Hindawi Publishing Corporation
EURASIP Journal on Applied Signal Processing
Volume 2006, Article ID 97950, Pages 1–10
DOI 10.1155/ASP/2006/97950

Rigid Molecule Docking: FPGA Reconfiguration for Alternative Force Laws

Tom VanCourt, Yongfeng Gu, Vikas Mundada, and Martin Herbordt

Department of Electrical and Computer Engineering, Boston University, Boston, MA 02215, USA

Received 1 May 2005; Revised 7 September 2005; Accepted 1 December 2005

Molecular docking is one of the primary computational methods used by pharmaceutical companies to try to reduce the cost of drug discovery. A common docking technique, used for low-resolution screening or as an intermediate step, performs a three-dimensional correlation between two molecules to test for favorable interactions between them. We extend our previous work on FPGA-based docking accelerators, using reconfigurability for customization of the physical laws and geometric models that describe molecule interaction. Our approach, based on direct summation, allows straightforward combination of multiple forces and enables nonlinear force models; the latter, in particular, are incompatible with the transform-based techniques typically used. Our approach has the further advantage of supporting spatially oriented values in molecule models, as well as the detection of multiple positions representing favorable interactions. We report performance measurements on several different models of chemical behavior and show speedups of from 130× to 1100× over a PC.

1. INTRODUCTION

Noncovalent bonding between molecules is basic to the processes of life and to the effectiveness of pharmaceuticals. Chemical experiments are not always practical for measuring binding strength, and may be prohibitively expensive for screening 100,000 or more drug candidates against one molecule of medical importance. Instead, a number of computational approaches have been developed. The most precise computational models are based on quantum mechanics, or on approximations such as density functional theory. These techniques are computationally exorbitant, however, and infeasible for answering the first question: at what approximate offsets and orientations could the molecules possibly interact at all?

Less costly techniques are used for initial estimates of the *docked pose*, the relative offset and rotation that give the strongest interaction. Although most large biomolecules (*substrates*) and small-molecule drug candidates (*ligands*) can flex or rotate around chemical bonds, many applications [1–5] assume rigid structure as a simplifying approximation. This still allows the modeling of many different rules or *force laws* governing interaction between molecules, including electrostatic, geometric, atomic contact potential, solvent effect, and many others.

Since its introduction, 3D correlation [3] has become a standard technique for determining the best fit between digitized representations of rigid molecule approximations. The technique is based on 3D voxel grids representing the substrate and ligand. It uses correlation to detect strong similarity between the 3D structure of the ligand and the 3D shape of the active "pocket" within the substrate molecule. Spatial correlation determines only the relative offset at which docking occurs, so it must be repeated at many three-axis rotations, over 10^4 of them for 10° sampling intervals. The standard PC implementations use transform techniques that reduce the polynomial complexity of the correlation, but still take hours of computation to test each drug candidate against a biomolecule—and must be repeated for each of the 10^3–10^5 drug candidates in a pharmacological screen.

Contributions of this report are based on the reconfigurability of FPGAs. We demonstrate an entire family of docking accelerators, each specific to a different force law, but based on a common family architecture. Chemistry models are generalized to arbitrary tuple data types, allowing combinations of phenomena and even vector-valued phenomena in the force laws. Scoring functions are also generalized to allow nonlinear expressions, which are impossible for transform-based correlation techniques. System performance measurements are presented, based on a Xilinx Virtex-II Pro implementation. We examine the tradeoff of complexity in the chemistry model versus accelerator performance, showing how reconfigurability makes the best use of the FPGA resources for each specific calculation.

The significance of the hundred-fold to thousand-fold speed increases at modest cost reported here is that this could enable new kinds of screening: more drug candidates per dollar of computing budget, increased accuracy in chemistry models yielding drug leads of better quality, testing of alternative conformations in ligands and substrates, and testing of more substrates for medical effect or side effects.

2. BACKGROUND AND METHOD

Docking is a central problem in computational biochemistry with a long history and a vast literature; some recent surveys are [6–8]. 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 [1–4]. Collisions, where the drug candidate and protein would 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.

Standard PC-based applications [2, 4] for correlation-based docking use 3D Fourier transforms for performing correlation, sometimes complex transforms [3]. Experiments show that, on a serial processor, the improved asymptotic complexity of transform-based correlation gives better performance for problems of common sizes. The applications are found to have two important features in common, however: initial problem statements in terms of simple data types, and simple scoring functions. Recognizing the highly regular structure of the computation, correlation by direct summation can be implemented using hundreds or thousands of parallel processing elements in today's FPGAs. The FPGA implementation also eliminates the overhead of the inner loops (because of dedicated control hardware) and load/store costs (because of pipelining). We show here that when using direct summation, FPGA-based accelerators offer dramatic acceleration of the correlation, for typical problems.

Our family of accelerators does the following: it accelerates the correlation form of docking, integrates 3D rotation with correlation, handles rotation of vector-valued voxels, and uses hardware filtering to reduce the volume of data transferred from the accelerator to the host. We also observe that correlation hardware typically specifies a particular size of computation array as one of the design inputs. As will be seen in Section 6, exact array size is not a constraint on these computation arrays. Our approach also differs in using the largest array size possible, sized according to the FPGA resources available and the complexity of a given computation.

3. RELATED WORK

Correlation is well known as a staple of object recognition. Two-dimensional correlation and convolution have been so important in image processing that efficient structures for hardware acceleration have a long literature [9, 10]. Some of those authors presented 3D computation structures, or 2D structures that can readily be extended to higher dimensions. Older literature proposed application-specific integrated circuits (ASICs) for correlations. More recent authors use FPGAs to create accelerators that can implement complicated, possibly nonlinear two-dimensional filters [11, 12].

Object matching typically requires rotation of the voxel image. Special 3D memory structures have been proposed [13], 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 [1]. These 2D access transformations have straightforward extensions to three dimensions.

Docking computations have long been carried out on serial computers. Lately there have also been parallel implementations [14, 15], including one on a SIMD processor [16]. We believe that this work is the first study of using FPGAs to accelerate this application. Please note that docking has little resemblance—either in problem being addressed or in method used—to many of bio-related applications that have previously been implemented on FPGAs, including homology modeling, microarray data analysis, reaction modeling, mass spectrometry, sequence assembly, phylogenetic analysis, and sequence analysis. Molecular dynamics has lately also been implemented on FPGAs and is sometimes used as a late stage of complex docking applications, but also is unrelated to current work in that it addresses a different part of the docking problem and uses different methods.

4. THE TARGET SYSTEM

When formulating the system, our target was a generic high-end 2004-era FPGA. One such chip, the Xilinx Virtex-II Pro XC2VP70 has 74,448 configurable logic cells, 328 hardwired multipliers, 328 block RAMs totaling 5.9 Mbits, and up to 992 pins for memory interfaces and I/O connections. We also targeted a generic PCI-based FPGA development board, with host interface and ancillary memory. We implemented our designs using a standard tool flow.

When prototyping our system for proof-of-concept and timing experiments, we used a commercially available FPGA accelerator board, the Annapolis Microsystems Wildstar-II. This board plugs into the PCI bus of a standard PC, as shown in Figure 1. Our Wildstar board contains two XC2VP70-5 FPGAs, only one of which is used in the current study.

The docking application is partitioned so that the accelerator performs the correlation operation and summarizes the results. The host PC is responsible for loading the molecule models from disk (e.g., in PDB format), downloading models to the accelerator board, and setting accelerator control parameters. It is also responsible for collecting correlation results computed at each three-axis rotation, and aggregating results from all rotations into a final result.

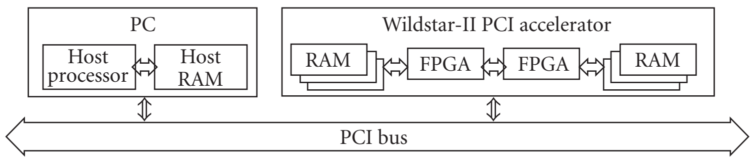

Figure 1: PC and FPGA accelerator configuration.

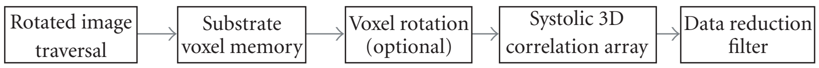

Figure 2: FPGA computation pipeline for 3D correlation.

5. THE CONFIGURABLE ACCELERATOR: ALGORITHMS AND DESIGN

5.1. Configurable pipeline overview

Figure 2 illustrates the basic structure of the accelerator's computation pipeline. In left-to-right order, system components are the following.

(i) Rotated image traversal. The two molecules must be tested against each other in rotated orientations that cover the range of possibilities in adequate detail. The standard approach would be to take the original molecule image, create a rotated version of it in a new buffer, and process the rotated image. Instead, this structure eliminates rotation as a separate step by accessing the image in a rotated indexing sequence, without the need for a separate processing step or memory allocation.

(ii) Substrate voxel memory. The substrate molecule's voxels are stored in the FPGA's on-chip memory. This is a RAM of conventional structure, with indexing logic for the three-dimensional grid containing the molecule image. This holds the larger of the molecule grids, typically up to $100 \times 100 \times 100$, depending on resource availability.

(iii) Optional voxel rotation. Some models of chemical interaction involve spatially oriented vector values. Examples include normal vectors for checking that molecule surfaces are roughly parallel, or values representing the directional specificity of hydrogen bond donors and receptors.

(iv) Systolic 3D correlation array. This consists of a three-dimensional array or processing elements (PEs) padded with synchronous FIFOs for matching the sizes of input and result data. The array serves two purposes: each processing element store one voxel value for one of the molecules, and it performs the scoring and summation arithmetic. Because FPGA logic can hold fewer voxels than the block RAMs can, this array stores the smaller of the two molecule grids.

(v) Data reduction filter. The correlation result may consists of 10^6 or more individual scores, but only the highest-scoring positions in the correlation result are of interest. Since the docking calculations are only approximate, it is realized that the correct docked pose may not be assigned the highest of all scores. Instead, some set of high-scoring poses are recorded, with the expectation that the docked pose will be among those with the highest scores. This filter collects a set of locally maximal scores, providing multiple results without requiring that the host process the entire correlation result.

The remainder of this section discusses each component and its configurable features in more detail. The next section discusses operation of the array as a whole.

5.2. Rotated image traversal

Template matching applications based on two-dimensional correlation often precompute images at fixed rotations, then use the rotated images to look for matches of rotated objects. When rotations are sampled at 10° intervals, that technique requires storage of 36 rotated images. In general, that implies $O(N)$ images when the 360° circle is divided into N parts. There are $O(N^3)$ three-axis rotations, however. Equal 10° intervals in latitude, longitude, and roll angles give around 23,000 rotations. Icosahedral decomposition gives the same angular resolution with fewer samples, but still requires about 13,000 rotations. Each rotated image is typically around 100^3 voxels, plus padding, so precomputation and transfer of every rotated image would require processing and transferring over 10^{10} voxels. This is clearly impractical.

A straightforward implementation might rotate the smaller molecule as a separate step, then perform the correlation operation on the rotated image. In this implementation, however, the smaller molecule is stored in the limited resources of the computation array, where it is undesirable to waste space on padding. It is also undesirable to reload the voxel values at the start of each correlation operation.

Instead, this implementation rotates the larger of the molecules by accessing the voxel memory in rotated order, as suggested in Figure 3. Since the rotation and padding logic is part of the address computation pipeline that fetches voxels from the molecule memory, it does not affect throughput rates at all. Logic for rotating the fetch order consumes very little of the FPGA resources, so it has modest logic cost as well.

A linear transformation converts the (i, j, k) traversal indices into (x, y, z) subscripts for the 3D array of voxels. The transform coefficients represent the three-axis rotation of

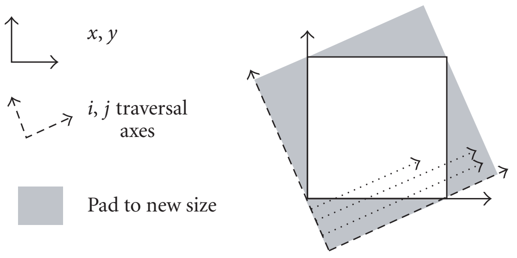

FIGURE 3: Indexing and padding in rotated order.

the molecule, and an initial offset keeps the indexing vectors within the array of voxels. After rotation and offset, the (x, y, z) values are mapped from 3D array subscripts to linear memory addresses as usual. The logic implementation incorporates the following observations.

(i) Maximum ranges of traversal indices (i, j, k) and voxel 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 the minimal number of bits.

(ii) The (i, j, k) indices may cover a wider range than the (x, y, z) values, because the molecule needs to be padded to the larger bounding box of the rotated image. Simple inequality tests on the (x, y, z) values from the following linear transformation of array indices determine whether the (i, j, k) index is outside of the (x, y, z) range, that is, whether it represents padding or not.

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

(iii) 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. Regular traversal order also allows the time for address computation to be concealed by pipelining, so index transformation has little or no impact on performance.

(iv) The B rotation matrix and X_0 offset vector are computed in the host and downloaded to the accelerator now for each correlation at a new three-axis rotation. As a result, the accelerator itself need not worry about how the coefficients are computed.

The result is that, for each correlation in a rotated orientation, 18 index rotation parameters need to be set by the host, and possibly a few more for voxel rotation (Section 5.4). This is a significant improvement over reloading a rotated image of 10^6 voxels or more. The molecule images are loaded once and reused across all rotations. As a result, the load time, when amortized over 10^4 or more correlations at different three-axis rotations, is negligible.

Configurability

This logic depends on the data types of the voxel values, if only indirectly. When large voxel values require that FPGA RAM be configured into wider words but fewer of them (i.e., fewer address bits), the indexing coefficients need fewer bits of precision to maintain $\pm 1/2$ unit address accuracy. Since coefficient widths are deduced from memory sizes using closed-form expressions, this is a minor concern.

5.3. *Substrate voxel memory*

The memory for holding the substrate molecule's voxel values is a linear RAM of conventional structure. Addressing logic converts 3D (x, y, z) index values into linear addresses. For efficient implementation, it is assumed that the 3D memory has fixed upper limits in each axis, and molecules may be of any size (in each axis) up to that size. When the fixed upper limits are of the form 2^J (for a positive integer J), index to address mapping is just concatenation of the three index values to form the linear address.

Configurability

Word width is the number of bits needed to hold one voxel value. As a result, the fixed amount of RAM available on the FPGA may need to be configured into fewer words for larger voxel values, or more words for narrower values. Since the array is currently implemented as a cube, and since it should have edge dimension 2^J for efficiency reasons described above, the total number of words is always 2^{3J}. Since FPGA RAMs offer different tradeoffs of number of words versus word width, the current implementations use 128^3 or 64^3, 2 M or 256 K values according to the number of bits per voxel.

The second dependency on application details comes from the implementation of the rotated addressing logic. When the address computation unit determines that the rotated image needs to be padded, an empty voxel value is substituted for RAM output. The bit pattern representing an empty voxel is, at least in principle, different for each specific application.

5.4. *Optional voxel rotation*

Simple force laws deal only with a molecule's interior and surface regions. If the molecule as a whole is subject to a rigid rotation, the positions but not the meanings of the interior and surface regions are unchanged. Many other force laws, such as those involving electrostatic field strength or shape complementarity, also involve scalar values with meanings unchanged by rotation. Force laws may, however, describe molecules using vector values with inherent spatial orientation. Examples include hydrogen bonding, which depends on the angle between the donor and acceptor [7], and effects due to ring orientation [11]. Rotating a grid containing oriented voxel values requires not just a rigid rotation of voxels relative to each other, but also rotation of the voxel vector quantities themselves.

Configurability

Rotation of voxel values fits conveniently into the computation pipeline in Figure 2, when required by the application. The simplest cases do not have oriented voxel data. In that case, the voxel rotation unit is present, for architectural consistency, but passes voxel input to output unchanged—an identity operation. If the voxel value includes one or more oriented values, they must be transformed. A linear transformation on one (x, y, z) vector would require nine multipliers, assuming maximum parallelism, that is, less than 3% of a Xilinx XC2VP70's multipliers, a modest allocation of resources. If azimuth and elevation angles suffice to describe an oriented phenomenon, rotation requires even fewer resources. When more than one cycle is needed for rotation, delay matching may be required for the unrotated parts of the voxel.

5.5. Systolic 3D correlation array

The high-level structure of the correlation array is based on the McWhirther-McCanny structure [10], generalized to three dimensions (shown in Figures 4 and 5), and to scoring functions that are different for each force law. The important feature of this array is that it performs direct, not transform-based correlation, using massive fine-grained parallelism. Voxels of molecule B, the ligand, are stored in individual cells of Figure 4's *1D correlation* arrays. Substrate molecule A is stored in on-chip RAM. Molecule A is rotated on the fly and padded as described above, then presented to the correlation array at one voxel per cycle. The array generates one correlation value per clock cycle. FIFOs extend the computation array to the size of the correlation result. We have generalized Figure 4's 1D correlation to handle not just the standard sum-of-products correlation, but any "correlation" score S of the form shown in the following equation:

$$\mathbf{S}_{xyz} = \sum\nolimits_{i,j,k} \mathbf{F}(\mathbf{a}_{x+i,y+j,z+k}, \mathbf{b}_{ijk}) \quad \text{(generalized correlation).} \tag{2}$$

Configurability

Three parts of this array are configurable. The most important configurable element is the basic computation cell in the 1D correlation, $F(a, b)$. Each accelerator in this application family uses a different function F, customized to represent the force laws in the chemistry model of interest. Because F may be nonlinear, this array can handle functions like $F(a, b) = |a - b|$ or solid angle complementarity that are impossible for transform-based correlation. The array size must be configured as part of the resource tradeoff: elaborate

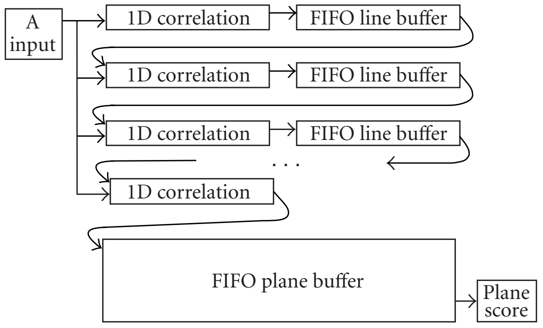

Figure 4: 2D correlation arrays built from 1D arrays and RAM-based FIFOs.

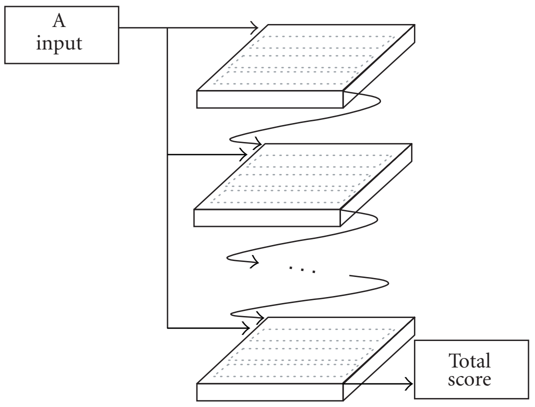

Figure 5: 3D correlation array built from stacked planes.

scoring functions require more logic resources, so fewer instances of them can be implemented in the fixed resources of any particular FPGA.

The second configurable feature is the voxel data type. This configures the communication pathways between Figure 2's *substrate voxel memory* and the correlation array. It also defines the voxel values stored in each of the scoring cells within Figure 4's *1D correlation* block. Voxel values use tuple data types to represent multiple phenomena, as in the model that handles both van der Waals collisions and surface interactions [3]. Part of the voxel tuple can also represent an oriented value, such as a surface normal vector [17]. Some models (e.g., [2]) use different representations for the voxels in the two molecules. Since the substrate and ligand voxels are stored in different parts of the computation structure, it is easy to use two different voxel data types, possibly with different bit widths, to represent the two molecules.

The third configurable feature is the score data type, used to set the sizes of the communication paths and synchronous, RAM-based FIFOs in Figures 4 and 5. We currently use a fixed-point value for scoring, with positive and negative overflow detection. This provides the same net effect as saturated arithmetic, but with somewhat less logic.

Cross-correlation of the A and B molecules of sizes (A_x, A_y, A_z) and (B_x, B_y, B_z) gives a result of size $(A_x + B_x - 1, A_y + B_y - 1, A_z + B_z - 1)$. This means that the stream of data from the substrate voxel memory must be padded in the x and y dimensions (z padding is not needed) up to the size of the result array. The rotated image traversal logic already injects the padding shown in Figure 3, so additional padding for size matching is a matter of parameter values rather than new logic.

5.6. Data reduction filter

Cross-correlation of two 3D grids gives a result of size $(A_x + B_x - 1, A_y + B_y - 1, A_z + B_z - 1)$. In one form of our initial implementation, the A grid is a cube 100 units on a side, up to $100 \times \sqrt{3} = 173$ units on a side after worst-case rotation and padding. The B grid, the one held fixed, is 14 units on a side. The result, then, may be up to $(173 + 14)^3$ or 6.4 M 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 implement a post-processing step for "peak filtering," to select the highest correlation scores. As noted above, multiple maxima must be reported, because of mathematical approximations and because there may be more than one interaction site on the protein.

One complicating factor is that groups of high scores often cluster around a broad local maximum [2]. Simple schemes tend to report that one maximum redundantly, because of the many high-scoring neighbors, but might not report other local maxima. We address multiple maxima by dividing the result grid into subblocks and collecting the highest score reported in each subblock, as shown in Figure 6. Our implementation uses subblock numbers based on the most significant bits of the (x_c, y_c, z_c) address of the score within the correlation result, for example, $(x_c/8, y_c/8, z_c/8)$. This reduces the volume of result data by a factor of $(1/8)^3$, or 1/512. Instead of 6.4 M result values, the host handles at most 12.6 K. If subblocks were cubes 16 units along each edge, the host would process no more than 1.6 K 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 overlap computation of results at the next rotation.

The subblock scheme may miss multiple local maxima that all occur within one subblock. The number of omitted maxima can be reduced by increasing the number of subblocks, however. Figure 6 also shows examples where this scheme may report individual maxima redundantly, in cases of broad peaks that cross subblock boundaries. Such situations can be detected by later processing on the host processor. Despite these potential problems, collecting one local maximum per subblock offers the advantages of simple implementation, handling of multiple maxima, data collection at the computation clock rate, and modest hardware requirements. In particular, we note dramatic reduction in the RAM resources committed to result storage even with double buffering of storage for the local maxima.

Inputs to the result collector are the current score, the subblock number, and a tag value that represents the exact (x, y, z) address of the score. When enabled (i.e., when the (x, y, z) index does not represent padding), the collector compares the new score to the previous best for that subblock. If the collector's record for that subblock is cleared or if the new score is better than the old one, the new score and its tag value are saved for that subblock.

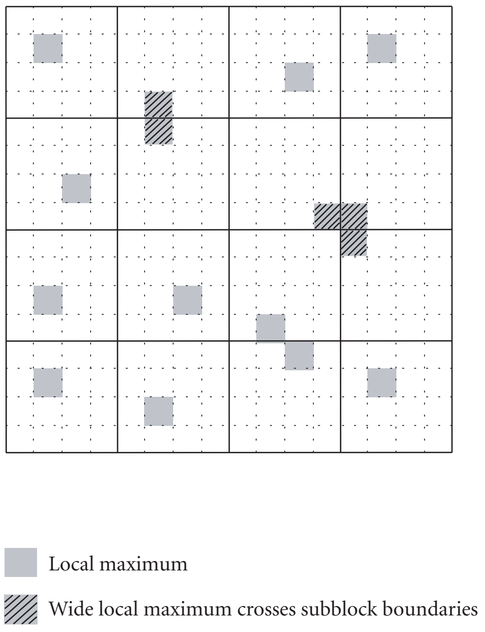

FIGURE 6: Peak filtering, collecting multiple local maxima.

Configurability

Data elements stored in the array are pairs: a score and a tag value indicating the exact position within the subblock at which the score occurred. Both the filter RAM length and the tag size depend on two factors: the maximum correlation size possible and the size of a subblock. Memory length versus width tradeoffs can be implemented directly in FPGA RAMs, because of the many configuration options in those RAMs.

The "maximum" function is used to summarize all the scores in a given subblock, but that summarizing function is swappable. Under some assumptions about molecule behavior, broad scoring peaks may be preferable to higher but very narrow maxima. We are not aware of functions that quantify the desirability of broad peaks, but have demonstrated other summary functions as proof of the architectural principal. Since there is only one instance of the summary logic in the entire system, any credible summary function would consume only tiny amounts of the FPGA's computing resources.

6. SYSTEM CONFIGURATION AND OPERATION

Once synthesized, an accelerator can be reused with many molecule pairs. Resynthesis for changes in the force law is basically a recompilation step, ensuring the largest possible number of processing elements in the correlation array and the highest throughput.

Table 1: System components affected by configuration choices.

System component	Configuration features			
	Voxel data type	Score data type	Scoring function	Array sizes
Rotated image traversal				Address bit allocation
Molecule memory	Memory width			Memory length
Voxel rotation (optional)	Rotation function, if any			
3D correlation array		Communication paths, FIFO widths	Correlation computation	FIFO lengths
Data reduction filter		Summary function, memory width		Memory length

Each different force model is represented by modifying the configurable elements of the correlation accelerator and synthesizing an accelerator for that force law. Synthesis, placement, and routing were performed using Synplicity's Synplify 7.7.1 tool set, without hand tuning except to set the sizes of the correlation and memory arrays. The system as a whole consists of more than 60 VHDL source code files, but changes to force laws affect only the five files defining the following features:

(i) voxel data type and optional rotation function,
(ii) score data type and inequality test,
(iii) scoring function in each "correlation" cell,
(iv) sizes of the computation array and molecule memory, dependent on the amounts of logic and RAM required for the voxel and score data types.

Scores may be simple fixed-point numbers, values in saturated arithmetic, or any convenient representation. Most of the test cases share a common score data type and only one test case uses voxel rotation, so fewer than the maximum number of source files need to change for each force law. Those few inputs affect the following system components as shown in Table 1.

Our initial implementation uses only on-chip resources for computation. Other than an IO periphery for setting up and for collecting results, there are no dependencies on the accelerator board in which the FPGA is mounted. Also, since the computation core is written in VHDL, it is readily portable to larger members of the Xilinx Virtex-II Pro family, where larger amounts of resources increase the sizes of problems that can be addressed. For the same reason, the application can readily be ported to different FPGA fabrics with comparable computing resources, such as Altera's Stratix-II family.

Once the application-specific accelerator is built and loaded into the FPGA hardware, the host interacts with it in the following steps.

(1) Load the large molecule's image into the *substrate voxel memory*.
(2) Load the small molecule's image into the *systolic 3D correlation array*.
(3) For each of the three-axis rotations.
    (3.1) Load rotation parameters into the *rotated image traversal*.
    (3.2) If necessary, load rotation parameters into the optional *voxel rotation* logic.
    (3.3) Start correlation and wait for completion.
    (3.4) Fetch results from the *data reduction filter* and clear the filter memory.

The molecule images are loaded only once, and reused until different molecules are desired. Only a few dozen control values need to be set up for each three-axis rotation, and setup can be overlapped with the previous correlation's execution. Also, there are relatively few results to fetch from the data reduction filter. Because of hardware double buffering, setup and result collection can be overlapped with the next correlation's operation. Given the small number of data transfers at the start and end of each correlation, these factors combine to make possible back-to-back correlations with very little dead time in the accelerator hardware.

7. PERFORMANCE RESULTS

Table 2 lists performance results for docking calculations based on various force laws. Each force law's voxel representation requires different numbers of bits, and the scoring calculations require different amounts of logic. As a result, the largest correlation array possible on a given FPGA is different for each force law, and the array sizes are shown. Each force law was also implemented using direct summation on a 3 GHz PC using normal good coding style but without aggressive optimization. Relative to direct summation on a PC, the FPGA implementation gave speedups of 130× to 1100×.

Performance results are based on the Xilinx Virtex-II Pro XC2VP70-5, in a Wildstar-II coprocessor board from Annapolis Microsystems. Performance is measured in billions of cell sums per second (CS/s), evaluations and additions of the function F in (2). These are directly analogous to multiply-accumulates (MACs) in a standard correlation.

The results clearly show the benefit of reconfiguration for handling the different possible force laws. Suppose that, instead of configurations specific to each force law, a single type of PE could have been used, programmable so as to handle any of the force laws reported in Table 2 [2]. Assume that the programmable PE took the same logic resources and clock rate as the ACP force law. Then, all of the simpler force laws would have been forced to run with that smaller number of PEs and lower clock rate—up to 86% performance

Table 2: Performance results by force law.

Force law	Bits per voxel	Correlation array size	Clock MHz	FPGA performance 10^9 CS/s	PC correlation performance 10^9 CS/s	Speedup[6] over PC, incl. IO
KK[1]	2	$14^3 = 2744$	98.9	271.4	0.212	1024.1
PSC[2]	7	$12^3 = 1728$	88.3	152.6	0.232	526.2
GSC[2]	2	$10^3 = 1000$	59.8	59.8	0.221	216.4
ACP[3]	5	$8^3 = 512$	72.6	37.2	0.221	134.6
ES[4]	6	$10^3 = 1000$	72.8	72.8	0.121	481.3
SNORM[5]	7	$11^3 = 1331$	90.4	120.3	0.084	1145.7

(1) Similar to Katchalski-Katzir [3]. (2) Described in [2]. (3) Simplified desolvation energies, adapted from [18]. (4) Electrostatic force and collision detection. (5) Surface normals and collision detection [17]. (6) FPGA computation time is derated by 20% to allow last results to shift out of FIFO buffers into the data reduction filter.

degradation due to generality. It is the FPGA's reconfigurability that allows the PE for each force law to use different amounts of resources, resulting in different numbers of PEs and computations per clock cycle.

Timing values in Table 2 assume the −5 (slowest) speed grade for the FPGA. If the −7 (fastest) speed grade was used instead, clock rates would improve significantly. The larger VP100 chips are available in the same FPGA family, with 33% more logic and 44% more RAM, but were not used in these performance measurements. This table is based on standard synthesis, placement, and routing, without hand tuning. Careful optimization would have improved performance yet more. These implementation constraints allow us to focus on system architecture rather than technology details—further performance improvements are clearly within reach.

Our PC implementations of correlation-based docking use 3 GHz Xeon processors and direct summation, for a 50^3 substrate and ligand of the largest size supported by the correlation array. PC performance does not include rotation or result filtering, only correlation. Typical PC implementations use 3D FFTs for correlation, but direct summation was found to be faster for these problem sizes and was thus used for comparison. It is worth noting that the ACP force law is nonlinear and cannot be implemented using FFTs, and that the SNORM force law would require four FFTs at each rotated orientation. Actual FPGA performance is less than the raw performance figure, because of the additional cycles needed to shift final results from the computation array into the data reduction filter.

The FPGA accelerator is a commodity PCI board. It can easily be used in cluster configurations, where each PC in the cluster has its own accelerator board. Scoring of multiple ligands, multiple substrates, and thousands of three-axis rotations per substrate/ligand pair is an embarrassingly parallel problem. As a result, we anticipate near-linear speedup in performance as accelerators are added to the cluster.

8. CONCLUSIONS AND FUTURE DIRECTIONS

Correlation-based molecule docking has great scientific and commercial importance, but even the "fast" screening algorithms take hours on standard PC. Current FPGAs have been shown to be cost-effective tools for accelerating docking calculations by factors of hundreds to a thousand or more. Every different model of chemical interaction represents a different version of the correlation problem, and the nonlinear models simply cannot be handled using transform techniques. As a result, the FPGA's configurability is well suited to the idea of docking as a family of applications.

We present a computation pipeline that has fixed structure, but variable data types and scoring functions that are configured to each application. FPGA-based reconfiguration allows handling of a wide range of physical phenomena, including spatially oriented vector quantities. Since every instance of the accelerator is specific to one chemistry model, we achieve high resource utilization and avoid wasted logic due to over-generality of the individual processing elements. Using only standard synthesis, placement, and routing tools, without hand tuning, we achieve computation rates of 10^{10}–10^{11} correlation cell sums per second. Because every accelerator is specific to one force law, models with faster computations are not constrained to run at the speed of slower computations, or with as few PEs as more complex computations. Likewise, complex models are not cramped by fixed datapath or PE sizes. This family-based design goes a long way towards resolving the conflict between application-specific tuning and wide reusability in a computation accelerator.

We are working towards many improvements in designing accelerator families. The essentially one-dimensional data path through the computing array allows larger arrays to be built when larger FPGAs are used, with near linear speedup. We also look forward to creating larger correlation arrays distributed across multiple FPGAs—again, possible because of the array's one-dimensional data path. To date, we have used only the RAM available within the FPGA for storing molecule images and correlation results. We are currently working on extensions that will allow larger molecule models to be split into parts, staged in on-board memory near the FPGA, and reassembled into larger results. Other extensions, including more complex families of scoring, will enable even larger families of force laws.

In the larger sense, families of configurable accelerators seem applicable to many application domains outside of molecule interactions. The basic concept presents an optimized computation array of fixed structure but parameterized size. That fixed array is built around application-specific data types and processing elements that vary between family members. We have applied this concept not only to docking problems, but also to families of approximate string matching problems that occur in bioinformatics and in natural language processing. Our current research generalizes the concept even farther into creation of tools that support FPGA-based accelerators for many different families of applications.

ACKNOWLEDGMENT

This work was supported in part by the NIH through award RR020209-01, and facilitated by donations of software and equipment from Xilinx Corporation.

REFERENCES

[1] 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,” in *Proceedinds of IEEE Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence (CAPAMI ’87)*, pp. 88–93, Seattle, Wash, USA, October 1987.

[2] R. Cheng and Z. Weng, “A novel shape complementarity scoring function for protein-protein docking,” *Proteins: Structure, Function and Genetics*, vol. 51, no. 3, pp. 397–408, 2003.

[3] E. Katchalski-Katzir, I. Shariv, M. Eisenstein, A. A. Friesem, C. Afalo, and I. A. Vakser, “Molecular surface recognition: Determination of geometric fit between proteins and their ligands by correlation techniques,” *Proceedings of the National Academy of Sciences of the United States of America*, vol. 89, pp. 2195–2199, 1992.

[4] D. Ritchie and G. J. Kemp, “Protein docking using spherical polar Fourier correlations,” *Journal of Molecular Biology*, vol. 39, no. 2, pp. 178–194, 2000.

[5] W. Wriggers, R. A. Milligan, and J. A. McCammon, “Situs: a package for docking crystal structures into low-resolution maps from electron microscopy,” *Journal of Structural Biology*, vol. 125, no. 2-3, pp. 185–195, 1999.

[6] N. Brooijmans and I. D. Kuntz, “Molecular recognition and docking algorithms,” *Annual Review of Biophysics and Biomolecular Structure*, vol. 32, pp. 335–373, 2003.

[7] M. D. Eldridge, C. W. Murray, T. R. Auton, G. V. Paolini, and R. P. Mee, “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*, vol. 11, pp. 425–445, 1997.

[8] W. Wriggers and P. Chacon, “Modeling tricks and fitting techniques for multi-resolution structures,” *Structure*, vol. 9, pp. 779–788, 2001.

[9] H. T. Kung and R. L. Picard, “One-dimensional systolic arrays for multidimensional convolution and resampling,” in *VLSI for Pattern Recognition and Image Processing*, K.-S. Fu, Ed., Springer, New York, NY, USA, 1984.

[10] E. Swartzlander Jr., *Systolic Signal Processing Systems*, Marcel Dekker, New York, NY, USA, 1987.

[11] B. Draper, W. Najjar, W. Böhm, et al., “Compiling and optimizing image processing algorithms for FPGA’s,” in *Proceedings of IEEE International Workshop on Computer Architectures for Machine Perception (CAMP ’00)*, pp. 222–231, Padova, Italy, September 2000.

[12] N. K. Ratha, A. K. Jain, and D. T. Rover, “Convolution on Splash 2,” in *Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines*, pp. 204–213, Napa Valley, Calif, USA, April 1995.

[13] A. Kaufman, “Memory organization for a cubic frame buffer,” in *Proceedings of the European Computer Graphics Conference and Exhibition (Eurographics ’86)*, pp. 93–100, Lisbon, Portugal, August 1986.

[14] Fujistsu Computer Systems Corp., Biosciences Group. (confirmed on 6 September 2005), http://www.fujitsu.com/downloads/SOL/bsg/bioserver.pdf.

[15] M. Taufer, M. Crowley, D. Price, A. A. Chien, and C. L. Brooks III, “Study of a highly accurate and fast-protein ligand docking algorithm based upon molecular dynamics,” in *Proceedings of the 3rd IEEE International Workshop on High Performance Computational Biology (HiCOMB ’04)*, Santa Fe, NM, USA, April 2004.

[16] ClearSpeed Technology plc. (confirmed on 6 September 2005) http://www.clearspeed.com.

[17] I. Halperin, B. Ma, H. Wolfson, and R. Nussinov, “Principles of docking: an overview of search algorithms and a guide to scoring functions,” *Proteins: Structure, Function and Genetics*, vol. 47, no. 4, pp. 409–443, 2002.

[18] C. Zhang, G. Vasmatzis, J. L. Cornette, and C. DeLisi, “Determination of atomic desolvation energies from the structures of crystallized proteins,” *Journal of Molecular Biology*, vol. 267, no. 3, pp. 707–726, 1997.

Tom VanCourt is a doctoral candidate in Department of Electrical and Computer Engineering, Boston University. He previously earned his B.S. degree in engineering at Cornell University in 1978, and his M.S. degree in computer science at Boston University in 2001. He spent a number of years in industry developing operating systems, embedded code, CAD tools, networking software and protocols, and system utilities. He also teaches at Metropolitan College, Boston University; his teaching topics include advanced software design techniques, operating systems, and software engineering. His current areas of research include design of application-specific processors using reconfigurable logic, and use of software design technologies in developing hardware systems. Target applications for these configurable processors include computational chemistry, molecular dynamics, and bioinformatics.

Yongfeng Gu is in his third year of Ph.D. program in Electrical and Computer Engineering Department of Boston University. He received his Master’s and Bachelor degrees in computer science from Fudan University, Shanghai, China, in 2000 and 2003. He worked on embedded system for his Master’s degree, which involved ARM architecture debug tool and mobile platforms development. His current research focuses on reconfigurable circuit acceleration for bioinformatics and computational biology algorithms, including rigid protein docking, molecular dynamics, and BLAST.

Vikas Mundada received his B.S. degree in electrical engineering with his focus in computer systems in 2003 from the University of Houston where he developed two robots for IEEE competitions. In 2005, he completed his M.S. degree in computer systems engineering from Boston University. He focused on computer architecture, embedded systems, and wireless systems and also developed a wireless computer controlled minihovercraft. His previous industry experience involves international space station simulations for NASA and wireless networking. Currently, he works for Raytheon Network Centric Systems on the Future Combat Systems project as a Systems Engineer.

Martin Herbordt received the B.A. degree in physics and philosophy from the University of Pennsylvania in 1981 and the M.S. and Ph.D. degrees in computer science from the University of Massachusetts in 1990 and 1994, respectively. He is currently an Associate Professor of electrical and computer engineering at Boston University, where he is Director of the Computer Architecture and Automated Design Lab. His research interests are in computer architecture, design automation, and high-performance computing. Current projects include using configurable circuits to supply high-performance computing for bioinformatics and computational biology, reliable space-based computing, and computer vision architectures.