

Three-Dimensional Template Correlation: Object Recognition in 3D Voxel Data

Tom VanCourt *Student Member IEEE*, Yongfeng Gu, and Martin C. Herbordt *Member IEEE*

Abstract— Correlation is a standard technique for recognizing known patterns in two-dimensional grid (pixel) images. Its obvious importance has led to numerous hardware implementations and variations. Images captured directly onto 3D voxel grids are becoming more common, including those from confocal microscopy and medical imaging technologies. To our knowledge, no one has yet addressed correlation as a technique for recognizing 3D templates in such 3D voxel data. We find that this problem includes a number of issues: efficient three-axis rotation of a template with respect to 3D image, large volume of results from the correlation, and the possibility of a template matching an image multiple times. We briefly review techniques that have been used in 2D template matching, and examine analogies to a molecule interaction problem in computational chemistry, including its similarity to multispectral images. We report on a hardware accelerator for the 3D correlation problem, based on a commodity coprocessor board containing Field Programmable Logic Arrays (FPGAs). Because the convolution processor is built from reconfigurable logic, it can be adapted to non-linear scoring algorithms using complex data values at each voxel, and can be tailored to solve other problems such as anisotropic grid axes. We present initial performance results for the FPGA implementation, and note that accelerator performance is likely to grow roughly linearly with FPGA capacity, process improvements, and number of FPGAs.

Index Terms— Computer vision implementation, Coprocessor, Field programmable gate arrays, Object recognition, Application-specific architecture

I. INTRODUCTION

Recent image-capture technologies, including confocal microscopy, MRI, and medical tomography, collect 3D images in which 3D patterns may be sought. Unlike 3D objects reconstructed from 2D images, these 3D images are captured directly. Some of these imaging technologies create complex data, combining input from different data acquisition channels at each volume element (voxel). Two-dimensional correlations have long been a standard technique for finding known 2D template patterns in images digitized onto 2D pixel grids. This paper explores the unique problems of template matching in such 3D data sets, and presents hardware structures for accelerating the technique using a

reconfigurable, FPGA-based coprocessor.

The contributions of this paper relate to the practical aspects of hardware acceleration for 3D template matching. We present a computation pipeline for variations on 3D correlation, allowing nonlinear scoring functions using multiple data channels. We show efficient structures for 3D model rotation, and show how those structures can be adapted to non-cubical voxel grids or to coprocessors with memory bandwidth constraints. We describe a data collection filter that reduces result bandwidth requirements by orders of magnitude, while preserving the ability to find multiple instances of a 3D template within the image volume. We discuss scaling, in numbers of FPGAs and FPGA capacity, and report results of our initial implementations.

II. PREVIOUS WORK

Correlation is well-known as a staple of object recognition. Relatively recently, correlation techniques have become popular as a step in docking problems in computational chemistry. There, the goal is to find out where and how well a potential drug molecule interacts with a medically significant protein. Researchers treat each drug candidate as a 3D template and search the protein's 'image' for regions that match the drug [1], [2]. Collisions, where the drug candidate and protein occupy the same volume, are penalized. Good matching or 'docking' represents strong chemical interaction, indicating that the molecule may have desirable drug-like effects on that protein. This application does not address scaling of the template for finding different-sized instances of the pattern, because the grids for the two molecule models are created at the same pitch.

Two-dimensional correlation and convolution have been so important in image processing, that efficient structures for hardware acceleration have a long literature [3], [4]. Some of those authors have proposed actual structures computing 3D correlations. Others have presented structures for 2D correlations that can readily be extended to higher dimensions. Older literature proposed application-specific integrated circuits (ASICs) for such accelerators. More recent authors use FPGAs to create reconfigurable accelerators that can implement complicated, possibly nonlinear two-dimensional

Manuscript received November 25, 2004. This work was supported in part by the National Institutes of Health Award #RR020209-01.

Tom VanCourt, Yongfeng Gu, and Martin C. Herbordt are with the CAAD lab (<http://www.bu.edu/caadlab>) at Boston University, Boston MA 02215 USA. Email: {tvancour, maplegu, herbordt}@bu.edu.

filters [5], [6], [7].

Multispectral analysis in two dimensions, where each 2D pixel includes multiple data values from different inputs, also has a long literature [8]. This experience transfers directly to three dimensional images, for example signals generated by multiple fluorescent dyes in confocal microscopy [9]. The 3D docking problem has also used multiple values per voxel, representing physical structure, electrostatic charge, and other phenomena. Docking may involve oriented data, such as normal vectors in voxels representing the surface of the molecule. Techniques for handling oriented data are directly applicable to other data sets that include polarization or other vector quantities.

Object matching typically requires three-axis rotation of the voxel image. Special 3D memory structures have been proposed [10], designed to reduce conflicts when accessing multiple memory locations. Other authors have concentrated on converting 2D computation coordinates directly to memory locations in untransformed coordinates [11]. These 2D access transformations have straightforward extensions to three dimensions. Section IV shows how such transformations can be adapted for direct, rotated access to data sets sampled on non-cubical grids, without need for resampling.

III. COMPUTING WITH FPGA COPROCESSORS

Three-dimensional correlation is a good candidate for hardware acceleration for a variety of reasons. First, efficient computation structures are well understood [3], and can be customized to the specifics of each template matching problem. Second, the high parallelism, regular communication patterns, and predictable patterns of data access work well in optimized computation arrays. On-chip communication within an FPGA is especially fast and cheap, compared to communication in other kinds of parallel processing engines. This allows use of communication patterns, such as broadcasting, that are undesirable in other computing models [12]. Third, competing model representations and scoring functions exist, even within the class of models based on correlations. No single scoring computation is appropriate in

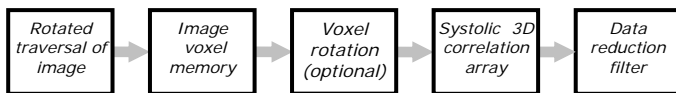


Figure 1: Computation pipeline for 3D correlation

all cases, so the flexibility of an FPGA coprocessor is helpful in creating computing structures customized to each different model.

Fig. 1 illustrates our computation pipeline, currently implemented on one of the FPGA chips. Host interface logic is omitted for clarity. The *Image memory* is a linear RAM that holds the voxel grid representing the 3D image in which the template pattern is sought, and has conventional structure. Each memory word represents a single voxel, but may contain multiple data elements that describe different physical

phenomena present in the model. The remainder of this discussion takes each other segment of the computation pipeline in order:

- 1) *Rotated traversal*. This creates the sequence of memory references and padding values that represent a three-axis rotation of one grid with respect to the other.
- 2) *Voxel rotation*. Some molecule models include oriented (vector) information. When the 3D molecule model undergoes a rigid rotation, the oriented voxel data must be rotated through the same angle.
- 3) *Systolic 3D correlation array*. This structure has two functions: it holds the template voxel values, and it uses an array of processing elements to perform over 1500 parallel score-accumulate operations, depending on the complexity of the scoring function.
- 4) *Data reduction*. This phase reduces the volume of data coming from the correlation array, to reduce the bandwidth requirements between the FPGA coprocessor and the host.

This paper first presents a 3D extension to known techniques for traversing a grid template in rotated order, instead of creating a rotated image and processing that. We discuss practical issues, including logic optimization, padding of the rotated image, and grids with non-cubical voxels. We present a generalized correlation pipeline with optional elements for rotation of oriented voxel data values, and note specific FPGA features that support the computation. Finally, we present an FPGA implementation of a post-processing filter for reducing the volume of data reported by the correlation, and note its relationship to the *Rotated traversal* logic and to the *Correlation array*.

IV. MODEL ROTATION: TRAVERSAL IN ROTATED ORDER

A 2D image need only be rotated around one axis to create a rotated form. An image of modest size, perhaps a square 100 pixels on a side, contains 10^4 pixels. There are 36 rotations at 10° intervals, so all of the rotated images can be stored using $\sim 3.6 \times 10^5$ pixels.

A 3D image with 100 pixels on each side contains 10^6 pixels. Angular resolution can be specified in terms of N , the number divisions into which the 360° circle is cut. Then the number of three-axis rotations grows $O(N^3)$ as the number of steps of angular resolution increases. There are over 12,000 three-axis rotations at intervals near 10° . Based on those assumptions, precomputed rotations would require storage for more than 10^{10} voxels. A better approach is to compute the rotated images as they are needed.

It has long been known that indexing a grid in rotated order is an effective alternative to creating a separate rotated image in a separate memory buffer [11]. We briefly review the technique, including efficient hardware implementation and logic for padding the rotated image.

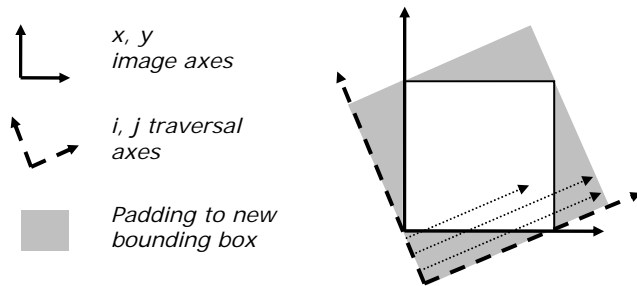


Figure 2: Indexing and padding in rotated order

Fig. 2 summarizes the approach for the 2D case. The image, plus any needed padding, is traversed in rotated coordinates. In three dimensions, the axis conversion is given by:

$$\begin{bmatrix} b_{ix} & b_{jx} & b_{kx} \\ b_{iy} & b_{jy} & b_{ky} \\ b_{iz} & b_{jz} & b_{kz} \end{bmatrix} \times \begin{bmatrix} i \\ j \\ k \end{bmatrix} + X_0 = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} b_{ix}i + b_{jx}j + b_{kx}k + x_0 \\ b_{iy}i + b_{jy}j + b_{ky}k + y_0 \\ b_{iz}i + b_{jz}j + b_{kz}k + z_0 \end{bmatrix} \quad (1)$$

After transformation, (x, y, z) image coordinates are mapped to linear memory addresses as usual. A few points are worth noting:

- Maximum ranges of traversal indices (i, j, k) and template indices (x, y, z) are known in advance. That information yields the precision needed for fixed-point computations across the whole (i, j, k) range to maintain $\pm 1/2$ unit accuracy in the (x, y, z) domain, using as few bits as necessary.
- The (i, j, k) indices may cover a wider range than the (x, y, z) values, because the image needs to be padded to the larger bounding box for the rotated image. Simple inequality tests on the (x, y, z) values from (1) determine whether the (i, j, k) index is outside of the (x, y, z) range, i.e. whether it represents padding or not.
- In our application, (i, j, k) indices are traversed sequentially. Standard strength reduction techniques convert all multiplications in (1) into additions, and those additions can be performed in parallel. This makes good use of the FPGA's logic resources, without claiming limited and potentially slow hardware multipliers.

This logic can be implemented using only modest amounts of FPGA resources. The highly predictable traversal order means that rotated address computation and range checking can be pipeline stages in the memory access, and need not have any effect on system throughput. Given the low amount of logic needed, rotation and padding are very nearly free.

Our implementation uses eighteen parameters (nine matrix coefficients, three offset values, and six range limits) to set up one rotation. Even if parameter loading is not allowed to overlap with operation of the address generator, this is a brief transfer of data at the start of correlation for any one rotation, since the voxel images in on-board and on-chip memory are reused. The rotation parameter values are computed off-line.

Three dimensional medical images are often captured with relatively fine resolution in one scanning plane, but coarser resolution along the axis between scanning planes. For example, confocal microscopy collects 3D images and has finer resolution limits along the horizontal x and y axes (to

~ 100 nm) than in the vertical z direction (to ~ 400 nm) [9]. In that case, the raw voxel values are non-cubical, rectangular prisms. Proper choice of coefficients in the linear transformation (1) represents this anisotropic scaling, or its inverse, in a natural way. Thus, this address generation scheme adapts to on-the-fly rescaling of non-uniform grids, without need for a separate buffer or resampling step.

This structure can also perform isotropic scaling, allowing the template to be tested at different sizes relative to the image. The general linear transformation in (1) also allows a mirror reflection. In the original chemistry application, this could conceivably represent a molecule's opposite stereoisomer. We observe this as a curiosity only, not as a meaningful optimization to the chemistry application. Shear transforms are also possible, but await practical application.

Changes to lengths of the unit vectors are also expected to be useful in systems where memory bandwidth throttles the computation array. In one implementation, the design duplicates the entire rotated-traversal logic and template memory. Assuming that k is the fastest-moving traversal index, each traversal unit is assigned a (b_{kx}, b_{ky}, b_{kz}) vector of length two, with staggered starting positions representing even and odd k values respectively. Each traversal unit would leapfrog the other at every access. Duplication gives twice the access rate of a single memory, with easy generalization to N -way replication for N -fold speedup. It may seem extravagant to replicate the entire RAM and addressing subsystem. The Wildstar-II Pro board has seven independently addressable off-chip RAMs, however, and one instance of this addressing logic requires about 0.5% of a Xilinx XC2VP70 FPGA's logic resources. Complete replication of the image memory subsystem can be an effective use of available resources.

It is worth noting that the correlation operation gives the same result whether the template is rotated relative to the image or vice versa. Our implementation creates an efficient computing structure by holding template voxel values in the processing elements of the convolution array. Changing the template orientation would require reloading the template coefficients in the convolution array. Instead, we hold the template fixed and traverse the 3D image in rotated order. This may be counter-intuitive, but rotated traversal means that neither the correlation array nor the image RAM needs to be reloaded between successive rotated orientations. The image is typically larger than the template, so image rotation requires somewhat more padding than if the template were rotated. We have found it effective to accept the cost of that extra padding in order to gain efficiency in the computation array and reduced host communication.

V. ROTATION OF VOXEL VALUES

Two-dimensional images traditionally consist of scalar brightness values. Their meaning is independent of orientation. Many properties in docking applications also have

rotation-independent meanings, including electrostatic charge and solid structure. Chemistry models may, however, describe molecules using vector values with inherent orientation. Examples include hydrogen bonding, which depends on the angle between the donor and acceptor [14], and effects due to ring orientation [15]. Two- and three-dimensional images may also include oriented data, such as motion vectors in diffusion tensor imaging [16] or polarization axes. Rotating an image with oriented voxel values requires not just a rigid rotation of voxels relative to each other, but also rotation of the voxel vector quantities themselves.

Rotation of voxel values fits conveniently into the computation pipeline in Fig. 1, when required by the application. In the general case, voxel values are tuples, possibly consisting of vector and scalar quantities. When there is more than one vector, each needs to be rotated separately. Voxel tuple values may include scalars as well as vectors, in which case the scalars do not need to be rotated. There may, however, one or more clock times of latency in the rotation logic. In that case, the scalar data needs to be delayed to keep in step with the rotated parts of the voxel values.

Rotation of voxel values is a linear transformation, which can be performed efficiently using an FPGA's block multipliers. Rotated traversal order uses a predictable sequence of (i, j, k) values, so strength reduction allows the transform's multiplications to be replaced by additions. The voxel contents themselves, however, are not assumed to have any such regularity in their sequences of vector values. Strength reduction is not possible for transformation of these vectors, so hardware multipliers are attractive implementation tools. Transformation of one three-component vector would require nine multipliers, assuming maximum parallelism. That is less than 3% of a Xilinx XC2VP70's multipliers, a very modest allocation of resources.

VI. CORRELATION ARRAY

A description of the systolic array for 3D correlation has been presented elsewhere [17]. The high-level structure of the array is based on the McWhirther-McCanny style of array [4], generalized to three dimensions and to application-dependent scoring functions. The important feature of this array is that it performs direct, not transform-based correlation, using massive fine-grained parallelism. The computation array stores the voxel values for the smaller of the 3D grids, generally taken to be the template. It accepts one molecule or padding voxel per clock cycle, and generates one correlation value per clock cycle. Although our original form of the systolic array handled very simple voxel descriptions, we have generalized the structure to handle any "correlation" score S of the form

$$S_{xyz} = \sum_{i,j,k} F(A_{x+i,y+j,z+k}, B_{ijk}), \quad \text{not just } F(s,t) = s \times t.$$

This allows nonlinear scoring functions that can not be handled using transform-based correlations, including

$F(s,t) = |s-t|$, opposition of surface normal vectors [15], and solid angle complementarity at each voxel. It also allows non-trivial data types for voxels, such as tuples of mixed value types possibly including vectors.

It is well known that direct correlation has a polynomial complexity of $O(N^6)$ operations for a cube of size N , and that correlation using Fourier transforms requires only $O(N^3 \log N)$ steps. For asymptotically large values of N , FFT-based correlation runs faster than direct summation. Comparisons of polynomial complexity are not meaningful for small values of N , however, since constant factors and low-order terms may dominate estimates of the number of computing steps. This is the case in our direct implementation of correlation. Direct summation offers a number of advantages, including

- massive parallelism, over 1700 processing elements in our initial implementation,
- no loop overhead because of hardwired control, and
- no rotation, indexing, or memory access cost, due to pipelined operation.

Together, these advantages give the FPGA's direct correlation a speed advantage up to 1000:1 over a 3GHz PC running FFT-based correlation, using standard 3D FFT software [18], for values of N that arise in our applications. Each additional channel in multi-spectral data would require an additional 3D FFT, and non-linear phenomena simply can not be evaluated using transforms; both factors tend to make direct convolution more attractive. Despite the theoretical disadvantages in algorithmic complexity, we expect the direct FPGA-based implementation to offer practical performance advantages through the visible future.

VII. DATA REDUCTION FILTERING

The cross-correlation of two 3D grids is itself a 3D grid, larger than either of the input grids. A major performance bottleneck in an FPGA coprocessor system is the system bus connecting the coprocessor to the host. Two-dimensional correlations generate $O(N^2)$ pixels in the correlation result, but 3D correlations generate $O(N^3)$ voxels, a significantly worse problem. It is highly desirable that transfers between the host and the coprocessor not limit performance, so we present a filter for reducing the volume of result data and relating it to the 3D image in its unrotated frame.

Cross-correlation of two 3D grids, of sizes (A_x, A_y, A_z) and (B_x, B_y, B_z) , gives a result of size $(A_x+B_x-1, A_y+B_y-1, A_z+B_z-1)$. In our initial implementation, the A grid was a cube 100 units on a side, up to $100\sqrt{3} = 173$ units on a side in rotated configurations. The B grid, the one held fixed, was 14 units on a side. The result, then, would be up to 186^3 or 6.4M voxels. If this volume of data had to be transferred to the host and processed for each rotation, the benefit of the FPGA accelerator would be largely negated. Instead, we chose a post-processing step for 'peak filtering', to select the highest correlation scores.

Our original application, like many others, admits the possibility of multiple different matches, all of which must be

reported. One complicating factor, however, is that many high

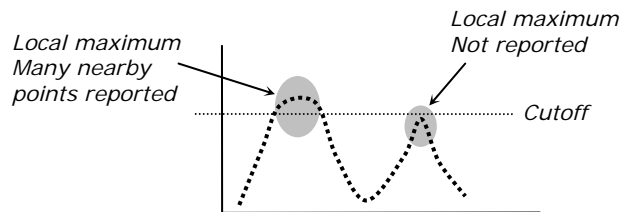


Figure 3: Local maxima—redundant reporting and omissions

scores tend to cluster around broad local maxima, as shown in Figure 3. Simple schemes would tend to report that one maximum redundantly, because of high-scoring neighbors, but not report other local maxima almost as high. We address multiple maxima by dividing the result grid into sub-blocks and collecting the highest score reported in each sub-block, as shown in Fig. 4. Our implementation uses sub-block numbers be based on the (x_c, y_c, z_c) address of the score within the correlation result, e.g. $(x_c/8, y_c/8, z_c/8)$. This reduces the volume of result data by a factor of $(1/8)^3$, or $1/512$. Instead of 6.4M result values, the host handles at most 12.6K. If sub-blocks were cubes 16 units along each edge, the host would process no more than 1.6K scores for each rotation. We double-buffer the RAM used to collect maxima, so uploading and processing these few selected values to the host normally

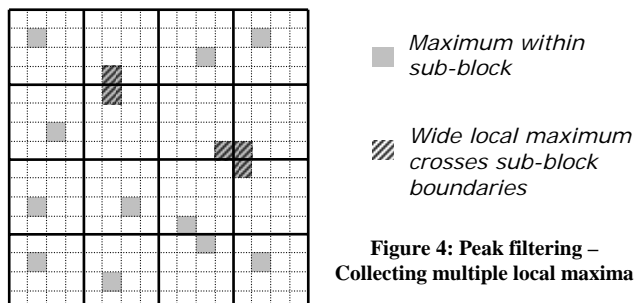


Figure 4: Peak filtering – Collecting multiple local maxima

overlaps computation of results at the next three-axis rotation.

This scheme may miss multiple local maxima that all occur within one sub-block. The number of omitted maxima can be reduced by increasing the number of sub-blocks, however. Fig. 4 also shows examples where this scheme may also report individual maxima redundantly, in cases of broad peaks crossing sub-block boundaries. Such situations can be resolved by later processing on the host processor. Despite these potential problems, collecting one local maximum per sub-block offers the advantages of simple implementation, handling of multiple maxima, data collection at the computation clock rate, and modest hardware requirements.

Inputs to the result collector are the current score, the sub-block number, and a tag value. When enabled, the collector compares the new score to the previous best for that sub-block. If the collector’s record for that sub-block is cleared or if the new score is better than the old one, the new score and its tag value are saved for that sub-block. The tag value encodes the exact (x, y, z) position at which that maximum correlation value was recorded.

For simplicity, our current implementation finds a sub-block number based on indices within the 3D convolution result after rotation. Future implementations will exploit tag values and block numbers based on unrotated (x, y, z) coordinates generated by the traversal logic. That means that each voxel in the unrotated image can be handled in the same sub-block of the peak filter, no matter how the image is rotated for correlation, so sub-block values for different rotations can be compared more easily. Some care needs to be taken in handling the padding values for rotated images, delaying (x, y, z) values to match computation delays, and in handling the fact that the correlation’s output grid is larger than the original image. The addressing logic already generates a padding signal, however, and that simplifies some of the book-keeping.

VIII. IMPLEMENTATION AND PERFORMANCE RESULTS

We have implemented the convolution hardware accelerator on a Xilinx XC2VP70, mounted on a Wildstar II Pro board. The implementation includes the full pipeline shown in Fig. 1, minus voxel rotation, plus host interface logic. Our initial implementation uses a two-bit voxel representation and 10-bit sums, implemented with saturating arithmetic. This implementation handles 3D models up to $50 \times 50 \times 50$ voxels in any rotation, limited by the amount of on-chip RAM. It also handles templates up to $12 \times 12 \times 12$, limited by the amount of logic available. Each template voxel is stored in a separate processing element, allowing up to 12^3 or 1728-way parallelism at each clock cycle.

The current implementation runs at 95 MHz, processing one voxel per cycle. A complete 3D correlation of maximum size, not rotated, can be completed in about 2.4 ms. On the average, padding around a rotated model adds about 66% to the model’s volume, so the average maximum-size correlation takes 4.0 ms when all rotations are considered. A standard PC-based implementation, using the 3D Fourier transform technique reported in the literature [1] and a standard implementation of the 3D transform [18], was measured at 4250 ms on a 3GHz Xeon processor, a speedup over 1000 \times .

We look forward future improvements in several areas:

- This implementation is limited by the on-chip RAM for storing the 3D image in which the template is sought. Off-chip RAM is available on the Wildstar board, and would allow significantly larger images.
- The systolic array used for correlation has an essentially linear structure, and can be extended as far as resources allow. We expect near-linear increase in number of processing elements and in template volume as logic resources are increased, either using larger chips (such as Xilinx’s XC2VP100) or connecting the two FPGAs on the Wildstar board.
- Larger images and templates can be handled if they are broken into smaller pieces, scored piecewise, and the pieces summed. This is an area of active development.

- More complex voxel representations and voxel scoring functions can be accommodated.

As the functions and data types increase in complexity, however, each processing element requires a larger portion of the FPGA's logic resources, so fewer processing elements would be possible per FPGA. More complex processing elements and reduced numbers of them will affect system performance. This will require detailed analysis for meaningful comparisons to the performance of PC-based implementations.

IX. SUMMARY AND CONCLUSIONS

Current scientific and medical instruments create volumetric data sets, amenable to automated searching for 3D features of interest. We present a reconfigurable, FPGA-based accelerator architecture for finding instances of 3D template objects in such data sets. These accelerators address many 3D searching issues, including:

- 1) Complex data values at each volume element, including tuples and oriented (vector) values,
- 2) Customized, possibly nonlinear scoring functions for voxel comparison,
- 3) Three-axis template rotation,
- 4) Correction of non-cubical sampling grids, and
- 5) Automated searching for multiple matches to the template.

Our implementation uses a commodity PCI coprocessor board. It is implemented in a standard high-level hardware description language, so it should port readily to new FPGAs and boards as they become available.

This structure was originally developed for a pattern-matching application in rational drug design, but is well suited to template matching in the 3D data produced by current scientific and medical instruments. The convolution array is essentially a one-dimensional structure, so it should give near-linear performance improvement as additional computing resources are added. Our parameterized design can easily increase the size of the correlation array when larger FPGAs are used, giving a roughly linear increase in performance as computing resources are added. The basic computing structure also scales well to multi-FPGA systems.

REFERENCES

- [1] Ephraim Katchalski-Katzir, Isaac Shariv, Miriam Eisenstein, Asher A. Friesem, Claude Afalo, and Ilya A. Vakser. "Molecular surface recognition: Determination of geometric fit between proteins and their ligands by correlation techniques." *Proc. Natl. Acad. Sci.* 89:2195-2199, 1992
- [2] Rong Chen and Zhiping Wen. "Docking Unbound Proteins using Shape Complementarity, Desolvation, and Electrostatics" *Proteins: Structure, Function and Genetics* 47:281-294, 2002
- [3] H. T. Kung and R. L. Picard. "One-dimensional Systolic Arrays for Multidimensional Convolution and Resampling", in *VLSI for Pattern Recognition and Image Processing* (ed. King-sun Fu). Springer-Verlag, 1984
- [4] E. Swartzlander. *Systolic Signal Processing Systems*. Marcel Dekker, Inc., 1987
- [5] Nalini K. Ratha, Anil K. Jain, and Siane T. Rover. "Convolution on Splash 2". *Proc. IEEE Symposium on FPGAs for Custom Computing Machines* (1995), pp.204-213
- [6] Nalini K. Ratha and Anil K. Jain. "FPGA-based Computing in Computer Vision." *Proc. Computer Architectures for Machine Perception* 1997
- [7] Bruce Draper, Walid Najjar, Wim Böhm, Jef Hammes, Bob Rinker, Charlie Ross, Monica Chawathe, and José Bins. "Compiling and Optimizing Image Processing Algorithms for FPGAs." *Proc. CAMP 2000* p.222-231
- [8] Robert A.Schowengerdt. *Techniques for Image Processing and Classification in Remote Sensing*. Academic Press, Inc., 1983
- [9] C. J. R. Sheppard and D. M. Shotton. *Confocal Laser Scanning Microscopy*. Bios Scientific Publishers, 1997
- [10] Arie Kaufman. "Memory Organization for a Cubic Frame Buffer". *EUROGRAPHICS '86: Proceedings of the European Computer Graphics Conference and Exhibition*
- [11] W. B. Baringer, B.C.Richards, and R. W. Brodersen. "A VLSI Implementation of PPPE for Real-Time Image Processing in Radon Space - Work in Progress". *Computer Architecture for Pattern Analysis and Machine Intelligence (CAPAMI) 1987*, pp. 88-93
- [12] Sanjay Ranka and Sartaj Sahni. "Convolution on Mesh Connected Multicomputers." *IEEE Trans. On Pattern Analysis and Machine Intelligence* 12(3)315-318. 1990
- [13] Xilinx, Inc. (2004) <http://www.xilinx.com/products/tables/fpga.htm> (URL verified 11/16/04)
- [14] Matthew D. Eldridge, Christopher W. Murray, Timothy R. Auton, Gaia V. Paolini, and Roger P. Mee. (1997). "Empirical scoring functions: I. The development of a fast empirical scoring function to estimate the binding affinity of ligands in receptor complexes." *Journal of Computer-Aided Molecular Design* 11:425-445, 1997
- [15] Inbal Halperin, Buyong Ma, Haim Wolfson, and Ruth Nussinov. Principles of Docking: An Overview of Search Algorithms and a Guide to Scoring Functions. *Proteins: Structure, Function and Genetics* 47:409-443, 2002
- [16] Denis Le Bihan, Jean-François Mangin, Cyril Poupon, Chris A. Clark, Sabina Pappata, Nicolas Molko, and Hughes Chabriat. "Diffusion Tensor Imaging: Concepts and Applications." *Journal of Magnetic Resonance Imaging* 13:534-546. 2001.
- [17] Tom VanCourt, Yonfgeng Gu, and Martin Herboldt. "FPGA Acceleration of Rigid Molecule Interactions." *Proc. Field Programmable Logic and Applications FPL 2004*, pp. 863-867.
- [18] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992