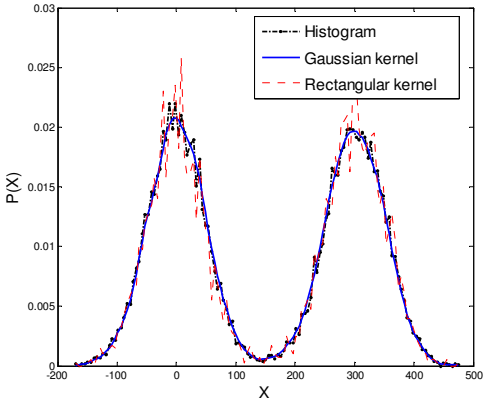

Figure 1. .PDF estimation using Histogram, Gaussian, Rectangular kernels

In specific, the probability of x is,

$$p_h(x) = \frac{1}{Nh}\sum_{i=1}^{N}\frac{1}{\sigma\sqrt{2\pi}}e^{\frac{-(x-x_i)^2}{2\sigma h^2}} \quad (4)$$

Many complex mathematical operators (sines, cosines, logarithms and exponentials to name a few) are implemented in hardware as Look-Up Tables (LUTs). But, to be applicable to a wider application community that deals with parameters with varying dynamic range, there is a necessity to take steps further than just LUTs. In this work, a second order Taylor series expansion of the exponential function is utilized to develop the core in hardware.

$$\begin{aligned}\varphi(x, x_i, h) &= \frac{1}{\sigma\sqrt{2\pi}}\sum_{k=0}^{\infty}\frac{(-1)^k}{\left(2(\sigma h)^2\right)^k k!}(x-x_i)^{2k} \\ &= \frac{1}{\sigma\sqrt{2\pi}}\left(1-\frac{(x-x_i)^2}{\left(2(\sigma h)^2\right)}\right)\end{aligned} \quad (5)$$

III. METHODOLOGY

A. Parallelization Process

At a high level, the job of parallelization involves identifying the work that can be done in parallel, determining how to distribute the work and perhaps the data among the processing nodes and managing the data access, communication, and synchronization [14]. The primary objective of parallel programming is to execute the program faster than the sequential counterpart. This necessitates load balancing, reducing inter-node and intra-node (in a multiprocessor node) communication and reducing overheads in terms of synchronization and parallelism management. Decisions regarding coarse-grained and fine-grained tasks are made based on availability of resources. We follow four steps in parallelizing a sequential program [14].

- *Decomposition* of the computation into tasks – The Parzen window technique is an embarrassingly parallel

algorithm. Here, a task comprises of all the operations that are performed on every data sample x_i. We have domain decomposition wherein data samples are distributed among processing elements and each element performs the same set of operations on each data sample.

- *Assignment* of tasks to processes – In this work, a process is considered as the computation of the probability density function value for a particular data sample x_i at one point x. All the operations performed on a single data sample are accounted as a process.
- *Orchestration* of the necessary data access, communication and synchronization among processes – Every processing element performs the process defined in the assignment stage over a set of x_i and x independently. Hence, there is no communication between processing elements and synchronization requirements are minimal. It should also be noted that data accesses are from a centralized memory.
- *Mapping* or binding of processes to processing elements – This is a straightforward step as each processing element performs the same set of operations on the data samples independently. Multiple processes are carried out in each processing element that functions in parallel on a round robin basis.

B. Development Stages

In this work, the following naming convention has been adopted to describe the architecture and development stages of each entity in the system. The kernel in the design refers to a processing element and a core to a node. This is analogous to a multiprocessor node environment wherein multiple kernels are housed in a single core. The development stages in the system design is illustrated in Fig. 2 and explained below

- *Preliminary analysis* – Decisions regarding bit precision are made and basic resources available for computation are studied. An important motivation towards using fixed point implementation in this case is because of the fact that probability values lie between 0 and 1 ($0 \le p \le 1$). Hence, more precision can be allocated to the fractional part relative to the integer part.
- *Kernel and core design* – The basic process in this work is the computation of: $\left\{1-\left[(x-x_i)\times(x-x_i)\right]\right\}$. A core contains a number of kernels (k) with each kernel in the design performing the above mentioned computation. The parameter k is chosen based on the preliminary analysis performed as a part of system design in the previous step.
- *Test bench and simulation* – Memory instantiations are made, test bench files are generated and a functional level simulation is carried out at the kernel and core level.

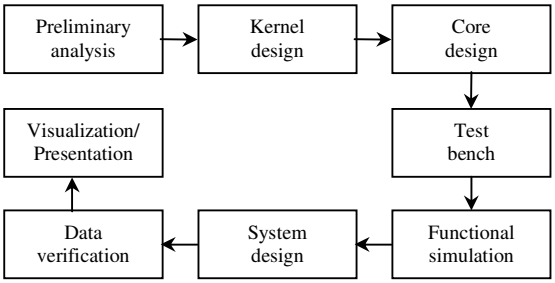

Figure 2. Development stages in the system design

- *Overall system design, verification and visualization*– Integration of the core with the host processor (middleware design) is developed followed by data verification and visualization. The latter part is performed in a high level language (HLL). Design updates are made based on these analyses.

C. Design Architecture

Since the application is ported on an FPGA, a suitable architecture is designed after careful consideration of the availability of dedicated arithmetic units and memory blocks. Not only should the available resources be efficiently used but the architecture in general should scale well in terms of application complexity and platform variability. Taking these points into account, a multi core design with a key design parameter (k – number of kernels in a core) is proposed. Single and dual core design architectures and the underlying state machine are discussed below.

1) Single Core Design

The core communicates with the host processor over an interconnect (e.g., PCI, PCIX, RapidIO) and accesses data from a centralized memory. Data and control flow to the multiple kernels housed in the core is regulated by a finite state machine illustrated in Fig. 3.

- State 1 – During the initialization state, the FPGA is reset and the first batch of data samples x_i and set of all x are loaded onto the on-chip memory of the FPGA.

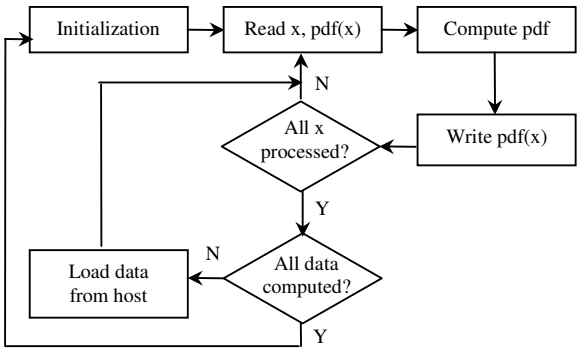

Figure 3. State transition diagram for a single core design

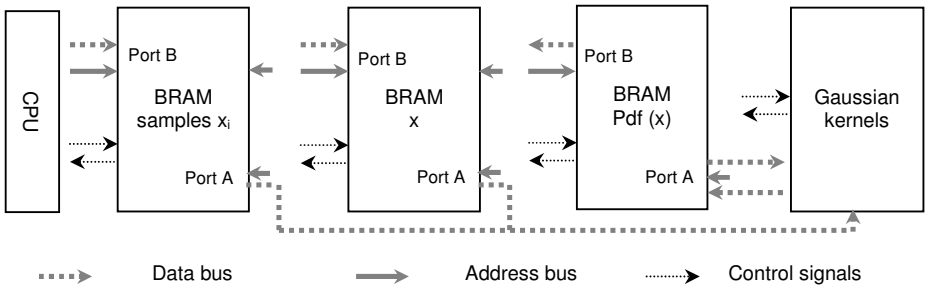

Figure 4. Single core design

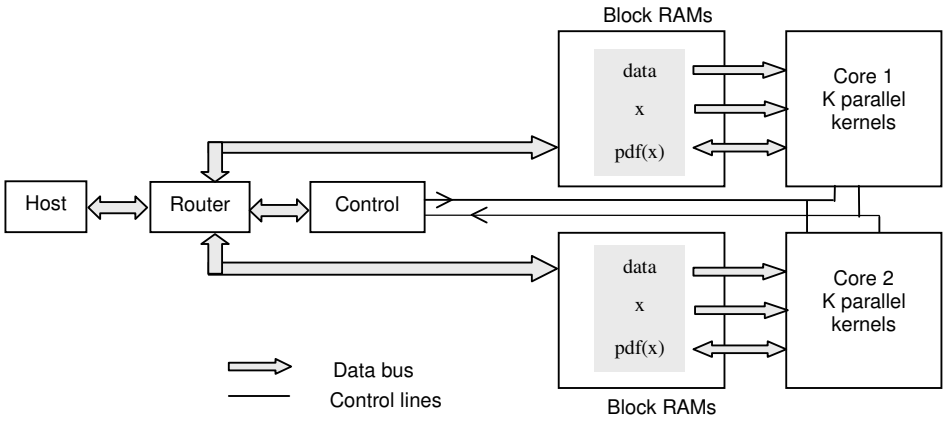

Figure 5. Dual core design

- State 2 – Set of x and PDF values computed in the previous iteration are sent to the kernels.
- State 3 – PDF values are computed on the dataset currently loaded in memory.
- State 4 – PDF values are updated in the memory. If the PDF has not been computed for all x, go to state 2. Else, the state machine checks if all data has been processed. If the condition is false, the next set of data samples are loaded from the host processor to the on-chip FPGA memory and the state machine goes to state 2. If the condition is true, the state machine goes to the initial state.

The basic blocks in the design are pictorially represented in Fig. 4. The design flow from the host end is illustrated in Fig. 6. A set of data samples are first loaded onto the FPGA and is signaled to start the computation. As the FPGA processes data the host processor polls for a "process complete" signal from the FPGA. The FPGA sends the "process complete" signal once the computation is over and the host sends in the next set of data samples until all data are processed.

2) Dual Core Design

The architectural details and the design flow of the dual core design are illustrated in Figs. 5 and 6 respectively. The state machine for a single core design and the corresponding core are replicated with changes made to the data flow at the host end. Since multiple cores share the interconnect, some form of arbitration is performed by the host while loading data onto the on-chip memory of the FPGA and polling for the "process complete" signal. Data is written to the first core and while the first core processes the data, the next set of data samples are loaded to the second core to process. The host then starts polling the first core for the "process complete" signal while the second core processes its data.

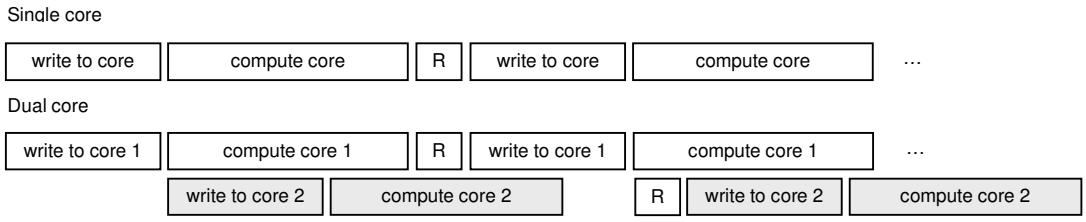

Figure 6. Design flow for a single (top) and dual (bottom) core design

It should be noted that there could be a situation where the host cannot poll one of the cores while it is loading data to the other core. The core that is not being polled sits idle during this contention period.

IV. RESULTS

The Nallatech boards with Virtex4LX100 FPGAs were utilized in this work. These Xilinx chips have dedicated arithmetic blocks called DSP48 slices that perform 18 bit multiplies and multiply-accumulates at a maximum speed of 500MHz. The boards communicate with the host processor over a PCIX interconnect. We use [32, 9] bit (Note: [a,b] – a is the total number of bits of which b bits are reserved for the fractional part) fixed point format for the data samples and a [18, 9] bit configuration for performing multiplications. The number of kernels (k) in a core and the number of data samples per transfer were set to 8 and 512 respectively taking into consideration the FPGA resources and additional resources used by the middleware. The basic criteria was to come up with a tight design where the resources (e.g. DSP48s, block RAMs, slices) are uniformly used up. These parameters would possibly change from one platform to another. Functional level simulation was performed in ActiveHDL and the overall system design was developed in DIMETalk, a design tool provided by Nallatech.

A. Device Utilization and Performance Analysis

The device utilization for a single core design is given in Table A. The number of multipliers (DSP48 units) scales linearly with the number of kernels in the core. Though only 3 RAMB16s were utilized as part of the core design, the middleware consumes a significant percentage of it. This is due to the fact that data are transferred in packets rather than word-by-word.

Table A. Device utilization for a single core

Device	Single core design – Utilization		
	Used	Available	% Used
DSP48s	8	96	8
External IOBs	93	768	12
RAMB16s	29	240	12
Slices	5418	49152	11

The utilization summary for the dual core design is given in Table B. The number of DSP48 units scale linearly with the number of parallel cores. As it can be seen, the parameter selection (k = 8 and data transfer size = 512 words) is quite optimum for both the designs and for the chosen platform as it results in a uniform consumption of resources.

Table B. Device utilization for a dual core

Device	Dual core design – Utilization		
	Used	Available	% Used
DSP48s	16	96	16
External IOBs	93	768	12
RAMB16s	37	240	15
Slices	8072	49152	16

The primary objective of this work is to obtain a speedup in execution by exploiting parallelism at the hardware level when compared to the sequential version run on a General Purpose Processor (GPP).

Speedup is defined as the ratio of execution time on a GPP (t_{GPP}) to the execution time on an FPGA (t_{FPGA}). The designs operated at an FPGA clock speed of 150MHz. The sequential version was run on a 3.2 GHz Intel Xeon processor. The speedups obtained for the two designs are given in Table C. The speedup does not double up between a single and dual core design because of interconnect contention (Refer section III).

$$speedup = \frac{t_{GPP}}{t_{FPGA}} \quad (6)$$

Table C. Speedup comparison

Design	t_{GPP} in secs	t_{FPGA} in secs	Speedup
Single core	0.578	0.0734	7.87
Dual core	0.578	0.0432	13.38

B. Data Verification

The computed PDF values were read and verified in MATLAB and error statistics in the solution was computed. A maximum error percentage of 3.8% calculated as per equation 7 was obtained.

$$\%\ Max\ Error = \frac{\max(abs(p(x)_{FPGA} - p(x)_{GPP}))}{p(x)_{GPP}} \times 100 \quad (7)$$

The error in the estimates obtained from a GPP and FPGA implementation is due to the Taylor series truncation of the Gaussian function rather than the fixed point effects. The resulting PDF shown in Fig. 7 was plotted in MATLAB for visualization purposes.

C. Scalabitlity Issues

Though the device utilization summarized in the previous discussion confirms that the design uniformly consumes the FPGA resources, there is still a significant slice portion left unused. Maximum utilization of the resources and memory bandwidth can be obtained by altering the design for a 2D PDF estimation. This computation, however, necessitates the design of a multi-core system which can be easily obtained by extending the dual core design.

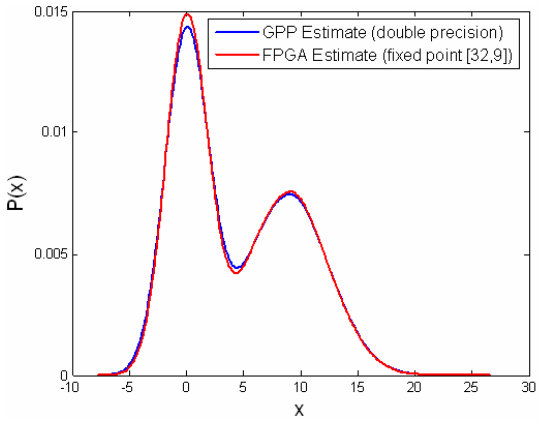

Figure 7. PDF estimates: GPP vs FPGA

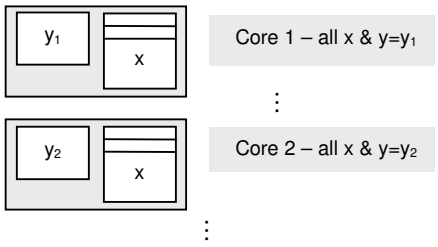

Figure 8. Proposed design architecture for estimating 2D PDFs

Fig. 8 shows a possible architecture for evaluating a 2D PDF. In this case, the PDF is evaluated at $x \times y$ points. Each core would perform computations on all x and a single y. The algorithm remains embarrassingly parallel and the same parallelization process (section III) is adopted.

V. Conclusion

Significant performance improvements in terms of speedups were obtained from porting the application to an FPGA as against a GPP implementation. The proposed design architecture was developed considering scalability issues. Key design parameters were identified that would help successfully port the design to different platforms. Precision effects were investigated and data verification along with error statistics suggested a sufficient fixed point configuration. The PDF estimation scales exponentially with increasing dimension and offer an apt case study for multi-FPGA environments. This work is a step forward towards proposing FPGAs as suitable platforms for solving a general class of problems.

Acknowledgment

We would like to thank Dr. Alan D. George and the High-performance Computing & Simulation Research Lab for providing us access to the necessary design tools and platforms used in this work.

References

[1] K.S. Hemmert, and K.D. Underwood, "An analysis of the double-precision floating-point FFT on FPGAs," IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 171–180, April 2005.

[2] E. Jamaro, and K. Wiatr, "Convolution operation implemented in FPGA structures for real-time image processing," IEEE Symposium on Image and Signal Processing and Analysis, pp. 417 – 422 , June 2001.

[3] G. Govindu, S. Choi, V. Prasanna, V. Daga, S. Gangadharpalli, and V. Sridhar, "A high-performance and energy-efficient architecture for floating-point based LU decomposition on FPGAs," IEEE Symposium on Parallel and Distributed Processing, pp. 149, April 2004.

[4] Steven M. Kay, Albert H. Nuttall, and Paul M. Baggenstoss, "Multidimensional Probability Density Function Approximations for Detection, Classification, and Model Order Selection," IEEE Transactions on Signal Processing, vol. 49(10), pp. 2240 – 2252, 2003.

[5] Fabrizio Lillo, Salvatore Basile, and Rosario N. Mantegna, "Comparative Genomics Study of Inverted Repeats in Bacteria," Bioinformatics, vol. 18(7), pp. 971 – 979, 2002.

[6] M. Llyas, "General Probability Density Function of Packet Service Times for Computer Networks," Electronics Letters, vol. 23(1), pp. 31 – 32, 1987.

[7] Robert R. Bliss, and Nikolaos Panigirtzoglou, "Testing the stability of implied probability density functions," Journal of Banking and Finance, pp. 381 – 422, March 2002.

[8] Martin Scheicher, and Ernst Glatzer, "Modelling the implied probability of stock market movements," Working Paper Series 212 European Central Bank, 2003.

[9] F. Pitie, A.C. Kokaram, and R. Dahyot. 2005, "N-Dimensional Probability Density Function Transfer and its Application to Color Transfer," Proc. Tenth International Conference on Computer Vision, pp. 1434 – 1439, October 2005.

[10] J. S. Kim, P. Mangalagiri, K. Irick, N. Vijaykrishnan, M. Kandemir, L. Deng, K. Sobti, C. Chakrabarti, N. Pitsianis, and X. Sun, "TANOR: A Tool for Accelerating N-body Simulations on Reconfigurable Platform,".

[11] I. Frohlich, A. Gabriel, D. Kirschner, J. Lehert, E. Lins, M. Petri, T. Perez-Cavalcanti, J. Ritman, D. Schafer, A. Toia, M. Traxler, and W.Kuehn, "Pattern Recognition in the HADES - Spectrometer: An Application of FPGA Technology in Nuclear and Particle Physics," Proc International Conference on Field-Programmable Technology (FPT), pp. 443 – 444, December2002.

[12] H. Schmit, and D. Thomas, "Hidden Markov modeling and fuzzy controllers in FPGAs," Proc Symposium on FPGAs for Custom Computing Machines, pp. 214 – 221, April 1995.

[13] T. VanCourt, and M. Herbordt, "Three Dimensional Template Correlation: Object Recognition in 3D Voxel Data," Proc. Computer Architecture for Machine Perception, pp. 153-158, 2005.

[14] David E. Culler, and Jaswinder Pal Singh, "Parallel computer architecture – A hardware / software approach," Morgan Kauffmann publishers, August 1999.