CAAD Lab Technical Report 2004-02

FPGA Acceleration of Rigid Molecule Interactions[*]

Tom Van Court, Yongfeng Gu, and Martin Herbordt

Department of Electrical and Computer Engineering
Boston University, Boston, MA 02215
`herbordt|maplegu|tvancour@bu.edu`

Abstract. Modeling of molecule interactions often uses two or more successive models of increasing complexity. Rigid models based on correlation techniques are common as early screening passes—to detect interactions worth costlier examination—and are often at the heart of later passes as well. Even these rigid models are time-consuming when applied to large models at $10^3 - 10^5$ different three-axis rotations. This paper presents an FPGA structure for performing the correlations efficiently by using a systolic array for 3-D correlation and arithmetic tailored to the application. The system includes a novel addressing technique for performing a three-axis rotation of a 3-D voxel model using modest amounts of logic and nearly no cost in time or buffer space. We compare our FPGA implementation with one on a PC using the standard transform-based method and find a speed-up of a factor of 200. We present extensions for handling implementation technologies with different performance characteristics and for handling models too large to fit on-chip.

1 Introduction

Molecule interaction modeling is the simulation of chemical systems to understand how two compounds will bind. This is important in many biological interactions, including screening drugs for human disease, understanding regulatory networks in living systems, and evaluating treatments for emerging pathogens.

Molecule models vary widely in the precision with which they capture chemical interactions and in their computational costs. Complex models may include many subtle phenomena, including quantum mechanical effects and flexing. Such models typically require hours of computation to describe one interaction, even when the starting position is close to the biologically significant or *docked* configuration. Initial screening programs test the small drug molecules (called *ligands*)

[*] This work was supported in part by the National Science Foundation through CAREER award #9702483 and the National Institues of Health through award RR020209-01.

against the large biomolecule (*substrate*), looking for relative positions of possible interest; this is a six-dimensional search involving all three-dimensional translations and three-axis rotations.

Screening typically makes a number of simplifying assumptions. One is to treat the flexible molecule as a rigid body. This approximation is used in ZDOCK, RDOCK, Situs, and other applications [1–3]. Another is that forces between molecules may be approximated as scalar values at fixed geometric relationships to the molecules themselves. This simplification still allows for geometric, electrostatic, atomic contact energy, and desolvation energy [4] effects.

Even these models require hours for processing just one ligand/substrate pair. Assuming 10° angular sampling intervals, there are 9588 three-axis rotations to try, and all (x, y, z) translations are checked at each rotation. This computation must be repeated for each drug candidate in the library, typically $10^3 - 10^5$ of them. The significance of faster screening is that it would enable the testing of more candidates, or the higher quality of testing of each candidate.

We report an FPGA-based hardware accelerator for rigid interaction models. This implementation keeps the chemists' original docking computation, but converts each step into FPGA-compatible terms. The result is an FGPA application accelerator that gives a 200× speedup in the rigid docking model.

2 Application Detail and Serial Reference Code

Rigid molecule behavior and scalar values describing interactions are captured by digitizing each molecule onto a three-dimensional grid. The two grids representing the two molecules are given some relative placement: an (x, y, z) offset and a three-axis rotation. Corresponding voxels (3-D grid cells) in each model are tested against each other and scored. The score for one relative placement is the sum of voxel scores in that placement. Total scores for all relative placements are compared, and the highest-scoring placements are processed further.

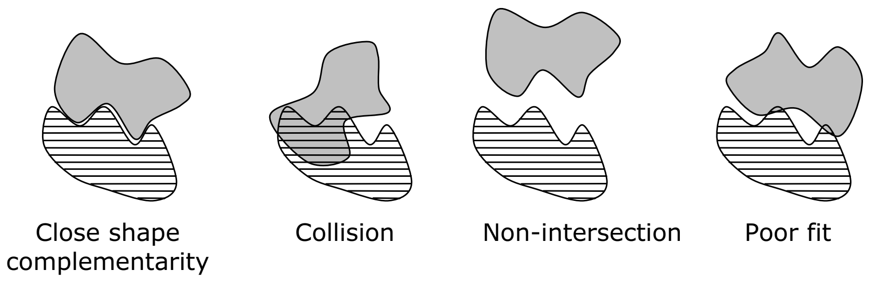

Fig. 1. Shape complementarity, collisions, misses, and poor matches

Figure 1 shows a basic match rule: shape complementarity. Matching voxels represent the interiors of both molecules and indicate a collision. That decreases the score for the particular relative placement. A few collisions do not automatically disqualify a relative placement, since real, flexible molecules might shift during the interaction. Desirable surface interactions represent positive scores. More complex phenomena, such as electrostatics or solvation effects, can also increase or decrease the value of a voxel interaction.

One scoring technique represents each voxel as a vector, with one element per effect modeled. Chen [4] represents the interior of each molecule as a positive imaginary value $\rho\, i$. When two interior values are multiplied together, this yields a negative score value $-\rho^2$. Other effects, such as electrostatic charge, are represented as positive or negative real values in other vector elements. The score at each voxel position is the dot product of the vectors at corresponding voxels in the A and B grids, and the total score $\mathcal{S}$ for the relative placement is the sum of voxel scores:

$$\mathcal{S} = \sum_i \sum_j \sum_k a_{i,j,k} \cdot b_{i,j,k}$$

A relative displacement of (x, y, z) between the two voxel grids has score

$$\mathcal{S}_{x,y,z} = \sum_i \sum_j \sum_k a_{i-x,j-y,k-z} \cdot b_{i,j,k}$$

where array elements with out-of-bounds subscripts have 0 value. A change of index variables creates $A' = A$ where $a'_{x-i,y-j,z-k} = a_{i-x,j-y,k-z}$. In other words the score $\mathcal{S}_{x,y,z}$ is the convolution $A' \star B$. Direct 3-D convolution takes time $O(N^6)$ for cubes of edge size N. In order to avoid this computational penalty, researchers observe that time for the Fourier transform $\mathcal{F}(A')$ (or the inverse transform) grows only as $O(N^3\ \log\ N)$, and that

$$\mathcal{S}_{x,y,z} = A' \star B = \mathcal{F}^{-1}(\mathcal{F}(A') \times \mathcal{F}(B))$$

Only one of the molecules (assume B) needs to be rotated with respect to the other, so only one Fourier transform needs to be performed at each rotation. Assume $\mathcal{F}(A')$ is computed once and its cost is amortized over all rotations. Then, the computation sequence for each rotation consists of:

- a three-axis 3-D rotation of grid B, giving grid B',
- a 3-D Fourier transform $\mathcal{F}(B')$,
- the 3-D multiplication $\mathcal{F}(A') \times \mathcal{F}(B')$,and
- the inverse Fourier transform $\mathcal{F}^{-1}$ on the product.

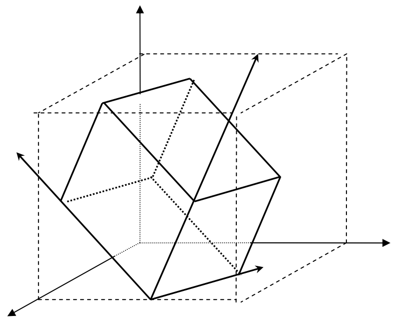

Fig. 2. Bounding box of rotated model

A few practical points are worth noting. First, in the worst case, the grid B may contain non-empty voxels in all of its corners. The rotated grid B' must be at least the size of the bounding box around that rotation of B, as shown in Figure 2. That means that the edge dimension of grid B' may be up to $\sqrt{3} \approx 1.73$ the size of grid B, or $\sqrt{3}^3 \approx 5.2$ the number of voxels. Since the molecule is not a repeating structure, the grids containing A' and B' must both be padded in three dimensions to a size that can contain both molecules without overlap. Padding and expansion in rotation do not change the polynomial complexity of the calculation, but might worsen the computation time by a factor of 40 or more, relative to the un-padded, un-rotated computation. The rotation and multiplication steps run in $O(N^3)$, so do not affect the polynomial complexity.

3 FPGA Algorithm, Implementation, and Results

The purpose of the Fourier transform and inverse is to avoid direct convolution. Convolution is a staple of signal processing, however, and so has been well studied; efficient structures are available for performing it. In fact, many FPGAs are optimized specifically for convolution or similar operations. The modest values and dynamic ranges of the voxel model inputs match the assumptions built into the FPGA arithmetic structures. Common current models have sizes up to $N = 128 = 2^7$, or $2^{21} = 2M$ voxels. That fits readily into current FPGAs' on-chip or on-board memory. Thus, a large, contemporary FPGA is well-suited to a highly parallel systolic convolution.

The rest of this section shows an accelerator for rigid molecule docking. First, we present the individual cells used in the convolution array. Second, we present a modified McWhirther-McCanny convolver that generates nearly one $\mathcal{S}$ score value per clock cycle. Next, we present a novel technique for addressing the voxel model so that 3-D rotation is eliminated as a separate step. This reduces four passes in the serial algorithm—rotation, transformation, multiplication, and back-transformation—to one pass in the FPGA algorithm.

Finally we demonstrate that, for realistic N, direct convolution can run faster than the transform-based algorithm. The time needed for 3-D floating point Fourier transforms and inverses slows the serial algorithm. Also, the FPGA algorithm benefits from zero loop and load/store overhead, from massive parallelism, and from our other optimizations. The net effect is an FPGA algorithm roughly 200× as fast as PC-based serial code.

3.1 Scoring arithmetic

Transform-based correlation is generally implemented using floating point values. Various scoring effects, representing shape complementarity, electrostatics, etc, are combined into a single score. Higher score values represent a higher net balance of attractive forces, lower scores represent stronger net effects due to collisions and repulsive forces. Computation of score values is indirect: score sums are output from the inverse Fourier transform.

We consider each component of the score separately, however. For example, collisions (when one spatial location contains interior voxels of both molecules) affect scores negatively. It is reasonable to assume that, once a certain number of core collisions has been detected, the relative positioning of the molecules is infeasible. Strong attractions elsewhere cannot overcome the collision penalty, and higher collision counts will not make that relative position any worse. The collision component of the score can therefore be implemented as an incrementing counter that saturates at some score specified by the biochemists.

Other attractive and repulsive effects often counter each other and tend to have different relative weights. Naive worst-case analysis requires that the highest absolute value for any voxel score be assigned to all $(N_A + N_B)^3$ voxels. Given each $N \approx 128$, this allocates at least 24 bits, or more bits depending on the precision of the voxel scores, and a sign bit. Realistically, however, the worst case would count only the number of surface or near-surface voxels that could participate in the inter-molecule interaction. Surface area tends to grow as N^2, not N^3, reducing the range of total scores to consider. Positive and negative voxel scores often cancel. Above some positive threshold, however, the attractive forces may unconditionally qualify a positioning for further analysis. Likewise, some negative threshold indicates that repulsive forces disqualify the positioning. Either way, saturating arithmetic captures the score's meaning and limits the number of bits needed for scoring arithmetic.

Our initial implementation models only shape complementarity. One bit in each voxel distinguishes molecule interiors from exteriors, and a second bit marks voxels on the molecule's surface. Separate counters record interior-interior overlaps (collisions) and surface-surface interactions. The multipliers and accumulators of standard convolution reduce to AND gates and increment operations.

3.2 Systolic array for three-dimensional convolution

Our 3-D convolution is based on the non-recursive form of the systolic array described by McWhirther and McCanny (MM) [5]. The biggest modification is the extension of the array from its original 2-D form to 3-D form. This extension starts with the MM structure shown in Figure 3. Each row in the MM array consists of a 1-D convolver and a fully synchronous FIFO line buffer.[1]

The 1-D array represents a row of A having length x_A. The 1-D convolution of respective rows in A and B generates a result of length $x_A + x_B - 1$; results beyond the x_A length of the 1-D convolver are held in the FIFO. The sizes of A and B can both vary according to the molecules being modeled and the rotation of B, so the FIFO has adjustable length. Variations in the size of A are accomodated by using parts of the 1-D convolver as storage elements. We assign the smaller molecule to grid A and the larger to grid B. This allows the relatively gate-intensive computation cells to hold the A values and the RAM FIFO to hold the rest of the results. Since the 1-D convolvers store the voxel values for the smaller grid, the number of convolution cells sets the upper bound on the size of the smaller molecule's grid.

FIFOs (line buffers) extend the 1-D convolver from length x_A to $x_A + x_B - 1$ before sending results to the next row in that 2-D plane. Likewise, the 2-D plane consists of y_A rows of the 1-D convolver-plus-FIFO units. The full size of the

[1] The shown connections between 1-D convolvers, FIFOs, and input differ somewhat from the classic MM structure.

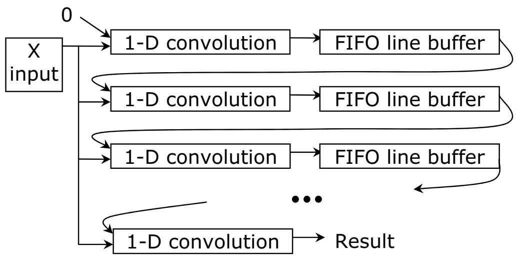

Fig. 3. Modified McWhirther/McCanny 2-D convolution array

2-D result is $x_A + x_B - 1$ by $y_A + y_B - 1$, so an additional buffer of length $(y_B - 1) \times (x_A + x_B - 1)$ is added to each plane. Like the line buffers, these plane buffers are RAM FIFOs that can be programmed to handle different values of $(y_B - 1) \times (x_A + x_B - 1)$, according to molecule sizes and rotations. The entire 3-D array consists of z_A of these 2-D planes, as shown in Figure 4.

This structure yields one convolution value per clock cycle, except for a relatively small number of control cycles. Voxel values for the A grid are loaded once and reused for all rotations of the B grid. B values then stream past the convolvers. Since the B input has size (x_B, y_B, z_B) and the convolution result has size $(x_A + x_B, y_A + y_B, z_A + z_B)$, there are more output values than inputs. The same number of values are clocked in as are clocked out, so the B voxel values are padded after every line, after every plane, and after the last plane.

After the extension to 3-D and technology-specific implementation choices, our third modification of the MM structure customizes the arithmetic operation in each systolic cell. As described above, shape complementarity can be summarized using one bit each for the molecule's surface and interior. The convolution's multiply/ accumulate step is replaced by one-bit *AND* and saturated conditional incrementing. The classic MM array used single values at each stage of the convolution; this implementation uses value-pairs.

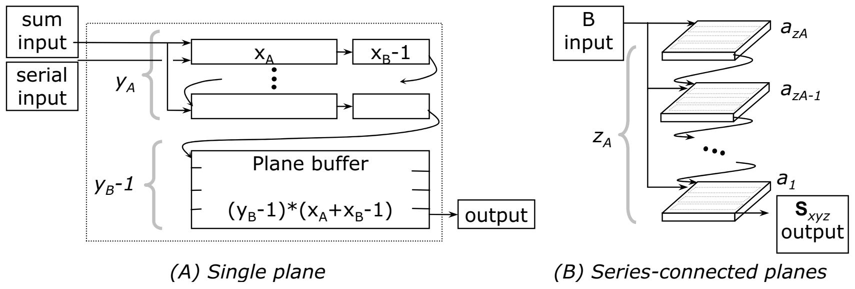

Fig. 4. Two-dimensional convolver extended to 3-D

3.3 Address generation for 3-axis rotation

Before each 3-D convolution, docking requires a three-axis rotation of one molecule relative to the other. The naive approach would be for the host or FPGA to create a new buffer, then create the rotated image in that buffer. After this $O(N^3)$ step, the convolver would scan the rotated model in linear order. Instead, our implementation scans the unrotated model directly in rotated order.

Let (x, y, z) coordinates index the un-rotated 3-D model. Also let (i, j, k) coordinates index the model in its rotated position. The conversion of (x, y, z) to (i, j, k) is a change of basis vectors, M, such that $M(x, y, z)^T = (i, j, k)^T$. The M transformation is obviously invertible, since the rotations that it represents can be reversed. Let M' represent the inverse of M, such that $(x, y, z)^T = M' \, (i, j, k)^T$. Expand that as

$$\begin{pmatrix} m'_{ix} & m'_{jx} & m'_{kx} \\ m'_{iy} & m'_{jy} & m'_{ky} \\ m'_{iz} & m'_{jx} & m'_{kz} \end{pmatrix} \begin{pmatrix} i \\ j \\ k \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

The vector $(m'_{ix}, m'_{iy}, m'_{iz})$ is a unit vector, representing the projection of i onto the (x, y, z) axes. A unit step in the i direction is identically a step along i's direction cosines in the (x, y, z) framework. Corresponding vectors exist for j and k, with corresponding interpretations.

The systolic convolution array described earlier requires that the input molecule be traversed in (i, j, k) order:

```
1. for indices $i$
2.  for indices $j$
3.      for indices $k$
4.          use element $(i, j, k)$ of rotated form of $B$
```

Line 4. can be rewritten as

```
4a. compute $(x, y, z)$ from $(i, j, k)$
4b. use element $B_{x,y,z}$
```

or

```
4a. $x = round(m'_{ix}i + m'_{jx}j + m'_{kx}k)$
4b. $y = round(m'_{iy}i + m'_{jy}j + m'_{ky}k)$
4c. $z = round(m'_{iz}i + m'_{jz}j + m'_{kz}k)$
4d. use element $B_{x,y,z}$
```

The bounding box of the rotated model may be larger than the original model, as shown in Figure 2. In that case, the range of the (i, j, k) indices exceeds the range of the (x, y, z) limits of the unrotated molecule. As a result, some values of (i, j, k) do not correspond to values of (x, y, z) that lie within the original model. Line 4d in the algorithm above must be modified to perform a range check and to provide padding when the computed (x, y, z) value is out of range.

A few more observations allow this expression to be implemented efficiently in hardware:

– Direction cosines m' have values $1 \leq m' \leq 1$, limiting the number of more significant bits required.

– The x, y, and z sums must be precise to within $\pm$ 0.5 over the whole range of (i, j, k). This sets the number of fixed-point fraction bits needed.
– Out of range values must be detected in x, y, and z - this sets the number of fixed-point integer bits required.
– The $round(t)$ can be closely approximated by truncation $\lfloor\ t + 0.5\ \rfloor$.
– The i, j, and k values increment predictably, so strength-reduction converts the multiplications in 4a-4c to additions.
– After strength reduction, the 0.5 value needed for rounding can be pushed into the initialization values for the additions.

A final operation converts the (x, y, z) index into a linear memory address. Given B dimensions (x_B, y_B, z_B), the address polynomial $x + x_B(y + z\ y_B)$ performs that conversion. This is trival when x_B and y_B are powers of two. Hardware for the address polynomial is simplified if the (x_B, y_B, z_B) values are fixed and smaller models are padded to the size of the larger grid.

For each rotation, simple off-line computations convert three Euler angles into 18 control parameters: direction cosines, offsets, and range limits. Each new rotated convolution starts by downloading the parameter values that represent the desired rotation. Pipeline registers allow new parameters to be loaded while the previous convolution is still running, so setup time is negligible.

Our VHDL implementation of this addressing scheme requires 268 logic slices in a Xilinx Virtex-II Pro to handle rotation of a $128 \times 128 \times 128$ molecule grid, using only ordinary optimization, with no multipliers. This includes computation of the linear 21-bit memory address, detection of (i, j, k) values that lie outside the unrotated molecule, and setup and control features. If a molecule model does not fill the whole 128^3 grid, loop counters cover only the section of the grid needed. The (i, j, k) bounding box around a rotated model is always larger than the original (x, y, z) size, except when rotating by exact multiples of $90°$. On the other hand, the average bounding box is considerably smaller than the worst case. Address computation presents each rotated form in the smallest (i, j, k) box able to hold that particular rotation, resulting in a 30% average savings in result voxels and computation cycles.

3.4 FPGA implementation

We require a computation grid that accomodates typical drug candidates: a $25 \times 25 \times 25$Å model handles the majority of pharmacophores of interest. The computation must also handle common stubstrate molecules, or at least their active regions. A model $200 \times 200 \times 200$Å is acceptable for our initial implementation. Resolution around 2Å is common in models of the substrate proteins [6]; grid cells representing 2Å units matches that resolution. The ligand and substrate should be at least 12^3 and 100^3 grid cells, respectively.

We use 8-bit counters to handle overlap and surface contact scores. A convolution cell requires two 8-bit scoring registers and a 2-bit voxel register: 16 bits for results or 18 bits for a computation cell. The Xilinx Virtex-II Pro XC2VP100 FPGA can hold a cubical convolution array 14 cells along each side, with memory adequate for 100^3 grid cells in the substrate model and convolution buffers.

3.5 Performance results

We synthesized an accelerator with the following characteristics for a Xilinx XC2VP100 FPGA:

– two-bit models of the substrate and ligand molecules, describing only shape complementarity,
– pairs of eight-bit, unsigned, saturated values as convolution results,
– 14^3 cells in the computation array, allowing 14^3-sized ligands
– buffer capacity for substrates to 100^3, and
– rotated addressing for access to the substrate model.

Limited use of timing constraints led to a 24.9ns clock rate (47.8MHz). This student design gives usable results, but is almost certainly open to more aggressive optimization.

For comparison, we implemented the computation described in Section 2 in C and ran it on a 3GHz Intel Xeon processor. The 3-D transform and inverse were taken from [7]. The C code operates only on cubes of size 2^L for integer L, so we could not compare directly to the size-114 limit of our FPGA convolution. FPGA timings are based on a post place and route clock estimate of 24.9ns. We also assume 5376 rotations, corresponding to angles sampled at 12.7° intervals. Loading of models is amortized over the 5376 rotations, so FPGA load times are ignored.

Table 1: Performance results, FPGA vs. serial C code

		FPGA		*3GHz Xeon*		FPGA
	Result size	once (ms)	5376×	once (ms)	5376×	Speedup
1	$100 \star 14$	35.8	3:13	-	-	-
2	$66 \star 14$	12.2	1:06	4,250	6:20:48	347
3	worst $66 \star 14$	52.1	4:40	4,250	6:20:48	82
4	avg $66 \star 14$	20.3	1:49	4,250	6:20:48	209
5	result size 256	-	-	36,840	55:00:52	-
6	worst $100 \star 14$	159.8	14:19	36,840	55:00:52	230
7	avg $100 \star 14$	73.5	6:35	36,840	55:00:52	501

Table 1 shows timing comparisons between our FPGA implementation and the serial algorithm. Table entries have the following meanings:

1. "$100 \star 14$" represents convolution of a 100^3 grid with a 14^3 grid, giving a result of size $(100 + 14 - 1)^3$. The PC algorithm computes only results with edge lengths that are powers of two, so no matching PC result is available.
2. The PC implementation can produce results of size 128^3. That is taken to be a $115 \star 14$ convolution. A box of size 115 can hold the worst-case rotations of a box size 66, so 66 is the molecule size used for comparison. Even unrotated, the PC implementation must round $66 \star 14$ up to size 128, so time for 128 is reported.

3. The "worst $66 \star 14$" row reports the worst-case rotation of a molecule size 66^3, i.e. 115^3. Since the unrotated grid has size ≤ 100, the FPGA handles handles this directly.
4. The bounding box of a rotated size 66 varies over a $1 : \sqrt{3}$ range in each dimension. This line reports the average time for convolving a rotated box of size 66 and a size-14 box.
5. The FPGA cannot compute a convolution with result size 256, but the PC implementation can.
6. The FPGA can compute the worst case of a rotated size-100 box convolved with a size-14 box. The PC implementation must round this up to a size-256 result.
7. This compares the average rotation of a size-100 box convolved with a size-14 result. The PC implementation is assumed to round up to a size-256 result in all cases.

4 Discussion and Extensions

One limitation of the accelerator implementation as just described is that the ligand must be small relative to the substrate molecule. Although this is the most important case in drug design, study of protein-protein interactions requires two large molecules. The extension is simple. Convolution is a linear operation: if $A = A_1 + A_2$, then $B \star A = B \star (A_1 + A_2) = B \star A_1 + B \star A_2$. If A_1 and A_2 partition A, then the total convolution is the sum of the partition convolutions.

Also, we have implemented only scoring functions based on shape complementarity. Many other functions exist, based on electrostatics and other phenomena, whose implementations are a straightforward extension. More complex force models remain to be explored.

References

1. Chen, R., Weng, Z.: A novel shape complementarity scoring function for protein-protein docking. Proteins **51** (2003) 397–408
2. Ritchie, D., Kemp, G.: Protein docking using spherical polar fourier correlations. Journal of Molecular Biology **39** (2000) 178–194
3. Wriggers, W., Milligan, R., McCammon, J.: Situs: A package for docking crystal structures into low resolution maps from electron microscopy. J. Structural Biology **125** (1999) 185–195
4. Chen, R., Weng, Z.: Docking unbound proteins using shape complementarity, desolvation, and electrostatics. Proteins **47** (2002) 281–294
5. Swartzlander, E.: Systolic Signal Processing Systems. Marcel Drekker, Inc. (1987)
6. Berman, H., et al.: The protein data bank. Nucleic Acids Research **28** (2000) 235–242
7. Press, W., Teukolsky, S., Vetterling, W., Flannery, B.: Numerical Recipes in C: The Art of Scientific Computing. Cambridge University Press (1992)